Automated Ontology Evolution for an E-Commerce Recommender*

Elmar P. Wach[1,2]

[1] STI Innsbruck, University of Innsbruck,
Technikerstraße 21a, 6020 Innsbruck, Austria
elmar.wach@sti2.at
[2] Elmar/P/Wach eCommerce Consulting,
Hummelsbüttler Hauptstraße 43,
22339 Hamburg, Germany
wach@elmarpwach.com

Abstract. This research proposes a completely automated OWL product domain ontology (PDO) evolution by 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. This strategy decides when and how to evolve by evaluating the impact of the evolution in the precedent feedback cycle and is implemented in a new adaptation layer. The adaptation strategy was validated/ firstly "instantiated" by applying it to a real-world conversational content-based e-commerce recommender as use case.

Keywords: Ontology Evolution, Ontology Versioning, 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 product 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) modelling the products offered in the e-commerce application 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. Hence, an automated adaptation of the recommendation

* 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*).

[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 [Gruber, T. R. 1993].

W. Abramowicz, L. Maciaszek, K. Węcel (Eds.): BIS 2011 Workshops, LNBIP 97, pp. 166–177, 2011.

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 a completely automated OWL PDO evolution (without a human inspection) based on given user feedbacks[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).

2 Related Work

Previous approaches to the topic of this research can be found in concepts for ontology evolution like formulated frameworks for ontology evolution, e.g. [Haase, P. et al. 2005], [Klein, M. and Noy N. F. 2003], [Konstantinidis, G. et al. 2007], [Noy, N. F. et al. 2006], [Stojanovic, L. et al. 2002], [Stojanovic, N. et al. 2003], [Zablith, F. 2009]. Due to the specific challenges of the present research like the automated ontology evolution process, none of the identified frameworks can be completely used as basis, e.g. all of the 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 [Stojanovic, L. et al. 2002] – 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. This research aims at eliminating both manual steps in [Stojanovic, L. et al. 2002] with the adaptation strategy and its implementation. To automate (i), the ontology evolution is conceptualised and implemented as a complete feedback cycle [Bennett, K. H. and Rajlich, V. T. 2000]. 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. This approach is addressed with the adaptation strategy and its implementation as a new adaptation layer consisting of two components [Broy, M. et al. 2009].

3 Adaptation Strategy

The adaptation strategy addresses when a change has to be executed and how the changes will be executed in the PDO by evaluating the impact of the evolution in the

[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 [Haase, P. and Stojanovic, L. 2005] and [Suárez-Figueroa, M. C. and Gómez-Pérez, A. 2008].

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

precedent feedback cycle. The first question defines the (temporal and causal) trigger initiating the PDO change. This is addressed with the feedback transformation strategy (confer section 3.1) which is implemented in the Feedback Transformer.

The second question defines the changing of the PDO with annotated instances (i.e. products). This is evolving the PDO and will be addressed with the PDO evolution strategy (confer section 3.2) which is implemented in the Adaptation Manager.

By following the principles of adaptive systems [Broy, M. et al. 2009], the strategies are implemented in a new adaptation layer (confer figure 1) 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).

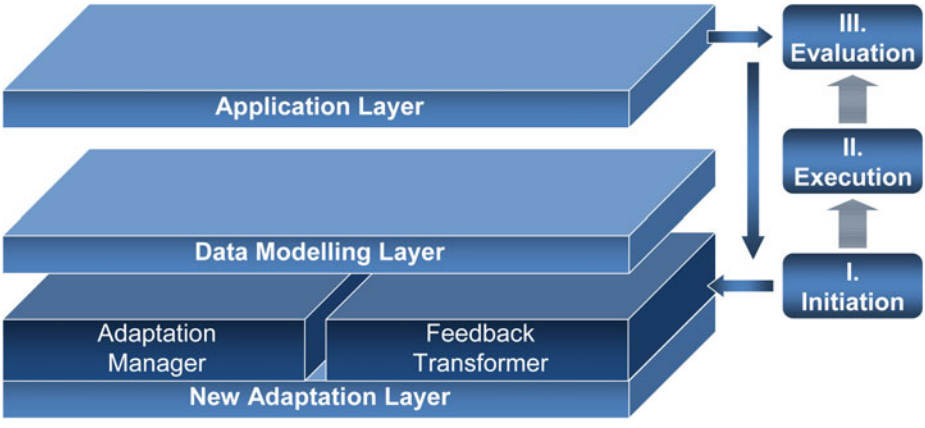

Fig. 1. Evolution cycle with a new adaptation layer

3.1 Feedback Transformation Strategy

The feedback transformation strategy defines when the PDO change. It transforms different kinds of user feedback (e.g. implicit, explicit) into ontology input (i.e. calculating Success Trends ST). This strategy is implemented in the Feedback Transformer where the user feedback channels and the PDO affected by the feedback reported are identified, the feedback is analysed and gathered, and eventually transformed.

