Towards an ontology for product version management

María Soledad Sonzini*

INGAR – Instituto de Desarrollo y Diseño (CP 3000),
Avellaneda 3657, Province of Santa Fe, Argentina
and
Universidad Nacional de La Rioja (CP 5300),
Luis M. de la Fuente S/N, Province of La Rioja, Argentina
Email: ssonzini@santafe-conicet.gob.ar
*Corresponding author

Marcela Vegetti and Horacio Leone

INGAR – Instituto de Desarrollo y Diseño (CP 3000),
Avellaneda 3657, Province of Santa Fe, Argentina
Email: mvegetti@santafe-conicet.gob.ar
Email: hleone@santafe-conicet.gob.ar

Abstract: During its lifecycle, products are affected by market, technology, and user requirements. Without a process for efficiently handling product changes, product data, which is spread in different areas and systems, might become unusable, incomplete, or inconsistent. A simple change on product information may trigger a domino effect that could be very difficult to control. In order to reduce this effect, knowledge is important to answer what, when, why, and how a change occurred. This article proposes an ontology that allows capturing product changes in order to answer the aforementioned questions. The proposed ontology extends PRoductONTOlogy (PRONTO) (Vegetti et al., 2011) to represent product family changes. OWL implementation of the proposed ontology is presented with a simple case study to validate it.

Keywords: product lifecycle management; PLM; ontology; product version; product ontology; control version; engineering change management; ECM; product information; ontology web language; OWL; bill of materials; BOM; semantic web rule language; SWRL.

Reference to this paper should be made as follows: Sonzini, M.S., Vegetti, M. and Leone, H. (2015) 'Towards an ontology for product version management', *Int. J. Product Lifecycle Management*, Vol. 8, No. 1, pp.80–97.

Biographical notes: María Soledad Sonzini is a graduate in Information Systems Engineering of Universidad Nacional de La Rioja, La Rioja, Argentina. She is currently a PhD student in Information Systems Engineering of Universidad Tecnológica Nacional (UTN). She also works at Instituto de Desarrollo y Diseño (INGAR) on a Research Fellowship granted by Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) (*Argentine Council for Scientific and Technical Research of Argentina*). Her research interests are focused on applying ontologies and semantic web technologies to represent product information in manufacturing industries.

Marcela Vegetti is an Associate Professor at the Information System Engineering Department, Facultad Regional Santa Fe, Universidad Tecnológica Nacional, Santa Fe, Argentina. She also holds an Assistant Researcher position at *Instituto de Desarrollo y Diseño* of CONICET. She obtained her PhD in Information Systems Engineering at 'Universidad Tecnológica Nacional', Santa Fe, Argentina in 2007. Her current research activities are focused on ontologies and semantic technologies for supply chain information systems and enterprise modelling.

Horacio Leone is a Full Professor at the Information System Engineering Department, Facultad Regional Santa Fe, Universidad Tecnológica Nacional, since 1989. He also has an Independent Researcher position at Instituto de Desarrollo y Diseño of CONICET of Argentina. He received his Chemical Engineer degree from Universidad Tecnológica Nacional in 1981. He obtained his PhD in Chemical Engineering at Universidad Nacional del Litoral in 1986. From 1986 to 1989, he was a Postdoctoral Fellow at the Laboratory for Intelligent Systems in Process Engineering (LISPE) at Massachusetts Institute of Technology. His research interests are focused on improving the understanding of the engineering design process, mainly through the application of different modelling frameworks to diverse engineering domains. In software architecture design process, he has worked on characterising design decisions and the rationale behind them. He has also explored semantic web applications to supply chain information systems and enterprise modelling.

1 Introduction

In the past decade, vast research has been increasingly focused on product lifecycle management (PLM). This discipline is developed within the industrial environment and its objective is sharing and managing product information during product lifecycle.

PLM requires robust solutions to represent product data models, enabling information exchange across different organisations, stakeholders, processes, stages in the product lifecycle, and several business activities. In recent years, research has shifted its course to include ontology technology with the aim of holistically integrating product information representation. This integration should consider changes in domain and user requirements that naturally occur for several reasons such as global competition, advances in technology, and demands for customised products.

During product lifecycle, the above mentioned changes should be reflected in the product information in order to improve efficiency in lifecycle phases and to shorten product development times. Without a process for efficiently handling changes, product data might become unusable, incomplete, or inconsistent. This management process will enable assessing the impact upon the affected components and making scheduling and design decisions. Sjoberg (1995) considers that in order to obtain a sophisticated maintenance tool and predict change consequences, it is necessary to take into account a set of questions about the change event, which are formulated as follows:

- What were the affected components?
- When did it occur?

- Why did it occur?
- How did it change?
- How was the change register?

It is widely accepted that ontologies constitute an enabling technology for information management through a commonly understandable semantic representation of domain-specific information, which can be communicated between applications (Gruninger, 2004). Therefore, even though several ontologies for product information modelling have been proposed in the last decade, few contributions take into account the product change management issue.

One of the proposed ontologies is called PRoductONTOlogy (PRONTO) (Vegetti et al., 2011). This ontology provides a product conceptual model that introduces two hierarchies to consistently represent abstract and structural product information. In addition, PRONTO allows representing aggregation and disaggregation processes to obtain a different bill of materials (BOM) or product configuration description. However, PRONTO lacks the ability to represent control and change management of a product.

In order to fill this gap, this proposal extends PRONTO by introducing new entities and relationships in order to capture changes and evolution of products. The proposal is able to show how a change affects a specific family and allows for gathering answers to the aforementioned questions.

This work is organised in sections. Section 2 provides an overview of the principles of change management engineering, change control within a demanding environment, and the existing product ontologies. The proposal of a conceptual model extending PRONTO for change management and its implications is defined in Section 3. Then, Section 4 discusses the implementation of the proposed model in ontology web language (OWL). Moreover, a simple case study is presented in Section 5 to show the proposal application. Finally, conclusions are outlined in Section 6.

2 Related works

This section describes several topics necessary to understand the present proposal. First, it describes some existing proposals for engineering change management (ECM) in manufacturing industries. Then, it introduces some ontologies that are proposed to represent product information. At the end, a brief description of PRONTO is included.

2.1 Background of ECM

Product manufacturing industry undergoes changes caused by several factors such as new technologies and design decision components that are withdrawn from the market, among others. A change in the structure of a product may affect another aspect such as schedule or project budget. Propagation of product changes should receive special treatment, particularly in products with complex structures. Thus, Jarratt et al. (2004) consider that companies must adopt new methodologies to survive in the market and improve their product design processes through effective change management. To take up this challenge, they propose ECM process as "the process of making modifications and alterations to parts, software, and drawing that have been already released at any point in the product lifecycle". This process consists of five steps, the evaluation of possible impacts of a change being the most critical. ECM process can affect two features: the product itself or the development process. In this paper, we focus on the former, considering the impact upon product information.

