ACTA

UNIVERSITATIS OULUENSIS

Simo Ruokamo

SINGLE SHARED MODEL APPROACH FOR BUILDING INFORMATION MODELLING

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF TECHNOLOGY

ACTA UNIVERSITATIS OULUENSIS
C Technica 715

SIMO RUOKAMO

**SINGLE SHARED MODEL APPROACH
FOR BUILDING INFORMATION
MODELLING**

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in the Saalasti hall, Linnanmaa, on 4 November 2019, at 12 noon

UNIVERSITY OF OULU, OULU 2019

Copyright © 2019
Acta Univ. Oul. C 715, 2019

Supervised by
Professor Rauno Heikkilä

Reviewed by
Professor Kalle Kähkönen
Professor André Borrmann

Opponents
Professor Kalle Kähkönen
Doctor Tapio Koivu

ISBN 978-952-62-2369-8 (Paperback)
ISBN 978-952-62-2370-4 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2019

Ruokamo, Simo, Single shared model approach for building information modelling.

University of Oulu Graduate School; University of Oulu, Faculty of Technology

Acta Univ. Oul. C 715, 2019

University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

The current practice for information sharing with building information modelling (BIM) is distributed data sharing based on conversions. Conversions are problematic due to data loss, redundancy, and conflicting information. The hypotheses of this research were that i) a conversion-free data exchange is a feasible approach for BIM, ii) benefits can be achieved with a conversion-free information sharing, and iii) no impediment in principle exists for wider industrial use.

The use of a single data schema by all applications is a requisite for a conversion-free data collaboration. For enabling the free evolution of the data content, a version free data schema is necessary. A model arrangement implementing partial models is needed for the growing size of models. A single shared model approach eliminates data conflicts and duplicates. For the best availability, the location for the shared model should be on a cloud service. Accessing the cloud model only through a web service, which encapsulates all model handling functionality, will ensure data integration and validity.

The validity of IT solutions can only be confirmed with real software. For testing the conversion-free BIM method, a software development kit (SDK) with required functionality was programmed. Three applications and a cloud service for handling the shared model were developed with the help of SDK. In the experiments, *Leonardo* application was used for modelling walls, *3DTrussme* for trusses, and *Viewer* for viewing the model. All applications were using the same shared model on the cloud.

In the experimental test, the information exchange occurred without conversions, and all the data were saved only once on the cloud database. Without conversions, less conflicts and redundancies occurred, which lead to better data integrity and integration. Using SDK, there was no technical barrier for applications to join the single shared model ecosystem, but a drawback was that existing BIM programs are not compatible without remarkable changes. The performance was acceptable in the test run, but in real use, the size of the model and the number of applications and users will be much larger. However, a conversion-free single shared model approach can be a possible trend to the next generation BIM as well as a potential alternative for current data sharing methods using distributed files, conversions, and linked data.

Keywords: 3D geometry, BIM, building information modelling, conversion-free data sharing, single shared model

Ruokamo, Simo, Yhden jaetun mallin menetelmä rakentamisen tietomallinnuksessa.

Oulun yliopiston tutkijakoulu; Oulun yliopisto, Teknillinen tiedekunta

Acta Univ. Oul. C 715, 2019

Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Rakentamisen tietomallinnuksen (BIM) nykyisenä tiedonjakamisen käytäntönä on hajautettu tietojärjestelmä, joka perustuu konversioihin. Konversiot ovat ongelmallisia tiedon häviämisen, ristiriitojen ja päällekkäisyyksien vuoksi. Tämän tutkimuksen hypoteesit olivat: i) konversiovapaa tiedonjakaminen on mahdollista, ii) etuja on saavutettavissa ilman konversioita tapahtuvassa tiedonsiirrossa ja iii) laajemmalle teolliselle käytölle ei ole periaatteellisia esteitä.

Konversiovapaa tiedon jakaminen edellyttää yhden dataformaatin käyttöä. Alati kasvavien tietomallien koko vaatii tiedon järjestämismenetelmän, joko mahdollistaa osamallit. Datan ristiriidat ja päällekkäisyydet voidaan estää yhden jaetun mallin menetelmällä. Informaatio on parhaiten kaikkien osapuolien saatavilla, kun jaettu malli sijoitetaan pilvipalveluun. Kun tietomallia käsitellään vain web-palvelun rajapintafunktioilla, tiedon eheys ja kelpoisuus säilyvät.

Tietoteknisten ratkaisujen kelpoisuus voidaan viime kädessä osoittaa vain toimivilla ohjelmistoilla. Konversiovapaan menetelmän testausta varten kehitettiin vaadittavat ominaisuudet sisältävä ohjelmistokehityspaketti (SDK), joka on edellytys sovellusten yhteensopivuudelle. Kolme sovellusta eli *3DTrussme*, *Leonardo* ja *Viewer* ohjelmoitiin SDK:n avulla. Ohjelmointirajapinta sisälsi tarvittavat toiminnallisuudet tiedonjakamiseen, ja se toteutettiin pilvipalveluna. Testiajossa *Leonardo*-sovelluksella mallinnettiin seinät, *3DTrussmellä* suunniteltiin naulalevyristikot ja *Viewer*-sovelluksella tarkasteltiin mallia. Kaikki kolme sovellusta käyttivät samaa jaettua mallia pilvipalvelussa.

Suoritetussa testissä kaikki informaatio jaettiin ilman konversioita ja tallennettiin vain kerran. Ilman konversioita ja päällekkäisyyksiä saavutettiin parempi datan eheys ja integraatio. SDK:n avulla uudet sovellukset pystyivät liittymään yhden jaetun mallin ekosysteemiin ilman teknisiä esteitä. Toisaalta, ilman merkittäviä muutoksia nykyiset BIM sovellukset eivät ole yhteensopivia. Testiajossa suorituskkyky oli hyväksyttävä, mutta todellisuudessa mallien koko sekä sovellusten ja käyttäjien lukumäärä ovat paljon suurempia. Tutkimus osoitti, että konversiovapaa yhden jaetun mallin menetelmä voi olla seuraava BIM-kehityssuunta ja vaihtoehto nykyisille tiedonsiirtoratkaisuille, jotka käyttävät erillisiä tiedostoja, konversioita ja linkkejä.

Asiasanat: 3D geometria, BIM, informaation jakaminen ilman konversioita, rakentamisen tietomallinnus, yhden jaetun mallin menetelmä

To my Family

Acknowledgements

The research behind of this doctoral dissertation was carried out at the Department of Civil Engineering of the University of Oulu during the years of 2012 to 2019. The experience of developing 3D and BIM software from 1986 to today has produced useful knowledge for the research work. I would like to thank Professor Rauno Heikkilä, who was the supervisor of this thesis and Professor Mikko Malaska, who was the supervisor at the start of this research project. To the next follow-up group members Professor Pasi Kuvaja, Professor David Porter, and Professor emeritus Jussi Karjalainen I extend my thanks for their general guidance on this doctoral dissertation work. The best thanks also to the pre-examiners of the doctoral dissertation, Professor Kalle Kähkönen and Professor André Borrmann, for their valuable comments.

The completion of the doctoral dissertation was supported by Tekniikan Edistämissäätiö and Kiinteistöalan Koulutussäätiö.

Enterpuxe Software Ltd has made the execution of the research work possible by offering all the needed computers, programs, and software development tools. Special thanks to Mr. Tapio Ristimäki for encouraging and instructive discussions about building design and its requirements during the past 30 years.

Kiiminki, September 27, 2019

Simo Ruokamo

Abbreviations

2D	Two Dimensional
3D	Three Dimensional
AEC	Architectural, Engineering, and Construction
AIA	American Institute of Architects
BIM	Building Information Modelling
BPC	Base Public Classes that are used as a start point for developing application-specific, private data models
CAM	Computer Aided Manufacturing
CNC	Computer Numerical Control
DDM	Distributed Data Management. A data sharing system based on separate files, conversions, and exchange files.
DXF	Drawing eXchange Format developed by Autodesk
ECEF	Earth Centered Earth Fixed
FM	Facility Management
GCS	Global Coordinate System
GIS	Geographical Information Systems
GUID	Globally Unique Identifier
HVAC	Heating, Ventilation and Air Conditioning
IAI	International Alliance of Interoperability
IDM	Information Delivery Manual
IFC	Industry Foundation Classes
ifcOWL	Ontology Web Language of IFC
ISO	International Organization for Standardization
ISP	Internet Service Provider
ISV	Independent Software Vendor
IT	Information Technology
LCS	Local Coordinate System
LSC	Local Storage for the Client
LOD	Level Of Development or Level Of Detail
MTC	Model Toolkit for the Client
MVD	Model View Definition
OODBMS	Object-Oriented Database Management System
ORDBMS	Object Relational Database Management System
PDS	Permanent Data Storage. Data stored on a hard disk where data is kept up without any running application or computer.

RAM	Random Access Memory
RDBMS	Relational Database Management System
ROI	Return Of Investment
SDK	Software Development Kit
SES	Smallest Editable Share
SSM	Single Shared Model. A data sharing system with only one model on which all participants have concurrent access. No conversions occur.
SSMS	SQL Server Management Studio
SQL	Standard Query Language
STEP	Standard for The Exchange of Product model data
TDS	Temporary Data Storage. Data kept in runtime memory of an application during the run of the application. Data is not available after the exit of the application.
UCS	User Coordinate System. A coordinate system defined by a user for easing 3D modelling work
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WCS	World Coordinate System
WSTC	Web Service Toolkit for the Client
XML	eXtensible Markup Language
XPath	XML Path Language

Key concepts and terms

BIM repository	A repository with BIM is a server or database system that brings together and facilitates the management and coordination of data stored in various files and documents.
Building Information Modelling	Building Information Modelling (BIM) is a digital representation of physical and functional characteristics of a facility. BIM is a process for creating and managing information on a construction project across the project whole lifecycle.
Concurrency control	In information technology and computer science, especially in the fields of computer programming, operating systems, multiprocessors, and databases, concurrency control ensures that correct results for concurrent or parallel operations are generated.
Data model schema	A data model is an abstract model that organises data elements and defines their relations. Data modelling is an information technology concept and it is an abstraction of reality and real world entities. A schema of a data model defines exactly how the data model is saved to a file or database. As an example, a schema for a relational database defines tables and columns that are needed for storing the data model.
Digital twin	A digital twin is a replica of physical assets, processes and systems. A digital twin is a real mapping of all components in the product life cycle using physical data, virtual data and interaction data between them.
Distributed data management	A data management concept, where data is scattered in various files using varied formats

	and data storages are located at multiple places.
Extensible Markup language	Extensible Markup Language (XML) is a textual data format that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. XML is developed by standards organization World Wide Web Consortium (W3C).
Industry Foundation Classes (IFC)	A data model definition that is intended to describe architectural, building and construction industry data.
Permanent data storage	Data storage that can keep the data also during the shutdown of the computer.
Single shared model	A data sharing concept with only one model, that is used and shared by all stakeholders. The shared model must be located on a server or as a cloud service for enabling equal access to all participants.
Smallest editable share	With the hierarchical arrangement, the smallest or lowest node with all its descendants that can be handled as one. Nodes under the smallest editable share cannot be handled alone but only as a part of the smallest editable share.

Contents

Abstract

Tiivistelmä

Acknowledgements 9

Abbreviations 11

Key concepts and terms 13

Contents 15

1 Introduction 17

1.1 Information modelling during the building's life-cycle 17

1.1.1 Data process with BIM 19

1.2 Current data modelling 20

1.2.1 Conversion-based collaboration with exchange files 21

1.2.2 Single model approach 23

1.3 Potentiality and qualifications for a single shared model aspect 25

1.4 Hypotheses of the research 28

2 Development of the single shared building information modelling 31

2.1 Computational principles for the building data model 31

2.1.1 Data model classes 33

2.1.2 Data content 35

2.1.3 Data storage 42

2.1.4 Data saving and reading 45

2.1.5 Data arrangement 47

2.1.6 Data complexity, integrity and integration 51

2.1.7 Data availability 52

2.2 Programming principles of the single shared data model system 54

2.2.1 Data storage tables 55

2.2.2 Development of applications 56

2.3 Arrangement and execution of the experimentation 58

2.3.1 Cloud service 58

2.3.2 Applications 58

2.3.3 Execution and data recording 60

3 Results 67

3.1 Prototypical implementation of the single shared data model
system 67

3.1.1 Software data model 67

3.1.2 Data storage 76

3.1.3 Data sharing.....	77
3.2 Feasibility of conversion-free data exchange	81
3.3 Pros and cons of the conversion-free data exchange.....	83
3.4 Technical realization for the industrial use.....	86
3.4.1 Permanent storage content.....	87
3.4.2 Expanding the industry coverage of the experiment	94
4 Conclusions	97
4.1 Conclusions for research hypotheses	97
4.2 Single shared model with conversion-free data exchange	98
4.3 Reliability and validity	104
4.4 Economical aspects	107
4.5 Recommendations for future work.....	108
List of references	111
Appendices	117

1 Introduction

Before computers, building design was limited to handmade, 2D drawings on paper. In the 1970s, the first 2D design software came to the market and enabled drawing on the screen instead of paper. Called Computer Aided Design, or CAD, it became well-known across the construction industry as the leading standard. In the 1980s, pioneers like Finnish companies Cadex and Tekla developed 3D design applications for building design. Building Information Modelling (BIM)¹ is the successor for those 3D applications although the difference might be difficult to see. Early 3D modelling software, that is the foundation for BIM, already had the possibility of both geometrical and non-geometrical information. It was probably Jerry Laiserin who first introduced acronym BIM for Building Information Modelling (Laiserin, 2002; Miettinen & Paavola, 2014).

The successful research of BIM data models² requires knowledge from both the construction and computer science. Civil engineers and architects have the competence to describe the requirements and functions of BIM, whereas software engineers are capable of creating computer applications that implement those needed features. Although information technology mainly acts as an assistant role with BIM, in today's world, computers and software applications are a requisite part of all competitive business.

1.1 Information modelling during the building's life-cycle

There are a great number of resources where characteristics of BIM are defined from various points of view (Isikdag, Aouad, Underwood, & Wu, 2007). Data stored with BIM should be rich and cover all physical and functional characteristics of the building and the content of the information is an evolving field (Turk, 2006). Storing the data is one function for building information models. Depending on the application environment, BIM can contain many different functions and operations related to building and its processes (Isikdag, 2012).

¹ Building Information Modelling (BIM) is a digital representation of physical and functional characteristics of a facility. As such, it serves as a shared knowledge resource for information about a facility, forming a reliable basis for decisions during its life cycle from the earliest conception to demolition. A basic premise of BIM is to insert, extract, update or modify the information about the facility by different stakeholders during the whole life-cycle. (NIBS, 2019).

² A data model is an abstract model that organises data elements and defines their relations. Data modelling is an information technology concept and it is an abstraction of reality and real world entities.

During the life cycle of the building, numerous participants, like engineers, architects, facility managers, owners, users, authorities, contractors, and fabricators, are using various applications for modelling and utilising the data of the building. A successful collaboration between all stakeholders requires an efficient and functional sharing of the building information (Shen et al., 2010). Therefore, the data exchange is a vital feature for BIM.

Most of the BIM data is created during the design and construction stages of the building project. Information related to various disciplines is needed for making the building ready for practical use. After the completion of the construction stage, new information is still created but not at the same rate. Unfortunately, the data flow more or less breaks off or continues inadequately when the construction is completed (Figure 1) (Xu, Ma, & Ding, 2014). The data model schema³ of diverse applications are not properly compatible, which inhibits the smooth flow of data throughout the entire life cycle of the building. A seamless information flow is a pre-requisite for improving the data utilising within facility management processes (Nicał & Wodyński, 2016).

Fig. 1. The data flow through the entire life cycle of the building continues inadequately after the construction phase.

The sizes of building information models are growing constantly. Software houses develop new applications and new features which require new stored data. New entities and new attributes need to be added for data model schemas for actualising all the new developed characteristics. Schema changes generate new data versions, which complicate maintaining the compatibility between the numerous and various applications. Development work on each related application must be done for taking schema changes into account.

³ A data model schema refers to the storage format of the data stored on a file or database.

1.1.1 Data process with BIM

The entire construction life-cycle includes stages in which many participants are involved. ISO 29481 (Finnish Standards Association, 2017)⁴ lists eleven stages starting from the pre-life cycle stage *portfolio requirements* and ending with the post-construction stage *disposal* (Table 1). The information created during different stages is stored, for example, in written documents, schematic drawings and detailed BIM models. The first stages do not include a geometrical model, and after the construction, the accuracy of the model almost corresponds to reality. The model and improvement in the content of the information does not progress unbroken from stage to stage, but separate information models are made for various purposes.

Table 1. Life cycle stages according ISO 29481.

Principal stage	Stage number	Stage name
Pre-life cycle stages	0	Portfolio requirements
	1	Conception of need
	2	Outline feasibility
	3	Substantive feasibility
Pre-construction stages	4	Outline conceptual design
	5	Full conceptual design
	6	Coordinated design (and procurement)
Construction stages	7	Production information
	8	Construction
Post-construction stages	9	Operation and maintenance
	10	Disposal

In the early stages in schematic drawings, a wall can only be presented as lines. An architect often models the whole facade as one wall whereas the construction engineer divides the facade into individual wall elements with detailed information for manufacturing. It is typical that different participants and applications need different data and different levels of detail from the model information. This can be handled by creating several models and combining or linking the same data found in these separate models (Rakennustietosäätiö RTS, 2013). Another alternative is

⁴ ISO 29481 standard is originally published by International Organization for Standardization on May 2016. Its title is Building information models – Information delivery manual – Part 1: Methodology and format

to use less models and extract the required information or level of detail for each purpose.

Different disciplines usually have their own data area and components. Only an HVAC engineer will model ventilation ducts, and only an electrical engineer will model electrical wires. The exception is the architecture and construction engineering disciplines, which both need, amongst other things, access to walls, columns and beams. Making holes in structures is only allowed for an architect and structural engineer. An HVAC engineer is not allowed to make the opening for a ventilation duct without permission from the architect or structural engineer.

1.2 Current data modelling

The current standard for data exchange is a distributed data management (DDM) approach. There are various applications available, which all have their native data formats for information handling and storing. Collaboration and data exchange occur by conversions since applications work only with their native formats. In general, conversions are not made between native formats, but an exchange format is used for data transfer. Using a special exchange format requires two conversions between two applications, but a commonly understood data exchange file reduces the total amount of import and export formats each application needs to implement (Isikdag & Underwood, 2010). A conversion between native formats requires only one conversion, which is less error prone than two conversions with a special exchange format. On the other hand, more development work is required since each supported format must be programmed. Translations between languages are similar to conversions from one data format to another. German text can be translated into French directly or by first making a translation from German to English and then from English to French.

Another data sharing approach is a single shared model (SSM). With one model, no conversions are needed for the data exchange between applications. The challenge with the single model is concurrent changes made by different users. Rules must exist for defining which change is valid, when overlapping modifications happen. However, handling conflicting changes is even a bigger problem for the distributed data management system. Different applications and users are free to change everything on their proprietary data models, which will likely create inconsistencies. Figure 2 illustrates the different exchange methods from the perspective of conversions.

Information exchange using conversions between applications. Conversion tool between each application pair needed.

Information exchange using exchange file. Each application implements conversion tool only for exchange file.

Information exchange with single model without conversions.

Fig. 2. The basic differences for the needed conversions with the different data exchange methods. With application to application, the total number of conversions is the largest, whereas with a single shared model no conversions are made.

1.2.1 Conversion-based collaboration with exchange files

Interoperability is the ability to exchange information between applications and it has traditionally relied on file based formats (Eastman, Teicholz, Sacks, & Liston, 2011). In a distributed data management system, each participant creates, owns, handles, and publishes his or hers data model. Information is distributed over several models, which typically include overlapping and occasionally conflicting data. The fundamental principle for exchange files is conversions, which must be made every time data is transferred from one system to another.

While communicating between various parties, files and models may originate from several applications, as well as being converted to and from their original formats. Information exchange is mostly not done between the native formats of those various programs, but some standard or widespread accepted method is used instead. The original private files are converted to some accepted format, which is requested for communications between parties. The most widely used information management distribution system is based on Industry Foundation Class (IFC) models (Lee, Eastman, & Solihin, 2016), but certainly other formats are also used. As an example, *Trimble's Tekla Structures* lists approximately 40 compatible formats that can be used for importing and exporting data (Trimble Inc., 2018).

Data exchange in a distributed system is realised by delivering files to all involved project members. File types can be many kinds, like a document made with various office programs, drawings in 2D or 3D, 3D models, or a picture file. Files are not necessarily merged together; they are kept separate. In principle, the

file-based data exchange is only a digital evolution of paper documents. Instead of delivering and comparing paper documents, files are compared by viewing them with applications. *Trimble's* free of charge program *Tekla BIMsight* is an example, on which several 3D models from different disciplines can be viewed parallel and at the same time.

A document repository⁵ as a web service is good assistance for gathering all the documents and files into a single access point. All the considerable players on the market are offering cloud-based tools for sharing files. Some to mention are *Trimble Connect*, *Autodesk A360*, and *ProjectWise* by *Bentley*. *BIM Server* from *TNO Netherlands* and *TU of Eindhoven* and *Jotne EDMmodelServer(ifc)* are services based on IFC. Viewing, comparing, merging, and converting files are typical tools, which are offered by the cloud services. However, the foundation of this functionality is the conversion, which must be done at some point in the process. Applications use their own native formats and a conversion is an obligatory action for data interoperability. Basically, a cloud service only offers a single address for the building information that is collected from fragmented sources through conversions.

Linked Data Models

Distributed data models of the same building typically contain overlapping information. Keeping that clashing data uniform is a complicated matter that has not been properly addressed. Törmä (2013) presents a linking approach for solving the interoperability between the separate files of the same building project. All of the distributed models are related to each other at different levels. By establishing links between the models, they can be kept separate, and the data exchange can be handled with links. The link is not just an address: it contains functionality and rules on how the data are interrelated and exchanged between distributed models or data sources.

The current research for applying the linked data and semantic web technologies for the AEC/FM field is based on the IFC standard. The ontology web language representation of IFC, ifcOWL, has been published by The BuildingSmart Linked Data Working Group. Linked data is seen as a promising technology for the information exchange and interoperability among many industries, but linked

⁵A repository with BIM is a server or database system that brings together and facilitates the management and coordination of data stored in various files and documents.

building data research is still in its infancy (Pauwels, McGlinn, Törmä, & Beetz, 2018).

The main challenge for linked data is the address definition and creation of linked relations. This is a problem much like pairing the identifiers of the same objects in diverse distributed files. The objects are initially created at multiple origins, which results in divergent identifiers. Pairing those unequal identifiers of the same object is a major problem in a distributed environment. Information cannot be exchanged between objects if the unequal identifiers from diverse origins are not connected. The number of objects in a building data model is so numerous that the manual pairing of objects would be impractical and too slow. It is an excessively time-consuming task to define link addresses manually between related model objects in diverse models. An automated solution is required. According to Beetz (2009), within the construction industry, relations between interacting parties and fragmented information may even be unique in the single project context, which hampers the automated exchange of data. Radulovic et al. (2015) present a detailed linked data generation process but do not propose any method for the automatic pairing.

