This article is published within the book: Alias Abdul Rahman (ed.), Advances in 3D Geoinformation, Lecture Notes in Cartography and Geoinformation, Springer, 2016, ISBN 978-3-319-25691-7

Managing versions and history within semantic 3D city models for the next generation of CityGML

Kanishk Chaturvedi[1*], Carl Stephen Smyth[2], Gilles Gesquière[3], Tatjana Kutzner[1] and Thomas H. Kolbe[1]

[1]Chair of Geoinformatics, Technische Universität München, Germany
(kanishk.chaturvedi,kutzner,thomas.kolbe)@tum.de
[2]OpenSitePlan
steve@opensiteplan.org
[3]LIRIS, Université de Lyon
gilles.gesquiere@liris.cnrs.fr

Abstract Semantic 3D city models describe city entities by objects with thematic and spatial attributes and their interrelationships. Today, more and more cities worldwide are representing their 3D city models according to the CityGML standard issued by the Open Geospatial Consortium (OGC). Various application areas of 3D city models such as urban planning or architecture require that authorities or stakeholders manage parallel alternative versions of city models and their evolution over time, which is currently not supported by the CityGML standard 2.0.

In this paper, we propose a concept and a data model extending CityGML by denoting versions of models or model elements as planning alternatives. We support transitions between these versions to manage history or evolution of the city models over time. This approach facilitates the interoperable integration and exchange of different versions of a 3D city model within one dataset, including a possibly complex history of a repository. Such an integrated dataset can be used by different software systems to visualize and work with all the versions. The versions and version transitions in our proposed data model are bi-temporal in nature. They are defined as separate feature types, which allow the users to manage versioning and to perform queries about versions using an OGC Web Feature Service. We apply this data model to a use case of planning concurrent versions and demonstrate it with example instance data. The concept is general in the sense that it can be directly applied to other GML-based application schemas including the European INSPIRE data themes and national standards for topography and cadasters like the British Ordnance Survey Mastermap or the German cadaster standard ALKIS.

Keywords Semantic 3D city models, CityGML, planning versions, history, city model lifecycle

1 Introduction

Semantic 3D city models describe the urban topography by decomposing and classifying the occupied physical space according to a semantic data model. The relevant real world entities are represented by objects with thematic and spatial attributes and interrelationships to other objects. However, authorities or city planners often face issues with data heterogeneity as the data is gathered from different sources at different times with differing geometric and semantic modeling tech-

niques (Kolbe 2009). CityGML (Gröger et al. 2012) is an international standard issued by the OGC, facilitating the integration of heterogeneous data from multiple sources and the representation of the geometrical and semantic attributes of the city level objects along with their interrelationship to other objects – making it a highly semantically enriched data model in this way. As a result, today, more and more cities worldwide are representing their 3D city models according to the CityGML standard.

CityGML is an important source of information in urban planning, architecture, business development and tourism. These areas of application often address planning alternatives of buildings or other structures, e.g. for comparison by a reviewing body. The planning alternatives are not different versions of actual structures at different times but different structures that might be substituted for one another. However, CityGML currently does not support versions. Thus, the motivation behind this work is to extend CityGML by mechanisms for denoting versions of models or model elements as planning alternatives.

In the paper, we propose an approach to extend the CityGML data model and exchange format to support different versions and version transitions to allow identification and organization of multiple states in a city model. The approach is helpful in dealing with two important facets of multi-representation of semantic 3D city models (Gröger et al. 2005). The first facet is maintenance of the complete history or evolution of the city model, which is supported by version transitions having bi-temporal attributes answering the questions "How did the *city* look like at a specific point in time?" and "How did the *city model* look like at a specific point in time?". The second facet of multi-representation is managing parallel alternative designs of the objects at the same time. Although there have been a few approaches to manage versions and history of 3D city models within 3D geodatabases (Gröger et al. 2005), our approach extends CityGML to support versions and version transitions. This approach allows the different versions to be used in an interoperable exchange format and exchanging all the versions of a repository as one dataset. Furthermore, this single dataset can be used by different software systems to visualize and work with all the versions.

In the following sections, after a brief introduction of use cases and related work for supporting versions, we explain our proposed data model. Further, we give an illustration of the concept by demonstrating a scenario with the help of an example CityGML document.

2 Use Cases

Semantic 3D city models are used in applications like urban planning or for helping in decision making processes. As cities are constantly evolving, it is also necessary to record their changes over time. It is important that each city object may be represented with its own lifecycle. For example, for a time sequence, a building may be constructed, modified, destructed and replaced by other ones (Pfeiffer et

al. 2013) as illustrated in Figure 1. Similar needs can also be identified in serious game projects where objects of a scene may evolve according to a given scenario. For instance, a building may be represented in different states like “destroyed”, “burned” or “partially destroyed” that can be called by the application in coordination with user actions (Chambelland et al. 2011).