Early studies on computer tools for ECM have focused on stand-alone computer-aided systems for storing and retrieving engineering change documents. Then, ECM systems based on workflow systems have been proposed (Huang and Mak, 1998; Huang et al., 2001) to provide a wide set of functionalities for supporting the entire engineering change process. There are also several knowledge management systems for product development that are intended to capture process knowledge and share product data (May et al., 2000; May and Carter, 2001; Monplaisir, 1999; Numata and Taura, 1996; Ramesh and Tiwana, 1999; Yoo and Kim, 2002).

However, these proposals are still limited in their ability to capture and reuse the knowledge involved in engineering changes. Therefore, engineering change teams currently depend heavily on off-line collaboration without codifying knowledge explicitly. In order to overcome this difficult, Lee et al. (2006) introduce the use of ontology in their proposal.

Today, there are several commercially available tools to support computer-based ECM, especially tools as part of enterprise resource planning (ERP), PLM, or product data management (PDM) to track and register changes. As an extension of PLM systems, Horváth (2007) proposes a new method for human-computer communication for decision-making in engineering based on change management for product development in modelling environments. However, any of these proposals use ontologies to formalise product change information, which would allow inferring answers to the questions mentioned in Section 1.

2.2 Ontologies for product data representation

In the last decade, ontologies have been proposed as a mechanism to support semantic integration in the context of semantic web (Shadbolt et al., 2006). Thus, ontologies are defined to establish a common vocabulary among areas within an organisation, different organisations, and various applications. Ontology is a formal model which explicitly represents the consensual knowledge of a domain (Brandt et al., 2008).

There are principles, design criteria, and stages for ontologies development. For instance, Henning (2012) describes a methodology for deploying an ontology including four stages, which are summarised as follows:

1 requirements specification stage to identify the ontology aim

2 conceptualisation stage to semi-formalise the previous stage and obtain a view of the domain

3 implementation stage, in which the ontology is codified by a formal language

4 evaluation stage to make a judgment of usefulness and ontology quality with respect to the stage number 1.

These stages are not sequential; in contrast, the ontology development is an iterative and incremental process. This work implements stage 2 by a conceptualisation of the proposal for change management in Section 3. In Section 4, the proposed ontology is implemented using OWL (stage 3), and the proposal evaluation is established by the solution of the set of already mentioned question (Section 5).

Ontologies have been also developed for manufacturing environments. An important proposal is Toronto virtual enterprise (TOVE) project (Fox, 1992), which defined a set of ontologies to be used in the representation of enterprises. One of such ontologies introduced product related concepts. However, it adopted a traditional representation of product variants, which are not efficient enough and only consider products that are obtained by assembling component parts.

In nowadays electronic commerce context, business-to-business (B2B) applications require effective communication among computers. This led to the emergence of some standards which improve product information exchange among suppliers and customers. As an example, it is possible to mention product data representation and exchange (STEP) model (ISO 10303, 1991), United Nations Standard Products and Services Code (UNSPSC), NAICs (North American Industry Classification System), eCl@ss, Electronic Open Technical Dictionary (eOTD) or RosetaNet's technical dictionary. Such standards are agreed by a wide group of organisations. However, they only represent concept taxonomies and do not allow for modelling product structures. In the last years, some proposals (Klein, 2002; Hepp, 2006; Zhao and Liu, 2008) arose to codify such standards into languages for Semantic Web.

Hepp (2008) introduces an ontology that can be used for describing products and services offered on the web. It is focused on the representation of web resources, offers made by means of those web resources legal entities, prices, terms, and conditions. Yang et al. (2009) successfully developed a model of product configuration knowledge in which structural knowledge is represented in OWL and constraint knowledge is described in semantic web rule language (SWRL). Matsokis and Kiritsis (2010) also use OWL in their proposal of an ontology for representing documents, resources, and activities in PLM.

Semantic web-based open engineering platform (SWOP) project proposes a semantic product modelling approach to define four ontologies: product, representation, rule, and operation, which are collectively called product modelling ontology (PMO) (Böhms et al., 2008). On the one hand, PMO states the concepts using representation of product geometry. On the other hand, such ontology allows representing the relation between a product and its constituent parts.

PRONTO is proposed by Vegetti et al. (2010, 2011) to be used as a common vocabulary in order to reach the semantic integration of product information systems. It is based on both the generative BOM and the product family concepts in order to provide an efficient management of multiple variants. The proposal modifies the traditional two-level product family representation (product family – family member) by adding a new level between them. The new three-level hierarchy approach allows the natural modelling of product information with different aggregation degrees, which are needed for planning activities taking place at various time horizons.

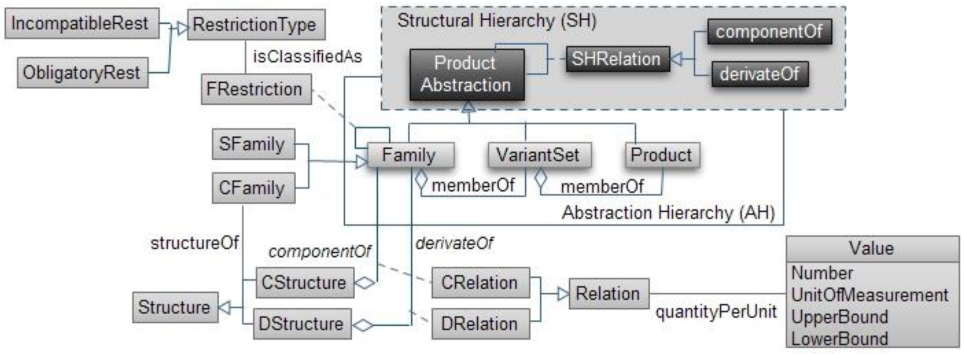
As it is presented in previous paragraphs, there are several ontologies capable of representing different aspects of product data. Some of them are focused on representing product structure (Böhms et al., 2008; Zhao and Liu, 2008, Fox, 1992; Hepp, 2008), product catalogues (Klein, 2002; Hepp, 2006), and design documents (Matsokis and

Kiritsis, 2010), among others. Even though there exist many ontologies for product information modelling, few contributions take into account the product change management problem.

2.3 PRoductONTOlogy

PRONTO is a general ontology that could be extended to represent product in different industrial domains. The concepts that are relevant to this paper are illustrated in Figure 1. PRONTO introduces two hierarchies to represent product information: abstraction hierarchy (AH) and structural hierarchy (SH). The former organises product information at three abstraction levels.

