A Model for a Distributed Building Information System

Tasos Varoudis[1], Panagiotis Patlakas[2]
[1]Bartlett School of Graduate Studies, University College London, UK
[2]Technology School, Southampton Solent University, UK
[1]http://www.bartlett.ucl.ac.uk [2]http://www.solent.ac.uk
[1]t.varoudis@ucl.ac.uk [2]panagiotis.patlakas@solent.ac.uk

This paper sets the theoretical and technological framework for the development of a distributed model for a holistic Building Information System. It commences by summarizing versioning and revision control models in software engineering. Then it proceeds to establish parallels between the distributed revision control process and the building design process and argues that the underlying structure of Building Information Modelling can be exploited for the development of a similar structure for building design. The fundamental framework of such a system, called Distributed Building Information System (DBIS), is described and implementation strategies are discussed, while the potential difficulties are also addressed.

Keywords: *BIM, Distributed Building Information System, Source Code Management*

INTRODUCTION

The internet explosion of the past 20 years has brought the advent of a wide range of technologies and, perhaps for the first time, achieved full integration of the digital in every realm of professional and personal life. In architecture, engineering, and construction (AEC) the three key developments were software advancements in the fields of visualization (Computer-Aided Design, 3d modelling, computational and parametric design etc.), engineering analysis and design (finite elements and computational fluid dynamics methods, structural design packages etc.), and management (project and program management tools). Building Information Modelling (BIM) aims to bring together these three aspects introducing the concept of a "master model" in which the aforementioned tools work in tandem feeding from, and back to, each other [1].

Despite the significant interest of the AEC industry in digital tools though, little attention has been paid to software developments that appear external to the sector. This is particularly pronounced for innovations that concentrate on "pure" or "core" software engineering topics, as they are considered not applicable to AEC. However this is not the case; as design assimilates the digital more and more, it also absorbs exactly those "core" computer science and software engineering aspects. Often these provide important opportunities for innovation. This paper attempts to highlight exactly such an opportunity.

AIMS AND SIGNIFICANCE

This paper aims to provide the theoretical and technological background for a new 3d model digital framework that allows a new way of collaboration in modelling, analysis, and design among the AEC disciplines. Based on the BIM concept, it extends and expands it to facilitate both intra-disciplinary and cross-disciplinary working. For this purpose, it utilizes the software engineering paradigm of Revision Control systems (RCS) and Source Code Management (SCM) systems, effectively viewing the geometry and associated information as a type of source code. A secondary aim is to introduce the fundamental concepts that will guide implementation and distribution via a cloud-based architecture.

The significance of this work is threefold. Firstly it illustrates how software engineering concepts typically considered as outside the domain of AEC can have direct beneficial impact; since it is highly likely that in the near future the digital will become more and more intertwined with the physical, this can be just the initial development of many.

Secondly the paper introduces the concept of the analogy between programming source code and the digital representation of a building, including both geometry and associated information. This is not meant to be a linking metaphor, as it is often employed in digital, computational, and parametric design; instead it assumes the two concepts are directly analogous, and sets up a clear blueprint for taking advantage of their similarities. This is not confined to the topic presented here, but can ignite a wider conversation on the importance and consequences of this relationship. Finally, by establishing a clear theoretical framework, it paves the way for implementation as discussed in the closing sections.

RCS AND BIM

An introduction to RCS and SCM

Managing complexity has been a problem of software engineering since its inception. The advent of the internet brought to programming the capability of organizing and coordinating projects remotely, as code could be sent around the globe practically instantaneously. At the same time, remote servers expanded the possibilities of the code repository. Thus versioning and revision control, a fundamental concept in programming, evolved significantly with the development of systems such as Apache Subversion (SVN) [2].

