# Managing Changes in Distributed Biomedical Ontologies using Hierarchical Distributed Graph Transformation

Arash Shaban-Nejad*

McGill Centre for Clinical and Health Informatics,
Department of Epidemiology and Biostatistics, Faculty of Medicine,
McGill University,
1140 Pine Avenue West
Montreal, Quebec, H3A 1A3
Canada
Fax: + (514) 843-1551
E-mail: arash.shaban-nejad@mcgill.ca
*Corresponding author

Volker Haarslev

Department of Computer Science and Software Engineering,
Concordia University,
1455 de Maisonneuve Blvd. W.,
Montreal, Quebec  H3G 1M8
Canada
E-mail: haarslev@cse.concordia.ca

**Abstract:** Ontologies play a crucial role in current web-based biomedical applications for capturing contextual knowledge in the domain of life sciences. They are continuously evolving in order to fix the problems and provide valid knowledge. As our knowledge improves, the related definitions in the ontologies will be altered. This issue is inadequately addressed by available tools and algorithms, mostly due to the lack of suitable knowledge representation formalisms to deal with temporal abstract notations, and the overreliance on human factors. Also most of the current approaches have been focused on changes within the internal structure of ontologies, and interactions with other existing ontologies have been widely neglected. In our research, after revealing and classifying some of the common alterations in a number of popular biomedical ontologies, we present a novel agent-based framework, RLR (Represent, Legitimate, and Reproduce), to semi-automatically manage the evolution of bio-ontologies, with emphasis on the FungalWeb Ontology, with minimal human intervention. RLR assists and guides ontology engineers through the change management process in general, and aids in tracking and representing the changes, particularly through the use of category theory. We have also employed rule-based hierarchical graph transformation to propose a more specific semantics for analyzing ontological changes and transformations between different versions of an ontology, as well as tracking the effects of a change in different levels of abstractions.

**Keywords:** Controlled vocabulary; Bio-ontologies, Graph Transformation, Category theory; Change management.

**Reference** to this paper should be made as follows: Shaban-Nejad, A., Haarslev V. (2012) 'Managing Changes in Distributed Biomedical Ontologies using Hierarchical Distributed Graph Transformation', *Int. J. Data Mining and Bioinformatics*, Vol. x, No. xx, pp.x–xx.

Copyright © 200x Inderscience Enterprises Ltd.

**Biographical notes:** Arash Shaban-Nejad is a postdoctoral fellow in McGill Clinical & Health Informatics Centre at McGill University, where he conducts research on the knowledge representation and modeling for different applications in healthcare. He received his M.S. and Ph.D. in Computer Science from Concordia University in 2005 and 2010, respectively. His primary research interest is knowledge representation and semantic web, particularly ontologies and knowledge bases, description logics, category theory, intelligent agents, reasoning and inferencing, with special emphasis on applications from health and biomedical domains.

Volker Haarslev is a professor at the department of Computer Science and Software Engineering, Concordia University. He is a leading expert in knowledge representation, semantic web, and description logics. Volker haarslev is the co-founder of RACER, a highly optimized reasoner for ABoxes and TBoxes in description logics (DLs), and RacerPro, which supports OWL reasoning for the Semantic Web.

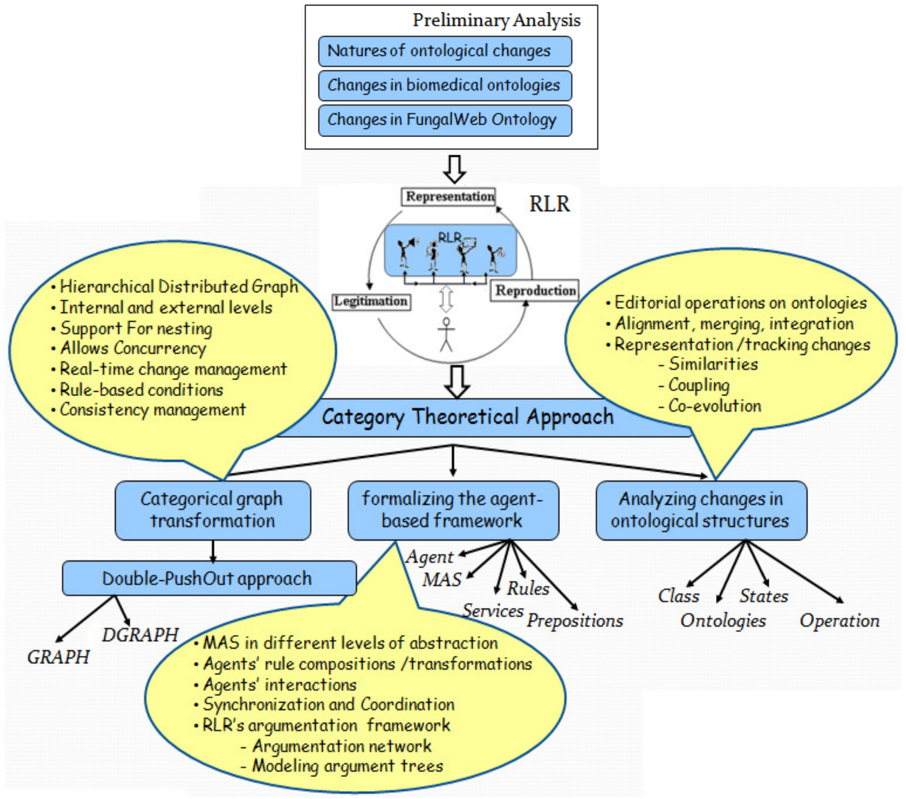
## 1 Introduction

Using clinical vocabularies and lexicons has a long history in medicine and life sciences. However, a new trend is emerging to use ontologies, as “specification of a conceptualization” (Gruber, 1993) to provide an underlying discipline of sharing knowledge and modeling biomedical applications by defining concepts, properties and axioms. Ontologies are widely used as a vehicle for knowledge management in current biomedical applications, for sharing common vocabularies, describing semantics of programming interfaces, providing a structure to organize knowledge, reducing the development effort for generic tools and systems, improving the data and tool integration, reusing organizational knowledge, and capturing behavioral knowledge. The main components of ontologies are concepts (classes), relations (properties), individuals (instances) and axioms. Concepts represent a set or class of entities within a domain. Relations describe the interactions between individuals of those concepts. Individuals are the “things” that exist in the real world, represented by a concept. Axioms are being used to constrain values for concepts or individuals. Ontologies capture knowledge from a domain of interest in order to share it between both machines and humans. When the knowledge changes, then definitions will be altered. Ontologies are evolving over time in order to fix errors, reclassify the taxonomy, adding/removing concepts, attributes, relations and instances. Generally most of the existing change management approaches have been faced with the following three issues:

1- Overreliance on human factor;
2- Lack of a suitable formalism to deal with temporal abstract notions;
3- Neglecting the interactions with other existing ontologies and focusing on changes in internal structure of ontologies.

In order to address these issues our study has been focused on finding a suitable formalism to capture, represent and analyze the ontological alterations in the domain of life sciences with minimum human intervention. Specifically we analyze how these changes can affect the dependent artifacts in a distributed environment. To this end, after analyzing the context of the problem and reviewing other existing techniques for change management in some existing biomedical ontologies (Shaban-Nejad and Haarslev, 2009), we presented a novel multi-agent-based approach, RLR (Represent, Legitimate, and Reproduce) (Shaban-Nejad and Haarslev, 2008) (Figure 1) to manage the evolving structure of biomedical ontologies in a semi-automatic and consistent manner with reproducible results. The RLR framework aims to assist and guide ontology engineers through the change management process in general, and aids in tracking and representing the changes, particularly through the use of graph transformation empowered with category theory as a mathematical notation, which is independent of any specific choice of ontology language or particular implementation.

**Figure 1** A general representation of RLR and the associated categorical graph-oriented formalism for managing changes in biomedical ontologies. As shown category theoritical approach has been used to perform change management in bio-ontologies by defining categories of *Class,States, Ontologies and Opearation* to analyze ontological changes at both internal and external levels. Also categories of *Agent, MAS,Services, Rules* and *Preposition* have been used to formalize agents interactions and negotiation (Ehrig, 1979). Moreover categorical representation has been employed as a main formal method for performing hierarchical distributed graph transformation, specifically through two categories of GRAPH and DGRAPH (distributed graphs).


Preliminary Analysis
Natures of ontological changes
Changes in biomedical ontologies
Changes in FungalWeb Ontology
Representation
RLR
Legitimation
Reproduction
• Hierarchical Distributed Graph
• Internal and external levels
• Support For nesting
• Allows Concurrency
• Real-time change management
• Rule-based conditions
• Consistency management
• Editorial operations on ontologies
• Alignment, merging, integration
• Representation /tracking changes
- Similarities
- Coupling
- Co-evolution
Category Theoretical Approach
Categorical graph transformation
formalizing the agent-based framework
Analyzing changes in ontological structures
Double-PushOut approach
GRAPH
DGRAPH
Agent
MAS
Rules
Services
Prepositions
Class
States
Ontologies
Operation
• MAS in different levels of abstraction
• Agents' rule compositions /transformations
• Agents' interactions
• Synchronization and Coordination
• RLR's argumentation framework
- Argumentation network
- Modeling argument trees

As we will demonstrate throughout this manuscript categories and graph transformation – more specifically hierarchically distributed (HD) graph transformation– provide abstract yet expressive enough formalism to address the second and third issues stated above. HD graph transformation as an adapted type of standard graph transformation has been employed to maintain the hierarchically structured knowledge in the Semantic Web environment. The graph transformation rules describe the structural changes placed during a knowledge base operation. To perform the transformation we employ the well-known category theoretical method double-pushout (Ehrig et al., 1973). In contrast to some of the existing works on ontology evolution, we specifically focus on changes in distributed ontologies, not as standalone artifacts but in contact with other ontologies in an open semantic web environment. Our proposed formalism can overcome the *decidability* issue that occurs in temporal description logics and the rigidity of OWL's single semantic structure. We demonstrate the technical correctness and feasibility of our approach through a set of case studies.