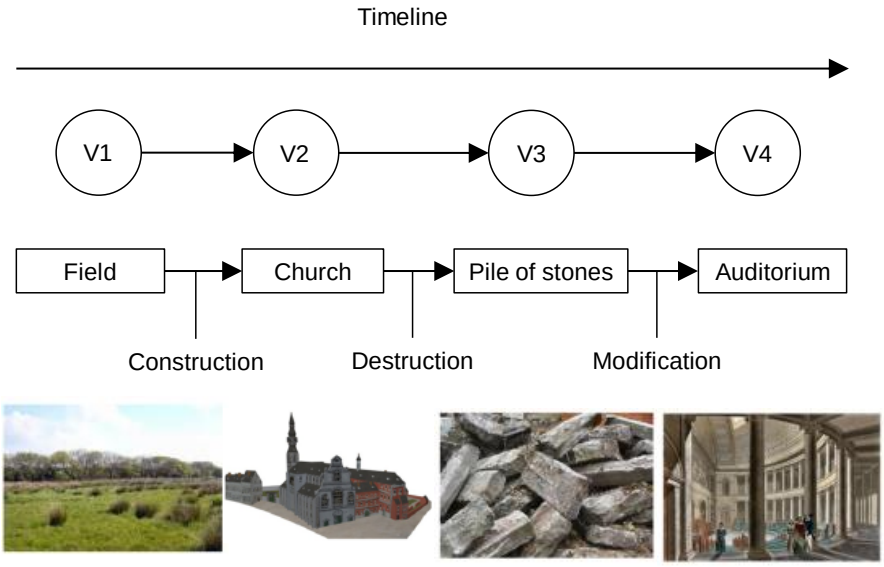

Fig. 1. An example of “historical succession”. Image adapted from (Pfeiffer et al. 2013)

Semantic 3D models also have a growing role in the documentation and reconstruction of both historical and contemporary events. Examples include crime scene and accident reconstructions, representation of battles and other historical events, archival descriptions of historical structures before demolition, documentation of construction and demolition of buildings. Each of these involves a sequence of versions of a “reality” at a certain time or in a certain time sequence. Events are often reconstructed from conflicting and incomplete evidence and a complete reconstruction must allow branching to handle the alternative possibilities (see Figure 2 on the next page). Modeling such approach would also allow backward compatibility to handle multiple representations of the past of a city. A given date may be a starting point to imagine the past and constructing several scenarios. Any other framework does not support such feature so far.

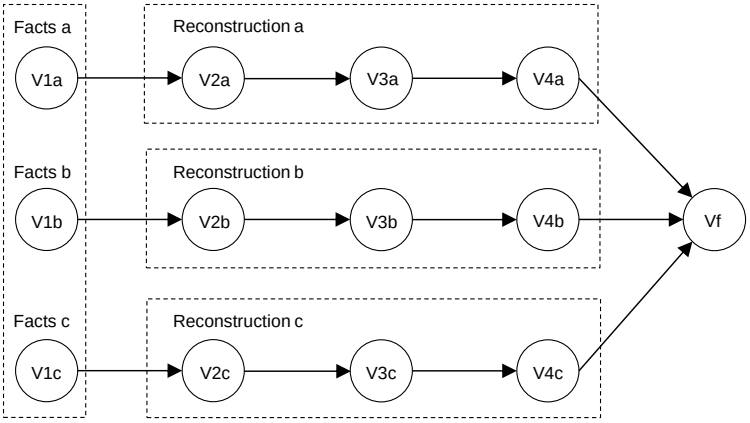

Fig. 2. Reconstruction of events to handle alternative models

As shown in Figure 3, the model should be capable of representing two different states of the city in the past, indicated here by V0 and V0'. In urban planning scenarios, different planning authorities can also work with alternative planned versions (V2 and V3 respectively) at the same time to insert a newly generated object or delete or update any existing object.

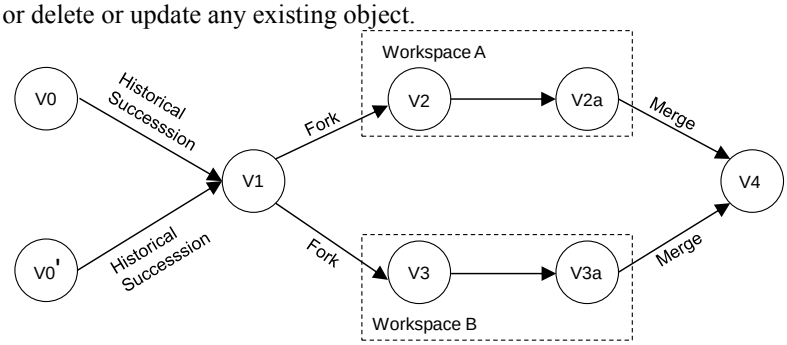

Fig. 3. History and version management

3 Related Work

3.1 Related Standards and Tools

3.1.1 AAA/INSPIRE

Two semantic information models are to be mentioned here which provide the concept of chronological versioning, i.e. the INSPIRE data specifications which define Europe-wide consistent conceptual schemas for various data themes as part of the European spatial data infrastructure initiative INSPIRE (Infrastructure for

Spatial Information in the European Community) and the German AFIS-ALKIS-ATKIS (AAA) Reference Model which defines conceptual schemas for the German geospatial base data of geotopography and real estate cadaster.

