IFChub: Multimodal Collaboration Platform with Solid Transactional Guarantees

Vitaly Semenov, sem@ispras.ru
Institute for System Programming RAS, Russian Federation

Stephan Jones, stephan@ifchub.com
IFChub Ltd., United Kingdom

Abstract
This paper presents an innovative software platform IFChub intended to build advanced collaborative environments and to conduct multidisciplinary projects in architecture, engineering, and construction. Partial attention in the paper is paid to principal features of the platform and its key distinctions from existing BIM servers. The platform enables IFC-driven building information to be shared concurrently among project stakeholders under different collaboration modes and to be managed strongly in correspondence with ACID or BASE transaction principles. In this paper we argue the importance of underlying BASE principles for BIM servers and explain how they can be satisfied within the IFChub platform using optimistic replication. Supporting both standard file exchange and system integration via unified software interfaces, the platform assumes a relatively easy migration path from existing single-site applications to advanced multi-modal collaborative environments. The presented IFChub platform v1.0 has been released and is freely available for validation and integration purposes.

Keywords: BIM servers, IFC, DBMS, ACID, BASE

1 Introduction

Recently, commercial and free model servers are becoming available and start to gain popularity to conduct complex multidisciplinary projects in industrial sectors such as shipbuilding, automotive, petrochemical, pharmaceutical, and construction. The key factors motivating the deployment of model servers and the building collaborative environments for such purposes are as follows:

— participation of small, medium and large enterprises having heterogeneous information infrastructures corresponding to their specific business profiles and practices

— involvement of a large number of individuals and teams with own roles, responsibilities, expertise, skills and tools to solve particular problems

— high concurrency of design and engineering processes

— complexity of the information circulated among project stakeholders

— large volumes of data to be managed and processed effectively.

International industrial and academic communities made a significant contribution to the standardization of information models and software interfaces in the scope of STEP, IFC, CIS/2, POSC/CAESAR initiatives and corresponding ISO technical committee activities (ISO 10303, 1994). However, these have not been comprehensive from a software engineering perspective, the mentioned standards admit a variety of implementations based on different principles, architectures, middleware, and datastores. Particularly, the standards do not regulate the issues principal for the creation and the use of model servers and, particularly, BIM servers as database management systems (Semenov et al 2004).

BIM servers are special database systems that underpin building information sharing between multiple users. Most servers are based on propriety data models, which simplifies system integration within the same product line but prevents interoperability with/between third-party software systems and multi-disciplinary collaboration.

Some companies have declared IFC-compliant collaboration platforms. Particularly, these are A360 Collaboration for Revit by Autodesk (A360, 2015) and BIM+ by Nemetschek (BIM+, 2015), BIMsight by Tekla (BIMsight, 2015). However, these platforms have limited capabilities for updating and sharing IFC models among multiple users and cannot therefore be regarded as IFC model servers. Current features including visualization and analysis are clearly insufficient for these purposes.

IFC model servers are database systems that represent building models in strong correspondence with the IFC standard and provide query evaluation and data manipulation facilities to manage multi-user collaboration sessions. Reference examples include EDM Model Server by Jotne IT, Eurostep Model Server, IFC Model Server by VTT Building and Transport and SECOM Co., and Open Source BIM Server. For brevity we omit here features and implementation details of these systems, addressing the readers to analytical surveys (Jorgensen et al 2008; Shafiq et al 2012).

2 ACID versus BASE

In computer science, ACID (Atomicity, Consistency, Isolation, and Durability) is a set of principles that guarantee database transactions are processed correctly. Although the abbreviation was first introduced in 1983 (Haerder & Reuter 1983), recently the ACID principles are commonly recommended for any traditional DBMS.

Atomicity principle implies that either all the operations in a transaction must happen, or none of them. The transaction must be completed if successful or rolled back if it fails. This principle works well for short transactions but it is unlikely to be productive for long transactions peculiar to typical collaborative sessions and workgroup design and engineering activities. The obtained results of individual works should be consolidated as much as possible even if some particular operations are unsuccessful. Decomposition of transactions into independent parts might be one of the promising approaches to meaningful relaxation of this principle.

