Aligner des données XML avec une ontologie avec 3M

Auteur(s) : Florian Hivert, Olivier Marlet,
Institution émettrice : consortium MASA
Contact mail : olivier.marlet@univ-tours.fr

I. Introduction

Mapping Memory Manager (3M) est une application gratuite en ligne développée par l'équipe ICS-FORTH (Héraklion, Grèce). 3M permet de faire l'appariement entre des données au format XML et des ontologies notamment le CIDOC-CRM et ses extensions, en vue de porter ces données sur le Web sémantique. Cet appariement (ou *mapping*) rend possible l'exportation des données dans des formats standards et adaptées au Web Sémantique comme le Turtle, le RDFS et le N-Triple.

Pour réaliser cet objectif, 3M utilise principalement deux langages :

- XPath qui permet d'explorer une arborescence XML pour retrouver, sélectionner et retourner les données (voir Partie 6) ;
- X3ML qui permet d'utiliser des modèles de triplets appelés (voir Partie 5) afin de structurer les données du fichier XML dans un fichier de sortie au format Turtle, RDFS ou N-Triple.

II. Accès à l'application en ligne

3M est gratuitement accessible en ligne : http://139.91.183.3/3M/

Pour permettre la sauvegarde des *mappings* et gérer les droits sur les *mappings* disponibles sur la plateforme, la création d'un compte est nécessaire.

La présentation de ce tutoriel se structure sur la présentation des différents onglets de l'application qui déterminent le processus de *mapping* et de transformation.

III. Principes généraux pour écrire des triplets

La structuration des *mappings* dans 3M s'appuie sur le principe de base du web sémantique et le fonctionnement d'un triplet RDF. Un triplet RDF est construit à l'aide de trois éléments :

Sujet - Prédicat - Objet

Le sujet est toujours une instance définie par une entité ou une classe ;

- Le prédicat ou propriété est le verbe du triplet, il exprime l'association entre le sujet et l'objet, cette association est défini par rapport à la classe du sujet (domaine) et de l'objet (co-domaine) ;
- L'objet est une instance similaire au sujet OU une donnée littérale typée.

```
 <URI Sujet (Domaine)> <URI Prédicat (Propriété)> <URI Objet (Co-Domaine)> ou "valeur"
```

La définition des classes se fait au début de l'écriture du triplet afin de respecter les règles de domaine et de co-domaine des ontologies, et d'attribuer une classe (entité de l'ontologie) à une instance (une donnée).

IV. Onglet INFO

« Info » regroupe les données, fichiers et les ontologies nécessaires à la réalisation du *mapping* :

- Le bloc « *General* » contient le fichier XML de et les différentes métadonnées de présentation du .
- Le bloc « *Source* » enregistre le fichier XSD à l'origine de la création des fichiers XML que l'on souhaite mapper

à l'aide de 3M.

- Le bloc « *Target* » reçoit toutes les ontologies utilisées dans notre (CIDOC-CRM, ses extensions et d'autres si besoin), les ontologies autres que le CIDOC nécessitent d'être importées via le fichier RDFS.

- Le bloc « *Namespaces* » permet d'enregistrer des (espace de nom = le contexte dans lequel les balises et leurs structures sont définies) qui ne sont pas issus d'ontologies mais nécessaires au des données. L'utilisateur peut ainsi raccourcir les liens vers le site web des données en déclarant un qui contient l'URL du site.

- Le bloc « *Mapping* » détient les métadonnées qui décrivent les institutions et personnes à l'origine du .

- Le bloc « *Sample data* » reçoit le fichier XML source, le fichier qui contient les X3ML nécessaires à la transformation des données, le fichier de sortie sous le nom de et le fichier Thésaurus qui permet d'importer une terminologie déjà établie pour l'exploiter.