INSPIRE defines several requirements and recommendations for modeling life-cycle information of spatial objects which include UML stereotypes and properties allowing for bi-temporal modeling of geospatial objects. Furthermore, a separate property exists for denoting a specific version of a geospatial object (INSPIRE Drafting Team "Data Specifications" 2014). However, currently only exchanging the last version of spatial objects is supported by INSPIRE; historic versions cannot be provided yet (and especially not within one data file).

Similarly, AAA allows for managing multiple versions by means of a specific AAA versioning schema, which makes use of life-cycle properties and an object container, where versioned objects are registered. However, the AAA versioning schema exists only as a concept: no concrete implementation is available. It is also unclear how an implementation would look like.

3.1.2 IFC

The Industry Foundation Classes (IFC) have been developed by buildingSMART (the former International Alliance for Interoperability), as an open standard for sharing Building Information Models (BIM) data among different software applications. To the best of our knowledge, the concept of temporal version is not currently implemented in IFC, but some extensions have been proposed in the literature. For instance, in (Beer et al. 2004), it is possible to find an approach inspired from the CAD domain.

Extensions of the IFC standard have been proposed in (Zada et al. 2014). Six existing entities from the IFC standard have been suggested to be modified to represent as new entities within the IFC schema to support the idea of object versioning that holds the history of changes to objects of the BIM model. In (Nour and Beucke 2010), a novel approach is introduced, where both object versioning and IFC model are integrated together in an open multidisciplinary collaborative environment. Object versioning gives the possibility to have several versions of the content (attributes' values) of an object. The development of design in terms of addition of new objects, deletion of objects or modifications of attributes' values of pre-existing objects can be captured in a graph structure.

3.1.3 Tools for Supporting Versions

The concept of versions has successfully been incorporated by Oracle using the Oracle Workspace Manager (Beauregard and Speckhard 2014). The Oracle Workspace Manager allows managing multiple versions of the data in the same Oracle Relational Database Management System in the form of workspaces. A workspace is a virtual copy of the data, which separates the collection of changes in the dif-

ferent versions from the live (production) data. The version-enabled data are stored in separate tables with additional columns representing the version metadata. Such additional columns contain the version and workspace of each data row along with the date and time of each update. The database view is created on the version-enabled table and triggers are defined to enable SQL operations such as insert, delete, and update. This approach allows preservation of the structure of the original table and shows the data of only the respective version. The Oracle Workspace Manager also provides the management of historical data by using savepoints and the means for resolving possible conflicts during the merging of the different versions.

Similarly, versions can also be managed within ESRI ArcSDE Geodatabases (ESRI 2004). ESRI also supports managing history, performing "what-if" analysis and conflict detection and resolution. However, there are no interoperable exchange formats related to both Oracle and ESRI which would allow exchanging all versions of a repository as one dataset, nor which would allow use of the same dataset by both Oracle and ESRI.

3.1.4 GIT/SVC

Version or revision control systems (VCS/RCS) such as git (2015a), Mercurial (2015b), Concurrent Versions System (CVS) (Vesperman 2006), and Subversion (SVN) (2015c), have a structure and goals similar to the approach described in this paper. VCS were developed primarily to support parallel development within a single project. Although there are operational and architectural differences between these systems, they all maintain versions of collections of files comprising a project. These collections are often organized in a tree structure, similar to a computer file system directory hierarchy.

A VCS has the representational power to manage changes, parallel updates, and merges of versions of CityGML models with one exception: versions always represent change in the forward direction of time. The collection maintained by a VCS is rooted in an original version, that is, the versions form a rooted directed acyclic graph (DAG). The original version is the oldest version and it is not possible to create versions earlier than the root.

In addition to the problem of forward-only temporality, a VCS also has a designated node in the DAG, usually called the head, and a distinguished path in the DAG, from the root to the head, usually called the "trunk" or "main branch". Neither of these is required for maintaining versions of CityGML models. Although it might be possible to gain the needed representational power by piecing together multiple VCS projects, which share a common root, it would be awkward, at best.

3.2 Temporality in semantic 3D city models

Objects that compose the city such as buildings, bridges, vegetation, and terrain change over time. For instance, for a building, different kinds of transformations can be identified. (De Luca et al. 2010) propose to define the following states for a building: creation, destruction, reconstruction, division (the building is separated in several parts), union, and variation (modifications). Other transformations are linked to the semantic part of the building, for instance when the owner of a house changes (Stefani et al. 2008). Describing the building lifecycle implies taking into account states and spatial changes along a temporal arc (Stefani et al. 2011). However, this scheme must also be extended to all objects of the city including terrain and vegetation, for example. It must take into account semantic, topological, appearance and geometric changes. Changes involving buildings can be sudden or gradual. For instance, a change of property is a sudden event. On the other hand, changes may be progressive: for example, building demolition is a short event, but the construction of a Gothic cathedral is a long event that lasts several centuries. Furthermore, historical building deteriorations may take centuries or millennia (Stefani et al. 2010).