## 2 Changes in Biomedical Ontologies

There are currently a growing number of ontologies and controlled vocabularies in various areas of life science. It is not a surprise that many of them do not have sufficient requirement to be considered as a formal ontology. Especially most ontologies in the biomedical domain are known to be seriously defective in both terminological and ontological perspectives. In this paper we selected some of the most popular ontologies and controlled vocabularies in health science to find the evidences for various types of possible changes. The following ontologies have been selected based on several criteria such as availability, popularity, complexity and accessibility to the source and documentation. The Gene Ontology (GO) (Ashburner et al., 2000), Clinical Term Version 3 (The Read Codes) (NHS Information Authority, 2000), Health Level 7 (HL7) (Health Level 7 Reference Information Model), UMLS Semantic Network (McCray and Nelson, 1995), The Foundational Model of Anatomy (FMA) (Rosse and Mejino, 2003), Medical Subject Heading (MeSH) (Coletti MH, Bleich,2001) and Terminologia Anatomica (TA) (Whitmore, 1999). Based on our research of the literature, observing different releases of ontologies, surveys, and interviews with several domain experts and ontology engineers, we distinguished about 74 different types of changes that frequently occur in life cycles of existing bio-ontologies. We classified the changes under 10 groups: addition, deletion, retirement (obsoletion), merging, splitting, replacement (edit or rename), movement, importing, integration, or changes to file structure (Table 1).

**Table 1** Common changes in some of the popular bio-ontologies

| Type of change | Definition | Observed Ontology | Example |
|---|---|---|---|
| Addition | Improving ontological structure by adding one or more components to the available makeup. The most common additions in the observed bio-ontologies are of the following elements: Namespace, identifier code, concept, attribute, abbreviation, super-class, sub-class, attribute value, synonym, constraint (cardinality, type and min/max, inverse roles, default value), associative relationships (relationships to other individuals), annotation description, class-status (hidden/public), and instance. | Gene Ontology (GO) | The curators at MGI, who were reviewing the existing terms for comprehensive annotation of mammalian genes involved in the regulation of blood pressure, realized that the existing GO terms were not sufficient to annotate genes involved in the various processes that regulate blood pressure. They then proposed 43 new GO terms, which were discussed and refined with other GO curators through the GO discussion forum. They efforts yielded new annotations for mouse genes directly involved in the process of blood pressure regulation (GO Newsletter, 2006). |

| | | | |
|---|---|---|---|
| **Deletion** | Erasing the selected element(s) when it does not reflect the ontological 'truth' anymore. The most common deletions are of the following elements: Namespace, identifier code, concept, synonym, abbreviation, annotation (description), constraint (cardinality, type and min/max), attribute value, super-class, sub-class, constraint (cardinality, type and min/max, inverse roles, default value), associative relationships, annotation description, class-status (hidden /public), and instance. | Gene Ontology (GO) | The GO terms must characterize biological entities (i.e., functional activities that are catalyzed by enzymes). The terms classified as "Unknown" violated this principle, so the decision was made to delete the following terms: biological process unknown; GO:0000004, molecular function unknown; GO:0005554 and cellular component unknown; and GO:0008372 from the ontology. The new annotations signify that a given gene product should have a molecular function, biological process, or cellular component, but that no information was available as of the date of annotation (GO News letter, 2007). |
| **Retirement (Obsolescence)** | Deprecating an older element when a newer, more functional element or meaning supersedes it. The older version can be kept somewhere for future use, but its usage will be discouraged (Cimino , 1996). The retirement can usually be seen for the concepts, attributes, identifier codes, instances and relationships. | Health Level 7 (HL7) | In the release 2.0 of HL7, the components: ClinicalDocument.copyTime, MaintainedEntity, CodedEntry, inkHtml.name,table.border, table. cellspacing and table.cellpadding are retained for backwards compatibility with HL7 Clinical Document Architecture (CDA), Release 1.0, and have been retired. Further use of these components is discouraged (Dolin et al., 2004). |
| **Merging** | The process of creating a consistent and coherent ontological element that includes information from 2 or more basic elements. It can be seen as following: Merging two or more concepts into one of the concepts or into a new concept (Cimino , 1996), two or more attributes into one of the attributes or into a new attribute, two or more associative relations into one of the relations or into a new relation, two or more identifier codes into one of the codes or into a new code. | Health Level 7 (HL7) | In HL7, the purpose of the header is to enable clinical document exchange across and within institutions, facilitate clinical document management, and facilitate compilation of an individual patient's clinical documents into a lifetime electronic patient record (Dolin et al., 2004). In HL7's Clinical Document Architecture (CDA), Release 2.0, two concepts in the header (service_actor and service_target) have been merged (Dolin et al., 2004). |
| **Splitting** | An ontological element may be split into two or more new elements. This means that a concept can be split into two or more new concepts, an attribute into two or more new attributes, an associative relationship into two or more new relationships, or an identifier code into two or more codes. | Terminologia Anatomica (TA) | In TA, terms that share an id code are treated as synonyms. But, this does not hold for sexually dimorphic anatomical parts, such as 'Ovarian artery' and 'Testicular artery'. These two share the same TA code (A12.2.12.086) and therefore might be thought of as synonyms, but the two arteries are distinct and have different connections and other spatial relationships (Whitmore, 1999). So, they have to be modeled as two separated concepts, it means the code A12.2.12.086 can be split into A12.2.12.086-1 for 'Ovarian artery' and A12.2.12.086-2 for 'Testicular artery'. |
| **Replacement** (Edit, Rename) | This process is for editing available labels and values. This editing mostly happens to change namespace, concept name, concept definition, attribute value, attribute name, attribute definition, and concept role. | Health Level 7 (HL7) | A typical scenario (Dolin et al., 2004) from HL7 Release 2.0 is a simple replacement of Clinical Document.id "1.2.345.6789.266" replacing ClinicalDocument.id "1.2.345.6789. 123" |
| **Movement** (Transition) | The transition of one or more ontological elements across the ontological hierarchy. This transition can happen to identifier codes, concepts, attributes, super-class, sub-class, associative relationships, and instances. | Gene Ontology (GO) | GO terms representing transporter activity in the Molecular Function are gradually being overtaken to better represent current scientific knowledge. A new high-level term called "transmembrane transporter activity" (GO:0022857) was introduced. So, the related child terms and sub-classes have been moved under GO terms that describe the activity of the transporters, such as channel activity, active transporter activity, and symporter, antiporter and uniporter activity (GO Newsletter, Aug 2007) |

| | | | |
|---|---|---|---|
| **Importing** | Importing refers to the process of bringing an existing ontology (a tree) or parts of an existing ontology (sub-tree) into another ontological structure. | Gene Ontology (GO) | In 2001, the GO developers imported the first pass annotation from SWISS-PROT, trEMBL and Ensembl. Also, 7316 GO annotations were imported from Proteome and literature associations (GO Meeting, 2001). |
| **Integration** | In data integration, process data is extracted from different sources with different data formats, and then normalized into a consistent syntactic representation and semantic frame of reference (Buttler et al., 2002). The semantic integration is more complex than data integration. | Foundational Model of Anatomy (FMA) | In order to meet the need for an expressive ontology in neuroinformatics, the FMA developers have integrated the extensive terminologies of NeuroNames and Terminologia Anatomica into FMA. They have enhanced the FMA to accommodate information unique to neuronal structures, such as axonal input/output relationships (Martin et al., 2003). |
| **Change to Release File** (File Structure) | By the advancement of technology for storing and retrieving data files and the emergence of new standards, the format of file structures can be changed. | Read Codes | In Read Codes, Ver. 1.0 four character codes determined the position of a term in a hierarchy (4-Byte Set). The restrictions imposed by only 4 levels of hierarchy led to the development of a 5-Byte Set, which expanded the set to support secondary and tertiary care. This set was released in two structurally different versions. Ver. 1.0 has shorter terms and keys than Ver. 2.0. The more complex Ver. 3.0 structure is a superset of all old versions, and supports the character structures of both Ver. 1.0 and Ver. 2.0 (Robinson et al., 1997). |

As an application scenario, we consider the FungalWeb Ontology (Baker et al., 2006), an integrated formal bio-ontology in the domain of fungal genomics. The Fungal taxonomy is not stable. Most of the alterations are changes in names and taxonomic structure and relationships. Fungal names reflect data about the organisms; thus, as our understanding of the relationships among taxa improves, these names will need to be changed, as they will no longer convey the correct information to the user. Most fungi names are currently based on phenotypes (visible characteristics of an organism). These name changes may cause confusion and affect the validity of different queries. The morphological conceptualization of fungi is not sufficient, and will no longer work because all of the names based only on morphology must be re-evaluated. In addition, the phylogenetic-based conceptualization has its own limitations, since the decision of where to draw the line between different species is not always easy to make (Whitmore, 1999). To manage this process of continuous change, we rely on ontological conceptualization, where names in taxonomy are only meaningful once linked to descriptive datasets, which are extracted and managed from various databases and literature in an integrated environment.

## 3 The RLR Framework

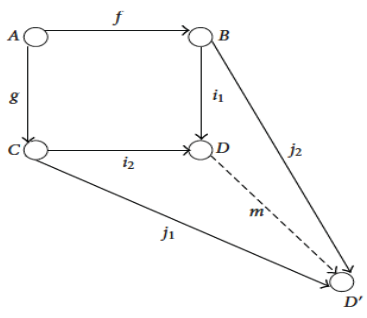
The RLR framework for change management consists of a set of intelligent agents designed to perform several intelligent tasks including learning, reasoning, capturing changes and negotiation within a collaborative environment. In a typical scenario within the RLR argumentative architecture, a user (human or agent) initially sends a request to an ontology engineer for a particular change in the ontological structure. Based on the system's background knowledge and the choice of the ontology engineer, various options are available to implement a change. The negotiation agent, along with the reasoning agent, provides arguments for the acceptance or rejection of a change proposal. An

"Argument Generator" determines appropriate responses based on the negotiation rules. Different arguments attack each other to enforce their rules and defeat their peers by sending counter-arguments. The inferred arguments can increase the possibility of higher quality agreements (Capobianco et al., 2005). The Negotiation Protocols in the RLR architecture contain the rules that dictate a protocol. As a knowledge base evolves, the historical information about different changes will be accumulated in the change logs. This information will be used by the learner agent, which acts as a basis for a recommender system, to propose different alternatives for the implementation of future changes. The reasoning and negotiation agents can change the rules if necessary and send modifications to the learning agent. In order to maintain agents' argumentation for automation of ontology evolution, we employ the "dialectical databases" (Capobianco et al., 2005). In argumentation-based multi-agent systems, a dialectical database tends to improve the speed of inference responses by storing pre-compiled knowledge about potential dialectical trees (Bryant and Krause, 2008). The dialectical trees represent sets of possible dialectical confrontations between the arguments to accept or deny a proposal to deal with a particular change. For the detail on the structure of RLR we refer the reader to (Shaban-Nejad and Haarslev, 2008). We use category theory and graph transformation to explore systematic changes in ontologies, analyze rule based transformations, and study various dependencies between the ontological elements, as well as formalizing agents' interactions and communications in the RLR framework.