V. Onglet MATCHING TABLE

« *Matching Table* » correspond à l'onglet pour effectuer le *mapping*. Lorsque l'utilisateur crée un *mapping* en cliquant sur le bouton « Map », un tableau va s'ouvrir où les premières manipulations seront de choisir un nœud source et de choisir la classe CIDOC-CRM à attribuer pour les données de ce nœud.

Exemple : le nœud source <Site> du fichier XML sera associé à la classe crm:E27_Site du CIDOC-CRM. L'interface permet ensuite d'ajouter soit des « *Link* », soit des propriétés supplémentaires sur le nœud source. Cette fonctionnalité s'avère utile seulement lorsque le nœud source possède soit des attributs soit des nœuds enfants. Le nœud <Site> possède l'attribut @id qui contient l'identifiant du site. L'utilisateur peut créer un « *Link* » avec l'attribut @id comme source et apparier l'attribut @id à la classe crm:E42_Identifier via la propriété crm:P48_has_preferred_identifier.

La fonctionnalité de « *Link* » ne sert pas que simplement à associer chaque nœud enfant/attribut à une propriété et une classe. L'outil permet en effet d'ajouter des « *Intermediate* » soit des propriétés/entités intermédiaires dans un cheminement. Cette fonctionnalité donne la possibilité de former plusieurs triplets dans un seul attribut/nœud en fonction des besoins de l'appariement.

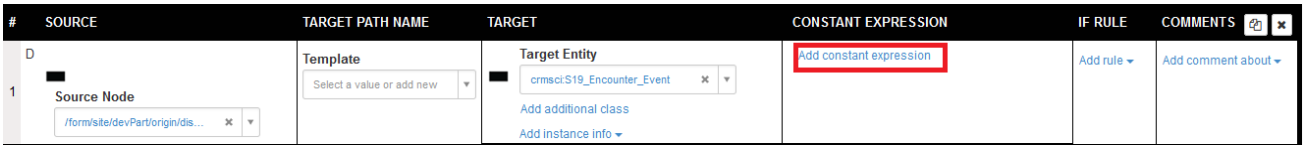
Exemple : pour ajouter la dimension temporelle d'un objet, le CIDOC-CRM (et plus particulièrement le modèle OpenArchaeo du consortium MASA sur lequel nous appuyons nos *mappings*) demande de d'abord lier l'entité avec un objet de la classe crm:E52_Time-Span à l'aide de la propriété crm:P4_has_time-span puis, de lier l'objet crm:E52_Time-Span avec un xsd#dateTime à l'aide des propriétés crm:P81b_end_of_the_end et crm:P82a_begin_of_the_begin. Pour réaliser cela dans 3M, l'utilisateur doit créer une première relation entre crm:E22_Man-Made_Feature (par exemple) et crm:E52_Time-Span avec la propriété crm:P4_has_time-span puis, ajouter la seconde relation entre crm:E52_Time-Span et xsd#dateTime, à l'aide des propriétés crm:P81b_end_of_the_end ou crm:P82a_begin_of_the_begin, avec un *intermediate*.

Ce qui donne :

On apparie ainsi dans un seul nœud enfant de la source <date> avec l'entité correspondante dans l'ontologie en explicitant toute l'arborescence des propriétés et classes nécessaires pour l'atteindre.

