Automated Ontology Evolution as a Basis for User-Adaptive Recommender Interfaces

Elmar P. Wach
STI Innsbruck, University of Innsbruck/
Elmar/P/Wach eCommerce Consulting
Hummelsbüttler Hauptstraße 43
22339 Hamburg
+49 172 713 6928

elmar.wach@sti2.at,
wach@elmarpwach.com

ABSTRACT

This research proposes an automated OWL product domain ontology (PDO) evolution (without a human inspection) based on given user feedback and enhancing an existing ontology evolution concept. Its manual activities are eliminated by formulating an adaptation strategy for the conceptual aspects of an automated PDO evolution and establishing a feedback cycle. The adaptation strategy consists of a feedback transformation strategy and a PDO evolution strategy and decides when and how to evolve by evaluating the impact of the evolution on the application. An evolution heuristic and evolution strategies are utilised. The adaptation strategy was validated/ firstly "instantiated" by applying it to a real-world conversational content-based e-commerce recommender system as use case. The evolved PDO is going to be evaluated with an experiment and validated with the use case as well.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design – *Methodologies*. H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *relevance feedback*. H.3.5 [**Information Storage and Retrieval**]: On-line Information Services – *commercial services, web-based services*. I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods – *representations (procedural and rule-based), semantic networks*. I.2.6 [**Artificial Intelligence**]: Learning – *concept learning, knowledge acquisition*. K.4.3 [**Computers and Society**]: Organizational Impacts – *automation*.

General Terms

Management, Measurement, Experimentation, Standardization.

Keywords

Ontology Evolution, Recommender Systems, Self-Adapting Information Systems, Heuristics.

1. INTRODUCTION

Recommender systems in e-commerce applications have become business relevant in filtering the vast information available in e-shops (and the Internet) to present useful recommendations to the user. As the range of products and customer needs and preferences change, it is necessary to adapt the recommendation process. Doing that manually is inefficient and usually very expensive.

Recommenders based on product domain ontologies[1] (PDO) can extract questions about the product characteristics and features to investigate the user preference and eventually recommend products that match the needs of the user. By changing the PDO, such a recommender generates different questions and/ or their order and herewith adapts the recommender interface to the user preference. Hence, an automated adaptation of the recommendation process can be realised by automatically evolving the PDO[2]. The high cost of the manual adaptation of the recommendation process and the underlying PDO can herewith be minimised.

This research proposes an automated OWL PDO evolution (without a human inspection) based on given user feedback[3] and enhancing an existing ontology evolution concept. Its manual activities are eliminated by formulating an adaptation strategy for the conceptual aspects of an automated PDO evolution and establishing a feedback cycle. Automatically evolving the PDO is more efficient and less expensive than manually doing it. The present research tackles an automated process for the first time (to the best knowledge of the author).

Figure 1 depicts the starting basis schematically.

In the data modelling layer the OWL PDO evolution is induced by different kinds of user feedback, i.e. from external and internal data sources. When evolving the PDO, it can be necessary to adapt instance data (i.e. products) as well in order to keep them correctly annotated. Afterwards, the new PDO version including associated instance data is provided to the application layer. There

[1] A product domain ontology (PDO) is defined as the formal, explicit specification of a shared conceptualisation of a product description based on OWL DL; this definition is derived from [6]

[2] Ontology evolution is defined as the timely adaptation of a PDO by preserving its consistency (a PDO is consistent if and only if it preserves the OWL DL constraints); this definition is derived from [7] and [16]

[3] In order to focus this research on developing an automated ontology evolution, the feedback is given

UCERSTI 2, October 23, 2011, Chicago, IL, USA

and in the external data sources, the effect of the PDO evolution is evaluated and again reported to the data modelling layer which concludes the feedback cycle.

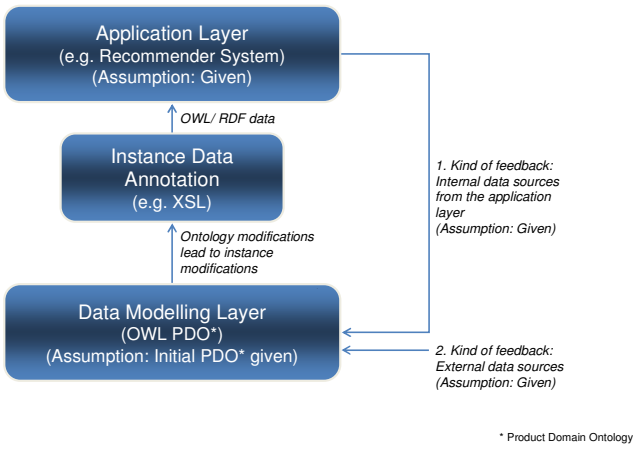

Figure 1. PDO evolution induced by user feedback

The main research question is: How can an automated[4] product domain ontology evolution be realised based on feedback?

2. RELATED WORK

Previous approaches in the topic of this research can be found in concepts for ontology evolution like formulated frameworks for ontology evolution.