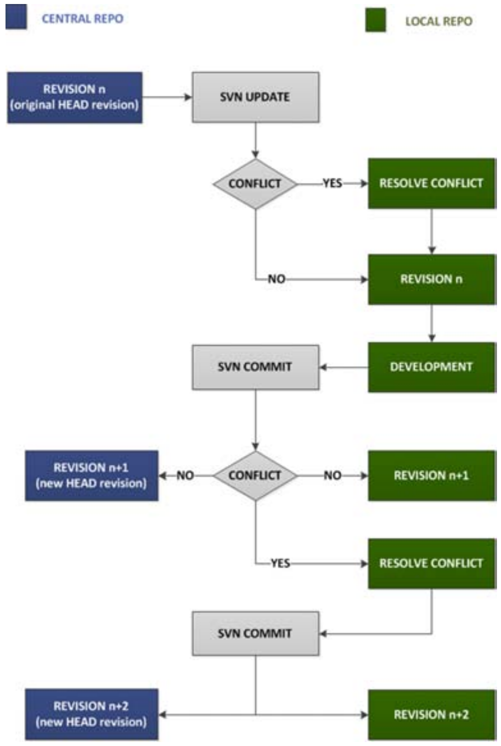


Figure 1
Typical SVN workflow

Conceptually SVN revolves around a central repository (repo), which represents the "correct" version of the source code. Prior to working an individual contributor must update her local copy to reflect the latest version of the central repository (the HEAD revision). Then the contributor can work on her code locally; when she is finished she has to commit her changes to the central repo. This creates a new HEAD revision (Figure 1).

While SVN was a significant advance over traditional versioning control, it had a number of disadvantages, most prominent of which were the difficulties of sorting conflicts (effectively discrepancies between different local copies) as well as resource-heavy "branching" (creating copies of the main source code for different purposes).

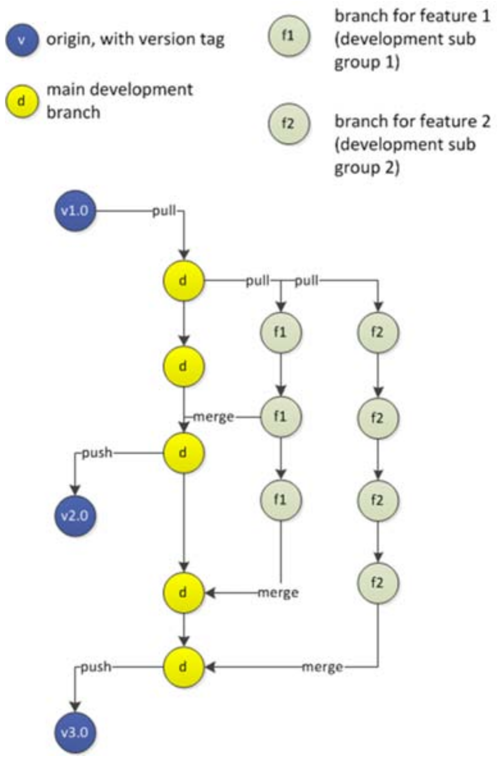


Figure 2
Git workflow

The next significant conceptual jump came from Linux creator Linus Torvalds [3]. Git introduced the concept source code manage-ment (SCM) via a distributed revision control system (DRVS). This eliminated the concept of the central ("true") repo and allowed the development of many parallel versions based on an original starting point [4]. Simultaneously, it introduced a number of significant performance and safety improvements that resulted in a much more agile system [5]. The DRVS concept proved particularly suited to the needs of the open source community, enabling the creation of initiative likes Github, but today it has been adapted by many commercial vendors such as Microsoft [6] and Google [7].

The possibilities Git opened led to many different ways of working, with various approaches considered during the first few years. Today however, good practice tends to be based on variations of "the successful Git branching model", initially reported by Vincent Driessen [8] and now adopted by popular Git clients such as SmartGit (Figure 2).

The success of this model relies on two factors. Firstly it provides a clear way to manage the final project while enabling the development of different features and experimentation. Secondly, it supports this workflow while preserving a major Git feature, the opportunity for specialist groups to work on specific topics before integrating ("pushing" in Git terminology) their changes back to the common model.

Analogies with BIM

The main concept of Building Information Modelling presents some striking analogies with the versioning and revision control systems presented above. An idealized BIM system revolves around a "master model" which contains the entirety of information relating to a building. While visualization, design, and analysis are of paramount importance, the ambition is for this master model to be used for the whole range of activities involving a building, not only in its design stage but also in its entire life-cycle (Figure 3).