## *3.1. Category Theory*

Category theory is a relatively new domain of mathematics, introduced and formulated in 1945 (Eilenberg and Mac Lane, 1945). Category theory is closely connected with computation and logic, which allows an ontology engineer to implement different states of design models to represent the reality. Using categories, one can recognize certain regularities to distinguish a variety of objects, capture and compose their interactions and differentiate equivalent interactions, identify patterns of interacting objects and extract some invariants in their action, or decompose a complex object into basic components. Categorical notations consist of diagrams with arrows. Each arrow $f: X \rightarrow Y$ represents a function. A Category $C$ includes: A class of objects and a class of morphisms ("arrows"), and for each morphism *f* there exists one object (A) as the domain of *f*, and one object (B) as the codomain; For each object A, an identity morphism, which has domain A and codomain A ("$\mathrm{ID}_A$"); and For each pair of morphisms $f$:A→B and $g$:B→C, (i.e., cod($f$) = dom($g$)), a *composite morphism*, $g \circ f$: A→C exists.

**Figure 2** A diagrammatic representation of categorical pushout


$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "i_1"] \arrow[ddr, "j_2"] & \\
C \arrow[r, "i_2"'] \arrow[drr, "j_1"'] & D \arrow[dr, dashed, "m"] & \\
 & & D'
\end{tikzcd}$$

Some of the primitive constructors of category theory that we use in our framework for ontology change management are as follows: Products, Co-products, Functors, Natural Transformation, Pushout and Pullback. More information on these categorical notions can be found in (Asperti and Longo, 1991). The *pushout* for two morphisms *f*: A→B & *g:* A→C is an object D, and two morphisms $i_1$: B→D & $i_2$: C→D exist such that the square in Figure 2 commutes. D is the initial object in the full subcategory of all candidates D' (i.e., for all objects D' with morphisms $j_1$ and $j_2$, there is a unique morphism from D to D'). The pullback is the dual notion to the pushout. Functors are defined as morphisms in the category of all small categories (where classes are defined as categories) (Awodey, 2006). In other words they are structure-preserving maps between categories. The maps between functors (morphism of functors) can be described by Natural Transformation.

### *3.2. Graph Transformation DPO approach*

The rule-based graph transformation can be studied based on the following three activities (Heckel, 2006):

- Creating the conceptual generalizations of the reality and transferring them from "reality" to its representation in a model;
- The definition of rules as specifications of state transformations;
- Using graphs as a means to represent snapshots, concepts, and rules.

Generally applying a transformation rule (production) p: L → R denotes finding a proper match of L (Left hand side) in the source graph and replacing L by R (Right hand side), leading to the target graph of the graph transformation. The major question in graph transformations is how to delete L from a source graph and connect R with the context in the target graph (Ehrig et al., 2006). Following the double-pushout approach (DPO) (Ehrig et al., 1973), a transformation rule is defined as a pair $t: L \leftarrow I \rightarrow R$ of morphisms $l: I \rightarrow L$ and $r: I \rightarrow R$ such that $l$ is injective, where the graphs $L$ and $R$ are called the left and right-hand sides respectively, and $I$ is called the interface or gluing graph. It is not necessary for the morphism $r: I \rightarrow R$ to be injective, which allows one to identify different nodes or edges in various transformations. Also, the injectivity of $l: I \rightarrow L$ ensures the uniqueness of the results in backward tracing in a transformation. The rule $t$ transforms a graph $O_G$ into a graph $O_H$, denoted by $O_G \Rightarrow_t O_H$ if there is an injective occurrence morphism $m: L \rightarrow O_G$, and two pushouts as represented in Figure 3.

**Figure 3** Double-Pushout approach for graph transformation

$$\begin{tikzcd}
L \arrow[d, "m"'] & I \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] & R \arrow[d] \\
O_G \arrow[ur, phantom, "(I)", near start] & K \arrow[l] \arrow[r] \arrow[ur, phantom, "(II)", near start] & O_H
\end{tikzcd}$$

In Figure 3, the morphism *m*, which models an occurrence of L in $O_{G,}$ is called a match. The transformation, which is performed by the specified rule, represents the change of the graph $O_G$ to the graph $O_H$. In more complex transformations we usually see a sequence of simpler transformations and a set of several transformation rules. As stated in (Ehrig et al., 1990), by considering the dangling points (those points in L, a subgraph of $O_G$, that are the source or target of arcs in $O_G$ minus L) and the identification points (those points in L that are identified in $O_G$) in the transformation of $O_G$, the gluing points of L (identified by $K_L$) can be identified if both dangling and identification conditions are satisfied. These two conditions together form the gluing condition, which ensures the transformation is valid.

$$\textit{Dangling condition} \cup \textit{Identification condition} \subseteq \textit{Gluing Condition}$$

Based on the previous definitions, the pushout exist if and only if $m$ satisfies the dangling condition with respect to $l$, and in this case $O_G$, $t$, and $m$ determine $O_H$ uniquely up to isomorphism. A graph transformation system is usually defined as a set of transformation rules (productions) $P$. In summary DPO should be performed through the following steps when a rule $L \leftarrow I \rightarrow R$ is given.

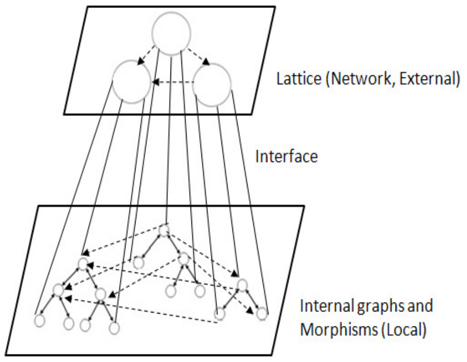
1. Find the elements of L in the given graph G, i.e. a match $m$: L $\rightarrow$ G.
2. Delete from G all the elements specified in L, which are not in the gluing graph $I$. This means to find a graph K and graph morphisms K $\rightarrow$ $O_G$, and $I \rightarrow$ K such that the square is a pushout.
3. Add to graph K all the elements of R, which are not in the gluing graph $I$ and create the second pushout and obtain a derived graph H.

## 4 Employing HD Graph Transformation for Ontology Change Management

Changes in an area due to technical, industrial, cultural, or social matters force the existing systems and applications to adapt themselves to the new state. Particularly, large systems and knowledge bases built upon smaller reusable sub-systems are in greater danger and should be continuously monitored to ensure the correctness and consistency of the entire infrastructure. In an ontological sense, concepts in an ontology naturally match with nodes of a graph, while the relationships in an ontology correspond to edges. The graph-based representation of the biomedical ontologies has a great tendency to become large, complex, and hard to grasp, understand, or maintain in a very short time. In applications dealing with compound graphs in layered organizations, the notion of graph can be extended to hierarchical graph. Hierarchical graphs attract broad attentions in theoretical computer science (e.g., object oriented design (Van Eetvelde, 2003), database (Elmasri and Navathe, 2007), and computational molecular biology (Mason and Verwoerd, 2008)), mostly for representing semantically complex and interrelated network structures. Different models, including the ones in (Engels and Heckel, 2000; Engels and Schürr, 2005) have been studied concerning the issue of hierarchical transformation of dynamic complex graphs, and several models (Hoffmann, 1999; Palacz, 2008) have been implemented using the rule-based approaches.

In order to mimic the actual nested hierarchical structure of the Semantic Web, where information is distributed in the nodes (graphs) and edges (relations between the graphs), we employ hierarchical distributed (HD) graphs (Taentzer, 1999), which enables us to perform the transformation on different levels of abstraction. The hierarchical graphs have richer semantics and are more expressive in comparison with regular flat graphs. In addition, they reduce the complexity of representation of large interrelated systems by allowing one to describe a system on a more abstract level through hiding the irrelevant details in encapsulated sub-graphs (Engels and Schürr, 2005). Hierarchical graph transformation can be performed using the extended double-pushout notion to represent various aspects of dynamic structures (e.g., the rearrangements of some temporal parts, describing the changes in relations, creation/deletion of communication channels, and performing operations such as “splitting” a graph into two or more graphs or “joining” distributed graphs into one graph (Taentzer, 1994)). As defined by (Taentzer et al., 1999), distributed graphs distinguish between two levels, namely local (internal), and network (external or lattice) (Figure 4).

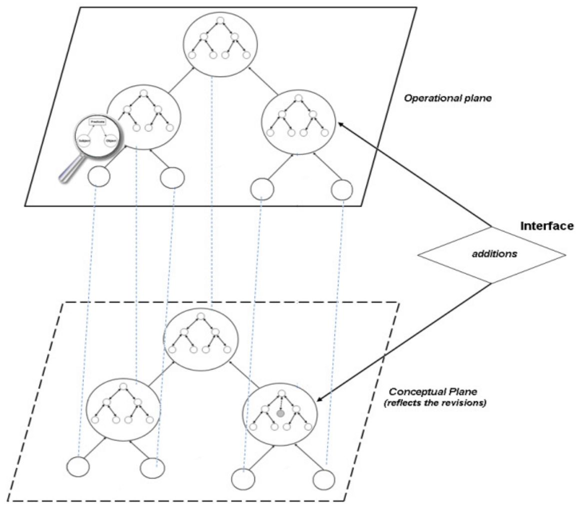
**Figure 4** An schematic representation of a distributed graph.



The communication between internal graphs can be performed via interfaces. In our model, the hierarchical graph (the lattice) consists of a set of internal graphs (which may be hierarchical graphs as well), the root of the hierarchy, and a set of edges that relates the internal graphs to each other. Each editorial action is expressed through a graph transformation and every state of the ontological structure is modeled in a graph with the nodes denoting objects and the edges representing the connections linking them. The categorical graph grammar supports the flexible change of complex interrelated compositions while providing explanations for corresponding actions performed by graph transformation. Various states can be produced by internal or external actions, and their communications can be modeled and simulated using graphs and state transitions, then represented and described by means of graph transformation. The double-pushout technique has been extended from flat to hierarchical graphs (Drewes et al., 2002), where the associated transformation rules can be applied at all hierarchical levels. This facilitates changes of the graph’s entries (i.e., by insertion or deletion) regardless of their size and configuration, with adaptation of the “dangling condition” from the flat graphs transformations (Drewes et al., 2002). The compound state of the entire system can be known by analyzing several other internal graphs, each having an internal state and behavior. There are also lattice-like dependency graphs representing dependencies between different internal graphs. In the process of change management for the lattice-like structure, several concerns related to sequential, parallel, or concurrent evolution of its components arise.