1.2.2 Single model approach

With the single model approach, all applications connect to one shared model. Simultaneous changes to the same data are possible and the concurrency control⁶ can be handled by the following methods:

1. Last change is always valid. Parallel earlier changes are overwritten. A change to an object that has already been removed by other user can lead to an unsolved and indeterminate case.
2. First change is valid, storing unchanged data must be done. Later changes on the same data portion are ignored.
3. Data is compared and only actual changes are taken into account. When changing large portions of data and parallel changes occur, finding real changes can be difficult and lead to incorrect updates. On the other hand, changes at the attribute level of the entity can be handled, unlike with the first and last change

⁶ With information technology and computer science, especially in the fields of computer programming, operating systems, multiprocessors, and databases, the concurrency control ensures that correct results for concurrent or parallel operations are generated.

valid methods, where the whole entity is usually updated disregarding what was really changing.

4. Reserving and releasing the editable shares of the model. This method requires that a model arrangement realises partial models or a set of entities that can be reserved and released. Reserving the whole model by a single user is not an acceptable policy since it permits editing only by one user at a time.

The first three methods do not prevent concurrent changes that are made to the same data by separate users. They only define the rules for whose change will remain in effect. Only reserving and releasing portions of the model guarantee, that nobody does wasted work by making changes that are never validated. Schapke, Beetz, König, Koch and Borrmann (2018) divide the concurrency control into an optimistic and pessimistic approach. With optimistic control, changes to the same information are allowed and possible conflicts are attempted to resolve. Whereas pessimistic approach uses data locking mechanism for avoiding inconsistent changes. On the shown list, the fourth method is pessimistic and third optimistic. The first and second cannot be clearly categorized, but they are pessimistic in that regard that no conflicts are attempted to resolve. On the other hand, first and second method rely, that no overlapping changes occur or possible conflicts can be resolved. From that viewpoint, last and first change valid methods can be seen as optimistic.

The single model concept is not new and has existed for at least 40 years (Howard & Björk, 2008). Yet no full scale single model is available, only a few solutions within the scope of the same application. IFC-standard has now been developed around 20 years, but it is still only used as an exchange format. There is no clear reason for the lack of the single model systems. Perhaps the information technology has not been mature enough, or BIM application developers and users have no interest for single shared models. In addition, current research is concentrated on IFC compatible topics which endorses the dominant distributed data systems.

However, some commercial single model systems have been developed over the years.

- Already in the 90s, Finnish company *Cadex* had a multi-user system for use with their *SteelCad* application. It was using the “last change valid” method. *Cadex* was acquired by another Finnish company *Tekla* in 1998, thus applications developed by *Cadex* are no longer available.

- *Tekla Model Sharing* developed by *Trimble* (*Tekla* was acquired by *Trimble* in 2012) can only be used with *Tekla Structures* application. It supports local models and sharing occurs through a cloud service. Only changes are updated between the local and cloud models. When conflicts occur, the first change remains. (Trimble Inc., 2018). *Trimble* has a patented method for resolving conflicts when simultaneous changes occur (Rantanen, 2015).
- *Autodesk Revit Server* is based on reserving and releasing the collection of elements called worksets (Autodesk Inc., 2018). It can only be used with Revit family programs.
- *Graphisoft BIM Server* also uses reserving and releasing for preventing conflicting changes (Graphisoft, 2018). *Graphisoft* has a patented *Delta BIM Server* technology for handling the sharing of a single model data (Balla & Mohacsi, 2013). *Graphisoft BIM Server* is also only compatible with *Graphisoft Archicad* software.

All of the above listed single model systems can only be used with one respective application. Data sharing with other programs must be done with conversions and exchange files.

1.3 Potentiality and qualifications for a single shared model aspect

The current data collaboration and exchange are based on conversions and exchange files. A single shared data model system, that can share the information without conversions between different applications and disciplines, has not yet been implemented into industry practice. Use of IFC as the data carrier is currently growing. However, IFC does not carry all the relevant information for multi-featured construction business (Redmond, Hore, Alshawi, & West, 2012). Moreover, conversions are always problematic causing data loss, redundancy, and overlapping conflicting information. The variety of BIM-based software among the collaborators during the whole life cycle of the building is complicating the process for the data sharing. Due the diversity of applications, a variety of problems can occur, including the loss of data, difficulty in communication, and poor work efficiency (Oh, Lee, Hong, & Jeong, 2015). Data exchanges are not properly interoperable and the slow development of data standards hampers the implementation of BIM for the whole life cycle of the building (Volk, Stengel, & Schultmann, 2014).

In a single shared model, every object has the same origin. Each object is initially created to the single common model, where they obtain a unique identifier. That permanent identifier is carried by the object through the entire life cycle of the building data model. All applications accessing the objects of the shared model operate through that one and only identifier. With distributed data collaboration, same objects are created in multiple sources where they get a different identifier. When merging, the same objects must be discovered. Pairing the identifiers is a troublesome task that must be done at some point with exchange files. Manual pairing would work perfectly, but it is absolutely a too slow method. Rules to automatically find equal objects must be defined, but, with diverse files and formats a sufficient level of automation is hard to achieve. Whereas, with the single model system, no pairing is needed which is a significant difference. Figure 3 illustrates the basic difference of identifiers between the distributed data management system and the single shared model.

Fig. 3. The principal difference in handling the object identifiers between the distributed information management and the single shared model. With the single model, every object has only one identifier that is used by all applications which makes identifier pairing unnecessary.

A conversion free data sharing requires that all applications use the same data model format. Using a native and private data schema for saving the building information is not allowed. The current industry status is everybody using their own file formats which lead to a data sharing based on conversions. One data format

does not guarantee the integrity of the data. Separate files or databases by numerous applications can still include overlapping and conflicting information although they are all in the same format. Concurrent changes to the same data will raise the question which information is correct. Rules for handling or avoiding simultaneous changes must be defined and implemented for keeping the integrity of the data with the single model approach.

According to Turk (2016), improving database features is one of the future trends of BIM and another improvement is a true multiuser database accessed simultaneously by several users. A single shared data model using a true database as the permanent storage realizes both improvements.

The sizes of building information models are growing constantly. Various and numerous applications require more and more data. Nowadays, the storage capacity of hard disks is large and cheap, so from this point of view, the model size is not a problem. When applications are running, the model is no longer stored on the hard disk, but it is loaded into the random-access-memory (RAM) of the computer. The performance capacity of the computer still has a clear limit, and large models will slow down the execution speed. On a single model system, all information is stored as one unit, and this will likely lead to an enormous model size that will overwhelm most computers. Hence, the single model system must implement a model arrangement technique that divides the model into smaller parts called partial models. Without breaking down the single model into partial models, applications cannot handle the size of a shared single model. Model filtering tools are also a feasible method for reducing the model size. However, the key point is to decrease the amount of data loaded into the RAM memory and the count of geometrical objects drawn on the screen.

The data content required by various BIM applications is evolving continuously. To keep the software competitive, new and improved functionalities must be implemented. This type of work often requires the definition of new attributes and new entity types for the data model. It is out of the question to argue that building information content can be fully determined and then be kept unchanged. It is the same as saying that technological development will stop. Software companies will not accept a standardised data schema that they cannot further develop. With their private native data formats, companies have unlimited freedom when it comes to schema changes. A public data model schema must implement freely extendable data content; otherwise, it will be rejected by the industry.

Data complexity is related to the size, schema and variety. According to Johnson (2014), BIM complexity consists of task complexity, data explosion, software and people. Tasks performed by many people are variable throughout the whole project life cycle. The tendency is to include more and more information on the building information model by using the growing accuracy level. The use of an application can be difficult to learn, and various versions cause compatibility problems. The incompatibility between diverse private data formats and the data exchange using import and export functionalities will not simplify the complexity. The dependencies and relations between BIM model objects increase the complexity, especially when used externally between separate models. The single model approach should not increase the complexity of BIM data. On the contrary, there is an opportunity for simplification when the data are stored into one model only without duplicates or inconsistencies.

With distributed data sharing, all applications save their native models in private data storages. Single shared models must be available equally for all stakeholders constantly; thus, saving the information on somebody's local disk is not a valid solution. Today, advancements in cloud technology have made it more widely used, allowing a single shared model to be stored on the cloud and accessible to anyone at any time of the day.

1.4 Hypotheses of the research

A conversion-free data exchange requires the use of only one data format. To fully avoid the problem of overlapping and conflicting information, a single data schema is not enough. Separate models, although in the same format, can still include inconsistent data. With the single shared model approach, all of the information is saved only once, which eliminates data duplicates.

The key difference between the distributed data management and the single model concept lies in conversions. Operating without conversions makes a significant change for data collaboration. The single model approach can be seen as a natural selection for achieving the conversion-free data exchange. However, removing conversions is the ultimate objective. Hence, the hypotheses for the research:

1. A conversion-free data exchange and collaboration with building information modelling is an alternative for the distributed conversion-based approach.
2. A conversion-free information sharing method reduces data defects, keeps data less complex and improves data integrity and integration.
3. A conversion-free data exchange is technically realisable, and no impediment exists for its wider industrial use.

2 Development of the single shared building information modelling

The development of software starts by setting the goals that are required by the application. Available technologies and skills will guide the planning before the actual coding work. After the testing phase, the software is published for the end users and the maintenance period starts. In reality, the development process can be divided into more phases (Garzás, Pino, Piattini, & Fernández, 2013) which can progress alongside one another.

In this research, the main goal for the development was a conversion-free data collaboration and sharing method for building information modelling that fully maintains the data integrity of the model. A single data format that is used by all applications makes conversions unnecessary and a single shared model makes import and export actions useless for the data exchange.

2.1 Computational principles for the building data model

It is common to mix applications, data models, and storages. The regular users of any computer software see only the interface and are only interested in the functionalities of the application. If a single application does not have all necessary functionality, user will seek out other alternatives. However, from a software technology point of view, application, data model, and storage can be categorised apart. They have their own functions and objectives which together make up the whole software that a user sees as an application.

An application is a computer program designed to run under user supervision and to help he or she to perform daily actions. The user interface provides the method for the control of the program and is, therefore, a vital part of an application. A program can run on a standalone personal computer, tablet, or smartphone. In principle, any electronic device having a user interface includes an application.

An application is not running all the time. The user starts the program, uses it for desired time, and ends the usage by exiting the application. During program running time, information is available at the main memory of the computer, which is often called RAM. When the program is closed, the information disappears from the RAM and data must be stored to a permanent storage. Otherwise, all information needs to be retyped with every run of the program. That permanent storage is in the data storage device, which can retain its information even when

powered off. The standing memory of every computer is a hard disk, which has evolved from magnetic disks to today’s flash memories.

The information bridge between the application and permanent storage is the data model (Figure 4). A file or database is read into run time memory of the application as a data model, then modified during the program run and saved back to the permanent storage when the application exits. The data model is an important and close part of an application, but a good programming practice is to keep the data model in an isolated segment inside the programming structure of an application.

Fig. 4. Data model acting as data bridge between the application and permanent storage. Data model and application make up a temporary data storage (TDS). Data storing is permanent only when saved to a database or file located on a hard disk of the computer.

AEC/FM programs are using native formats for permanent storage and that naturally leads also to native data models. IFC is a data model originally standardised by International Alliance of Interoperability (IAI) (Eastman et al., 2011), but programs are not using IFC as their internal native data model. IFC is used with the data exchange between applications and together with ISO-STEP or ifcXML⁷ file as the permanent storage. IAI organisation has been renamed to *buildingSMART*, which is the current developer and supporter for IFC (BuildingSMART, 2018).

The permanent storage and data model are a reliant pair. They hold the same information, but in a totally different format and structure from a software technology point of view. Data conversion or mapping between the model and storage must be fully functional and the best result is achieved by developing and

⁷ ISO-STEP is ISO 10303 standard that describes how to represent and exchange digital product information. STEP is an abbreviation of Standard for the Exchange of Product Model Data. ISO 10303-28 standard specifies the use of the Extensible Markup Language (XML) as a data exchange format. With IFC, ifcXML is used as the file extension for XML files and for ISO-STEP files an extension ifc is used.

designing both in parallel. Though it must be kept in mind that a data model and storage have a separate aim and function although they are so tightly connected to each other.

2.1.1 Data model classes

The data model is the container and organiser for the information during the run time of the program. It will never independently change the data but acts as a passive data carrier. A data model can offer tools and functions for data-related tasks that applications can freely utilise. Operations for data copies, transformations, and geometry are examples of generic functionalities that are included in data model software. However, the use of functionality built into the data model is invariably under the full control of the application. The data model is the passive party that receives the tasks performed by the application.

The most common practice for implementing a data model in today's software technology is an object oriented approach (Berdonosov, Zhivotova, & Sycheva, 2015). Each different object is encoded as a software class that holds the data of the object and implements the object functionality. A requisite set of objects and classes then constitutes the whole data model. The functionality that is implemented into software classes should only be related to objects and their data. Application level tasks and operations should be encoded in the application itself for keeping the data model classes only for the data handling and separate from the functionality of applications.

Classes, objects, and instances

A class with information technology is a container for an object and its information. A class defines certain amount of data, which can be assembled under the same concept and instantiates an abstract or physical component into the model. An instance of the class is an object that is created from the class definition into the data model. All objects get a globally unique identifier when they are initially created and the identifier is stored into the permanent storage along with the object. One software class can be instantiated into the data model as multiple objects, each having a different unique identifier.

Rectangular sawn lumber is a good example of a physical component that can be defined as a software class, whereas a material definition is not a physical object when made as a separate class. A single class or object seldom makes up a complete

standalone component that users work within a data model. Sawn lumber can be developed as a one class, but a good and recommended programming practice is to split subjects and issues into smaller parts that can be reused and handled uncomplicated.

Standalone object and sub object

A standalone object or component is understood to be a single unit and is typically seen as one physical object. It is a collection of software classes with one primary that dominates other classes. Subobjects never exist alone in the data model, they are always owned and controlled by a master standalone class instance. A material is an example of a subobject that cannot be created alone into a model. A master standalone component, like concrete column or steel section, is needed as a holder for a material object.

All standalone components are not physical objects. A complete building data model includes also a lot of non-physical information. The designer's personal data, design code information, and results of structural calculations are examples of non-physical objects. Often some kind of graphical view for non-physical components is advantageous and helps user to analyse and handle the non-physical elements of the data model.

Assembly

An assembly is a collection of standalone components and subobjects. From the software technology point of view, a standalone component and assembly are identical. Both are a collection of classes with one as a main, which is mastering all others. Thus, an assembly is a building data model and construction process related concept (Kamali & Hewage, 2016), not software technology related. A concrete sandwich wall, window, and truss are decent examples of assemblies in the building model. Often an assembly is defined as a factory made component that is prefabricated as a single element before transportation to the construction site. That kind of specification is, anyhow, too restrictive and inessential from the viewpoint of the software technology.

2.1.2 Data content

The data model is established through a list of software classes that all define a set of attributes that are specific for each class. The list of the very fundamental types used for storing information and values with objects is very short: *number*, *text*, and *binary*. In software engineering, there are more data types available, as example, C# programming language by Microsoft lists 15 built-in types (Microsoft Corporation, 2019). Numbers are divided to *whole* and *floating point numbers*, *bool* is a number, and *date* and *time* are saved as a number or text format. In software technology, text is a *string* or *character* and is mostly used as storage for readable subjects, although almost all data can be converted as text and backwards. Extensible Markup Language⁸ (XML) is a popular data format nowadays and can be basically seen as text. Third base type binary is used for data that cannot be presented as number or text. The various picture, image, and file formats are categorised and best suited for the binary data type.

A universally unique identifier (UUID) or globally unique identifier (GUID) is a special data type for an identifier. It is a huge number with the max value of 2^{128} , and most computer software development tools and databases define it as a specific data type. GUID is value presented as 32 hexadecimal symbols and, although each value is randomly and independently generated, they are quarantined to be globally unique (Microsoft Corporation, 2019).

In a building data model, each stored value has a name and the information is a collection of *value-name* pairs. For dimensions, typical names are width, height, length, and thickness. Using a running number instead of a name is also functional, but makes pairs unclear and not so self-explanatory, thus, using a running number is not a recommended practice. A group of values can be attached to the same name, which can form points, coordinates, or other collections belonging to the same concept. With IFC, value-name pairs are called property sets (Eastman et al., 2011).

When talking about stored information in building models, the terms *parameter* and *attribute* are generally used. Both are value-name pairs, but parameters typically define geometrical and attributes define non-geometrical subjects. Parameterising is an operation on which parameters are used for generating new detailed information and the most natural target for parameterising

⁸ Extensible Markup Language (XML) is a textual data format that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. XML is developed by standards organization World Wide Web Consortium (W3C).

is geometry. Although parameters can occasionally also control other aspects than geometry, in this thesis the term *parameter* is used for geometrical values.

Location

In a 3D data model, every physical component must hold a position, which is best accomplished by attaching a Cartesian coordinate system to an object. An origin point and three perpendicular vectors unambiguously define the position of a 3D object. With the mathematical right-hand rule and cross product, third axis can always be calculated but mirroring operation turns the coordinate system left-handed. Fixing the mirrored location correctly needs all three axes, from which one direction must be switched for the restore of the right handing. A symmetry of the component along the shifted axis is needed for keeping the object unaltered. The symmetrical direction does not always exist. However, a left-handed coordinate system for an object is not allowed, because transformations and other geometrical calculations rely on right-handedness.

In a data model using parent-child relations, it is rational to express the component coordinate system in parent coordinates. This way the change in the location of a component is automatically reflected on the whole subtree without an actual change of the coordinate values of its children. The whole building with all its components is moved by just changing the coordinate system position of the building object.

The IFC specification includes three alternative techniques for defining object placement:

1. relative to world coordinate system (WCS),
2. relative to some other object,
3. relative to grid.

The world coordinate system is the default method and is called absolute placement. Relative to other object and grid placement techniques make cyclic relative placements possible, which must be prevented on the application level. Relative positions cannot be freely used as IFC defines conventions for some object types forcing the use of a certain placement system (BuildingSMART, 2018).

What is then a world coordinate system (WCS) or a global coordinate system (GCS)? It is the uppermost coordinate system on which all component positions are related. Moving the GCS will move all objects in the data model. Is it a corner of the building, a known place on site, or something else? There are many

geographic coordinate systems available, which are based on latitude, longitude, and elevation. Maps require a projection to two-dimensional coordinates onto a flat surface. The Earth-Centered Earth-Fixed (ECEF) is a Cartesian coordinate system with origin on the mass center of the Earth. ECEF expresses all the locations on Earth as well-known three-dimensional points (Zhu, 1994). The last one is evidently the best choice as WCS or GCS for the building data model since the same Cartesian coordinates are used for building components and no conversions between coordinate systems are the needed.

Geometry

The foundation of the building data model is a three-dimensional geometry. Hence, defining the geometry format for physical components is a vital task in the design process of the data model schema. A rectangular polyhedron is a common shape for building model objects and is defined with three dimensions. When all three dimensions are equal, the result is a cube. A cylinder is another base shape, but for complex shapes an extrusion method is used. By defining a 2D section with a closed polygon and extruding it along a straight line or mathematical curve, most of the shapes needed with the building model can be generated. Revolving is a special case of extrusion for rotationally symmetric shapes.

A solid is defined as surfaces, edges, and vertex points. Usually points are saved with an index and surfaces and edges use indexes for their point definitions. The Euler-Poincare formula can be used for checking the solid validity (Wilson, 1985) (Figure 5).

Euler-Poincare law: $\text{vertices} - \text{edges} + \text{faces} = 2(\text{solids} - \text{holes}) + \text{rings}$. Here vertices = 8, edges = 12, faces = 6, solids = 1, holes = 0, and rings = 0, which makes $8 - 12 + 6 = 2 = 2(1 - 0) + 0 = 2$.

Fig. 5. A solid is encompassed by faces. A face is a closed polygon, on which every consecutive point pair forms an edge. In a solid, each edge belongs to two faces. Face polygon points are vertex points of a solid. Each vertex belongs to at least three faces. The Euler-Poincare law defines the rule for a valid solid.

A surface is a plane with an indeterminate number of vertex points. The simplest surface is a triangle and it is the only surface that is always convex. A solid made up of triangle faces is called a triangular mesh (Peng, Kim, & Jay Kuo, 2005). Geometrical calculations are more straightforward with the convex polygon than with concave. With triangle surfaces, straight planes are triangulated and some of the triangle edges are invisible, not real physical edges. That must be kept in the mind when drawing only the visible edges of the solid. The term wire mesh is used for the collection of the visible edges of the solid.

The curved faces of a solid are approximated as straight planes, which results in a large amount of planes. Naturally, the number of surfaces for a curved component depends on the desired accuracy level. Therefore, with solids containing curved parts, it is a balancing between the geometrical accuracy and computer performance.

Solids that are generated from parameters are called *geometry primitives*. Parametric modelling is often inadequate and various operations between primitives need to be performed for the final shape. Those functions are called Boolean operations, which join, intersect, and subtract a pair of solids (Chang, 2015). Boolean operations are a heavy task and only full-scale applications support operations between geometry primitives. Therefore, it is reasonable to attach the final geometry also as surfaces, edges, and vertex points into the object data. That enables lightweight applications, which do not have a Boolean engine, to utilise the ready-made surface model for the physical component. Saving the surface mesh data of the solid geometry requires a lot of storage space. A rectangular polyhedron has six faces, 12 edges, and eight vertex points, but only three geometrical parameters, which together require the same storage space as one three-dimensional vertex point. Therefore, for parametrised geometry primitives, it is more rational to save only geometrical parameters and generate solid model faces and points on the run. Especially with curved shapes saving only the parametric geometry and not the triangular mesh saves a lot of storage space.

Each physical object has only one exact and correct geometrical representation in reality. However, that detailed geometry is not rational for all applications and designs. A structural engineer does not need the exact representation of a faucet and nor is an interior designer interested in reinforcement bars inside the concrete. Therefore, most objects should implement a separate geometry for different geometrical accuracy levels. Each application can then request that accuracy of the geometry that suits best for their needs. The American Institute of Architects (AIA) has defined five Levels of Development (LOD) for various design and construction

stages (Wood, Panuwatwanich, & Doh, 2014). Each LOD defines the accuracy and reliability of the geometrical representation. LOD can also be an acronym for Level of Detail, but it is not strictly speaking the same as Level of Development, although both handle the geometrical representation.

Non-physical objects can have a non-solid representation in which case geometry is not enclosed by planes. Spaces in buildings can be presented as the surfaces of floors, walls, or ceilings with openings included. The geometrical data of the collection of surfaces can be saved as a triangle mesh, but it does not follow the Euler-Poincare formula for solids. However, in the real world, all objects are solids that are bound by surfaces.

The IFC standard supports a wide range of geometry, including parametric shapes, volume enclosing faces, and definition for Boolean operations. These cover most of the construction and design requirements (BuildingSMART, 2018; Eastman et al., 2011).

Links and relations

A link or relation is a connection and data carrier from one object to another. Information is transferred from the source object to the target directly without altering or through simple or complex functionality. Bidirectional relations also exist, in case information is moving both ways with definite logic and calculations. A Boolean operation between two components is a good example of a link, which often is bidirectional and includes complex calculations that modify the geometry of both objects.