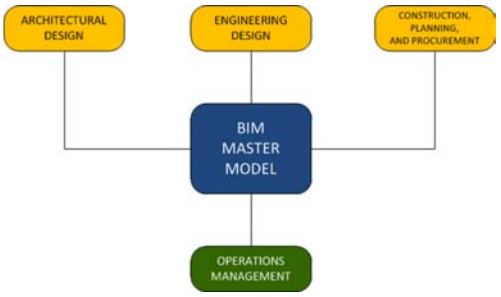


Figure 3
BIM conceptual model

In this system direct parallels can be drawn between BIM and an SVN system. The master model is effectively a central repo. The various AEC specialists correspond to the developers and associated professionals working in software (though intended for code only, most SVN systems eventually included assets such as graphics etc). Initiatives like Autodesk's "BIM In The Cloud" make the similarities even more pronounced. Effectively, BIM is meant to support a practically identical workflow, where individuals work to a central repo/master model, and have to solve conflicts manually every time a revision calls it (in SVN terminology).

This also highlights the ambition to describe and manage all building information instead of just the visual aspects in one model, something which has led both in semantic confusion between the M in BIM, as well as suggestions for acronyms like BIMM (Building Information Modelling and Management). Irrespective of the terminology that will prevail in the industry, the aspect that is pertinent to this paper is the change of focus for the digital; models now are redefined semantically. The digital model is not a visual model of a physical object, but a multi-dimensional (4D, 5D, 6D, effectively nD) representation of consecutive semantic layers of information.

CURRENT APPROACHES AND LIMITATIONS OF BIM

When there is talk about the limitations of BIM, the industry tends to concentrate on requesting additional features, usually as implemented in specific software packages. While this is understandable from a business perspective, where immediate benefits and return of investment are of primary importance, it is safe to say that these are issues that can be addressed in the short- or mid-term. What this conversation ignores however is an obvious limitation with the existing BIM concept.

As established above, the "BIM in the cloud" objective has striking conceptual similarities with SVN development. The limitations of SVN were mentioned above, but BIM might be faced with such limitations much more quickly. Despite the various specialisms, software engineering tends to be far more uniform, with different programmers generally sharing a mindset, approach, and method of working. This is very different to building design where the gap that separates even directly interfacing disciplines, such as architects and structural engineers, is considerable.

This places a disproportionate load on the programmers of BIM software; effectively they are asked not only to combine a number of highly specialized software packages but also to have this accessible, editable, and updateable by all specialties. The problem of resolving HEAD revision conflicts was a major one in SVN; it can stop BIM in its tracks. It is not a coincidence that when BIM success stories are presented, the advantages tend to be concentrated in visualization and clash detection [9]

The current standard approach to deal with the limitations of the "master model" comes from the concept of the reference (or even sub-reference) models, with the ambition to integrate this in future BIM developments such as BIM Maturity Level 3 in the UK. The implementation of this in actual commercial applications is limited, and it is doubtful how effective this is in practice. Simultaneously it raises significant technological challenges; the proprietary underlying data structures of the leading BIM packages make interoperability and integration particularly difficult . Infamously, Autodesk Revit, arguably the leading BIM application, does not save down to an older Revit version, thus requiring the same model and the software to be in the same version for the entire project team. Enabling cross-discipline collaboration in an extremely diverse and fragmented industry such as the construction industry appears particularly challenging with current approaches.

Researchers have addressed these issues by utilizing the Model View Definitions (MVDs) which allow the filtering of data according to pre-set parameters [10]. This enables the identification of discipline-specific aspects thus assisting different specialists with the BIM transition. The issue of the underlying data model though remains extremely important. The traditional answer by researchers is the Industry Foundation Classes (IFC). Despite being in development for more than a decade, and overseen by the buildingSMART alliance which includes the main software vendors, their actual adaption in practice has been very limited. Cerovsek for example reports that IFCs have failed to achieve interoperability, trust, and comparability [11]. This is likely to be due to the mega-schema approach adopted by the IFCs. Despite being an open and extensible schema, its attempt to fit everything within one data structure makes its chances of large-scale industry adoption unlikely.