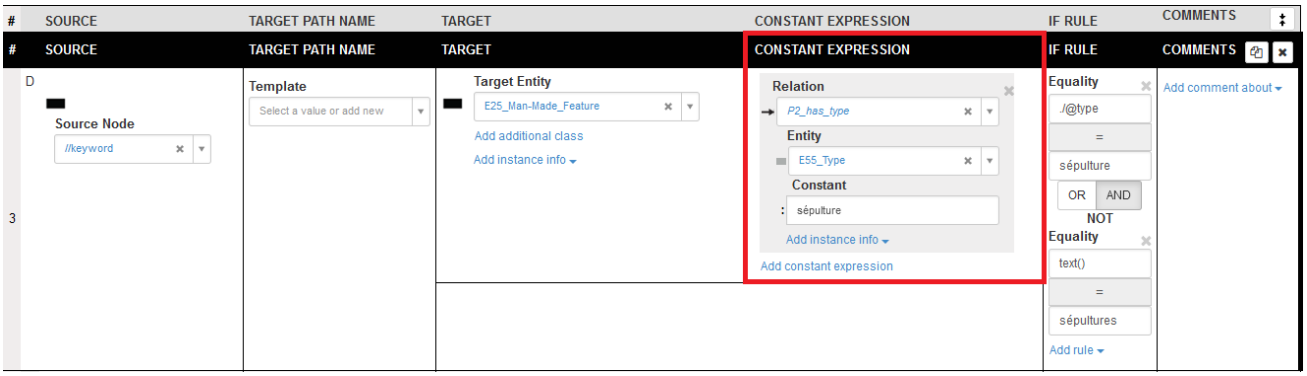
Dans l'exemple ci-dessus, crm:E52_Time-Span possède une indication supplémentaire écrite en jaune [time_discovery]. Le menu déroulant « *Add instance info* » ajoute cinq options pour caractériser les entités mobilisées dans le *mapping* :

- *is Constant* : Plutôt que de mobiliser une variable qui change en fonction de la position du nœud dans le document XML, cette option permet de donner une valeur constante (une valeur qui ne changera pas) à toutes les entités de la classe sélectionnée.
- *is same as (map)* : cette option offre deux possibilités :
- Indiquer que de multiples instances dans une table du *mapping* parlent de la même chose (par exemple si l'on utilise plusieurs fois E12_Production dans une même table de et qu'il s'agit du même objet répété plusieurs fois) : il faut alors donner un nom identique aux éléments identiques.
- Distinguer des *mapping* effectués sur un même nœud. Par exemple, si le contenu textuel d'un nœud contient plusieurs informations à mettre dans des classes CIDOC-CRM distinctes, *is same as (map)* permet de différencier les deux alignements sur le même nœud : il faut alors donner des noms distincts aux éléments à distinguer.
- *is same as (global)* : Même principe que l'option précédente mais celle-ci s'applique sur tous les et pas sur le seul en cours.
- *Description* : place une propriété rdfs:description avec du contenu par défaut dans les instances de la classe.
- *Language* : désigne la langue par défaut des instances de la classe qui sera proposée dans leur label.



L'option « *add constant expression* » permet d'ajouter un triplet dont le sujet sera le *mapping* dans lequel on se trouve (ici *.../discovery* instancié dans la classe crmsci:S19_Encounter_Event). Ce triplet sera décrit par une propriété (prédicat) et une classe (objet) dont la valeur sera fixée pour toutes les instances de cette classe.

Dans l'exemple du modèle de OpenArchaeo, cette fonctionnalité se révèle utile pour les instances de crm:E25_Man-Made_Feature où l'entité doit être distinguer en trois types : Fait, Sépulture ou Mur.



Ici, les instances de crm:E25_Man-Made_Feature qui concernent les sépultures ont reçu en *Constant Expression* la relation crm:P2_has_type vers une instance constante de la classe crm:E55_Type. Cette instance contiendra l'id Ark

(identifiant pérenne) du terme “sépulture” de PACTOLS (https://ark.frantiq.fr/ark:/26678/pcrt795b632nWw). Dans le cas présent, l’information n’existe pas dans le document XML et elle est ajoutée pour faire correspondre le mapping avec le modèle de données de OpenArchaeo.

L’exemple ci-dessus présente également « *IF RULE* » qui permet de mettre en place une structure conditionnelle pour appliquer certaines règles en fonction des situations.