The strategy comprises the following steps:

1. Identify the user feedback channels
2. Analyse and gather the user feedback
3. Transform the user feedback

Ad 1. Identify the user feedback channels

In this step the application setup is analysed with regard to the available user feedback. In order to focus this research on developing an automated PDO evolution, the feedback is assumed to be given, and thus extracting the information is out of scope. The application can provide two kinds of user feedback to get a complete view

of the user: Internal data sources from the application layer like the KPI[4] or statistical evaluations of the usage. As the application is based on PDO, PDO changes influence the application behaviour, and KPI and statistical evaluations of the usage of the application layer are a valid feedback for the impact of the PDO evolution.

The application setup can also provide external data sources like data and information extractions from the Web, databases, or ontologies. E.g., discussions in Blogs and portals, official and unofficial product and product feature ratings, and appearances of new features and product aspects are valuable PDO information.

The two kinds of user feedback are delivered via different feedback channels that have to be identified and analysed with regard to the feedback representation, its accessibility, and the PDO affected. As the PDO is the backbone of a semantic application, the feedback is assumed to be RDF data. In case it is not, it is recommended to convert it to RDF[5]. A crucial aspect is the accessibility of the user feedback – can it be programmatically retrieved by the Feedback Transformer, e.g. via an API or from a SPARQL endpoint?

Ad 2. Analyse and gather the user feedback

In this step the user feedback channels and the feedback delivered are analysed with regard to the feedback content, structure, and meaning. In case the feedback is in RDF, e.g. it can be dynamically queried with SPARQL SELECT statements.[6] In order to adequately interpret the feedback, the metrics delivered have to be identified as well as their meaning have to be clear. Generally, there can be two types of feedback: Explicit user feedback could be provided by answering questions about the user satisfaction with the application. As this effort cannot be expected from a user, an alternative is to extract feedback from the Web that could also deliver new information and aspects about the products offered. Implicit user feedback is given by the user as a side-effect of the usage behaviour, e.g. by clicking on the product recommended.

Currently, two feedback channels with two types of feedbacks are defined:

- Implicit feedback channel (user feedback derived from user interactions in the application layer) "KPI trend": The implicit feedback mainly evaluates the success/ usage/ usability of the e-commerce recommender; it is PDO-based
- Explicit feedback channel (user feedback extracted from the Web) "Feature relevance": The explicit feedback gathers information about products based on PDO extractions and is represented as an annotation property; it is PDO- and property-based.

The RDF of the KPI trend feedback includes a value indicating a positive or negative trend of the defined KPI between two PDO versions (i.e. relating the currently evaluated feedback to the precedent one based on the previous changes). It is defined as KPI(t) with the range [-1…∞]. In rare cases, the value can calculatory be larger than +1. This feedback repository is queried with SPARQL SELECT statements to retrieve the KPI for each latest PDO version that represent a valid PDO version test.

[4] Key Performance Indicator, e.g. click-out rate (i.e. clicks-to-recommendations ratio).
[5] Generating the RDF data is out of scope.
[6] Due to space limitations, SPARQL statements are omitted.

The RDF of the feature relevance feedback includes the property (i.e. feature name) and its relevance, e.g. based on the count of appearances in the Web over a period of time. It is defined as Feat(t) with the range [0…+100]. After having retrieved feedback from the first feedback channel, this feedback repository is queried with SPARQL SELECT statements to retrieve the relevance for each latest PDO version to be changed.

Ad 3. Transform the user feedback

In this step the different types of feedback are transformed to ontology input, and thus a PDO change is requested. The impact of the change is measured by calculating adequate metrics for the new user feedback from the application layer and external data sources reported to the adaptation layer and for each feedback channel, defined as Success Trends ST_{ch}, e.g. with an algorithm, formula, or transformation. In case the feedback includes information extracted from the PDO, the transformed feedback has to be in the same representation as before (e.g. ontological entity, range).

In case several feedback channels deliver analogue feedbacks, the respective channels have to be weighted separately. The channel weight is a factor that expresses the relative importance of either feedback channel for the PDO evolution. It can be changed between two feedback cycles, though it is recommended to observe the quality of the feedbacks over time before tuning it. The weights for the corresponding analogue feedback channels sum up to 100%. Additionally, the ST calculation can respect the certainty of a feedback channel. The certainty expresses the probability of the correctness of the reported feedback as a percentage value.

