Computers in Industry xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

An ontology change management approach for facility management

Perrine Pittet *, Christophe Cruz, Christophe Nicolle

Le2i Laboratory, UMR CNRS 6306, University of Burgundy, Dijon, France

ARTICLE INFO

Article history:
Received 1 July 2013
Received in revised form 30 May 2014
Accepted 11 July 2014
Available online xxx

Keywords:
Ontology change management
Ontology evolution
Building information modeling
Facility management
$\mathcal{SHOIN}(\mathcal{D})$
OWL DL

ABSTRACT

Facility management (FM) or technical property management is an approach to operate, maintain, improve and adapt buildings and infrastructures of organizations. A FM project requires the cooperation of many actors from different domains so it has to be automated in a constrained collaborative environment. This paper proposes a new approach for ontology change management applied on facility management of such projects. The industrial challenge is, firstly, to ensure consistency of a FM project knowledge from the construction phase to the technical property management phase (after delivery). Secondly, it has to provide to each actor of the project a personal up-to-date "view" of the building knowledge related to its business profile and allow its evolution. The scientific approach, called OntoVersionGraph, is a change management methodology for managing ontology life cycle including ontology evolution and versioning features, in conjunction with contextual view modeling. Its contribution is the impact management of changes between the ontology and its different views.

1. Introduction

Facility management (FM) or technical property management is an approach to operate, maintain, improve and adapt buildings and infrastructures of organizations [24]. In practice, it can cover a wide range of services including property management, financial management of the building, management of human resources related to the building, health and associated risks of the building, maintenance (domestic services such as cleaning, security...) and supplies management. A FM project requires the cooperation of many actors from different domains such as architects, civil engineer, plumbers... During such projects, teams set up many business processes independently from other teams in other domains.

To achieve facility management of such projects, the facility manager needs powerful tools to organize the knowledge produced by each of the actors during the building life cycle. The creation of an information system dedicated to the use of facility management requires enormous phases of knowledge modeling. In addition, the knowledge of the building changes during its life cycle in response of the different types of information generated by the actors of the project linked to the MOP Law ("loi de Maîtrise d'Ouvrage Publique" defined by the French government, stating authority control and project management relationships for the public market since 1985). Consequently modeling the facility management of a building during its life cycle addresses the problem of heterogeneity of information exchanged between the actors. It demonstrates the need to homogenize the representation of these exchanges with the building knowledge all along its lifecycle.

Modeling building information during its life cycle is the aim of the BIM (Building Information Model) technology. It aims at facilitating integration, interoperability and collaboration in the building industry. BIM can be defined as the process of generating, storing, managing, exchanging, and sharing building information in an interoperable and reusable way. To enable building information sharing, Industry Foundation Classes (IFCs) are the standard that specifies object representations for FM projects (Vanlande et al., 2003); they include object specifications, or classes, and provide a useful structure for data sharing among applications. A BIM system is a tool that enables users to integrate and reuse IFC building information and domain knowledge throughout the lifecycle of a building. However, the main feature of a BIM system described by an IFC file is the 3D modeling of a building with data management, data sharing and data exchange during the building life cycle. To resolve the heterogeneity issues of facility management and ensure a complete interoperability between actors of the project, a semantic layer is missing.

The semantic heterogeneity issue has been addressed by the Semantic Web, which marks a shift from publishing data in human

* Corresponding author. Tel.: +33 380396857.
E-mail addresses: perrine.pittet@u-bourgogne.fr, perrine.pittet@checksem.fr (P. Pittet), christophe.cruz@u-bourgogne.fr (C. Cruz), cnicolle@u-bourgogne.fr (C. Nicolle).

http://dx.doi.org/10.1016/j.compind.2014.07.006
0166-3615/

readable HTML documents to machine-readable documents. This Web 3.0 technology has provided a powerful tool to represent the knowledge of a domain with the aim of making heterogeneous information understandable and easy to process by both human and machine: ontology. Nevertheless, ontology, to be useful at the industrial level must not only be used as a meta-model. It has to focus on the project content to help exploiting its business processes through semantic representation. Besides mechanisms of ontology evolution can help maintaining the consistency of this knowledge throughout the project life cycle.

The problem is then the following for facility management: how to build a long-term vision of knowledge taking into account the life cycle and the heterogeneity of the actors of the building?

This paper proposes a new approach for ontology change management applied on the facility management domain. The industrial challenge is, firstly, to ensure consistency of a FM project knowledge from the construction phase to the technical property management phase (after delivery). Secondly, it has to provide to each actor of the FM project a personal "view" of the building knowledge related to its business profile. The scientific approach is based on a methodology for managing ontology life cycle including ontology evolution and versioning features, in conjunction with contextual view modeling. The major contribution is the realization of a versioning system exploiting evolution logs as storage model, for evolving and maintaining the ontology consistency, and extracting and managing ontology views. The main results are the improvement of an innovative collaborative platform dedicated to facility management and the foundation of a scientific approach for ontology change management based on evolution logs. This is a real revolution on the market of uses. The methodology has been implemented in the OntoVersioning API, extending the Java Jena API dedicated to OWL ontology management with the change management features cited above.

This paper is articulated in two main sections. Section 2 presents a state of art on ontology change management and a discussion on the related deadlocks of the existing proposals regarding contextual and user-centered systems like facility management systems. A state of art on ontology views gives the further objectives change management should try to reach to bridge these gaps. Section 3 describes our proposal called OntoVersionGraph, which is an ontology change management methodology formalized for change management of $\mathcal{SHOJN}(\mathcal{D})$ ontologies in contextual and user-centered systems. The $\mathcal{SHOJN}(\mathcal{D})$ description logic was chosen to formalize OntoVersionGraph because description logics are logics dedicated to knowledge representation and this particular logic has the same semantics as the OWL DL ontology language, which is massively used to represent formal ontologies. However, the methodology can be extended to other description logics by undertaking more or less language constructors for change modeling. The methodology process is applied to resolve a facility management common issue implying evolution of a BIM ontology impacting different views and actors of an FM project.

2. Change management state of art and discussion

This section aims at identifying the deadlocks of ontology change management regarding collaborative and user-centered systems based on ontologies such as FM projects and introducing the solution we chose for our methodology. The first part presents a state of art on ontology life cycle and its management. The second part lists the common issues identified by the literature concerning change management of ontologies in collaborative systems, like the need of specialization or profiling such ontologies. The third part deals with the ontology view solution to bridge this gap in the context of FM projects multi-user ontology management.

2.1. From ontology life cycle to change management: a state of the art

In recent years, building ontologies are gaining ground to provide to the Semantic Web clear semantics in an agreed, consistent and shared encodings. Actually, ontologies make possible to application, enterprise, and community boundaries of any domain to bridge the gap of semantic heterogeneity. Ontology development, to be correctly achieved, requires a dynamic and incremental process [1]. It starts with a rigorous ontological analysis [2] that provides a conceptualization of the domain to model agreed by the community. The ontology, specified in a formal language, approximates the intended models of the conceptualization [3]: the closer it is the better it is. The ontology needs to be revised and refined until an ontological commitment is found. Ulterior updates of the ontology, addressed by ontology evolution, aim at responding to changes in the domain and/or the conceptualization [4]. Changes are consequently inherent in the ontology life cycle. Ref. [5] defines an ontology change as an action on an ontology resulting in an ontology that is different from the original version. Changes help in incorporating new features by improving the ontology conceptualization, however their application can generate inconsistencies.

An ontology becomes inconsistent when its conceptualization and specification does not respect the constraints of the model and axioms of this ontology (Haase and Stojanovic [6]). Two main inconsistency types can affect the ontology: structural and logical. Structural consistency maintenance implies a change modeling respecting the language constructors, which are structural constraints. Logical consistency maintenance deals with the respect of logical constraints in order to protect the ontology semantics. In the literature, ensuring structural and logical consistent updates of ontologies during their lifecycle is the activity studied by the Ontology Evolution research domain.

A couple of ontology evolution methodologies have been proposed like [7–9]. Among them, the AIFB methodology [9], which is one of the most popular, identifies 6 phases to ensure the quality of the ontology evolution process: detection, representation, semantics, implementation, propagation and validation. The validation of one evolution process iteration gives birth to an evolved version of the ontology. However, the AIFB process as such does not take into account the management of the different versions of the ontology generated all along its lifecycle. Accessing different versions of an ontology, rollback to a previous one or comparing two of them are features that are provided by the ontology versioning studies. Ontology versioning refers to the capacity of managing ontology evolution by creating and managing its different versions (Plessers and De Troyer [10]). A versioning system is often based on the use of evolution or versioning logs to trace changes applied at each evolution iteration. Such logs can usually be queried to retrieve and compare ontology versions in a transparent way.

Ontology evolution and versioning are so linked that Ref. [5] introduces the notion of change management to define the interactions between these two activities. Change management combines ontology evolution and versioning features to manage ontology changes and their impacts. Following these guidelines, numerous change management methodologies are proposed in the literature.

2.2. Change management related works deadlocks

Ontology change management systems (OCMS) are direct implementation of these change management methodologies. Since 2007, many works have combined ontology evolution and versioning into OCMS [4,11–17]. The evolution subject has been deeply studied in these works. They especially addressed the

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

consistency issue for the application of changes on the ontology. These proposals constituted a consequent background for ontology change management but they did not take into account certain specificities of ontologies.

One of them is the fact that ontologies are decentralized data [25]. It means that multiple versions or views of the same ontology evolution are bound to exist over the Web and must be supported. It is the case for ontology-based collaborative systems that have to coordinate multiple user modifications on this ontology. Each user-version of the current ontology updated has to be confronted to the other ones and evaluated in order to commit the changes to the original one. It implies that chronological evolution of a single ontology is not enough to manage ontologies in multi-contextual or multi-user systems. Actually, managing different parallel versions or user-specializations of a same ontology would bridge this gap.

