Explanation-Aware Software Design of the Semantic Search Engine *KOIOS*

Björn Forcher[1], Thomas Roth-Berghofer[1,2], Michael Sintek[2], and Andreas Dengel[1,2]

[1] Knowledge Management Department, German Research Center for Artificial Intelligence (DFKI) GmbH Trippstadter Straße 122, 67663 Kaiserslautern, Germany

[2] Knowledge-Based Systems Group, Department of Computer Science, University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern

{first.lastname}@dfki.de

Abstract. The provision of explanations in knowledge based systems has a long tradition in computer science. Regarding expert systems, several interesting contributions were proposed, but an overall method for explanation generation is missing. Furthermore, various approaches have never been realised or have never been followed up with respect to current technologies. Explanation-Aware Software Design (EASD) aims at making software systems smarter in interactions with their users. The long-term goal is to develop methods and tools for engineering and improving such capabilities. In this paper we present the idea of EASD including an abstract model for explanation generation. We describe in detail the realisation of that approach for the semantic search engine *KOIOS* using Semantic Web technology.[3]

Key words: explanation-awareness, explanation generation, semantic search

1 Introduction

Software systems need the ability to explain 'reasoning processes' and their results as those abilities can substantially affect their usability and acceptance. Explanation-aware Software Design (EASD) aims at making software systems smarter in interactions with their users [1]. The long-term goal is to develop methods and tools for engineering and improving such capabilities.

KOIOS[4] is a semantic search engine that enables keyword-based search on graph-shaped RDF data. It first computes a set of relevant SPARQL queries allowing users to select the most appropriate ones. Subsequently, a selected query

[3] This research work was supported in part by the research program THESEUS in the MEDICO project, funded by the German Federal Ministry of Economics and Technology (01MQ07016). Responsibility for this publication lies with the authors.

[4] http://projects.dfki.uni-kl.de/KoiosWebUI/

is used to search in a respective triple store. Since SPARQL queries are sometimes hard to read, it is necessary to prepare the queries in a more user-friendly way. In addition, it is helpful to know how the keywords relate to the SPARQL queries. System developers, for instance, may want to test the search engine. In this context, explanations are interesting when the system presents unexpected results. It may turn out that the implementation of the search algorithm is incorrect. Hence, explanations can help to correct a system or improve it. Non-expert users are not interested in the exact implementation of the search algorithm. Instead, they just need simple support to choose the most useful query.

For addressing these issues, we developed and integrated an explanation facility into *KOIOS*. The facility is used to justify computed search results by revealing a connection between keywords and queries. The justifications are depicted as semantic networks in an attempt to make search results more plausible for users. The facility is based on the concept of EASD including a complex model for explanation generation. That means that the provision of explanations is an integral part of the development of *KOIOS* the mentioned user groups and their explanation needs.

The paper is structured as follows. The next section gives a brief overview of relevant research on explanation generation. Sections 3 and 4 presents the concept of EASD and our current model for computing explanations. Section 5 describes the semantic search engine *KOIOS* and motivates the need for explanations. Finally, we illustrate how we realised the explanation component of *KOIOS* regarding EASD. We conclude the paper with a summary and outlook.

2 Related Work

Explanation facilities were important components of Expert Systems (ES) supporting the user's needs and decisions. In those early systems, explanations were often nothing more than (badly) paraphrased rules that lacked important aspects, or too much information was given at once [2].

Wick and Thompson [3] implemented the Reconstructive Explainer (REX) that implements the concept of *reconstructive explanations* for expert systems. REX transforms a trace, *i. e.*, a line of reasoning, into a plausible explanation story, *i. e.*, a line of explanation. The transformation is an active, complex problem-solving process using additional domain knowledge. The degree of coupling between the trace and the explanation is controlled by a filter that can be set to one of four states regulating the transparency of the filter. The more information of the trace is let through the filter, the more closely the line of explanation follows the line of reasoning. We took up the theme of (re-)constructing explanations in our current work.

In [4], Ducassé and Noyé describe a stratified model for user-oriented program analysis including explanations. It contains three steps to explain solutions of logic programs, namely *Extraction*, *Analysis*, and *Visualisations*. The model uses several sources of input such as source codes, execution information and other specifications which are integrated by the extraction step. The result of the

extraction is the trace that is analysed and abstracted in the following step. The resulting data of the second step is presented to the user. Ishizaka and Lusti [5] extend that model by defining a trace model as first step.

