

OntoEdit: Guiding Ontology Development by Methodology and Inferencing

York Sure¹, Juergen Angele², and
Steffen Staab^{1,2}

¹ Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{sure, staab}@aifb.uni-karlsruhe.de
<http://www.aifb.uni-karlsruhe.de/WBS/>

² Ontoprise GmbH, Haid-und-Neu-Str. 7, 76131 Karlsruhe, Germany,
{angele}@ontoprise.de
<http://www.ontoprise.de/>

Abstract. Ontologies now play an important role for many knowledge-intensive applications for which they provide a source of precisely defined terms. The terms are used for concise communication across people and applications. OntoEdit is an ontology editor that has been developed keeping five main objectives in mind: 1. Ease of use. 2. Methodology-guided development of ontologies. 3. Ontology development with help of inferencing. 4. Development of ontology axioms. 5. Extensibility through plug-in structure. This paper is about the first four of these items.

1 Introduction

Ontologies now play an important role for many knowledge-intensive applications for which they provide a source of formally defined terms. They aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused, shared, and operationalized across applications and groups. However, because of their size, their complexity and their formal underpinnings ontologies are still far from being a commodity.

This urgent need motivated researchers in recent years to support the ontology engineering process. Mainly, we have seen three directions. Firstly, several seminal proposals for guiding and supporting the ontology development process have been proposed [UK95,LGPSS99,GW02]. Secondly, a considerable number of tools that support the ontology engineering process [DSW⁺99,NFM00,ACFLGP01,Dom98] have been developed. Third, inferencing mechanisms for large ontologies have been developed and implemented (cf., e.g., [Hor98]). However, only few of these seminal works have worked towards integrating these aspects.

OntoEdit is an ontology editor that is rather unique in its kind as it is based on a recent methodology for ontology development and as it makes comprehensive use of inferencing. In particular, OntoEdit focuses on three main steps for ontology de-

velopment (as described in [SSSS01]), viz. requirements specification, refinement and evaluation³.

First, all requirements of the envisaged ontology are collected. Typically an ontology engineer captures domain and goal of the ontology, design guidelines, available knowledge sources (e.g. reusable ontologies and thesauri etc.), potential users and use cases and applications supported by the ontology. Output of this phase is a semi-formal description of the ontology. Second, during the refinement phase the semi-formal description is extended and completely formalized into an appropriate representation language. The language is chosen according to the requirements for the ontology, e.g. identified applications that will be supported by the ontology. Output of this phase is a mature ontology (aka. “target ontology”). Third, the target ontology needs to be evaluated according to the requirement specifications and formal evaluation criteria (as proposed in the OntoClean methodology, cf. [GW02]). Typically this phase serves as a proof for the usefulness of developed ontologies.

Support for these development steps is a crucial objective that must be merged with the conflicting needs for ease of use and the construction of complex ontology structures.

In the following, we will first present a brief, real-life case study on configuration management that motivates our examples given thereafter. Section 3 explains theoretical and practical issues of the inference engine that constitutes the internal knowledge model of OntoEdit. The Sections 4 to 6 correspond to the core steps of the ontology development methodology sketched above. In particular, they elucidate how inferencing is used for supporting methodology-based ontology construction and evaluation and how this support is clad into a user-friendly environment.

2 Case study: Configuration Management

We have developed an ontology based configuration tool OnKo for the German Telecom. It represents a set of IT components together with their complex interrelationships. It supports an interactive configuration process of such components to IT systems and IT landscapes. It contains intelligent search and retrieval of already existing IT systems with similar functionality and thus integrates already existing experience of the company.

The ontology is exploited for navigation purposes. The user can switch between different views, i.e. *specialization view*, *part-of view* etc. to the available components and thus the user is able to choose the best presentation for the current configuration task. These views are represented as hierarchical trees and additional links between the nodes enabling a simple interactive search within the available IT components. By this way the user subsequently searches for components and adds them to the current configuration. In each step the interrelationships of the components of the current configuration are checked for consistency and feed back is delivered. Thus upcoming dead ends of configurations which would not work are recognized early. The system automatically derives new knowledge and makes further suggestions about possible extensions of the

³ In a recently accepted companion paper [SEA⁺02], we have described how OntoEdit supports collaborative ontology engineering.

current configuration which therefore must not be specified by the user. This “mixed initiative intelligence” strategy enables the development of a configuration in close cooperation with the user and thus supports the user without telling him what to do.

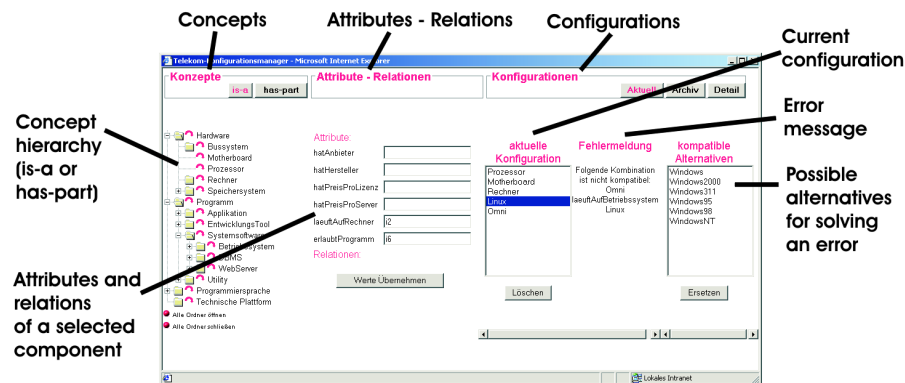


Fig. 1. OnKo an interactive configuration tool for the German Telecom

In Figure 1 a screenshot of OnKo is shown. In the left frame the user navigates within an is-a hierarchy of the components. This view may be switched to a part-of hierarchy. The attribute values of a selected component are editable in a form in the middle frame. The selected component together with its attribute values are used to derive attribute values of dependent components, i.e. the frequency of the processor is propagated upwards in the part-of hierarchy to the frequency of the entire computer. The current configuration with all its components is shown in the right frame. If a configuration contains inconsistent components which is checked by applying consistency rules, appropriate error and warning messages are immediately given and alternatives for a selected component are presented to the user which make the configuration consistent. In each step the current configuration may be compared to similar existing configurations in the company. This makes existing experience transparent to the user compiling a new configuration.