Another unaddressed known characteristic is that ontologies are meant to grow during their lifecycle and may become too large to be used in its original scale by potential applications. Indeed, ontology development implies a dynamic and incremental process starting from the creation of a brute ontology, which has to be revised and refined [12]. Refinement often leads to the improvement of the ontology level of detail corresponding to the addition of new elements to its conceptualization and population. Therefore the ontology size may increase after each refinement iteration, with no guaranty that the ontology is still manageable and understandable by humans. The overall understanding of a complex ontology by the community may be impossible [25]. A human cannot then either realize evolution of such ontologies. A solution would be to specialize the ontology according to user business skills or needs into which would decrease its size and improve its understandability for target users.

In the literature, ontology views have been defined to bridge this ontology specialization issue and improve ontology reusability. Several definitions and implementations of ontology views have been studied in the Ontology View Management specific research field. However no agreement was found. Nevertheless, a view generally is a subset specification on an ontology, which allows extracting a manageable portion of the ontology capable to be used and queried by applications like the whole ontology. Besides, the resulting sub-ontology can be generated not only as a sub-graph of the ontology but also as an independent ontology, itself being a new interpretation of the domain. It can be considered as a new parallel version of the actual ontology validating the decentralized quality of ontologies.

The following part shows how ontology views can help in change management of FM projects.

2.3. Ontology Views for Facility Management Projects

Building Information Modeling ontologies represent the knowledge of a building and the associated **FM** project features. Considered as upper-level application ontologies, they are intended to support the domain knowledge requirements of multiple disparate users. They are often too large or too complex, depending on the building knowledge complexity, however, for any specific actor of the project. One additional difficulty is the fact that users of such user-centered ontologies are normally not ontologists or computer scientists. This is the case of the actors of **FM** projects. Automated or semi-automated evolution of these large ontologies is then often tricky to realize because of their complexity.

In addition, the global view of the building and its associated **FM** project features provided by such ontologies may not exactly match the views required by particular actors. For example, the real estate manager will not need to access to the knowledge of the electrical installations to change the color of a sofa in a room. The same issue is identified in the literature for domain ontologies that are often partially accessed by specific applications (Noy and Musen). Additionally, unnecessary accesses to useless parts of the ontology can be slow if the ontology is complex. In order to utilize these ontologies, therefore, applications often require custom ontology views tailored for use within their specific context. In **FM** projects, an architect view of the ontology will contain only architectural knowledge of the building and the related features for facility management. In user-centered applications like **FM** dedicated ones, usually, users just need to use a small portion of their resources or may not have the right to access certain parts of the ontology. Views are used then to manage access policies, profile, context and data security for users [25]. Views can as well optimize the access time and query processing of ontology by only loading a small portion of this ontology. In recent years, many academic and industrial research directions have focused on these issues and some of the notable works like [26].

The concept of ontology views providing an understandable, specialized and lightweighted portion of an ontology, fit very well with the need of **FM** projects. The different actors of the building face many difficulties to understand the whole information of the Building Information Modeling. Moreover, each actors use specific heterogeneous softwares. The use of views as subontologies specialized according to data and business skills and processes bridges this gap of heterogeneity.

Modeling and extraction of such ontology views is beyond the scope of this paper, which far focuses on the change management impact aspects of such views evolutions on the ontology and inversely. Indeed, to cope with our objectives we present in the following section our proposal, called OntoVersionGraph. The OntoVersionGraph main contribution comparing to the related works cited before, remains in the extension of ontology change management approaches to the management of $\mathcal{SHOJN}(\mathcal{D})$ ontology views along the ontology lifecycle.

3. OntoVersionGraph proposal

This section is articulated in two parts. The first part describes our methodology around a six-phased process to evolve, maintain and version $\mathcal{SHOJN}(\mathcal{D})$ ontologies and their views. The second part illustrates the purpose of our methodology in resolving a common change management issue of a FM project involving several actors.

3.1. OntoVersionGraph change management summary

OntoVersionGraph is a change management methodology dedicated to $\mathcal{SHOJN}(\mathcal{D})$ ontologies and their adaptation in multiple contexts. In this paper, we call $\mathcal{SHOJN}(\mathcal{D})$ ontology any ontology specified in a language whose expressivity level is equivalent to $\mathcal{SHOJN}(\mathcal{D})$ description logic (see $\mathcal{SHOJN}(\mathcal{D})$ Ontology Model in Annex 1). Following the change management definition of Klein [5], it combines ontology evolution and versioning. The evolution process is divided in the six phases recommended by Stojanovic [18]. To support specialization and modularity of ontologies with multi-user requirements all along their lifecycle, the methodology provides a change management methodology improved with ontology views specification and impact management. Views, defined here as sub-ontologies, i.e. parallel specialized sub-versions of the ontology, can be derived to specialize the ontology or adapt it to a certain user profile, decreasing its useless complexity and cutting off unused parts of it. The methodology can be applied to build and evolve any $\mathcal{SHOJN}(\mathcal{D})$ ontology from scratch. The construction of a global change log tracing every change implemented since its creation

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

allows performing versioning and consistency qualification tasks. Existing ontologies can also be evolved; yet, a reconstruction phase of a global change log for the ontology is required to display those tasks. We have defined a $\mathcal{SHOJN}(\mathcal{D})$ ontology model (see Annex 1) to model $\mathcal{SHOJN}(\mathcal{D})$ ontologies and their change modeling (i.e. changes representation and change logs evolution).

The change management methodology is briefly described in the following paragraph according to each phase illustrated in Fig. 1. It is important to note that the entire change management process is automatized by the use of the OntoVersioning API, which implements each phase except the Change Detection phase.

3.1.1. Change detection phase description

First, the user establishes the list of modifications needed for the ontology: (cf. Fig. 1 change detection phase). The change detection phase is left free to users of the methodology in the case of a classical evolution process of an ontology or a view. They can list the changes required during the construction of the ontology or from feedback from users of the latter. For instance, a real estate manager wants to transform a meeting room in a computer room. This objective implies a succession of modifications related to his business skills, that he has to identify. It also involves modifications related to other areas of business like network installations for instance. The concerned actor of the building will manage this task after they find an agreement. In case of the evolution of an ontology impacted by the evolution of a view, this phase is replaced by the propagation phase of the evolution process of the view.

3.1.2. Change modeling phase description

The modeling phase stands for the translation of the detected changes into formal ontological operations. A domain expert can easily model changes by first choosing an ontological change type from the list of OWL DL change operator types displayed by the API in order to be formally represented (cf. Fig. 1 change representation step). Their formalization in $\mathcal{SHOJN}(\mathcal{D})$ change operator types is given in Annex 2 (see Table 1 in Annex 2) A priori, removing a table "*tablel*" from the meeting room "meetingRooml" can be represented by the deletion of the instantiation "AddInstanceOfObjectProperty(locatedIn, tablel, meetingRooml" of the role "locatedIn". Second, the domain expert can instantiate each chosen change

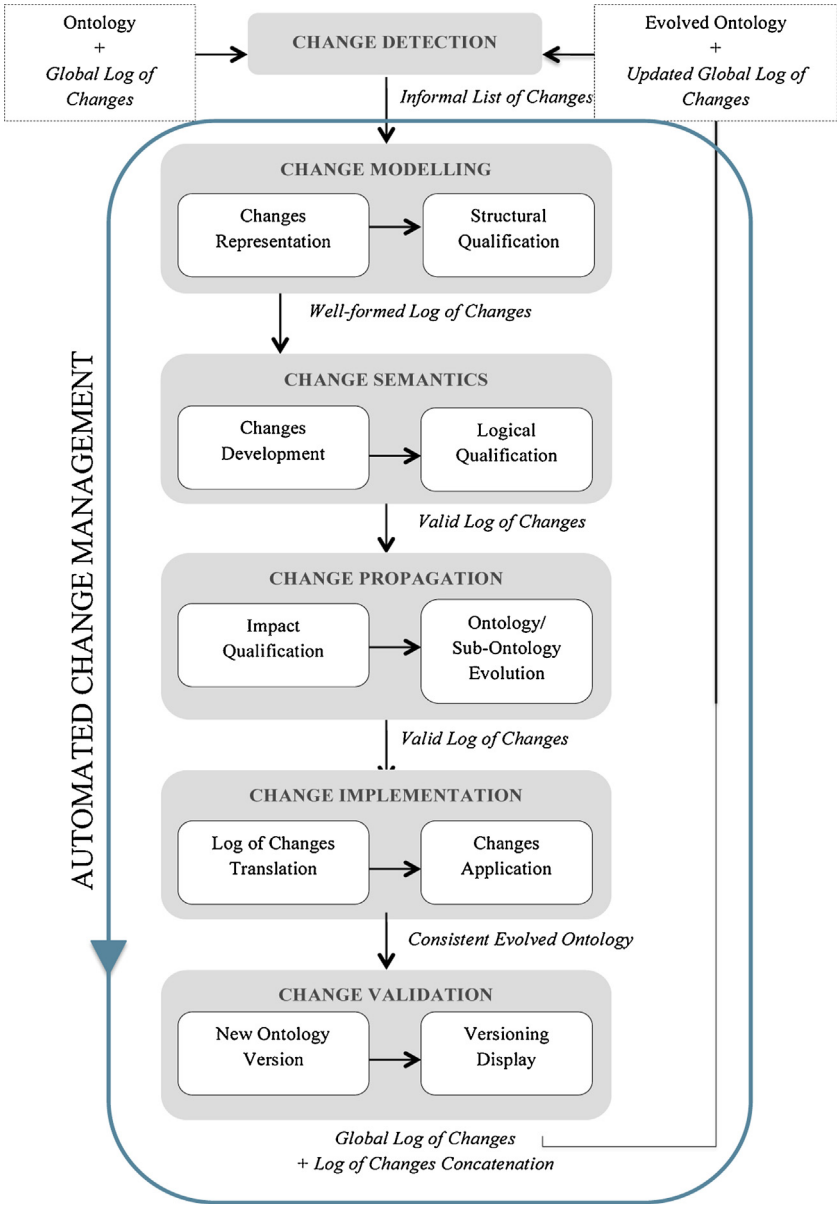

Fig. 1. Ontology change management methodology schema.