Consistency principle assumes that any transaction brings the data from one valid state to another. This term is broad in scope and may have different meanings depending on the specific context. Usually it means both logical coherence and correctness of data. Valid data must avoid contradictory states and must satisfy the imposed integrity rules. In fact, the data coherence is a property that allows the attenuation caused by pending updates of partitioned or replicated data. The data correctness is a critically important requirement as any violated integrity rule may result in the data corruption and uselessness for further transactions.

Isolation principle avoids any interference of simultaneous transactions with each other. Intermediate results within a transaction must remain invisible to other transactions. Often this property is defined as a serializability implying that the concurrent execution of transactions results in a data state that would be obtained if transactions were executed serially, one after the other. Insufficient isolation may result in well-known anomalous phenomena such as dirty writes, dirty reads, non-repeatable reads, lost update, phantoms, skew reads, and skew writes (Berenson et al 1995). This principle seems to be a strong requirement for collaborative environments as results of individual sessions should be deterministic and predictable even if collective activities run in parallel.

Durability principle assumes that results of completed transactions cannot be discarded. They must persist even in the event of power loss, system crashes or errors. To defend against these events, the transaction results need to be stored permanently in a non-volatile memory, e.g. hard disks.

The IFC model servers, being special DBMS, should follow the ACID properties too. However, of the four ACID properties, consistency and isolation principles are often neglected or relaxed to a significant extent. But these are crucial for the practical success of BIM servers operating with complex IFC models and providing AEC clients with effective multi-access to the shared models.

As mentioned above, the database consistency implies, in particular, that all the imposed integrity rules must be satisfied. Notice that the IFC schema is quite complex from a semantic point of view. Table 1 demonstrates significant evolution and complexity growth of the IFC schema since 1997 when its first version was adopted, to 2013 when the recent IFC 4 version shipped. Over those years the total number of data types, procedures, functions and rules has been significantly increased.

Table 1 Some statistics for the evolved standard IFC schema

IFC SCHEMAS (ADOPTION YEAR)	IFC 1.5.1 (1997)	IFC 2.0 (1999)	IFC 2.x (2000)	IFC2x2 (2003)	IFC2x3 (2006)	IFC 4 (2013)
ENTITIES	186	290	370	623	653	766
USER-DEFINED TYPES	95	157	228	312	327	391
DERIVED ATTRIBUTES	44	45	41	55	96	105
PROCEDURES FUNCTIONS	25	27	25	37	38	42
ENTITY WHERE RULES	107	168	196	271	340	638
TYPE WHERE RULES	12	12	13	16	24	24
UNIQUE RULES	1	3	14	14	17	4
GLOBAL RULES	0	0	3	3	2	2

Several ambiguous operations or a few uncoordinated transactions can violate integrity rules and destroy the consistency of the whole IFC model. In such cases the model becomes incorrect and useless for subsequent processing. Maintaining the IFC model in a consistent state is a serious problem that must be resolved by both BIM server and client application sides. The responsibility of application-side code is to guarantee the correctness of isolated transactions in terms of maintained data consistency. It is quite difficult for the application programmer to write robust code that would take into account all combinations of operations and be able to analyse their potential affect on the validity of the IFC rules. Therefore, the server-side code must incorporate calls to check the committed transactions and to confirm that the target IFC model undergoes consistent revisions allowing subsequent processing sessions.

Unfortunately, none of the mentioned BIM servers can control that every committed transaction left the IFC model in a consistent state, none of these solutions can guarantee that the conflicts between concurrent transactions, if they occurred, resolved in a consistent manner. Thereby, the consistency property is neglected in recent BIM server implementations.