An address is a vital part of a relation and must be saved along with the target, source, or both. GUID is a generally used value, but the link address need only be unique in the given context. As an example, a strength class name is a valid link address in the scope of sawn timber or concrete. More flexible addressing is obtained by using a name and identifier pair as an address. The name is a freely defined name of a tool, application, function, or issue concerning the relation and the identifier is a unique value in the region defined by name. The mixing of links is avoided with named relation which also limits the usage of the link to the desired issue and functionality only.

For an external link address, a *uniform resource locator* (URL) is a known and familiar concept. It is shown by most web browsers on their address bar. URL is technically a *uniform resource identifier* (URI), which identifies the name of a web resource.

Standard data and catalogues are well suited information for the use as a linked data. Numerous objects use the same data that very seldom changes, which makes copying of the data for every object irrational. That kind of relation can be named as a *read link*, on which data is never copied to the target object, but instead read always from a source. Editing rights are not required for a data flow event from the source to target although the target follows source changes. When using a read link, it must be accepted that the alteration in the source data is directly reflected on the target.

When data is written from the source object to the target, a read link becomes as a *write link*. New or changed information is written to the target object instead of just reading it from the source. Both read and write links can contain rules and calculations that limit or modify the transferred data from the source to the target component. The source of the information can also be external, in case the source and target are not part of the same data model. With the write link, the data transfer can only occur with editing rights for the target object. A write link can also be seen as a change link because writing will change the data carried by the target object.

Write links can also transfer data in both directions in case source and target are mixed. A relation can then be named as a *two-way write link*. Both objects must have editing rights for link activities because both are changing. A joint between two objects is often a two-way write link. As an example, a bolted joint requires holes in both parts and thus changes both parts.

Grouping objects for sharing some information equally between all related components is *group linking*. It can be established using a name and identifier pair as a link address, which is defined equally for every object in a group. All connected objects are usually similar members and a change in one object is reflected on all others with the same rules. Each link on group linking can be separated as write link and, thus shares the same conventions for external objects and editing rights. An example of group linking is keeping selected members of linked trusses parallel and on the same line. Table 2 lists some features for different link types.

A data model without a single link would be the best as any link violates the specified and designed arrangement and increases the data complexity. Relations are especially awkward for parent-child ordering as they break the ideal segmentation of the model. However, a data model without links is only a theoretical dream that will never come true in practice. Relations between components are inevitably needed and must be used for establishing various functionalities. Nevertheless, every link has to be added with caution and serious consideration, not just dismissively. Circular linking is an issue that must always

be kept in mind when defining relations between objects. It is easy to construct a loop with just a few relations that make up an endless ring of activities.

With the shared model method that is reserving and releasing model portions, extra consideration with links is required. Changing component data requires reservation for avoiding overlapping editing by two users. With badly designed links, especially two-way write and group links, the whole model can become fixed as a single editable unit and therefore any change requires the reservation of the whole model.

Table 2. Link count and required editing rights with various relation types for connected components.

Relation type	Target data changes	Editing rights for target needed	Many targets can link to same source	External source feasible
Read link	No	No	Yes	Yes
Write link	Yes	Yes	Yes	Yes
Two-way write	Yes, source also	Yes, source also	No	No
Group linking	Yes, source also	Yes, source also	Yes	No

Data scheme evolution

List of attributes and classes are not constant but are an always evolving subject. New classes and new attributes for existing classes are needed which brings up the challenge of the data compatibility. Every new class or attribute can change the schema of the data model and permanent storage leading to a new version. Handling an older version with a newer one can usually be managed, but an older version cannot predict all the future classes and attributes. It would be ideal to define the whole needed data model content and keep that version forever. But it would be the same as stopping the technological development of the world.

Although the schema of the data content is an ever-changing issue, standards have a significant role with the building information modelling. Standards and specifications are common languages which realise the universal and admitted understanding for the content and structure of the building information. It is the backbone for collaboration and data sharing, as obscure and unreadable information cannot be utilised. Standardising has many levels which can range from a simple agreement of two parties to worldwide and national norms. For the building and construction sector, there are many international and national standards for diverse purposes like IFC, bSDD, OmniClass, Masterformat,

Unifomat, COBie, OpenGIS, gbXML, ifcXML, aecXML, BCF, CityGML, and agcXML (Eastman et al., 2011).

Standards establish public understanding, but they also limit the content of the information. The set of attributes needed for each object does not remain constant as new methods for the model are developed by applications. It is not an infrequent request that new functionalities are implemented which require additional attributes for components or an entirely new assembly composed from software classes.

A data model schema supporting only standardised data will never evolve as a commonly used system. Software applications at AEC/FM market are in a competitive position and they do not bind their advancement to a standardised data context. Each program keeps using their native nonstandard data models as primary storage formats, which they can freely develop. The standard data models are used only for the data interchange. That kind of activity leads to distributed information management and distributed information models, not to the use of a single shared model.

The flexibility and extensibility of the information content are key features of an open data model. It is advisable to also support standardised data, but by allowing applications to freely specify additional information content, technical barriers are eliminated from the use of a single data model schema. Part of the model data can be standardised data, while other portions can be defined by particular applications. It is not obligatory that all applications recognise or utilise all stored information fully. They can skip and ignore those portions of the model that do not belong in there are of interest.

2.1.3 Data storage

The simplest permanent storage format on a hard disk is a file. Most applications save their information into a file, which has a tailored internal structure. File formats and schemas have also been standardised for public use, but programs do not use them as permanent storage. Standards don't include all the data definitions that are needed by various software, which leads to the use of native files.

A database has a more organised data schema, which enables the partial use of the stored data. A file is read and saved as one by a single user at time, whereas a database allows partial data access and sharing with several users (Figure 6). Current design applications in AEC/FM area of business use files as a permanent storage, which is usually adequate for a single user running applications on his or hers personal computer. For the data exchange, a widely used standard file format

is ISO-STEP, which is the permanent storage format for IFC. For a single shared model, the self-evident selection is a true database storage system due to the requirement of many concurrent users. A better data organisation is also an advantage to a database system.

Fig. 6. Basic difference between a file and database used as permanent data storage (PDS). A file can only be accessed by one user at a time whereas database allows multiple users.

A database management system is a software application that handles data operations and database administration tasks. The relational data model was introduced in the early 1970s and it presents data in tabular form. This is called Relational Database Management System (RDBMS). The second basic database system is Object-Oriented Database Management System (OODBMS). Object-oriented database maps software classes as objects into the database. Both support query language for reading and manipulating stored data. Relational data is simple, but supports complex queries. For an object-oriented database, resolving queries is difficult but storing complex data is easy. By combining those two main database systems as Object Relational Database Management System (ORDBMS), the best features of both database models can be achieved. (Vanathi & Uthariaraj, 2011).

Extensible Markup Language (XML) has become the principal standard for information exchange over the web (Qtaish & Ahmad, 2015). XML is a platform for an independent human readable data format with a hierarchical, flexible, and expandable schema. It has a query language called XPath for selecting nodes from an XML document. XML can be saved to a file or mapped to a relational or object-oriented database system. Some commercial database systems support XML data type making direct queries into XML data possible.

Big data is a relatively new subject in the computer science. It is related to cloud computing and, therefore, big data is always located in cloud. The amount of stored data is increasing constantly and utilising that growing amount of diverse

data is a huge challenge for information technology. Big data can be characterised by four Vs. *Volume* for data amount, *variety* for the diversity of data, *velocity* for data transfer speed, and *value* for discovering the actual values of various data types (Hashem et al., 2015).

Database schema

A database schema defines how the data is organized inside the database. The classes of the entity data model do not have the same attributes. The name and number of attributes vary between different classes. The attributes for a reinforcement bar are not the same as for a door handle. The following three alternatives are technically possible as a database schema for storing the data of classes:

1. Separate table for each class with a separate column for each attribute. This schema has traditionally been used.
2. The vertical database schema also called an entity-attribute-value model (EAV) (Nadkarni et al., 1999).
3. XML schema, where whole data of the class is packed as XML data.

The 1st one is the traditional way for a relational database. Each class has a separate table with class specific columns. In the vertical schema, the database has tables for base software data types like integer, double and string. Each table has three columns: an object identifier, attribute name, and attribute value. With XML schema there is only one table with XML column, which contains the whole data of each object.

Juola (2014) compared all the three alternatives by implementing them as a SQL Server database and testing all three with an application. Changes in classes are difficult to handle with the traditional schema approach. Every new class or new attribute for an existing class requires a change to database tables. It is very difficult to handle table changes automatically and even more troublesome to keep the forward and backward compatibility. Therefore, the traditional schema using a separate table for each class is suitable only, if the data content is changing very rarely. The data content of BIM data model is evolving all the time which makes the traditional schema an unsuitable solution for the data storage of building information.

With the vertical and XML schema, class changes do not change the database schema, which fully eliminates all forward and backward compatibility problems.

The execution speed was the best with XML schema in tests made by Juola (2014). Data queries are complicated with the vertical schema, whereas traditional and XML schemas offer a simple formatted data reading directly from the database. On the whole, the XML schema was found the best suitable format for use with a building data model that has an always evolving content. Table 3 summarise the test results made by Juola (2014).

Table 3. Comparison of the results between three database schemas made by Juola (2014). XML proved to be the best schema solution for the building data model with the relational database management system SQL Server.

Database schema	Data content scaling	Execution speed	Queries
Separate table for each class	Difficult to keep tables and columns up-to-date due to class changes. Almost unfeasible to implement version free schema.	Reading average, saving slow.	Fluent and straightforward
Vertical schema	No schema changes due to class changes. Schema is version free, static.	Reading average, saving slowest	Complicated
XML schema	No schema changes due to class changes. Schema is version free, static	Reading relatively fast, saving clearly fastest	Fluent and straightforward when XML queries are supported by database system

2.1.4 Data saving and reading

The data model and permanent storage are a close pair of the data handling system for the building model. The data model holds the information in software classes, which exist only during the runtime of the application. The data in classes is saved into and read from the permanent storage, which is located on a hard disk of the computer. The development of the model and storage goes hand-in-hand for ensuring the perfect compatibility.

The definition of an associated class must be saved into storage along with the class data. Otherwise correct class cannot be instantiated in the data model. There are many ways to store the class specification. It can be a name or number in case an *if-else if-else* block needs to be coded where the right class based on the name or number is created. In the C# programming language, class objects are created with a *new* operator followed by the class name. This class instantiating method is called an *early* or *static binding*. The drawback of that kind of method is, that every

time a new class is added, the code must be updated and a new version of the application must be published. Static binding is used generally by BIM applications, but an exact practice cannot be known since the code of commercial applications is not available.

*Windows .NET Framework*⁹ contains a technique for creating a class instance when the full class name and assembly containing the compiled code of the class are known and available. That class instantiating method is called a *late* or *dynamic binding*. The combination of the full class name and assembly is called *AssemblyQualifiedName* in .NET framework. With *CreateInstance* function, a class can be instantiated when *AssemblyQualifiedName* is known. (Microsoft Corporation, 2019). When the class and assembly names are stored, no *if-else if-else* block is needed and classes can be instantiated using only the stored definition of *AssemblyQualifiedName* of the class. The assembly containing the compiled code of the class must be available for the application which is instantiating classes. When classes are placed in separate assemblies, only those assemblies containing the changed classes need to be updated. Even new classes can be added without making a new version of the application that is creating class instances into RAM.

With a C# *new* operator, instantiated classes must be available and known when code is compiled as an executable or assembly. On the contrary, *CreateInstance* function does not require existence and availability of the class during compilation time but only when compiled code is executed. This is a significant difference and makes the addition of new classes into the data model much simpler. Only writing the code for a new class is needed. Data model classes can be implemented in numerous assemblies, which can be developed and updated independently. Each application can update assemblies or add new ones on the run without publishing a new version. Nor needs other applications using the same model to publish a new version. To get the full advantage of using the *dynamic binding* method instead of the *static binding*, the data model and storage need additional features, which are presented in Chapter 3.1.2. IFC-standard does not include a definition for the use of the *dynamic binding* method.

⁹ Windows .NET framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library that provides programming tools for user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications.

2.1.5 Data arrangement

The number of objects in a data model can grow huge. It is self-evident that some kind of organising and grouping is needed for maintaining the fluent manageability of the model. The arrangement can be realized with automatic or fixed ordering that is more or less the practice among current applications and data models. Objects are organised based on their material, structural behaviour, type, name, position, feature, or some other data. Often applications offer users several methods for organising, sorting, and filtering the data model. As a result, the amount of active objects is decreased making the handling of the model more flexible. IFC supports Model View Definition (MVD)¹⁰ and Information Delivery Manual (IDM)¹¹ standards for defining a data subset (Lee et al., 2016). Several studies have been made for extracting IFC partial model based on MVD (Won, Lee, & Cho, 2013).

Hierarchical organisation is a well-known arrangement method for every computer user. Files on hard disk are arranged in a hierarchical manner, which user can freely define. That method has been applied very rarely among AEC/FM applications although it is also a suitable method for a building data model. The hierarchical arrangement naturally enables model division into partial models, which is a fundamental feature for large building data models.

The hierarchical arrangement is simply constructed by defining the parent object for every object, which makes up the hierarchical tree of the whole model. A partial model is then made up of an object and all its children at all sub hierarchy levels. With the hierarchical arrangement, a partial model is any subtree from the whole model tree. The partial model can be a standalone component, assembly or any other subtree extracted from the whole model tree. The IFC data model has a static spatial hierarchy, which organise building elements into a fixed hierarchy of *Project* → *Site* → *Building* → *BuildingStorey* → *Space* (Eastman et al., 2011).

¹⁰ Model View Definition (MVD) is a subset of the overall IFC schema to describe a data exchange for a specific use or workflow narrowing the scope depending on the need of the receiver.

¹¹ The ISO 29481-1:2010 “*Building information modelling - Information delivery manual - Part 1: Methodology and format*” standard has been developed by buildingSMART for having a methodology to specify processes and information flow required by various stakeholders during the lifecycle of a facility.

Free user defined hierarchy

A fixed hierarchy is not what we are expecting of the file system arrangement on our personal computer. We want to organise our files into a directory structure, which we can build up ourselves. The same is also valid for the hierarchy of the building information model. Singh, Gu and Wang (2011) present that a static hierarchy is inadequate and the ordering of the data model should be flexible for meeting user requirements. Division by design discipline, block, floor, and frame or element type are commonly used for the model organising. Combining all those in various sequences will lead to numerous alternatives for hierarchical arrangements, from which none is the one and right choice. Figure 7 shows three simple hierarchical arrangements for a building information model.

Shool	Warehouse	Hospital
Block A	Frame 1	Architectural
Floor 1	Columns	Floor 1
Columns	Beams	Columns
Beams	Slabs	Beams
Walls	Frame 2	Walls
Slabs	Columns	Slabs
Floor 2	Beams	Floor 2
Columns	Slabs	Columns
Beams	Frame 3	Beams
Walls	Frame 4	Walls
Slabs	Roof	Slabs
Floor 3	Beams	Electrical
Floor 4	Slabs	Floor 1
Roof	Walls	Floor 2
Beams	Outer walls	Plumbing
Slabs	Inner walls	Floor 1
Block B		Floor 2

Fig. 7. Three simple hierarchical arrangements for a building model tree. One organised by blocks and horizontal floors, second by vertical frames, and third by design disciplines. By mixing those three arrangements, more detailed model organising is achieved.

The free model hierarchy has a significant role by realising the essential model segmentation for partial models. Every node with all its descendants formulates one partial model. In Figure 7 *Floor 1* and *Columns* are examples of roots for the partial model. The first task for every modelling project is the definition of the basic

model hierarchy. The model arrangement is constructed by taking into account the project type, extent, participants, and many other things.

Fixed hierarchy of components

The freely defined model hierarchy is not the only arrangement that is used with building information models. Model objects and components (Chapter 2.1.1) are organised in hierarchies that have exact and fixed rules for parent-child relations. As an example, a reinforcement bar is most often located inside concrete, not in the air, hereby, an ordinary parent for the reinforcement bar is a concrete column, beam, slab, or wall.

The fixed component hierarchy is not managed by users. Applications and application developers must take over the fixed hierarchical arrangement of the components. Figure 8 shows fixed parent-child relations for a *Wood truss* and *Concrete wall* element. From the software technology point of view, free and fixed hierarchy are the same. Both are constructed equally by defining the parent object for each object.

Fig. 8. Examples of a fixed arrangement for two hierarchical physical components of the building model: Wood truss and Concrete wall.

Smallest editable share

Each object in the model has only one identifier. The ideal solution is alter never the object identifier, but every software developer knows, that keeping all object identifiers unchangeable is a troublesome issue. A good example is altering the parameters of a parametrised truss. The whole tree of descendants will change and it is hard to decide which descendants will change, which should be deleted, and which ones must be recreated. It is much easier to delete the whole subtree and replace old objects with new ones. The disadvantage of the descendant replacement is that every new object will get a new unique identifier.

An object identifier should be considered as a permanent address. All model actions target the identifier, which is an address to the correct object. Moreover, links based on permanent identifiers are broken if the identifier is changed. The challenge for permanent identifiers can be reduced by defining a *smallest editable share* (SES). SES is a hierarchical component which should always be handled and edited as one unity. Every change in any object belonging to SES component is a change for the SES and all objects in the SES tree can be updated ignoring whether an object has truly changed or not. A permanent identifier is only required for the SES root node, all descendants of SES root object can alter their identifiers freely. The disadvantage of SES is that identifier links cannot point to SES descendants, only to the root object of SES. Inside SES links are permissible, because they can be updated along with every change of SES.

Definition of the smallest editable shares is a question for an application developer. It is balancing between the handling of permanent identifiers, the size of SES component, and the need for links using permanent identifiers. In Figure 8 *Wood truss* and *Concrete wall* are advisable candidates as an SES component.

Hierarchical locations

Every model object has a location defined as a local coordinate system. In the hierarchical model, the location of an object is expressed through the coordinate system of the parent. Therefore, the whole model is moved by simply changing the location of the root object of the model. If all object locations are defined in the same common and global coordinates, every move must be carried out for all objects on the tree. It is much more convenient to handle the position of the child object in the coordinate system of the parent, than in the global coordinate system, which origin can even be in the mass centre of earth.

2.1.6 Data complexity, integrity and integration

A complexity describes a model or system that is composed of several parts whose relations can be nebulous. For data, the complexity increases due the size and intricacy. In general, large data is more complex than small data. Relations between data elements will complicate the data schema. Increasing heterogeneity and variety of the data results in more complexity. The growing size of BIM models is the current trend (Johnson, 2014). With diverse files, the individual size of each file remains smaller than with the single shared model, but the total amount of the stored data is the same. With distributed data management using conversions, diverse data formats are used, which increases the overall data complexity. Using only one data structure can keep the data complexity on a lower level. The Linked Building Data approach is based on the use of relations that will lead to more complex data handling. Links and relations are essential parts of BIM model, but the dependency between data relationships and complexity should not be forgotten.

Data integrity and integration are terms for the validity and reliability of the stored information. Data integrity is more related to the database schema, maintenance, and keeping the data up-to-date (Beynon-Davies, 1991). The saved information must reflect the real world and the data on the storage must be kept consistent and unbroken. Data integration is connected to sharing and combining information from different sources. Duplicated or redundant information should not exist after data is merged as one. With distributed files using conversions for the data exchange, keeping data integration is an awkward task. Conversions do not support all the data content and combining the overlapping information as one without flaws is cumbersome. For the single shared data model method, integration is mainly related to the handling of concurrent accesses to the same data.

The data concurrency control can be either optimistic or pessimistic (Wette, Pierre, & Conan, 2004). With an optimistic control, it is assumed that simultaneous operations causing conflicts are rare and can be resolved. When a concurrent data event occurs, the optimistic technique tries to resolve conflicts or keep only one event. The pessimistic control uses locks, which fully prevent any concurrent data events and conflicts. In Chapter 1.2.2, four methods for conflict handing with the single shared data model system were presented. Only the reserving and releasing method is categorised as pessimistic control, other three are optimistic. The data integrity and integration are best maintained with pessimistic data concurrency control, but the optimistic method can offer a better overall performance or fluency for the practical working.

The hierarchical arrangement divides the data model into numerous subtrees or partial models that can be reserved and released. It is straightforward to keep track of locked subtrees and give out the write access only for one user at a time. Reading does not need to be restricted. The single data model system and the hierarchical arrangement with reserving and releasing subtrees together make up a pessimistic data concurrency control system for ensuring the validity, reliability, and consistency of the information in the building model.

2.1.7 Data availability

For the best productive usage of the data, the latest up-to-date information should be available for all stakeholders from the early design steps to demolition. Native models are not typically shared, but instead exchange formats like IFC are used. That requires conversions at both ends, which makes the delivery process inflexible, slow, and onerous. Updates are shared rarely which leads to outdated shared data among users.

With the single shared model, all information can be shared continuously without delays and conversions. That naturally requires a cloud based system, which is the only rational choice for a single shared model in the current internet world. Accessing the cloud database directly is technically possible, but a local copy, which is synchronised with the cloud storage, gives some benefits. First, a local copy enables incremental updates, which greatly reduces the amount and size of data transfers (Eastman et al., 2011). With a local copy, by keeping a change log, only changed data needs to be synchronised between the cloud and local storage. Secondly, reserved data can be first saved to the local database until it is ready to be made public by sending it to the cloud. It is a human nature not to show unfinished work to other participants. Thirdly, a local copy of the cloud storage enables working offline without a connection to the internet although that occurs quite rarely nowadays.

Cloud service

Applications cannot access the shared data model on the cloud in the same way as it is possible locally. Allowing direct access to the database on the cloud would be a clear risk for the security and data integrity. A proper user control prevents an unauthorised usage of the information stored on the cloud database. Data operations

must be built up as functions, which fully maintain the integrity of the database. It is not accepted that data corruption occurs due to a false data event.

A cloud service implementing the functionality for user handling and data operations makes the access of the cloud storage safe and secure. The following operations are needed for the basic functionality:

1. *Registration* of the user for getting valid credentials and access to the cloud service.
2. *EstablishModel* for the creation of a new cloud model. User credentials must be included for checking the user validity.
3. *GetModels* for getting the list of models available for the user in the cloud service. User credentials must be included on the function call.
4. *Connect* to an existing shared model. The model is selected from the list returned by *GetModels* function. User credentials must be included for checking the user validity.
5. *Load* for getting the whole model or the desired partial model available locally for the client application. The loaded part is saved to the local database, thus, no new load is needed with next connections to the same model. After the load, the model data can be read and utilised by an application but not changed.
6. *Unload* for unloading a partial model from the runtime memory of the application. Unload is made only locally and no unload function is needed in the web service. Unload just deletes the partial model from the runtime memory and load reads the partial model from the local or cloud permanent storage.
7. *Check out* for reserving the desired partial model for editing. The reserved part is locked, other users can read the locked model part but cannot make a second reservation for the same partial model. The changes made during the reservation are saved into the local database of the user who made the reservation. Without *Check out*, the whole model is only available for reading, the reservation using *Check out* must be made for enabling data changes. When checking out, the latest data must be loaded from the cloud storage.
8. *Check in* for releasing the reserved partial model. The changes made locally are uploaded to the shared cloud database and made available for other users that have access into the same model. The write lock is also released. Alternatively, *Check in* can be made without releasing the lock in case the node tree remains reserved for the user in question.