Figure 1 Simplified conceptual model of PRONTO (see online version for colours)



The highest one, *Family* level, represents a set of similar products that have similar structures, characteristics, and production routes. A family is related to a superstructure, which includes all possible structures that the members of such a family may have. The second level, called *VariantSet*, models a subset of *Family* members that share the same structure and/or similar characteristics, i.e., a subfamily. Hence, a *VariantSet* is a *memberOf* a *Family*. Thus, when a *VariantSet* is specified, one of the structures contained in Family´s superstructure must be selected. In addition, certain modifications and constraints can be introduced into that structure. *Product* is the lowest level and represents individual items with physical existences which are members of a particular *VariantSet*. Therefore, all products associated with a given *VariantSet* have the same structure, which is that defined for such *VariantSet*. Minor modifications in some parameter values (e.g., flavour, colour, etc.) can be also introduced at this lower level.

SH organises the specific knowledge about product structural information. This hierarchy is a tool to handle product information associated with the multiple available recipes or processes for manufacturing a particular product or a set of similar products.

PRONTO allows representing BOMs of products that are manufactured by assembling component parts or disaggregating non-atomic raw materials, and products with hybrid BOMs. Therefore, SH considers two types of structural hierarchies: one which relates a product with its component parts and another one which links a product with its derivative constituents. The relationships that are used to represent each of these types are called *componentOf* and *derivativeOf*, respectively. As shown in Figure 1, both relations are a specialisation of *SHRelation* class, which links a *ProductAbstraction* instance (*whole*) with zero or more *ProductAbstraction* instances (*part*) defined at the same abstraction level. The family concept is specialised into simple family (*SFamily*) and composite family (*CFamily*) in order to denote families without structures and families having one or more structures. Composite family may represent products that:

1 are manufactured by assembling parts (typical of discrete manufacturing environments)

2 are decomposed/disaggregated to obtain intermediate products (characteristic of dairy, meat, or petrochemical industries), which can participate as components of other products

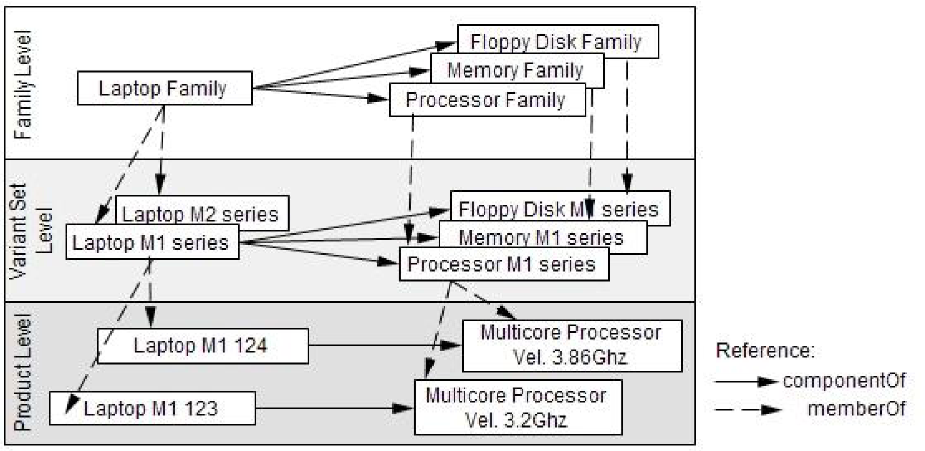
3 have hybrid structures.

Hence, PRONTO considers two types of structures and relations: composition relation (*CRelation*) to identify the component of the composite structure (*CStructure*) and decomposition relation (*DRelation*) to identify the derivatives of decomposition structure (*DStructure*). Each one is linked to the *Value* concept by *quantityPerUnit* relationship, which represents the amount of the part that is required to produce one unit of the whole. This concept specifies a numerical value (*number*), the unit in which the number should be measured (*UnitOfMeasurement*), and boundaries (*upperBound* and *lowerBound*) that constraint the numerical value.

Furthermore, constraints associated with the specification of valid products are introduced in PRONTO. This is a very important feature in production environments where customer specifications have a strong influence on defining the products to be manufactured/ assembled. Thus, it prevents a customer from requiring an incorrect product configuration. Due to the scope of this work, Figure 1 only shows restrictions related to Family level (*FRestriction*). Each restriction is identified as obligatory (*ObligatoryRest*) or incompatible (*IncompatibleRest*). The former forces a component to participate in a SH. In contrast, incompatible restrictions exclude a component from the SH.

Figure 2 illustrates the application of both hierarchies in a computer industry example. It shows the abstraction hierarchies of laptop product line, as well as the ones corresponding to the components needed for its manufacture (*FloppyDisk*, *Memory* and *Processor* families). Dashed lines between entities located at different levels represent *memberOf* associations that are comprised in the different abstraction hierarchies. On the other hand, structural hierarchies (solid lines) of *Laptop* family, *Laptop M1 series* variant set, and *Laptop M1 124* products are included at *Family*, *VariantSet*, and *Product* levels, respectively. These solid lines represent *componentOf* relationships that are defined among entities belonging to the same abstraction level. It can be seen that at the Family level, *FloppyDisk*, *Memory*, and *Processor* families are components of *Laptop* family. At the middle level, *LaptopM1* series variant set is composed of a *FloppyDisk M1serie* floppy disk (member of *FloppyDisk Family*), a *Memory M1 series* memory (member of *Memory Family*), and a *Processor M1 series* processor (member of *Processor Family*). Similarly, at the lowest level, the particular components of *laptop M1 124* product, which is a member of *Laptop M1* series variant set, are shown.

Figure 2 Laptop family example



However, PRONTO lacks the mechanism to represent product change management. Change management is an important activity to maintain the visibility of product information evolution; and it could be useful to trace product versions during product lifecycle and make several decisions.

3 Ontology for product version management

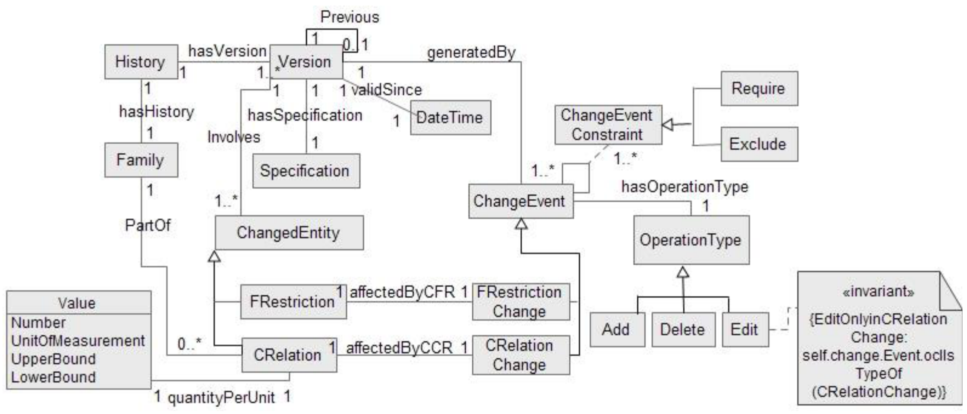
The lack of change management in PRONTO impacts directly on product family information since a change occurrence could corrupt the original structure. This section presents the concepts and relations that should be added to PRONTO in order to capture, trace product versions, and answer the set of above mentioned questions. The scope of the model is applied in the highest level of PRONTO AH.