De la même manière qu’un langage informatique, 3M propose d’utiliser dans son interface graphique le système de conditions de xPath. L’utilisateur peut alors préciser qu’un *mapping* s’effectuera seulement si la valeur d’un attribut est égale à la valeur indiquée. C’est le cas ici avec la condition, “Equality ./@type = sépulture”, soit si l’attribut type du nœud actuel est égal à sépulture alors le *mapping* pourra être effectué. La deuxième condition, “AND NOT Equality text() = sépultures”. “*AND*” ajoute une deuxième condition à vérifier pour que le *mapping* s’effectue, ici que le texte du nœud actuel ne soit pas égal à “sépultures”.

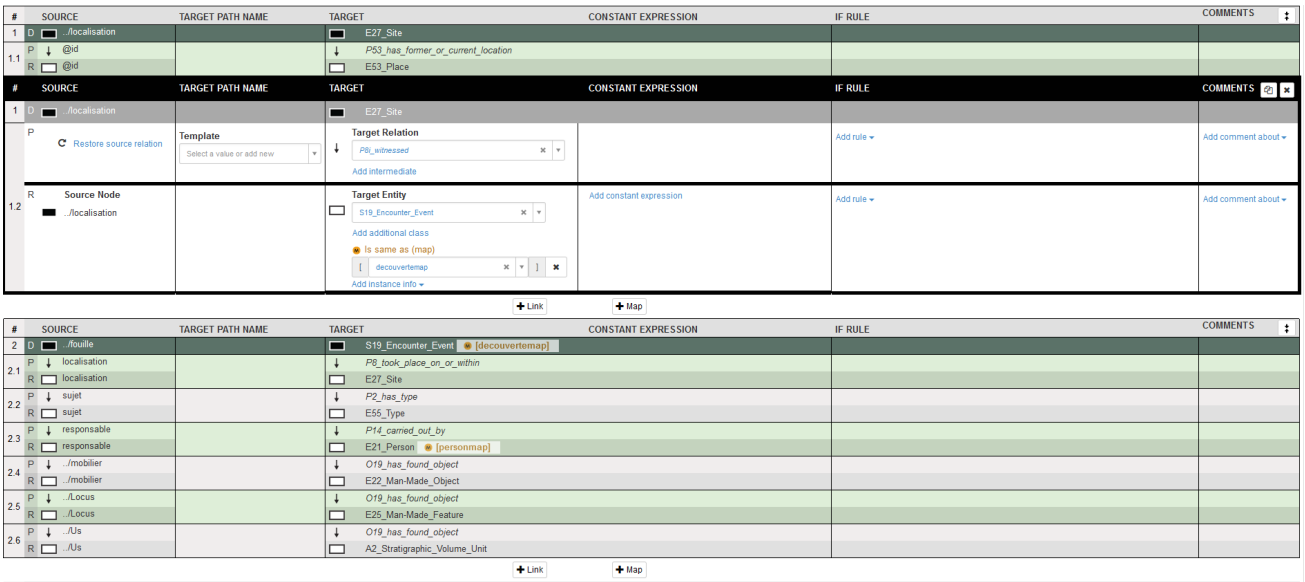
Parmi les options autres que *Equality* et *Inequality/NOT equality*, on retrouve :

- Existence, qui vérifie qu’un nœud, un attribut ou une contenu textuel existe dans ce nœud pour activer l’appariement ;
- *Nonexistence*, qui est l’inverse de *Existence*;
- *ExactMatch*, fonctionne uniquement sur skos:exactMatch ;
- *Narrower*, qui vérifie que le nœud actuel possède un nœud enfant particulier ;
- *Broader*, qui vérifie que le nœud actuel possède un nœud parent particulier.

Dans le cas où l’utilisateur ajoute plusieurs conditions, celui-ci peut décider avec *OR* ou *AND* si toutes les conditions demandent d’être toutes vraies ou seulement une pour que le *mapping* s’effectue.

La partie « *Comments* » sert à rédiger des commentaires dans le *mapping*.