9. *Get changes* for getting the changes due to releases made by the other users. The local database is synchronised with the cloud database. Only changes are transferred.
10. *Add* for adding a new node on the model hierarchy tree. The master node for the added node cannot be reserved by other users. This function is only for adding a node under a non-reserved node. New descendants for the reserved tree are handled through the *check in-out* process.
11. *Delete* for removing a node from the model hierarchy tree. The master node for the deleted node cannot be reserved. This function is only for removing a node under a non-reserved node. The deletion of descendants on the reserved tree is handled through the *check in-out* process.
12. *Disconnect* from the shared model on the cloud.
13. *Logout* from the service.

In a true implementation, the number of actual functions can differ, but they still implement the same functionality. The above operations minimise data transfers and overlapping changes are fully prevented. No data conflicts or redundancies occur and the latest information is available all the time for all stakeholders.

2.2 Programming principles of the single shared data model system

With the object-oriented approach, a data model is composed of an indefinite collection of classes that contain attributes and functionalities. To avoid duplicate code for similar classes, a derivation programming technique must be used. A derived class inherits everything from the parent class. The new derived class can develop the inherited content further by changing and adding attributes and functionalities. The most common practice is to define a root class from which all other classes are originally derived. With IFC, the root class is called *IfcRoot* (BuildingSMART, 2018).

Each class defines its data as attributes, which are saved to the database and can be read back. Saving and reading requires the functionality for mapping the data between the class instance and storage. When those data handling tools are implemented in the model's root class, the derived classes inherit the functionality from the root. In general, a good programming practice is to define all possible functionalities on the top of the inheritance to avoid creating duplicate codes.

The amount of software data model classes that are needed during the whole life-cycle of the building is vast. Moreover, the evolution of the data model is an unending process. Thus, the development and maintenance of data classes are not tasks for a single software house. Each application developer or software house must be able to define their own data classes for their specific needs.

The public part of the data model should be as large as possible to increase its uniformity, common understanding and ability to co-operate. The public portion of the data model can be seen as the standardised part. Many organisations in various countries are developing various BIM standards for diverse purposes (Barbosa, Pauwels, Ferreira, & Mateus, 2016). BuildingSMART International has published a few standards for building information content and data exchange (BuildingSMART, 2018). However, standardisation is not a requirement for making a data class publicly available.

The key public data classes containing the key functionalities must be available for all developers, and there cannot be multiple versions of those base classes. Thus, an assembly that includes the base public classes (BPC) is the foundation of data model development. There can exist only one BPC assembly. Application-specific classes are located on the application-specific assemblies. Every developer will then have an assembly of specific classes for the application. When *AssemblyQualifiedName* is saved into the database, as described in Chapter 2.1.4, and the assembly for a class is available during the runtime of the application, that class can be instantiated. However, all software houses will not want to make their special classes available for public and free use. In that case, a class cannot be instantiated by other applications. Instead, a BPC assembly class, from which the derivation of the private class has been started, can be instantiated. The detailed solution for the handling of classes in private assemblies is presented in Chapter 3.1.2.

2.2.1 Data storage tables

The database is the permanent storage area for the whole data of the building information model. A class instance stores the data on dictionaries as value-name pairs. The XML schema is the best one for data storing, as described in Chapter 2.1.3, and it is a straightforward task to map data from dictionaries to XML and back. Each data model class instance can be mapped as a single row on a database table, and the whole data model can be mapped on one database table.

A access to the cloud service must be controlled, and each user needs to register for getting valid credentials that are used at login. A table for registered users must exist within the cloud service. Every user cannot have access to all models, so a separate table defining access to the models is needed. Most of the cloud service calls are directed at one model only. After login, the user connects to a model, and the next calls are targeted to that model until disconnection. The period between the connection and disconnection is a session that includes the definition of the user and model. With a connection to the model, a session identifier is generated, and that identifier is used on calls to the cloud service from then on. Sessions are not permanent, but a new session is generated with each connection to a model, and with a disconnection, the session is invalidated. Without the session concept, user credentials must be sent to the client side, but with sessions, only the session identifier is used. It is against security principles to make user credentials available on the client side. For storing sessions, a separate table must be implemented on the cloud storage.

Event logging enables the tracking of reservations and incremental synchronising of the local and server model. When each event has a time stamp, the reservation status of each node and needed updates after a certain time can be discovered. In total, five tables are needed at the permanent storage location:

1. *Model* for model data.
2. *User* for registered users.
3. *Model access* for controlling user access to models.
4. *Session* for sessions between the connection and disconnection to the model.
5. *Event* for storing model events.

On the cloud storage side, all five tables are needed, but only the *Model* table is enough for the local storage. On the client side, the last model update time must be stored for enabling incremental updates. Only events that have occurred after a known time need to be handled when local storage is synchronised against server storage.

2.2.2 Development of applications

It is technically possible for the client applications to refer and use the web interface directly. The data transfer over the internet has some limitations, so it makes sense to offer a public helper assembly for the development of applications. This assembly can be named as a web service toolkit for the client (WSTC). It will refer

to the web interface and take care of the required data mapping for the web interface function calls. Client applications can then use WSTC interface instead of directly calling the web interface functions. WSTC implements the same functions as the web interface and thus can be seen as a wrapper library.

Another issue for the application development is the use of the local storage. For simplifying and speeding up software work, a ready-made solution for local storage is advisable. Local storage can be implemented in a separate assembly called a local storage for the client (LSC). Synchronising the data between the client and server storage is a task, which all applications must implement. That functionality can be implemented in a public assembly called a model toolkit for the client (MTC). MTC calls WSTC for cloud service calls and LSC for synchronising the local storage. MTC implements the same functions as WSTC. Those three assemblies, WSTC, LSC, and MTC, together with BPC assembly make up a software development kit (SDK). SDK is public and available for all application developers. Figure 9 shows a diagram of the whole system. It is an ecosystem that offers a conversion and version free data sharing system for the information of the building. Using the SDK, software houses can develop applications for the ecosystem. All applications can share all models which are available on the cloud service.

Fig. 9. A diagram showing the architecture for the single shared building model on the cloud and for the development environment for applications.

2.3 Arrangement and execution of the experimentation

For a full scale testing of the whole system, cloud service and applications need to be implemented. Applications are developed using the public SDK, which is common for all programs. Only one cloud service is needed for testing. All models can be saved into the same service and all developed applications can connect and use the same service.

2.3.1 Cloud service

Cloud service was running on a Windows Server operating system. The web service was implemented using Windows Communication Foundation¹² (WCF) (Microsoft Corporation, 2019). Storage system used was a relational database Microsoft SQL Server¹³. The required user and model information for testing was directly added to User and Model Access tables with SQL Server Management Studio¹⁴.

2.3.2 Applications

All applications were developed using the public SDK, which was developed as part of this research. SDK includes four assemblies as shown in Figure 9. BPC assembly implements over 150 classes, from which all application specific classes are derived. A local storage system as LocalDB is included in LSC assembly and is available for applications. LocalDB is a minimal version of a Microsoft SQL Server designed for developers and it is free of charge. The schema of the cloud and local databases is the same, which reduces the amount of coding work.

3DTrussME

3DTrussME is a 3D modelling and structural analysis software for wooden trusses. It has been developed along with the public SDK, and *3DTrussME* has been the

¹² The Windows Communication Foundation (WCF), is a runtime and a set of APIs (Application Programming Interfaces) in the .NET Framework for developing connected, service-oriented applications.

¹³ Microsoft SQL Server is a relational database management system with the primary function of storing and retrieving data as requested by other programs running on the same or on another computer across a network.

¹⁴ SQL Server Management Studio (SSMS) is an application for configuring, managing, and administering components within Microsoft SQL Server.

first and main testing application for the single shared model system. It is owned by a Finnish company, Ristek Oy, and the programming is carried out by another Finnish company, Enterprixe Software Ltd. (Kajava, 2017; Ristek Oy, 2018). *3DTrussME* is a commercial application currently used in Finland, Norway and Estonia. Some other countries are on testing phase before starting the actual use. The development work of *3DTrussME* started at May 2013 and is ongoing.

All the application specific classes for *3DTrussME* application are derived from classes available in BPC assembly. Almost 300 classes have already been programmed for defining the special data content for the design and analysis of wooden trusses.

Leonardo

Leonardo is a 3D design application for concrete structures. The development of *Leonardo* began in the year 2016, thus it does not have as large specific class library as *3DTrussME*. The first supported structures are a concrete column and wall. Later other concrete structures will be developed. *Leonardo* is not yet available for a practical use.

Viewer

The *viewer* application has no application specific classes. It will only use the public classes available in BPC assembly. The *viewer* is capable of viewing the geometry of all objects. Geometrical classes are publicly available in BPC assembly, hence the *viewer* can show the 3D representation of the whole model. Figure 10 shows the general arrangement of the testing environment.

Fig. 10. A diagram showing the general arrangement of the testing environment with three applications, public SDK package, and cloud service.

2.3.3 Execution and data recording

Three users are participating in the test event. User #1 runs *3DTrussMe* application, User #2 *Leonardo*, and the application for User #3 is *Viewer*. Application specific data classes developed with *3DTrussMe* are not available for *Leonardo* and *Viewer* applications. Also *Leonardo*'s application specific data classes are private. The *Viewer* application has no specific data classes and it uses only the classes in BPC assembly (Figure 10). The test was executed with steps shown in the next list. Referred web service function names are listed in Chapter 2.1.7:

1. Registration of users is done by adding users into the user table at the server database. Adding is done by using Microsoft SQL Server Management Studio.
2. User #1 starts *3DTrussMe* and establishes a new shared model *Demo* on the cloud service. *EstablishModel* web service function is used with user credentials included, that are checked from the user table on the cloud database. The model establishment includes following tasks.
 - a) A new model is created by adding one row into the model table. That single row is the model root node, which is the unique identifier for the whole model.
 - b) User #1 must have access to the new model, thus a row on the model access table needs to be added.

- c) On the event table, a *model established* event is added.
 - d) A session identifier is generated for the user and model combination, saved as a row on the session table, and returned to the calling client application for use with the next web service calls during the same program run.
 - e) At the client side, a local storage for the established model is created having one row for the model root node. The local storage for the local model is a LocalDB database.
3. User #1 reserves the whole model. *Check out* web service function is used. A *node checked out* event is added on the event table.
 4. User #1 models the base hierarchy for the model with *3DTrussMe* application. Changes are saved to the local model during modelling.
 5. User #1 releases the whole model. *Check in* web service function is used. Changes made locally are uploaded on the cloud model and a *node checked in* event is added on the event table. Figure 11 shows a snapshot from *3DTrussMe* after step #5.

Fig. 11. Basic hierarchy after step #5. All nodes are not checked out which is marked as a closed black lock icon in the project explorer tree. *Reference Planes* and *Load Combinations* nodes are specific for calculations and modelling in *3DTrussMe* application and were not part of the demo. (Reprinted by permission from Ristek Oy and Enterprisex Software Ltd).

6. Access to the model is added for User #2 and #3. This is done by adding rows on the model access table.

7. User #2 starts *Leonardo*. *GetModels* web service function is used for getting the available models. The model *Demo* is available and User #2 opens and loads it from the cloud. The data of the model is saved to local storage. Web service functions *Connect* and *Load* are used.
8. User #2 reserves the node *Walls*. *Check out* web service function is used. A *node checked out* event is added on the event table. User #2 starts to model the walls for the building under the *Walls* node.
9. User #1 checks the changes from the server. There is an event for reservation made by User #2 and node *Walls* is marked as reserved and locked for User #1. Web service function *Get changes* is used.
10. User #2 has finished the modelling of walls and releases the node *Walls*. *Check in* web service function is used. Changes made locally are uploaded on the cloud model and a *node checked in* event is added on the event table.
11. User #1 checks the changes from the server. There is an event for the release of the node *Walls* made by User #2 and User #1 will get the walls modelled by User #2. Web service function *Get changes* is used. Local storage is also updated. *Leonardo* application had developed specific wall classes that are not available for other applications. *Leonardo* wall class is derived from the public wall class that is available for all applications in BPC assembly. *3DTrussMe* cannot instantiate *Leonardo* wall object but instead uses generic wall object, which can show the geometry of walls.
12. User #1 reserves the node *Roof*. *Check out* web service function is used. A *node checked out* event is added on the event table. User #1 starts to model the roof to the building under the *Roof* node.
13. User #3 starts *Viewer*. *GetModels* web service function is used for getting the available models. The model *Demo* is available and User #3 opens it from the cloud. Web service function *Connect* is used. After first connect, User #3 sees the base hierarchy of the model and the reservation status of nodes.
14. User #3 loads the node *Walls* and he can see the walls of the building. Web service function *Load* is used. The local storage is also updated. *Viewer* can only instantiate class objects that are defined in BPC assembly. All entities must implement their geometrical representation using the public geometry classes available in BPC assembly. Thus, the *Viewer* application can show the geometry of all objects and also for the application specific objects that cannot be instantiated using the original class. Figure 12 shows a snapshot of the 3D model in *Viewer* application.

Fig. 12. Walls in Viewer application after step #13. Viewer uses generic public wall classes for showing the geometry. (Reprinted by permission from Ristek Oy and Enterprise Software Ltd).

15. User #1 notices that one wall is at a wrong position and reserves it. The position is a property defined by the generic wall in BPC assembly. *3DTrussMe* application can use the generic wall instance for changing the position. Web service function *Check out* is used. One event is added to the table of events.
16. After changes are made, User #1 releases the wall and the shared cloud model will be updated. Web service function *Check in* is used. One event is added to the table of events.
17. The data model of *Leonardo* is further developed. New attributes for the wall class are added and a new class concrete column is programmed.
18. User #2 updates *Leonardo* and updates the local model by getting changes from the cloud model. The position change for one wall made by User #1 is visible. User #2 reserves *Walls* node, changes the node name to *Walls and Columns*, and models columns under it. Web service functions *Get changes* and *Check out* are used. A *node checked out* event is added on the event table.
19. User #2 releases the node tree *Walls and Columns* and the cloud model will be updated. Web service function *Check in* is used. A *node checked in* event is added on the event table.
20. User #1 has finished the modelling of the roof trusses and releases the *Roof* node. The cloud model will be updated. Web service function *Check in* is used and a *node checked in* event is added to the table of events.

21. User #3 updates the local model. User #3 has not yet loaded *Roof* node, thus only changes made to *Walls and Columns* are updated. Web service function *Get changes* is used.
22. User #3 loads the *Roof* node and trusses modelled by User #1 will be visible. Web service function *Load* is used.
23. User #1 updates the local model and gets the latest changes for node *Walls and Columns* made by User #2. Web service function *Get changes* is used.
24. User #2 updates the local model and gets the latest changes for node *Roof* made by User #1. Web service function *Get changes* is used. Final model is shown in Figure 13.

Fig. 13. Final model after step #23 in *Leonardo* application. (Reprinted by permission from Ristek Oy and Enterprixe Software Ltd).

25. User #2 unloads the *Roof* node and roof trusses are no longer available. Local function *Unload* is used.
26. All users disconnect from the model and close applications. Session identifiers are invalidated on the server side. Web service function *Disconnect* is used.

Table 4 shows the executions steps with users involved, simplified description and check out nodes after each step on a tabular form.

Table 4. Steps for the execution.

Step	User #1	User #2	User #3	Description	Check out node(s) after event
#1	X	X	X	Registration of users	
#2	X			Model establishment	
#3	X			Whole model check out	Whole model for User #1
#4	X			Base hierarchy for the model is defined.	Whole model for User #1
#5	X			Whole model check in	
#6		X	X	Access to established model is added	
#7		X		Connect and full load of the shared model on the cloud.	
#8		X		Partial model Walls checked out	Walls for User #2
#9	X			Model updated	Walls for User #2
#10		X		Check out of partial model Roof	Roof for User #1
#11	X			Local model synchronized. Partial model Walls is updated.	
#12	X			Partial model Roof checked out	Roof for User #1
#13			X	Connect to the shared model on the cloud	Roof for User #1
#14			X	Load of the partial model Walls	Roof for User #1
#15	X			Change of the position for one wall	Roof for User #1, Wall for User #1
#16				Wall checked in	Roof for User #1
#17		X		Further development of the specific data model for <i>Leonardo</i> application	Roof for User #1
#18		X		<i>Leonardo</i> application updated and local model synchronised. The position of one wall updated.	Roof for User #1, Walls for User #2
				Partial model Walls checked out.	
#19		X		Node name Walls changed to Walls and Columns and checked in.	Roof for User #1
#20	X			Partial model Roof checked in on the cloud storage	
#21			X	Local model synchronised. Walls and Columns node tree updated.	
#22			X	Partial model roof loaded from server to local.	
#23	X			Local model synchronised. Partial model Walls and Columns updated	
#24		X		Local model synchronised. Partial model Roof updated	
#25		X		Partial model Roof unloaded from runtime memory of the application. A load is needed for getting Roof back for the application	
#26	X	X	X	Disconnect from the shared model. Connect must be made for continuing the work with the model.	

3 Results

Computational principles (Chapter 2.1) for the building information modelling are the same for all BIM software. There is no difference whether the developed system is for the distributed data management or for the single model concept. The data handling method does not determine software technology or data requirements. The dissimilarity only appears from the needed functionality, where the main difference is in methods for conversions and data sharing.

3.1 Prototypical implementation of the single shared data model system

The selection of tools and techniques is the first task for every programming project. In this research, the goal was not to find the best programming language or database type, but tools that are capable and suitable for the job. The selection was based on experience, knowledge, and familiarity achieved by a long-term working as software developer for the construction industry. Thus, C# was selected as the programming language and the relational database as the permanent storage solution. All coding was performed on the Microsoft Windows operating system, which understandably leads to the use of Microsoft software development tools and services only.

During this research, thousands of lines of code were written, which is typical with the software development. It is out of the question to present all the written code lines. Only the main principles of the software development can be shown.

3.1.1 Software data model

The key item of the software data model is the root class. The more functionality and features are implemented in the root class or in top classes in the inheritance tree, the less duplicate code exist in derived child classes. All data handling tools, including saving to the data database and reading back, can be defined in the model root class *BaseDataNode* in the following manner:

1. Data is defined as lists of name-value pairs only in the model root class *BaseDataNode*. In C#, lists can be defined as *Dictionary* data type (Microsoft Corporation, 2019). In each dictionary, every name must be unique, which is the requirement for attribute names in the scope of one list.

2. Derived classes get access to lists directly or by get and set functions. Using the get and set practice is the recommended and safer way.
3. Derived classes add desired attributes as items to dictionary collections, which collections are defined only in the root class *BaseDataNode*.
4. When saving the data into the database and reading it back, the mapping of the data is done fully with the root class *BaseDataNode* using dictionaries.

It could be possible to use only one dictionary list by converting all data as text or binary. However, for better human understanding and self-explanatory, defining the following dictionaries is advisable:

1. Dictionary for integers (*int* in C#).
2. Dictionary for texts (*string* in C#).
3. Dictionary for floating point numbers (*double* in C#).
4. Dictionary for true or false values (*bool* in C#).
5. Dictionary for two-dimensional points or vectors (*Point* or *Vector* in C# .NET Framework). With mathematics, the point and vector are different, but from the storage point of view, both are double values with x- and y-coordinates.
6. Dictionary for three-dimensional points or vectors (*Point3D* or *Vector3D* in C# .NET Framework).
7. Dictionary for internal links. This is used only for links between objects on the same model. In the data model, during the runtime of the application, the link address value is an instance of the source object. On the database, the link address value is the unique identifier GUID of the source object. When reading the data model from the database, links are first read as GUIDs. After the model is read, GUID links are parsed as object instances. When saving, the GUID is directly available from the object instance.
8. Dictionary for general links. Both the name and value of the link are text type and can be defined freely. The difference with respect to the internal link is that the value is not only a GUID and no link parsing as an object instance is made when reading the model from the database. A general link can be used for pointing outside the model, for group linking or for special purposes by an application. The name of the generic link restricts the usage to a fixed functionality by an application that recognises the name of the link. *See Links and relations* in Chapter 2.1.2 for more information.
9. Dictionary for pictures and other binary data has not been defined. Binary data can be converted as text and back, thus, saving as text is a feasible course of action. Also, in Chapter 2.1.3, it was presented, that XML schema is the best

solution for a database. XML is a text string and saving using XML schema requires that all data is converted as text when saved into a database.

A beam with a rectangular section needs a parameter height for defining the geometry. First, the model root class *BaseDataNode* must define the dictionary collection for doubles as a field variable:

```
private Dictionary<string, double> _DoubleAttributes;
```

Where *string* is the name of the attribute and *double* is the value. A good programming practice is to hide the class field variable with the *private* keyword, which makes the field inaccessible for derived classes. For the height of the beam section, the attribute name is *Height* and value is any reasonable dimension for the height of the beam. First, functions for getting and setting double values must be defined in the model root class. The *protected* keyword makes the functions available also for derived classes:

```
protected T GetPropertyValue<T>(string name) {  
    if  
        .....  
    } else if (typeof(T).Equals(typeof(double))) {  
        return (T)(object) _DoubleAttributes[name];  
    }  
    .....  
}
```

```
protected SetPropertyValue(string name, double value) {  
    if (IsValidPropertyName(name)) { _DoubleAttributes[key] = value; }  
    else { throw new ArgumentException("The given name is not a valid name.");}  
}
```

Above, the C# generic type parameter *T* is used, which enables the use of the same get function for all dictionaries. In the above code, only the code section for doubles is shown. The beam class defines the get and set property the section height in the following way:

```

public double Height {
    get { return (double) GetPropertyValue<double>("Height"); }
    set { SetPropertyValue("Height", 400); }
}

```

Similarly, all other dictionaries and all other attributes in all other classes can be handled.

Geometry classes

Model geometry is an indispensable portion of the building information model. It is the most common information that is shared between applications and involved people. Therefore, it is essential to include geometry definitions and classes in the data model. Otherwise, all developers define their own geometry structure which makes geometry sharing impractical. A common geometry handling is achieved with the following steps:

1. Classes for the geometry definition are designed. Triangular mesh is a commonly used data format for geometry nowadays and is valid also for the BIM data model. Geometry classes are never standalone, they require a physical model object as a parent. A root class *ModelGeometry3D* for geometry is designed from which all other geometry classes are derived. Both triangular mesh geometry and parametric geometry need to be supported (see *Geometry* in Chapter 2.1.2).
2. A root class *ModelBaseNode* for model physical objects is developed, which defines the required geometry interface and functionality. All classes for physical objects are derived from the model physical root and they must implement the geometry tools defined by the physical model root class. Geometry for a physical model object is implemented by a geometry class as a child.