It is worth mentioning well-known software tools to validate IFC files against rules specified by the IFC schema (IFC Toolboxes, 2015). Basically, the available validation tools are EDMmodelChecker, Express Engine, IfcObjectCounter, STEP Conformance Checker, and SemanticSTEP Checker. Invoking these for each committed transaction looks unrealistic due to significant computational costs exceeding the processing of transactions. Being invoked periodically the tools are also ineffective because they they fail to identify transactions and client applications that have disturbed the data consistency. Therefore, obvious requirement is to roll back all subsequent transactions as last revisions cannot be trusted.

Insufficient isolation is another problem preventing the reliable use of BIM servers for collaboration purposes. Interference of transactions and loss of ability to serialize, results in the uncertainty of collective sessions and data consistency violations even if the clients run correctly when being isolated. The concept of the shared BIM assumes that its parts can be accessed concurrently. To coordinate concurrent access the data items are usually locked within so-called pessimistic transaction models. Serializable transactions acquire share locks on the data items they read and exclusive locks on the items they write. These locks are held until the transaction commits or rolls back. Pessimistic transaction models help to avoid some anomalies, but degrade the concurrency due to increased data contention.

BIM data items can be potentially locked at different granularity and isolation levels. We distinguish three levels for IFC models being represented within an object-oriented paradigm: at an entire model, at level of objects, and at level of object attributes. Isolation level and concurrency are inversely related. A lower isolation level enables greater concurrency, but with greater risk of data inconsistencies. A higher isolation level provides a higher degree of data consistency, but at the expense of concurrency.

Shared access to entire IFC model in read mode and exclusive access to entire IFC model in write mode correspond to high isolation level. After a transaction has obtained an exclusive model-level lock, the data becomes entirely unavailable for all other transactions which have to be blocked until the transaction commits or rolls back. A transaction that needs data access and requires an exclusive lock cannot start until there are no active transactions on the model. High isolation prevents all sorts of anomalies and guarantees the IFC model remains consistent, certainly, on the condition that individual clients, being IFC-compliant applications update it concordantly. This transaction model is well suited to downstream design and engineering workflows allowing serial execution of works. However, it is quite onerous for concurrent activities performed within large work packages by teams under strict time deadlines. As typical transactions in CAD/CAE applications are quite long, such isolation may result in the unproductive collaboration sessions.

The object-level and attribute-level locking is more promising for concurrent design and engineering sessions. But they cannot preserve semantically complex data on violations. Indeed, according to the Table 1, the IFC schema in EXPRESS language predefines so-called WHERE rules interrelating attributes of separate objects as well as UNIQUE and GLOBAL rules establishing relationships on object type extents. To support the ability to serialise transactions and to maintain the IFC model in a consistent state, the clients must preliminary lock those objects and/or attributes which are subjected to reads or updates. Obviously, that attribute-level locking would be not sufficient to reliably keep the data in states concordant with the WHERE rule. More detailed analysis shows that object-level locking is also not safe from the consistency violations and entire object type extents should be locked to eliminate risks of disturbance of the UNIQUE and GLOBAL rules. But this way brings us closer to model-level isolation with severe restrictions on the data availability and the transaction concurrency.

Thus, reasonable trade-offs between consistency and concurrency is hardly achieved when enforcing ACID properties. This observation is a consequence of a more general "CAP theorem" (Brewer 2010). In practice, many DBMS allow selective relaxation of these properties for better performance and scalability. For example, NoSQL systems such as Amazon's Dynamo, Google's BigTable, HBase, Hypertable, Cassandra, SimpleDB generally do not provide ACID transactional properties, but follow to BASE (Basically Available, Soft state, Eventually consistent) principles (Cattell 2010). As opposed to *immediate* consistency, *eventual* consistency here means that fetched data is not guaranteed to be up-to-date, but updates are guaranteed to be propagated to all replicas eventually. This allows them to replicate and partition data over many servers and support extremely high loads of simple operations. With the advent of the web sites where thousands and millions of users may both read and write electronic mail, personal profiles, web postings, wikis, and many other kinds of data, scalability and performance have become a more important requirement than permanent consistency.

