3D-USE  
Temporalité et gestion de versions

Documentation

-

Jérémy Edert

Octobre 2015

# I - Intro

Ce document présente les modifications apportées à 3D-USE dans le cadre de mon stage sur la temporalité.

L’objectif de ce travail était de réaliser une implémentation du modèle proposé dans l’article "*Managing versions and history within semantically enriched 3D city models*" de K. Chaturvedi. Cette proposition consiste à modifier le CityGML pour mieux prendre en compte la dimension temporelle des modèles urbains.

La réalisation de ce travail passait par 2 phases : la création d’une ADE pour CityGML (sous forme de fichier XSD), et l’implémentation dans 3D-USE d’outils nécessaires à l’utilisation de ce nouveau modèle. Ce document a pour objet les travaux effectués dans 3D-USE. La création de l’ADE ainsi que les modifications apportées au modèle proposé sont détaillées à part.

Le travail réalisé peut se diviser en 3 parties principales :

* Amélioration de la gestion des dates déjà implémentée
* Gestion des xLink tels qu’utilisés dans la proposition de l’article
* Ajout de la notion de versions au modèle implémenté dans 3D-USE

# II - Gestion des dates

Ce travail se base en grande partie sur les réalisations de Maxime Morel pour l’implémentation de la temporalité à l’aide de « tags » et « states ».

L’implémentation déjà existante permettait d’afficher un curseur temporel sur l’interface de 3D-USE, et d’afficher ou non les bâtiments en fonction de la date du curseur et des dates de construction et de démolition des bâtiments. Bien que cette fonctionnalité soit principalement pensée pour aller de pair avec les « tags » et les « states », elle marche pour n’importe quel Building muni des attributs « *yearOfConstruction* » et « *yearOfDemolition* », définis dans la classe CityGML *Building*.

Les fonctions liées à cette fonctionnalité ont été modifiées pour prendre également en compte les attributs de date « *creationDate* » et « *terminationDate* » définis dans la classe CityGML *AbstractCityObject*. De plus, les fonctions d’export du modèle vers un fichier CityGML ont été légèrement modifiées afin d’obtenir un meilleur export de ces 4 attributs de date.

D’autre part, dans le cadre du projet Alaric, des boîtes de dialogue afin de pouvoir rajouter les informations de dates aux bâtiments ont été ajoutées dans l’interface de 3D-USE.

Grace au travail effectué dans cette partie, n’importe quel type d’objet muni des dates *creationDate* et *terminationDate* peut être affiché en fonction du curseur temporel. **Ceci est également valable lors d’utilisation de xLinks ou de Versions** (qui seront détaillés plus tard).

## II.a - Ajout de creationDate et terminationDate

Les éléments creationDate et terminationDate peuvent se trouver dans tous les éléments de type CityObject dans un fichier CityGML. Lors du parsing du fichier, ils sont ajoutés à l’instance du CityObject en tant que « attributs », mais n’étaient pas utilisés par la suite dans l’implémentation précédente.

Exemple d’un fichier CityGML avec *creationDate* et *terminationDate* :

|  |
| --- |
| <cityObjectMember>  <bldg:Building gml:id="bldg\_bldg1-0">  <core:creationDate>2000-01-01</core:creationDate>  <core:terminationDate>2009-12-31</core:terminationDate>  <bldg:boundedBy>  < !—geometries (WallSurfaces, etc.) ici -->  </bldg:boundedBy>  </bldg:Building>  </cityObjectMember> |

Les fonctions suivantes ont été modifiées afin de faire suivre aux attributs *creationDate* et *terminationDate* le même workflow que *yearOfConstruction* et *yearOfDemolition* :

* **Fonction ReaderOsgCityGML::createCityObject()**

Cette méthode est appelée lors de la création de la structure de données OSG et crée l’objet OSG correspondant à un CityObject.

Cette méthode a été modifiée afin de récupérer les attributs *creationDate* et *terminationDate* dans la liste d’attributs du CityObject et de les ajouter en tant que UserValue de l’objet OSG (comme c’est le cas pour *yearOfConstruction* et *yearOfDemolition*).