operator with the ontological element(s) (concept, instance, role, attribute, data type and/or data value) on which the modification has to be realized (see change definition in Annex 3). This task can be helped by using an ontology visualization tool to let the user visually choose the right elements. This visualization tool is however not yet implemented in the OntoVersioning API. Also, in the API, the instantiation consists in adding ontological elements URIs as parameter of the chosen change operators. This instantiation automatically generates a brute list of changes represented in a log of changes (see log of change definition in Annex 3), which is then analyzed during a structural qualification instantiating the latter example, the instantiated change corresponds to the instantiation of the role "locatedIn" with "table1" in the domain and "meetingRoom1" in the range: "DeleteInstanceOfObjectProperty(locatedIn, table1, meetingRoom1)". Third the structural consistency preventive resolution approach based on structural dependencies analysis (see Table 2 in Annex 4) prevents the ontology from the introduction of structural inconsistencies when the changes are applied on it. Finally, if structural dependencies are found, the list of changes is updated with structural resolutions to provide a new structural validated log of changes. In our example, there is no structural constraint identified for the deletion of an instantiation of a role. Therefore its modeling does not require any additional addition or deletion of changes. The log of changes remains the same as at the end the previous phase. The log containing the current changes, is different from the ontology global log of changes as it contains only the list of changes of the current evolution iteration. The ontology global log of changes thus contains all the changes applied on the ontology since its creation before the current iteration. The complete change modeling process and the structural consistency analysis are presented in the technical report "Change Modelling and Structural Consistency of $\mathcal{SHOJN}(\mathcal{D})$ Ontologies" (Pittet [19]).

3.1.3. Change semantics phase description

In order allow the user predicting the effects on the ontology logical consistency of the changes modeled in the log of changes, each change is turned semantically understandable (cf. Fig. 1 change semantics phase). First, complex changes are decomposed into lists of basic changes. Potential logical inconsistencies can then be easily identified according to the ontology logical constraints derived from the set of $\mathcal{SHOJN}(\mathcal{D})$ basic changes. For instance, moving a meeting table "table1" in another meeting room "meetingRoom2", corresponding to the edition of the range of an instantiation "EditRangeInstanceOfObjectProperty(locatedIn, table1, meetingRoom2)" of the role "locatedIn", is a complex change that need to be decomposed in a deletion and an addition of two basic changes: "DeleteInstanceOfObjectProperty(locatedI, table1, meetingRoom1), AddInstanceOfObjectProperty(locatedI, table1, meetingRoom2)". Second, the decomposed changes replace the composed ones in the current log of changes. Following the last example, the complex change "*EditRangeInstanceOfObjectProperty(locatedIn, table1, meetingRoom1)*" is then replaced by "*DeleteInstanceOfObjectProperty(locatedIn, table1, meetingRoom2)*", *DeleteInstanceOfObjectProperty(locatedIn, table1, meetingRoom2)*". The log is then concatenated with the ontology global log of changes in order to proceed to a logical consistency preventive resolution on the whole ontology. Indeed if the application of the changes is qualified consistent for the whole ontology then it is also consistent for each user views. It allows to validate the logical consistency of all user views without repeating a logical consistency analysis for each of them when changes will be propagated. Third, this automated logical consistency qualification is handled by the global change log analysis regarding logical constraints. It consists in the identification of inconsistent successions of basic changes in the global change log thanks to their logical insatisfiabilities (processed from the changes logical constraints) formalized by a set of potential generic succession of changes. To illustrate, if the ontology global log of changes contains an role value restriction for the "meetIn" role with the meeting room meetingRoom1 "AddRoleValueRestriction(meetIn, meetingRooml)", a logical insatisfiability, derived from the logical constraints of the "AddRoleValueRestriction(Instance)" basic change, is the interdiction of using any other instance as value for this role. The addition of the change "AddRoleInstanciation(meetIn, person1, meetingRoom2)" is then logically inconsistent. The methodology associated with this logical inconsistency preventive resolution approach is called CLOCk (Change Log Ontology Checking) and has been presented in the conference paper [20]. Finally, the system provides the user the list of inconsistent set of changes detected in the ontology global change log and returns the changes in cause in the current log of changes, to correct them before they are implemented. Each modification done by the user on the log of changes is propagated to the ontology global change log and implies a new logical consistency preventive resolution until no more inconsistency is found. In the previous example, the identified inconsistent combination of changes would be AddRoleValueRestriction(Instance), AddRoleInstanciation(meetIn, person1, meetingRoom2)" and the change causing the inconsistency "AddRoleInstanciation(meetIn, person1, meetingRoom2)". The user would understand he has to choose the other meeting room for person1 to meet.

3.1.4. Change implementation phase description

When the semantics of the global change log is validated, changes recorded in the log of changes can then be implemented (cf. Fig. 1 change implementation phase). The change implementation phase follows two steps. First it generates a copy of the ontology and all its views. Second it applies the changes on the duplicated ontology on both terminological and assertional levels. It avoids turning the source ontology incompatible or inaccessible to the users and applications before a final version is committed. Consequently when evolving an ontology, the original ontology is still available for querying. The evolved copy of the ontology is then ready to be checked by a logical consistency corrective approach to detect the logical inconsistencies that potentially may have been forgotten by the preventive approach. The OntoVersionGraph methodology uses the Pellet reasoner to achieve this task. If logical inconsistencies are detected by the reasoner, then a new change modeling phase is launched. The change propagation phase cannot be started until the ontology is not assumed as structurally and logically consistent.

3.1.5. Change propagation phase description

When the implementation is done, dependent artifacts of an ontology can also be impacted by the ontology update. The change propagation phase (cf. Fig. 1 change propagation phase) aims at informing and assuring the consistency of each of them. In our methodology we focused on two kinds of change propagation. First, the change propagation from ontologies to the sub-ontologies extracted from them. The corresponding approach is called Top-Down. Second the change propagation from sub-ontologies to their source ontology and the other sub-ontologies extracted from it. The corresponding approach is called Bottom-Up. Sub-ontologies are called "views" in this paper, in reference to the different points of view of the actors of an FM project. Indeed each actor of an FM has its own user view on the BIM ontology according to his business skills.

To illustrate the Top-Down and the Bottom-Up approaches, let us go back to our first example. If the network manager has added network jacks in one of the walls of the new computer room and connected computers to these network jacks, this update has to be

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

accessible for the electrician. A first Bottom-Up Propagation is required to spread the changes done by the network manager to the BIM ontology. When the BIM ontology is consistently evolved, a Top-Down propagation is required to apply the changes on all the other user views. First, the Bottom-Up step begins with the propagation of the changes recorded in the log of changes during the network manager view evolution to the BIM ontology. It is done by simply adding them to a new log of changes, which will be used during the BIM ontology evolution process. The log of changes of the evolving network manager view contains "*AddRoleInstanctiation(hasNetworkJack, spaceWall32, networkJack1), AddRoleInstantiation(isConnectedTo(computer21, netWorkJack1), AddRoleInstantiation(hasNetworkJack, spaceWall33, networkJack2), AddRoleInstantiation(isConnectedTo(computer45, netWorkJack2)*". These changes are consistently assessed during the BIM ontology evolution process and, if consistent, they are implemented in the ontology. If not, the network manager is asked to revise the changes to apply on his view according to the logical consistency analysis results. When the changes are implemented on the BIM ontology, the corresponding change propagation phase starts a Top-Down propagation. Second, the Top-Down approach begins with the assessment of the impact of the ontology changes to its views and which generates a propagation log for each of them. In our example, the view global change log of the electrician view is analyzed to identify which changes of the current log are related to its elements. Also, if the electrician view does model computer network connections, then the propagation log will contain "AddRoleInstantiation(hasNetworkJack, spaceWall32, networkJack1), AddRoleInstantiation(hasNetworkJack, spaceWall33, networkJack2)". Third, the changes concerning ontological elements of the view, contained in the propagation log, are added to each view global change log and are directly implemented. No additional consistency analysis is required as it has been assessed on the entire BIM ontology. In our example, the propagation log is concatenated to the global change log of the electrician view and the changes are applied to the view so that the electrician can visualize the new network jacks installation when the update is validated.

The complete description of the impact management of change between a $\mathcal{SHOJN}(\mathcal{D})$ ontology and its sub-ontologies (views) is available in the PhD thesis entitled "OntoVersionGraph: A Change Management Methodology dedicated to Formal Ontologies and their User Views in a Collaborative Context: Application to $\mathcal{SHOJN}(\mathcal{D})$ SS" [19]. This document also deals with sub-ontology extraction, which is beyond the scope of this paper.

3.1.6. Change validation phase description

Finally, if the user validates the changes implemented and propagated (cf. Fig. 1 change validation phase). The corresponding evolved copy ontology and its updated views are committed as new versions. The new ontology and/or views versions replace the current ones. The current global logs of changes are archived as previous versions in the versioning system. The ontology and/or views logs of changes issued from the validated evolution processes are concatenated to copies of the corresponding archived global logs of changes, producing new global logs of changes. The new ontology and/or views global logs of changes are stored in the versioning system. Other versioning tasks are displayed taking into account this new version like committing a rollback of the changes or accessing several versions according to the criteria defined by the user as stated in [21]. It can also delimit compatibility by retracing evolution operations.

3.2. OntoVersionGraph proposal comparison with related works

This part presents a discussion on ontology change management (OCM) related works. Eight methodologies described in [5,9,11,12,15,16,22,23] are compared in Table 1, according to the eight criteria defined for our proposal and which are satisfied by OntoVersionGraph.