Keeping the possibility of exchanging data with other tools and making a snapshot for a given date of the city is essential. Several papers propose methods based on CityGML. For instance in (Pfeiffer et al. 2013), the CityGML scheme is modified to add temporal information on buildings. However, this method allows to register only definite states. CityGML scheme modification and possible standardized exports are not discussed. Another method proposed is based on a modification of CityGML (Morel and Gesquière 2014). In this paper the authors propose to add two additional concepts to take into account the possibility for a city object to change and the time value which fixes this change in the city lifecycle. However, this method does not support the possibility of having different scenarios.

3.3 Requirements

Considering the limitations in related standards and tools, the following requirements have been gathered to be included in the proposed approach:

- None of the standards supports managing multiple historic versions. INSPIRE supports exchanging only the last version of spatial objects. The proposed methodology should allow supporting multiple historic versions within one data file.
- The existing approaches allow only forward-temporality. The proposed approach should allow backward compatibility to handle multiple representations of the past of a city.
- Although the DBMS systems such as Oracle or ESRI ArcSDE Geodatabases already support versions and conflict managements, the proposed approach

within CityGML documents would allow exchanging all versions of a repository as one dataset and furthermore, the same dataset would be used by DBMS systems such as Oracle and ESRI.

4 Methodology

In our proposed methodology, versions become first class objects in CityGML and are modeled as feature types. Figure 4 presents the methodology as a UML model.

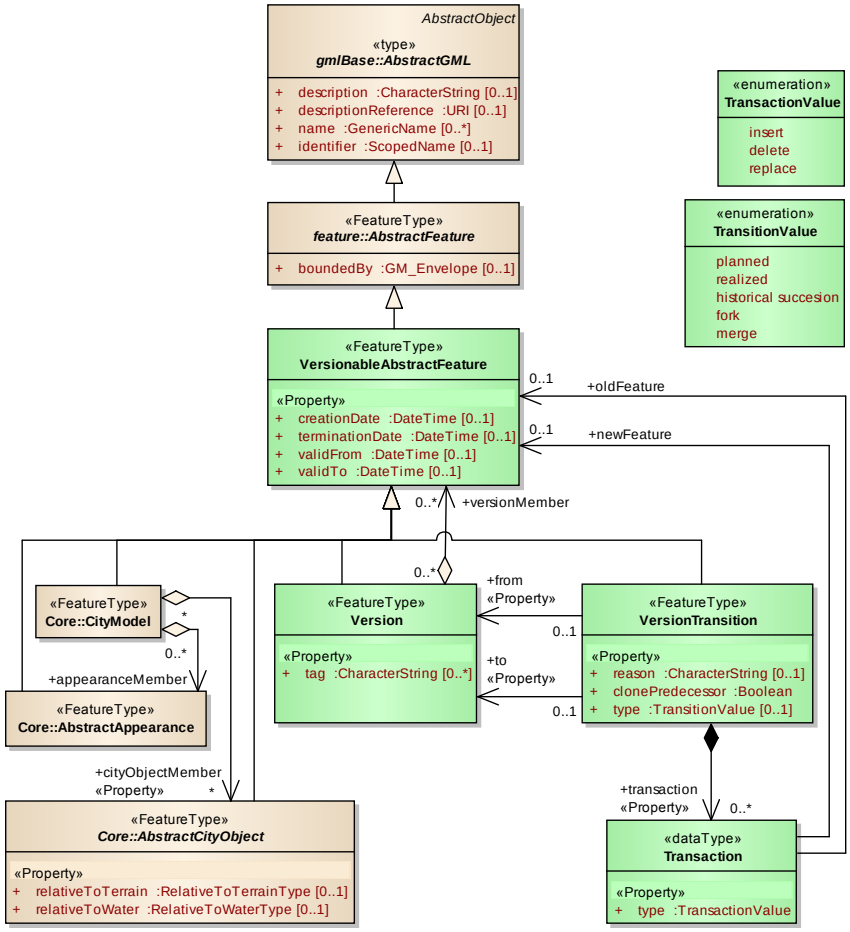

Fig. 4. Version and version transitions defined as UML model. Newly introduced classes are shown in green. Names of abstract classes are in italics.

The city object within a specific version is assigned a stable object ID for the entire lifetime of the object – the so-called major ID. This ID is supported in GML 3.2.1 through the element *identifier* in the class *AbstractGML*, which is used for providing globally unique identifiers. Further, an extension to this major ID is given in the form of a sub ID or minor ID to distinguish different versions of the same real world entity. A separator symbol is introduced to separate the identifier from the version. For example, the specific version of a city object can be denoted as *Building1020_Version1*, where *Building1020* is the *gml:identifier* representing the stable major ID globally and *Version1* is the minor ID to represent the specific version of the building object *Building1020*. This concatenated major and minor ID (*Building1020_Version1*) is used as the *gml:id* to distinguish the different versions of the same real world object. One CityGML instance document can therefore include multiple versions of the same real world object having different *gml:id* but identical *gml:identifier* values. The idea of minor ID and major ID has been adopted from the German AAA model and the INSPIRE Data Specifications (INSPIRE Drafting Team 2014). Referencing objects by either their minor ID or their major ID is inspired by (Cox 2006).