* **Note importante : Je n’ai pas trouvé de moyen d’insérer de UserValue ou autre élément dans le modèle OSG qui ne soit pas un *integer*. J’ai donc du transformer les dates *creationDate* et *terminationDate* qui sont des *string* au format yyyy-MM-dd en entier. La méthode pour cela n’est pas très orthodoxe et pourrait être améliorée : les caractères 5 et 8 de la string (les tirets) sont retirés afin d’obtenir une *string* au format yyyyMMdd qui peut ensuite être transformée en *integer* facilement. *yearOfConstruction* et *yearOfDemolition* étant juste des années (des strings au format yyyy), ce problème ne se posait pas avant.**
* **Fonction OsgScene::setDateRec()**

C’est cette méthode qui gère l’affichage ou non des objets en fonction de leur date. Elle est appelée par exemple lors d’un mouvement du curseur temporel, et explore l’arbre OSG récursivement.

Cette méthode a été modifiée afin de récupérer les UserValues *creationDate* et *terminationDate* dans l’objet OSG, et si elles sont présentes, les comparer à la date actuelle du curseur temporel et déterminer si les objets doivent être affichés ou non.

## II.b Modification de l’export CityGML

Cette partie introduit les modifications apportées aux fonctions d’export vers un fichier CityGML (menu File > Export citygml)

Dans l’implémentation précédente, toutes les données présentes dans la liste d’attributs d’un objet étaient exportées sous forme d’attributs génériques. C’était donc le cas pour les dates *yearOfConstruction*, *yearOfDemolition*, *creationDate* et *terminationDate*.

Par exemple, si on importait un fichier CityGML puis qu’on l’exportait immédiatement, une balise

|  |
| --- |
| <bldg:yearOfConstruction>2000</bldg:yearOfConstruction> |

devenait

|  |
| --- |
| <gen:stringAttribute name="yearOfConstruction">  <gen:value>2000</gen:value>  </gen:stringAttribute> |

Il est à noter que cela ne concerne probablement pas que les dates et que c’est le cas pour toutes les données ajoutées qui sont dans la liste d’attributs de l’objet, mais ces cas n’ont pas été gérés.

Méthodes modifiées :

* **Fonction ExporterCityGML::exportCityObjetGenericXml()**

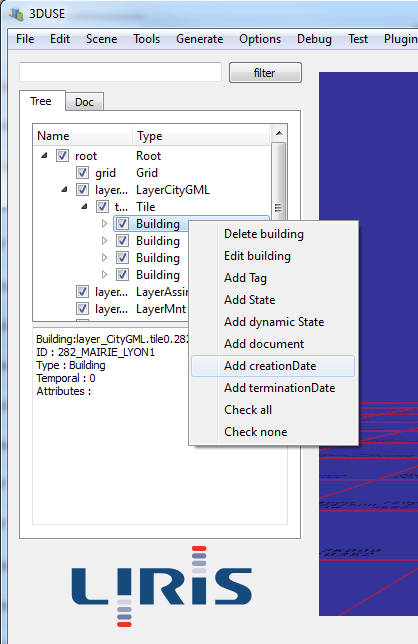
Cette méthode est appelée lors de l’export vers le format CityGML, lorsque l’objet actuellement en cours d’export est un CityObject. Cette méthode a été modifiée lors du parcours de la liste d’attribut du CityObject, et prend en compte le cas spécifique où l’attribut est *yearOfConstruction*, *yearOfDemolition*, *creationDate* ou *terminationDate* et réalise l’export sous la forme adaptée.

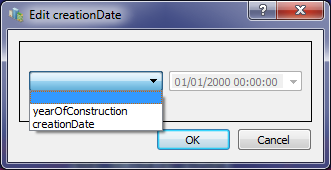
## II.c Boites de dialogue « add creationDate » et « add terminationDate »

Deux boîtes de dialogues ont été rajoutées à 3D-USE. L’une d’entre elle permet d’ajouter ou modifier la date *yearOfConstruction* OU *creationDate* à un objet, et l’autre permet d’ajouter ou modifier la date *yearOfDemolition* OU *terminationDate* à un objet.

* **Note :** la possibilité de supprimer ces dates n’est pas implémentée

Ces boites de dialogue sont accessibles en effectuant un clic droit sur un objet dans la tree view





Ces boites de dialogue sont assez simples, une droplist permet de choisir si on veut ajouter une *creationDate* ou une *yearOfConstruction* (ou *terminationDate* et *yearOfDemolition*), et le champ de date sur la droite permet de rentrer la date désirée.