During product lifecycle, product family structure may vary due to external factors and design decisions. In the example of computer machine industry domain (Section 2.1), a professional designer makes a decision about removable data storage. Initially, a *Laptop Family* is composed of a *Floppy Disk Family*. But an external factor such as technology advances leads to redesigning the family information, adding a new component such as *USB Drive Family*, and deleting *Floppy Disk Family*.

The proposed model, which is introduced in Figure 3, represents each change affecting a product family by means of the *ChangeEvent* concept. The occurrence of a *ChangeEvent* generates a new family version. It would be important to capture all versions of a specific family. Therefore, the conceptual model includes the *History* concept to represent all changes undergone by a family during its lifecycle. Family *History* is the set of family versions own by a *Family*. Each *Version* is linked to *DateTime* entity which specifies the time and date when the version became valid (see *validSince* relation in Figure 3). Moreover, each *Version* can be linked to its predecessor version by *Previous* association. Only the first version is not related to another.

Specification entity allows capturing more information about the creation of the version, e.g., causes of change, designer responsible of the version, etc. As already mentioned, a *Version* is generated by the occurrence of a *ChangeEvent* concept (see *generatedBy* association in Figure 3) and the elements affected by *Involves* relationship are associated to that version.

Figure 3 Conceptual model for product version management



Within the PRONTO concepts explained in Section 2.3, there are three elements that can be affected by a change:

1 one of the family structures, through the addition or removal of a Relation in the structure

2 the *quantityPerUnit* value of a relation

3 a constraint between families.

Therefore, the model introduces the *ChangedEntity* concept to represent the element that may be affected by a change event. At family level, particularly, the entities that could be affected by a change are *CRelation* and *FRestriction*.

In order to represent the different possible changes, the ontology specialises *ChangeEvent* concept into: *FRestrictionChange* and *CRelationChange*. The former represents the change of a family restriction and the latter is used for changes in relations as a whole and in its *quantityPer* attribute. These kinds of change events are associated to the restriction or the relation affected by the change respectively (see *affectedByCFR* and *affectedByCCR* in Figure3). Both change event types must be conformed according to certain restrictions. In particular, *CRelationChange* is associated to a constraint to ensure their only application in a *CFamily*, since a *SFamily* has not a structure. In contrast, *FRestriction* can be applied in both *CFamily* and *SFamily*.

Each *ChangeEvent* might affect a *ChangeEntity* by three different operations. Shaban-Nejad and Haarslev (2009) distinguished 74 different types of operations that frequently occur in bio-ontologies life cycles, represented by ten general terms. Within the scope of this work, only three of these types are adopted: *Add*, *Delete*, and *Edit*. A change event is related to a type of operation by *hasOperationType* association. *Add* operation allows the addition of a restriction (*FRestriction*) or a relation (*CRelation*) in a product family definition. Likewise, *Delete* operation allows the removal of a restriction or a relation. Nevertheless, not all operations can be applied to all affected components. A *FRestriction* may be affected by an *Add* or *Delete* operation, but it cannot be edited. In contrast, an instance of *CRelation* could be affected by the three defined operation types.

It is possible to *Add* or *Delete* a *CRelation* as well as to edit the *Value* associated to it by the *quantityPerUnit* link. In other words, product designers can add or delete a *FRestriction* between families or a relation in one family structure. However, the value of *quantityPer* attribute of a relation can be only edited by modifying its number, unit of measurement, or its bounds. Therefore, the definition of a restriction is necessary in order to constrain the proposed model. This restriction is represented in Figure 3 by a note associated to *Edit* entity.

The ontology defines a *ChangeEventConstraint* entity, which describes a dependency relationship between two change events. A specific change event may *Requires* or *Excludes* the consideration of another *ChangeEvent*, e.g., a change event which specifies the removal of a certain restriction may be needed to define a new (*Add*) restriction. Constraints are necessary to maintain consistency and the correct interpretation of the conceptual model.

The proposed conceptual model can answer the aforementioned set of questions. To answer the first question: 'What were the affected components?', the proposal defines *ChangedEntity* and its specialisation concepts. Likewise, to answer 'How did it change?', the model defines two concepts: *CRelationChange* and *FRestrictionChange* to identify the kind of the affected entity, and *OperationType* concept to specify the type of change. *AffectedByCCR* and *affectedByCFR* relationships allow obtaining the connection between *EntityChangeEvent* and *ChangeEvent*. In turn, *DateTime* and *Specification* concepts allow for responses to 'When did it occur?' and 'Why?', respectively. Finally, to answer 'How was it registered?', the model defines a *History* concept to capture all versions of a *Family* in its lifecycle, and a *Version* concept to register all information about the occurring change.

4 OWL implementation

This section presents the implementation of the conceptual model described in Section 3 by using OWL. For this implementation, Protégé 3.5 and Pellet 1.5.2 reasoner are chosen. First, the identified concepts were classified and organised to represent a hierarchical structure. Then, data types as well as properties and their domain and Range were defined (see Table 1). Each property has a domain, which consists of a list of classes of the individuals that can be placed on the left hand side of the property, and a range, which consists of the list of classes of individuals that can be placed in the right hand side of the property. Domain and range are used for reasoning or inferred new knowledge. After that, the class expression and asserted conditions were specified. Asserted conditions ensure consistency of product models based on the proposed ontology. In particular, the following assertions were defined as necessary conditions: 'a *ChangeEvent* must be associated to at least one *OperationType*', 'a *Family* class must be associated to a *History* class', and '*Edit* concept is associated only to a *Value* concept'.

In order to define the behaviour and semantics of relationships, SWRL is used. SWRL enables acquiring more powerful deductive reasoning. A SWRL rule contains an antecedent part, which is referred to as body, and a consequent part, which is referred to as head. Both, body and head consist of positive conjunctions of atoms. Informally, a SWRL rule may be read to mean that if all atoms in the antecedent are true, then the consequent must be also true. Predicate symbols can include OWL classes, properties, or data types. Arguments can be OWL individuals or data values, or variables referring to them. All variables in SWRL are treated as universally quantified, their scope being limited to a given rule.