We introduce a new abstract subclass *VersionableAbstractFeature* of class *AbstractFeature*, from which, in the future, all geo-object types shall be derived in order to become version managed. The class *VersionableAbstractFeature* contains four time attributes for expressing a bi-temporal existence model for versions. These attributes are: *creationDate* and *terminationDate*, reflecting the database transaction time, and *validFrom* and *validTo*, reflecting the actual world time. This approach is similar to the existing INSPIRE model where these attributes can be used to query how the *city model* looks like at a specific point in time and how the actual *city* looks like at a specific point in time. These attributes can be defined as an extension to the CityGML core module and may replace the existing attributes *yearOfCreation* and *yearOfDemolition* attributes in the CityGML building module. Furthermore, the class *Version* allows each version to be denoted by a set of user-defined *tag* attributes. With the help of such *tag* attributes, the user can, e.g, search for a version developed by a specific worker. The city objects within each version can be referenced in two ways: by using a simple XLink to the *gml:id* of the referenced object which references a specific version of a real world object, or by using the XML Path Language (XPath) (W3C Recommendation 2011). XPath in conjunction with XLink allows referencing an object element in a remote XML document (or GML object repository) using the *gml:identifier* property of that object (Cox 2006). The XPath-XLink approach provides a general reference to a real world object by its major ID and does not take into account a specific version. This approach allows selecting multiple instances with the same *gml:identifier* value, but with a different *gml:id*. For example, by using a single XPath-XLink query, the user can retrieve multiple versions of the CityGML building parts within the same version of the building. However, it is necessary for the application to determine which specific version of the real world object representation should be used. The attributes *creationDate*, *terminationDate*, *validFrom*, and *validTo* can

also be used to choose the appropriate version that was valid at a specific database or real world time, respectively.

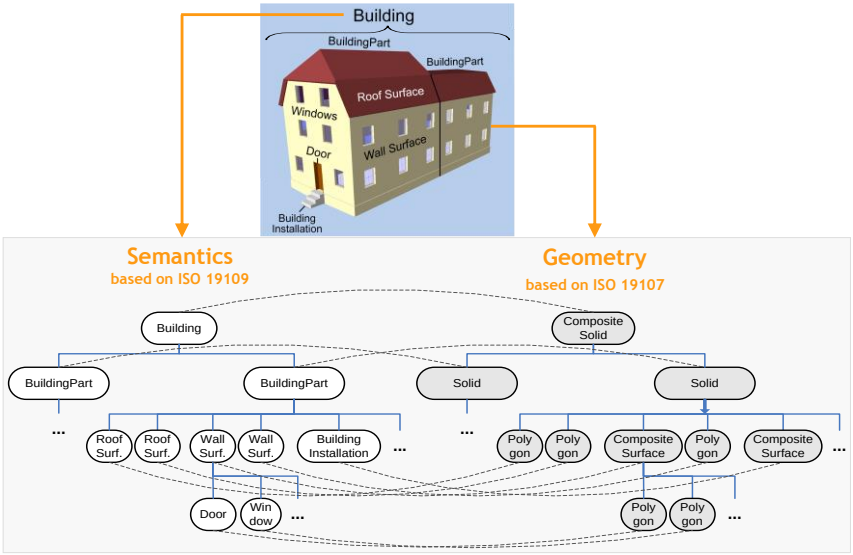

Fig. 5. Issues with versioning of aggregated features. Image taken from (Stadler and Kolbe 2007)

In CityGML, features can also have aggregated sub-features. For example, a building feature can consist of features such as roof surface or wall surface, which may further consist of sub-features such as window or door. However, in case of a change in any of the sub-features, the model would require changing all the parent features in the aggregation levels above because aggregate objects point to their parts. If the part is replaced by a new version with a new *gml:id*, the pointer in the aggregate object also will have to be updated. This will create a new object version also for the aggregate object, and so on, following up the aggregation hierarchy. For example, the window in Figure 5 has been replaced by a new window with insulated glazing and a new frame. In the new version, the window will have to be changed along with its parent features. In our approach, this issue has been resolved by referencing the Major ID attribute using the XPath-XLink mechanism (see an example in section 5).

Furthermore, in order to manage history or evolution of city models, version transitions are also supported. Version transitions represent the causal relationships between the version snapshots. A snapshot is a representation of the state of all features of the entire city model at a specific point in time. It explicitly links to all objects in their versions belonging to the respective city model version (see Figure 6).

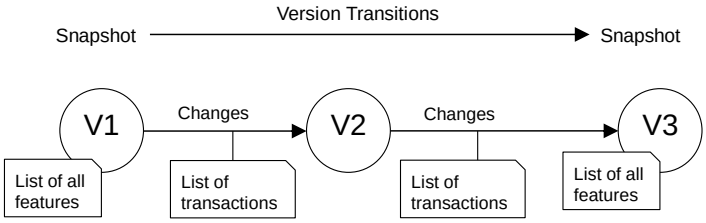

Fig. 6. Representation of version transitions.