BASE principles are widely deployed in distributed DBMS, often under the moniker of optimistic replication. In contrast to pessimistic transactions restricting concurrent access to shared data, optimistic replication avoids blocking data items and provides immediate access to data entirely without any preliminary locking. As opposed to pessimistic replication enforcing replicas to be identical, there is no need to wait for all of the diverged replicas to be synchronized when updating data. But the advantages in concurrency are realized due to weakening permanent consistency. Moreover, different diverged replicas must be periodically converged using syntactic or semantic reconciliation, which might then prove difficult or even insoluble. By this reason, BASE is sometimes criticized as increasing the complexity of distributed software applications (Gilbert & Lynch 2002). However, sacrificing ACID in favor of BASE, optimistic replication technologies are widely used in such domains as collaborative environments, software configuration solutions, mobile databases, distributed services (Saito & Shapiro 2005). These observations inspired us to develop IFChub platform mainly following to BASE assumptions, but maintaining ACID properties when it is possible.

3 IFChub collaboration platform

IFChub is an acronym for **IFC** Compliant, **H**orizontally Scalable, **U**niform Accessible, ACID/**B**ASE Solid collaboration platform for architecture, engineering and construction. The platform enables IFC-driven building information to be shared concurrently among project stakeholders under different collaboration modes and to be managed with solid ACID/BASE guarantees (IFChub, 2015).

The platform comprises an IFChub model server running under Tomcat application server, general-purpose PostgreSQL DBMS used mainly for administration purposes as well as STEP metadata services intended to manage IFC-driven data and to solve related computational problems.

Figure 1 presents a typical configuration of the deployed collaboration environment. With this configuration IFChub model server, DBMS and computational services run on different computers providing the best scalability and performance. Other configurations, for example, hosting IFChub server, DBMS and single STEP service on the same computer, are also admitted. To administrate the collaboration environment and to perform some end-user activities interactively, IFChub Web client to the server is provided too.

Within IFChub platform BIM data are interpreted and processed in strong correspondence with the standard IFC2x2, IFC2x3, and IFC4 schemas formally specified in EXPRESS data modelling language. Complete sets of data types, consistency rules, derived and inverse attributes are supported without any limitations and misinterpretations of the schema semantics. It is achieved due to deployment and utilization of the metadata dictionaries semantically equivalent to the original IFC specifications.

IFChub platform supports and leverages both vertical and horizontal scaling. The platform capacity can be increased (vertical scaled) by adding more resources, e.g. updating multi-core CPU, extending RAM, adding extra or larger hard drives to the existing computer network. It tends to provide higher availability of data and improves performance. However, scaling up usually requires downtime while new resources are being added and has limits induced by known technical and economic reasons. It’s generally better to spread the cost of infrastructure across many inexpensive and relatively disposable nodes than concentrate it in a few very high value components. Single or few points of failure are best avoided.

An important feature is that IFChub model server can be scaled out horizontally. It is accomplished by adding more commodity hardware nodes to the IFChub cluster and running more STEP services there. They can be spread across heterogeneous infrastructures of the organisations involved in a multidisciplinary project, they can be also located in datacenters if the organisations have no proper infrastructure of their own. Running on top of a computer cluster the IFChub model server balances load of STEP services taking into account available resources of each individual node and computational tasks to be carried out. The architecture of IFChub platform and, particularly, supported automatic registration of services, provides administrators with the ability to increase capacity of the platform as a single logical unit on the fly without disrupting workability of the deployed collaboration environment.

IFChub provides a straightforward JSON-based API through HTTP and protocol buffers for basic (PUT, GET, POST, and DELETE) functions. Uniform IFChub API separates clients from the model server so that clients are not concerned with data storage and processing, and the server is not concerned with the user interfaces and client functionalities. IFChub API is uniform in the sense that it allows clients to access databases, projects, revisions, deltas, objects, attributes, elements in a uniform way independently of a particular IFC schema. Noteworthy, deltas between the nearest revisions are also represented as collections of persistent objects enabling clients to compare different revisions and to identify what changes have occurred since the previous session. Deltas are also used to submit transactions entirely using their representations encoded in JSON and SPFF file formats.