Triangular mesh is defined by points, lines, and triangles. A C# class *MeshTriangle* for a mesh triangle is defined as follows:

```

public class MeshTriangle {
    protected int _vertex1Index;
    protected int _vertex2Index;
    protected int _vertex3Index;
}

```



```

    protected int _surfaceIndex;
}

```

Index attributes of the mesh triangle are saved to a database through the dictionary as presented earlier. Each triangle can contain visible and invisible lines. That information can be stored as a sign of the vertex point index. A negative value is a start to an invisible line, a positive one to a visible line. A surface index is pointing to the surface indexes of the solid. Here, a surface is thought to have only visible edge lines. As an example, a polyhedron has six surfaces each with two triangles. When the edge line visibilities of triangles are known and each triangle holds the index to a surface, six surfaces of the polyhedron can easily be constructed (Figure 14). The wired mesh of only visible vertex lines of the solid can also be easily constructed when the visibility index of lines in triangles is stored.

The rotation order of triangle vertex points must be fixed for knowing the visible side of the triangle face. Counter clockwise rotation when looking from outside of the solid is the positive mathematical direction and is thus a self-evident choice. Then any cross product between consecutive line vectors on a convex surface produces a plane normal vector that points out from the solid.

Fig. 14. Constructing the surfaces of a solid is a straightforward operation, when triangles have the visibility flag for lines and the surface index is attached to each triangle. A surface is then constructed by combining the visible edge lines of triangles in the right order.

Solid geometry made up of triangles is defined as a list of triangles and points. Triangles are *MeshTriangle* classes and are saved as child nodes for the solid entity class *SolidMesh*. Points are 3D point attributes in the point or vector dictionary in a similar way as described earlier for double attributes. The key to each indexed point is P_n , where n is the index of the point. Getting and setting points can be programmed using C# indexer in the following way:

```

public System.Windows.Media.Media3D.Point3D this[int index] {
    get {
        string sKey = "P" + index.ToString();
        return GetPropetyValue<Point3D>(sKey);
    } set {
        SetPropertyValue("P" + index.ToString(), value);
    }
}

```

For parametric geometry, saving points and triangles is unnecessary because the solid mesh can be generated on the fly with a few lines of code. Classes *RectangularCuboid*, *ExtrudedSection*, and *Cylinder* represent the very basic parametric shapes. Root class for all geometries is named as *ModelGeometry3D*, from which *SolidMesh*, *RectangularCuboid*, *Cylinder*, and all other geometry classes are derived. *ModelGeometry3D* is an abstract class for common definitions and specifies functions for getting solid meshes, surfaces, visible lines, and vertex points:

```

public abstract List<SolidMesh> GetSolidMeshes()
public abstract List<Surface3D> GetSurfaces()
public abstract List<Line3D> GetVisibleEdgeLines()
public abstract List<Point3D> GetVertexPoints

```

All functions are defined as abstract, which in C# requires that all derived classes must implement their own function. The root for the model physical objects is defined as a *ModelBaseNode* class. It is the root for all entities that have a real physical outlook defined as a class derived from *ModelGeometry3D*. *ModelBaseNode* implements the next function for getting the geometry:

```

public virtual ModelGeometry3D GetGeometry3D(int accuracyLevel)

```

The accuracy level is an attribute for *ModelGeometry3D* class. Each physical object can implement multiple geometrical representations with different accuracy that are all implemented as *ModelGeometry3D* objects. Applications can then request the geometry with desired accuracy from physical objects in the building model.

A wooden purlin with a rectangular section is defined as a class *Purlin*, that is originally derived from *ModelBaseNode*. The geometry is defined as a straight extrusion using class *ExtrudedSection*. *Purlin* overrides the *GetGeometry3D*

implementation and returns an *ExtrudedSection* instance. *ExtrudedSection* is parametric geometry, which generates the solid mesh geometry on the fly. Therefore, no triangular mesh as points and triangles is saved into database but instead a parametric geometry. It should be noted that after a Boolean operation parametrising is lost, which requires the saving of the solid mesh with triangles and points as a child object *SolidMesh*. Another solution is to save the Boolean operation, but it requires that applications are able to do the requested Boolean task. Light applications, like those running on mobile devices, usually do not support Booleans. That must be noted, when deciding whether to save the Boolean operation or the results of it.

Each real physical entity in the model is derived originally from *ModelBaseNode* and each class for a physical object implements the *GetGeometry3D* function. The returned instance of type *ModelGeometry3D* offers functions for getting triangle meshes, surfaces, lines, and points of the geometry for various purposes like 3D shading, line graphics for drawings, part numbering, hidden line generation, and Boolean operations. More geometrical shapes can be added by deriving new classes from *ModelGeometry3D* and new physical objects are added by creating new classes by derivation from class *ModelBaseNode*. This way the access to the geometry remains the same.

The development of the software data model classes must start from the data model root class *ModelBaseNode*. All classes must be originally derived from that root, since it only implements the indispensable data handling as described earlier. Geometry is the most important shared data and must be accessible for all applications. Therefore, all geometry classes derived from *ModelGeometry3D* must be publicly available for all applications. In principle, the model root class and geometry classes are the only ones that are absolutely required to be public. The publishing of classes does not require the publishing of the source code. Instead, an assembly containing the compiled code of classes is delivered for the use with other applications. Publishing only the assembly prevents class changes and guarantees that all applications use fully equal classes. The key public classes are published as BPC assembly. In this research, over 150 data classes were developed for public use and are available as BPC assembly.

The software data model is programmed as a list of classes. Table 5 summarises the key items of the data model development. Figure 15 shows UML class diagram for the basic derivation hierarchy for the key classes. The diagram includes only very basic attributes and functions of classes. The full implementation of classes

cannot be shown in one page due to the big number of classes and class attributes and functions. Appendix 1 lists all the classes developed during the research work

Data arrangement

The hierarchical model arrangement is implemented simply by defining a parent for each object. Technically, parent definition is similar to the internal link. On the database, a parent is saved as a unique identifier GUID and parsed as an object instance when reading the model data from the storage. Both the free and fixed hierarchies (see Chapter 2.1.5) are defined equally.

Table 5. Key items for the data model development.

Key item	Description
BaseDataNode	The root class for the data model classes. It is the carrier for all data in dictionaries and handles all saving and reading functionality between the data model and database. The data model root class must be public and available for all application developers. Must be included in the BPC. All classes must be originally derived from BaseDataNode class.
ModelBaseNode	The root class for real physical entities that require a geometrical representation. All classes for physical objects must originally be derived from ModelBaseNode. It defines the interface for the functionality of the geometry, that all derived classes must implement. Must be included in the BPC.
ModelGeometry3D	The root software class for the geometry definitions. It defines the interface for pure geometrical issues. All geometry classes must be originally derived from ModelGeometry3D. A geometry class is never a standalone but is always connected to a class derived from ModelBaseNode. Complex geometry can be saved as a triangular mesh with indexed points and triangles, whereas parametric shapes can generate the required geometry on the fly. Must be included in the BPC.
Public classes	Data model classes that are publicly available for all. BaseDataNode and classes defining geometry must be public. For better collaboration and data sharing public portion should include as many classes as possible. There must exist single public assembly (BPC) that contains key classes and is used by all data model developers.
Private classes	Application specific classes that are not freely available for all stakeholders. It is a decision by the developer whether to make classes public or not. All private classes must originally be derived from a BPC class.

Fig. 15. An UML diagram for the basic inheritance hierarchy of the data model classes. The full list of BPC classes is listed in appendix 1.

3.1.2 Data storage

The database must store all the data of object instances, hierarchical arrangement and object unique identifier. Dictionaries hold all class data as presented in previous chapter 3.1.1. With XML schema it is a simple task to move data from dictionaries to XML and back. XML definition contains *XmlElement* node, which includes *XMLAttributeCollection* for a list of attributes (Microsoft Corporation, 2019). Each *XMLAttribute* has a name and value, which is identical to data dictionaries of a class.

To enable the use of the *CreateInstance* function for the *late binding* technique, *AssemblyQualifiedName* must also be stored (see Chapter 2.1.4). Saving the whole data model into a single relational database table requires the following five columns:

1. Unique identifier of the object. In SQL Server database the column type is UNIQUEIDENTIFIER.
2. Unique identifier of the parent object for storing the hierarchical arrangement. In SQL Server database the column type is UNIQUEIDENTIFIER.
3. *AssemblyQualifiedName* of the data model class for the use of the function *CreateInstance*. In SQL Server database the column type is NVARCHAR.
4. Name of the object as text. Every object in the data model has a name which makes it reasonable to have a separate column for the name. In SQL Server database the column type is NVARCHAR.
5. Data of the object as XML. In SQL Server database the column type is XML.

CreateInstance function will fail if the assembly containing the compiled code of the class is not available. One solution would be to create the data model root class, which is available for everybody in BPC assembly. Better practice is to instantiate the BPC class that is the lowest in the derivation tree. As an example, when BPC contains a generic wall, all wall classes that are derived from that generic wall and are in private assemblies, can be instantiated as BPC generic walls. Since all data is handled by the data model root class, no data loss will occur although the class is not created as the original. Naturally, all the functionality of the private wall class is not available when using generic BPC wall class. To enable generic class instantiating instead of data model root, one column must be added for the database table:

6. *AssemblyQualifiedName* of the BPC data model class for the use of function *CreateInstance*. Used for instantiating the class if assembly containing the

original class is not available. In SQL Server database the column type is NVARCHAR.

AssemblyQualifiedName of the original class and BPC class can easily be inquired in C# programming language. Class does not need to define the name itself but it can be obtained automatically from the system when saved to data storage. Figure 16 shows the basic steps for opening a model from the database.

Fig. 16. A diagram showing the basic steps made during the opening of the model from the database.

3.1.3 Data sharing

Access to the cloud database of the shared model cannot be direct and public. It is a risk for security, data integration, and integrity to make a cloud database visible over the internet. A secure web service that encapsulates the required functionality is the correct technique. The list of needed functions was presented in Chapter 2.1.7. Each function must include user credentials in preventing illegal access, thus, the list of authorised users must be available on the cloud service. Updating the local storage incrementally requires an event log, which must also be stored on the cloud database.

Authorising users

The minimum information needed for each authorised user is the unique identifier, name, and password. This can be stored in a database table of three columns:

1. Unique identifier for the user. This is generated automatically during user registration. In SQL Server database column type is UNIQUEIDENTIFIER.
2. User login name. Common practice nowadays is to use the email address as the login name. In SQL Server database column type is NVARCHAR.
3. User password. In SQL Server database column type is NVARCHAR.

User credentials must be saved in a secure way and there are many techniques available for storing and hashing passwords (Microsoft Corporation, 2019). For simplicity, passwords were not hashed in this research, which cannot be the way in a real world system. Those three columns are the minimum information needed about the user but usually more information is stored. The first name, last name, address, phone number, and company are typical information that is stored for the user.

All users are not allowed to access all models. For that, another table with the following columns is needed:

1. Unique identifier of the user. In SQL Server database column type is UNIQUEIDENTIFIER.
2. Unique identifier of the model. This is the identifier of the root node of the model. Each model has one and only root node and all other nodes are descendants of the model root. In SQL Server database column type is UNIQUEIDENTIFIER.

Every web service call must identify the user and model. This can be simplified by generating a temporary session identifier when the user connects to the shared model through the login function (see Chapter 2.2.1). The session identifier can then be used with the other web service functions instead of user credentials and model. Sessions need a table with these columns:

1. Session identifier that is generated automatically when a user connects to the model. In SQL Server database column type is UNIQUEIDENTIFIER.
2. Unique identifier of the user. In SQL Server database column type is UNIQUEIDENTIFIER.
3. Unique identifier of the model. In SQL Server database column type is UNIQUEIDENTIFIER.

4. Validity flag. This is invalidated during logout.
5. Last access time. This is updated with every call to service. For unexpected behaviour and disabling everlasting sessions, a time out period must be defined. After certain unused time, a session will automatically become invalid. In SQL Server database column type is SMALLDATETIME.

Using the session identifier instead of user credentials and model identifier is a more secure technique. The session table also implements the recording of the service usage.

Event log

The following events need to be recorded for enabling fluent and conversion-free data sharing:

1. Model established. A single event for each model when it is created.
2. Node added. An event when a new node is added under an unreserved parent node.
3. Node deleted. An event when a node is deleted under an unreserved parent node.
4. Node checked out. An event for a reservation of a node for editing. The node with all its descendants is reserved. Only one reservation for one node is allowed. Two users cannot reserve the same node simultaneously.
5. Node checked in. An event for releasing a node and all its descendants. The whole tree is also updated into the shared cloud storage.

The data of events is stored in a database table with the following columns:

1. Model identifier. This is the unique identifier of the model root node. Needed for a case, when multiple models are saved on the same cloud database. In SQL Server database column type is UNIQUEIDENTIFIER.
2. Unique identifier of the user who did the event. In SQL Server database column type is UNIQUEIDENTIFIER.
3. Event type. See previous list. In SQL Server database column type is NVARCHAR.
4. Unique identifier of the node for the event. In SQL Server database column type is UNIQUEIDENTIFIER.
5. Unique identifier of the parent node for the event. In SQL Server database column type is UNIQUEIDENTIFIER.

6. Event time. The exact time for the event. In SQL Server database column type is DATETIME.

When the events are sorted by the time, it is an easy task to get all events for the defined model after defined time. Incremental updates also require, that applications keep track of the last update time. The list of reserved nodes is obtained by going through the check out and in events.

Table 6 shows all the columns for user, model access, session, event, and model tables.

Table 6. Columns for user, model access, session, event, and model tables in the cloud database. Many of the columns are used in multiple tables.

Column	User	Model access	Session	Event	Model
User identifier	X	X	X	X	
User login name	X				
User password	X				
Model identifier		X	X	X	
Session identifier			X		
Validity flag			X		
Last access time			X		
Event type				X	
Node identifier				X	X
Parent node identifier				X	X
Event time				X	
AssemblyQualified Name					X
Node name					X
Node data as XML					X
AssemblyQualified Name for BPC class					X

Web service functions

The use of the single shared model in the cloud goes through a web service. It is a list of functions which encapsulates the use of the data model. The needed functions for shared model access on the cloud are listed in Chapter 2.1.7. A skilled programmer is able to write the code for the web interface functions. As an example, the *check-out* function has the following parameters:

1. Session identifier that has been generated during the user login and returned to the client. This is an *in* parameter.

2. Last update time. The time when the client model has been last updated. This is an in parameter.
3. Model node that will be checked out. This is an in and out parameter. No descendants need to be sent to the server. In case the node or any of its descendants have changed after the last update time, the latest version of the whole node tree is returned into the client. When the node tree is reserved for editing, the latest version of the data must always be loaded into the client side.
4. Node changed flag, which shows if the reserved node tree has changed after the last update time. This is an out parameter.
5. Return code which indicates the success of a function. There can be many reasons for the failure of the function. The most common error is that a node has already been reserved by another user.

The function will first check that the incoming session identifier exists at the session table and is still valid. The user and model identifier can then be read from the session table. After that, the event log table is used for finding out the current reservation status of the node. If the whole node tree is unreserved, reservation can be made by adding a reserved event to the event log table. Before returning to the client, *check-out* function will check if the node has changed after the incoming parameter last update time. That is easily done by going through the reserve and release events on the event log table. If all is successful, no failure code is sent back to the client. If there is an error, the failure code in question will be returned.

3.2 Feasibility of conversion-free data exchange

Conversion-free data exchange can be realised if and only if just one data format is used. All applications must use exactly the same data model schema, which must be open and public to all participants. The continuous evolution of BIM data content fully prevents the use of a fixed and standardised information only. Standards have their places, but they are not enough for a competitive business. For example, IFC is a good standard for building information. Nevertheless, it is only used as an exchange format, because commercial applications cannot store all needed information with IFC.

The expandability of the data model schema requires a division into public and private portions of the data elements. The public portion must be available for all, whereas the private parts are the specific content of applications. The definition of the new content is made by adding new classes, which raises the question of

compatibility. It is out of the question that every change or addition to the data model schema would require coding changes to all other applications.

The technical specification for a freely expandable data model keeping the compatibility backward and forward was presented in Chapter 3.1.1. The key points of the definition are as follows:

- The data model schema is divided into public and private portions. The public portion is available for all applications and the private part is the extension of the data content made by various applications.
- All private classes must be based on public classes. In software technology, this is called derivation. A derived class inherits its attributes and functionality from the parent class.
- Data carriers and all data saving into and reading from the permanent storage is handled by public classes. Data carriers are defined as the lists of name-value pairs. Private classes define new content by adding new attributes to the lists of data carriers, but do not need to take care of the data saving and reading. Data saving and reading is done through attribute lists, and the count of items on each list can vary without incompatibility issues.
- The class creation method that requires class availability during the compile time is not used because with that method, all classes must be known at compile time, and every addition requires a new compilation of all applications. With information technology, the compile time method is called *static* or *early binding*, and the runtime method is called *late* or *dynamic binding*. Using the *late binding* method enables the addition of new classes into the data model freely at any time by all developers. To enable the late binding of the class, the information technological name is saved into permanent storage. The IT name of the class is a part of the *type* definition in the area of software technology.
- In addition to the information technological name of the original (private) class, the type name of the public class from which the derivation has started is saved to the storage. The public class is instantiated instead of the private class if the original private class is not available during the application execution time. The public class can carry all the data of the original class without problems.
- The permanent storage schema allows data content changes (adding new classes to the data model) without any coding changes.
- A software development kit (SDK) for the development of applications includes public data classes and local storage. The use of an SDK compels applications to use the same data format. An SDK also implements all the key

points of this list, and thus, applications developed using an SDK automatically realise all key points and are compatible with each other. Therefore, an SDK is an essential element for conversion-free data exchange.

The dependent pair for the data model is the permanent storage, which must also fully support the expandability of the data content. The database schema solution for the expanding data content was presented in Chapter 3.1.2. The whole data from the class instances are mapped using an XML format and stored in one column on the database. The mapping of name-value pairs from dictionaries into XML and back is a straightforward programming task. Furthermore, using an XML format, there will not exist any compatibility issues because of changes on the definition of the data content. The database schema is independent of the count of items on the attribute lists.

3.3 Pros and cons of the conversion-free data exchange

No conversions are needed when using only one data model and database format. Conversions are problematic because of data loss, conflicts and incompatibility between diverse data formats (Oh et al., 2015; Redmond et al., 2012; Volk et al., 2014). There is no benefit in using private data formats and conversions generally. Keeping the data private and hidden from other applications because of competitive reasons could be one argument for the use of a proprietary and private data formats.

Data duplicates are not prevented by using only one data model and storage structure. A single model approach, where all the data are saved only once, is needed for removing the overlapping information. The single shared model is accessed simultaneously by all participants, which is a significant difference when compared with distributed private storages on numerous computers. The technical method for sharing the data within a single model approach is presented in Chapter 3.1.3, which is based on the following key elements:

- The single shared database is placed on the cloud, enabling an equal access for all participants.
- Access control is implemented limiting the entry only for registered and authorised users. Furthermore, access to models per user is controlled.
- All connections to the shared cloud model are performed through a web service that has the required functionality (Chapter 2.1.7). Direct access to the database storage is not allowed for keeping up the integrity and integration of the data (Chapter 2.1.6). It is a security risk to allow direct access to the database, which

may allow for illegal and incompatible changes in the database. When all data operations are executed through specific functions, only legal and valid data changes will occur.

- A pessimistic concurrency control is used (Chapter 2.1.6). The editable portion must be reserved before making any changes. During reservation, other users can see the unchanged version but are not allowed to make any changes to the reserved portion. When releasing the edit lock, changes are made visible to the other users.
- An event log keeps track of reservations, releases and data change events (Chapter 3.1.3). Event recording enables the monitoring of the reservation state and incremental updates of the local storage. Hence, loading the whole model from the cloud with every open of the model is not needed.
- The model arrangement is realised using a hierarchical approach (Chapter 2.1.5). With the hierarchy, the model can be divided into numerous and varying size subtrees. These subtrees are called partial models, which are the units for reservation. Applications can also limit the size of the model loaded onto the running memory by loading only the desired partial models from the permanent storage.
- A public SDK is used for the development of applications enforcing the compatibility. An SDK also eases establishing connections to the single shared model.

More and more data with more details are included in building information models. Conversion has problems, and diverse private data formats are not fully compatible. The linked building data approach adds links between model parts in diverse files. All these features increase the complexity of the building data. With the single shared model approach, no conversions and links between diverse models are needed, which will decrease the complexity of the data. Internal links inside each model between different objects are required much like in the distributed data management and single model approach. The variety of these links and relations was explained in Chapter 2.1.2.

In the presented database schema for the single model (Chapter 3.1.2), all the data are saved as a list of name-value pairs using an XML-format. The database is fully uncorrelated with the content of the data. The number of different classes or the number of class attributes has no meaning for the storage schema. All classes are saved and read using the same method. Hereby, an increase in the data complexity because of the expansion of the data content definition does not concern

the presented database schema solution for the single shared model approach. The semantic of the stored data is based on the attribute names. The more self-explanatory the name is, the more semantic the data will be. The best semantics are available through the original private class, which will have a full understanding of all attributes. If a private class is not available and a public class is instead instantiated, the data semantics will be worse depending on how much new content the private class has defined.

The overall progress of information technology is fast, and old techniques are substituted with newer ones. During the whole lifespan of a building project, the information technology will take significant development steps. It is a challenge to keep up with the digital data available and readable. With distributed data management, information is stored using divergent data formats, which all must be maintained. A neutral open data format such as IFC can be used if it carries all information without defects and inconsistencies. With the single data format, only one data saving method needs to be kept alive. When a widely used storage format, such as a commercial relational database system, is selected, the best guarantee for the data availability during the whole building life-cycle can be achieved.

The content of the data model evolves all the time creating an issue for the backward and forward compatibility. When newer software can handle older data, it is called backward compatibility. With forward compatibility, older applications can open newer data. For a software developer, backward compatibility is usually easy: the change is known, which makes the needed coding action straightforward. With forward compatibility, the change will happen in the future and is not known when writing the code. Therefore, forward compatibility very rarely exists, but backward compatibility can be seen as a standard required feature for applications.

In the execution of step #15, User #1 updated the position of a wall. In step #17, User #2 had a newer version of the *Leonardo* application and handled the older version of the wall without errors. This is a proof for the backward compatibility.

In the test run, the specific data model of the *Leonardo* application changed when a new class and new attributes for an existing class were added (step #16). User #2 updated *Leonardo* by adding a new class and changing the attributes of an existing class (step #17). Users #1 and #3 received those data changes without updating their applications (steps #20 and #22). These steps are a proof for the forward compatibility.

The data integrity and integration improve when data sharing occurs without conversions and when no overlapping information exists. This is achieved with the presented single data schema and single shared model approach. Linked data

increases data complexity, but with the single model, external links between the diverse files are dispensable. The data complexity management is less complicated when all data are on a single model. In the test run, all modelled data of all applications were stored on the same database on the cloud (step #2). Users must register with the system and, furthermore, have the right to access the model (steps #1 and #6). Every user can reserve a desired part of the model with the *Check out* functionality (steps #3, #8, #12, #15, and #17). The *Check out* event gives the editing rights for the model part in question. Because no duplicate check outs are allowed, there will never be overlapping or contradictory changes, which guarantees the data integration of the whole model.