In the proposed data model shown in Figure 4, version transition is modeled as a separate feature type *VersionTransition*, and represented by the attributes: (i) *reason*, reflecting the reason for the change in version, (ii) *clonePredecessor*, which is an indicator whether the list of features is derived from a predecessor version, (iii) *type* of transition, whether the transition is planned, realized, a historical succession, fork or merge, and (iv) *transaction*, a list of updates/transactions from a predecessor value, reflecting what types of transaction values are contained such as insert, delete or update. There are a few advantages with version transitions. This approach requires low memory or storage requirements. It is similar to the combination of full back-ups and incremental back-ups. It may also be used to stream dynamic changes.

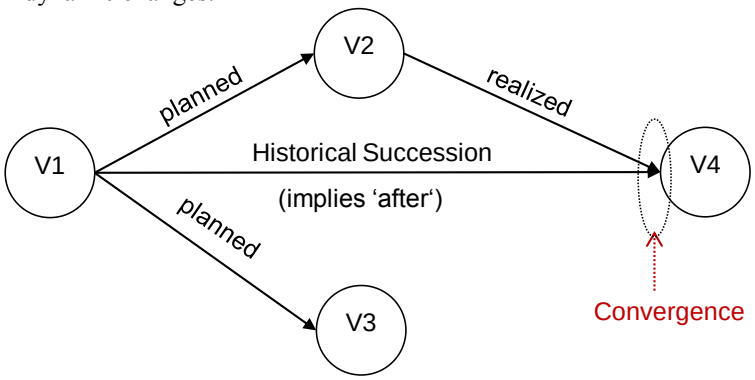

Fig. 7. Conflicts/convergence of multiple versions

However, a limitation with this approach is that it may be necessary to determine the actual members of a version by going back via the predecessors. One important aspect with version transitions is the merging of two different versions, which may lead to possible conflicts. For all such convergence situations, it must be ensured that the members of the converged version/state can be determined unambiguously. As shown in Figure 7, the easiest and safest way is to require that at maximum one of the incoming transitions has transactions. Such transitions are required to be able to detect who has changed an object and whether there are any conflicts. (Doboš and Steed 2012) have identified the methods supporting differencing and merging of 3D models. Several methods have also been identified to

detect changes in CityGML files such as (Pédrinis et al. 2015) and (Redweik and Becker 2015). These methods may be useful tools to resolve conflicts in similar way as with the "diff" command in SVN.

5 Illustration of Concept

This section explains an example scenario for managing different versions within CityGML documents. Figure 8(a) represents the evolution of the city in the form of different versions of CityGML documents.

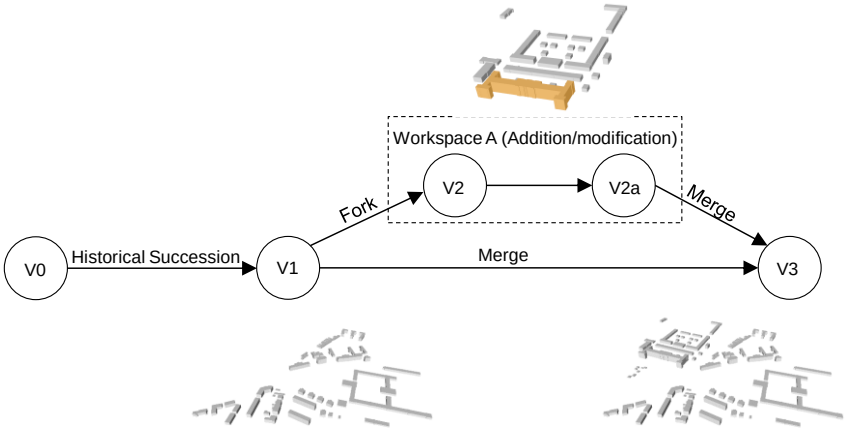

Fig. 8(a). An example scenario of evolution of city in the form of different versions

The successive versions represent the state of all features of the entire city at specific points in time. In addition, the authorities can work, in parallel, with different workspaces or branches to insert, delete or modify the objects. Such additions can be merged with the earlier versions of the CityGML documents to form the final versions. Figure 8(b) represents the evolution from initial to final versions.

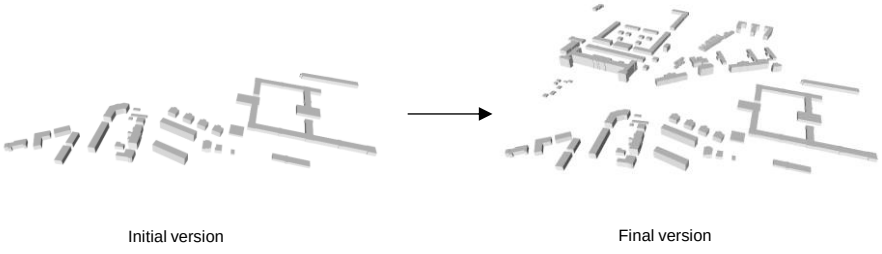

Fig. 8(b). The initial version versus the final version

However, looking more into details, the following example represents one such possibility of modification scenarios. As shown in Figure 9, a building with the major ID *B1020* has a *function* property Office and one of its building parts with major ID *BP12* has a *roofType* property *Flat*. Over a period of time, the building's function property is changed to *Living* which has been captured in version 2. Furthermore, at a point in time, the *roofType* property of the same building has been changed to *Saddle*.