In our screenshot the current configuration contains two incompatible components: “Omni” does not work on “Linux”. To solve this incompatibility, the system suggests to use “Windows” or “Windows2000” or ... etc. instead of “Linux”. A user may now replace incompatible components to get a valid configuration for a computer system.

In the following, we will show some examples derived from the experiences in this case study.

3 Inferencing — Theoretical and Practical Issues

Theoretical Issues. In order to provide a clearly defined semantics to the knowledge model of OntoEdit, the knowledge structures of OntoEdit correspond to a well-understood logical framework, viz. F-Logic [KLW95] (“F” stands for “Frames”).

F-Logic allows for concise definitions with object oriented-like primitives (classes, attributes, OO-style relations, instances) that fit very nicely with the OntoEdit GUI. Furthermore, it also has PL-1 like primitives (predicates, function symbols). Furthermore,

F-Logic allows for axioms that further constrain the interpretation of the model. Axioms may either be used to describe constraints or they may define rules, e.g. in order to define a relation R by the composition of two other relations S and Q .

F-Logic rules have the expressive power of Horn-Logic with negation and may be transformed into Horn-Logic rules. The semantics for a set of F-Logic statements is defined by the well-founded semantics [GRS91]. This semantics is close to first-order semantics. In contrast to first order semantics not all possible models are considered but one “most obvious” model is selected as the semantics of a set of rules and facts. It is a three valued logic, i.e. the model consists of a set of true facts and a set of unknown facts and a set of facts known to be false.

Unlike Description Logics (DL), F-Logic does not provide means for subsumption [Hor98], but (also unlike DL) it provides for efficient reasoning with instances and for the capability to express arbitrary powerful rules, e.g. ones that quantify over the set of classes.

The most widely published operational semantics for F-Logic is the alternating fixed point procedure. This is a forward chaining method which computes the entire model for the set of rules, i.e. the set of true and unknown facts. For answering a query the entire model must be computed (if possible) and the variable substitutions for the query are then derived. In contrast, our inference engine Ontobroker performs a mixture of forward and backward chaining based on the dynamic filtering algorithm [KL86] to compute (the smallest possible) subset of the model for answering the query. In most cases this is much more efficient than the simple evaluation strategy. These techniques stem from the deductive data base community and are optimized to deliver all answers instead of one single answer as e.g. resolution does.

Within our F-Logic compiler F-Logic statements are translated to normal programs. Normal programs are horn programs where rules may contain negated literals in their bodies. Horn logic is turing complete, thus F-Logic programs are not decidable in principle. The semantics defined for these normal programs is the wellfounded semantics [Gel93]. In contrast to the stratified semantics the wellfounded semantics is also applicable for rules which depend on cycles containing negative rule bodies. Because F-Logic is very flexible, during the translation to normal programs such negative cycles often arise. In [GRS91] the alternating fixpoint has been described as a method to operationalize such logic programs. This method has been shown to be very inefficient. Therefore our inference engine realizes dynamic filtering [KL86] which combines top-down and bottom-up inferencing. Together with an appropriate extension to compute the wellfounded semantics this method has been proven to be very efficient compared to other horn based inference engines.

We have shown this for test cases where all paths in large graphs are computed. The results are shown in Figure 2. We measured the time in milliseconds Ontobroker (versions 3.1 and 3.2) and XSB⁴ needed for computing all paths of a certain number of graphs given. XSB is a “Logic Programming and Deductive Database system for Unix and Windows” and is a comparable inference engine to Ontobroker. XSB needs exponentially more time for computation when data sets rise and therefore does not scale up very well. Both Ontobroker versions have almost a linear growth of time, Ontobroker

⁴ Available at <http://xsb.sourceforge.net/>.

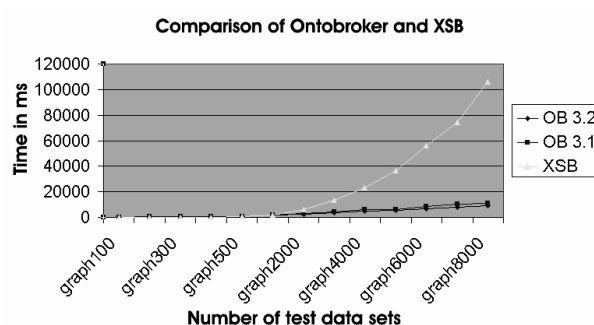


Fig. 2. Comparison of Ontobroker and XSB

therefore scales up very nicely for this scenario. However, this test is not exhaustive at all and therefore only provides limited information about the scale ratio of the two systems.

Practical Issues. Our inference engine Ontobroker (cf. [DEFS99]) comes with several features that makes it adequate as a backbone for an ontology editor. In particular, it provides:

- A namespace mechanism: Thus, several ontologies (or ontology parts) may be syntactically split into modules and processed by different inference engines.
- Switch-off: It is possible to switch of (possibly singleton) sets of definitions. Thus, one may test interactions and easily distinguish between modules.
- DB Connectors: Thus, one may easily map db tables into predicates via JDBC.
- User-definable built-Ins: Besides of standard built-ins like “multiply”, the user may define his own ones for special purposes.
- An extensive API: Thus, one may remotely connect to the inference engine and one may also import and export several standards (e.g., RDF(S)).

The use of all but the last of these items will be explained in more details subsequently.

4 Requirements Specification

Like in software engineering and as proposed by [LGPSS99], we start ontology development with collecting requirements for the envisaged ontology. By nature this task is performed by a team of experts for the domain accompanied by experts for modeling. The outcome of this phase is (i) a document that contains all relevant requirement specifications (domain and goal of the ontology, design guidelines, available knowledge sources, potential users and use cases and applications supported by the ontology) (ii) a semi-formal ontology description, i.e. a graph of named nodes and (un-)named, (un-)directed edges, both of which may be linked with further descriptive text.

To operationalize a methodology it is desirable to have a tool that reflects and supports all steps of the methodology and guides users step by step through the ontology engineering process. Along with the development of the methodology we therefore ex-

tended the core functionalities of OntoEdit by two plug-ins to support first stages of the ontology development, viz. OntoKick and Mind2Onto⁵.