[13] focused on the evolution process and have defined six phases consisting of capturing, representation, semantics of change (i.e. a rich description about the semantic role of an ontology entity in order to get more information for solving inconsistencies), implementation, propagation, and validation of ontology changes. This process is implemented in the KAON[5] framework and the Ontologging[6] system. Evolution strategies have been formulated defining elementary and composite changes for executing a change request and eventually deciding the evolution path. [9] focused on detecting ontology changes and have defined five components relating the different change representations to each other. They have proposed a component-based framework for ontology evolution supporting data transformation between two ontology versions, update of remote ontologies, consistent reasoning, verification and approval of ontology changes, and data access to an old ontology via the new one. [14] focused on the user interaction and have provided a usage-based approach implemented in the OntoManager[7] system. The conceptual architecture is based on the MAPE model (Monitor – Analyse – Plan – Execute). The activities of a user are captured in a semantic log and are instances of a user log ontology. The log data is aggregated and visualised helping an ontology manager in adapting the ontology. Eventually, the ontology evolution process guarantees a transfer from one ontology version to another while preserving consistency. [8] focused on handling inconsistency in changing ontologies and have defined a framework consisting of four approaches addressing the consistent ontology evolution, the repairing of inconsistencies, the reasoning with inconsistent ontologies, and multi-version reasoning. For the first three approaches consistency algorithms have been formulated. A consistent ontology evolution is ensured by removing axioms that are structurally connected with the conflicting axioms. [11] focused on collaborative environments and have developed a set of Protégé[8] plugins to support different ontology evolution scenarios. Those include synchronous (i.e. online)/ asynchronous ontology editing, continuous editing/ periodic archiving (i.e. versions), curation (i.e. inspection by a human)/ no curation, and monitored (i.e. record of changes)/ non-monitored ontology changes. The central element is a change and annotation ontology (ChAO) which gathers and provides information about the ontology changes including meta-information like the author and timestamp. [10] introduced a general framework answering the essential questions of what can be changed in an ontology and how each change should be implemented. It is split in five steps comprising the ontology model selection, supported operations, consistency model (i.e. integrity rules), inconsistency resolution, and action selection based on a preference ordering. [18] proposed Evolva, a framework and tool for the whole ontology evolution cycle which decreases user input by making use of background knowledge like lexical databases, online ontologies and unstructured Web documents. It consists of the components information discovery (i.e. extracts content from external data sources manually specified), data validation (i.e. identifies new terms and checks the quality), ontology changes (i.e. integrates the new information to the ontology), evolution validation (i.e. handles conflicts), and evolution management (i.e. manually controlling the evolution (modifying, filtering), records changes and propagates them to dependent ontologies).

Due to the specific challenges of the present research like the automated ontology evolution process, none of the frameworks discussed can be completely used as basis, e.g. all frameworks include a step for the human inspection of the ontology changes before they are executed. The closest work to the research in this paper is [13] – in the six phase evolution process, two steps include manual activities, namely (i) "implementation" in which the implications of an ontology change are presented to the user and have to be approved by her before execution, and (ii) "validation" in which performed changes can get manually validated. The research in this paper aims at eliminating both manual steps in [13] with the adaptation strategy and its implementation. To automate (i), the ontology evolution is conceptualised and implemented as a complete feedback cycle. An insufficient ontology change is indicated by decreased metrics and gets revised according to the evolution strategy chosen. Hence, the ontology changes do not have to get manually approved before execution. To automate (ii), the PDO changes are predefined and application-oriented. Hence, only valid changes are executed, and nobody has to manually validate them.

3. APPROACH AND PROPOSED SOLUTION

The aim of this research is to combine the use of PDO with processing user feedback. The work focuses on how the given

[4] Without human inspection

[5] http://kaon.semanticweb.org

[6] European Commission project IST-2000-28293

[7] German BMBF project SemIPort (08C5939) and European Commission project Ontologging

[8] http://protege.stanford.edu

feedback can lead to a self-improvement of the semantic application by adapting the PDO. In this context self-improvement means that by automatically processing user feedback and evolving the PDO, the defined key performance indicators (KPI) of the application will increase.

The use case is a real-world conversational content-based e-commerce recommender system based on PDO that semantically describe the products offered in e-commerce applications according to GoodRelations[9]. Four types of PDO changes are defined with the following impact on the user dialogue in the recommender system:

- Switching individuals (i.e. properties are related to other individuals within the same class): This leads to a different clustering of the questions
- Switching datatype property ranges (i.e. properties get Boolean ranges instead of string ranges and vice versa (where applicable)): This leads to textual modifications of the questions
- Switching annotation properties label and comment (i.e. properties get different labels and comments extracted from another information source): This leads to textual modifications of the questions (and maybe a need-based sales approach instead of a technology-prone one)
- Changing annotation property priority (i.e. different priority values): This leads to a different ranking of the questions and skips the ones with low priorities

In this paper the PDO change switching individuals is used as an example (confer section 4.2). A digital camera has a feature HDMI. This PDO change defines in which feature-related section the question is nestled whether the camera should offer HDMI.