Lorsque l’on veut récupérer une information extérieure aux nœuds enfants et aux attributs du nœud source, on peut utiliser le bouton « *set blank source relation* ». Toutefois, dans une nouvelle relation source, l’entité objet doit avoir une « *instance info* » « *is same as (map)* » ou « *is same as (global)* » pour être utilisable. Par conséquent, l’élément doit déjà faire l’objet d’un *mapping* spécifique pour pouvoir être ré-exploité dans d’autres *mappings*.



VI. Onglet GENERATORS

Une fois qu’ont été mises en place les règles d’appariement des nœuds avec le CIDOC-CRM, les triplets doivent être

créés en les peuplant avec les instances du document XML.

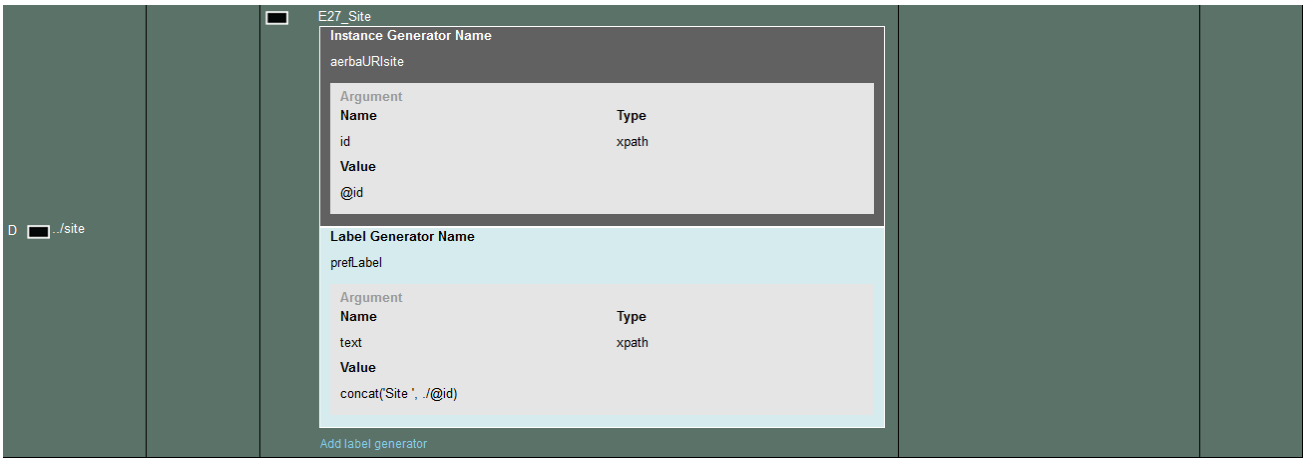
3M fonctionne avec X3ML qui exploite des “*Generators*” pour structurer les données récupérées à l’aide de xPath (voir manuel X3ML generator).

Depuis notre fichier *Generator Policy* enregistré dans l’onglet *Info*, il est possible d’ajouter des *generators* personnalisés, afin de proposer des URIs adaptées aux données.

Par exemple, dans le cas d’un *mapping*, il est possible de créer plusieurs *generators* afin de créer des URIs propres à chaque classe d’objets :

```
 <generator name="aerbaURIsite" prefix="aerba">
 <pattern>fiche.html?id={id}</pattern>
 </generator>
```

Le préfixe “aerba” correspond à l’adresse http de la base AERBA qui a été indiquée dans l’onglet “Info”. Cette adresse est complétée avec « fiche.html » à laquelle on ajoute un paramètre « id » dont la valeur sera la variable indiquée entre accolades, variable récupérée par xPath depuis l’onglet *Generators* de 3M.



Ci-dessus se trouve le *Generator* des entités crm:E27_Site du nœud “/site”. Deux *Generators* y sont présentés :

- Le *Instance Generator Name* qui génère l’URI de l’entité, comme montré au-dessus : ici le lien est créé à partir de l’identifiant (id) du Site.
- Le *Label Generator Name* qui donne le label à l’instance. Ici, le label apparaîtra sous la propriété *Preflabel*. Le *Generator* par défaut “*prefLabel*” permet de créer des labels associés à cette propriété. Toute autre *Generator* utilisera rdfs:label.