OntoKick targets at (i) creation of the requirement specification document and (ii) extraction of relevant structures for the building of the semi-formal ontology description. Mind2Onto targets at integration of brainstorming processes to build relevant structures of the semi-formal ontology description. As computer science researchers we were familiar with software development and preferred to start with a requirement specification of the ontology, i.e. OntoKick. People who are not so familiar with software design principles often prefer to start with “doing something”. Brain storming is a good method to quickly and intuitively start a project, therefore one also might begin the ontology development process with Mind2Onto.

OntoKick OntoKick is an OntoEdit plug-in that extends the functionality of OntoEdit by support for requirements specification (describing the plug-in structure itself is beyond the scope of this paper). OntoKick allows for describing important aspects of the ontology, viz.: the domain and the goal of the ontology, the design guidelines, the available knowledge sources (e.g. domain experts, reusable ontologies etc.), the potential users, the use cases, and the applications supported by the ontology. OntoKick stores these descriptions with the ontology definitions.

As proposed by [UK95], we use competency questions (CQ) to define requirements for an ontology. Each CQ defines a query that the ontology should be able to answer and therefore defines explicit requirements for the ontology. Typically, CQs are derived from interviews with domain experts and help to structure knowledge. We take further advantage of using them to create an initial version of the semi-formal description of the ontology. Based on the assumption that each CQ contains valuable information about the domain of the ontology we extract relevant concepts and relations (see example below). Furthermore, OntoKick establishes and maintains links between CQs and concepts derived from them. This allows for better traceability of the origins of concept definitions in later stages.

We illustrate the usage of CQs by an example from our case study. Figure 3 shows a screenshot of our ontology environment OntoEdit presenting the configuration management ontology. In the left column, one may recognize an excerpt of the *is-a* hierarchy of concepts from our case study ontology. One concept is selected, i.e. highlighted, (“Bussysteme” which is the German word for bus systems of microprocessors), in the right column all relations and attributes for the selected concept are shown. A context menu for the selected concept offers possibilities for typical modification (e.g. id, external representations and documentations in multiple languages etc.) including the feature to show corresponding competency questions.

The methodology is supported as follows. First, the ontology engineer has interviewed an expert in configuration management. Thereby they have identified CQs, e.g. “Which bus systems are available for the AS400?” (English equivalent of the CQ shown in Figure 3). Based on these CQs the ontology engineer has created the semi-formal description of the ontology. He has identified relevant concepts and relations from of the above-mentioned CQ, e.g. “Bussysteme”. After capturing CQs and modeling the ontol-

⁵ Describing the plug-in framework is beyond the scope of this paper, it is described in [Han01]. In a nutshell, one might easily expand OntoEdit’s functionalities through plug-ins.

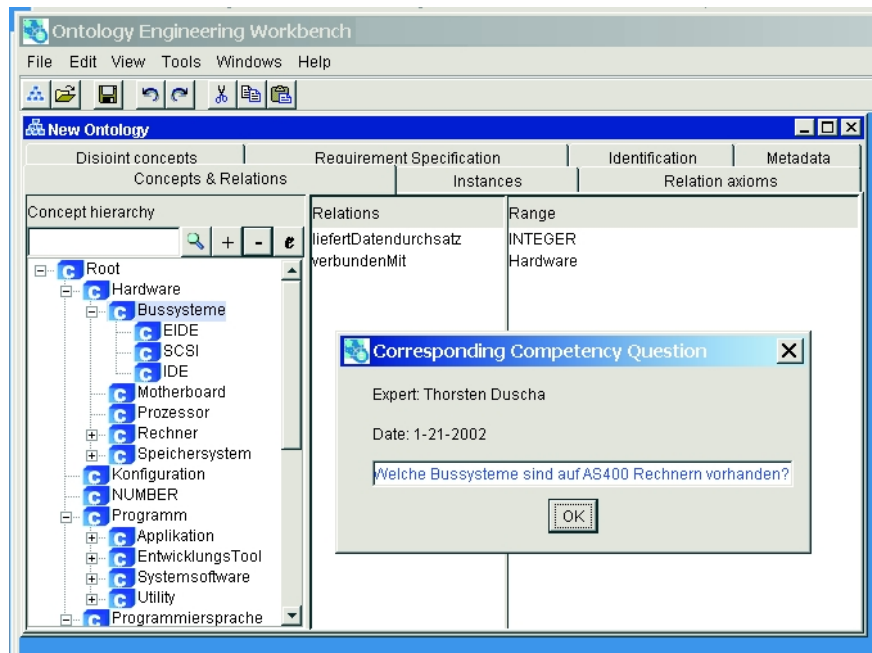


Fig. 3. A Competency Question (CQ) in the Ontology Engineering Environment OntoEdit

ogy with OntoEdit the ontology engineer has been able to retrieve the corresponding CQ for each concept and relation, helping him to identify the context in which they were modelled.

Mind2Onto is a plug-in for supporting brainstorming and discussion about ontology structures. Especially during early stages of projects in general, brainstorming methods are commonly used to quickly capture pieces of relevant knowledge. A widely used method are mind mapsTM [Buz74], they were originally developed to support more efficient learning and evolved to a management technique used by numerous companies. In general, a mind mapTM provides information about a topic that is structured in a tree. Each branch of the tree is typically named and associatively refined by its subbranches. Icons and pictures as well as different colors and fonts might be used for illustration based on the assumption that our memory performance is improved by visual aspects. There already exist numerous tools for the electronically creation of mind mapsTM. Many people from academia and industry are familiar with mind mapsTM and related tools – including potential ontology engineers and domain experts. Therefore the integration of electronic mind mapsTM into the ontology development process is very attractive (cf. e.g. [LS02]).