The success and thus the KPI of an e-commerce recommender are usually defined by the click-out rate (i.e. clicks-to-recommendations) or conversion rate (i.e. customers-to-recommender users). The user gives feedback to the quality of a product recommendation in following the recommendation (i.e. click-out) or even buying the product (i.e. conversion).

In the approach a six step adaptation strategy for the conceptual aspects of an automated PDO evolution has been formulated and a feedback cycle established. The adaptation strategy answers the questions when and how to evolve the PDO by evaluating the impact of the evolution in the precedent feedback cycle. The first question defines the (temporal and causal) trigger initiating the PDO change. Basically, this is receiving and transforming the feedback into ontology input and will be addressed with the feedback transformation strategy. The second question defines the changing of the PDO with annotated instances. This is evolving the PDO and will be addressed with the PDO evolution strategy. Due to space limitations and the focus on realising a user-centric evaluation, the adaptation strategy is not elaborated in this paper. The strategy is used to concisely describe the application for which the automated PDO evolution should be implemented and the impacts of PDO changes on the application behaviour. The interested reader is referred to [17].

[9] www.purl.org/goodrelations

3.1 Evolution Heuristic and Evolution Strategies

The automated ontology evolution is realised by utilising an evolution heuristic and evolution strategies. Those are defined in the fifth step of the adaptation strategy "Decide the adequate PDO evolution". The impact of the PDO change is measured in the Feedback Transformer (confer section 3.2) component by calculating the Success Trend ST for the new user feedback from the application layer and external data sources. The ST is analysed by a heuristic that defines the PDO change to be executed. A heuristic is a strategy that uses accessible and loosely applicable information to solve a problem of a human being or a machine [12] and leads to a solution of a complex problem with simplified conceptual aspects or reduced computation power. [3] mentioned first the term metaheuristic for a computational method that makes few or no assumptions about the problem being optimised and introduced the tabu search metaheuristic [4]. The tabu search enhances a local search (i.e. iteratively improving a criterion in the search space) metaheuristic by using "taboos" – a solution is not executed again according to the criteria defined in the tabu list. The philosophy when utilising a heuristic should be that the highest precedent ST defines the next PDO change to always choose the best evolution. The relevant characteristics of the heuristic have initially to be defined, confer section 4.1. This manual effort is rewarded with a greater conceptual flexibility resulting in an evolution that is more application-oriented. The relevant metrics have to be defined and the calculations formulated.

The PDO evolution is decided based on the ST. In case the feedback includes information extracted from the PDO (e.g. property-based feedback), the subsequent evolution (i.e. type of PDO change) is defined by implementing the ST in the same representation as before (e.g. ontological entity, range), and neither statistical means nor a heuristic has to be applied.

This research proposes to additionally formulate evolution strategies that decide the general evolution behaviour (e.g. executing the same type of PDO change or a rollback) by correlating the types of PDO changes needed to the ST calculated. Additionally, the path for determining the initial ST has to be defined, e.g. the order of the different types of PDO changes and for which PDO they are executed (i.e. ramp-up of the evolution strategies). The philosophy should be that the development (and its strength) of the precedent ST defines the next type of PDO change to distinguish different evolution impacts.

A positive ST means a positive trend (i.e. an increase) of the metrics, a negative the opposite. The larger the figure is, the stronger the development of the metrics (in either direction) from the precedent to the current cycle has been. So, there are two criteria (i.e. ST and its strength) to decide about the next type of PDO change. Basically, there can be two resulting user behaviours in the e-commerce recommender system:

- The user is satisfied with the product recommendation and clicks to see the detail page or order it; in that case the metrics increase, but it still has to be decided if a change should be made
- The user is not satisfied with the product recommendation and leaves the recommender; the metrics decrease, though we do not know why she was not pleased, and a PDO change is advisable

In the first case, one can argue either way – a change is luring to even further increase the metrics. On the other hand, one could keep everything as it is and wait for the next feedback. The latter case is more urging for a change. It has still to be decided if it is a change or just a rollback to retrieve the previous setting. So, it is advisable to define evolution strategies reflecting different behaviours with associated types of PDO changes. In the following, these strategies are predefined and discussed.