According to Table 1, the OCM methodologies of the seven proposals all propose a formal methodology, which at least allow to apply formal changes and maintain the ontology consistency and expressivity level. Most of them have chosen a formal representation language expressive enough to specify formal ontologies. So the 1st and 2nd criteria are satisfied. They especially addressed the logical and structural consistency issues related to the application of changes on an ontology, which respectively constitute the 4th and the 5th criteria of my thesis. However, four criteria remain unsatisfied or partially satisfied by the existing methodologies: 3rd, 6th, 7th and 8th. None of the proposals handle change management on formal ontologies representing the several views of a domain (3rd criterion). The closest proposal is Plessers (2006)'s approach, whose issue is the change management of semantically heterogeneous ontologies representing different views of a domain in a distributed environment. However the propagation of changes is not addressed between these ontologies but from each ontology to their dependent artifacts (applications, reusing ontologies). Evolution and versioning tasks are only done on the ontology in most of the proposals, and none proposes their use on the different views of this ontology (6th criterion). Most solutions for the change propagation phase focus on the impact resolution of changes on the ontology instances, on the dependant ontologies and artifacts such as applications, semantic annotations, semantic SEO. None of them addresses the propagation of changes from the ontology to its views (7th criterion). The adaptation of an ontology to new points of view is not addressed either (8th criterion).

This state of the art highlights the following observations. OCM is performed over time on an ontology that chronologically evolves in its entirety. It is mainly used for the update of the ontology to correct its conceptualization, when its described field has evolved, or when errors or potential improvements are identified. It does not address the evolution and versioning management of the specific assumptions users might have on the domain. It has not therefore been thought to manage formal ontology as defined in the previous chapter, i.e. an ontology representing all the assumptions of users on the domain. It is not intended to guide the specialization of an ontology to a sub-domain that it describes. The propagation of changes from an ontology to various specialized views is thus not addressed either.

3.3. Application to an FM project change management problem

This second part describes the successive steps using our methodology for the resolution of a common change management issue of an FM project involving several actors.

3.3.1. FM projects change management

In this part, we consider the ontology of representing the knowledge of a building and its FM related features. We call it a BIM ontology. The lifecycle of a building can be divided into 4 main stages (cf. Fig. 2), which are the study, the project, the exploitation and the restructuration phases.

Each phase is generally managed independently and is divided into superimposed layers. These layers represent aspects of the building, which are handled by various methods, tools and specific business skills.

The layers of the project phase correspond to the various trade associations, which intervene in the construction of the building. The layers of the exploitation stage correspond to the various services of management and production, which use the structures of the building. The layers of the restructuration phase correspond

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

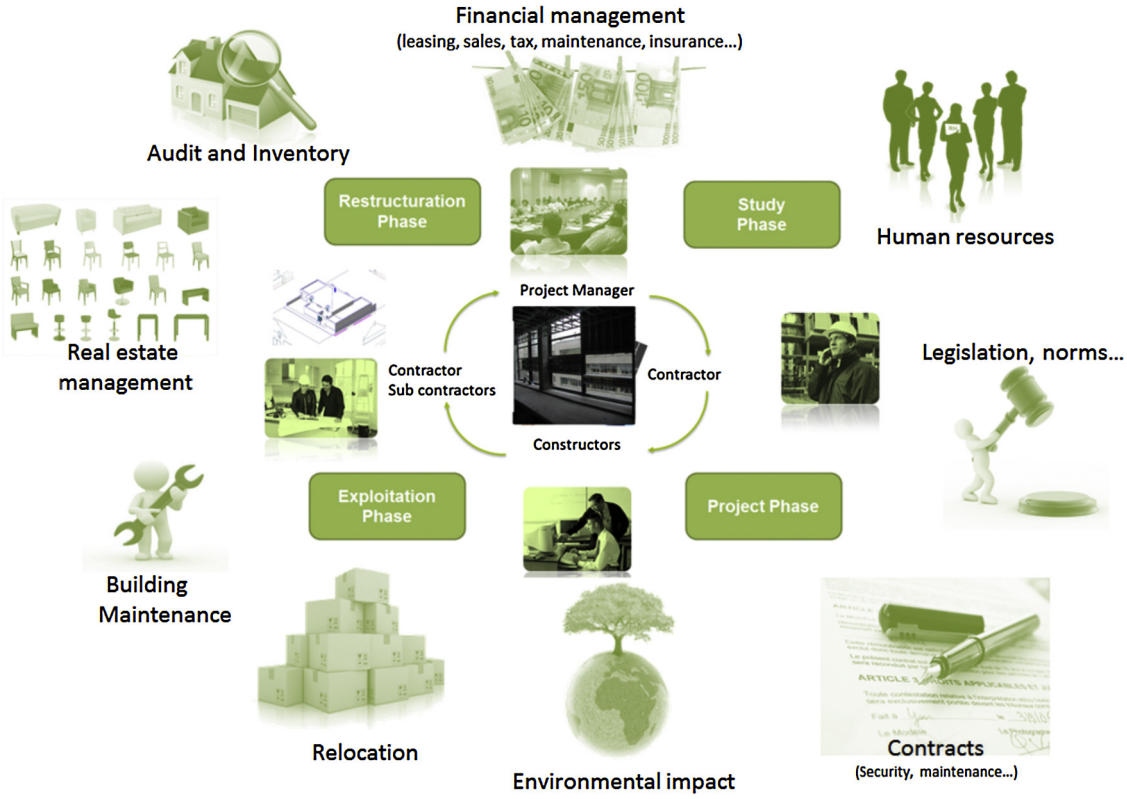

Fig. 2. Lifecycle of a building and integrated processes.

to the outside contributors or to the services of maintenance and management of the building. Each layer has its own information. Each layer exchanges partial information with the other layers. Each stage exchanges partial information with the other phase. Moreover the FM team, which design and build the final building, is not the same team, which maintains the building.

This is the role of facility management to coordinate the physical workplace with the people and work of the organization for each layers and at each phase of the project. Also, a collaborative system based on an evolvable ontology describing the knowledge of both building and facility management features (BIM ontology) can automatize this task all along the building lifecycle.

However a BIM ontology describing every layer of each phase of the building lifecycle is huge and complex. Accessing the entire ontology is also not necessary for a user that only needs to access the building information dedicated to its own business rules according to a certain layer of a certain phase of the project etc. Therefore, each user of each layer should have a personal access to the knowledge of the building according to its business skills, privacy and access rights and at its own scale. Then to deal with these requirements, change management of an FM project should allow facilities manager to organize all knowledge generated during building lifecycle in end-user contextual views (cf. Fig. 3).

In FM projects, the main source of inconsistency is related on norms evolution. More than twenty different domains have to be coordinated. Each domain is defined by norms and laws, which can evolve every year. Another source of inconsistency occurs during the modification of the building during the maintenance phase. In this paper, we chose to apply our change management methodology on the resolution of this second source of inconsistency.

Fig. 3 resumes the application of the change management process on a case of building maintenance. An actor of the building detects changes (see step 1a in Fig. 3) to apply from its own view of the BIM ontology. He is not aware of the entire knowledge of the BIM ontology but he can however model these changes on it as the modeling phase (see step 2a in Fig. 3) of the change management process contains an automated structural qualification analysis on the whole ontology that [23] will complete his modeling with additional required changes. If the structural resolution implies prior changes from other business domains, the concerned actors are alerted and the current change management process is suspended until the required modifications are done. Then the change semantics phase (see step 3a in Fig. 3), also automated and applied on the whole ontology, guides the actor in the logical correction of his modeling. If the logical resolution strategy chosen by the actor implies logical corrections from other business domains, again the change management process is suspended until the required modifications are done. When consistency is validated, these changes are implemented on a copy of the view (see step 4a in Fig. 3) and propagated (see step 5a in Fig. 3) to the BIM ontology. The changes propagated are modeled and assessed on the BIM ontology (see step 1b to 3b in Fig. 3). If assessed consistent they are implemented on a copy of the ontology (see step 4b in Fig. 3) and propagated to the other views of the other actors of the building. Then the implementation of the changes (see step 5c in Fig. 3) is realized on the impacted views. Finally, the actor validates the evolution of its view and indirectly the BIM ontology and other impacted views ones. This change management process is used for the resolution of the FM project common issue described in the following paragraph.

3.3.2. Application of the change management methodology to resolve an FM project issue example

Let us focus on a common issue occurring in the lifecycle of a building: an architect wants to make a hole in an existing wall to set a doorway.

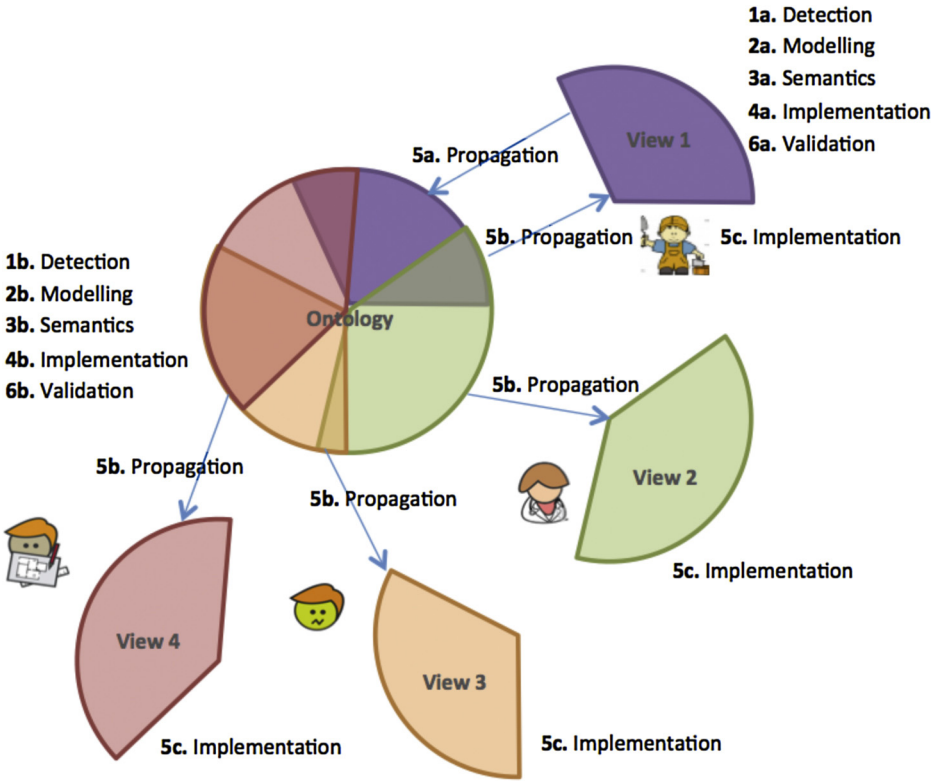

Fig. 3. Facility management Bottom-Up Approach.