We relied on a widely used commercial tool⁶ for the creation of mind mapsTM. It has advanced facilities for graphical presentations of hierarchical structures, e.g. easy to use copy&paste functionalities and different highlighting mechanisms. Its strength but also its weakness lies in the intuitive user interface and the simple but effective

⁶ MindManagerTM 2002 Business Edition, cf. <http://www.mindjet.com>

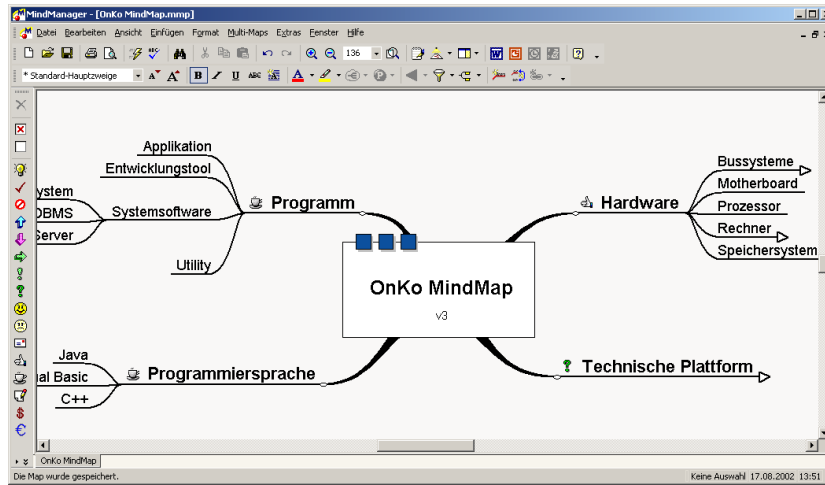


Fig. 4. Mind Map for OnKo

usability, which allows for quick creation of mind maps™ but lacks of expressiveness for advanced ontology modeling. By nature, mind maps™ have (almost) no assumptions for it's semantics, i.e. branches are somehow “associatively related” to each other. This assumption fits perfectly well during early stages of ontology development for quick and effective capturing of relevant knowledge pieces and makes the mind map™ tool a valuable add-on. Figure 4 shows a draft mind map™ for the OnKo system.

Mind2Onto integrates the mind map™ tool into the ontology engineering methodology. Currently OntoEdit and the mind map™ tool interoperate through import and export facilities based on XML.

5 Ontology Refinement

In the ontology refinement phase the semi-formal description of the ontology is extended and completely formalized in order to make it machine understandable and processable. In this phase we take advantage of the inferencing capabilities of OntoEdit for several purposes.

Reuse by semantic integration. Reuse in the refinement phase comes in two flavors. First, we may exploit existing thesauri, e.g. ones that are stored in databases by semantically integrating them into the inferencing model. As in general information integration [WG97], this involves two steps. The first step concerns the mapping onto a common data model. For this purpose, we take advantage of the inference engine's capabilities, viz. to read in RDF(S) [W3C99], to connect to relational databases, to provide further built-ins (e.g. for connection to XML repositories). The outcome of this step are data in F-Logic structures, but with some rather arbitrary semantics. In the second step, we build rules to map the outcome of the first step into the desired categories. For instance, we may map a database table-like structure into a target structure of sub- and superconcepts. An example may be given with a database table that contains WordNet hyponyms

and synonyms. Our example is based on a locally installed MySQL database system that contains a 'wordnet-db' database.

1. In the first step we map this table into an equivalent predicate HYPONYM:

$$\text{FORALL } C, D \text{ HYPONYM}(C, D) \leftarrow \\ \text{DBACCESS}(\text{hyponym}, F(\text{'sub'}, C, \text{'super'}, D), \text{'mysql'}, \\ \text{'wordnet-db'}, \text{'localhost'}).$$
2. In the second step, we define the objects of the predicted hyponym to be subconcepts of *Computer* if it is known that one of their superconcepts is a subconcept of *Computer*:

$$\text{FORALL } C, D \text{ } C :: D \leftarrow \text{HYPONYM}(C, D) \text{ AND } D :: \text{Computer}.$$

Taken the two steps together, this means: "Specify every C to be a subclass of D if in the MySQL database called 'wordnet-db' on my local machine the table called hyponym there is a row with attribute 'sub' being C and 'super' being D and simultaneously D is already known to be a subclass of *Computer*."

Combining such a query for hyponyms with search for synonyms yields about 100 terms, not all of which are obvious and which may be considered for inclusion in the ontology. Thus, one may easily reuse existing thesauri or database schema in order to generate a large number of concepts fast.

Reusing axioms integrating them into the ontology. Secondly, one may reuse axiom definitions from a library of ontology modules that are distinguished by namespace mechanism. A set of axiom definitions specified in one domain is reusable in another domain by the inference engine's capability to store and load axioms from a library to and into different namespaces in a way that is reusable for another domain.

For instance, *partonomic role propagation* may be given for a medical ontology (cf. [HSR99b] for a comprehensive description and appendix A for an illustrating example), but it has been reused for describing properties of computer systems. The underlying idea of partonomic role propagation is that some properties of parts of a system are propagated to the whole. For instance, the clock frequency of the CPU is frequently used as being descriptive of the overall computer system — while others like frequency rates of the bus system are not used for that abstracting purpose.⁷

We specify an axiom library by a meta-predicate. In this current example, this predicate is named PARTONOMICROLEPROPAGATION. This meta-predicate takes four input parameters (cf. the formal specification in Table 1):

1. The relation that is propagated (FREQUENCYOF), because not every relation is propagated from a part to a whole.
2. The relation that is propagating (PHYSICALPARTOF), because not every relation that may be propagated is propagated along all part-whole relations.
3. The whole up to which the relation is maximally propagated (*ComputerSystem*), because propagation may be stopped (e.g. FREQUENCYOF should not be propagated to a car that the computer system is a part of).

⁷ Similarly, the color of the car body is typically equated with the color of the car. This is not true for the color of the seats, though seats and car body are both parts of the car.

4. The concept for the instances of which the relation may be propagated (e.g., a (here fictitious) *WheelFrequency* might not be propagated), because not every class is treated the same.

0	PARTONOMICROLEPROPAGATION(FREQUENCYOF, PHYSICALPARTOF, <i>ComputerSystem</i> , <i>ClockFrequency</i>)
1	$\forall x, y, z, R, S, C, D x[R \rightarrow z] \leftarrow$ PARTONOMICROLEPROPAGATION(R, S, C, D) AND $x : D$ AND $x[R \rightarrow y]$ AND $y[S \rightarrow z]$ AND PARTINSTANCEOF(z, C, S).
2	$\forall z, C, S$ PARTINSTANCEOF(z, C, S) \leftarrow $\exists E z : E$ AND PARTOFALONG(E, C, S).
3	$\forall C, S$ PARTOFALONG(C, C, S).
4	$\forall E, C, S$ PARTOFALONG(E, C, S) \leftarrow $\exists F$ PARTOFALONG(E, F, S) AND $\exists Q F[Q \rightarrow C]$ AND $Q :: S$.
5	$\forall x, y, S, R x[S \rightarrow y] \leftarrow R :: S$ AND $x[R \Rightarrow y]$.