Risky Evolution:

An evolution is induced in either case, i.e. a positive or a negative trend. Different types of PDO changes than in the precedent feedback cycle are executed. This behaviour tries to radically improve the metrics by all means and can be described as "always evolve differently". The decision criteria are as follows:

- Increase of the KPI (i.e. $0 \leq ST \leq 1$)
- Decrease of the KPI (i.e. $-1 \leq ST < 0$)

Progressive Evolution:

An evolution depends on the leap in the ST between two consecutive feedback cycles and can be fine-tuned with a threshold defining the trend significance (i.e. the increase of the ST between the precedent and the current cycle). In case of a significant positive trend, the same type of PDO change as in the precedent feedback cycle is executed. In case of a moderately positive trend, a different type of PDO change than in the precedent feedback cycle is executed. In case of a negative trend, it is optional to either do a different type of PDO change than in the precedent feedback cycle or a rollback (to be selected in the administration interface of the Adaptation Manager). This behaviour tries to repeat a significant increase by the same means but gives also the option to revert a negative development. It can be described as "learn from the past". Additionally, the "risk" of the evolution can be adjusted with the threshold. The higher it is the more unlikely the same type of PDO change as in the precedent feedback cycle is executed, and the strategy is tuned towards the Risky Evolution (with a higher threshold). Initially, the threshold is defined to be 20%[10] and can be changed in the administration interface as well. The decision criteria are as follows:

- Significant increase of the KPI (for the beginning, the threshold is defined to be 20%, i.e. $0{,}2 \leq ST \leq 1$)
- Moderate increase of the KPI (i.e. $0 \leq ST < 0{,}2$)
- Decrease of the KPI (i.e. $-1 \leq ST < 0$)

Safe Evolution:

An evolution is induced only by a negative trend. In that case, a rollback is executed. This behaviour tries only to revert a negative development. It can be described as "only revert negative trends". The decision criteria are as follows:

- Increase of the KPI (i.e. $0 \leq ST \leq 1$)
- Decrease of the KPI (i.e. $-1 \leq ST < 0$)

[10] Increase of the ST by 20 basis points between the precedent and the current feedback cycle

Rollback:
This "strategy" reverts the PDO changes from the precedent feedback cycle (i.e. rolling back to the precedent PDO version) and is based on any reason or decision of the manager. It is executed only once but can be manually chosen multiple times. The behaviour can be described as "undo the PDO changes".

The evolution strategies introduced above are considered as basic categories. They can be fine-tuned with regard to the associated types of PDO changes as well as the threshold defining the trend significance. Table 1 sums up the predefined evolution strategies, decision criteria (ST), and the type of PDO changes to be executed in the feedback cycle.

Table 1. Evolution strategy, Success Trend ST, and associated type of PDO change

Evolution Strategy	Decision Criteria	Type of PDO Change
Risky Evolution ("always evolve differently")	$-1 \leq ST \leq 1$	Different than before
Progressive Evolution ("learn from the past")	$0{,}2^* \leq ST \leq 1$ $0 \leq ST < 0{,}2^*$ $-1 \leq ST < 0$	Same as before Different than before Different than before or Rollback
Safe Evolution ("only revert negative trends")	$0 \leq ST \leq 1$ $-1 \leq ST < 0$	None Rollback
Rollback ("undo the PDO changes")	Manually	Rollback

* Increase of the ST by 20 basis points between the precedent and the current feedback cycle

Each evolution strategy besides Rollback ensures an adaptive change of the PDO and thus the recommender interface. By selecting a strategy in the administration interface, the business manager decides how fundamental the evolution will be.

3.2 Implementing the Strategy by Programming an Application

By following the principles of adaptive systems [2], the adaptation strategy is implemented in a new adaptation layer (confer figure 2) consisting of components in which the user feedback gets transformed (i.e. Feedback Transformer) and the respective actions are decided and initiated (i.e. Adaptation Manager). This system creates an evolved PDO with associated instances.

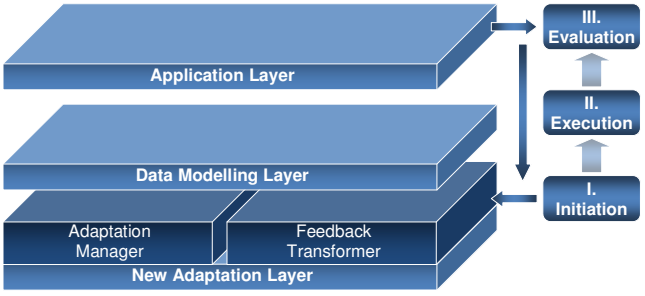

Figure 2. PDO evolution cycle with a new adaptation layer

The whole evolution cycle is based on the generic change process model [1] consisting of three iterative phases and defining four activities:

1. Phase "initiation" – Activities: Requesting the change and analysing/ planning the change
2. Phase "execution" – Activity: Implementing the change
3. Phase "evaluation" – Activity: Verifying/ validating the change

The three layers (i.e. application layer, data modelling layer, and adaptation layer) interact during the three phases of the generic change process model forming the basis of the automated PDO evolution process.