Different ontologies in Semantic Web are usually connected in a lattice-like structure and interact with each other through one or more interfaces. This lattice can be modeled as a directed graph with individual ontologies (internal graphs) as its nodes and the links between these ontologies as its edges (Figure 5). The described configuration is analogous to HD-graph (Taentzer, 1994 and 1999), where each of the links connecting the internal graphs contains a graph morphism specifying the relation between two internal states. When the internal graphs are faced with any change (e.g., adding/deleting a concept or relation), their state would be changed, which would affect other dependent graphs, and a synchronization unit within the RLR framework, which stores all the states in the change logs, forces the lattice-like structure and the mediator interface to change their states accordingly. Following the approach given in (Taentzer, 1994), this structure can be modeled in two different but related planes, namely conceptual (shows all existing and potential relations, paths, and their revisions) and operational (shows only actual existing nodes and relations) (Figure 5).

**Figure 5** A hierarchical graph for managing distributed ontologies representing the relations between different states of a lattice-like structure consisting different distributed ontologies. The changes can be performed in an interface graph that consists of all the nodes which have a matching node in the related internal graphs. In this way, the transformation of objects and morphisms allow the change of an evolving structure by changing its interfaces.



In order to categorically analyze the distributed transformations we employ the category of distributed graphs DGRAPH with distributed graphs as objects and distributed graph morphisms as arrows[1], to define a transformation using an adapted version of double pushout approach described in (Taentzer, 1999). Then we can define a transformation using double-pushout, in the category DGRAPH. Based on the definitions of graph, and the category Graph, a distributed graph can be defined as following. Consider G (a network graph) from category Graph, a diagram $\hat{\mathbf{G}} \rightarrow$ Graph is called a distributed graph while $\hat{\mathbf{G}}$ $(i)$ refers to the local graph that is related to node *i* of network graph (lattice) G (Taentzer, 1999). Then a distributed graph morphism for two distributed graphs $\hat{\mathbf{G}}$ and $\hat{\mathbf{H}}$ can be defined as $\hat{\mathbf{f}}$ (i): $\hat{\mathbf{G}} \rightarrow \hat{\mathbf{H}}$ such that $\hat{\mathbf{f}}$ is a natural transformation of $\hat{\mathbf{f}}$ (i): $\hat{\mathbf{G}} \rightarrow \hat{\mathbf{H}} \circ f$ in category Graph with *f*: G → H as its morphism (network morphism). So, distributed graphs and distributed graph morphisms serve as objects and arrows for the category DGRAPH (Taentzer, 1999). For the details of proofs and other

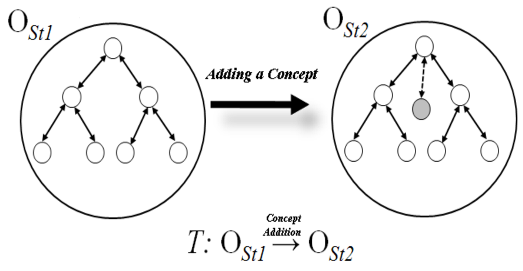
[1] Notice the similarities with the notion of categorical functor.

related categorical notions in distributed graph transformation one may refer to (Taentzer, 1999; Ehrig et al., 2006).

### *4.1. Analyzing Events and Actions in Rule-Based Model Transformation*

In order to analyze different events that trigger actions during the ontological evolution process, we consider events as part of the rule condition in a graph transformation. The actions are described by productions and the events will occur if certain predefined conditions are assessed to be true. To formalize graph transformation, we employ the notion of double-pushout from category theory, which needs certain requirements to compute production (describes actions in graph grammar) and its corresponding element in other graphs. One of the requirements is satisfying the gluing condition to derive a new graph by finding a match of the left side of the rule in the given graph, then deleting it (except the gluing point) and adding the right side of the rule (see (Ehrig et al., 1990) for the details). By following the approach proposed in (Taentzer, 1994), we use hierarchical distributed graph rules covering both internal and external production describing the internal and external actions respectively. Since the lattice-like structure covering the internal graphs is less likely to be changed by internal actions, which affect mostly internal graphs, the external graph is transformed through an identical production that preserves the external graph nodes. A typical example, illustrated in Figure 6, is the addition of an ontological element (i.e., a concept) to an existing ontology, which causes the state of the ontological structure (internal graph) to be changed. This action does not have a significant effect on the lattice-like structure (external graph). As represented in Figure 6, the hierarchical graph production "concept addition" demonstrates an internal action that transforms the ontological structure O from state $St_1$ to state $St_2$. This production will not alter the external graph represented in Figure 5.

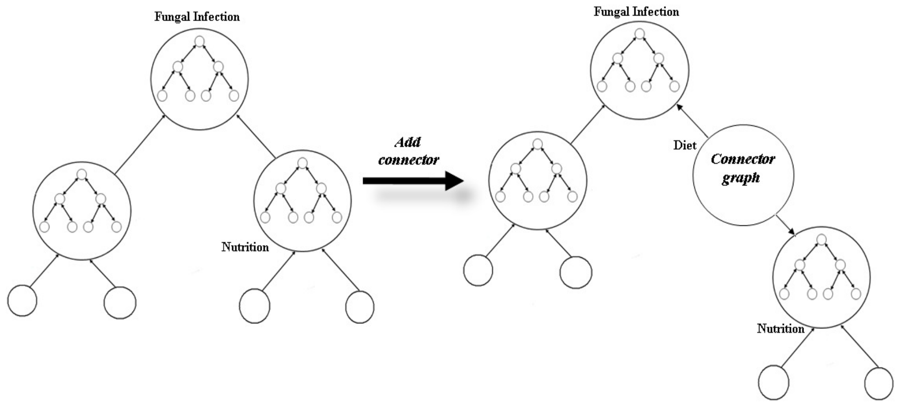
**Figure 6** Adding a new concept to an individual ontology that is part of a lattice made from several interconnected ontologies.



If one wants to delete an ontological element that has referenced a relation from other distributed ontologies in the lattice, then an external action needs to be performed. The external actions are capable of transforming the external graph. Controlling these transformations is a central task in the ontology engineering domain, since they can easily give rise to different types of inconsistencies, especially in cases that involve several parallel actions and transformations.

An example of alterations in the lattice in our application domain is the insertion of connective internal graphs (nodes) between two or more other internal graphs (nodes). For instance, it is known that "a daily cup of yogurt significantly reduces the risk of candida infection and colonization" (Hilton et al., 1992), but this diet might not seem appropriate for lactose intolerant patients. Also, some studies show that some nutrition is beneficial to reduce the risk and severity of candida infections if consumed in a proper diet. Some of the examples[2] are Probiotics (up to 900 mg daily of beneficial bacteria), Fructooligosaccharides (up to 4 g daily), Goldenseal (250 to 750 mg daily), Lactoferrin (300 mg daily), Topical tea tree oil (based on the prescription), Oil of oregano (460 mg daily), Garlic (600 mg daily), and Boric acid (600 mg daily for 2-3 weeks, shown effective in 65% of women with vaginal candida infections(Sobel et al., 2003)). In order to conceptualize these facts in an ontological framework, we use a connecting node (internal graph) "diet" to connect two structure fungal infections and "nutrition" through the hierarchical distributed graph production "add connector" (Figure 7).

**Figure 7** The hierarchical distributed graph production "add connector" is represented in a way that the state of the graph "fungal infection" is now related to the graph "diet", rather than "nutrition".



A transformation rule can determine conditions such as: 'the deletion of a lattice node should be performed after deleting its corresponding internal graphs'. As long as the actions (e.g. deletion, insertion) do not violate the defined conditions in the production rules several actions can be executed in parallel at the local level (e.g deletion/creation of internal elements). As mentioned, the external lattice production describes the structural changes of the external graph, and we can model the external actions using a hierarchical distributed graph production in such a way that an identical production for the internal graphs of every node of the external graph (individual ontological structures) must be performed. If the stated predefined conditions for insertion/deletion of the nodes in the internal graphs are satisfied, then the hierarchical distributed graph production can be applied at the external (lattice) level (for adding/deleting edges, a set of morphisms will be described instead).

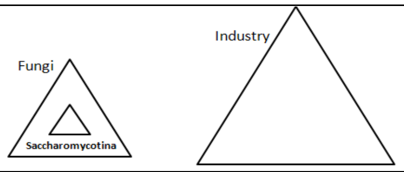
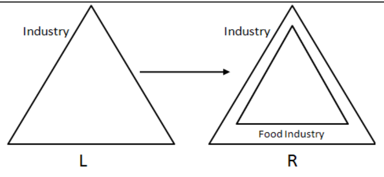
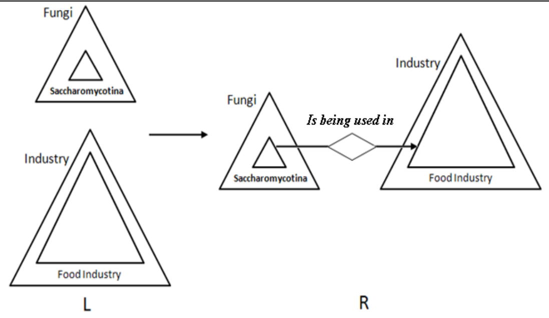
## *4.2. Transformation Rules for Changes in Ontologies*

The transformation rules in ontology evolution determine what types of changes are allowed and can be performed on the ontological elements and axioms. Padberg (2008)

[2] Fungal Infections (Candida). Life Extension E-Magazine: www.lef.org/protocols/infections/fungal_infections_candida_01.htm