Table 1. Partonomic Role Propagation

Reusing axioms applying them to the ontology. Besides of integrating axioms from a library into the ontology, one may apply axioms in order to enforce constraints on the ontology. We distinguish three major types:

1. **Axioms of F-Logic:** They are an integral part of the F-Logic definition. However, not all of them are needed for inferencing during the usage of the ontology. For instance, type coercion at the conceptual level:

$$\text{FORALL } C, D, E, A, T E :: T \leftarrow C[A \Rightarrow T] \text{ AND } D :: C[A \Rightarrow E].$$
“Specify E as a subclass of T if some concept C has an attribute A of type T and a subclass D of C has an attribute A with type E .”
2. **Axioms for domain-specific consistency:** They enforce consistency constraints at building time. E.g., they may enforce that the domain specific relation HASPHYSICALPART is without cycles:

$$\text{NONCYCLIC}(\text{HASPHYSICALPART}).$$

$$\text{FORALL } X, R \text{ UNDEFINED } \leftarrow \text{NonCyclic}(R) \text{ AND } X[A \rightarrow X].$$
“HASPHYSICALPART is of type NONCYCLIC. Indicate consistency violation if an attribute A is of type NonCyclic and X is related via A to itself.”
3. **Axioms enforcing modeling policies:** Such axioms do not add to the semantic description, but they are applied in order to enforce semiotic constraints, e.g. that no subconcept should have more than n subconcepts, that no hierarchy should be deeper than m , or that every attribute symbol should begin with a lower case letter:

$$\text{FORALL } A \text{ UNDEFINED } \leftarrow$$

$$\text{EXISTS } X, Y X[A \Rightarrow Y] \text{ AND NOT regexp}(\text{“}^{\wedge}[a - z]^{\wedge}\text{”, } A).$$
“Indicate consistency violation if there is an attribute symbol A between some classes X, Y and it does not match with a string beginning with lowercase alphabetical letters.”

The three types of axioms just described are not integrated into the ontology, because once the ontology is fixed and remains unchanged they are not violated anyway. Still, switching them off improves performance, because they need not be revisited and checked again.

6 Evaluation

The last step in ontology development is about evaluating the formal ontology.

Analysis of Typical Queries. For this purpose, the Ontology engineer may interactively construct and save instances and axioms into modules. OntoEdit contains a simple instance editor that the ontology engineer can use to create test sets. The test set can be automatically processed and checked for consistency. Once, the ontology goes into the evolution phase and needs changes to remain up-to-date, these test sets may be re-used for checking validity of the ontology.

For instance, one may create a test case for partonomic role propagation (test case cf. Table 2, the partonomic role propagation is explained in Appendix A or, more detailed, in [SEM01]), viz. an instance CPU1 of CPU with a clock frequency of 1600MHz (line 1), an instance PC1 of PC which has a MB456 (line 2) that is of type Motherboard and that has the CPU1 on board (line 3).

The test case is completed by the query that asks for all clock frequencies of all PCs. This query is reformulated as a rule (line 4), in order to allow for comparison with an intended set of result tuples (line 5), by a generally applicable rule (lines 6 – 8).

Test instances
1 CPU1:CPU[hasClockFrequency→'1600MHz'].
2 PC1:PC[hasPhysicalPart→MB456].
3 MB456:Motherboard[hasPhysicalPart→CPU1].
Query formulated as test query
4 FORALL X, Y test1(X, Y) ← X :CPU[hasClockFrequency→ Y].
Intended set of result tuples
5 test2(CPU1,'1600MHz').
General rule for comparing query results with intended results
6 FORALL X, Y Undefined ←
7 (test1(X, Y) AND NOT test2(X, Y)) OR
8 (test2(X, Y) AND NOT test1(X, Y)).

Table 2. Formalizing Test Cases

In this small example, we have provided only one result tuple for testing, but the specification is modular and general enough to easily integrate sets of test cases.

Error Avoidance and Location. While the generation and validation of test cases allows for detection of errors, it does not really support the localization of errors. The set of all axioms, class and instance definitions express sometimes complex relationships

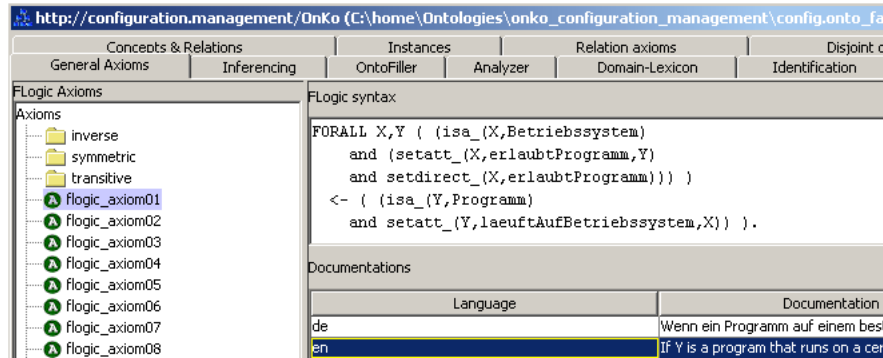


Fig. 5. Specifying F-Logic axioms in OntoEdit

and axioms often interact with other axioms when processed. Thus it is frequently very difficult to overview the correctness of a set of axioms and detect the faulty ones.

In principle there exist three types of problems with axioms:

- Axioms contain typing errors like variables not specified by a quantifier, typos in concept names or relationship names etc.
- Axioms contain semantic errors, i.e. the rules do not express the intended meaning.
- Performance issues, like axioms defined such that evaluation needs a lot of time, which is not always easily recognizable by the user.

In order to avoid problems, OntoEdit offers several means:

1. Some axiom definitions may be generated by asserting through clicks that relations or concepts belong to particular types. OntoEdit allows for defining several properties of relationships by clicking on the GUI, viz. symmetry, transitivity and composition of relations. Database connections as shown above in Section 5 need not be specified in F-Logic, but can be composed by drag-and-drop.
2. For other types of axioms a graphical rule editor is available which avoids syntactical errors, delivers axioms which are optimal in their performance (as seen in isolation from other axioms) and supports users not familiar in F-Logic.
3. Third, there are axioms that cannot be specified by either 1. or 2. For them, OntoEdit provides at least syntax highlighting in order to support the user avoiding syntactical errors.