POINT IMPORTANT : Il n’est pas nécessaire de faire apparaître le triplet pour créer le label d’une entité dans le *mapping* . Les labels peuvent directement être créés dans l’onglet *Generator* en utilisant les Label *Generator Name*. Dans le cas de *mapping* vers OpenArchaeo, il est conseillé d’utiliser le *Generator* “prefLabel” pour toutes les labels sauf pour les instances de la classe E42_Identifier.

Pour récupérer les données du fichier XML et les intégrer dans l’URI de l’instance ou le label de l’instance, l’utilisateur doit spécifier l’emplacement de la valeur à l’aide de xPath.

Pour rappel, xPath est le langage de requête qui permet de localiser une portion d’un document XML (voir Partie 6).

Dans l’exemple ci-dessus, on se trouve dans le nœud /site. L’id que l’on souhaite récupérer est un attribut du nœud /site soit <site @id=’mon_id’>, pour récupérer @id, l’une des syntaxes possibles est celle présentée, soit simplement @id. L’autre possibilité, plus sûre, est d’écrire « ./@id », « ./ » étant l’équivalent de « depuis le nœud actuel, c’est-à-dire l’élément "site", se positionner sur l’attribut "id" ».

VII. Langage Xpath

Sélectionner des nœuds :

nom_du_nœud	Sélectionne tous les nœuds qui correspondent au nom
/	Sélectionne depuis le nœud racine
//	Sélectionne tous les nœuds dans le document depuis le nœud actuel qui correspondent à la sélection peu importe où les nœuds se trouvent
.	Sélectionne dans le nœud actuel
..	Sélectionne dans le nœud parent du nœud actuel
@	Sélectionne un attribut.

Exemple :

site	Sélectionne tous les nœuds avec le nom site
/site	Sélectionne le nœud racine site
site/idPart	Sélectionne tous les éléments idPart qui sont enfant du nœud site
//site	Sélectionne tous les éléments site peu importe où ils se trouvent dans le document
site//discovery	Sélectionne tous les éléments discovery qui sont enfant d'un nœud site peu importe où ils se trouvent en dessous de l'élément site
//@id	Sélectionne tous les attributs qui s'appellent @id

Prédicat :

/infoType/keyword[1]	Sélectionne le premier keyword enfant de infoType
/infoType/keyword[last()]	Sélectionne le dernier keyword enfant de infoType
/infoType/keyword[last()-1]	Sélectionne l'avant-dernier keyword enfant de infoType
/infoType/keyword[position() < 3]	Sélectionne les 2 premiers keyword enfant de infoType
//keyword[@type]	Sélectionne tous les keywords qui possèdent un attribut @type peu importe où ils se trouvent
//keyword[@type = 'mur']	Sélectionne tous les keywords qui possèdent un attribut @type égal à 'mur' peu importe où ils se trouvent
./text()	Sélectionne le texte du nœud où l'on se trouve
../@idModel	Imaginons que l'on soit dans le nœud @type, on retourne dans <keyword>, soit le nœud parent de @type, et on choisit cette fois-ci l'attribut @idModel.