It is here where learning from RCS can be not only particularly useful, but also open a great range of possibilities for further development. What is proposed in this paper is effectively a method for tagging, linking, and combining these conceptual layers. The framework can be seen as a Distributed Building Information System, henceforth referred to as DBIS.

A METHODOLOGY FOR A DBIS

The development of a DBIS framework required addressing five key areas with regard to building information:

- how the information is represented as data
- how the data is organized, classified, and linked
- how the data is transmitted, stored, and shared
- how the data is manipulated, edited, and analysed
- conflict resolution

Obviously a single paper cannot address all these points in implementation-ready depth. However the top-level methodology is provided here, in order to lay the groundwork of the framework and illustrate how implementation is achievable.

Information representation

The representation of all building-related information should be on an XML-like structure, based on a human-readable text-format and a hierarchical structure. Variations such as JSON [12], and various compression formats are a possibility for size optimization and transmission purposes.

A key difference between DBIS and BIM is that a building model is not constrained in one file. Instead the project consists of various files that represent different layers of information. Hence, not all the personnel working on a building will need to engage with all files. Instead, different specialists work with different files, depending on project requirements and/or contractual rights (for example a building services contractor might not have access to the geotechnical engineering components). This is a radical departure from the MVD/IFC model which still assumes that all model-related information is constrained in one file.

In this, the DBIS corresponds to developing a software application; building-related information is split in packages, further subdivided in objects. Each object is an individual text file, with variables that describe different properties. In addition, an object includes a list of the allowed actions that can be performed on it; these can be also restricted by contributors.

Table 1
Sample object structure

filename	b12.beam
Object Name	**Beam B12**

key variable	type=column
	topConstraint=s1.slab
	bottomConstraint=c1.column
	…

linked children files	material=b12.mat
	geometry=b12.geo
	…

allowed functions	changeWidth changeSupport …
locked functions	~~changeMaterial~~ ~~changeDepth~~ …

So for example a Beam object might allow a changeDepth function, but an architect might have changed this to false for a specific beam to prevent a contributing engineer from increasing the depth (Table 1). The main differences with a programming class are two:

- this "building source code" is typically not manipulated directly, but via software applications which will render the information and provide specific tools

- the actions that an object can receive do not contain their own instructions (as a programming class) but simply provide a flag for organizing the data

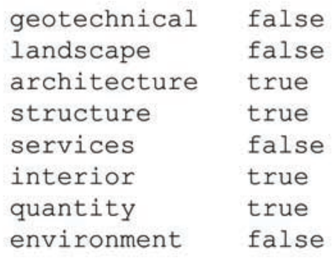
Data organization, classification and linkage

The internal representation of the information should correspond to the notion of one base Object as it is standard practice in modern object-oriented programming languages such as Java. This will enable the internal consistency of all building information, as well as backwards compatibility.

The first main classification of objects is between tangible and intangible properties, on corresponding to physical objects (e.g. a column) and the other to human-defined concepts (e.g a room). With such a distinction established, the rest of the properties can be described.

The XML-like structure allows for categorizations; for example a geometry file includes <Shape> nodes which include vertex arrays, etc, drawing from the established Scene-graph paradigm in 3D graphics frameworks [13].

An important aspect is the need to classify objects depending on their usability. Each object should include a range of boolean tags that would identify the aspects to which it is pertinent to. For example a column on the third floor could have the following tags demonstrated in table 2.



geotechnical	false
landscape	false
architecture	true
structure	true
services	false
interior	true
quantity	true
environment	false

Table 2
Example of object classification and tagging system.