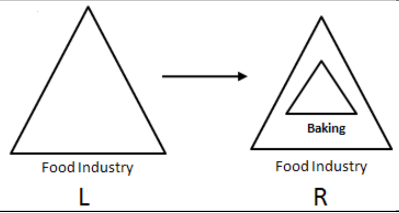
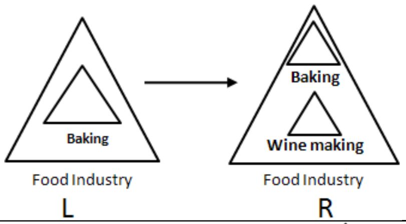
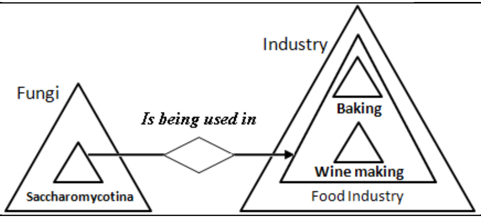
describes the notion of rule-based refinement as an extension of transformations with added refinement morphisms alongside the rules, which can be applied for maintaining component-based applications. We found that the ontology evolution process, through subsequent refinements, is generally analogous and compatible with rule-based hierarchical graph transformation and refinement. Generally, in a DPO approach, a rule-based transformation indicates the changes of $O_G$ to $O_H$ based on the defined rule. The rules can be atomic or compound and will be examined to ensure the compatibility and consistency[3] of the transformations. Our proposed rule-based transformation method for ontologies determines the circumstances under which an ontological element can be changed or refined. Table 2 represents some examples[4] of graph transformation rules, which can transform a typical graph such as Industry (Diagram 2). Diagram 3 represents the establishment of the relation "is being used in" to connect two graphs, "Fungi" and "Industry". Diagrams 4 and 5 show the rules that specify the internal structure of the food industry. By applying these transformation rules, Diagram 6 is obtained, which gives us two potential matches (baking and wine industry) on the left. A transformation can be defined to be conditional (Habel et al., 1996) in such a way that under certain conditions, the graph production (rules) transform a source graph into the target graph. These conditions, which impose a set of restrictions on the transformation processes, can help one to avoid inconsistencies and conflicts (e.g., the conflicts due to dangling edges).

**Table 2** Some examples of the graph transformation rules for part of the FungalWeb Ontology.

| | | |
|---|---|---|
| 1 |  | Two individual graphs Fungi and Industry are in their initial state |
| 2 |  | Transforming the Industry graph (R) to the new version (L) to cover more detailed information (adding child) |
| 3 |  Fungi<br>Saccharomycotina<br>Industry<br>Food Industry<br>Fungi<br>Saccharomycotina<br>Is being used in<br>Industry<br>Food Industry<br>L<br>R | Defining the relation *"is being used in"* to connect the two graphs Fungi and Industry. |

[3] In fact using graph transformation as the underlying formalism can guarantee the consistency of the results (Taentzer , 1998). This is an important point, since the distributing nature of evolving structures gives rise to different types of inconsistencies.

[4] For demonstrating the transformation rules in our model, we have employed the diagrammatical notions (Table 2) introduced in (Palacz, 2004).

| | | |
|---|---|---|
| 4 |  | Adding a child node to specify the internal structure of the food industry. |
| 5 |  Baking<br>Food Industry<br>L<br>Baking<br>Wine making<br>Food Industry<br>R | Adding another child node to specify the internal structure of the food industry. |
| 6 |  Fungi<br>Saccharomycotina<br>Is being used in<br>Industry<br>Baking<br>Wine making<br>Food Industry | The two potential matches (baking and wine industry) can be chosen from the left hand side. |

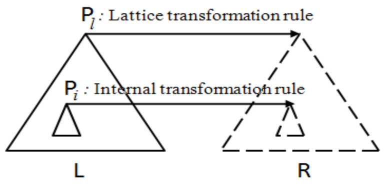
### *4.3. Formalizing the Ontology Change Model in Distributed Environments*

In our model, the hierarchical distributed graph has been used for analyzing dynamic distributed models and their transitions by describing the initial state, internal and external actions and defining communicating channels for synchronization. Category theory is used as a complementary formalism for supporting graph grammar describing the initial graph and a set of all hierarchical graph productions modeling various actions (e.g., additions, modification of relations, and so on) in a distributed system. The DPO approach to graph transformation as a constructor within the categorical framework is comprehensively described in (Ehrig et al., 2006) for directed and labeled graphs. This method has been generalized to so-called high-level replacement (HLR) systems in (Ehrig et al., 1990; 2006) by abstracting the results into arbitrary objects and morphisms. It has been proven (Taentzer, 1994) that the hierarchical distributed graph transformation is a highly appropriate scenario for HLR systems. Reflecting this approach into our framework, we consider the lattice *L* consisting of all interacting ontologies as a hierarchical distributed graph, with a set of transformation rules (e.g., rules for node addition/deletion), which is defined as a functor $HD$: $L \rightarrow$ G, where G is the category of all labeled graph and L$\in$G. To define the HD-morphism we can use natural transformations, which are simply the morphisms in the category of functors. Categorically speaking, the distributed ontological structures can be considered as objects and the links between them, which shape the lattice structure, as morphisms. This approach allows one to study the behavior of evolving categorical systems in different layers (analogous to the modular definition of ontologies) and different levels of abstraction.

## 5 Distributed Change Management within the RLR Framework

In our approach, we adapted the graph transformation methods for realizing the problem of specifying changes in distributed ontologies in two levels of abstractions, namely micro level (changes in internal structure of an ontology, e.g., adding/deleting a concept to/from an ontology) and macro level (when the internal changes spread out to an interrelated ontological organization, e.g., changing the state of an ontology or adding/deleting an ontology to/from interrelated system). The propagation of changes may need to be performed during the runtime of many critical systems (e.g., knowledge bases supporting robotic surgeries or aviations); therefore, these two levels always need to interact closely to ensure the success of a change management strategy. The distributed graph transformation can act on different levels of abstractions, ranging from explaining the details of local actions to the rule-based analysis of different interactions and operations (e.g., inter-communication, migration, and synchronization) (Taentzer et al., 1998) before or after a transformation. In order to successfully manage changes in a specific dynamic system, it would be essential to know, or at least have a reasonably accurate guess, about all the possible states of that system at different times. The fact that the dynamic system acts in a distributed environment makes this need more vital. The concept of distributed graphs has been defined in (Taentzer et al., 1998) as networked compound graphs with a set of internal graphs as the nodes expressing a internal state of the system, and a set of graph morphisms as the edges connecting the nodes (internal graphs) to each other. Distributed graph transformation aims to mediate between these two levels of abstractions (networks and nodes) and can be used to model many different types of dynamic network reconfiguration by applying a set of rules for each of the levels (Figure 8).

**Figure 8** $P_l$ and $P_i$ respectively specify sets of lattice and internal transformation rules.


$P_l$ : Lattice transformation rule
$P_i$ : Internal transformation rule
L
R

The rules contain the instructions for performing different changes (either in the network topology or in the nodes) and transformation in a dynamic system via defined actions at different levels of a distributed graph. The rules also determine whether or not a change operation is eligible to occur. The communication between lattice and internal rules performed within a coordinated channel can be used to synchronize different actions in node and lattice levels.

### *5.1. Synchronization and Coordination*

Managing several concurrent internal and external actions is also vital in the Semantic Web domain. Considering the Semantic Web as a hierarchically organized graph-like structure, each action on a graph has consequences in its modified consecutive version, which helps in tracing the events while preserving the reference state, or in some cases reconstruction of the past, if it has been removed from the original version. The changes in a lattice-like structure can be performed at the nodes (e.g., replace/rename a node), edges (e.g., replace an edge) or hierarchical structure (e.g., adding/deleting one or more nodes).

A hierarchical distributed graph production can be used for synchronization purposes by checking whether the external production is identical (or compatible) with what is performed by internal actions. More precisely, it checks if the lattice nodes and edges, in coordination with internal actions, have been identically replaced in the interface with respect to the gluing condition. For example, a graph production can describe a synchronous communication channel (Taentzer, 1994) between two different versions of an internal graph by highlighting the revisions in the original state and the current state through the use of an interface graph. Later on, the action that causes a change in the internal graph needs to be synchronized with other actions on dependent internal graphs and finally with the actions that alter the external graph. In real world applications, this synchronization usually results in a series of mappings between the previous and current states. To manage the interaction between the actions on different levels, we generalize the change model proposed in (Kramer and Magee, 1990) for the software engineering domain to classify the changes in a dynamic network at nodes and network levels.

The agents in the RLR framework interact with each other through a set of communication channels to control actions at different levels. This control assures the consistency and integrity of changes by defining quiescent[5] nodes and states. The nodes are assumed to be in a quiescent state (non-active/passive state) when changes occur at the lattice level. According to (Kramer and Magee, 1990), a quiescent state for a node is a state wherein the whole system is consistent and no active communication exists between the nodes or within their environment. The notification for changing the node's state from active to passive (and vice versa) is given through the established communication channel between the defined abstraction levels. In RLR, upon detection of the alterations by the set of change capture agents, the current state of the system would be assigned to the newly affected elements (e.g., newly added nodes) and an alert would be sent to the other involved components to inform them about the latest state of the system.

The state of a system should be determined and declared by an agent to allow some actions to be performed in a proper state of the system, to postpone them for later states, or to prevent them from acting on some of the preserved elements. For example, in the case of deleting or splitting a node, it acts like the lock mechanism in the database. The synchronization begins with assigning the states to each element, starting with the initial state upon its creation and continuing until the final state is assigned upon its termination. RLR controls the changes by incorporating the transformation rules (at different levels) along with other pre-defined consistency conditions. The synchronization of two different nodes (internal graphs) in a distributed graph can be performed through an interface (Taentzer et al., 1998) that connects these nodes together. The transformation is performed by a sequence of simpler transformations, each meeting certain conditions to ensure the target graph is still a distributed graph and to avoid any side-effects (explicit or implicit) on the graph structure. Some of these conditions are as follows (Taentzer et al., 1998):

- Gluing condition of the double-pushout approach for the rules at different levels;
- Connection condition, which prevents the deletion of the nodes and the edges if they are being used by other components.

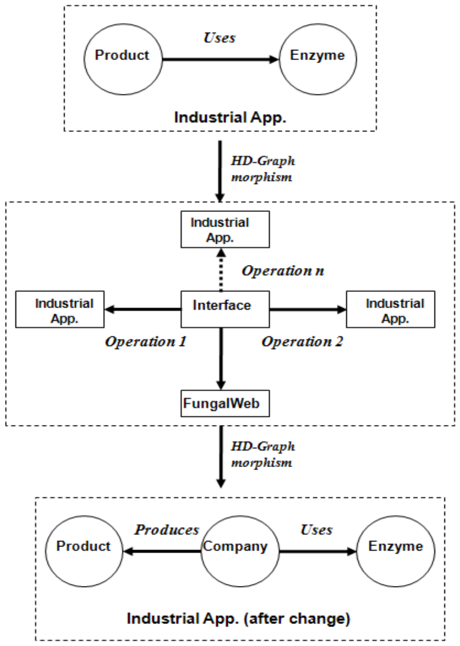
[5] This strategy is similar to "locking" in database research.