Sélectionner des nœuds inconnus :

*	S'aligne sur n'importe quels nœuds
@*	S'aligne sur n'importe quels attributs

node()	S’aligne sur n’importe quels nœuds peu importe son type

Axes :

Les axes permettent de récupérer des données dans le cas d’une relation avec le nœud où l’on se trouve, plusieurs commandes retrouvent différents nœuds en fonction de leur emplacement dans le fichier par rapport à un nœud source :

ancestor	Sélectionne tous les ancêtres (parent, grands-parents, arrière grands-parents, etc..) du nœud actuel
ancestor-or-self	Sélectionne tous les ancêtres (parents, grands-parents, arrières grands-parents, etc..) du nœud actuel et le nœud actuel
attribute	Sélectionne tous les attributs du nœud actuel
child	Sélectionne tous les nœuds enfants du nœud actuel (seulement première descendance)
descendant	Sélectionne tous les descendants (enfants, petits-enfants, arrières petits-enfants, etc..) du nœud actuel
following	Sélectionne l’intégralité du document après la fin du nœud actuel (après la balise </nœud>)
following-sibling	Sélectionne tous les nœuds similaires au nœud actuel apparaissant après ce dernier dans le document.
namespace	Sélectionne tous les namespaces du nœud actuel
parent	Sélectionne le nœud parent du nœud actuel
preceding	Sélectionne l’intégralité du document avant le début du nœud actuel (avant la balise </nœud>
preceding-sibling	Sélectionne tous les nœuds similaires au nœud actuel apparaissant avant ce dernier dans le document.
self	Sélectionne le nœud actuel

Quelques exemples d’utilisations d’axes XPath :

ancestor::fouille	Sélectionne tous les ancêtres fouille du nœud actuel
child::text()	Sélectionne tous les nœuds textuels enfant du nœud actuel
parent::*/child::keyword	Sélectionne tous les nœuds enfants keyword des nœuds parents du nœud actuel
attribute::*	Sélectionne tous les attributs du nœud actuel

En fonction de l’organisation des données du fichier XML, les axes peuvent servir à recréer facilement les liens O19i entre un objet/fait/US ou une structure avec une découverte ou même de retrouver les US et les faits auxquels sont liés un mobilier. Avec l’axe child, on peut également récupérer tous les nœuds similaires liés à un nœud parent. La connaissance des axes XPath s’avère indispensable à utiliser dans les conditions des *Matching Tables* (voir Partie 4) ou pour récupérer des données dans les *Generators* (voir Partie 5) dans certains cas.

VIII. Quelques conseils…

La fonction concat() peut toujours être utilisée pour lier deux valeurs récupérées de xPath.

Par exemple, concat(./text(), ‘ ‘, ./@idModel) lorsque l’on se trouve dans un nœud <keyword> récupérera le mot-clé et son ID et l’assemblera pour donner : « Mot-clé 010101 ».

Lorsqu'une URI ou un lien existe déjà dans le document et que le *Generator* n'est pas obligé de la construire comme pour l'exemple du Site ci-dessus, il faudra TOUJOURS choisir URIorUUID en *Instance Generator Name*. Autrement, l'URI/le lien sera considéré comme du texte.

Il est préférable d'utiliser le *label generator* « prefLabel » pour un objet que de stipuler le triplet « prefLabel -> rdfs:litteral » dans l'onglet *Matching Table*.

Certaines entités du modèle OpenArchaeo restent uniquement nécessaires pour le fonctionnement de celui-ci. Par exemple, il sera très peu probable que les objets de la classe E65_Creation aient réellement une valeur tirée des données sources. On préférera alors créer une fausse URI ne contenant aucune donnée (pas de Label), comme "urisite:idobjet/creation" pour bien indiquer le chemin jusqu'à une entité E52_Time-Span. La même réflexion s'applique pour les objets E42_Identifier où on peut inventer une URI "urisite:idobjet/identifiant" et E5_Event avec "urisite:idobjet/datation".

Lors du *mapping* entre deux objets dans l'onglet *Matching Table*, il est important de le faire pour la propriété ET la propriété inverse. Si l'on crée pour les Unités Stratigraphiques le *mapping* : `A2_Stratigraphic_Volume_Unit > P70i_is_documented_in > E31_Document` , alors il faut aussi mettre en place oiur les Documents un *mapping* `E31_Document > P70_documents > A2_Stratigraphic_Volume_Unit.`