In order to locate problems, OntoEdit takes advantage of the inference engine Ontobroker itself, which allows for introspection and also comes with a debugger. Axioms are operationalized by posing queries (e.g. on the test cases specified as seen above). Based on queries one may pursue several alternatives:

First, a very simple but effective method to test axioms with test cases is to switch off and switch on axioms or parts of the axiom premises. The different answers from Ontobroker then allow to draw conclusions about possible errors.

Second, for a given query the results and their dependencies on existing test instances and intermediate results may be examined by visualizing the proof tree. This

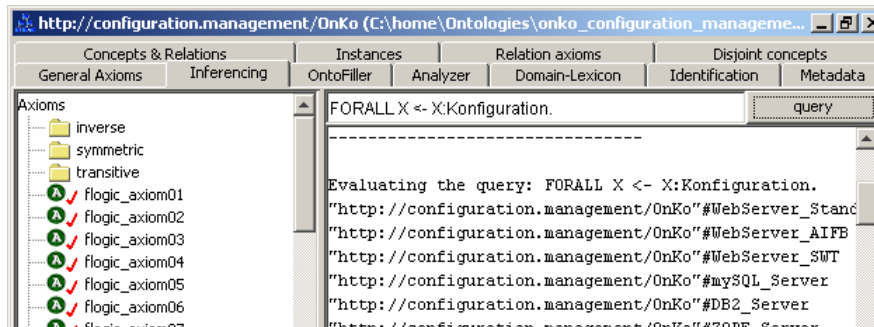


Fig. 6. Inferencing with Ontobroker in OntoEdit

proof tree shows graphically which instances or intermediate results are combined by which rules to the final answers. Thus the drawn inferences may be traced back to the test instances and semantic errors in rules may be discovered.

Third, the inference engine may be “observed” during evaluation. A graphical presentation of the set of axioms as a graph structure indicates which axiom is evaluated at the moment and also shows which intermediate results have already been created up to now and thus “have flown” in the axiom graph to other axioms. This also gives the user a feeling how much time it is needed to evaluate special rules.

An example is given by the two Figures 5 and 6. The former illustrates how F-Logic axioms can be specified in OntoEdit in the “General Axioms” plugin (the more advanced graphical rule editor is near completion). On the left side are listed all specified axioms, on the right side one may see a selected F-Logic axiom as well as its documentation in different languages. The latter shows the GUI of the “Inferencing” plugin for OntoEdit, that integrates Ontobroker into OntoEdit. On the left side each previously specified axiom can be switched on or off. On the right side one may enter an F-Logic query (here: give me all instances of the concept “Konfiguration”) which is subsequently answered by Ontobroker. The results are presented below the query, here one might see that for each result item the name of an instance includes the corresponding namespace.

In the future, it is planned to take more care about the efficient construction of efficiently handable ontologies. For this purpose, OntoEdit will provide a profiler that will deliver statistics about evaluation times.

Formal Evaluation with OntoClean. Beside the above mentioned process oriented and pragmatic evaluation methods, there exist also formal evaluation methodologies for ontologies. One of the most prominent is the OntoClean methodology (cf., e.g., [GW02]), which is based on philosophical notions. It focuses on the cleaning of taxonomies and e.g. is currently being applied for cleaning the upper level of the WordNet taxonomy (cf. [GGOB02]). Core to the methodology are the four fundamental ontological notions of *rigidity*, *identity*, *unity* and *dependence*. By attaching them as meta-relations to concepts in a taxonomy they are used to represent the behavior of the concepts. I.e. these meta-relations impose constraints on the way subsumption is used to model a domain (cf. [GW00]). We can only briefly and simplified sketch the methodology (please note that *property* is used here in our sense of “concept”):