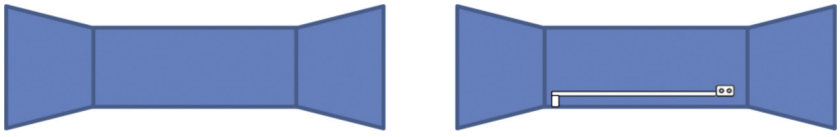

Fig. 4. The wall architect point of view (on the left) and electrician point of view (on the right).

3.3.2.1. Introduction to the problem. Let us state that, from the architect point of view the wall does not contain any architectural constraint that would prevent from setting this doorway (c.f. Fig. 4 left view). But from an electrician point of view (c.f. Fig. 4 right view), making a hole in this wall may contradict the electrician view conceptualization as well as the ontology one.

Indeed if an electrical conduit passes along the wall, setting a doorway may cut the conduit. More formally it means that the doorway wall space overlaps with the electrical conduit one (c.f. Fig. 5). We imagine it is not permitted by the logical constraints of wall spaces, which can contain maximum one object. To simplify the description of the example, let us model the wall like a grid of 12 wall spaces as illustrated below (c.f. Fig. 6):

The red area is composed of the wall spaces (c.f. wallSpace1, wallSpace2, wallSpace3 and wallSpace4 in Fig. 6) containing the electrical conduit. The green area delimits the space of the doorway (c.f. wallSpace2 and wallSpace6 in Fig. 6). The striped wall space (c.f. wallSpace2 in Fig. 6) is the one where the electrical conduit and the doorway overlap.

A consistent application of the modification would imply to divert the electrical conduit or to remove it. In practice, several solutions can be chosen: remove the electrical conduit and the

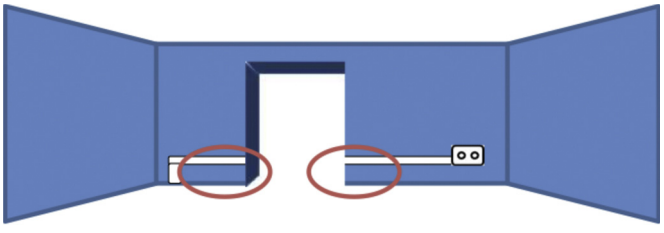

Fig. 5. Inconsistent brute application of the modification from the electrician point of View.

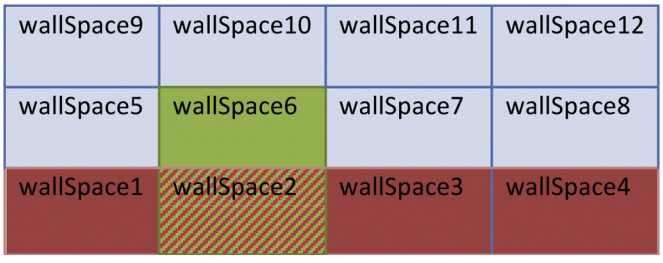

Fig. 6. The wall as a grid of wall spaces with the inconsistent brute application.

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

Fig. 7. Consistent application of the modification from the electrician point of view.

outlet, passing it at the top of the doorway, or in the ground (c.f. Fig. 7) or in the ceiling or along another wall.

Let us model this consistent application on the grid of wall space (c.f. Fig. 8). As we can see, the doorway space does not overlap anymore with the electrical conduit one.

But how can we handle this consistent application of change automatically on the BIM ontology if applied by the architect? What will be the impact for other user views? How can we handle the automated propagation of the change to them?

3.3.3. Architect change management process (round 1)

This paragraph describes the first round of the change management process launched by the architect.

3.3.3.1. Change detection. For the architect expert, who is not aware of the electrical conduit presence on "wallSpace2", the need of change corresponds to set a doorway on the space defined by "wallSpace2" and "wallSpace6".

3.3.3.2. Change modeling. From a change modeling point of view, this modification refers to the addition of an instance of the concept "Doorway", and the addition of the instantiation of the role "containsPartOf" with this instance in the range and the instances of the concept WallSpace concerned in the domain ("wallSpace2" and "wallSpace6"). The corresponding log of changes logi can be represented: log_i=((addConceptInstanciation,(-doorway1, Doorway)), (addRoleInstantiation,(containsPartOf, wallspace2, doorwayl)), (addRoleInstantiation, (containsPartOf, wallspace6, doorwayl))). According to the structural consistency constraints, this sequence of changes does not contain any impacting change. There are only additions of basic changes. So the structural consistency is preserved. The log of changes log_l does not need to be structurally resolved.

3.3.3.3. Change semantics. As changes represented in log_l are only basic changes, they do not need to be developed in order to extract the unsatisfiability constraints of their corresponding basic change operator type. The logical consistency preventive approach checks log_l and does not detect any inconsistency. Indeed the changes chosen by the architect expert does not contradict the knowledge of his own view.

wallSpace9	wallSpace10	wallSpace11	wallSpace12
wallSpace5	wallSpace6	wallSpace7	wallSpace8
wallSpace1	wallSpace2	wallSpace3	wallSpace4

Fig. 8. The wall as a grid of wall spaces with a consistent application.

3.3.3.4. Change Implementation. Changes of log_1 assumed as structurally and logically consistent are then applied on a copy of the sub-ontoloy. The second logical consistency check is handled by the reasoner Pellet. As expected is assessed consistent. Changes can therefore being propagated to the source ontology.

3.3.3.5. Change propagation. As changes come from the evolution of a sub-ontology extracted by a view from the ontology, the change propagation type corresponds here to the Bottom-Up Propagation approach. It implies starting a new evolution process for the source ontology according the changes represented in the log.

The sub-ontology evolution process is stopped until the new evolution process of the source ontology is validated. Then it will be able to achieve the change validation phase.

3.3.4. BIM ontology evolution (round 1)

3.3.4.1. Change detection and modeling. These phases do not need to intervene as changes are already represented in the log of changes and have been assessed structurally consistent. Indeed if changes to apply on a sub-ontology extracted from a ontology are assessed structurally consistent, then they are also consistent on the ontology. There are two reasons. First, both sub-ontology and ontology respect the ontology model, so changes, which are also compliant with the model, cannot be miswritten. Second, only deletion changes can cause the remaining structural inconsistencies and if a deletion change applied on a subset of the ontology is structurally consistent, i.e. the ontological element to delete exists in the subset, then it is also consistent for the whole ontology, which contains the subset.

3.3.4.2. Change semantics. To check the consistency of the whole ontology, according to the log of changes, which is concatenated with the global log of changes of the ontology. Let us state that the global log contains the role maximal cardinality restriction addition change (addRoleMaxCardinalityRestriction, (containsPartOf, i)) and the previous role instantiation addition change (addRoleInstantiation, (containsPartOf, wallSpace2, electricalConduit1)). According to the logical unsatisfiabilities of the change operator type addRoleMaxCardinalityRestriction, the arity "1" of the maximal cardinality restriction of the role containsPartOf forbids the instantiation of this role with more than one instance in the range per instance in the domain. However the instance "wallSpace2" in the domain of containsPartOf has already been used with the instance "electricalConduit" in the range. The addition of the instantiation (addRoleInstantiation, (containsPartOf, wallSpace2, doorway1)) then it does not respect the maximal cardinality constraint. The application of this change is qualified logically inconsistent and has to be resolved by an alternative solution if the architect expert wants to apply the change. The following minimal inconsistent set of changes identified in the global log is given to the architect expert and all the sub-domain experts whose sub-ontology is impacted by this inconsistency: *(addMaxCardinalityProperty, (containsPartOf, 1)), (addRoleInstanciation, (containsPartOf, wallspace2)), (addRoleInstanciation, (containsPartOf, electricalConduit1)).*

The ontology evolution process is stopped until the sub-domain experts propose a new set of changes to apply on the ontology. In addition, the architect sub-ontology evolution process is canceled, the copy of its evolved view is deleted. Also, during the collaborative logical consistency resolution, the architect expert can, for example, ask the electrician expert to proceed to a modification of the electrical conduit path in order to avoid passing "wallSpace2" and also "wallSpace6". The current change management process round is interrupted and the current log log_l is stored until the architect expert is notified by the confirmation of the

demanded update. Then he will be invited to pursue this change management process in a second round.

3.3.5. Electrician change management process

This paragraph describes the change management process supervised by the electrician in response to the architect needs.

3.3.5.1. Change detection. The sub-domain experts have reached a consensus and the electrician expert has received the modification request of the architect expert. He is aware of the location of the doorway wanted by the architect and can choose another path for the electrical conduit. If he chooses to pass it in the ground then he can remove it from the space defined by ''wallSpace1'', ''wallSpace2'' and ''wallSpace3'' and adding it in similar ground spaces (not illustrated here to simplify the example presentation). He can model this change on its own sub-ontology.

3.3.5.2. Change modeling. This change refers to the deletion of the instantiations of the role containsPartOf with the instance ''electricalConduit1'' in the range for the corresponding wall spaces in the domain, and the addition of the similar instantiations of this role but with ground spaces in the domain. The corresponding log of changes can be modeled:

log2 = ((deleteRoleInstantiation, (containsPartOf, wallspace1, electricalConduit1)),
deleteRoleInstantiation, (containsPartOf, wallspace2, electricalConduit1)),
deleteRoleInstantiation, (containsPartOf, wallspace3, electricalConduit1)),
addRoleInstantiation, (containsPartOf, groundSpace1, electricalConduit1)),
addRoleInstantiation, (containsPartOf, groundSpace2, electricalConduit1),
addRoleInstantiation, (containsPartOf, groundSpace3, electricalConduit1)),
addRoleInstantiation, (containsPartOf, groundSpace4, electricalConduit1)).