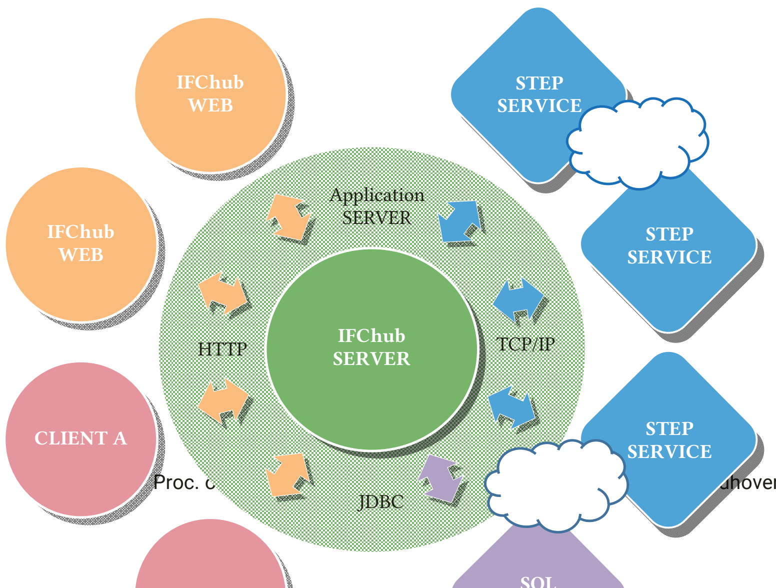

Figure 1 Typical configuration of collaboration environment based on the IFChub platform

IFChub API is organised as function sets basically intended to query IFC data, to manipulate data and to analyse data. To request data beloning to a particular IFC model revision, an object-oriented query language is used with the features described below. Special functions are provided to manipulate data at different aggregation levels corresponding to separate projects, revisions, objects, attributes, and collection elements. Manipulation operations can be transferred and interpreted sequentially one after another or within a single operation bucket providing higher performance. Finally, statistical data on a particular revision, summary results of the comparison of different revisions, or status of the revision compliance with an IFC schema can be obtained by invoking analysis functions. Such results are encoded as HTML files so that the prepared pages can be shown immediately using popular web browsers.

The IFChub query language combines capabilities to define both object predicates and navigation rules. The object predicates are specified as logic expressions in full correspondence with the EXPRESS language and the underlying IFC schema. So it looks natural the IFChub employs the same language to specify both data schema and queries to its instantiated data units. Navigation rules help to define spread queries aimed at extraction of heterogeneous collections of objects accessible though propagation along associations from one object to another. In combination spread queries and object predicates enable to specify complex queries, particularly, model view definitions (MVD) adopted by IFC community (MVD, 2015). IFChub queries may be specified and applied as ad-hoc requests and stored procedures. In both cases they are represented as objects and are encoded in JSON or SPFF formats.

4 Multirevision management

IFChub server processes client transactions which may be either read or write. A new IFC model revision appears whenever the client commits a write transaction or checks in a new IFC file. Read-only transactions do not produce revisions. Existing revisions can never be changed. Therefore, the full transaction history is represented as a model revision tree with the root corresponding to the initial data and leaves — to its terminal revisions.