In the first phase "initiation" the different kinds of user feedback are delivered to the adaptation layer and thus a PDO change requested. As the PDO is the backbone of a semantic application, the feedback is assumed to be RDF data. This feedback is converted to ontology input by the Feedback Transformer according to the feedback transformation strategy. The Feedback Transformer accesses the user feedback channels programmatically via SPARQL endpoints and identifies the PDO affected with SPARQL SELECT statements. Eventually, the Feedback Transformer calculates the Success Trends ST for each feedback channel, e.g. by a simple value transformation or by calculating the relative frequencies of the property values in the feedback. Then, the PDO evolution is prepared by identifying the next PDO change with the transformed feedback by the Adaptation Manager. The system has to decide which evolution actions to take according to the PDO evolution heuristic and strategy. The Adaptation Manager analyses the transformed feedback with a tabu search metaheuristic that chooses the PDO change with the highest ST. The tabu criteria are implemented for each type of feedback. Additionally, the predefined evolution strategies (i.e. Risky Evolution, Progressive Evolution, Safe Evolution, Rollback) are implemented and ramped-up. For determining the initial ST, the different types of PDO changes are sequentially executed in an alphabetical order with an exemplary PDO. These values are then valid as starting basis for all PDO. After this phase, the evolution strategy decides whether the (i) same or (ii) another type of PDO change is executed. In (i), a PDO change within the same type of PDO change is executed and ST(t+1) calculated, except a tabu criterion defined by the evolution heuristic is met. In this case, another type of PDO change is executed in contrary to the evolution strategy. In (ii), the type of PDO change and the PDO change to be executed are determined by the evolution heuristic, and ST(t+1) is calculated.

In the second phase "execution" the changes get implemented in the data modelling layer directed by the PDO evolution heuristic and strategy and by retaining a consistent PDO including correctly annotated instance data. In the Adaptation Manager the predefined PDO changes (for the use case they are switching individuals, switching datatype property ranges, switching annotation properties label and comment, changing annotation property priority, confer section 3.) are implemented and thus ensure a consistent ontology evolution. They are executed with SPARQL CONSTRUCT rules or programmatically. Eventually, the versioning is implemented according to the change-based concept and utilising an ontology with annotated logs. The new PDO version with associated instances is provided to the application layer.

The third phase "evaluation" concludes the feedback cycle by measuring the impact of the change. This is done by calculating adequate metrics relating the currently evaluated feedback from the application layer and external data sources reported to the adaptation layer to the precedent feedback.

The process from the feedback type to the resulting type of PDO change is depicted in the activity diagram in figure 3.

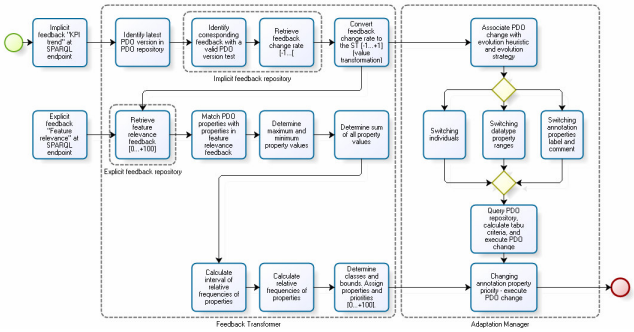

Figure 3. Activity diagram feedback type to type of PDO change

4. EVALUATION AND VALIDATION

The adaptation strategy has been validated/ "instantiated" by applying it to the use case. As this recommender is already used in live applications, it is a real-world scenario. In a conversational approach the actions and modifications done in the adaptation layer mainly lead to a changed user dialogue.

Implicit user feedback is derived from user interactions in the application layer and gathered by unobtrusively monitoring user needs. Explicit user feedback is gathered by extracting information from various websites. Both feedback channels deliver RDF data via separate SPARQL endpoints programmatically accessible.

Applying the adaptation strategy could be done quite smoothly. Only minor aspects of the strategy were clarified, restructured, and reformulated. After having applied the strategy, the use case was concisely described and conceived by the ontology engineer. Moreover, the result formed the basis of the technical specification and thus the development of the adaptation layer.

Due to space limitations the "instantiation" of the adaptation strategy is not completely elaborated in this paper. In the following the evolution heuristic based on tabu search is introduced (excluding its ramp-up).

4.1 Characteristics of the Evolution Heuristic

The evolution heuristic determines the PDO change to be executed. As the evolution strategies define if the same type of PDO change is repeated or another one is executed, the type has still to be determined in the latter case as well as the PDO change (e.g. switching the property weight from the individual WeightAndDimension to the individual GeneralCharacteristics). For this, a tabu search metaheuristic is utilised with the following characteristics: (i) Always the impact of the evolution in the precedent feedback cycle is evaluated, (ii) only one implicit PDO change is executed per cycle, and (iii) "greedy" approach: The

evolution heuristic chooses the PDO change with the highest ST. There are two types of ST for determining the PDO change to be executed: (i) $ST_{f\_pdo\_change\_x}$ is the ST for the forward PDO change x, and (ii) $ST_{b\_pdo\_change\_x}$ is the ST for the backward PDO change x (i.e. reverts the forward change). Forward PDO changes to be executed are determined with the highest $ST_{f\_pdo\_change\_x}$, backward PDO changes with the highest $ST_{b\_pdo\_change\_x}$.

In the following the tabu criteria are defined.