Table 2 shows some of the defined rules to infer knowledge. By means of these rules, it is possible to infer, for example, inverse functional properties (rules 1 to 7), the relation value that is edited by a change (rule 8), families that were deleted in a specific version (rule 9), or new restrictions added to a family in a specific version (rule 10).

Table 1 Set of properties defined in OWL

Property	*Domain*	*Range*
affectedByCCR	CRelation	CRelationChange
affectedByCFR	FRestriction	FRestrictionChange
constraintTo	ChangeEventConstraint	ChangeEvent
editValue	Edit	Value
GeneratedBy	Version	ChangeEvent
hasConstraint	ChangeEvent	ChangeEventConstraint
hasHistory	Family	History
hasOperationType	ChangeEvent	OperationType
hasSpecification	Version	Specification
hasVersion	History	Version
Involves	Version	ChangeEntity
isCEventOf	ChangeEvent	Version
isEditedBy	Value	Edit
isOperationTypeOf	OperationType	ChangeEvent
PartOf	CRelation	Family
quantityPerUnit	CRelation	Value
ValidSince	Version	DateTime

Table 2 Partial set of rules defined in SWRL

	Rules	*Explanations*
1	Family(?*x*) ∧ History(?*y*) ∧ hasHistory(?*x*, ?*y*) → isHistoryOf(?*y*, ?*x*)	*x* is a family associated to a history *y* and *y* is history of family *x*.
2	History(?*x*) ∧ Version(?*y*) ∧ hasVersion(?*x* ?*y*) → isVersionOf(?*y*, ?*x*)	*x* is a history and y is a version. *x* has a version *y*. *y* is version of *x*.
3	Version(?*x*) ∧ ChangedEntity(?*y*) ∧ involves(?*x*, ?*y*) → isInvolveIn(?*y*, ?*x*)	*x* is a version and involves an Entity changed *y*. Hence, *y* is involved in *x*.
4	Version(?*x*) ∧ ChangeEvent(?*y*) ∧ GeneratedBy(?*x*, ?*y*) → isCEventOf(?*y*, ?*x*)	*x* is a version and is generated by a ChangeEventy. Hence *y* is Change Event of a version *x*.
5	Version(?*x*) ∧ ChangeFRestriction (?*y*) ∧ FRestriction (?*z*) ∧ GeneratedBy(?*x*, ?*y*) ∧ Involves(?*x*, ?*z*) → affectToFR(?*y*, ?*z*)	A change *y* generates a version *x* and affects a FRestriction *z* involved in *x*. This rule is similar to affectToCR.

Table 2 Partial set of rules defined in SWRL (continued)

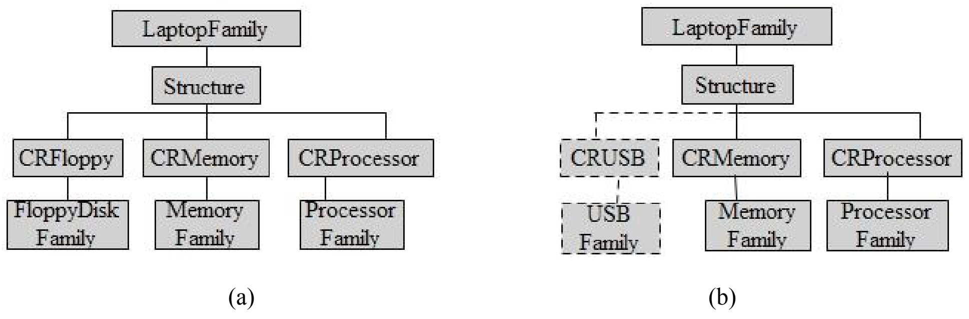
	Rules	*Explanations*
6	Version(?*x*) ∧ ChangeFRestriction (?*y*) ∧ FRestriction (?*z*) ∧ GeneratedBy(?*x*, ?*y*) ∧ Involves(?*x*, ?*z*) → affectedByCFR(?*z*, ?*y*)	A change *y* generates a version *x* and affects a FRestriction *z* involved in *x*. This rule is similar to affectedByCCR.
7	ChangeEvent(?*x*) ∧ ChangeConstraint(?*y*) ∧ hasConstraint(?*x*, ?*y*) → ConstraintTo(?*y*, ?*x*)	*x* is a change event and it has a constraint *y*.
8	ChangeRelation(?*x*) ∧ OperationType(?*y*) ∧ hasOperation(?*x*, ?*y*) ∧ Value(?*u*) ∧ Edit(?*t*) ∧ CRelation(?*z*) ∧ affectedByCCR(?*z*, ?*x*) ∧ *quantityPerUnit*(?*z*, ?*u*) → editValue(?*t*, ?*u*)	A ChangeRelation *x* applies an edit operation to value *u*.
9	Family(?*f*) ∧ History(?*h*) ∧ Version(?*v*) ∧ CRelation(?*e*) ∧ Involves(?*v*, ?*e*) ∧ Family(?*af*) ∧ partOf(?*e*, ?*af*) ∧ CRelationChange (?*ch*) ∧ GeneratedBy(?*v*, ?*ch*) ∧ affectedByCCR(?*ch*, ?*e*) ∧ Delete(?*o*) ∧ hasTypeOperation (?*ch*, ?*o*) → delFamilyFromStructureOf(?*af*, ?*f*)	This rule enables inferring a family *af* which was removed from a structure of a family *f* in version *v*.
10	Family(?*f*) ∧ History(?*h*) ∧ Version(?*v*) ∧ FRestrictionChange (?*ch*) ∧ GeneratedBy(?*v*, ?*ch*) ∧ Add(?*o*) ∧ hasTypeOperation (?*ch*, ?*o*) ∧ FRestriction (?*e*) → addRestrcitionToFamily(?*e*, ?*f*)	This rule enables inferring that a FRestriction *e* was added to a family *f* in version *v*.

5 Case study in computer machine industry domain

This section presents the instantiation of the proposed ontology into a case study related to the computer machine industry, considering the laptop product line as an example. A brief description of how PRONTO is applied to represent this product line is presented at the end of Section 2.1. Aforesaid, *Laptop Family* has component parts and these components are also families. Despite *Laptop family* has many components, the example considers only the three ones shown in Figure 2: memory, processor, and floppy disk drive. According to PRONTO, these components are part of the SH of *Laptop Family* and they are related to it by *componentOf* relationship. For space reasons, this case study only considers changes in the SH of products at family level and leaves aside changes affecting the AH. In Figure 4(a), the *Laptop Family* is represented, depicting its composite structure, and is considered as an initial version.