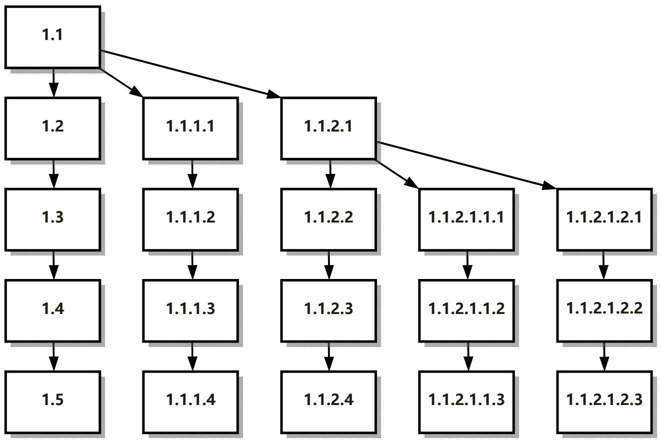

Figure 2 IFC model revision tree and indexing policy

The root revision is assigned the number 1.1. The next revision takes the number 1.2, the next succeeding — 1.3 and so on. Thus formed branch (with two digits in the number) is called main branch (see Figure 2 to understand the applied revision indexing policy). New minor branches (with four or more digits in the number) are created by committing write transactions on non-terminal revisions in the main or other branches. New branches can be created *intentionally* to maintain alternative realizations of the same AEC project. In such cases the branched revision can serve as a root IFC model for a new project alternative to the original one. New branches can be created *occasionally* as a result of committing concurrent transactions proceeding on the same revision. Collaboration environments must provide opportunities to both individuals and groups to work in parallel with a recent project revision. If all the transactions are successful, the first committed transaction produces a new revision on the original revision branch, and other committed transactions produce revisions on minor branches.

To reduce the entropy of occasionally produced revisions and to consolidate results of individual efforts and groupworks, the revisions should be merged. For this purpose, so-called three-way merging and delta calculus are applied. A common ancestor is determined and deltas for the merged revisions are computed beginning from the ancestor revision. Then, deltas are analysed against potential conflicts and if no conflicts are detected the deltas are consolidated into resulting one, which being applied to the ancestor produces a new final revision. Merging can be done in manual or automatic mode. The automatic mode enables this process to be hidden from software developers and end-users entirely and removes the transaction concurrency analysis from their responsibility.

Delta calculus is applied in the IFChub platform by another method. Storing all model revisions is ineffective from the standpoint of hard disk space consumption. The IFChub server manages changes and stores only partial deltas between nearest revisions instead of full representations. Taking into account large size of IFC models for typical AEC projects and relatively low percent of changes between sequential revisions, the total space saving can be significant.

Every partial delta is represented as a list of elementary changes induced by operations of a submitted write transaction or recognized by differencing of checked-in IFC files. Check-in operation behaves as a transaction with one difference that partial delta with respect to a previous model revision should be additionally computed. Because not all the IFC objects have their own GUID, delta computations are based on inexact matching of object type extents in recent and previous model revisions. Once all subsequent partial deltas are persistently stored, most saving policy would consist in preserving the root revision. Then any model revision as well as delta between any two revisions can be recovered when it becomes necessary.

Indeed, having an original revision R_i and a delta D_i^j, the subsequent revision R_j can be