A drawback is the incompatibility with the current industry practice. Existing applications cannot start using the single shared model approach without considerable changes being made to the application code. Another weakness is the size of the single shared model, which will grow notably larger than with distributed data management. However, the hierarchical arrangement implements partial models, which will ease the handling of huge models.

3.4 Technical realization for the industrial use

The main outcome of the programming work was the software development kit (SDK), which is freely available for all software developers. SDK is a compiled set of assemblies which include the required interfaces, functionality, and public data model classes. The source code of the *Viewer* application is also available free of charge. The *Viewer* application code can be used as a template for developing new 3D design modelling software. The programming of new software starts by taking the *Viewer* application code as a base and referring the assemblies from SDK to the development project. The other two applications, *Leonardo* and *3DTrussMe*, which were used on the execution of the test, are commercial applications, thus the source code of those two programs is not publicly available.

The cloud service used in the experimental test was implemented under web pages hosted by Enterprixe Software Ltd. The service is upgraded regularly and the latest version availability can be seen on Enterprixe web pages. Offering the cloud service for the single model sharing is a commercial business, therefore, the source code or compiled code of the cloud service is not available publicly or for free.

Three applications and the cloud service form up a true running system for the single shared model approach. The only outcome and substantiation of the

execution of the test event were the permanent data storage for the single shared model.

3.4.1 Permanent storage content

During the execution of the test event, data was collected and updated into four databases. On the cloud, there was a shared cloud database and each application had its own local database. Local databases were used as a local cache and for minimising the data transfer over the internet. The shared cloud database was the master storage and the data was synchronised between the cloud and local databases.

During the execution of the experiment, the content of five database tables changed many times. The results are not shown step by step for each table. For tables *User*, *Model Access*, *Session Identifier*, and *Event*, rows were only added thus presenting the final state is enough. The size of the *Model* table does not allow the presentation of all the data.

No conversions occurred during the execution of the test. All data between the cloud and local application storages was transferred unchanged.

User table

On the execution of step #1, three users were added to the table *User* on the cloud database. The result is shown in Table 7. The *User* table is needed for keeping up the list of registered users who are allowed to login on the cloud service.

Table 7. User table content after the execution of step #1. Three users have been added. The left most column is the row number of the table.

No.	User identifier	User login name	User password
1	127DE12E-432A-428D-A398-19BB11285FFA	User #1	3*qZnHdD
2	2087569B-E2C1-4E04-9853-7994851B5A8A	User #2	bxmvP%75
3	42D0F345-F382-425D-8A46-C3191E310A35	User #3	2FgW%9W!

Model access table

To enable the user access to the model, a row at the *Model access* table must exist. Without the *Model access* table, all users could access all models. On the test there was only one model, but in a real implementation there are many models under the

same service. The required rows were added in steps #2 and #6. The resulting rows are shown in Table 8.

Table 8. Model access table for users. Access to the test model is granted for all three users.

No.	User identifier	Model identifier
1	127DE12E-432A-428D-A398-19BB11285FFA	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8
2	2087569B-E2C1-4E04-9853-7994851B5A8A	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8
3	42D0F345-F382-425D-8A46-C3191E310A35	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8

Session table

A session id is generated when the user connects to the model with valid user credentials. The user must have registered to the service (Table 7) and have access to the model (Table 8) for getting a valid session id. The session id is used instead of credentials with the following calls after the initial login and connection to the cloud service. When a user disconnects from the model, the session id will be invalidated. After defined amount of unused time, session id will also be invalidated, which prevents session id staying valid forever. During the execution of steps #2, #7, and #13, valid session ids were generated and, on step #26, all were invalidated. Table 9 shows the content of the *Session* table after the execution of the test event. Session id binds the user and model and without the session id concept all calls to cloud service should define user credentials and the model. Sessions also contain the information of active connections.

Table 9. Session identifiers after the execution of the test event. Session identifiers for each user for the test model has been generated. Validity flag 0 is an invalid session identifier and 1 is valid. During the execution of the test, flag was 1 and after disconnection and logout flag was changed to 0.

No.	Session identifier	User identifier	Model identifier	Last access time	Validity flag
1	60CD60C8-0A67-	127DE12E-432A-	63C388D2-F1D8-	2017-08-31	0
	4E58-9C4B-	428D-A398-	48D3-A1A7-	14:41:31	
	D74C395DC86C	19BB11285FFA	7F42C450D4C8		
2	E96DF034-BA4A-	2087569B-E2C1-	63C388D2-F1D8-	2017-08-31	0
	4924-A1FC-	4E04-9853-	48D3-A1A7-	14:44:12	
	6535DEC2679C	7994851B5A8A	7F42C450D4C8		
3	2C8B71FC-61EF-	42D0F345-F382-	63C388D2-F1D8-	2017-08-31	0
	4D4D-9B8E-	425D-8A46-	48D3-A1A7-	14:43:09	
	6C64EBD3F17C	C3191E310A35	7F42C450D4C8		

Event table

During the execution of the test, events for *model established*, *check out* and *check in* occurred. Table 10 shows the recorded events on the *Event* table in the cloud database. In Table 11, a short description of each event is presented. Row numbers from 1 to 11 are the same on both tables and each row with an equal number is related to the same event. Events enable the bookkeeping of reserved nodes and incremental updates from the cloud to the client storage.

Table 10. Recorded events after the execution of the test event.

No.	User identifier	Model identifier	Event type	Node identifier	Parent node identifier	Event time
1	60CD60C8-0A67-4E58-9C4B-D74C395DC86C	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Model established	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	00000000-0000-0000-000000000000	2017-08-31 12:55:22
2	60CD60C8-0A67-4E58-9C4B-D74C395DC86C	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked out	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	00000000-0000-0000-000000000000	2017-08-31 12:56:14
3	60CD60C8-0A67-4E58-9C4B-D74C395DC86C	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked in	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	00000000-0000-0000-000000000000	2017-08-31 13:01:55
4	2087569B-E2C1-4E04-9853-7994851B5A8A	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked out	953D5B44-683E-48A4-97D0-D2B4E0BDE036	4B3507C3-04D0-49CE-AF95-09EEB1A695F8	2017-08-31 13:11:49
5	2087569B-E2C1-4E04-9853-7994851B5A8A	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked in	953D5B44-683E-48A4-97D0-D2B4E0BDE036	4B3507C3-04D0-49CE-AF95-09EEB1A695F8	2017-08-31 13:20:11
6	60CD60C8-0A67-4E58-9C4B-D74C395DC86C	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked out	938A627C-4C4E-42D0-B5B3-85D043B6D964	4B3507C3-04D0-49CE-AF95-09EEB1A695F8	2017-08-31 13:25:26
7	60CD60C8-0A67-4E58-9C4B-D74C395DC86C	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked out	9BA867F9-87B0-4A1E-BAAF-710718980D7A	953D5B44-683E-48A4-97D0-D2B4E0BDE036	2017-08-31 13:35:59
8	2087569B-E2C1-4E04-9853-7994851B5A8A	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked in	9BA867F9-87B0-4A1E-BAAF-710718980D7A	953D5B44-683E-48A4-97D0-D2B4E0BDE036	2017-08-31 13:46:12
9	2087569B-E2C1-4E04-9853-7994851B5A8A	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked out	953D5B44-683E-48A4-97D0-D2B4E0BDE036	4B3507C3-04D0-49CE-AF95-09EEB1A695F8	2017-08-31 13:50:05

No.	User identifier	Model identifier	Event type	Node identifier	Parent node identifier	Event time
10	2087569B-E2C1-4E04-9853-7994851B5A8A	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked in	953D5B44-683E-48A4-97D0-D2B4E0BDE036	4B3507C3-04D0-49CE-AF95-09EEB1A695F8	2017-08-31 14:15:48
11	60CD60C8-0A67-4E58-9C4B-D74C395DC86C	63C388D2-F1D8-48D3-A1A7-7F42C450D4C8	Node checked in	938A627C-4C4E-42D0-B5B3-85D043B6D964	4B3507C3-04D0-49CE-AF95-09EEB1A695F8	2017-08-31 14:25:03

Table 11. Recorded event steps and description. Row numbers from 1 to 11 are same as in Table 10. This table is not a database table but for clarifying Table 10 rows.

No.	User	Execution step	Event description
1	User #1	#2	Model establishment
2	User #1	#3	Whole model checked out
3	User #1	#5	Whole model checked in
4	User #2	#8	Walls node checked out
5	User #2	#10	Walls node checked in
6	User #1	#12	Roof node checked out
7	User #1	#15	Single wall checked out
8	User #1	#16	Single wall checked in
9	User #2	#18	Walls node checked out
10	User #2	#19	Walls and columns node checked in
11	User #1	#20	Roof node checked in

Model table

The shared information of the building model was saved into the *Model* table. The *Model* table contained about 5000 rows after the execution of the test event. Therefore, showing all the rows is unpractical and unreasonable. Table 12 shows some rows of the database, which were selected with SQL query statement. Only columns formulating the hierarchy are shown. The *Parent node name* column does not exist on the *Model* table, but it has been joined in the query for clarifying the hierarchical arrangement.

Table 12. Hierarchical arrangement in the model table. Parent node name column was joined with SQL query to the result and does not exist in the model table.

No.	Node identifier	Parent node identifier	Node name	Parent node name
1	9ACB2AE2-3A06-4642-BA27-E07EE8965140	00000000-0000-0000-0000-000000000000	Model root	
2	59D2487D-C49D-4044-A474-B848E8074858	9ACB2AE2-3A06-4642-BA27-E07EE8965140	Thesis	Model root
3	6641BA05-A97B-4448-9D12-8E9BC8B4AC87	59D2487D-C49D-4044-A474-B848E8074858	Building	Thesis
4	8987C3BB-0B56-438B-A8E8-36A84B8905E2	6641BA05-A97B-4448-9D12-8E9BC8B4AC87	Reference Planes	Building
5	4BD49D21-D7AA-4C04-9A11-437B2DA133C6	6641BA05-A97B-4448-9D12-8E9BC8B4AC87	Roof	Building
6	1A60C261-A332-46E1-B4FE-2484A96B9604	4BD49D21-D7AA-4C04-9A11-437B2DA133C6	Supports	Roof
7	125FA585-0DF5-4318-B5CE-F0201E0F1C9E	1A60C261-A332-46E1-B4FE-2484A96B9604	Support A	Supports
8	1B260261-F03C-428F-BC0F-B30E57B6680A	1A60C261-A332-46E1-B4FE-2484A96B9604	Support B	Supports
9	18C3C7E9-3C03-4089-BC07-29DC730F2EE6	1A60C261-A332-46E1-B4FE-2484A96B9604	Support C	Supports
10	A354B863-ED90-4839-B42D-6BB768E55F08	1A60C261-A332-46E1-B4FE-2484A96B9604	Support D	Supports
11	A1A80689-52F8-4F0A-8870-2E7140343283	4BD49D21-D7AA-4C04-9A11-437B2DA133C6	Ridge Trusses	Roof
12	07CC9AA9-3AEB-446A-8B8E-A627CB42A784	4BD49D21-D7AA-4C04-9A11-437B2DA133C6	Hip Trusses	Roof
13	DECB3662-EA4B-4513-8722-94D71C87F99E	4BD49D21-D7AA-4C04-9A11-437B2DA133C6	Hip End Trusses	Roof
14	C940192C-CEED-4EFA-BA26-70FACE9654BC	6641BA05-A97B-4448-9D12-8E9BC8B4AC87	Walls and Columns	Building
15	02806C3C-6C83-483F-8F75-066B10F7FB35	C940192C-CEED-4EFA-BA26-70FACE9654BC	Wall	Walls and Columns
16	AAF15213-2FB4-408F-B713-34310CAE4EB1	C940192C-CEED-4EFA-BA26-70FACE9654BC	Wall	Walls and Columns
17	91D1E602-E4D5-4EEB-853C-C2D6E56DAA8B	C940192C-CEED-4EFA-BA26-70FACE9654BC	Wall	Walls and Columns
18	21A4C80E-7952-4630-B9E5-209CEBC5B7CD	C940192C-CEED-4EFA-BA26-70FACE9654BC	Wall	Walls and Columns

Table 13 shows the whole content of one truss stored as one row on the *Model* table. Data of the truss is mapped in XML format. The XML tag names have been

shortened, like I for integers, P for three-dimensional points or vectors, and P2 for two-dimensional points or vectors. Attributes, which are used internally only by the application, can have short names that are not self-explanatory.

Table 13. Data row of one truss stored on the model table.

Column	Content
Node identifier	F29344E1-1976-452A-BDB3-7DEBA88FB7AB
Parent node identifier	DC00C0AE-099C-44B4-83B2-013F5CB374D9
Node name	Ridge Truss
AssemblyQualified Name	Epx.Ristek.Data.Models.ParametricTruss, LLDesigner.ProjectData
BPC AssemblyQualified Name	Epx.BIM.Models.PlanarTruss
Node data as XML	<pre> <R> <I BundleCount="1" BundleCountFire="1" /> <D BundleGap="3" Thickness="42" Spacing="600" /> <S Alignment="BottomCenter" UseClass="DryOutside" /> <B IEx="0" IS3="0" ISL="0" IsStiffeningStructure="0" UseOwnLoadCombinations="0" UseOwnSupportDeflection="0" USSF="0"/> <P> <O X="0" Y="4021" Z="0" /> <XA X="1" Y="0" Z="0" /> <YA X="0" Y="0" Z="1" /> </P> <P2> <DXFOrginShiftVector X="0" Y="0" /> <TransportDirection X="1" Y="0" /> <SymmetryAxisEndPoint X="4000" Y="3000" /> <SymmetryAxisStartPoint X="4000" Y="-20" /> </P2> </R> </pre>

During the test, a wall was edited by User #1 and User #2. User #1 was using *3DTrussMe* application and a public BPC class. User #2 was using *Leonardo* application and a private wall class derived from public BPC wall class. At the start, the private class contained an attribute called *Material*, that did not exist in the public BPC wall class. In execution step #17, *Leonardo* wall class was further developed and new attributes *Weightkg*, *Strength*, and *Shape* were added. Table 14 shows the stored data of the wall after check in events were made by users. The content of the wall data expanded during the test, but no incompatibility issues

occurred. The forward and backward compatibility did not broke at any time of the execution of the test.

Table 14. Stored wall data after execution of steps. On each step stored data changes but nothing disappears.

Step	Stored data after step	User
10	<pre> <R> <D Width="150" Height="2500" /> <S Alignment="BottomLeft" Material="Concrete"/> <P><O X="7903" Y="9978" Z="-1292" /><XA X="-7806" Y="0" Z="0" /> <YA X="0" Y="0" Z="1" /> </P> </R> </pre>	<p>Saved by User #2 using private <i>Leonardo</i> wall class</p>
16	<pre> <R> <D Width="150" Height="2500" /> <S Alignment="BottomLeft" Material="Concrete"/> <P><O X="8003" Y="9978" Z="-1292" /><XA X="-7706" Y="0" Z="0" /> <YA X="0" Y="0" Z="1" /> </P> </R> </pre>	<p>Saved by User #1 using a public BPC class, that does not have Material attribute defined</p>
19	<pre> <R> <D Width="150" Height="2500" Weightkg=7312.5"/> <S Alignment="BottomLeft" Material="Concrete" Strength="C35" Shape="Rectangle"/> <P><O X="8003" Y="9978" Z="-1292" /><XA X="-7706" Y="0" Z="0" /> <YA X="0" Y="0" Z="1" /> </P> </R> </pre>	<p>Saved by User #2 using private <i>Leonardo</i> class, that has been further developed by adding attributes Weightkg, Strength, and Shape</p>

Database validity

Database validity can be controlled by defining primary and foreign keys. A primary column enforces that a column cannot have equal value on multiple rows. Foreign key definition requires value existence on the referred table and column. Table 15 shows which columns on tables are defined as primary keys and which columns can be linked using foreign keys. During the execution of the test, the cloud database reported no errors related to primary and foreign key definitions.

Table 15. Primary and foreign keys for shared database tables on the cloud. Definitions help keep stored data valid.

Column	User	Model access	Session	Event	Model
User identifier	Primary	Foreign	Foreign	Foreign	
Model identifier		Foreign	Foreign	Foreign	
Session identifier			Primary		
Node identifier				Foreign	Primary
Parent node identifier				Foreign	Foreign

3.4.2 Expanding the industry coverage of the experiment

The experimental test included all the significant issues for conversion-free data sharing. The execution of the test produced a valid database storage for the single shared model. However, the test included only three applications that do not cover all the functionality needed by all building disciplines during the whole life-cycle of the building. Many more applications must join the single model ecosystem before the industrial coverage is acceptable. Hence, the following questions arise, when thinking about the expandability of the single shared model approach:

- Can all the data of numerous applications be defined and saved?
- Will compatibility remain between all applications when the content of the data is expanded independently by various applications?
- Can large models be handled?
- What is the interoperability level with existing systems?
- How much investments are required for making the industrial coverage acceptable?
- Is the industry interested in the single model approach?

The schema of the single data model and storage is version free as presented earlier. New data content by defining new classes can be added freely without breaking the forward or backward compatibility between applications. Thus, the data content evolution or the compatibility of the data is not a barrier.

As is the case now, with distributed data management, the sizes of some models can be too large for adequate computer performance (Johnson, 2014). With the single model approach, there will exist even larger models. The shared model is located on the cloud server, and local storages are updated incrementally. The model arrangement is implemented through the hierarchy, which formulates numerous counts of subtrees called partial models. Applications can utilise the

hierarchy for limiting the size of the model loaded onto the runtime memory at one time. In the experimental test, the size of the model was small, which does not show much regarding the overall execution speed. Calls to the cloud occur infrequently, so the server connection speed should not be critical for the overall performance. However, any slow functionality diminishes the reputation and is not accepted by users. The full-scale test for the execution speed can only be made with several different applications and numerous users accessing the same single shared model on the cloud.

The interoperability level of the presented single model approach with existing applications is infinitesimally small, which will greatly hamper the adoption. Conversions could be a helpful solution, but then, problems and defects in the conversions will be confronted. Adapting current systems would be a better way, but it might require an unacceptable amount of work. Finally, the interest and trust in switching to a new technology can be low, especially when the business is successful with the old approach. In summary, no technical barrier to the single shared model approach exists, but bad interoperability with existing systems, required investments and a lack of interest will keep the system from reaching wider.

4 Conclusions

The results of the research showed that a conversion-free building information sharing can be an alternative to the current practices that are based on conversions and native data formats. The best benefits are realised when the conversion-free concept is used together with the single model approach. The concurrent use of the model requires some extra principles and functionality for keeping up the integrity and integration of the data. On the other hand, no conversions and external links are needed. A real working single model system was developed as a part of the current research, which confirmed the technical validity of the conversion-free single model approach.

4.1 Conclusions for research hypotheses

The first hypothesis was that a conversion-free data sharing approach could work for building information modelling. The presented single data model and database schema can be used by all new and old BIM applications. An SDK is freely available and includes the base data model classes as a start point for the application-specific data content. Without limitations the schema allows the adding of the new content that applications need to store for their specific needs. The data model schema is a version-free solution, keeping the compatibility regardless of changes made by other developers. Putting this all together, the presented method makes up a feasible, conversion-free data exchange resolution based on a single extendable data model structure. The first hypothesis is correct: conversion-free data sharing is an alternative for distributed data management based on conversion. The use of a single data format makes conversions dispensable, and the single shared model approach realises a rational information sharing method.

The second hypothesis argued, that a conversion-free method would be beneficial, when compared with distributed data management using conversions. It is self-evident that working without conversions will eliminate all conversion shortcomings. Data duplicates will vanish when a single model schema is extended as a single shared model approach. The data integrity and integration improve when data sharing occurs without conversions and when no overlapping information exists. Linked data increases data complexity, but with the single model, the external links between diverse files are dispensable. The data complexity management is less complicated when all data are on a single model and when only one data format is used. The second hypothesis is correct: there are no benefits in

the use of conversions at least from the viewpoint of the information technology. Keeping native information unreadable for others could be one argument for private data models, but current information technology offers various other methods for hiding secret issues. Naturally, the current practice for the data sharing is distributed data management, which enforces to the use of conversions.

The last hypothesis claimed that no main technical impediment exists for the wider industrial use of the conversion-free data sharing concept. A true working single shared model system was developed, and the experimental test was executed successfully. Therefore, no technical barrier exists for the single model concept and conversion-free data sharing. To obtain the greatest benefit from the single shared model, software from various disciplines should be available. There is no limit regarding what types of applications can join the ecosystem: the only requirement is to use an SDK. A single model approach, made available throughout the whole life cycle of the building project is a potential data collaboration concept. In turn, this will create a seamless flow of information and increase the profitability of BIM process.

However, many questions can be asked about the wider industrial use of the presented conversion-free approach (see Chapter 3.3.2). The incompatibility with current practice, required investments and lack of interest restricts its expansion. The reputation and reliability of a new piece of technology or innovation is low at first. Rogers (2003) divides technology adopters into innovators (2.5%), early adopters (13.5%), early majority (34%), late majority (34%), and laggards (16%). The conversion-free single shared model approach needs innovators to start the technology diffusion. An SDK is free, but it is a risk to start the use of a brand new system that is not properly compatible with old data formats and systems. For new applications, the risk is lower because they do not need to worry about backward compatibility. Existing BIM applications have enormous lines of code, which, in the worst case and for practical reasons, mostly must be rewritten. Private and native data formats must be discarded, and old data should be converted into the new data format. The last hypothesis is only partially correct. The research showed that conversion-free data sharing is technically feasible, but many obstacles remain for its industrial use.

4.2 Single shared model with conversion-free data exchange

Cloud-BIM technology is considered as the second generation of the information management for the construction sector. It can improve cooperation, collaboration,

and real-time communication. Despite this, no review of the existing developments and research into cloud-BIM has been published. The major challenges for the adoption of cloud-BIM technology are i) a lack of clarity for the responsibility, liability and ownership of models, ii) a shortage of technology experts and iii) a need for professional education (Wong, Wang, Li, Chan, & Li, 2014).

No commercial single model system for multidisciplinary building information is available. Systems for sharing a model between the same applications have been developed (Chapter 1.2.2), but none can cross the application boundary. The objective for model sharing with *Tekla*, *Archicad* and *Revit* is to enable multiple people to work simultaneously with the same model. There is no reason to limit the model sharing only between the same applications as long as user and access control prevents conflicting and illegal changes. Indeed, according to Lu, Wu, Chang and Li (2017), there is a lack of BIM standards for model integration and management by multidisciplinary teams.

A single shared model located on the cloud offers a good data availability for all stakeholders. The latest up-to-date information is accessible to everybody, all the time. With distributed data management using conversions, there is always a delay in the data delivery. Conversions are not fully reliable because information can change and partly drop out. A single shared model that operates without conversions and is located on the cloud can be seen as the next generation of the cloud-BIM technology.

The fundamental concern for storing information in the cloud is security. Organisations and people have doubts whether their proprietary information stored on a cloud server is still private and confidential. The decision of adopting cloud technology depends on attitudes, knowledge, motivation, and intention to trust. On the other hand, the cloud provides opportunities for cost reduction by potentially improving collaboration, agility, scaling, and availability (Ho, Ocasio-Velázquez, & Booth, 2017). The current growing trend is towards cloud technology, and more and more, all kinds of information is stored on the cloud instead of local storage. However, when it comes to cloud storage, human nature often resists new experiences until enough advantages are known.