Below is an example representing the version management in a CityGML instance data set. The building object in version 1 can be denoted as *B1020_t1* at a specific point in time *t1*. However, XPath can be used with XLink to retrieve all the instances of the same building object.

```
<cityObjectMember>
    <Building gml:id="B1020_t1">
        <identifier>B1020</identifier>
        <consistsOfBuildingPart>
            <BuildingPart xlink:href="//identifier[text()='BP12']"/>
        </consistsOfBuildingPart>
        <creationDate>2012-08-02</creationDate>
        <terminationDate>2013-10-10</terminationDate>
        <function>Office</function>
    </Building>
</cityObjectMember>
<cityObjectMember>
    <Building gml:id="B1020_t2">
        <identifier>B1020</identifier>
        <consistsOfBuildingPart>
            <BuildingPart xlink:href="//identifier[text()='BP12']"/>
        </consistsOfBuildingPart>
        <creationDate>2013-10-10</creationDate>
        <function>Living</function>
    </Building>
</cityObjectMember>
<cityObjectMember>
    <BuildingPart gml:id="BP12_t1">
        <identifier>BP12</identifier>
        <creationDate>2012-08-02</creationDate>
        <terminationDate>2014-06-04</terminationDate>
        <roofType>Flat</roofType>
    </BuildingPart>
</cityObjectMember>
<cityObjectMember>
    <BuildingPart gml:id="BP12_t3">
        <identifier>BP12</identifier>
        <creationDate>2014-06-04</creationDate>
        <roofType>Saddle</roofType>
     </BuildingPart>
</cityObjectMember>
```

The instance data can also include <version> elements for managing different versions of an object. However, due to limited space availability, this example illustrates a simple version management where it is sufficient to just use the bitemporal time attributes and the major/minor IDs.

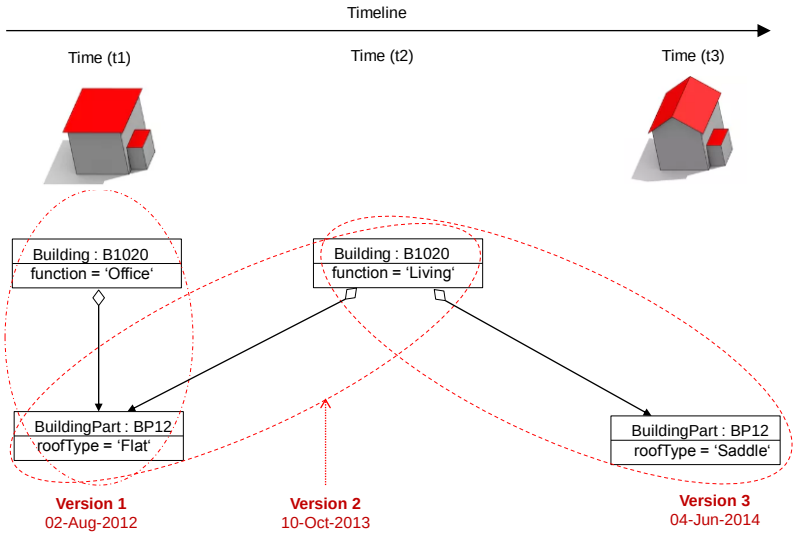

Fig. 9. An instance example of versions representing modification of a building

6 Conclusions

In this paper, we present a new modeling approach and a smart implementation for supporting the management of versions and history within CityGML. The advantage of our approach is that it not only facilitates the data model for supporting different versions, but also allows the different versions to be used in an interoperable exchange format and the exchange of all versions of a repository within one dataset. Such a dataset can be used by different software systems to visualize and work with all the versions. The approach not only addresses the implementation of versionable CityGML models but also considers new aspects to previous work such as managing multiple histories or multiple interpretations of the past of a city. However, currently such interpretations represent slower changes; for example, change of the real property values, geometries, or ownership over time. In the future, it is also intended to support highly dynamic properties (comparatively faster changes); for example, variations of thematic attributes such as changes of physical quantities like energy demands, temperature, solar irradiation levels. The proposed UML model handles versions and version transitions as feature types, which allows the version management to be completely handled using the OGC Web Feature Service. No extension of other OGC standards is required.

The mentioned concept is going to be proposed as an official extension to the next version of CityGML (version 3.0). The concept already addresses the possibility that every feature of CityGML can be made versionable. However, in the future it might be required to also make individual geometry objects within city models versionable. It is possible to introduce the versioning of objects at a higher

level in the class hierarchy of GML just below *AbstractObject*. In this way, every object of CityGML (and of GML in general) would become versionable. However, this would require changes in the GML specification, which is out of scope of the OGC CityGML standards working group (CityGML SWG). In the presented form the concept is completely modeled in the framework of the CityGML application schema and can be standardized by the CityGML SWG without changing other OGC or ISO specifications.