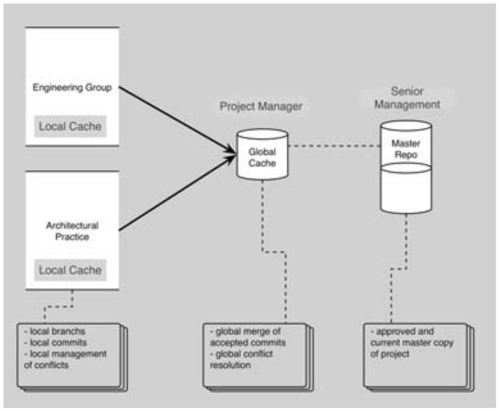
These tags determine not only the interoperability between objects, but also the software application(s) in which these can be manipulated (as will be explained later).

Data storage, transmission, sharing and infrastructure

The text-based nature of the file architecture, as well as the fact that different data is stored in different files means both that individual objects are lightweight, and that not all contributors need access to the entire file set. Filtering can be done by package, by the tag described above, or by additional tags that would describe contractual obligations.

Transmission and distribution is done via the web, following the DRVS principles established above. A key element here is that different groups of contributors have access to different versions of the model. This can mean that specialist groups can work on different iterations before feeding back to the common system. Examples of such actions could be the architects working on the design, the structural engineers working on the structure, the visualization specialists working on add-on site components etc (figure 4). This is done on local branches of the source file system, after the Git model presented earlier. The option of keeping only the required elements, as well as the text-based nature of the source data means that this is lightweight and hence achievable.

Figure 4
Example of distributed model



The system uses the client-server model from distributed computing in order to provide a scalable distributed application structure that partitions tasks and workloads in a way similar to specialist teams working on a project. The model can offer access and concurrency transparency for the users. DBIS servers can use a modern file system like ZFS [15] which amongst other features supports protection against data corruption, efficient data compression for the project repositories and continuous integrity checking; ideas that exist in the Git model. The DBIS client-server communication will be governed by a cross-platform application layer interface (API). As the model is based on the XML-like Object based system the application layer protocol can be implemented with both a sophisticated high performance network protocol with build-in data compression (Web/Cloud service etc.) or a backwards compatible text streaming Transmission Control Protocol (TCP) service for any software package that supports scripting via a high-level programming language. Below we discuss the importance of an easy to use and implement communication with any software used by experts.

Data manipulation, editing, and analysis

It is perhaps in this area that the DBIS offers the greatest potential. Despite significant advances in recent years, BIM software remains heavy, pushing the limits of the available computing power. At the same time, it has limited tolerance to errors, and it is absolutely fundamental to follow good practice when setting up the original model. Moreover, as it has been pointed out, despite the need for Integrated Project Delivery (IPD), most work remains intra-disciplinary and BIM has done little to address that [14].

The concept behind DBIS assumes that, noble as the aim of full interdisciplinary working might be, perhaps it is not achievable or even desirable. Different specialisms have evolved different methods of working largely for good reason; constraining them in a single platform by force is unlikely to lead to harmonious collaboration. Instead it is more likely to lead in increased friction.

In addition, the BIM software developer is almost condemned by the problem he has set for himself. By definition, the cutting edge in each discipline will be achieved by specialist software. It is unlikely that an all-inclusive BIM platform will be able to offer simultaneously cutting-edge dynamic analysis for earthquake engineering, cutting-edge visualization, and cutting-edge program management. It is also doubtful that this is needed.

What is needed is a common description of the project, not of the ways of manipulating it. The DBIS framework functions as a way of describing a design or an existing building. This can be fed back into specialist software for manipulation, editing, and analysis. Having performed the specialist operations, the specialist software will have two types of outputs. One will be the discipline-specific (e.g. the thermal comfort calculations by an environmental designer, a 3d walkthrough by a visualization expert etc). The other is any updates done to the project (e.g. changing the insulation of a wall), which will be fed back to the central DBIS.

At the same time, this approach addresses problems that are currently plaguing BIM, such as backwards compatibility and the "closed silo" effect. After all, the DBIS is not a file format, but a semantic cataloguing, organization, and description of building data. Also, the type-based structure means that upgrades, downgrades, and cross-platform transformation tasks can be automated much more simply.