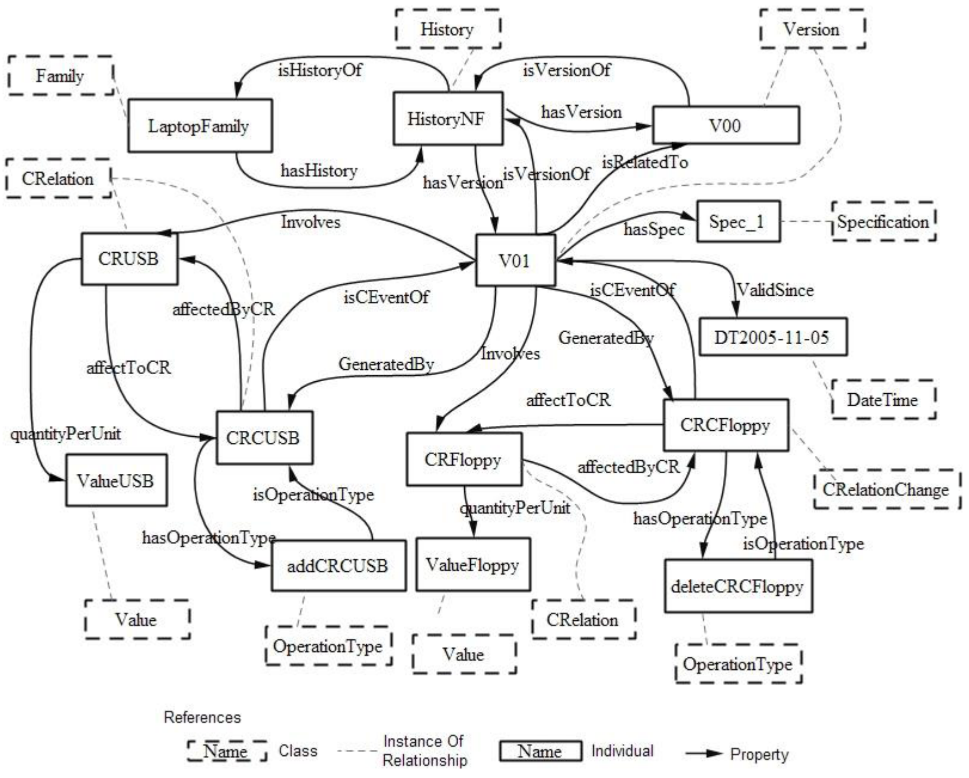
Each component of *Laptop Family* is related to the composite family structure (*CStructure*) by instances of *CRelation* class, which are called *CRMemory*, *CRProcessor*, and *CRFloppy*, respectively. These relations could be changed and therefore generate new versions of the family. In particular for this case, we consider a change caused by the introduction of a new storage device technology. This change affects directly the *LaptopFamily* structure where the relation (*CRFloppy*) between *LaptopFamily* and *FloppyDiskFamily* is removed and a new relation (*CRUSB*) is added between Laptop Family and USB storage device Family. This addition is represented by a dotted line in Figure 4(b).

Figure 4 Laptop family versions, (a) initial version (b) new version – changed structure



Modifications on the *Laptop Family* information can be implemented by the ontology described in Section 4. Figure 5 shows the *Laptop Family* represented by *LaptopFamily* individual (box with solid lines), which is an instance of *Family* class indicated by a box with dashed line. This individual is associated to *HistoryNF* individual (instance of *History* class) by *hasHistory* property. Based on this association, it is possible to deduce its inverse functional property called *isHistoryOf*, following the rules described in Table 2.

Figure 5 Ontology instance

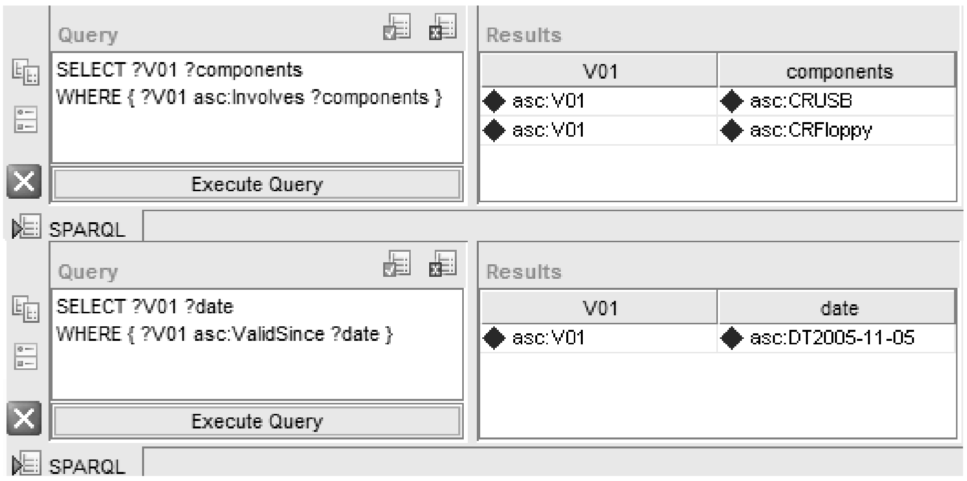


Each history is composed of a set of versions or a single initial version. In the example shown in Figure 4, the *HistoryNF* has two associated versions: *V00* (initial version) and *V01* (new version). Both versions are individuals instanced of Version class. The proposal considers the initial version as the first definition of laptop family. The occurrence of the aforementioned change events generates the new version (*V01*), which is valid from 2005-11-05 (see *DT2005-11-05* individual a *ValidSince* property). Also, specification (*Spec_1* is an individual of *Specification* class) is associated to *V01* version by *hasSpec* relation in order to capture information concerned with the causes and description of the event. In addition, the mentioned versions are linked to *CRFloppy* and *CRUSB* entities which are affected by changes belonging to it. Each one of these entities has its values associated by a *quantityPerUnit* property. The values are *ValueFloppy* and *ValueUSB* respectively, and indicate the amount of the parts that are required to produce members of the *LaptopFamily*.

V01 version is related with its predecessor version *V00* by means of *Previous* property. By navigating through versions using this property, it is possible to construct the family structure from the initial version to the current one. Change events are represented by *CRCUSB* and *CRCFloppy* entities and they are instances of *CRelationChange*. Each one is linked by means of *hasOperationType* property to its corresponding type: *addCRCUSB* or *deleteCRCFloppy*, respectively.

Once the information about the changes is captured and formalised, it is possible to answer the above mentioned set of questions, using SPARQL (SPARQL Protocol and RDF Query Language) language (SPARQL, 2013). This language, which allows writing queries to retrieve and manipulate data stored in triple pattern format, is selected to formalise the questions mentioned in Section 1. Figures 6 to 9 present screen captures of the SPARQL tab of Protégé editor with the formalisation of the queries. This tab allows defining queries and retrieving their results.