Les dates sont rajoutées à la fois à la structure de données OSG et à celle issue de libcitygml. Le fichier CityGML d’origine n’est pas modifié, mais si un nouveau fichier est exporté (par le menu File > Export citygml) les nouvelles dates seront insérées dans le fichier exporté.

L’ajout de dates peut être effectué sur tous les objets d’une *Tile* en effectuant un clic droit sur une *Tile* dans la tree view et accédant aux boites de dialogue d’ici.

Classes ajoutées à 3D-USE pour cette partie :

* **Classe DialogYearOfConst**
* **Classe DialogYearOfDemol**

La tree view a également été modifiée pour rajouter l’accès aux boites de dialogue dans le menu contextuel.

# III. Gestion de xLink et xPath

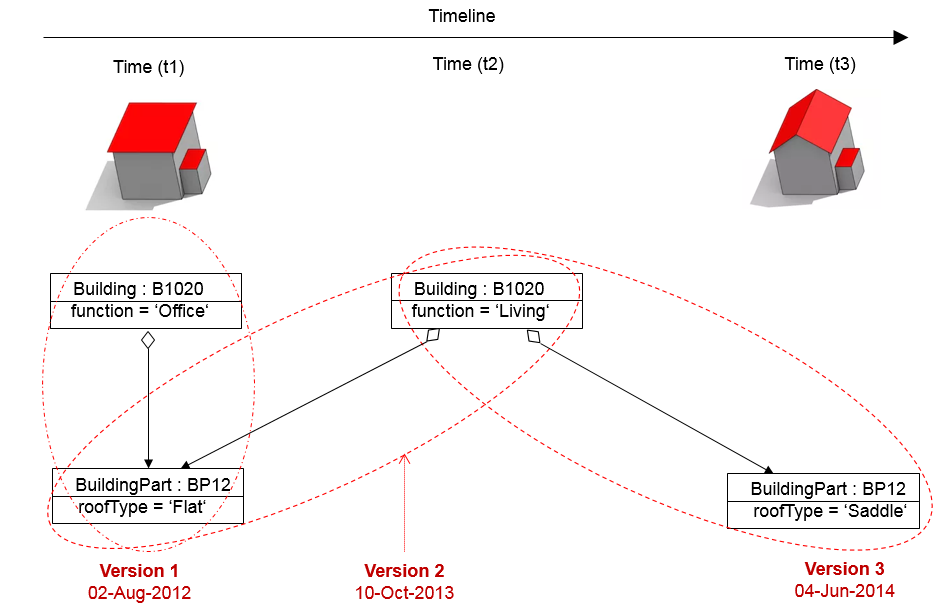
Bien que cet aspect ne soit pas directement visible dans le schéma XML de l’ADE, le modèle de l’article propose l’utilisation de xLink associé à xPath afin de relier certains objets entre eux au sein du fichier CityGML. Outre des liens « classiques » vers un objet en particulier, les auteurs proposent l’utilisation de requêtes xPath pour créer des liens vers toute une collection de résultats, par exemple pour référencer toutes les « versions » possible d’un objet.

La lecture des xLink et des requêtes xPath n’est pas assurée par la libcitygml. Il a donc fallu en implémenter un support.

**Cependant, la gestion de xLink et de xPath dans un parseur SAX (comme c’est le cas de libcitygml) est particulièrement complexe à implémenter et seul un support minimal a été réalisé ici.**

## III.a Cas d’utilisation traité

Le cas traité se base sur l’exemple présent dans la publication :



|  |
| --- |
| <cityObjectMember>  <Building gml:id=**"B1020\_t1"**>  <identifier>**B1020**</identifier>  <consistsOfBuildingPart>  <BuildingPart xlink:href=**"//identifier[text()='BP12']"**/>  </consistsOfBuildingPart>  <creationDate>**2012-08-02**</creationDate>  <terminationDate>**2013-10-09**</terminationDate>  <function>**Office**</function>  </Building>  </cityObjectMember>  <cityObjectMember>  <Building gml:id=**"B1020\_t2"**>  <identifier>**B1020**</identifier>  <consistsOfBuildingPart>  <BuildingPart xlink:href=**"//identifier[text()='BP12']"**/>  </consistsOfBuildingPart>  <creationDate>**2013-10-09**</creationDate>  <function>**Living**</function>  </Building>  </cityObjectMember>  <cityObjectMember>  <BuildingPart gml:id=**"BP12\_t1"**>  <identifier>**BP12**</identifier>  <creationDate>**2012-08-02**</creationDate>  <terminationDate>**2014-06-03**</terminationDate>  <roofType>**Flat**</roofType>  </BuildingPart>  </cityObjectMember>  <cityObjectMember>  <BuildingPart gml:id=**"BP12\_t3"**>  <identifier>**BP12**</identifier>  <creationDate>**2014-06-03**</creationDate>  <roofType>**Saddle**</roofType>  </BuildingPart>  </cityObjectMember> |