Also some other conditions and restrictions may be applied to each distributed rule, depending on its function. The main context conveyed by the lattice may be defined as protected to keep it unchanged. If the different actions and changes that are executed at the node's level have minimal or no interference with each other, they can operate in parallel. Assume a set of related ontologies, each with the ability to manage the changes in its own structure and each change potentially affecting other ontologies. An agent can initiate an action for changing each ontology in the lattice, based on imposed rules. This action can then be spread throughout the entire lattice. The distributed graph transformation can be used to model real-time changes, such as the insertion or deletion of ontologies. This is important since many changes and updates, unseen in the design phase, can be applied when the system is in operation if they do not cause any interruption. If we consider changing a node, it should be flagged as an inactive state, so it will not update the system's knowledge upon a change (neither initiate an update nor service any update request (Taentzer et al., 1998)).

## 5.2. Rule-based Patterns for Transformations

After each change, the system needs to be verified for consistency. In order to preserve the ontological elements' identities and guarantee the consistency and integrity of the changes, we can define a set of pre- and post-conditions to be satisfied. If all the conditions within a distributed graph transformation rule are satisfied, then the result of transforming an initial distributed graph would be a legitimate distributed graph as well.

**Figure 9** Representation of a change in a part of the FungalWeb Ontology using graph transformation.



Consider the three ontologies ($O_1$, $O_2$, and $O_3$), connected to each other in a lattice-like structure. Each node of the lattice represents an ontology and each edge signifies a graph morphism. The information about the state of each ontology and its relations with other ontologies in the lattice is stored in an interface node. The diagrams in category theory intuitively reflect the feasibility of our method, by demonstrating the interactions between the states and the information related to the changes. By following the method given in (Taentzer et al., 98), Figure 9 demonstrates the changes in industrial applications within the FungalWeb Ontology (as an internal graph in a whole integrated lattice), which consists of the concepts "enzyme" and "product", with the relation "uses". The figure depicts the effect of changes and the state of the ontology (starting from initial inactive state) in the lattice-like environment, along with its predecessor and successor versions, using the following distributed graphs:

In Figure 9 assume an update (internal action) starts at the FungalWeb Ontology to delete the existing relation "Uses" and add the new concept "Company" and the new relations "Uses" and "Produces" to relate the newly added concept with concepts "Enzyme" and "Product" respectively. We apply the following rules to perform this update:

*Add interface node ("FungalWeb Interface"),*
***Operation 1:*** *Add ontological element_Concept (FungalWeb, "Company");*
***Operation 2:*** *Delete ontological element_Relation (FungalWeb, "Uses");*
***Operation 3:*** *Add ontological element_Relation (FungalWeb,"Company", "Product", "Produces");*
***Operation 4:*** *Add ontological element_Relation (FungalWeb,"Company", "Enzyme", "Uses").*

To hide unnecessary details, the change processes and related interactions are performed via interfaces[6]. In using category theory, we focus on the interactions between objects rather than their internal structure. In summary, in our categorical representation of a hierarchical graph organization, anything other than nodes and edges (e.g., attributes such as data type properties for ontologies) are supposed to be marginal and not essential. Thus, the notion of graph transformation can be defined (Busatto et al., 2005) as $G,R \Rightarrow C,E$ , with $G$, $R$, $C$, $E$ respectively indicating a category of graphs, a category of rules, a category of control conditions, and a category of graph expressions (cf. (Busatto et al., 2005) for more information). Modeling the notion of graph transformation in an abstract way is significant in the sense that it hides the marginal information, which does not explicitly contribute in the transformation process. As an example, a transformation using the double-pushout (DPO) has been shown in Figure 10 for part of the FungalWeb taxonomy. The transformation rule determines a condition for a consistent deletion operation within an ontology by specifying that if a parent-node has to be deleted its children should be deleted as well.

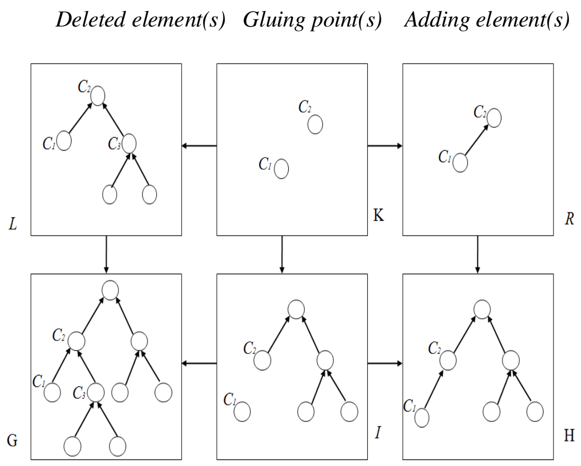
The double-pushout approach, constructed based on categorical pushout, in our example has been generally represented as the gluing of two graphs via a common interface. As shown in Figure 10, the left side indicates a pattern[7] to be located in the original graph (G); the right side represents the requested transformation, which transforms the original graph (G) to the transformed graph (H); and the middle section represents the gluing point(s) ($C_1$ and $C_2$), which are identified by $L \cap R$. In the RLR Framework the agents generalize the behaviors by systematically monitoring the

[6] "Interface generally refers to an abstraction that an entity provides of itself to the outside. This separates the methods of external communication from internal operation, and allows it to be internally modified without affecting the way outside entities interact with it." (Miller et al., 2010).

[7] To define a pattern to be always applicable it is sufficient to leave the left side of the associated rule empty.

transformations and encapsulating the changes from one point to the subsequent position to extract rules and generate the patterns. The patterns can be repaired, improved, and evolved through an intensive didactic teaching process, which enables the agents to derive rules from a sequence of trial state changes.

**Figure 10** The transformation of an ontological structure following the rule "deletes a parent node". The upper part represents the transformation rule, and the bottom left shows a given graph and the bottom right demonstrate the result of the transformation, which has been obtained by following the three steps in DPO.



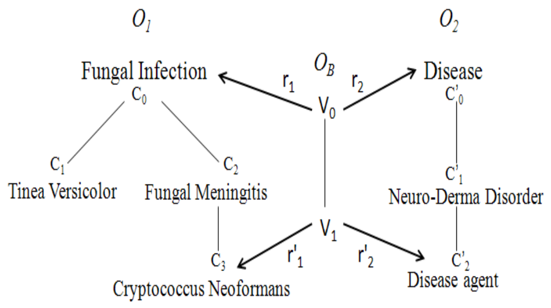
*5.3. Similarity Checking and Traceability*

A graph comparison methodology has been presented in (Drewes et al., 2002) to compare the contents of two graphs by considering the number of nodes and edges. The comparison has been performed based on applying the rules while considering the hierarchical dangling condition, to check whether a specific sub-graph exists or not (i.e. when one attempts to delete a graph). RLR intends to audit and monitor very large, heterogeneous, evolving biomedical ontologies and nomenclature scattered across the Web by highlighting changes between different versions of an ontology. In order to facilitate the change tracking process, we employ diagrammatic features on graph representation along with category theory, which enable us to represent the system's activity in different levels of abstraction. Our approach is similar to the tracking graph transformation approach (Busatto et al., 2005), which models the rules' internal structure by means of LHS (left-hand side) and RHS (right-hand side) graphs and a partial morphism between them, which facilitates the tracking of preserved graph components between two versions of a graph through a set of consistency constraints to check matching morphisms.

## 5 Case Study: Managing Changes in Distributed Ontologies

As the knowledge about fungi species grows and new methods become available one can anticipate a fundamental change in the current fungal taxonomy structure. A small percentage of discovered fungi have been linked to human diseases, including dangerous infections. Treating these diseases can be risky because, as mentioned above, human and fungal cells are very similar. Any medicine that kills the fungus may also damage the human cells. Therefore, greater knowledge of fungi and correct identification of each species is crucial to improving the quality of fungal-based products and identifying new and better ways to treat serious fungal infections in humans. From the other way since skin disorders have been historically categorized by appearance rather than scientific and systematic facts, the existing taxonomy of fungal diseases must be also modified based on the new knowledge to update the ontological truth. Many terms in current medical mycology vocabularies describing skin disorders originate as verbal descriptions of appearance, foods, people, mythological and religious texts, geographical places, and acronyms (Al-Aboud et al., 2003). Many names and terms are highly dependent on individual or regional preferences, causing redundancy, vagueness, and misclassification in current vocabularies. Thus, we study various alterations in both fungal taxonomy and fungal disease classification. As an example of changes in fungal terminologies, one can see several changes in the name of pathogenic fungi *Trichophyton* family (i.e. *Trichophyton Soudanense*, *Trichophyton megninii,* and *Trichophyton equinum*) in relatively short period of time. As another example, the pathogenic fungus *Candida glabrata* is now called *Torulopsis glabrata* (Cushion and Stringer, 2005). Usually changes in fungi taxonomy alter the related disease name and description. For instance, the name of the fungus, *Allescheria boydii* which can cause various infections in humans, was changed to *Petriellidium boydii* and then to *Pseudallescheria boydii* within a short time (Odds et al., 1992). Consequently, the infections caused by this organism were referred to as *allescheriasis*, *allescheriosis*, *petriellidosis*, and *pseudallescheriosis* in the medical literature (Odds and Rinaldi, 1995).

**Figure 11** The categorical representation of the alignment between two ontologies $O_1$ (Fungal disorders), and $O_2$ (Diseases) using a bridge ontology $O_B$ and a set of bridge axioms (r).



Fungal Meningitis is an infectious disease caused by a fungus, e.g. *Cryptococcus Neoformans*[8], *which* is typically seen in patients with immune deficiency such as AIDS. It usually results from an infection that spreads to patient's brain from another part of her body. This disease has been a subject for study in both dermatology (Leung, 1990) and neurology for a long time. The knowledge about this disease (i.e. symptoms, causes, etc.)