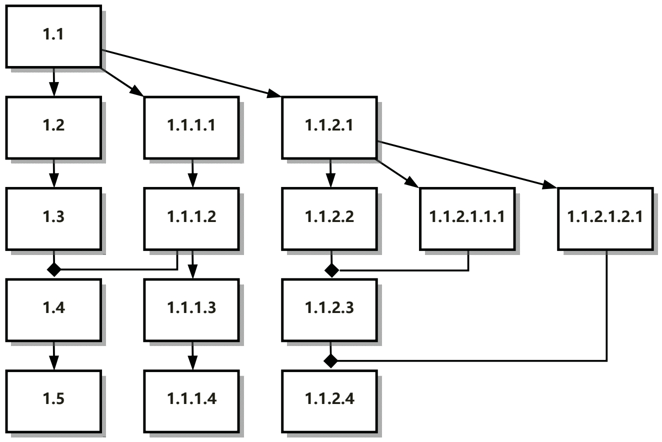

Figure 3 Merging IFC model revisions

obtained by applying delta changes to the original revision. Using algebraic operations it can be formally written as $R_j = R_i + D_i^j$. For the presented example, the revision $R_{1.4}$ can be obtained as $R_{1.4} = R_{1.1} + D_{1.1}^{1.2} + D_{1.2}^{1.3} + D_{1.3}^{1.4}$. The deltas are bi-directional ones meaning that backward process is also supported and there is a capability to roll back the recent revision to the original one using an inversion operation $D_i^j = -D_j^i$. Then, having a recent revision R_j and a delta D_i^j, the original revision R_i can be always recovered as $R_i = R_j - D_i^j$. Obviously, that the addition and subtraction operations are not commutative here. For deltas incident to a common revision, the following identities hold: $D_i^j = D_i^k + D_k^j$ and $D_i^j = D_i^k - D_j^k$. For the given example, if changes between the revisions $R_{1.4}$ and $R_{1.1.1.3}$ need to be obtained, it can be done using partial delta calculus as follows: $R_{1.4} - R_{1.1.1.3} = -D_{1.1.1.2}^{1.1.1.3} - D_{1.1.1.1}^{1.1.1.2} - D_{1.1.1}^{1.1.1.1} + D_{1.3}^{1.4} + D_{1.2}^{1.3} + D_{1.1}^{1.2}$. Finally, we define three-way merging operation $R(R_k, R_i, R_j) = R_k + (R_i - R_k) \oplus (R_j - R_k)$, where the revision R_k is a common ancestor of the revisions R_i, R_j to be merged and $\oplus$ is a delta reconciliation operation. It can be rewritten in terms of deltas as $R(R_k, D_i^k, D_j^k) = R_k + D_i^k \oplus D_j^k$. The reconciliation operation acts as a theoretically-set union of deltas if the deltas do not conflict, and as an empty set if some conflicts are detected. In the example presented, revision $R_{1.4}$ has been obtained by merging revisions $R_{1.3}$ and $R_{1.1.1.2}$ with common ancestor $R_{1.1}$.

5 Multimodal collaboration with solid transactional guarantees

IFChub model server provides both pessimistic and optimistic transactions. In combination they support multimodal collaboration sessions corresponding to varied levels of concurrency and consistency trade-offs. It is noteworthy that every supported transaction model strongly obeys either ACID or BASE principles. Ensuring these principles can be holistically satisfied using multi-version and change management facilities discussed in previous Section 4.

Because every successful transaction leads to a new revision, and unsuccessful — to roll back to a previous one, there is no problem to guarantee atomicity and durability. Any revision can be recovered by applying direct deltas to its persistent ancestor, for example, to the initial revision $R_{1.1}$ which is always stored on disk. Any revision can be obtained by subtracting deltas from its persistent successor, for example, from terminal revisions which are usually stored on disk in expectation of session recovery.

The isolation of pessimistic transactions and their serializability are provided by locking model revisions entirely. Locks at other granularity levels, i.e. at object and attribute levels, are not supported for the reasons explained in Section 2. Any model revision can be locked exclusively for a single write transaction. Locking a revision entirely provides full isolation, avoids any anomalies and eliminates potential semantic violations that could happen because of concurrent writes. There are no obstacles for multiple read transactions even if the revision has been preliminary locked by an exclusive write transaction. When it is necessary to avoid dirty read anomalies, a reading transaction should lock the revision as it would intend to write. Certainly, it is possible only if the revision has not been previously locked by another transaction.

The isolation of optimistic transactions is naturally supported due to the fact that each transaction proceeds with its own revision replica and thus does not affect the other concurrent transactions. When it is necessary the results of individual transactions can be merged. Risks of detecting conflicts and rolling back one of the concurrent transactions remain however it can often be avoided with proper policy of work distribution among collaborators.

The consistency and the integrity in particular are not trivial requirements especially in the case of semantically complex IFC models. The requirements can be also successfully satisfied within adopted pessimistic and optimistic transactions. The IFChub server provides capabilities needed to check any model revision against IFC schema rules. Moreover, these capabilities are effective enough to control not only separate model revisions, but also every committed transaction and thereby to guarantee that whole IFChub model DB is maintained in consistent way. These capabilities are achieved through incremental checks of committed transactions which can be carried out much more quickly that a full validation of the entire model. Indeed, short transactions usually contain a relatively small number of operations which have local impact on related data.

Therefore, typically there is no need to suffer the entire model revision to all checks. Only those data which relate to committed transaction operations and may violate some semantic rules should be subject to validation.