According to the structural consistency constraints, this set of changes does not contain any impacting change. So the structural consistency is preserved. log2 does not need to be structurally resolved.

3.3.5.3. Change semantics. Considering the new path modeled for the electrical conduit does not interfere with other constraints of the electrician expert view, the logical consistency preventive approach detects then no inconsistency, such that log2 is qualified consistent.

3.3.5.4. Change implementation. The sequence of changes is then implemented on a copy of the sub-ontology. The set of changes of log2 is concatenated with the ontology global log of changes. Then the second logical consistency check realized by Pellet assesses it as consistent. The change propagation to the source ontology is then launched. Here again, the Bottom-Up

3.3.5.5. Change propagation. Propagation approach is used, reactivating the current ontology evolution process.

3.3.6. Building ontology evolution (round 2)

3.3.6.1. Change detection and modeling. These phases do not need to intervene as changes are already represented in the log of changes log2 and have been assessed structurally consistent.

3.3.6.2. Change semantics. The set of changes of log2 is concatenated with the ontology global log of changes. Assuming here again, that the new path modeled for the electrical conduit does not interfere with other constraints of the building ontology, the changes, the logical consistency preventive approach detects then no inconsistency and such that log2 is qualified consistent for the ontology too.

3.3.6.3. Change implementation. The sequence of changes is then implemented on a copy of the ontology. Pellet assesses the its logical consistency and detects no inconsistency. Changes are ready to be propagated.

3.3.6.4. Change propagation. Now changes have been implemented, they have to be propagated to every sub-domain expert views such as the electrician's and architect's ones. The change propagation type corresponds to the Top-Down Propagation Approach. The impact of the changes on and on the sub-ontologies extracted from by its different views has to be evaluated, before update each view definition and generating the sub-ontologies new versions. For each of them the system generates the set of changes containing changes of log2 identified as impacting the view definition. The electrician expert view definition may be impacted as it contains the ontological elements deleted by these changes. The architect view is on the contrary not impacted because his view does not model electrical installations. Then each impacted view definition is updated with the corresponding extractions deletions. Finally all of the sub-ontologies are re-extracted from the evolved copy of the ontology, according to their updated or non updated view definitions, generating their new versions. The electrician expert sub-ontology is then re-extracted from the ontology, according to its updated view definition. The architect expert one, is also re-extracted even if the corresponding view definition is not impacted.

3.3.6.5. Change validation. The ontology evolution process is validated and the evolved copy of the ontology replaces the current one, which is removed from the versioning system. Its global log of changes is concatenated with log2 to produce the new global log of changes of the new ontology version,. Its current global log is stored in the versioning system as a previous global log of changes of the ontology to be exploited for ulterior versioning purposes. Each sub-domain expert sub-ontology is replaced by their evolved copy, such as their corresponding global logs of changes.

3.3.7. Architect change management process (round 2)

This paragraph briefly describes the second round of the change management process of the architect expert cancelled previously. The architect expert has received the update notification from the electrician. He is now assured that he can set the doorway on the space delimited by ''wallSpace2'' and ''wallSpace6''. He can load a new change management process on his evolved sub-ontology. He reuses the log of changes log1=((addConceptInstanciation, (doorway1, Doorway)), (addRoleInstantiation,(containsPartOf, wallspace2, doorwayl)), (addRoleInstantiation, (containsPartOf, wallspace6, doorwayl))), represented during the first change management process (c.f. Architect Change Management Process Round 1) in order to set the doorway. Like in the previous process, the changes are assessed structurally and logically consistent on the sub-ontology. They can be propagated to the source ontology using the Bottom-Up Propagation approach, launching a new ontology evolution process. As the update of the ontology has permitted to free the two wall spaces required for the doorway from the electrical conduit, setting the doorway does not cause any inconsistency. Changes are therefore assessed as logically

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

consistent on the ontology, such that the change propagation to the other views, using the Top-Down Propagation approach can be started. Every sub-ontology defined by a view on the ontology are updated. The sub-ontology and the ontology evolution processes are both validated, producing new versions similarly as the previous electrician change management process (c.f. Electrician Change Management Process).

3.3.8. Conclusion

This example shows how our ontology change management methodology can be applied on an evolution problem involving different actors with different views on an ontology.

4. Conclusion

OntoVersionGraph is new approach for ontology change management for multi-context and user-centered systems. It is based on a change management methodology for managing ontology life cycle including ontology evolution and versioning features, in conjunction with contextual view modeling. Its contribution is the impact management of changes between the ontology and its different views.

It has been applied in this paper on the facility management of FM projects. The application of the change management methodology on the resolution of a building maintenance task, has exposed how to ensure consistency of an FM project knowledge during the technical property management phase. It has also show how to maintain the personal "view" of each actor of the building during its lifecycle with the management of change impact.

This proposal has been applied the industrial project called Active3D, for the ontology modeling part and the view management one. The Active3D platform is mainly used to federate all the actions realized on a building during its lifecycle, to merge all information relating to these actions in semantic graphs, to extract some trade views of the building by combining information collected during the lifecycle from heterogeneous sources, and to handle all these views through a dynamic and adaptive 3D interface. Currently, the Active3D platform is used by more than 3000 actors from all civil engineering domains collaborate at each step of the building lifecycle. Today, more than sixty million square meters of buildings are managed with our architecture.

Acknowledgements

Authors would like to thank the Region Bourgogne and Active3D for their support. They particularly acknowledge Yoan Chabot and Maxime Demongeot for their important contribution on the application implementation and Jean-Luc Baril for his expertise on the ontological structure model.

Appendix A. Annexes

In order to formalize our framework the Karlsruhe Ontology Model [27] is used and extended to cover the whole $\mathcal{SHOJN}(\mathcal{D})$ constructors.

Annex 1. Structural $\mathcal{SHOJN}(\mathcal{D})$ model

From a mathematical point of view, an ontology can be defined as a structure. Formally, a structure is a triple $A = (S, \sigma, F)$ consisting of an underlying set S, a signature σ, and an interpretation function F that indicates how the signature is to be interpreted on S.

Definition 1. $\mathcal{SHOJN}(\mathcal{D})$ ontology model.

A $\mathcal{SHOJN}(\mathcal{D})$ ontology is a structure $O = (SO,\sigma O,FO)$ consisting of:

- The underlying set SO containing:
  - Six disjoint sets sC, sT, sR, sA, si, sV, sK_R and sK_A called concepts, datatypes, relations, attributes, instances, data values, relation characteristics (among Symmetric, Functional, Inverse Functional, Transitive) and attribute characteristics (Functional),
  - Four partial orders $\leq_C$, $\leq_T$, $\leq_R$ and $\leq_A$, respectively on sC called concept hierarchy or taxonomy, on sT called type hierarchy, on sR called relation hierarchy and on sA called attribute hierarchy,

    such that $SO := \{(sC, \leq_C), (sT, \leq_T), (sR, \leq_R), (sA, \leq_A), si, sV, sK_R, sK_A\}$,
- The signature σO containing two functions $\sigma_R{:}sR \rightarrow sC^2$ called relation signature and $\sigma_A{:}sA \rightarrow sC \times sT$ called attribute signature, such that $\sigma O := \{\sigma_R, \sigma_A\}$,
- The interpretation function FO containing:
  - A function $\iota_C{:}sC \rightarrow 2^{sI}$ called concept instantiation,
  - A function $\iota_T{:}sA \rightarrow 2^{sV}$ called data type instantiation,
  - A function $\iota_R{:}sC \rightarrow 2^{sI \times sI}$ called relation instantiation,
  - A function $\iota_A{:}sC \rightarrow 2^{sI \times sV}$ called attribute instantiation,
  - A function $\kappa_R{:}sR \rightarrow 2^{sKR}$ called relation characterization,
  - A function $\kappa_A{:}sA \rightarrow 2^{sKA}$ called attribute characterization,
  - A function $\varepsilon_C{:}sC \rightarrow 2^{sC}$ called concept equivalence,
  - A function $\varepsilon_R{:}sR \rightarrow 2^{sR}$ called relation equivalence,
  - A function $\varepsilon_A{:}sA \rightarrow 2^{sA}$ called attribute equivalence,
  - A function $\varepsilon_I{:}sI \rightarrow 2^{sI}$ called instance equivalence,
  - A function $\delta_C{:}sC \rightarrow 2^{sC}$ called concept disjunction,
  - A function $\delta_I{:}sI \rightarrow 2^{sI}$ called instance differentiation,
  - A function $-_C{:}sC \rightarrow 2^{sC}$ called concept complement specification,
  - A function $-_R{:}sR \rightarrow 2^{sR}$ called relation inverse specification,
  - A function $maxCardR{:}sR \rightarrow N$ called relation maximal cardinality restriction,
  - A function $minCardR{:}sR \rightarrow N$ called relation minimal cardinality restriction,
  - A function $\sqcap_C{:}sC \rightarrow 2^{sC}$ called concept intersection,
  - A function $\sqcup_C{:}sC \rightarrow 2^{sC}$ called concept union,
  - A function $\sqcup_{iC}{:}sI \rightarrow 2^{sC}$ called concept union enumeration,
  - A function $\sqcup_V{:}sV \rightarrow 2^{sC}$ called data value union,
  - A function $\sqcap_{iC}{:}sC \rightarrow 2^{sI}$ called concept intersection enumeration,
  - A function $\rho_{\exists R}{:}sR \rightarrow 2^{sC}$ called relation existential restriction,
  - A function $\rho_{\forall R}{:}sR \rightarrow 2^{sC}$ called relation universal restriction,
  - A function $\rho_{R:}{:}sR \rightarrow 2^{sI}$ called relation value restriction,
  - A function $\rho_{\exists A}{:}sA \rightarrow 2nd$ called attribute existential restriction,
  - A function $\rho_{\forall A}{:}sA \rightarrow 2nd$ called attribute universal restriction,
  - A function $\rho_{A:}{:}sA \rightarrow 2^{sV}$ called attribute value restriction,

such that $FO{:} = \{\iota_C, \iota_T, \iota_R, \iota_A, \kappa_R, \kappa_A, \varepsilon_C, \varepsilon_R, \varepsilon_A, \varepsilon_I, \delta_C, \delta_I, -_C, -_R, maxCard_R, minCard_R, \sqcap_C, \sqcup, \sqcup_{iC}, \sqcup_V, \sqcap_{iC}, \rho_{\exists R}, \rho_{\forall R}, \rho_{R:}, \rho_{\exists A}, \rho_{\forall A}, \rho_{A:}\}$.