[8] Here is the lineage of *Cryptococcus neoformans* in the FungalWeb Ontology: Fungi; Dikarya; Basidiomycota; Agaricomycotina; Tremellomycetes; Tremellales; Tremellaceae; Filobasidiella; Cryptococcus neoformans (Filobasidiella neoformans).

are scattered in several existing ontologies and knowledge bases, which need to be aligned. We model the alignment of two ontologies by means of a pair of ontology mappings from a bridge ontology using categorical notations (Figure 11). In order to achieve a composite knowledge of the disease's properties we have used the categorical product to represent this integrated view (Figure 12). As can be seen in the Figure 12 medical specialty is the product arrow of the two branches in medicine, which includes the attributes of both domains. In order to merge two unrelated ontologies we can simply perform the disjoint union (or co-product).

**Figure 12** Determining the medical specialty for a particular disease through product.

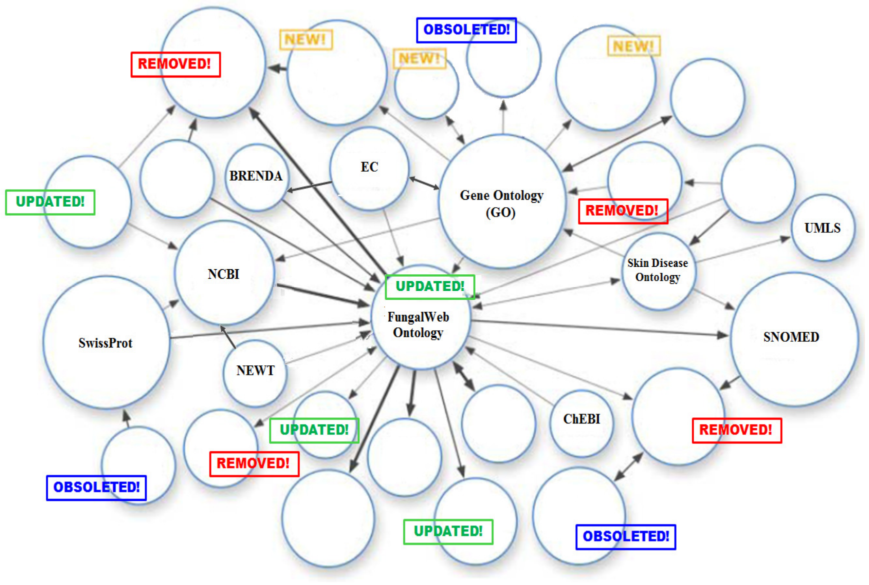
$$\begin{tikzcd}
& \text{Fungal Meningitis} \arrow[ld, dotted, "\text{Branch}"'] \arrow[d, "\text{medical specialty (MS)}"] \arrow[rd, dotted, "\text{Branch}"] & \\
\text{Dermatology} & \text{Neurology} \times \text{Dermatology} \arrow[l, "MS_1"] \arrow[r, "MS2"'] & \text{Neurology}
\end{tikzcd}$$

In our domain, we need to update and improve the ontological structure of the FungalWeb and SKDON (SKin Disease ONtology) Ontologies regularly for the annotation of fungal genes and analyzing the role of the fungi species in various diseases. For example, the older version of the FungalWeb Ontology did not have sufficient terminology to annotate genes involved in *Malassezia* infections. To meet this new requirement, the updated version of the ontology has gained 26 additional terms addressing these infections.

Category theory within the RLR framework has a significant potential to be considered as a supplementary tool to capture and represent the full semantics of ontology driven applications and it can provide a formal basis for analyzing complex evolving biomedical ontologies.

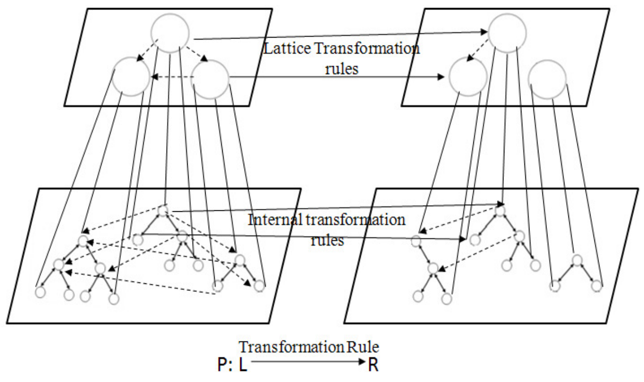
In fact ontologies are not isolated structures, but they tend to be reused as much as possible. The Semantic Web ultimate vision is to bring the existing ontologies, knowledge bases, controlled vocabularies, thesauri, databases and linked data sources under one umbrella, in such a way that they can communicate with each other and with users in a coordinated interactive manner. The FungalWeb ontology is in close contact with other resources such as Gene Ontology, TAMBIS (Baker et al., 1998), SwissProt (Bairoch, 2000), BRENDA (Schomburg et al., 2004), and etc (Figure 13). It is highly desirable that all changes within a resource can be tracked and all the impacts of such changes as well as their directions can be recognized and indentified. In our approach changes to each part of the ontology can cause that the conceptual design changes the state and also may cause alterations to other dependent artifacts.

**Figure 13** Interrelated distributed ontologies, knowledge bases and data sources in biomedical domain.



Defining appropriate transformation rules, such as what is represented in Figure 14, is the first step towards performing a transformation. Recalling the definition of category DGRAPH and using the approach proposed in (Taentzer, 1999), a pushout over distributed graph morphisms with respecting to both lattice (network) and internal (local) morphisms can be constructed, which enables us to apply the defined pushout-based transformation rules to describe changes in the distributed ontologies.

**Figure 14** A distributed transformation rule, which regulates the transformation of different interconnected ontologies in two abstraction levels, namely internal and lattice.



## 5 Discussions and Conclusion

Our proposed approach has still room for improvement in several areas, some of which have been considered for future work. Incorporating new knowledge in an ontology, must be in a way that it should not contradict the existing ‘truth’. Therefore as a vital part of ontology maintenance one should always watch for the consistency and coherency of the evolving ontologies. During the agents’ collaboration and negotiation in RLR, each action is evaluated for its potential consequences on the detected and identified inconsistencies in each context. Then, either the action should be banned or the inconsistencies must be resolved. Ideally these processes should be examined every time the state transition has occurred to ensure that the ontological consistency still holds. The consistency management in our model includes several options including:

- Enforcing the actions for prohibiting the alterations that may lead to inconsistencies that often inherit to different versions and endure over the substantial part of the ontology’s life cycle. This has been done by defining a set of conditions on transformations. Checking consistency of the graph transformation and whether a sound graph structure exists or not, along with controlling the consistency conditions have been broadly addressed in (Heckel and Wagner, 1995).
- Isomorphic Reasoning and Commutative Inference: In order to validate the categorical diagrams the partial isomorphism in the semantic web environment can be defined based on the similarity in structural relationships between syntax, semantics, and the resources of the knowledge in ontological frameworks. From a categorical point of view, the simplest type of isomorphic reasoning involves an explicit and continuous mapping of the correspondences and similarities at the syntactic level while ignoring the semantics. This method enables us to perform reasoning about the dynamic structure of ontologies. For example, in the case of context change in ontology evolution, since the applicability of specific knowledge in one context does not automatically indicate the validity of the reasoning in the new context, thus the isomorphism between different states of the ontological structures and the knowledge they implied needs to be carefully analyzed. A common sense approach to get insight into a categorical diagrammatic structure and trace its various states, is to follow and chase the diagrams depicting the objects and morphisms, to check whether the diagram is commutative or not and ensure the equality of the compositions. A diagram is commutative “if and only if whenever p and p' are paths with the same source and target, then the compositions of morphisms along these two paths are equal” (Goguen , 1991). Putting two commutative diagrams together yields another commutative diagram. The diagram chasing along with commutative inference allow us the state space analysis to examine all the potential state transitions based on a derived transformational pattern. Therefore, one of the fundamental functionalities in ontology engineering that is the traceability of isomorphic reasoning processes through time from an initial ontology version to its current operational version can be performed.

In order to fully utilize the potential of reasoning and consistency checking in our framework, we are still working on this part as our ongoing research. Based on our experience in dealing with category theory, we feel that this formalism still has plenty of potential left to be used for ontology change management; thus, the categorical constructors such as *sketches*, *n-categories,* and *enriched categories* are due for examination in future work.

In the employed graph transformation approach, we restricted ourselves to using typed labeled graphs; however, in order to increase the expressivity of the graph representation, one may want to employ hypergraphs instead. Although using hypergraphs increases the expressivity of our formalism, it also induces a tremendous amount of complexity on the reasoning process (comparable with using OWL Full as the representation language). In addition, extending the types of interactions between different change actions at the internal and external levels of our introduced HD graphs could be another possible enhancement. Moreover, modeling a rule-based query engine that enables us to pose complex queries to changing knowledge bases is another possible task to be pursued. From our experience so far, some of the advantages of our introduced model are:

- The representation of events, time, actions, and operations employed in different scenarios of a dynamic ontological framework is an effective way to trace model changes;
- The independency of the framework from any particular domain, algorithm, protocol, or implementation language and its abstractness makes it more flexible for reuse in many application domains that use different formalisms and platforms;
- Employing transformation rules to perform changes ensures the consistency of the evolving ontologies in different states;
- Following the double-pushout approach for defining model transformation, which isolates the parts that remain unchanged, enables concurrent changes within an integrated knowledge-based system with minimum interruption to the system's operation.

The abstract categorical notions and their ability to specify objects and their relations in different levels of granularities, together with graph oriented semantics, enable us to describe the complex evolving structure in a consistent manner, which is beyond the capability offered by OWL's single semantic structure

## Acknowledgements

This paper is the extended version of the workshop paper (Shaban-Nejad and Haarslev, 2011) presented at 2011 International Workshop on Biomedical and Health Informatics. The authors would like to thank anonymous reviewers for their constructive comments and suggestions.

## References

Al-Aboud K, Al-Hawsawi K, Ramesh V, Al-Aboud D, AL-Githami A: An Appraisal of Terms Used in Dermatology. *SKINmed* 2003, 2(3):151–153.

Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D. et al. (2000) Gene Ontology: tool for theunification of biology. Nat Genet., 25: 25–29.

Asperti A, Longo G: *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist.* MIT Press, 1991.

Awodey S: *Category Theory*. Oxford University Press, 2006.

Bairoch, A. (2000) The ENZYME database in 2000. Nucleic Acids Res, 28: 304–305.

Baker, C.J.O., Shaban-Nejad, A., Su, X., Haarslev, V., and Butler, G. (2006) Semantic web infrastructure for fungal enzyme biotechnologists. *Journal of Web Semantics*, 4(3):168-180.