4.1.1 Specific Tabu Criteria sw and ch

The specific tabu criteria are specifically calculated for each type of PDO change.

4.1.1.1 Allowed Number of Horizontal Switches sw

With sw one (set of) ontological entity of a PDO within the same type of PDO change is switched, e.g. a PDO change of one (set of) property or (set of) individual – most of times there is only one switch possible like changing the individual, the property range, or the annotation properties label and comment, and the next change would be reverting that change. This tabu is defined as follows:

$$sw = \begin{cases} 0, \text{ case: } p=1 \wedge c_{fix}=0 \\ 2+c_{fix}^2/2-c_{fix}, \text{ case: } p=1 \wedge c_{fix}=2*k,\ c_{fix}, k \in \mathbb{N}\setminus\{0\} \\ 1+c_{fix}*(c_{fix}-1)/2, \text{ case: } p=1 \wedge c_{fix}=2*k-1,\ k \in \mathbb{N}\setminus\{0\} \\ 1+p^2/2-p, \text{ case: } p>1 \wedge p=2*k,\ p \in \mathbb{N}\setminus\{0,1\},\ k \in \mathbb{N}\setminus\{0\} \\ p*(p-1)/2, \text{ case: } p>1 \wedge p=2*k-1,\ p \in \mathbb{N}\setminus\{0,1\},\ k \in \mathbb{N}\setminus\{0\} \end{cases} \quad (\mathbf{1})$$

(c_{fix} being the number of fixed candidates within a type of PDO change (i.e. to these candidates can be switched), p being the number of pools of sets of entities (e.g. each source for the properties is a pool like string ranges, Boolean ranges, DBpedia, or WordNet; p can be changed for each type of PDO change in the administration interface); a pool p can be switched on the level of ontological entity (s') or completely (s), i.e. all sets of ontological entities are switched at once (can be changed for each type of PDO change in the administration interface, in case of more than one data pool p), k being a natural number to indicate an even ($c_{fix} = 2 * k$, $p = 2 * k$) or odd ($c_{fix} = 2 * k - 1$, $p = 2 * k - 1$) number of fixed candidates or pools: The case for the even c_{fix} or p equates to an Eulerian trail, the case for the odd c_{fix} or p to an Eulerian circuit).

Result is the number of allowed switches sw. In case s is already connected to c_{fix} (e.g. s - c_{fix} = 1), the second and third case in (**1**) are lessen by this one "impossible" switch (i.e. sw_{fix} = sw - 1).

4.1.1.2 Allowed Number of Vertical PDO Change Iterations ch

With ch successive sw switches within the same type of PDO change are executed, i.e. the next (sets of) ontological entities are going to be switched. This tabu is defined as follows:

$$ch = \begin{cases} (s-ch_{fix})/n; \text{ case: } p=1,\ n \in \mathbb{N}\setminus\{0\},\ s,\ ch_{fix} \in \mathbb{N},\ s \geq ch_{fix} \\ s'/n, \text{ case: } p>1 \wedge s' \subset s \text{ (i.e. single sets)},\ n \in \mathbb{N}\setminus\{0\},\ s' \in \mathbb{N} \\ \text{Not applicable, case: } p>1 \wedge s' \equiv s \text{ (i.e. all sets at once)} \end{cases} \quad (\mathbf{2})$$

ch is truncated to the natural number.

(s being all sets of ontological entities within a type of PDO change (e.g. all sets of individuals, all sets of properties, all sets of annotation properties label and comment), s' being a single set of ontological entities within a type of PDO change (e.g. specific properties) to be switched to another pool, n being the fraction of the "free" sets (i.e. not connected to a c_{fix}) of entities within a type of PDO change allowed to be switched (e.g. n = 1: All free sets of entities, n = 2: Half of the free sets, etc.; n can be changed for each type of PDO change in the administration interface)).

Result is the number of allowed PDO change iterations ch. Analogous to the case distinction of the horizontal switches sw and sw_{fix}, ch is splitted in the first case in (**2**) into s is not connected to c_{fix} before switching (ch), and s is already connected to c_{fix} before switching (ch_{fix}).

4.1.2 General Tabu Criterion gt

To avoid an uniform optimisation and cycles, the PDO changes within the same type of PDO change are consecutively executed only as often as there are different types T of PDO changes not induced by a feedback based on a PDO extraction (here: Three times, T = 3, i.e. switching individuals, switching datatype property ranges, and switching annotation properties label and comment).

In case a type of PDO change has less than T PDO changes, the general tabu criterion gt is met when all PDO changes within the respective type of PDO change have been executed.