It is a common opinion among the AEC industry and BIM scientists that a single model BIM is an unfeasible solution. According to Day (2011), a single building model is only a daydream. On the other hand, Howard and Björk (2008) state that a single BIM is the holy grail, but there might not be willingness to achieve it. According to Turk (2016), a centralized shared database is impossible but in the future, BIM will approach it. The reasoning for this is mostly not

presented by these authors, but model differences between disciplines and the size of model are noted. Because of the rejection of the single BIM model, no research has been conducted of a true single shared model system. Under the umbrella term 'single BIM' scientific articles can be found, but they see single BIM as a common repository for distributed data sources. A cloud service or a single address to separate files is only one data delivery tool for distributed information. A true single model system is a shared database that can be accessed simultaneously by several users, and every piece of data is stored only once.

A single shared model is not a single application. This kind of system could be realised by adding tools for all disciplines and construction phases into a single application. However, this would lead to a large application with an unthinkable number of implemented functions. The key advantage of the single model is the shared information that can be utilised by smaller or larger applications. Each application can access only the desired part of the data that are needed for the functionality of the software.

The construction process includes many phases, from the conceptual design to demolition. It is slightly questionable if one single shared model can be used for all phases. First a wall can only be shown as 2D lines, and in the as-built model, the wall includes a detailed 3D geometry along with many kinds of non-geometrical information. In an object-oriented world, it is a big challenge to develop a class that can expand from a minimal data carrier to an entity holding realistic information about the final product. It might be technically possible, but more than one single shared model during the whole building life-cycle is a better practice. However, every construction project phase does not need a separate single model. At least when the detailed design starts that single model should be used for all following phases.

Many applications and many users are not usually interested in handling all the entities or all the data of the entities. A structural engineer is generally the only one who wants to see the reinforcement of concrete parts, while other participants want to simplify the model by ignoring the reinforcement inside the concrete. According to Beetz, Borrmann and Weise (2018) there are two solutions for defining data subsets or simplifying the data content: Model view definition (MVD) for IFC-based data exchange and level of development (LOD). LOD is not only for the geometry, but also for the other information attached to an object. The American Institute of Architects (AIA) has defined six LODs: LOD100, LOD 200, LOD 300, LOD 350, LOD 400 and LOD 500 (Weygant, 2011). In the single model solution presented in the present dissertation, LOD was used, but only for the geometry in

which case LOD is defined as level of detail. With a hierarchical arrangement, a model can be organized in desired subsets or partial models, but the best tool for filtering or simplifying the data is applications. When the single model contains all the information, it is a task for each application to filter, handle and show only the data that is needed for the scope of the application in question.

Sometimes, the content of the model required by different applications differs so much, that the use of a single object becomes cumbersome. This can be the case with the architectural and structural design, which typically handle the same building objects. With distributed data management, architectural and structural models are separate. With the single model approach, there are three possible solutions. Models can be separate, but then, model objects should be connected together with external links. Another method is to model the architectural and structural parts on the same model on a different hierarchy as partial models. The best concept is to use software derivation, where a structural class is derived from the architectural parent. This requires that the structural class can develop the content of the architectural class further without breaking the compatibility. The most troublesome building element is a wall on which an architect and structural engineer can have a different view of.

When two applications with an overlapping file of operation add new content to the data model schema, it is obvious that duplicate classes for the same purpose are created. This can be avoided by defining enough detailed public classes, but this requires a standardisation process accepted by all parties. A duplicate private class for the same purpose does not break the compatibility because public class can be used instead. However, standardisation would be a better method because it prevents the definition of equal content in multiple classes.

With some cases, the security aspects might deny the availability of all the data for all participants. One possible solution could be to let applications filter out the forbidden information. However, unallowed data cannot be handled at the application level: it must be controlled by user rights. The data model schema should implement a user access right setting at the entity level and even at the individual attribute level of entities.

According to Johnson (2014), the complexity of design tasks and software evolution raise questions about a single model solution. It is true that tasks performed by engineers and consultants are complex and various. Many kinds of applications are used for design tasks, and all software is evolving continuously. However, is that complexity troublesome only for the single shared model approach? The distributed data sharing system uses many data formats and conversions. Is this

more complex when compared with the single shared model operating without conversions? The data model and storage schema of the single model system presented in the current dissertation is extendable without breaking the backward and forward compatibility. It does not categorise data into complex and simple, but instead all data are stored using the same technical solution. Thus, the data complexity mainly arises through applications that are utilising the shared data.

A BIM model includes a lot of relations between entities, which greatly increases the data complexity. In the presented single model system, relation links can partially be decreased with a hierarchical arrangement by replacing the link as a parent-child relationship. On the other hand, links requiring a write access mix the optimal size of the reservation share. A link with a change access to the source data is problematic in all building model systems. Although editing rights are mostly not controlled at model level in distributed data sharing systems, changing everything in diverse and private building models is not freely allowed in practice. An HVAC engineer does not have the right to make an opening in the wall for the duct. With distributed models, an HVAC engineer can freely make the opening on his or her own model, but it has no practical meaning. The opening must be made to the architect's or construction engineer's models. On a single model, there is only one wall, and the HVAC engineer should ask the architect or construction engineer to make the opening. Instead of a human agreement about areas of responsibilities, keeping the wall reserved to one party will prevent changes being made by other users. However, controlling allowed actions based on partial models, entity types, users, user roles, and applications would be a better resolution. The simplest control can be realised by applications, which create and change strictly only those objects that belong to the scope of each application.

The demand for the wall opening can be handled by defining a relation between overlapping objects. The link between the duct and wall should be stored under the wall. With this direction for the relation link, the wall can control if openings emerging from ducts are practicable and valid. Always when possible, the link should be handled by the object that is changing because of the relation. On the other hand, is the link for the opening necessarily needed if the system includes a collision check that reports for needed holes? Every link increases the complexity, and for that reason the necessity of a link must always be thought carefully.

More and more information is stored in building information models, leading to a growing model sizes. With distributed files, individual models do not grow as big as a single model. However, distributed models contain duplicate information, making the combined size of all the files even bigger. In a distributed data

management system, each separate file can be seen as a partial model, whereas a single model must be divided into smaller portions using an arrangement. The hierarchical arrangement is a well-known method for organisation and will realise a partial model concept for a single model. Although the model size on the disk is large, by loading only a smaller partial model or models, the amount of data handled by an application at one time can be kept suitable. There is not a single hierarchy that could be used with all models. Every construction project is different from many perspectives. Very rarely will two buildings be exactly the same, and people have their own preferences. Thus, the model arrangement cannot be static but must be freely defined by users. Therefore, because of the nature of building objects, a fixed hierarchy is needed (see Chapter 2.1.5).

The current research produced a conversion-free single shared model system, and the experimental test proved the validity of the developed method. Thus, a single model should no longer be a utopian idea, at least from a technological viewpoint. The distributed information sharing system is currently then dominant one among stakeholders, especially in the academic community. Therefore, it is a rocky road to success for the single model system.

According to Miettinen and Paavola (2014), the benefits of using BIM is often reported by researchers and project participants, but the impact of BIM is difficult to isolate from the success of the entire project. Moreover, increases in the productivity in the construction business have been marginal when compared with other industry sectors (Koskenvesa, 2011). Assuming, that information technology has a notable influence on the improvement of productivity, why has digitalisation succeeded in other industry sectors but not in the construction business?

According to Johnson (2014), the scepticism against “One BIM” is alleviated with a data oriented model offering the following:

- An open source vendor-neutral elastic data structure.
- Enabling the interoperation of applications from multiple vendors.
- Sharing data in the design ecosystem without explicit import or export.
- Supporting different kind users, tasks, workflows, and stages in the design process.

It can be seen that the single shared model method presented in the current dissertation, will realise all the items in the above list. However, fulfilling the list will require a significant adjustment from current BIM practices and processes.

4.3 Reliability and validity

The testing of a software system is not a one-time process. There are immeasurable routes between the start and end of the application run. For ensuring 100% functionality, a complete test should go through all the alternative paths, which is impossible. However, a well-designed single test can be a proof for the principle operations of the programme. Passing the prime test confirms that a software system is able to perform the originally designed activities. The experimental test presented in Chapter 2.3.3 is the principle test for the software developed as part of the current dissertation.

A single open and scalable data model format is a requisite for sharing information without conversions. In the test run, all three applications were using the same data model on the cloud, but in the reality, many more applications are used with BIM. The number of applications is not limited because all software houses can develop or change their programmes' compatibility by using a public SDK, nor is the number of users limited to three as it was in the test run. However, the growing number of applications and users accessing the same model concurrently can slow down the performance.

Each model has links that break the abovementioned arrangement. It is not feasible to model a construction project without a single link, but every link will mix the optimal hierarchical arrangement. When the hierarchy is broken by links, the unity of partial models will lose its absolute accuracy; thus, caution must be used when designing the links for a hierarchically arranged model. Read-only links are less problematic because they do not need editing rights like write links. In the experimental test, read, write, and group links were used (see Chapter 2.1.2). The material was defined as a read link, the saw cuts between wooden members used write linking, and equal trusses were connected with group linking. No external links were used because with the single shared model, they are not necessary. With a linked material definition, a change in material properties will also change the entity. Members belonged to a truss, which was defined as the smallest editable share (see *smallest editable share (SES)* in Chapter 2.1.5). When a truss is reserved, the saw cut relations and members become editable. Thus, a saw cut relation causes no trouble. With group-linked equal trusses, when reserving the change rights for a truss, all equal trusses must be checked out.

Partial models are a fundamental feature of the presented single model approach and are involved in almost all functionalities of the data handling. A *Check out* action reserves the change rights for the partial model, which the user

can freely select and then make changes to (steps #3, #8, #12, #15, and #17). A *Check in* event releases the partial model in question and updates the changed data from the local storage and places them into the cloud (steps #5, #10, #15, #18, and #19). The *Get changes* function updates the changes of partial models made by other users from the cloud into local storage (steps #9, #11, #17, #20, #22, and #23). The target of the *Load* and *Unload* functions is a partial model that can be freely defined (steps #7, #14, #21, and #24). With partial models, the editing rights of each object can be given to one user at a time, which ensures data integration (see Chapter 3.2). Every user can freely push their actions into a partial model they have selected. In summary, a single model without partial models is practically inoperable.

In the test run, the model was located on a cloud server (step #2) and was available for all users at all times. Every user has the opportunity to get the latest up-to-date information from the cloud storage with the *Get changes* functionality. The data of the model were constantly available for all stakeholders and provided the latest information. The only exception was partial models, which were checked out. The changes made during the *Check out* period were only visible for the user editing the partial model in question. Changes made in the checked out partial models were saved to local storage during of the *Check out* time. Only when a user selected the *Check in* command was the data updated on the cloud storage system and made available for other users. Keeping incomplete work private is a natural behaviour for people. We do not want to show our works in progress to others but rather publish a finished task. With the *Check out* and *Check in* functionality, users can keep their work private until it is ready for publication. After check out, all data of the partial model in question will be available equally to all stakeholders who have access to the shared model.

Every database system has its limits. In the current research, a Microsoft SQL server was used as the cloud storage system and several size limitations exist. The maximum size of a SQL Server database is 524 272 terabytes (Microsoft Corporation, 2019), which is an incredible amount of information. Before the maximum size limit is reached, other performance issues will probably arise. Client applications do not need to make calls to the cloud service non-stop, but database queries can slow down the cloud service when the database is large. In the test run, there were only three users and one model on the cloud, which naturally cannot show much of the performance for real projects.

The client applications will *Connect* to the cloud service and model, *Load* the desired parts of the model into the local storage, *Check out* and *Check in* partial

models, and update changes made by other users with *Get changes* function. In a practical setting the interval between these events is measured more in hours than in minutes. The cloud service does not need to execute numerous non-stop calls without delays. Most of the time, clients are not calling the cloud, meaning the response time of the service is not critical for the overall performance of the system.

Microsoft LocalDB was used as a local storage system on the client side. The maximum size of the LocalDB database is 10 GB (Microsoft Corporation, 2019) which is much less than the maximum size of a SQL Server database. Clients do not need to load the whole server model, but they can control, with *Load* and *Unload* functions, what is available locally. However, local storage capacity will most obviously be the first bottleneck before any performance problems on the cloud service appear.

An internet service provider (ISP)¹⁵ and independent software vendor (ISV)¹⁶ for web service are needed for offering the cloud service for the single shared model. In the experimental test, only a minimal service without any special user interface was used. The required model and user settings were handled directly using SQL Server Management Studio. A commercial solution needs an internet service that has a friendly user interface for creating and modifying user and model settings. Furthermore, the cloud service should be up and running for the whole lifespan of the project, which, at maximum, is from the early design steps to demolition.

Figure 17 shows all the major players of the building project. Owners are top players, having the access control for all other participants. Owners own the real estate, and they should also own the digital twin¹⁷. ISVs should not hold a monopoly position in their field of activity. Application development is freely available for all enabling multiple software on the same purpose. Developing the web service can also be done separately by multiple ISVs. The developer and provider of the web service do not need to be the same. The web service is an application that can be purchased the same as any other software. Owners have the most interest to run the cloud service through the whole life cycle of the building project. ISPs and ISVs are commercial players, who can offer the needed software and services for setting up the cloud service.

¹⁵ An Internet service provider (ISP) is an organization that provides services for accessing, using, or participating in the internet.

¹⁶ An independent software vendor (ISV) is an organization specializing in making and selling commercial software.

¹⁷ A digital twin is a replica of physical assets, processes and systems. A digital twin is a real mapping of all components in the product life cycle using physical data, virtual data and interaction data between them.

There is only one public SDK, and a private commercial enterprise is not the best ISV for SDK. A public non-profit corporation or alliance would be a better ISV operator for the public SDK. When multiple ISVs are developing the cloud service independently, the compatibility must be ensured. It is a task for the ISV of the SDK to control the development of cloud services and check their validity and compatibility. The same validation concerns the development of applications.

Fig. 17. The players around BIM model during the whole life-cycle of the building project.

4.4 Economical aspects

According to Giel and Issa (2013), there is no industry-accepted standard for tracking the BIM’s return of investment (ROI). Nevertheless, various studies have reported high ROI results. BIM has a huge potential, but the implementation of BIM is not fully exploited, even in leading countries such as Finland, United Kingdom, New Zealand, the United States, and Canada (Ghaffarianhoseini et al., 2017).

BIM is still mainly used during the design and construction phases. The data flow more or less breaks off when the construction phase ends. According to Aziz, Nawawi and Ariff (2016), the valuable information for facility managers can only be found in scattered sources. Matějka et al. (2016) states that BIM has considerable advantages over facility management (FM), and many scientific publications advocate FM adopt BIM (Edirisinghe, London, Kalutara, & Aranda-Mena, 2017).

A single shared model method is an alternative for distributed data management using conversions. The data flow operates without breaks and conversions from the early design steps to demolition. With an SDK, applications

can be developed for the whole life cycle of the building, not just for the design and building phases. The more applications join the single shared model ecosystem, the more potential for financial savings there will be.

4.5 Recommendations for future work

According to Zhao (2017), one of the hot topics in BIM research is mobile computing. Smartphones and tablets are not used for elementary modelling, just mainly for viewing and small data editing. Mobile devices have small storage capacities; thus, they cannot load and save the full model locally. Smartphone and tablet applications must load the required portion of the data from the cloud for every run instead of using local storage. Mobile applications need their own mobile SDK for connecting to the cloud service. This will mostly have the same functionality as the SDK developed in the current research. The main difference is the local storage that will not be a part of the mobile SDK.

Web browser applications are for nearly similar purposes as mobile applications. Data querying, viewing, and lightweight changes are typical tasks for a web browser tool. Like smartphones and tablets, a web browser application typically runs without local storage; thus, the mobile SDK is also a practical tool for the development of web browser applications.

The integration of geographic information system (GIS) and BIM has recently emerged as an important research area. Studies have shown clear benefits for data sharing between BIM and GIS. The spatial information of GIS can be used for outdoor modelling, whereas BIM mainly handles indoor data (Kang & Hong, 2015). The key point for BIM and GIS integration is the location. The position systems do not need to be the same, but an exact conversion between locations must exist. Mobile devices have huge potential for BIM and GIS integration, especially regarding facility management. A smartphone or tablet application can automatically locate itself in the building information model. The application can then collect the nearby objects, which would greatly narrow the selection list shown to a user.

In true modelling, various companies from various disciplines take part in the modelling. When a partial model is checked in, it will be available for all model participants. A more acceptable type of action would be to first submit changes for supervision made by a company or group manager. Only after this validation will changes be published and made available to other groups. This functionality requires the definition of groups in addition to users. Every group will have a

manager responsible for the work of the group. To avoid having groups working with the same partial model, each group should first reserve a partial model. Check out by a single user should be limited to those partial models that are reserved for the group that the user belongs to. This kind of reservation and releasing can be seen as multilevel *Check out* and *in*. Group reserving can be called *Branching* and releasing *Publishing* for separating them from the single user *Check out* and *in* functionality.

Groups can also be hierarchical in the case a branch can only be made from the parent group branch and the publish function will make the branched tree available just for the parent group. After the validation the parent group can publish the branch one step up in the group hierarchy. Hierarchical groups and the *Branch-Publish* functionality would clarify the modelling responsibilities between various disciplines and enable the validation of the work done by group members. Preidel, Borrmann, Mattern, König and Schapke (2018) present three states for the data: *work in progress*, *shared* and *published*. For transition to the next state, information checking or authorising is needed. Preidel et al. (2018) also present the fourth state of *archived* for information that has been superseded, withdrawn or rejected during the data management. Hierarchical groups on the single shared model system could be a technique for implementing these four states.

The semantics of the data are based on the programmatic names of the attributes. Only the original class has a full understanding about the meaning. For the simplest cases, such as *width*, *thickness*, and *length*, the semantic is clear. However, with most attribute names, the semantic is not enough with the programmatic name only. An explanation of every attribute should also be stored in the model. Only this would enable the utilisation of the stored information without the availability of the original class. An example of this type of functionality is direct queries made on the database of a single shared model.

The drawback of a single shared model approach is its incompatibility with the current existing data formats. All data classes in old applications must be rewritten to accomplish full compatibility. This will likely be a threshold that many software houses are not willing to surpass. This means that conversions from existing formats are needed for constructing a data bridge to the new single model method presented in the current research; otherwise, the new system will be isolated without any connections to old data systems. One solution could be to rewrite all IFC classes by starting the derivation from the BPC classes that are available on the public SDK. After doing this, importing the IFC files could easily be done. However, exporting to IFC cannot properly carry all the data included in the single

shared model using a version-free data structure. IFC is not a fully open and extendable format like the single model schema presented in the present dissertation. It is possible to add new property sets for objects and use IfcProxy for entities that are not supported by IFC (Borrmann, Beetz, Koch, Liebich & Muhic, 2018). However, this will lead to a remarkable number of IfcProxies having totally different content if the applications start to export all the native data. Extending the IFC standard definition by adding new classes into the schema will break both the backward and forward compatibility, requiring coding changes in every system using IFC.

List of references

- Autodesk Inc. (2018) Autodesk. URI: <https://www.autodesk.com/>. Cited 24.6.2018.
- Aziz, N. D., Nawawi, A.H., & Ariff, N. R. M. (2016). Building Information Modelling (BIM) in Facilities Management: Opportunities to be Considered by Facility Managers. *Procedia - Social and Behavioral Sciences*, 234(Supplement C), 353–362. <https://doi.org/10.1016/j.sbspro.2016.10.252>
- Balla, M., & Mohacsi, P. (2013). Budapest (HU) Graphisoft, assignee. Active building information modeling apparatus and method. Patent US 8352218.
- Barbosa, M. J., Pauwels, P., Ferreira, V. & Mateus, L. (2016). Towards increased BIM usage for existing building interventions. *Structural Survey*, 34(2), 168–190. <https://doi.org/10.1108/SS-01-2015-0002>
- Beetz, J. (2009). Facilitating distributed collaboration in the AEC/FM sector using Semantic Web Technologies. Technische Universiteit Eindhoven.
- Beetz, J., Borrmann, A., & Weise, M. (2018). Process-Based Definition of Model Content. In: Borrmann A, König M, Koch C & Beetz J (eds) *Building Information Modeling: Technology Foundations and Industry Practice*. Springer International Publishing AG, 181–197. https://doi.org/10.1007/978-3-319-92862-3_6
- Berdonosov, V., Zhivotova, A., & Sycheva, T. (2015). TRIZ Evolution of the Object-Oriented Programming Languages. *Procedia Engineering*, (131), 333–342. <https://doi.org/10.1016/j.proeng.2015.12.411>
- Beynon-Davies, P. (1991). Relational database systems: A pragmatic approach. Hartnoll Ltd, Bodmin, Cornwall, Blackwell scientific publications.
- Borrmann, A., Beetz, J., Koch, C., Liebich, T., & Muhic, S. (2018). Industry Foundation Classes: A Standardization Data Model for the Vendor-Neutral Exchange of Digital Building Models. In: Borrmann, A., König, M., Koch, C., & Beetz, J (eds), *Building Information Modeling: Technology Foundations and Industry Practice* (pp. 81–126). Springer International Publishing AG. https://doi.org/10.1007/978-3-319-92862-3_5
- BuildingSMART. (2018). buildingSMART International. URI: <http://www.buildingsmart-tech.org/>. Cited 7.2.2018.
- Chang, K-H. (2015). Chapter 3 - Solid Modeling. In: Chang K (ed) *e-Design* (pp. 125–167). Boston, Academic Press.
- Day, M. (2011). The trouble with BIM. AECMagazine.
- Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). BIM Handbook. Hoboken, New Jersey, John Wiley & Sons, Inc.
- Edirisinghe, R., London, K. A., Kalutara, P., & Aranda-Mena, G. (2017). Building information modelling for facility management: Are we there yet? *Engineering, Construction and Architectural Management*, (24)6, 1119–1154. <https://doi.org/10.1108/ECAM-06-2016-0139>
- Finnish Standards Association. (2017). SFS-EN ISO 29481-1.
- Garzás, J., Pino, F. J., Piattini, M., & Fernández, C. M. (2013). A maturity model for the Spanish software industry based on ISO standards. *Computer Standards & Interfaces*, 35(6), 616–628. <https://doi.org/10.1016/j.csi.2013.04.002>