Baker, P.G., Brass, A., Bechhofer, S., Goble, C., Paton, N., Stevens, R. (1998) TAMBIS-Transparent Access to Multiple Bioinformatics Information Sources. Proc Int'l Conf Intell Syst Mol Biol, 6: 25–34.

Bryant D, Krause P: A review of current defeasible reasoning implementations. *Knowledge Eng. Review* 2008, 23(3):227-260.

Busatto G, Kreowski HJ, Kuske S: Abstract hierarchical graph transformation. *Mathematical Structures in Computer Science* 2005, 15(4):773–819.

Buttler D, Coleman M. T. et al.: Querying Multiple Bioinformatics Data Sources: Can Semantic Web Research Help? *SIGMOD Record* 2002, 31(4):59-64.

Capobianco M, Chesñevar CI, Simari GR: Argumentation and the Dynamics of Warranted Beliefs in Changing Environments. *Autonomous Agents and Multi-Agent Systems* 2005, 11(2):127-151.

Cimino JJ: Formal descriptions and adaptive mechanisms for changes in controlled medical vocabularies. *Methods of Information in Medicine* 1996, 35(3):202-210.

Coletti MH, Bleich HL: Technical milestone-Medical subject headings used to search the biomedical literature. *J. of the American Med Info. Asso.,* 2001, 8(4):317-323.

Cushion MT, Stringer JR: Has the Name Really Been Changed? It Has for Most Researchers. *Clinical Infectious Diseases* 2005, 41:1756-1758.

Dolin RH, Alschuler L, Boyer S, Beebe C, Behlen FM, Biron PV, Shabo A: HL7 Clinical Document Architecture, Release 2.0 (Published: Sun 12/12/2004).

Drewes F, Hoffmann B, Plump D: Hierarchical Graph Transformation. *J. Comput. Syst. Sci.* 2002, 64(2):249-283.

Ehrig H: Introduction to the algebraic theory of graph grammars (a survey). In: *proc. of workshop on Graph-Grammars and their App.*, *LNCS 73*, 1979, 1-69.

Ehrig H, Pfender M, Schneider HJ: Graph grammars: An algebraic approach. In: *Proc. of 14th Symposium on Foundations of Comp. Science*, IEEE, 1973, 167-180

Ehrig H, Orejas F, Prange U: Categorical Foundations of Distributed Graph Transformation. In: *Proc. of 3rd int'l Conference on Graph Transformations (ICGT'06)*, *LNCS 4178*, 2006, 215-229.

Ehrig H, Ehrig K, Prange U, Taentzer G: *Fundamentals of Algebraic Graph Transformation*. Monographs in Theori. Comp. Sci. An EATCS Series, Springer 2006.

Ehrig, H., Habel, A., Kreowski, H.J., and Parisi-Presicce, F. From Graph Grammars to High Level Replacement Systems. In Proc. of the 4th int'l workshop on Graph-Grammars and Their App. to Comp. Sci. Bremen, Germany, LNCS 532, Springer, 1990, pp. 269–291.

Ehrig H, Korff M, Löwe M: Tutorial Introduction to the Algebraic Approach of Graph Grammars Based on Double and Single Pushouts. *In Proc. of 4th Int'l Workshop Graph-Grammars and Their App., LNCS 532*, 1990, 24-37.

Eilenberg S, Mac Lane S: General Theory of Natural Equivalences. *T Am Math Soc* 1945, 58:231-294.

Elmasri R, Navathe SB: *Fundamentals of Database Systems*. 5th Ed., Addison Wesley 2007.

Engels G, Schürr A: Encapsulated hierarchical graphs, graph types, and meta types. *Electr. Notes Theor. Comput. Sci.* 2005, 2: 101–109.

Engels G, Heckel R: Graph Transformation as a Conceptual and Formal Framework for System Modeling and Model Evolution. In: *Proc. ICALP'00*, *LNCS 1853,* 2000, 127-150.

Goguen J: A Categorical Manifesto. *Mathematical Structures in Comp. Sci.* 1991*,* 1(1): 49–67.

GO Newsletter, No. 5 May 2007: http://www.geneontology.org/newsletter/archive/200705.pdf

GO Newsletter, Issue No. 1 May 2006. http://www.geneontology.org/newsletter/archive/200605.shtml#bp

GO Newsletter, Issue No. 6 Aug. 2007. (http://www.geneontology.org/newsletter/current-Newsletter.shtml)

GO Meeting collected notes, July 14-15,2001, Hosted by Judy Blake and the Jackson Lab in Bar Harbor, ME. compiled by L. Reiser. http://www.geneontology.org/minutes/collected_minutes.txt

Gruber TR: A translation approach to portable ontologies. *Knowledge Acquisition* 1993*,* 5(2):199-220.

Habel A, Heckel R, Taentzer G: Graph Grammars with Negative Application Conditions. *Fundam. Inform.* 1996, 26(3/4):287-313.

Health Level 7 Reference Information Model: http://healthinfo.med.dal.ca/hl7intro/CDA_R2_normativewebedition/infrastructure/rim/rim.htm

Heckel R (2006) Graph Transformation in a Nutshell. *Electr. Notes Theor. Comput. Sci.* 148(1):187-198.

Heckel R and Wagner A: Ensuring Consistency of Conditional Graph Grammars - A Constructive Approach. *Electronic Notes in Theoretical Comp. Sci.* 1995, 2:118-126.

Hilton E, Isenberg HD, Alperstein P (1992) Ingestion of yogurt containing Lactobacillus acidophilus as prophylaxis for candidal vaginitis. *Ann Intern Med*, 116:353-7.

Hoffmann B: From Graph Transformation to Rule-Based Programming with Diagrams. In: *Proc. of AGTIVE'99, LNCS 1779*, 1999, 165-180.

Kramer J, Magee J: The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering* 1990, 16(11):1293–1306.

Leung CY: Antifungal Therapy in Dermatology. *Journal of the Hong Kong Medical Association* 1990, 42(4):203-205.

Martin RF, Rickard K, Mejino JLV, Agoncillo AV, Brinkley JF, Rosse C: The Evolving Neuroanatomical Component of the Foundational Model of Anatomy. *Proc. of Am. Med. Info. Assoc. Fall Symp.* 2003, 927.

Mason O, Verwoerd M: *Graph Theory and Networks in Biology*. arXiv:q-bio/0604006v1 2008.

McCray, A.T., Nelson, S.J. (1995) The representation of meaning in the UMLS. *Method Inform Med* 34 (1/2):193-201.

Miller FP, Vandome AF, McBrewster J: *Interface (computer science): Interface, Abstraction (computer science), Polymorphism in Object- oriented Programming, Indirection, User Interface*. Alphascript Publishing 2010.

NHS Information Authority: The Clinical Terms Version 3 (The Read Codes): Managing Change: Description Change File. *Ref # 1999-IA-173 v1.0 issue,* March 2000.

Odds FC, Arai T, Di Salvo AC, Evans EGV, Hay RJ et al.: Nomenclature of fungal diseases. *A report from a Sub-Committee of the Intl' Society for Human and Animal Mycology (ISHAM),* 1992.

Odds FC, Rinaldi MG: Nomenclature of fungal diseases. *Curr. Top. Med. Mycol.* 1995, 6:33-46.

Padberg J. Integration of Categorical Frameworks: Rule-Based Refinement and Hierarchical Composition for Components. Applied Categorical Structures 2008, 16(3):333-364.

Palacz, W. (2004) Algebraic hierarchical graph transformation. Journal of Computer and System Sciences, 68(3): 497–520.

Palacz W: Hierarchical graph transformations with meta-rules. *Annales UMCS Informatica AI VIII* 2008, 2:89-96.

Robinson D, Comp D, Schulz E, Brown P. et al.: Updating the Read Codes: User-interactive Maintenance of a Dynamic Clinical Vocabulary. *J Am Med Inform Assoc* 1997, 4(6):465-472.

Rosse C, Mejino JL Jr: A reference ontology for bioinformatics: the Foundational Model of Anatomy. *J Biomed Inform.* 2003, 36(6):478-500.

Schomburg I, Chang A, Ebeling C, Gremse M, Heldt C, et al. (2004) BRENDA, the enzyme database: updates and major new developments. Nucleic Acids Res 32(DB issue):431–433.

Shaban-Nejad A, Haarslev V: Bio-medical Ontologies Maintenance and Change Management. In: *Sidhu & Dillon (Eds.) Biomedical Data and Applications*. Studies in Comput. Intelligence 224, Springer, 2009, 143-168.

Shaban-Nejad A, Haarslev V (2008) Incremental biomedical ontology change management through learning agents. In: *Proc. of 2nd KES Sympo. on Agent & Multi-Agent Sys. (KES-AMSTA '08), LNCS 4953*, 2008, 526–535.

Shaban-Nejad, A., Haarslev, V. (2011) An enhanced graph-oriented approach for change management in distributed biomedical ontologies and linked data. IEEE BIBM Workshops 2011, p. 615-622.

Sobel JD, Chaim W, Nagappan V, Leaman D: Treatment of vaginitis caused by Candida glabrata: use of topical boric acid and flucytosine. *American Journal of Obstetrics and Gynecology* 2003, 189(5):1297-1300.

Taentzer G. Distributed Graphs and Graph Transformation. *Applied Categorical Structures* 1999, 7(4):431-462.

Taentzer G (1994) Hierarchically Distributed Graph Transformation. In: *Proc. of 5th Int'l Workshop on Graph Grammars and App. (TAGT'94), LNCS 1073*, pp. 304-320.

Taentzer G, Fischer I, Koch M, Volle V: Distributed Graph Transformation with Application to Visual Design of Distributed Systems, In: *Ehrig H, Kreowski HJ et al. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, Vol 3, World Scientific* 1999.

Taentzer G, Goedicke M, Meyer T: Dynamic Change Management by Distributed Graph Transformation: Towards Configurable Distributed Systems. In: *Proc. of TAGT'98, LNCS 1764*, 1998, 179-193.

Van Eetvelde N, Janssens D: A Hierarchical Program Representation for Refactoring. *Electr. Notes Theor. Comput. Sci.* 2003, 82(7):91-104.

Whitmore I: Terminologia Anatomica: new terminology for the new anatomist. *Anat Rec.* 1999, 257(2):50-3

Whitmire SA: *Object Oriented Design Measurement.* John Wiley & Sons, 1997.