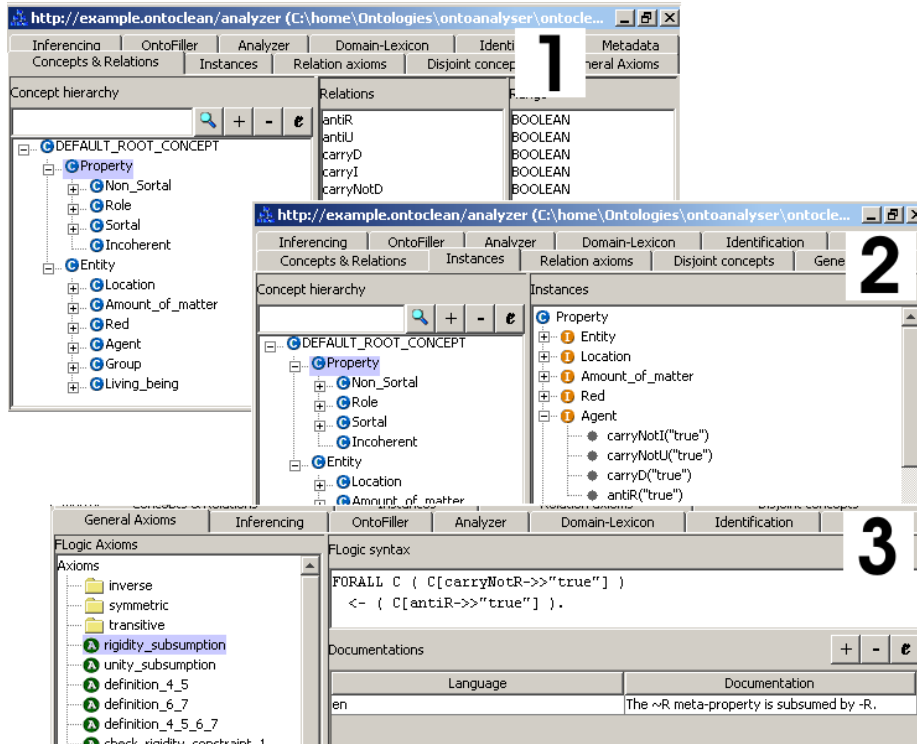


Fig. 7. Implementation of OntoClean in OntoEdit

Rigidity is defined based on the idea of essence. A *property* is essential to an individual if and only if necessarily holds for that individual. Thus, a *property* is rigid (+R) if and only if it is necessarily essential to all its instances. A *property* is non-rigid (-R) if and only if it is not essential to some of its instances, and anti-rigid (\sim R) if and only if it is not essential to all its instances. An **identity** criterion (IC) is carried by a *property* (+I) if and only if all its instances can be (re)identified by means of a suitable “sameness” relation. A *property* supplies an identity criterion (+O) if and only if such criterion is not inherited by any subsuming *property*. An individual X is constantly **dependent** on Y if and only if, at any time, X cannot be present unless Y is fully present, and Y is not part of X . A *property* is constantly dependent if and only if, for all its instances, there exists something they are constantly dependent on. **Unity** is defined by saying that an individual is a whole if and only if it is made by a set of parts unified by a relation R . A *property* P is said to carry unity (+U) if there is a common unifying relation R such that all the instances of P are wholes under R . A *property* carries anti-unity (\sim U) if all its instances can possibly be non-wholes.

Based on these meta-relations OntoClean classifies concepts into categories (Sortal, Non-sortal, Role etc.). E.g., a concept that is tagged with “+O +I +R” is called a “Type”. Beside these meta-relations OntoClean contains rules that can be applied to evaluate the correctness of a given taxonomy. For instance, a rule suggested in OntoClean is “a property carrying anti-unity has to be disjoint of a property carrying unity”. As a

consequence, “a property carrying unity cannot be a subclass of a property carrying anti-unity” and “a rigid property and an anti-rigid property are ever disjoint”, to name but a few.

To implement the OntoClean methodology in OntoEdit⁸, we formalized the meta-relations and classifications as a “meta ontology” that can be used to classify concepts of an ontology. We modelled both, the “meta ontology” and an example ontology (the example is taken from [GW02], we started to implement the methodology but did not yet apply it) that has to be evaluated, in OntoEdit and specified each concept of the regular ontology, i.e. all subconcepts of “Entity”, as an instance of the top-level concept “Property” of the meta ontology through an axiom:

$$\text{FORALL } A : \text{Property} \leftarrow A :: \text{Entity}.$$

Figure 7 shows the subsequent steps: (1) model the ontologies, (2) fill the meta relations with values (i.e. tag the concepts with “carryR” (+R) etc.) and (3) specify the definitions and constraints from OntoClean as axioms. Like shown in Figure 6 one can now ask queries to find inconsistencies according to the OntoClean methodology.

7 Related Work

There exist various ontology engineering environments, which we divide into two categories: *with* and *without* inferencing support.

A good overview, viz. a comparative study of existing tools up to 1999, is given in [DSW⁺99]. Typically the internal knowledge model of ontology engineering environments is capable of deriving is-a hierarchies of concepts and attached relations. On top of that we provide facilities for axiom modeling and debugging. Naturally, it could not fully consider the more recent developments, e.g. Protégé [NFM00], and WebODE [ACFLGP01].

About WebODE, [ACFLGP01] mentions that it offers inferencing services (developed in Prolog) and an axiom manager (providing functionalities such as an axiom library, axiom patterns and axiom parsing and verification), but the very brief mentioning of these functionalities is too short to assess precisely.

A system well-known for its reasoning support is OilEd in combination with the description logics (DL) reasoner FaCT [BHGS01]. Their focus is to use reasoning to check class consistency and to infer subsumption relationships which are typical DL tasks. We may not provide subsumption, but we provide extensive reasoning on instances — in particular rules — or axioms that specify user-definable consistency constraints. Furthermore, we support the whole methodology cycle for developing ontologies.

Environments like Protégé [NFM00] or Chimaera [MFRW00] offer sophisticated support for ontology engineering and merging of ontologies. Protégé has also a flexible plugin-structure, that allows for modular extension of the functionalities. However, they

⁸ There is also the group from the Artificial Intelligence Laboratory of the Technical University of Madrid (UPM) working on the integration of the philosophically oriented OntoClean [GW02] methodology with the process oriented METHONTOLOGY [LGPSS99] by extending the WebODE [ACFLGP01] ontology development environment (cf. <http://www.ontoweb.org/workshop/ontoweb2/slides/ontocleansig3.pdf>)

do not provide comprehensive methodological support for ontology engineering and it is also difficult to assess the extent that they exploit reasoning capabilities.

8 Conclusion

In this paper we have presented OntoEdit, a sophisticated ontology editor that supports methodology-based ontology construction and that takes comprehensive advantage of its inferencing capabilities. OntoEdit also has some features that could not be presented here in full detail, e.g. an extremely capable plug-in structure (cf. [Han01]), a lexicon component and support for collaborative engineering of ontologies (cf. [SEA⁺02]).

Obviously, there are a large number of ontology construction tools now available and many of them offer very intriguing features that are not in OntoEdit. However, according to our experiences the combination of a methodological basis with comprehensive reasoning on class and instance definitions with Ontobroker is a very powerful paradigm that has not been exploited to the extent that OntoEdit does.

For the future, OntoEdit is planned to be developed in several directions: (1) new im- and exports will be developed and (2) the integration of ontology construction with requirements specification documents will be generalized by means of semantic document annotation, (3) stronger support for the integration of mind mapsTM into the ontology development process, (4) finish the OntoClean implementation and apply it, to name but a few.

Acknowledgements. We thank especially our colleagues Alexander Maedche (now: Research Center FZI, Research Group WIM, Karlsruhe, Germany) and Dirk Wenke (now: chief developer of OntoEdit at Ontoprise GmbH, Karlsruhe, Germany). Together they initiated the development of OntoEdit, which is now being continued by the constant efforts of Dirk. We also thank Siggie Handschuh (Institute AIFB, University of Karlsruhe) for his plug-in framework OntoMat. Research for this paper was partially funded by EU in the project IST-1999-10132 “On-To-Knowledge”.