Annex 2. $\mathcal{SHOJN}(\mathcal{D})$ changes

Table 1 shows the correspondence of our model with $\mathcal{SHOJN}(\mathcal{D})$ constructors, axioms and facts. The first column gives the description logic syntax for the construction of axioms and facts of a knowledge base in $\mathcal{SHOJN}(\mathcal{D})$).

The letters C, T, R, A, I and V represent, respectively, names for classes (concepts), data types, relations, attributes, instances and data values. The second column gives the correspondence with the $\mathcal{SHOJN}(\mathcal{D})$ ontology model defined upper. The third column is described further.

Table 1
Ontology change management methodology schema.

	Criterion	Stojanovic	Klein	Luong	Jaziri	Djedidi	Rogozan	Eder	Plessers
1	Formal Evolution Process with: Change Analysis + Representation + Consistency Maintenance + Propagation	Change Detection + Representation + Semantics + Implementation + Propagation + Validation	Change Detection + Change Propagation + Change Representation + Change Implementation + Version Compatibility + Version Alignment	Change Representation + Consistency Maintenance + Propagation	Change Representation + Consistency Maintenance + Validation	Change Representation + Consistency Maintenance + Change Implementation	Change Detection + Representation + Consistency Maintenance + Validation + Implementation + Validation	Change Representation + Validation	Change Detection + Representation + Propagation + Validation
2	DL $\mathcal{SHOJN}(\mathcal{D})$ or OWL DL Language	KAON	OWL (including OWL DL)	RDF(S)	Language undefined	OWL DL	OWL DL	Oriented Graph	OWL DL
3	Formal Ontology representing the several views on the Domain	N.C.	Distributed Ontologies	Ontologies Used for Semantic Annotations	Any Ontology Types Representable in UML	Local Domain Ontologies	Web Semantic Referenced Ontologies	Temporal Ontologies	Semantically Heterogeneous Ontologies of the same Domain in Distributed Environment
4	DL $\mathcal{SHOJN}(\mathcal{D})$ or OWL DL Logical Consistency Maintenance:	KAON and OWL Lite Logical Consistency Maintenance	N.C.	RDF(S) Logical Consistency Maintenance	Language Independent Logical Consistency Maintenance	OWL DL Logical Consistency Maintenance	N.C.	N.C.	OWL DL Logical Consistency Maintenance
5	DL $\mathcal{SHOJN}(\mathcal{D})$ or OWL DL Structural Consistency Maintenance:	KAON Structural Consistency Maintenance	N.C.	RDF(S) Structural Consistency Maintenance	Language Independent Structural Consistency Maintenance	OWL DL Structural Consistency Maintenance	N.C.	N.C.	OWL DL Logical Consistency Maintenance
6	Evolution/Versioning of Ontologies and its user assumptions	Evolution/ Versioning of Ontology	Evolution/Versioning of Ontology	Evolution/Versioning of Ontology and Semantic Annotations	Evolution/ Versioning of Ontology	Evolution/ Versioning of Ontology	Evolution/Versioning of Ontology and its Semantic SEO	Evolution/ Versioning of Ontology	Evolution/ Versioning of Distributed Ontologies
7	Change Propagation: Ontology to its Views and Views to Ontology	From ontology to instances, dependant ontologies and artifacts	N.C.	From Ontology to Semantic Annotations	From Ontology Concepts to Ontology Sub-Concepts	N.C.	From Ontology to SEO	N.C.	From Ontologies To Dependant Artifacts
8	Adaptation of Ontology to new Points of View	No Adaptation	No Adaptation	No Adaptation	No Adaptation	No Adaptation	No Adaptation	N.C.	N.C.

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

$\mathcal{SHOJN}(\mathcal{D})$ DL syntax	$\mathcal{SHOJN}(\mathcal{D})$ ontology model	$\mathcal{SHOJN}(\mathcal{D})$-based changes abstract syntax
Descriptions		
C	sC	Class(Class)
$C_1 \sqcap \ldots \sqcap C_n$	$\bigsqcap_C$	IntersectionOf(Class$_1$,. . .,Class$_n$)
$C_1 \sqcup \ldots \sqcup C_n$	$\bigsqcup_C$	**UnionOf(Class$_1$,. . .,Class$_n$)**
$\neg C$	$-_C$	**ComplementOf(Class)**
$\{I_1\} \sqcup \ldots \sqcup \{I_n\}$	$\bigsqcup_{IC}$	**OneOf(Class, Instance$_1$. . .,Instance$_n$)**
$\exists R.C$	$\rho_{\exists R}$	SomeValuesFrom(ObjectProperty, Class)
$\forall R.C$	$\rho_{\forall R}$	AllValuesFrom(ObjectProperty, Class)
$R{:}I$	$\rho_{R:}$	HasValue(ObjectProperty, Instance)
$\geq n\,R$	$maxCard_R$	MinCardinalityProperty(ObjectProperty, n)
$\leq n\,R$	$minCard_R$	MaxCardinalityProperty(ObjectProperty, n)
$\exists A.T$	$\rho_{\exists A}$	SomeValuesFrom(DatatypeProperty, Datatype)
$\forall A.T$	$\rho_{\forall A}$	AllValuesFrom(DatatypeProperty, Datatype)
$A{:}V$	$\rho_{A:}$	HasValue(DatatypeProperty, Datavalue)
T	sT	Datatype(Datatype)
$\{V_1\} \sqcup \ldots \sqcup \{V_n\}$	$\bigsqcup_V$	**OneOf(Datavalue$_1$,. . .,Datavalue$_n$)**
R	sR	ObjectProperty(ObjectProperty)
R^-		
A	sA	DatatypeProperty(DatatypeProperty)
I	sI	Instance(Instance)
V	sV	Datavalue(Datavalue)
$C \equiv C_1 \sqcap \ldots \sqcap C_n$	$\bigsqcap_C$	**IntersectionClass(Class, (Class$_1$,. . .ClasS$_n$))**
$C_1 \sqsubseteq C_2$	$\leq_C$	SubClassOf(Class$_1$, Class$_2$)
$C_1 \equiv \ldots \equiv C_n$	ε_C	**EquivalentClass(Class$_1$,. . .Class$_n$)**
$\perp \equiv C_1 \sqcap C_2$	δ_C	DisjointClass(Class$_1$, Class$_2$)
$T_1 \sqsubseteq T_2$	$\leq_T$	SubDatatypeOf(Datatype$_1$, Datatype$_2$)
$V \in T_i$	ι_T	InstancesOfDatatype(Datavalue, Datatype)
$\geq 1R \sqsubseteq C_1$	σ_R	DomainProperty(ObjectProperty, Class)
$T \sqsubseteq \forall R.C_i$		RangeProperty(ObjectProperty, Class)
$R \equiv R_0^-$	$-_R$	InverseOf(ObjectProperty$_1$ ObjectProperty$_2$)
$R \equiv R^-$	κ_R	SymmetricProperty(ObjectProperty)
$T \sqsubseteq \leq 1R$		FunctionalProperty(ObjectProperty)
$T \sqsubseteq \leq 1R^-$		InverseFunctionalProperty(ObjectProperty)
$Tr(R)$		TransitiveProperty(ObjectProperty)
$R_1 \sqsubseteq R_2$	$\leq_R$	InheritanceObjectPropertyLink(ObjectProperty$_1$, ObjectProperty$_2$)
$R_1 \equiv \ldots \equiv R_n$	ε_R	**EquivalentProperty(ObjectProperty$_1$,. . .,ObjectProperty$_n$)**
$A \sqsubseteq A_i$	A	DatatypeProperty(DatatypeProperty)
$\geq 1A \sqsubseteq C_i$	ρ_A	DomainProperty(DatatypeProperty Class)
$T \sqsubseteq A.T_i$		RangeProperty(DatatypeProperty, Datatype)
$T \sqsubseteq \leq 1A$	κ_A	FunctionalProperty(DatatypeProperty)
$A_1 \sqsubseteq A_2$	$\leq_A$	InheritanceDatatypePropertyLink(DatatypeProperty$_1$, DatatypeProperty$_2$)
$A_1 \equiv \ldots \equiv A_n$	ε_A	**EquivalentProperty(DatatypeProperty$_1$,. . .,DatatypeProperty$_n$)**
$I \in C_i$	ι_C	InstancesOf(Instance, Class)
$\{I, I_1\} \in R_i$	ι_R	InstancesOfObjectProperty(Instance, Instance$_1$ ObjectProperty)
$\{I, V_1\} \in A_i$	ι_A	InstanceOfDatatypeProperty(Instance, Datavalue, DatatypeProperty)
$\{I_1\} \equiv \ldots \equiv \{I_n\}$	ε_I	**SameAs(Instance$_1$,. . .,Instance$_n$)**
$\{I_i\} \sqsubseteq \neg\{I_j\}$, $i \neq j$	δ_I	**DifferentFrom(Instance$_1$,. . .,Instance$_n$)**

Annex 3. $\mathcal{SHOJN}(\mathcal{D})$ change management definitions

In order to represent changes, we give the seven definitions below.

Definition 2. *Change*. A change m is the application of a modification on an ontology O, that potentially affects one or more elements of its structure as defined by the $\mathcal{SHOJN}(\mathcal{D})$ ontology model.

Definition 3. *Log of changes*. Given an ontology O a log of changes, noted log, is defined by an ordered set of changes (simple and complex) $<\omega_1, \ldots, \omega_n>$ that applied to O results in O.

Like in [5], 2 change types are distinguished: basic and complex.