This concept also does not require a database or GIS with specific version management capabilities. However, in the future, it will be interesting to find out how the proposed versioning schema can be mapped onto existing version management tools of 3D GIS and databases (c.f. section 3.1.3).

Acknowledgments Part of this work has been carried out within the project Modeling City Systems funded by the Climate-KIC of the European Institute of Technology and Innovation (EIT). We would like to thank Climate-KIC and the EIT for supporting this work.

References

Beauregard B, Speckhard B (2014) Oracle Database Workspace Manager Developer's Guide, 12c Release 1 (12.1) E49170-01.

Beer DG, Firmenich B, Richter T (2004) A Concept for CAD Systems with Persistent Versioned Data Models.

Chambelland J-C, Raffin R, Desbenoit B, Gesquière G (2011) SIMFOR: towards a collaborative software platform for urban crisis management.

Cox S (2006) Object identifiers in GML. https://www.seegrid.csiro.au/wiki/AppSchemas/GmlIdentifiers#gml:identifier_element. Accessed 29 Jun 2015

De Luca L, Busarayat C, Stefani C, et al (2010) An iconography-based modeling approach for the spatio-temporal analysis of architectural heritage. In: Shape Modeling International Conference (SMI), 2010. IEEE, pp 78–89

Doboš J, Steed A (2012) 3D Diff: an interactive approach to mesh differencing and conflict resolution. In: SIGGRAPH Asia 2012 Technical Briefs. ACM, p 20

ESRI (2004) Versioning | White Papers. http://support.esri.com/es/knowledgebase/whitepapers/view/productid/19/metaid/721. Accessed 26 Feb 2015

Gröger G, Kolbe TH, Nagel C, Häfele K-H (2012) OGC City Geography Markup Language (CityGML) Encoding Standard.

Gröger G, Kolbe TH, Schmittwilken J, et al (2005) Integrating versions, history and levels-of-detail within a 3D geodatabase.

INSPIRE Drafting Team "Data Specifications" (2014) INSPIRE Generic Conceptual Model. http://inspire.ec.europa.eu/documents/Data_Specifications/D2.5_v3.4.pdf. Accessed 10 Apr 2015

Kolbe TH (2009) Representing and exchanging 3D city models with CityGML. In: 3D geo-information sciences. Springer, pp 15–31

Morel M, Gesquière G (2014) Managing Temporal Change of Cities with CityGML. In: UDMV. pp 37–42

Nour M, Beucke K (2010) Object versioning as a basis for design change management within a BIM context. In: Proceedings of the 13th international conference on computing in civil and building engineering (ICCCBE-XIII), Nottingham, UK.

Pédrinis F, Morel M, Gesquière G (2015) Change Detection of Cities. In: 3D Geoinformation Science. Springer, pp 123–139

Pfeiffer M, Carré C, Delfosse V, Billen R (2013) Virtual Leodium: From an historical 3D city scale model to an archaeological information system. ISPRS Ann Photogramm Remote Sens Spat Inf Sci II-5 W 1:241–246.

Redweik R, Becker T (2015) Change Detection in CityGML Documents. In: 3D Geoinformation Science. Springer, pp 107–121

Stadler A, Kolbe TH (2007) Spatio-semantic coherence in the integration of 3D city models. In: Proceedings of the 5th International Symposium on Spatial Data Quality, Enschede.

Stefani C, De Luca L, Veron P, Florenzano M (2011) A Tool for the 3D Spatio-Temporal Structuring of Historic Building Reconstructions. Digit Media Its Appl Cult Herit 153–168.

Stefani C, De Luca L, Véron P, Florenzano M (2008) Reasoning about space-time changes: an approach for modeling the temporal dimension in architectural heritage. In: Proceedings of the IADIS International Conference.

Stefani C, De Luca L, Véron P, Florenzano M (2010) Time indeterminacy and spatio-temporal building transformations: an approach for architectural heritage understanding. Int J Interact Des Manuf IJIDeM 4:61–74.

Vesperman J (2006) Essential CVS. O'Reilly Media, Inc.

W3C Recommendation (2011) XML Path Language (XPath) 2.0 (Second Edition). http://www.w3.org/TR/xpath20/. Accessed 26 Feb 2015

Zada AJ, Tizani W, Oti AH (2014) Building Information Modelling (BIM)—Versioning for Collaborative Design. In: Computing in Civil and Building Engineering (2014). ASCE, pp 512–519

(2015a) Git User's Manual. https://www.kernel.org/pub/software/scm/git/docs/user-manual.html. Accessed 9 Mar 2015

(2015b) Mercurial SCM. http://mercurial.selenic.com/. Accessed 9 Mar 2015

(2015c) Apache Subversion. http://subversion.apache.org/. Accessed 9 Mar 2015