Comme on peut le voir sur cet exemple, les différents CityObjects peuvent correspondre à des instances différentes d’un même objet qui a évolué au cours du temps. Les CityObjects sont différents et possèdent donc des gml:id différents, mais ils possèdent un élément <identifier> afin d’indiquer qu’ils sont différentes versions d’un objet réel.

Dans cet exemple, le(s) bâtiment(s) identifiés par B1020 possèdent un membre correspondant aux BuildingParts BP12\_t1 et BP12\_t3. Plutôt que de créer 3 instances de bâtiment correspondant aux 3 associations possibles de B1020 et BP12, on utilise un élément <BuildingPart xlink:href=**"//identifier[text()='BP12']"**/> utilisant une requête xLink/xPath pour pointer vers tous les éléments possédant un <identifier> BP12. A la charge ensuite de l’application de gérer quelle instance de BP12 utiliser.

* **Note importante :** la requête xPath utilisée dans l’exemple (**//identifier[text()='BP12']**) ne donnerait pas exactement le résultat voulu. En effet, si on exécutait cette requête, on obtiendrait un résultat de ce type :

|  |
| --- |
| <identifier>**BP12**</identifier>  <identifier>**BP12**</identifier> |

Alors qu’on voudrait un résultat de ce type :

|  |
| --- |
| <BuildingPart gml:id=**"BP12\_t1"**>  <identifier>**BP12**</identifier>  <creationDate>**2012-08-02**</creationDate>  <terminationDate>**2014-06-03**</terminationDate>  <roofType>**Flat**</roofType>  </BuildingPart>  <BuildingPart gml:id=**"BP12\_t3"**>  <identifier>**BP12**</identifier>  <creationDate>**2014-06-03**</creationDate>  <roofType>**Saddle**</roofType>  </BuildingPart> |

Une requête correcte pour obtenir ce résultat serait par exemple «  **//identifier[text()='BP12']/..**»(noter la présence de « /.. » à la fin de la requête) **et c’est la seule syntaxe qui est supportée dans cette partie du travail** (Une autre syntaxe est utilisée dans le travail sur les Versions). D’autres possibilités pour obtenir le même résultat existent, telles que **//\*[identifier='BP12']** par exemple, mais elles ne sont pas supportées par le travail actuel.

## III.b Réalisation

Pendant le parsing :

* si on se trouve dans un élément CityObject (Building, etc.) mais qu’au lieu d’avoir un attribut gml:id, on a un attribut xlink:href, on crée une instance de citygml::CityObject marqué comme étant un xLink et on conserve la chaine de caractères correspondant au xLink dans cet objet
* si on se trouve dans un élément identifier, on place le CityObject actuel au sein d’un map qui répertorie les CityObjects en fonction de leur identifier

A la fin du parsing, on parcourt le citygml::CityModel créé.

* Si on tombe sur un CityObject marqué comme étant un xLink, on récupère la chaine de caractères de la requête.
* Si celle-ci est de la forme «  **//identifier[text()='XXXXXXX']/..**», on recherche dans la map tous les CityObjects associés à cet identifier.
* Ces CityObjects sont ajoutés aux membres du CityObject marqué comme xLink au sein d’un vecteur appelé xLinkTargets

Pendant la création de la structure OSG ou de la TreeView, les xLinkTargets sont considérés comme des enfants du CityObject xLink.

Des modifications ont notamment été apportées aux classes et méthodes suivantes :

* **Classe CityGMLHandler**

Cette classe étant celle où le parsing a lieu, elle a été modifiée afin de suivre l’algorithme indiqué plus haut. On notera entre autres la méthode endDocument() d’un parseur SAX qui n’était pas définie jusque-là dans libcitygml (car il n’y en avait pas besoin) et qui a donc été rajoutée pour effectuer les liens entre les liens et leurs cibles.