Definition 4. *Basic change*. A basic change on an ontology O is a function $\omega_B : sK \rightarrow 2^O$ with $sK := \{sC \cup sI \cup sR \cup sA\}$ that corresponds to an addition, a removal of a modification of one element E O.

Definition 5. *Complex change*. A complex change on an ontology O is a disjoint union of basic changes. It is a function $\omega_C : nsK \rightarrow 2^O$ such that $\omega_C := \omega_{BI} + \ldots + \omega_{Bn}$.

The application of a change on an ontology, basic or complex, can be an addition or a deletion. It is traced as such in the log of changes.

Definition 6. *Addition of a change*. The addition of a change ω_i traced in the log of changes log_i, noted $log_i + \{\omega_i\}$, is defined by the disjoint union between the two disjoint sets *logi* and $\{\omega_i\}$.

Definition 7. *Deletion of a change*. The deletion of a change ω_i traced in the log of changes log_i, noted $log_i - \{\omega_i\}$, is defined by the set-theoretic complement such that $\log_i - \{\omega_i\} = \{x \in \log_i | x \notin \{\omega_i\}\}$.

Annex 4. Structural consistency change interdependencies

Table 2 shows the interdependencies between basic changes organized as a dependency matrix. The value x of an element, i.e. $dependency[i][j] = x$, indicates that the application of a change related to the row i induces a change related to the column j with the corresponding element to maintain ontology structural consistency.

In terms of change application, a change ω_i^B has to be applied only after all changes ω_j^B with $j \geq 1$ and $j \leq n$ for which $dependency[i][j] = x$ are firstly applied.

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

Table 2
Structural dependency matrix of concepts, data types, roles, attributes, instances and data values deletions basic changes and other basic changes for a $\mathcal{SHOIN}(\mathcal{D})$ ontology.

−	IntersectionOf(Class$_1$,…,Class$_n$)	UnionOf(Class$_1$,…,Class$_n$)	ComplementOf(Class$_1$, Class$_2$)	OneOf(Class, Instance$_1$,…,Instance$_n$)	SomeValuesFrom(ObjectProperty, Class)	AllValuesFrom(ObjectProperty, Class)	HasValue(ObjectProperty, Instance)	MinCardinalityProperty(ObjectProperty, n)	MaxCardinalityProperty(ObjectProperty, n)	SomeValuesFrom(DatatypeProperty, Datatype)	AllValuesFrom(DatatypeProperty, Datatype)	HasValue(DatatypeProperty, Datavalue)	OneOf(Datavalue$_1$,…,Datavalue$_n$)	IntersectionClass(Class1,…Classn))	EnumeratedClass(Instance1,…Instancen)	SubClassOf(Class1, Class2)	EquivalentClass(Class$_1$,…Class$_n$)	DisjointClass(Class$_1$, Class$_2$)	SubDatatypeOf(Datatype$_1$, Datatype$_2$)	InstancesOfDatatype(Datavalue, Datatype)	DomainProperty(ObjectProperty, Class)	RangeProperty(ObjectProperty, Class)	InverseOf(ObjectProperty, ObjectProperty$_2$)	SymmetricProperty(ObjectProperty)	FunctionalProperty(ObjectProperty)	InverseFunctionalProperty(ObjectProperty)	TransitiveProperty(ObjectProperty)	InheritanceObjectPropertyLink(ObjectProperty$_1$, ObjectProperty$_2$)	EquivalentProperty(ObjectProperty$_1$,…,ObjectProperty$_n$)	DomainProperty(DatatypeProperty Class)	RangeProperty(DatatypeProperty, Datatype)	FunctionalProperty(DatatypeProperty)	InheritanceDatatypePropertyLink(DatatypeProperty$_1$, DatatypeProperty$_2$)	EquivalentProperty(DatatypeProperty$_1$,…,DatatypeProperty$_n$)	InstancesOf(Instance, Class)	InstancesOfObjectProperty(Instance, Instance$_1$, ObjectProperty)	InstanceOfDatatypeProperty(Instance, Datavalue, DatatypeProperty)	SameAs(Instance$_1$,…,Instance$_n$)	DifferentFrom(Instance$_1$,…,Instance$_n$)
Class(Class)	x	x	x	x	x	x				x	x			x	x	x	x	x		x	x																	
Datatype(Datatype)										x	x								x	x														x					
ObjectProperty(ObjectProperty)					x	x	x	x	x												x	x	x	x	x	x	x	x								x			
DatatypeProperty(DatatypeProperty)										x	x																			x	x	x	x	x			x			
Instance(Instance)				x			x								x																			x	x	x	x	x	
Datavalue(Datavalue)												x	x																							x			

References

[1] R. Djedidi, M.A. Aufaure, Change management patterns for ontology evolution process, in: IWOD at ISWC 2008, Karlsruhe, 2008.
[2] N. Guarino, Formal ontology, conceptual analysis and knowledge representation, International Journal of Human-Computer Studies 43 (5) (1995) 625–640.
[3] T.R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (2) (1993) 199–220.
[4] G.M. Flouris, Ontology change: classification & survey, Knowledge Engineering Review 23 (2) (2008) 117–152 (C.U. Press).
[5] M.C. Klein, Change Management for Distributed Ontologies, 2004.
[6] P. Haase, Stojanovic, Consistent evolution of OWL ontologies, Semantic Web: Research and Applications (2005) 182–197.
[7] N.F. Noy, Ontology Evolution: Not the Same as Schema Evolution, Stanford Medical Informatics, Stanford University, 2004.
[8] H.S. Pinto, DILIGENT: towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies, ECAI, vol. 16, 2004, p. 393.
[9] L. Stojanovic, Methods and Tools for Ontology Evolution, 2004.
[10] P.D. Plessers, Ontology change detection using a version log, in: Y. Gil, E. Motta, V. Richard Benjamins, M.A. Musen (Eds.), 4th International Semantic Web Conference, Springer-Verlag, Galway, Ireland, 2005, pp. 578–592.
[11] P.H. Luong, Gestion de devolution d'un Web semantique d'entreprise, (É.N. Doctoral dissertation), 2007.
[12] R. Djedidi, Approche devolution d'ontologie guidee par des patrons de gestion de changement, 2009.
[13] M.G. Hartung, COnto-Diff: generation of complex evolution mappings for life science ontologies, Journal of Biomedical Informatics 46 (1) (2013).
[14] M.A. Javed, Ontology change management and identification of change patterns, Journal on Data Semantics 2 (2–3) (2013).
[15] W. Jaziri, A methodology for ontology evolution and versioning, in: IEEE Advances in Semantic Processing, SEMAPRO'09, 2009, 15–21.
[16] C.D. Rogozan, Gestion de l'évolution des ontologies: methodes et outils pour un referencement semantique evolutif fonde sur une analyse des changements entre versions d'ontologie, (U.d. Doctoral dissertation), 2009.
[17] Z.C.-G. Sellami, DYNAMO-MAS: a multi-agent system for ontology evolution from text, Journal on Data Semantics 2 (2–3) (2013).
[18] L. Stojanovic, Ontology evolution within ontology editors, in: OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management, CEUR-WS.org, Siguenza, (2002), pp. 53–62.
[19] P. Pittet, OntoVersionGraph: A Change Management Methodology dedicated to Formal Ontologies and their User Views in a Collaborative Context: Application to SHOIN(D) Ontologies, Le2i, University of Burgundy, Dijon, Burgundy, France, 2014.
[20] M.P. Gueffaz, Inconsistency identification in dynamic ontologies based on model checking, in: INSTICC, ACM SIGMIS, 2012, 418–421.
[21] P.N. Pittet, Guidelines for a Dynamic Ontology-Integrating Tools of Evolution and Versioning in Ontology, 2012.
[22] J.K. Eder, Modelling changes in ontologies, in: On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, Springer, Berlin, Heidelberg, 2004, pp. D662–D673.
[23] P.D. Plessers, Understanding ontology evolution: a change detection approach, Web Semantics: Science, Services and Agents on the World Wide Web 5 (1) (2007).
[24] R. Vanlande, C. Nicolle, C. Cruz, IFC and building lifecycle management, Automation in Construction 18 (1) (2008) 70–78.
[25] R. Rajugan, E. Chang, T.S. Dillon, Ontology views: a theoretical perspective, in: On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Springer, Berlin Heidelberg, 2006, pp. 1814–1824.
[26] R. Volz, D. Oberle, R. Studer, Implementing views for light-weight web ontologies, in: Database Engineering and Applications Symposium, 2003. Proceedings. Seventh International, IEEE, 2003, pp. 160–169.
[27] M.H. Ehrig, Similarity for ontologies – a comprehensive framework, in: Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, at PAKM, 2004.

Perrine Pittet is a PhD student at the laboratory LE2I (Electronics, Image and Computer Science) since January 2011 & Teacher at the University Institute of Technology of Dijon (France) since December 2010. She obtained a Master in Information Technologies and Web of Lyon 1 in June 2010. Her research focuses on change management in OWL DL ontologies.She specifically works on the design of a methodology for versioning and evolution of ontology providing a Java API to guide the changes on the ontology and manage its consistency in its different versions and views.

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006

Christophe Cruz is associate professor at the IUT of Dijon (Technological Institute of the University of burgundy). He obtained a PhD in Computer Science at the University of Burgundy in 2004. He works on ontology modelisation.

Christophe Nicolle is professor in the Computer Science department at the University of Bourgogne. He received his Pd.D in Computer Science in 1996. Since, he is a member of the LE2I laboratory (Electronics, Informatics and Image) at the University of Bourgogne. His research interests include interoperability of heterogeneous information systems and the optimization of process and resources using semantics, combinatory and logical rules. Since 2001, he works on the Active3D project dedicated to the development of a semantic framework for the management of buildings during their lifecycle. In 2005, he participated in the creation of the Active3D Company. The company develops a web collaborative platform for facility management. Currently, the Active3D platform manages more than 60 millions of square meters of buildings.

Please cite this article in press as: P. Pittet, et al., An ontology change management approach for facility management, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.07.006