The KPI trend KPI(t) is converted by a simple value transformation to the ST with the range [-1…+1] relating the currently transformed feedback to the precedent one. In the rare case of a KPI value larger than +1, it will be normalised to +1.

The feature relevance Feat(t) is converted by calculating the new relevance of the properties with the relative frequencies of the properties in the feature relevance feedback. The ST with the range [0…+100] is calculated by determining classes correlated to that range and based on the interval of the relative frequencies of the properties. To the classes the corresponding properties (i.e. the relative frequency of the property in the feature relevance feedback is within the bounds of the respective class) as well as the respective relevance are assigned. The new relevance is represented as before (i.e. as an annotation property).

After having transformed the different feedback types, the calculated ST are reported to the next component, i.e. the Adaptation Manager.

3.2 PDO Evolution Strategy

The PDO evolution strategy defines how the PDO change. It associates an evolution action to the ST and ensures a consistent new PDO version. This strategy is implemented in the Adaptation Manager where the structure of the respective PDO gets queried with SPARQL SELECT statements and the PDO changes are executed with SPARQL CONSTRUCT rules or programmatically according to an evolution heuristic and predefined evolution strategies. Alternatively, a statistical analysis of the user feedback and its history can be conducted.

The strategy comprises the following steps:

4. Define the representation of PDO changes
5. Define the analysis of the transformed feedback
6. Ensure a consistent ontology evolution and versioning.

Ad 4. Define the representation of PDO changes

In this step options for the representation of PDO changes are defined, e.g. reusing an existing representation. The change representation defines the possible and allowed PDO changes.

For deciding whether an existing representation of ontology changes should be reused, adequate evolution criteria have to be defined. An existing representation has to be investigated with regard to the PDO representation language (e.g. OWL 1, OWL 2) and the PDO changes (e.g. switching a specific individual, switching the range of a specific property) offered – they have to constitute the types of PDO changes[7] needed by the application and to be executed and evaluated in the next feedback cycle, i.e. PDO evolution cycle. In case the application utilises a specific PDO representation, this is the preferred basis for the representation of its changes as well – in this research the PDO are based on GoodRelations[8]. In case the necessary evolution criteria are not met by an existing or application-oriented representation, a customised one has to be developed, e.g. a specific ontology of changes.

As the PDO model the knowledge queried by the user, it is helpful to describe probable user scenarios to predefine the types of PDO changes needed.

Ad 5. Define the analysis of the transformed feedback

In this step options for the analysis of the transformed feedback are defined, e.g. statistical means or utilising a heuristic, and the adequate PDO evolution is decided. The impact of the PDO change is measured in the Feedback Transformer by calculating the ST for the new user feedback from the application layer and external data sources reported to the adaptation layer and for each feedback channel. The ST can be analysed by statistical means. The method as well as the relevant metrics has to be defined and the calculations formulated. By programmatically calculating the relevant metrics, a complete automation of the analysis as well as the derived evolution actions can be achieved.

Another option is to formulate and utilise 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 [Pearl, J. 1983] and leads to a solution of a complex problem with simplified conceptual aspects or reduced computation power. [Glover, F. W. 1986] 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 [Glover, F. W. and Laguna, M. 1997] which is utilised in this research with the philosophy that the highest precedent ST ("greedy") defines the next PDO change to always choose the best evolution.

[7] Currently defined are switching individuals, switching datatype property ranges, switching annotation properties label and comment, and changing annotation property priority.

[8] `www.purl.org/goodrelations`

The relevant characteristics of the heuristic have initially to be defined (confer section 4.). This manual effort is rewarded with a greater conceptual flexibility resulting in a more specific application-oriented evolution behaviour with regard to its impact on the application. The relevant metrics have to be defined and the calculations formulated.

Regardless of the analysis method chosen, the PDO evolution is decided based on the ST. In case the feedback includes information extracted from the PDO, 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.

In case a heuristic is chosen, 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.

The predefined evolution strategies summarised in table 1 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. 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 ontology changes”)	Manually	Rollback

* Threshold trend significance: 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. By selecting a strategy in the administration interface, the business manager decides how fundamental the evolution will be.

Ad 6. Ensure a consistent ontology evolution and versioning

After having chosen the PDO changes to be executed, the PDO has to evolve depending on rules and by retaining its consistency to eventually provide its knowledge to the application layer. This is done by executing SPARQL CONSTRUCT rules or programmatically. Due to space limitations, the rules are omitted.