* **Classe citygml::Object**

A cette classe a été ajoutée une énum décrivant si l’objet est un xLink, une cible, ou un objet normal. On a aussi rajouté une liste d’Objects qui relie l’objet xLink à ses cibles.

* **Méthode CityObject::getNode()**
* **Méthode OsgScene::buildCityObject()**
* **Méthode TreeView::addCityObject()**

Ces trois méthodes ont été modifiées pour rendre compatible la nouvelle structure de données avec le reste de l’application. Elles font en sorte que les xLinkTargets d’un objet soient utilisés de la même manière que des descendants habituels d’un objet.

* **Méthode ExporterCityGML::exportCityObjetGenericXml()**

Cette méthode qui concerne l’export vers un fichier CityGML a été modifiée afin de gérer l’export des xLink (qui auraient été ignorés sinon).

## III.c Ajout de xLink depuis 3D-USE (tentative)

Une tentative d’insertion de xLink dans le modèle a été réalisée, mais propose des fonctionnalités très limitées. Cette partie du travail est fournie pour une éventuelle base de travail pour la suite, mais ne devrait probablement pas être utilisée telle quelle.

Une boite de dialogue a été créée pour ajouter un xLink dans le modèle. Elle permet d’indiquer quelle est la requête (la chaine de caractères désignée par xlink:href dans le XML) et d’importer un CityObject depuis un fichier CityGML supplémentaire vers le modèle actuel. Aucune vérification n’est faite de la cohérence de la requête xLink avec les objets insérés ou non.

La boite de dialogue est gérée dans la nouvelle classe dialogLink. La ligne de code permettant d’afficher cette boite de dialogue a été commentée, rendant cette fonctionnalité inaccessible pour éviter de faire des dégats.

Pour accéder à cette boite de dialogue, décommenter la ligne

//m\_tree->addAction(m\_actionAddLink); dans la méthode TreeView::slotSelectNode(). La boite de dialogue est alors accessible en effectuant un clic droit sur un objet dans la tree view grâce à l’option « add link to a new object ».

# IV. Versions, Transitions, etc.

Le modèle proposé et l’ADE qui en découle proposent un certain nombre de nouveaux concepts à introduire dans la modèle. Il s’agit notamment :

* Des attributs de date *validFrom* et *validTo* dans la classe CityObject
* De la classe Version
* De la classe VersionTransition
* De la classe Transaction
* Des énumérations TransitionValue et TransactionValue

Tous ces éléments dépendent de l’ADE et sont donc munis d’un namespace XML spécifique. On a donc écrit les fonctions de parsing de ces éléments dans une nouvelle classe TemporalHandler.

Dans les grandes lignes, une Version est une collection de CityObjects (correspondant à une version particulière du modèle). Une VersionTransition est un lien entre 2 versions consécutives du modèle et peut porter une liste de Transactions qui sont les opérations individuelles d’une version à une autre. Le tout peut être organisé de façon totalement linéaire, ou bien sous forme d’arbre avec des embranchements. (Pour plus de détail, consulter l’article)

* **Note 1 :** En l’absence d’exemple de l’utilisation de Versions au sein d’un fichier CityGML, on a créé de toutes pièces un fichier CityGML d’exemple en tentant de suivre les principes définis dans la publication. La structure du fichier est telle que tous les CityObjects membres du CityModel sont définis d’une part, et la liste des Versions et des VersionTransitions d’autre part. Il n’y a pas d’élément Version au milieu d’un Building, par exemple.

Utilisation de xLinks au sein des versions :

Chaque Version possède une liste d’objets. Chaque VersionTransition possède 2 membres Version. Ces deux relations peuvent être représentées par des xLinks et des requêtes xPath. Contrairement au cas cité dans la partie sur les xLinks, ici on ne cherche pas un identifier, mais bel et bien une instance d’objet particulier.

La seule requête xPath supportée dans le cadre des versionMembers de Version et des deux membres de VersionTransition est : «  **//\*[@id='XXXXXXX']**» où le paramètre est le *gml:id* de la cible. (voir le fichier exemple exemple\_versions.gml)

* **Note 2 :** Le namespace XML que j’ai utilisé pour l’ADE pendant mon travail était « temp », celui-ci est susceptible de changer. Il faudra alors songer à mettre à jour le code de la classe TemporalHandler avec le nouveau namespace.