On the condition that the initial revision has been successfully validated and all the committed transactions have been incrementally checked, it can be argued that every persistent revision is consistent. Similar statement is true for revisions obtained by means of merging persistent revisions. On the condition that common ancestor and two merged revisions are consistent, it can be guaranteed that the resulting revision is also consistent. Thus, the entire IFChub model DB can be maintained semantically consistent if the proper validation facilities are applied permanently.

6 Conclusions

Thus innovative software platform IFChub has been presented in the paper. The platform enables IFC-driven building information to be shared concurrently among project stakeholders under different collaboration modes and to be managed strongly in correspondence with the ACID or the BASE transactional principles. Support of alternative transaction models enables to hold collaboration sessions with varied levels of the data availability and the transaction concurrency. It has been argued that in any case the consistency requirement can be successfully satisfied.

Supporting both standard file exchange and system integration via unified software interfaces, the platform assumes a relatively easy migration path from existing single-site applications to advanced multi-modal collaborative environments. The presented IFChub platform v1.0 has been released and is freely available for validation and integration purposes.

References

Abadi, D., Thomson, A. (2010) The problems with ACID, and how to fix them without NoSQL. *DBMS Musings*, August 31, 2010. http://dbmsmusings.blogspot.ru/2010/08/problems-with-acid-and-how-to-fix-them.html

A360 (2015). “Autodesk A360 Collaboration for Revit”. http://www.autodesk.com/products/collaboration-for-revit

Berenson, H., Bernstein, P., Gray, J., Melton, J., O’Neil, E. & O'Neil, P. A Critique of ANSI SQL Isolation Levels, *SIGMOD Record Conference*, San Jose, CA USA, 1995, Vol. 24, No. 2, pp. 1-10.

Brewer, E. (2010) Towards Robust Distributed Systems, *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, July 2000, p. 7.

BIM+ (2015). “Make BIM happen”. https://www.bimplus.net

BIMsight (2015). “Improve the way you work today with Tekla BIMsight — the easiest to use construction collaboration software”. http://www.teklabimsight.com

Cattell, R. (2010) “Scalable SQL and NoSQL data stores”. *In Newsletter ACM SIGMOD*, 39, 4, pp. 12-27, December 2010.Haerder, T. & Reuter, A. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, Volume 15, Issue 4, December 1983, pp. 287 - 317.

Gilbert, S. & Lynch, N. (2002) “Brewer’s conjecture and the feasibility of consistent, available, and partition-tolerant web services”, *ACM SIGACT News 33*, 2, pp 51-59, March 2002.

IFChub (2015). “Work smart.. collaborate”. http://www/ifchub.com

IFC Toolboxes (2015) “buildingSMART. Implementation. IFC Toolboxes”, http://www.buildingsmart-tech.org/implementation/get-started/ifc-toolboxes

ISO 10303: 1994, Industrial automation systems and integration — Product data representation and exchange.

Jørgensen, K., Skauge J., Christiansson, P., Svidt K., Sørensen K.B., Mitchell, J. (2008) “Use of IFC Model Servers. Modelling Collaboration Possibilities in Practice”. *Technical report by Aalborg University and Aarhus School of Architecture. http://it.civil.aau.dk/it/reports/2008_Ifc_model_server.pdf*

MVD (2015) “buildingSMART. Specifications. Model View Definition Summary”, http://www.buildingsmart-tech.org/specifications/ifc-view-definition

Saito, Y. & Shapiro, M. (2005) “Optimistic Replication”. ACM Computing Surveys, 37(1):42-81, 2005.

Semenov, V., Bazhan A. & Morozov S. (2004) Distributed STEP-compliant platform for multi-modal collaboration in architecture, engineering and construction. *Proc. of X CIB W78 Conference*. Weimar, Germany, June 02–04, 2004, pp. 318.

Shafiq, M.T., Matthews, J. & Lockley, S. (2012) Requirements for model server enabled collaborating on building information models. *Proc. of First UK Academic Conference on BIM*, 5-7 September 2012, Northumbria University, Newcastle upon Tyne, UK.