References

- [ACFLGP01] J.C. Arprez, O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. WebODE: a scalable workbench for ontological engineering. In *Proceedings of the First International Conference on Knowledge Capture (K-CAP) Oct. 21-23, 2001, Victoria, B.C., Canada, 2001*.
- [BHGS01] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A reason-able ontology editor for the semantic web. In *KI-2001: Advances in Artificial Intelligence*, LNAI 2174, pages 396–408. Springer, 2001.
- [Buz74] T. Buzan. *Use your head*. BBC Books, 1974.
- [DEFS99] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman et al., editor, *Database Semantics: Semantic Issues in Multimedia Systems*. Kluwer Academic, 1999.
- [Dom98] J. Domingue. Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the web. In *Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop, April 18th-23rd. Banff, Canada, 1998*. <http://ksi.cpsc.ucalgary.ca/KAW/KAW98/KAW98Proc.html>.

- [DSW⁺99] A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, and V. R. Benjamins. Wondertools? A comparative study of ontological engineering tools. In *Proc. of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management. Banff, Alberta, Canada. October 16-21, 1999*, 1999. <http://sern.ucalgary.ca/KSI/KAW/KAW99/papers.html>.
- [Gel93] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47(1):185–221, 1993.
- [GGOB02] A. Gangemi, N. Guarino, A. Oltramari, and S. Borgo. Cleaning-up WordNet’s top-level. In *Proc. of the 1st International WordNet Conference*, January 2002.
- [GRS91] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.
- [GW00] N. Guarino and C. Welty. A formal ontology of properties. In R. Dieng and O. Corby, editors, *Knowledge Engineering and Knowledge Management: Methods, Models and Tools. 12th International Conference, EKAW2000*, pages 97–112. Springer Verlag, 2000.
- [GW02] N. Guarino and C. Welty. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 2(45):61–65, 2002.
- [Han01] Siegfried Handschuh. Ontoplugins – a flexible component framework. Technical report, University of Karlsruhe, May 2001.
- [Hor98] I. Horrocks. Using an expressive description logic: Fact or fiction? In *Proceedings of KR 1998*, pages 636–649. Morgan Kaufmann, 1998.
- [HSR99a] Udo Hahn, Stefan Schulz, and Martin Romacker. Partonomic reasoning as taxonomic reasoning in medicine. In *Proc. of AAAI-99*, pages 271–276, 1999.
- [HSR99b] U. Hahn, S. Schulz, and M. Romacker. Part-whole reasoning: A case study in medical ontology engineering. *IEEE Intelligent Systems*, 14(5):59–67, 1999.
- [KL86] M. Kifer and E. Lozinskii. A framework for an efficient implementation of deductive databases. In *Proceedings of the 6th Advanced Database Symposium*, pages 109–116, Tokyo, August 1986.
- [KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.
- [LGPSS99] M. F. Lopez, A. Gomez-Perez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using Methontology and the Ontology Design Environment. *Intelligent Systems*, 14(1):37–45, January/February 1999.
- [LS02] T. Lau and Y. Sure. Introducing ontology-based skills management at a large insurance company. In *Proc. of the Modellierung 2002*, Tutzing, Germany, March 2002.
- [MFRW00] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of KR 2000*, pages 483–493. Morgan Kaufmann, 2000.
- [NFM00] N. Fridman Noy, R. Fergerson, and M. Musen. The knowledge model of Protégé-2000: Combining interoperability and flexibility. In *Proceedings of EKAW 2000*, LNCS 1937, pages 17–32. Springer, 2000.
- [SEA⁺02] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. OntoEdit: Collaborative ontology development for the semantic web. In *Proc. of the International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardinia, Italia.*, 2002.
- [SEM01] S. Staab, M. Erdmann, and A. Maedche. Ontologies in RDF(S). *ETAI Journal – Section on Semantic Web (Linkoepping Electronic Articles in Computer and Information Science)*, 9(6), 2001.

- [SSSS01] S. Staab, H.-P. Schnurr, R. Studer, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems, Special Issue on Knowledge Management*, 16(1), Jan/Feb 2001.
- [UK95] M. Uschold and M. King. Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, Montreal, Canada, 1995.
- [W3C99] W3C. RDF Schema specification. <http://www.w3.org/TR/PR-rdf-schema/>, 1999.
- [WG97] G. Wiederhold and M. Genesereth. The Conceptual Basis for Mediation Services. *IEEE Expert / Intelligent Systems*, 12(5):38–47, September/October 1997.

A Example of Partonomic Role Propagation

Partonomic role propagation is about propagating particular property values from parts to wholes. For instance, if the engine of my car is defunct, the whole car is defunct (“defunct” being propagated from the part “engine” to the whole “car”). However, if the rear window is broken, it might be less safe to drive the car, but it would be strange to consider it defunct.

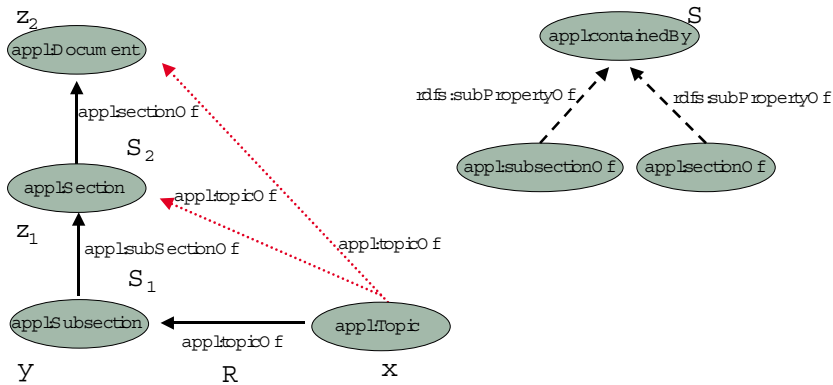


Fig. 8. Partonomic Role Propagation

Figure 8 depicts an example that propagates topics from subparts of documents to superparts (using the graphical notion of RDF(S), cf. [W3C99]). It takes four input parameters:

1. The relation that is propagated (TOPICOFF in Figure 8), because not every relation is propagated from a part to a whole.
2. The relation that is propagating (CONTAINEDBY in Figure 8), because not every relation that may be propagated is propagated along all part-whole relations.
3. The whole up to which the relation is maximally propagated (*Document* in Figure 8), because propagation may be stopped (e.g. TOPICOFF may be considered to be not propagated to an additional *Library*).
4. The concept for the instances of which the relation may be propagated (e.g., *Topic* in Figure 8), because not every class is treated the same (cf. [HSR99a] for comprehensive examples).