Les modifications suivantes ont été apportées au modèle :

* **Classe temporal::Version**
* **Classe temporal::VersionTransition**
* **Classe temporal::Transaction**

Les classes Version, VersionTransition et Transaction ainsi que les énumérations TransactionValue et TransitionValue ont été créées dans un nouveau namespace C++ « temporal »

* **Classe citygml::CityModel**

La classe CityModel a été modifiée afin d’accueillir une liste de Versions et une liste de VersionTransitions

* **Classe TemporalHandler**

Cette classe est une classe dérivée d’ADEHandler et permet de réaliser le parsing des éléments XML précédés du namespace « temp ».

Les dates *validFrom* et *validTo*, quant à elles, sont rajoutées aux objets en mémoire par le biais de leur liste d’attributs, comme cela est fait pour les dates *creationDate*, *terminationDate*, etc. (A noter que les dates validFrom et validTo ne sont **pas** utilisées pour gérer l’affichage des objets via le curseur temporel.)

A la fin du parsing, les éléments suivant sont stockés dans le CityModel :

* Une liste de Versions. Chaque version peut posséder des tags et une liste de pointeurs vers des CityObjects
* Une liste de VersionTransitions. Chaque transition possède deux pointeurs vers des objets Version (*from* et *to*) et une collection de Transactions.
* Dans le cadre des jeux d’essais que j’ai créé, les Transactions ont été ignorées, mais elles devraient normalement être construites de façon cohérente. Une Transaction appartient à une VersionTransition particulière et possède ou non des pointeurs vers deux CityObjects indiquant « l’ancienne » version et la « nouvelle ».

Il n’existe pour le moment pas d’élément d’interface permettant de visualiser les versions. La fonction « Test4 » du menu de test de 3D-USE a été utilisée pour générer une trace dans la console des deux listes de Versions et de VersionTransitions. Cette fonction affiche l’id de chaque élément des listes et l’id de ses membres.

# V. Workspaces

Le concept de Workspace n’est pas clair dans l’article, mais il a été interprété comme suit : un workspace (ou scénario) est une suite de Versions comprises entre un fork et un merge (assimilable à une « branche » de Git).

Techniquement, la fin d’un workspace pourrait peut-être se définir par l’absence de transition qui ne soit PAS « fork », « merge », « planned » ou « realized » ?

La notion de Workspace n’existe pas dans le modèle CityGML proposé par l’article, ni dans l’ADE. Une solution de repli pour réaliser l’implémentation de ce concept a été d’utiliser les attributs « tags » de la classe Version.

### V.a Workspaces nommés : utilisation de tags

Une Version peut posséder autant de tags définis par l’utilisateur que l’on veut. Nous proposons donc que chaque Version devant faire partie d’un Workspace soit munie d’un tag « WORKSPACE=XXXX », où XXXX est le nom du Workspace. **Il est important pour cela que le nom du Workspace soit unique.**

Comme pour les Versions, il n’existe pas d’interface pour visualiser les Workspaces, mais la fonction « Test4 » de 3D-USE a été utilisée pour générer une trace de la liste de workspaces du CityModel.

Modifications du modèle :

* **Classe temporal::Workspace**

Cette classe a été rajoutée au modèle pour implémenter la notion de Workspace. Elle contient le nom du Workspace et une liste de Versions.

* **Classe citygml::CityModel**

La classe CityModel a été modifiée afin d’accueillir une liste de Workspaces

### V.b Méthode automatique

Nous avons réfléchi également à une méthode pour retrouver les workspaces de façon automatique sans avoir à utiliser les « tags » et donc sans nommer les workspaces, mais nous ne l’avons pas implémentée faute de temps et en raison de sa possible complexité.

Cette méthode nécessiterait de modifier la classe Version afin de pouvoir accéder à toutes les VersionTransition qui sont liées à cette instance de Version (que ce soit par un *from* ou par un *to*). Par cette modification, on pourrait alors parcourir les Versions à la manière d’un arbre et retrouver les différentes branches. La présence d’une transition « fork » ou « merge » marquerait le début d’une nouvelle branche et donc d’un nouveau workspace, et en descendant dans la branche selon les transitions « historical succession » on propagerait l’appartenance des Versions à ce workspace.

Plusieurs cas particuliers seraient éventuellement à prendre en compte : la présence dans le document CityGML de plusieurs arbres indépendants, des arbres ascendants (plusieurs branches indépendantes à la base mais qui sont fusionnées à la fin), etc.