To calculate the general tabu criterion gt, the overall number of switches sw and ch executed has to be respected. Hence, this tabu is valid when having executed either all sw and ch switches within

the respective type of PDO change (case: Number of all switches ≤ T) or the number of switches executed within the same type of PDO change equals T (here: $T = 3$) (case: Number of all switches > T); this tabu is defined as follows:

$$gt = \begin{cases} sw*ch+(sw-1)*ch_{fix} \leq T, \text{ case: } p=1, sw, ch, ch_{fix}, T \in \mathbb{N} \\ sw*ch \leq T, \text{ case: } p>1 \wedge s' \subset s \text{ (i.e. single sets)}, sw, ch, T \in \mathbb{N} \quad (\mathbf{3}) \\ sw \leq T, \text{ case: } p>1 \wedge s' \equiv s \text{ (i.e. all sets at once)}, sw, T \in \mathbb{N} \end{cases}$$

Result is the number of allowed PDO changes gt. The PDO changes are sequentially executed and added to the tabu list. In case the tabu gt or T is met, another type of PDO change is going to be executed.

In case another type of PDO change is executed, the overall oldest tabu is deleted from the tabu list.

After the ramp-up and in case the general tabu criterion gt or T is met (here: The same type of PDO change shall be consecutively executed for the fourth time), the PDO change with the highest ST in another type of PDO change is going to be executed and ST(t+1) calculated.

In case the "allowed number of horizontal switches" sw is met, the PDO change with the second highest ST within the same type of PDO change is executed and ST(t+1) calculated.

4.2 Example Calculation of the Tabu Criteria

The tabu criteria are exemplarily calculated for the type of PDO change switching individuals. It has one data pool ($p = 1$, i.e. one set of individuals); p is manually entered in the Administration Interface. A digital camera has the following sets of properties and individuals {s, I}: {faceDetection, Features}, {weight, WeightAndDimension}, {videofunction, GeneralCharacteristics}, {HDMI, Ports}, {opticalZoomFactor, LensFeatures}, and {touchscreen, Display}. So, the question if the camera should offer HDMI is nestled between the port-related features of the camera. By observing the relationships, it is obvious that not all combinations make sense, e.g. HDMI cannot belong to WeightAndDimension, but it could belong to Features or GeneralCharacteristics. When switching the HDMI property to another individual, e.g. from Ports to GeneralCharacteristics, the question after HDMI could be placed aside the question for the video function which could make more sense from a customer point of view. The Feedback Transformer identifies the general individuals (i.e. c_{fix}) by parsing the strings. In the example the two individuals mentioned above are of general meaning, i.e. $c_{fix} = 2$.

- Specific tabu criterion "allowed number of horizontal switches" sw:

  (**1**), second case, with $c_{fix} = 2$:

  $sw = 2$ (case: s is not connected to c_{fix} before switching) and $sw_{fix} = 1$ (case: s is already connected to c_{fix} before switching)

  Result: The specific tabu criterion sw is met with two switches or one switch; in this case, the next set of individuals is going to be switched.

- Specific tabu criterion "allowed number of vertical PDO change iterations" ch:

  (**2**), first case, with $c_{fix} = 2$, $n = 2$ (i.e. half of the "free" sets; "free" meaning not connected to c_{fix} before switching), $s = 6$ (i.e. properties):

  $ch = 2$ (**3**)

  Result: The specific tabu criterion ch is met with switching two sets of individuals allowed to be switched.

- General tabu criterion gt:

  (**3**), first case:

  $gt = 6 \leq T$

  Result: The general tabu criterion gt is met with switching the minimum of six sets of individuals to c_{fix} and T; as $T = 3$ (i.e. three types of PDO changes not induced by a feedback based on a PDO extraction), the tabu is met with three individual switches; in this case, another type of PDO change is going to be executed

This means in case of a high ST for the switch of HDMI from Ports to GeneralCharacteristics, this switch will be within the first three individual switches and get executed. In case it is not, the question for HDMI will remain aside the port-related questions.

4.3 Future Work: Evaluation and Validation of the Adaptation Layer

The adaptation layer is going to be evaluated by conducting an experiment with approximately thirty ontology experts who evaluate the ontology evolution. The automatically evolved PDO is going to be compared with a manually evolved one by setting up and evaluating an experiment with ontology experts who analyse the feedback delivered and decide the PDO changes to be executed. Eventually, the PDO resulted from this manual evolution is compared with the automatically evolved one regarding the evaluation criteria consistency, completeness, conciseness, expandability, and sensitiveness [5].

The adaptation layer is going to be validated by programming the layer and measuring the effects in the e-commerce recommender system. Its success is defined by the click-out rate (i.e. clicks-to-recommendations; the user follows the recommendation by clicking on the product recommended) which measures the impact of the PDO evolution induced by the implicit and explicit user feedback.

The validation scenario will be to analyse and evaluate the impact of the PDO evolution with regard to the respective KPI reported to the adaptation layer after having accomplished the defined number of recommendation processes by utilising the formulated evolution strategies, i.e. Risky, Progressive, and Safe Evolution. In each feedback cycle the transformed feedback (i.e. ST) gets reported to the Adaptation Manager. The feedback is PDO-based

or PDO- and property-based. According to the feedback reported, the PDO evolves. The new PDO version is provided to the data modelling layer and the application layer, and eventually an adapted recommender interface is presented to the customer. The feedback circle of the automated system concludes with re-evaluating the KPI after having again accomplished the defined number of recommendation processes.