- Ghaffarianhoseini, A., Tookey, J., Ghaffarianhoseini, A., Naismith, N., Azhar, S., Efimova, O., & Raahemifar, K. (2017). Building Information Modelling (BIM) uptake: Clear benefits, understanding its implementation, risks and challenges. *Renewable and Sustainable Energy Reviews*, 75(Supplement C), 1046–1053. <https://doi.org/10.1016/j.rser.2016.11.083>
- Giel, B. K., & Issa, R. R. A. (2013). Return on investment analysis of using building information modeling in construction. *Journal of Computing in Civil Engineering*, 27(5), 511–521. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0000164](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000164)
- Graphisoft. (2018). BIM server. URI: http://www.graphisoft.com/bim_server/. Cited 7.2.2018.
- Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, U. S. (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47, 98–115. <https://doi.org/10.1016/j.is.2014.07.006>
- Ho, S. M., Ocasio-Velázquez, M., & Booth, C. (2017). Trust or consequences? Causal effects of perceived risk and subjective norms on cloud technology adoption. *Computers & Security*, 70(Supplement C), 581–595. <https://doi.org/10.1016/j.cose.2017.08.004>
- Howard, R., & Björk, B. (2008). Building information modelling – Experts' views on standardisation and industry deployment. *Advanced Engineering Informatics*, 22(2), 271–280. <https://doi.org/10.1016/j.aei.2007.03.001>
- Isikdag, U. (2012). Design patterns for BIM-based service-oriented architectures. *Automation in Construction* 25, 59–71. <https://doi.org/10.1016/j.autcon.2012.04.013>
- Isikdag, U., Aouad, G., Underwood, J., & Wu, S. (2007). Building information models: A review on storage and exchange mechanisms. *Proceedings of 24th W78 Conference* (pp. 135–144). Bringing ITC Knowledge to Work, Maribor,.
- Isikdag, U., & Underwood, J. (2010). Two design patterns for facilitating Building Information Model-based synchronous collaboration. *Automation in Construction* 19(5), 544–553. <https://doi.org/10.1016/j.autcon.2009.11.006>
- Johnson, B. R. (2014). One BIM to Rule Them All: Future Reality or Myth? In: Kensek KM & Noble DE (eds) *Building Information Modeling: BIM in Current and Future Practice*. John Wiley & Sons, 175–185.
- Juola, R. (2014). *Rakennuksen tietomallin palvelin pohjaisen tietokannan tallennusratkaisu*. Oulu, University of Oulu, Faculty of Information Technology and Electrical Engineering, Department of Computer Science and Engineering, Computer Science.
- Kajava, J. (2017). *Development of the three-dimensional nail plate structure design software*. Oulu, University of Oulu, Faculty of Technology, Mechanical Engineering.
- Kamali, M., & Hewage, K. (2016). Life cycle performance of modular buildings: A critical review. *Renewable and Sustainable Energy Reviews*, 62, 1171–1183. <https://doi.org/10.1016/j.rser.2016.05.031>
- Kang, T. W., & Hong, C. H. (2015). A study on software architecture for effective BIM/GIS-based facility management data integration. *Automation in Construction*, 54(Supplement C), 25–38. <https://doi.org/10.1016/j.autcon.2015.03.019>

- Koskenvesa, A. (2011). Rakennustyön tuottavuus 1975–2010. In: *Anonymous Rakentajan kalenteri 2011* (pp. 138–146). Rakennustieto Oy.
- Laiserin, J. (2002). Comparing Pommès and Naranjas. *LaiserinLetter*, 15. URI: <http://www.laiserin.com/features/issue15/feature01.php>. Cited 16.10.2013.
- Lee, Y., Eastman, C. M., & Solihin, W. (2016). An ontology-based approach for developing data exchange requirements and model views of building information modeling. *Advanced Engineering Informatics*, 30(3), 354–367. <https://doi.org/10.1016/j.aei.2016.04.008>
- Lu, Y., Wu, Z., Chang, R., & Li, Y. (2017). Building Information Modeling (BIM) for green buildings: A critical review and future directions. *Automation in Construction*, 83, 134–148. <https://doi.org/10.1016/j.autcon.2017.08.024>
- Matějka, P., Kosina, V., Tomek, A., Tomek, R., Berka, V., & Šulc, D. (2016). The Integration of BIM in Later Project Life Cycle Phases in Unprepared Environment from FM Perspective. *Procedia Engineering*, 164(Supplement C), 550–557. <https://doi.org/10.1016/j.proeng.2016.11.657>
- Microsoft Corporation. (2019). Microsoft developer network. URI: <https://msdn.microsoft.com>. Cited 7.2.2019.
- Miettinen, R., & Paavola, S. (2014). Beyond the BIM utopia: Approaches to the development and implementation of building information modeling. *Automation in Construction*, 43, 84–91. <https://doi.org/10.1016/j.autcon.2014.03.009>
- Nadkarni, P. M., Marengo, L., Chen, R., Skoufos, E., Shepherd, G., & Miller, P. (1999). Organization of heterogeneous scientific data using the EAV/CR representation. *Journal of the American Medical Informatics Association*, 6(6), 478–493. <https://doi.org/10.1136/jamia.1999.0060478>
- NIBS. (2019). National BIM Standard-United States ver 3. URI: <https://www.nationalbimstandard.org/>. Cited 13.7.2019.
- Nicał, A. K., & Wodyński, W. (2016). Enhancing Facility Management through BIM 6D. *Procedia Engineering*, 164, 299–306. <https://doi.org/10.1016/j.proeng.2016.11.623>
- Oh, M., Lee, J., Hong, S. W., & Jeong, Y. (2015). Integrated system for BIM-based collaborative design. *Automation in Construction*, 58, 196–206. <https://doi.org/10.1016/j.autcon.2015.07.015>
- Pauwels, P., McGlinn, K., Törmä, S. & Beetz, J. (2018). Linked Data. In: Borrmann A, König, M., Koch, C., & Beetz, J. (eds). *Building Information Modeling: Technology Foundations and Industry Practice* (pp. 181–197). Springer, Cham.
- Peng, J., Kim, C., & Jay Kuo, C. (2005). Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation* 16(6), 688–733. <https://doi.org/10.1016/j.jvcir.2005.03.001>
- Preidel, C., Borrmann, A., Mattern, H., König, M., & Schapke, S. (2018). Common Data Environment. In: Borrmann, A., König, M., Koch, C., & Beetz, J. (eds). *Building Information Modeling: Technology Foundations and Industry Practice* (pp. 279–291). Springer International Publishing AG.

- Qtaish, A., & Ahmad, K. (2015). Model mapping approaches for XML documents: A review. *Journal of Information Science*, 41(4), 444–466. <https://doi.org/10.1177/0165551515579995>
- Radulovic, F., Poveda-Villalón, M., Vila-Suero, D., Rodríguez-Doncel, V., García-Castro, R., & Gómez-Pérez, A. (2015). Guidelines for Linked Data generation and publication: An example in building energy consumption. *Automation in construction* 57, 178–187. <https://doi.org/10.1016/j.autcon.2015.04.002>
- Rakennustietosäätiö RTS. (2013). Common BIM Requirements 2012.
- Rantanen, T. (2015). Espoo (FI) Tekla Corporation, assignee. Managing integrity of shared data in a manner permitting delayed updates. Patent US 8966189.
- Redmond, A., Hore, A., Alshawi, M., & West, R. (2012). Exploring how information exchanges can be enhanced through Cloud BIM. *Automation in Construction*, 24, 175–183. <https://doi.org/10.1016/j.autcon.2012.02.003>
- Ristek Oy. (2018). 3DTrussME - 3D design software for wooden trusses. URI: http://ristek.fi/wp/?page_id=10&lang=en. Cited 27.1.2018.
- Rogers, E. M. (2003). Diffusion of Innovations (5th edition). New York, USA, Free Press.
- Schapke, S., Beetz, J., König, M., Koch, C., & Borrmann, A. (2018). Collaborative data management. In: Borrmann A, König M, Koch C & Beetz J (eds) *Building Information Modeling: Technology Foundations and Industry Practice* (pp. 251–277). Springer, Cham.
- Shen, W., Hao, Q., Mak, H., Neelamkavil, J., Xie, H., Dickinson, J., . . . Xue, H. (2010). Systems integration and collaboration in architecture, engineering, construction, and facilities management: A review. *Advanced Engineering Informatics*, 24(2), 196–207. <https://doi.org/10.1016/j.aei.2009.09.001>
- Singh, V., Gu, N., & Wang, X. (2011). A theoretical framework of a BIM-based multi-disciplinary collaboration platform. *Automation in Construction*, 20(2), 134–144. <https://doi.org/10.1016/j.autcon.2010.09.011>
- Törmä, S. (2013). Semantic linking of building information models. *Proceedings - 2013 IEEE 7th International Conference on Semantic Computing, ICSC 2013*, 412–419. <https://doi.org/10.1109/ICSC.2013.80>
- Trimble Inc. (2018). Trimble. URI: www.tekla.com. Cited 3.2.2018.
- Turk, Ž. (2006). Construction informatics: Definition and ontology. *Advanced Engineering Informatics*, 20(2), 187–199. <https://doi.org/10.1016/j.aei.2005.10.002>
- Turk, Ž. (2016). Ten questions concerning building information modelling. *Building and Environment*, 107, 274–284. <https://doi.org/10.1016/j.buildenv.2016.08.001>
- Vanathi, B., & Uthariaraj, V. R. (2011). Comparing and choosing appropriate database for storing context in context aware system. *Journal of Computer Science*, 7(7), 1027–1032. <https://doi.org/10.3844/jcssp.2011.1027.1032>
- Volk, R., Stengel, J., & Schultmann, F. (2014). Building Information Modeling (BIM) for existing buildings — Literature review and future needs. *Automation in Construction*, 38, 109–127.

- Wette, C., Pierre, S., & Conan, J. (2004). A comparative study of some concurrency control algorithms for cluster-based communication networks. *Computers & Electrical Engineering*, 30(8), 615–636. <https://doi.org/10.1016/j.compeleceng.2004.08.003>
- Weygant, R. S. (2011). BIM Content Development: Standards, Strategies, and Best Practices. Hoboken, New Jersey, John Wiley & Sons, Inc.
- Wilson, P. R. (1985). Euler Formulas and Geometric Modeling. *IEEE Computer Graphics and Applications*, 5(8), 24–36. <https://doi.org/10.1109/MCG.1985.276212>
- Won, J., Lee, G., & Cho, C. (2013). No-Schema Algorithm for Extracting a Partial Model from an IFC Instance Model. *Journal of Computing in Civil Engineering*, 27(6), 585–592. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0000320](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000320)
- Wong, J., Wang, X., Li, H., Chan, G., & Li, H. (2014). A review of cloud-based bim technology in the construction sector. *Journal of Information Technology in Construction*, 19, 281–291.
- Wood, J., Panuwatwanich, K., & Doh, J. (2014). Using LOD in Structural Cost Estimation during Building Design Stage: Pilot Study. *Procedia Engineering*, 85, 543–552. <https://doi.org/10.1016/j.proeng.2014.10.582>
- Xu, X., Ma, L., & Ding, L. (2014). A Framework for BIM-Enabled Life-Cycle Information Management of Construction Project. *International Journal of Advanced Robotic Systems*, 11(1). <https://doi.org/10.5772/58445>
- Zhao, X. (2017). A scientometric review of global BIM research: Analysis and visualization. *Automation in Construction* 80(Supplement C), 37–47. <https://doi.org/10.1016/j.autcon.2017.04.002>
- Zhu, J. (1994). Conversion of Earth-Centered Earth-Fixed Coordinates to Geodetic Coordinates. *IEEE Transactions on Aerospace and Electronic Systems*, 30(3), 957–961. <https://doi.org/10.1109/7.303772>

Appendices

Appendix 1: List of public data model classes available on software development kit (SDK). Indentation demonstrates the class derivation hierarchy.

<i>BaseDataNode</i>	Root class for all data model classes
<i>DataModel</i>	Root for the data model, container for all data model nodes
<i>Project</i>	Class representing project
<i>SettingsBaseNode</i>	Base class for various settings
<i>UserAttributes</i>	Class for user attributes
<i>FolderNode</i>	Class for folder
<i>LocalDataFolder</i>	Folder class for local data. Saved only to local storage
<i>SystemDataFolder</i>	Folder class for unshared data.
<i>LoadedNodes</i>	Class keeping list of loaded nodes
<i>ServerLoadedNodes</i>	Class keeping list of nodes loaded from server
<i>ReservedNode</i>	Class keeping information for a check out node
<i>UnloadedNode</i>	Class for node without data loaded from storage
<i>ModelBaseNode</i>	Base class for all model nodes
<i>Model3DNode</i>	Base class for all 3D model nodes
<i>Member3D</i>	Base class for 3D extruded nodes along length
<i>ConcreteMember3D</i>	Concrete member
<i>ConcreteColumn</i>	Concrete column
<i>ConcreteBeam</i>	Concrete beam
<i>SteelMember3D</i>	Steel member
<i>SteelColumn</i>	Steel column
<i>SteelBeam</i>	Steel beam
<i>TimberMember3D</i>	Timber member
<i>TimberColumn</i>	Timber column
<i>TimberBeam</i>	Timber beam
<i>Plate3D</i>	Base class for 3D plates, usually extruded along thickness
<i>Slab</i>	Base class for slabs, area nearly horizontal
<i>Wall</i>	Base class for walls, area nearly vertical
<i>Door</i>	Base class for doors
<i>Window</i>	Base class for windows
<i>Opening</i>	Base class for openings
<i>Truss</i>	Base class for trusses
<i>SpaceTruss</i>	Base class for space trusses
<i>PlanarTruss</i>	Base class for planar trusses, members in plane
<i>Connection3D</i>	Base class for connections between 3D model nodes
<i>GridMesh</i>	Base class regular reference grids
<i>Grid</i>	Reference grid with multiple levels
<i>GridUCS</i>	One level grid for user coordinate system (UCS)
<i>Mesh</i>	One level grid with constant spacings without labels
<i>ArbitraryMesh</i>	Base class arbitrary reference grids

<i>DXFMesh</i>	Reference mesh for DXF
<i>ReferencePlanes</i>	Base class reference planes
<i>SinglePlane</i>	Single reference plane
<i>RectanglePlane</i>	Rectangle reference plane
<i>PolylinePlane</i>	Polyline reference plane
<i>SymmetricRoof</i>	Symmetric roof reference planes
<i>ReferencePlanesCut</i>	Base class cuts between reference planes
<i>ReferencePlanesOneSideCut</i>	Cut between two reference planes
<i>ReferencePlanesPolygonCut</i>	Polygon cut for a reference plane
<i>ModelGeometry3D</i>	Base class for solid geometry as triangle mesh
<i>RectangularCuboid</i>	Parametric solid with three dimensions
<i>RectangularCuboidDouble</i>	Two equal rectangular cuboids
<i>Cylinder</i>	Parametric cylinder geometry
<i>ExtrudedSection</i>	Parametric extruded geometry
<i>ExtrudedSectionDouble</i>	Two equal extrusions
<i>ExtrudedSectionN</i>	Unlimited count of equal extrusions
<i>SolidMesh</i>	Non-parametric solid defined as triangles
<i>SolidMeshBundle</i>	Bundle of equal solid meshes
<i>ModelGeometryGroup</i>	Group of geometries
<i>EmptyGeometry</i>	Empty geometry for special cases
<i>Model2DNode</i>	Base class for all model nodes with 2D coordinate system. Requires Model3DNode as parent
<i>Member2D</i>	Base class for 2D extruded nodes
<i>ConcreteMember2D</i>	Concrete member
<i>SteelMember2D</i>	Steel member
<i>TimberMember2D</i>	Timber member
<i>Plate2D</i>	Base class for 2D plates
<i>Connection2D</i>	Base class for connections between 2D model nodes
<i>ModelDataOrFolderNode</i>	Base class for model data or folder nodes
<i>ModelFolderNode</i>	Base class for model folder nodes
<i>PartNumbering</i>	Class for part numbering data
<i>PartNumber</i>	Class for part number data
<i>PartNumberingGroupNode</i>	Group node for part numbering data
<i>PropertySums</i>	Property sums for part numbering
<i>SinglePropertySum</i>	Single property sum for part numbering
<i>GridLevelZ</i>	Level for grid
<i>GridLine2D</i>	Line on grid level
<i>MeshLine3D</i>	Line for arbitrary mesh
<i>DXFLine3D</i>	Line for DXF reference
<i>MeshTriangle</i>	Mesh triangle for solid geometry
<i>ReferencesGroupNode</i>	Group node for reference planes
<i>ReferencePlanesCutGroupNode</i>	Group node for reference plane cuts
<i>DocumentsBaseNode</i>	Base for document nodes for drawings and reports
<i>DrawingBaseNode</i>	Base class for drawing
<i>Drawing</i>	Class for drawing
<i>DrawingPage</i>	Class for drawing page
<i>DrawingExtensionPage</i>	Class for extension drawing page

<i>DrawingsGroupNode</i>	Group node for drawings
<i>DrawingFont</i>	Data class for font on drawing
<i>DrawingBaseBlock</i>	Base class for drawing block
<i>AutoBlockParts</i>	Group node for automatically generated block content
<i>ManualBlockParts</i>	Group node for manually made block content
<i>ManualLineParts</i>	Group node for manual lines of block
<i>LabelsGroupNode</i>	Group node for labels
<i>DrawingLabel</i>	Label on drawing
<i>SlopeMark</i>	Slope mark
<i>BucklingMark</i>	Buckling mark
<i>CompressBucklingMark</i>	Special buckling mark for compress
<i>ManualLine</i>	Manual line on drawing
<i>AutoLabelSettings</i>	Settings for automatic labels
<i>AutoDimensionLineSettings</i>	Settings for automatic dimension lines
<i>DimensionLinesGroupNode</i>	Group node for dimension lines
<i>DimensionLineGraphics</i>	Base class for detail graphical elements
<i>HalfCircleGraphicComponent</i>	Label frame with round ends
<i>DimensionStickLine</i>	Stick line of dimension line
<i>DimensionCrossStickLine</i>	Cross stick line on dimension line
<i>DimensionLineArrow</i>	Arrow on dimension line
<i>DimensionLineCircle</i>	Circle on dimension line
<i>DimensionLineImage</i>	Image on dimension line
<i>DimensionLineLine</i>	Line of dimension line
<i>DimensionLineText</i>	Text on dimension line
<i>DimensionLineBase</i>	Base class for dimension lines
<i>DimensionLine</i>	Class for dimension line
<i>DimensionLineTemplate</i>	Class for dimension line template
<i>DimensionSnapPoint</i>	Class for dimension snap point
<i>PluginNode</i>	Base class for plugins

Classes for data saving and sharing

<i>DataStorageFunctions</i>	Tool class for local storage operations
<i>LoadSave</i>	Tool class for saving and loading partial models
<i>Events</i>	Tool class for handling data model events
<i>Reservation</i>	Tool class for handling check out and in actions
<i>UserLoginProject</i>	Tool class for server connections

Classes for geometry handling

<i>GeometryMath</i>	Collection of functions related to geometry
<i>LineContainer</i>	Class for 2D line
<i>Line3DContainer</i>	Class for 3D line
<i>Matrix2DTool</i>	Tool class for 2D matrix generation
<i>Matrix3DTool</i>	Tool class for 3D matrix generation

<i>PolygonTools</i>	Tool class for polygon geometry
<i>Surface3D</i>	Class for surface
<i>Surface3Dhole</i>	Class for hole on surface
<i>SectionShapes</i>	Tool class for handling section shapes
<i>SectionShape</i>	Base class for geometrical section shapes
<i>Rectangle</i>	Rectangle section
<i>IConstant</i>	I-shaped section
<i>MaterialSections</i>	Tool class for section handling
<i>MaterialSection</i>	Base class for material based section shapes
<i>ConcreteSection</i>	Base class for concrete section shapes
<i>SteelSection</i>	Base class for steel section shapes
<i>HotRolledI</i>	Base class for hot rolled I section shapes
<i>HEABase</i>	Class for HEA section shapes
<i>TimberSection</i>	Base class for timber section shapes
<i>SectionPolygon</i>	Class for section shape polygon
<i>PolygonVertex</i>	Vertex on shape polygon

Classes for generation of automatic dimension lines

<i>DimensionLineEngine</i>	Tool class for generation of automatic dimension lines
<i>BlockElementReservedArea</i>	Area reservation for drawing graphics
<i>DimensionLineBinding</i>	Base class for dimension line bindings
<i>DimensionLineBindingLine</i>	Dimension line binding on line
<i>DimensionLineBindingPoint</i>	Dimension line binding on point
<i>DimensionLineSnapPoints</i>	Container class for dimension line snap points

Other classes

<i>NumberingTool</i>	Tool class for part number generation
<i>CopyMoveTool</i>	Tool class for copies and moves of nodes and trees
<i>ModelTools</i>	Tool class for 3D Boolean operations between solid geometries and for generating hidden lines

- 698. Mekonnen, Tenager (2019) Efficient resource management in Multimedia Internet of Things
- 699. Liu, Xin (2019) Human motion detection and gesture recognition using computer vision methods
- 700. Varghese, Jobin (2019) MoO_3 , PZ29 and TiO_2 based ultra-low fabrication temperature glass-ceramics for future microelectronic devices
- 701. Koivupalo, Maarit (2019) Health and safety management in a global steel company and in shared workplaces : case description and development needs
- 702. Ojala, Jonna (2019) Functionalized cellulose nanoparticles in the stabilization of oil-in-water emulsions : bio-based approach to chemical oil spill response
- 703. Vu, Kien (2019) Integrated access-backhaul for 5G wireless networks
- 704. Miettinen, Jyrki & Visuri, Ville-Valtteri & Fabritius, Timo (2019) Thermodynamic description of the Fe–Al–Mn–Si–C system for modelling solidification of steels
- 705. Karvinen, Tuulikki (2019) Ultra high consistency forming
- 706. Nguyen, Kien-Giang (2019) Energy-efficient transmission strategies for multiantenna systems
- 707. Visuri, Aku (2019) Wear-IT : implications of mobile & wearable technologies to human attention and interruptibility
- 708. Shahabuddin, Shahriar (2019) MIMO detection and precoding architectures
- 709. Lappi, Teemu (2019) Digitalizing Finland : governance of government ICT projects
- 710. Pitkänen, Olli (2019) On-device synthesis of customized carbon nanotube structures
- 711. Vielma, Tuomas (2019) Thermodynamic properties of concentrated zinc bearing solutions
- 712. Ramasetti, Eshwar Kumar (2019) Modelling of open-eye formation and mixing phenomena in a gas-stirred ladle for different operating parameters
- 713. Javaheri, Vahid (2019) Design, thermomechanical processing and induction hardening of a new medium-carbon steel microalloyed with niobium
- 714. Hautala, Ilkka (2019) From dataflow models to energy efficient application specific processors

UNIVERSITY OF OULU P.O. Box 8000 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

S E R I E S E D I T O R S

A SCIENTIAE RERUM NATURALIUM

University Lecturer Tuomo Glumoff

B HUMANIORA

University Lecturer Santeri Palviainen

C TECHNICA

Senior research fellow Jari Juuti

D MEDICA

Professor Olli Vuolteenaho

E SCIENTIAE RERUM SOCIALIUM

University Lecturer Veli-Matti Ulvinen

F SCRIPTA ACADEMICA

Planning Director Pertti Tikkanen

G OECONOMICA

Professor Jari Juga

H ARCHITECTONICA

University Lecturer Anu Soikkeli

EDITOR IN CHIEF

Professor Olli Vuolteenaho

PUBLICATIONS EDITOR

Publications Editor Kirsti Nurkkala

UNIVERSITY of OULU
OULUN YLIOPISTO

ISBN 978-952-62-2369-8 (Paperback)

ISBN 978-952-62-2370-4 (PDF)

ISSN 0355-3213 (Print)

ISSN 1796-2226 (Online)