When evolving the PDO, it has to be clear how the PDO has been evolved over time, i.e. the different PDO evolutions have to be versioned. By versioning a PDO, its changes get documented, and the historical path of evolution gets traceable. In the context of this research this is of paramount importance for deciding the next PDO change to be executed and reverting the changes executed in the precedent feedback cycle, i.e. a rollback.

The preferred concept of ontology versioning is change-based versioning (i.e. each state gets its own version number and additionally stores information about the changes made), because it facilitates change detection, integration, conflict management [Mädche, A. et al. 2003], and it allows the interpretation how PDO changes influence the metrics. A change-based versioning can be best realised by tracking the PDO changes in a semantic log [Mädche, A. et al. 2002].

4 Evaluation and Validation

The adaptation strategy has been validated/ "instantiated" by applying it to the use case which 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. 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. Four types of PDO changes are defined, i.e. switching individuals, switching datatype property ranges, switching annotation properties label and comment, and changing annotation property priority.

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 in extracts (excluding its ramp-up, for instance). The "taboos" are defined as follows:

– **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
  - Exception: In case a type of PDO change has less than T PDO changes, the general tabu criterion is met when all PDO changes within the respective type of PDO change have been executed
  - The general tabu criterion gt is calculated by multiplying the two specific tabu criteria defined below; result is the number of allowed PDO changes gt; the PDO changes are sequentially executed and added to the tabu list

- After the ramp-up phase and in case the general tabu criterion gt or T is met, the PDO change with the highest ST in another type of PDO change is going to be executed and ST(t+1) calculated.

– **Specific tabu criteria (specifically calculated for each type of PDO change):**

- "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:

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

(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). In case sw is met, the PDO change with the second highest ST within the same type of PDO change is going to be executed and ST(t+1) calculated.

- "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:

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

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 sets of entities within a type of PDO change allowed to be switched (e.g. n = 1: All sets of entities, n = 2: Half of the 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}).

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

In addition to this validation, the adaptation strategy is going to be evaluated by conducting an experiment with approximately thirty ontology experts who analyse and formulate ontology evolution characteristics. These are then aligned with the adaptation strategy and adopted accordingly where applicable.

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 feedbacks 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 [Gómez-Pérez, A. 2001].

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) that measures the impact of the PDO evolution induced by the implicit and explicit user feedback.

The intended results are a highly 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. 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" [Gruber, T. R. 1993]. "Shared" means the knowledge contained in an ontology is consensual, i.e. it has been accepted by a group of people. 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 [Gruber, T. R. 1993]. The PDO are application-specific and evolve according to the needs of their users. Hence, they offer the advantages of both worlds.

References

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

[Broy, M. et al. 2009] Formalizing the notion of adaptive system behavior. In: Proceedings of the 2009 ACM Symposium on Applied Computing (SAC 2009), pp. 1029–1033 (2009)

[Glover, F. 1986] Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. 13, 533–549

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

[Gómez-Pérez, A. 2001] Evaluation of ontologies. International Journal of Intelligent Systems 16, 391–409

[Gruber, T. R. 1993] Toward principles for the design of ontologies used for knowledge sharing. In: Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers

[Haase, P. and Stojanovic, L. 2005] Haase, P., Stojanovic, L.: Consistent evolution of OWL ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 182–197. Springer, Heidelberg (2005)

[Haase, P. et al. 2005] Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y.: A framework for handling inconsistency in changing ontologies. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 353–367. Springer, Heidelberg (2005)

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

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

[Mädche, A. et al. 2002] Managing multiple ontologies and ontology evolution in Ontologging. In: Proceedings of the IFIP 17th World Computer Congress – TC12 Stream on Intelligent Information Processing, pp. 51–63

[Mädche, A. et al. 2003] Managing multiple and distributed Ontologies on the Semantic Web. The VLDB Journal – The International Journal on Very Large Data Bases 12(4), 286–302

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

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

[Stojanovic, L. et al. 2002] Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 285–300. Springer, Heidelberg (2002)

[Stojanovic, N. et al. 2003] Stojanovic, L., Stojanovic, N., González, J., Studer, R.: OntoManager – A system for the usage-based ontology management. In: Chung, S., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 858–875. Springer, Heidelberg (2003)

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

[Zablith, F. 2009] Zablith, F.: Evolva: A comprehensive approach to ontology evolution. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 944–948. Springer, Heidelberg (2009)