Conflict resolution
Conflict resolution is obviously a very important topic, not only in software engineering or in building design, but in any collaborative activity. As mentioned previously, the introduction of BIM has improved conflict resolution in aspects such as clash detection. In this, the DBIS offers at least an equal, if not much improved, performance. Clashes, changes, and conflicts are identified in a similar way. However, not constraining the source models to one all-inclusive application means that the software's task is much easier. This is likely to allow for more seamless working. Indeed the only software that might be shared by all members of a team could be a viewer-only application that visualizes all the semantic layers of the DBIS.

CONCLUSIONS
The framework presented in this paper aims to improve and expand on concepts that already exist in nascent form behind BIM. The technology exists for implementation, and the framework is generic allowing development in different languages and accommodating different programming structures.

Drawing from the example presented above, it is important to make some key observations from this workflow:

- users only download the files that are applicable to their project
- different applications are used for different tasks, thus allowing for far leaner software; the DBIS functions as a data exchange medium
- a common lightweight viewer is shared by all parties; this functions as a geometry visualization tool, conflict identifier and updated, and data viewer
- parallel working is enabled
- branching is cheap in resources
- specialist groups can try a number of iterations and different designs without interacting with a "master" model
- significant improvent in back-up and version control; this not only enhances collaboration but addresses the issue of As-Built drawings / models, as well as Whole Life-Cycle Monitoring
- the similarity of the structure with Git suggests better security for project data

As is the case for any framework, many issues remain to be addressed.

Key possible concerns are:

- cross-platform and cross-vendor "buy-in" for a single data-exchange model; similar problems have appeared for file formats that deal with geometry representation such as COLLADA and gbXML. The ambition is that the lightweight nature of this approach will make implementation much easier.
- the cost in time and effort of resolving conflicts would probably require a specialist, effectively a new professional role
- software vendors of individual DBIS-compatible applications would have to observe specific requirements to ensure full compatibility; it is likely that detailed schemas need to be developed

While these concerns are important, previous examples in computer science and software engineering indicate that these obstacles are not insurmountable, either from a single vendors (e.g. Sun Microsystems and the development of Java) or from the wider open source community (as with Git itself).

An immediate first step is implementation in a specific system. A strategy for implementation in Java, with XML as the data representation system is currently being developed. Fittingly, the initial data exchange system will rely on a Git implementation.

The promise of DBISs is significant; it harnesses the good points behind BIM, as well as the power of cloud and distributed computing. Simultaneously, it addresses issues that have plagued BIM from the start, both technical (complexity and resources requirements of software) and professional (such as intellectual property and contractual issues between collaborators). At the same time, it deals directly with aspects of the Built Environment, such as whole lifecycle monitoring and management, which are becoming constantly more pertinent. More research and development is needed to examine this promise further and identify its limitations.

REFERENCES

Angel, E L and Shreiner, D 2011, *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL*, Pearson Education

Cerovsek, T 2011, 'A review and outlook for a ‘Building Information Model’ (BIM): A multi-standpoint framework for technological developmen', *Advanced Engineering Informatics*, 25, pp. 224-244

Chacon, S 2009, *Pro Git*, Apress

Holzer, D 2011, 'BIM’s seven deadly sins', *International Journal of Architec-tural Computing*, 9, pp. 463-480

Katranuschkov, P, Weise, M, Windisch, R and Fuchs, R 2010, 'BIM-based generation of multi-model views', *CIB*, W78 , pp. 10-15

Oliver, A 2013, *BIM: Change Culture*, New Civil Engineer

[1] http://www.nationalbimstandard.org/about.php

[2] http://subversion.apache.org/features.html

[3] http://www.youtube.com/watch?v=4XpnKHJAok8

[4] https://git.wiki.kernel.org/index.php/GitSvnComparison

[5] http://msdn.microsoft.com/en-us/library/hh850437.aspx

[6] http://googlecode.blogspot.co.uk/2011/08/announcing-git-support-for-google-code.html

[7] http://nvie.com/posts/a-successful-git-branching-model/

[8] http://www.json.org/fatfree.html

[9] http://www.open-zfs.org/wiki/Announcement