Figure 6 Queries in SPARQL to answer 'What were the affected components?' and 'When did it occur?'



```
SELECT ?V01 ?components
WHERE { ?V01 asc:Involves ?components }
```

V01	components
◆ asc:V01	◆ asc:CRUSB
◆ asc:V01	◆ asc:CRFloppy

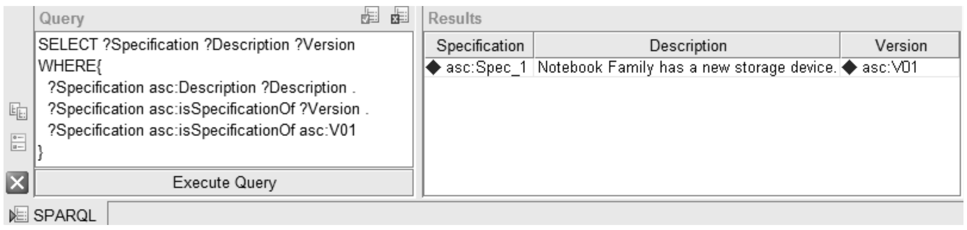
```
SELECT ?V01 ?date
WHERE { ?V01 asc:ValidSince ?date }
```

V01	date
◆ asc:V01	◆ asc:DT2005-11-05

Figure 6 shows the formalisation of the questions 'What were the affected components?' and 'When did it occur?' The left part of Figure 6 presents SPARQL queries while the right part shows their results. The first query asks about the components which are changed by version *V01*. Particularly, the changes belonging to *V01* affect *CRUSB* and *CRFloppy* relations. The second query asks about the time when the changes occurred, which is captured by the *ValidSince* property of *V01*, which links the version with the individual *DT2005-11-05*.

For the answer to ‘Why did it occur?’, we develop a simple query in SPARQL (Figure 7). The description field is a data type associated to *Spec_1* Specification and it is composed of a set of characters to explain for the reason of changes that caused the new version (in this case the *V01* version), and some useful information.

Figure 7 Queries in SPARQL to answer: ‘Why did it occur?’

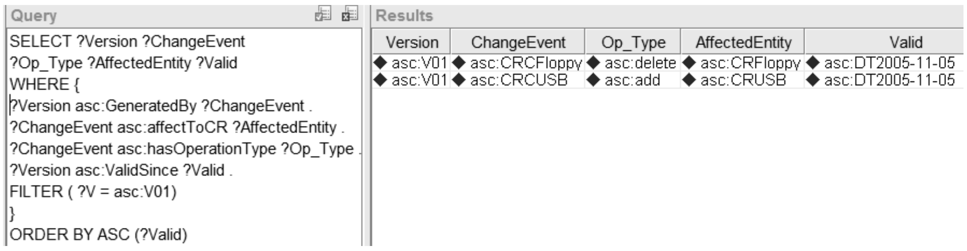


```
SELECT ?Specification ?Description ?Version
WHERE{
  ?Specification asc:Description ?Description .
  ?Specification asc:isSpecificationOf ?Version .
  ?Specification asc:isSpecificationOf asc:V01
}
```

Specification	Description	Version
◆ asc:Spec_1	Notebook Family has a new storage device.	◆ asc:V01

Similarly, to analyse the impact of a change, a query is written to select change events, operation types, the affected entities and the valid date filtering a specific version (Figure 8). Results are ordered in an ascendant way, based on the valid date.

Figure 8 Queries in SPARQL to answer: ‘how did it change?’

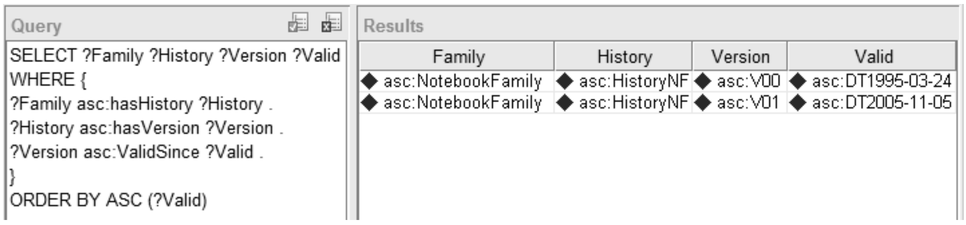


```
SELECT ?Version ?ChangeEvent
?Op_Type ?AffectedEntity ?Valid
WHERE {
?Version asc:GeneratedBy ?ChangeEvent .
?ChangeEvent asc:affectToCR ?AffectedEntity .
?ChangeEvent asc:hasOperationType ?Op_Type .
?Version asc:ValidSince ?Valid .
FILTER ( ?V = asc:V01)
}
ORDER BY ASC (?Valid)
```

Version	ChangeEvent	Op_Type	AffectedEntity	Valid
◆ asc:V01	◆ asc:CRCFloppy	◆ asc:delete	◆ asc:CRFloppy	◆ asc:DT2005-11-05
◆ asc:V01	◆ asc:CRCUSB	◆ asc:add	◆ asc:CRUSB	◆ asc:DT2005-11-05

Figure 9 presents the last question, which asks how the change was registered. A query is written to obtain the modified family, its history, and the versions that belong to it. In this way, it is possible to register information about different versions associated to a specific product.

Figure 9 Queries in SPARQL to answer: ‘How was the change registered?’



```
SELECT ?Family ?History ?Version ?Valid
WHERE {
?Family asc:hasHistory ?History .
?History asc:hasVersion ?Version .
?Version asc:ValidSince ?Valid .
}
ORDER BY ASC (?Valid)
```

Family	History	Version	Valid
◆ asc:NotebookFamily	◆ asc:HistoryNF	◆ asc:V00	◆ asc:DT1995-03-24
◆ asc:NotebookFamily	◆ asc:HistoryNF	◆ asc:V01	◆ asc:DT2005-11-05

All mentioned questions provide appropriate knowledge for a suitable change management of product information at family level during product life cycle. This case study is a simple demonstration to register a version caused by change occurrence, but this may be extended to handle more products, versions, and changes.

6 Conclusions

To obtain efficient software solutions to meet PLM needs, it is important to adopt processes or methodologies that allow managing, assessing, and registering the change that may be experienced by product information during product lifecycle. The proposal introduces a conceptual model to represent the information about the occurrence of changes in product structure, extending PRONTO model. The proposed model considers the registration of all versions that a product may suffer during its lifecycle.

An implementation of the proposed model is also introduced. OWL and SWRL languages were selected for the implementation. Finally, the article shows how the implemented ontology allows inferring knowledge about change events.