The intended results are a highly user-adaptive system and eventually better recommendations given to the customer leading to an increase of the defined KPI. The expected business impacts are a higher customer satisfaction and loyalty and eventually increased revenue for the provider of the e-commerce application (and the recommender system).

5. CONCLUSION

The need for automatically updating and evolving ontologies is urging in today's usage scenarios. Here, it is the basis for creating a user-adaptive recommender interface. The present research tackles an automated process for the first time (to the best knowledge of the author). The reason for that can be found in the ontology definition "formal, explicit specification of a shared conceptualisation" [6]. "Shared" means the knowledge contained in an ontology is consensual, i.e. it has been accepted by a group of people [15]. Entailed from that, one can argue that by processing feedback in an ontology and evolving it, it is no longer a shared conceptualisation but an application-specific data model. On the other hand, it is still shared by the group of people who are using the application. It may even be argued that the ontology has been optimised for the usage of that group (in a specific context or application) and thus is a new way of interpreting ontologies: They can also be a specifically tailored and usage-based knowledge representation derived from an initial ontology – an ontology view, preserving most of the advantages like the support of automatically processing information. Thus, this changed way of conceiving ontologies could facilitate the adoption and spread of using this powerful representation mechanism in the real world, as it is easier to accomplish consensus within a smaller group of people than a larger one.

In this research the PDO are based on GoodRelations and evolve within that upper ontology. This ontology as well as the "subsumed" PDO conforms to the ontology definition by [6]. The PDO are application-specific and evolve according to the needs of their users. Hence, they offer the advantages of both worlds.

In the next steps of this research the adaptation layer is going to be evaluated and validated.

6. ACKNOWLEDGMENTS

The research presented in this paper is funded by the Austrian Research Promotion Agency (FFG) and the Federal Ministry of Transport, Innovation, and Technology (BMVIT) under the FIT-IT "Semantic Systems" program (contract number *825061*).

7. REFERENCES

[1] Bennett, K. H. and Rajlich, V. T. 2000. Software maintenance and evolution: A roadmap, *Proceedings of the Conference on the Future of Software Engineering*, pp. 73-87.

[2] Broy, M. et al. 2009. Formalizing the notion of adaptive system behavior, *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09)*, pp. 1029-1033.

[3] Glover, F. W. 1986. Future paths for integer programming and links to artificial intelligence, *Comput. Oper. Res.*, Volume 13, pp. 533-549.

[4] Glover, F. W. and Laguna, M. 1997. *Tabu Search*, Kluwer Academic Publishers.

[5] Gómez-Pérez, A. 2001. Evaluation of ontologies, *International Journal of Intelligent Systems*, Volume 16, pp. 391-409.

[6] Gruber, T. R. 1993. Toward principles for the design of ontologies used for knowledge sharing, *Formal ontology in conceptual analysis and knowledge representation*, Kluwer Academic Publishers.

[7] Haase, P. and Stojanovic, L. 2005. Consistent evolution of OWL ontologies, *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005)*, pp. 182 - 197.

[8] Haase, P. et al. 2005. A framework for handling inconsistency in changing ontologies, *Proceedings of the 2005 International Semantic Web Conference (ISWC05)*, pp. 353-367.

[9] Klein, M. and Noy N. F. 2003. A component-based framework for ontology evolution, *Proceedings of the IJCAI-03 Workshop on Ontologies and Distributed Systems*.

[10] Konstantinidis, G. et al. 2007. Ontology evolution: A framework and its application to RDF, *Proceedings of the Joint ODBIS & SWDB Workshop on Semantic Web, Ontologies, Databases*.

[11] Noy, N. F. et al. 2006. A framework for ontology evolution in collaborative environments, Proceedings of the 2005 International Semantic Web Conference (ISWC05), pp. 544-558.

[12] Pearl, J. 1983. Heuristics: Intelligent search strategies for computer problem solving, Addison-Wesley.

[13] Stojanovic, L. et al. 2002. User-driven ontology evolution management, *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW '02)*, pp. 285-300.

[14] Stojanovic, N. et al. 2003. The OntoManager – a system for the usage-based ontology management, *LNCS 2888*, pp. 858-875.

[15] Studer, R. et al. 1998. Knowledge engineering: Principles and methods, *Data & Knowledge Engineering*, Volume 25, Number 1-2, pp. 161-198.

[16] Suárez-Figueroa, M. C. and Gómez-Pérez, A. 2008. Towards a glossary of activities in the ontology engineering field, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC '08)*.

[17] Wach, E. P., 2011. Automated ontology evolution for an e-commerce recommender, *Proceedings of the 14th International Conference on Business Information Systems (BIS 2011)*, in press.

[18] Zablith, F. 2009. Evolva: A comprehensive approach to ontology evolution, *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*.