Ontology technology provides suitable mechanisms for capturing important change management concepts. The ontology can help us register the history of a product family. In addition, this extension facilitates navigation through all product family versions or infers new knowledge about it.

As future work, there are two goals to achieve: the extension of the proposal at variant set and product level in PRONTO AH, and the implementation of the described ontology by a merge operation with PRONTO ontology to validate this proposal.

Acknowledgements

We thank financial support jointly granted by CONICET, ANPCyT (PICT 2315), UTN (PID 25-O156), and Universidad Nacional de La Rioja. We are grateful to Translator María Inés Fidalgo for her contributions in the writing process.

References

Böhms, M., Bonsma, M.P., Willems, P., Zarli, A., Bourdeau, M., Pascual, E., Storer, G., Kazi, S., Hannus, M., García Sedano, J.A., Triguero, L., Tsahalis, H., Eckstein, H., Josefiak, F. and Schevers, H. (2008) *The SWOP Semantic Product Modelling Approach*, Deliverable 23 of the Semantic Web-based Open Engineering Platform Project [online] http://www.w3.org/2005/Incubator/w3pm/wiki/images/c/c3/SWOP_D23_WP2_T2300_TNO_2008-04-15_v12.pdf (accessed 16 September 2014).

Brandt, S.C., Morbach, J., Miatidis, M., Theißen, M., Jarke, M. and Marquardt, W. (2008) 'An ontology-based approach to knowledge management in design processes', *Computers and Chemical Engineering*, Vol. 32, No. 1, pp.320–342.

Fox, M. (1992) 'The TOVE project: a common-sense model of the enterprise', *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 604, pp.25–34, ISSN 1865-1348.

Gruninger, M. (2004) 'Ontologies and semantics for seamless connectivity', *SIGMOD Record*, Vol. 33, No. 4, pp.58–64.

Henning, G. (2012) 'Formal specification of batch scheduling problems: a step toward integration and benchmarking', *Advances in Production Management Systems*, Vol. 398, No. 2013, pp.96–103.

Hepp, M. (2006) 'Products and services ontologies: a methodology for deriving OWL ontologies from industrial categorization standards', *International Journal on Semantic Web & Information Systems (IJSWIS)*, Vol. 2, No. 1, pp.72–99.

Hepp, M. (2008) 'GoodRelations: an ontology for describing products and services offers on the web', *Lecture Notes in Computer Sciences*, Springer-Verlag, Vol. 5268, pp.329–346, ISSN 0302-9743.

Horváth, L. (2007) *New Design Objective and Human Intent-based Management of Changes or Product Modelling*, Institute of Intelligent Engineering Systems, John von Neumann Faculty of Informatics, Budapest Tech.

Huang, G.Q. and Mak, K.L. (1998) 'Computer aids for engineering change control', *Journal of Materials Processing Technology*, Vol. 76, No. 1, pp.187–191.

Huang, G.Q., Yee, W.Y. and Mak, K.L. (2001) 'Development of a web-based system for engineering change management', *Robotics and Computer Integrated Manufacturing*, Vol. 17, No. 1, pp.255–267.

ISO 10303 (1991) *Product Data Representation and Exchange – Part 44*, Product Structure Configuration.

Jarratt, T., Eckert, C. and Clarkson, J. (2004) 'Development of a product model to support engineering change management', *Proceedings of the TMCE 2004*.

Klein, M. (2002) *DAML+OIL and RDF Schema Representation of UNSPSC*.

Lee, H.J., Ahn, H.J., Kim, J.W. and Park, S.J. (2006) 'Capturing and reusing knowledge in engineering change management: a case of automobile development', *InfSyst Front*, Vol. 8, No. 5, pp.375–394.

Matsokis, A. and Kiritsis, D. (2010) 'An ontology-based approach for product lifecycle management', *Computers in Industry*, Vol. 61, No. 8, pp.787–797, ISSN: 0166-3615.

May, A. and Carter, C. (2001) 'A case study of virtual team working in the European automotive industry', *International Journal of Industrial Ergonomics*, Vol. 27, No. 3, pp.171–186.

May, A., Carter, C. and Joyner, S. (2000) 'Virtual team working in the European automotive industry: user requirements and a case study approach', *Human Factors and Ergonomics in Manufacturing*, Vol. 10, No. 3, pp.273–289.

Monplaisir, L. (1999) 'An integrated CSCW architecture for integrated product/process design and development', *Robotics and Computer-Integrated Manufacturing*, Vol. 15, No. 2, pp.145–153.

Numata, J. and Taura, T. (1996) 'A case study: a network system for knowledge amplification in the product development process', *IEEE Transactions on Engineering Management*, Vol. 43, No. 4, pp.356–367.

Ramesh, B. and Tiwana, A. (1999) 'Supporting collaborative process knowledge management in new product development teams', *Decision Support Systems*, Vol. 27, Nos. 1–2, pp.213–235.

Shaban-Nejad, A. and Haarslev, V. (2009) *Bio-medical Ontologies Maintenance and Change Management*, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada

Shadbolt, N., Hall, W. and Berness-Lee, T. (2006) 'The semantic web revisited', *IEEE Intelligent Systems*, May–June, Vol. 21, No. 3, pp.96–101.

Sjoberg, D. (1995) *Managing Change in Information Systems: Technological Challenges*, Department of Informatics, University of Oslo, N-0316 Oslo, Norway.

SPARQL (2013) *Query Language for RDF*, SPARQL Tutorial created by W3C SPARQL Working Group [online] http://www.w3.org/TR/rdf-sparql-query (accessed 16 September 2014).

Vegetti, M., Leone, H. and Henning, G.P. (2010) 'Document a three level abstraction hierarchy to represent product structural information', *Proceedings of the 12th International Conference on Enterprise Information Systems (ICEIS 2010)*.

Vegetti, M., Leone, H. and Henning, G.P. (2011) 'PRONTO: an ontology for comprehensive and consistent representation of product information', *Engineering Applications of Artificial Intelligence*, Vol. 24, No. 8, pp.1305–1327.

Yang, D., Miao, R., Wu, H. and Zhou, Y. (2009) 'Product configuration knowledge modeling using ontology we language', *Expert System with Applications*, Vol. 36, No. 3, pp.4399–4411.

Yoo, S.B. and Kim, Y. (2002) 'Web-based knowledge management for sharing product data in virtual enterprises', *International Journal of Production Economics*, Vol. 75, No. 2, pp.173–183.

Zhao, W. and Liu, J.K. (2008) 'OWL/SWRL representation methodology for EXPRESS-driven product information model. Part I. Implementation methodology', *Computers in Industry*, Vol. 59, No. 6, pp.580–589.