The Semantic Web community also addresses the issue of explainability. The Inference Web effort [6] realises an explanation infrastructure for complex Semantic Web applications. Inference Web includes the *Proof Markup Language* (PML) for capturing explanation information. PML can be used as interlingua and offers constructs to represent where information came from (provenance) or how it was manipulated (justifications). The Inference Web comprises different tools and services in order to manipulate and edit PML models including various interaction modes such as exploration or filtered explanation views. We also consider different explanation views in order to provide useful kinds of interaction for realising understandable explanatory dialogues. However, the current work is concerned with explaining system processes. As PML is especially designed for representing justifications of reasoning results it is not part of the explanation facility of *KOIOS*.

3 Explanation-Aware Software Design

In this section we present our current research about *Explanation-Aware Software Design* (EASD). The concept comprises two key aspects regarding development and runtime of the system. In the following, we will elaborate on some general aspects regarding integrating explanations in knowledge-based systems. Subsequently, we present an abstract architecture and a multi-layered explanation model for corresponding facilities.

The first key aspect aims at integrating explanation capabilities into the entire system development process. Swartout and Moore [7] formulated the five desiderata *fidelity*, *understandability*, *sufficiency*, *low construction overhead*, and *efficiency*. Explanations respect fidelity if they build on the same knowledge the system uses for reasoning. Understandability comprises such factors as *user sensitivity* and *feedback*. User sensitivity addresses the user's goals and preferences but also her knowledge of the system and respective domain. Feedback is important as users do not necessarily understand a given explanation. The system should offer certain kinds of dialogue and different perspectives so that users can get a better understanding. This includes that the explanation is sufficient. Finally, integrating such explanation capabilities should not be too complicated and the explanation component should not affect the performance of the system.

We consider the understandability and sufficiency aspects as highly relevant for EASD. In general, the system is used by various kinds of users such as end-users, experts or developers. Each user group has different requirements and comes with different a priori knowledge with respect to the system itself and according domains. In this context, the explanation need of the system functionality must be addressing goals and preferences of users [8], such as *Transparency* and *Justification*. The first goal aims at imparting an understanding of how a system found an answer. This allows users to check the system by examining

the way it reasons and allows them to look for explanations for why the system has reached a surprising or anomalous result. The justification goal aims at increasing confidence in the advice or solution offered by a system by giving some kind of support for the conclusion suggested by the system. This goal allows for a simplification of the explanation compared to the actual process the system goes through to find a solution.

Depending on the goal, a particular explanation method is indicated. Here, the term 'method' denotes the dialogue and presentation of the explanation in a graphical or textual way, but also the kind of explanation. Spieker [9] distinguishes several useful kinds, among them *concept explanations* and *action explanations*. Concept explanations concern questions such as *What is (the semantics of) ... ?*. The purpose of action explanations is to explain the cause or provide a justification for a fact or a current situation.

Obviously, it is not possible to foresee all conceivable explanations with respect to all user groups. However, this is not the claim of EASD. The main contribution is not to add the explanation capability after a system is developed or to provide arbitrary explanations as it happened in many expert systems [2]. Instead, EASD intends to give developers the means to integrate suitable and understandable explanations into the design process of the entire system. For instance, one of our long-term goals is to extend UML in order to provide explanations in Java programs.

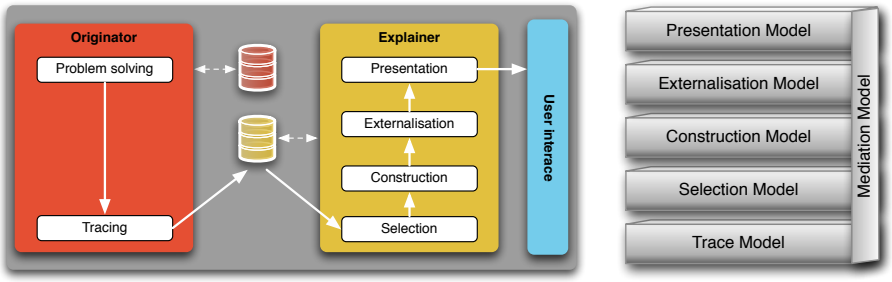
As mentioned before, the second key aspect of EASD becomes important at runtime of the system. The system has to be *aware* of the explanation scenario itself enabling a goal-oriented dialogue. In general, an explanation problem cannot be solved by a single explanation. Often, only an explanatory dialogue [10] can satisfy the explanation need.

We consider three participants in any explanation scenario [11]. We call the system or agent that provides the solution to some problem, a technical device, a plan, or a decision to be explained the *originator*. This agent is interested in which way the user reacts after receiving the explanation. The *user* is the second participant. He is the addressee of the explanation. Finally, the *explainer* presents the explanation to the user. This agent is interested in transferring the intention of the originator to the user as correctly as possible. The explainer selects the style of the explanation and is responsible for the computational aspects as well as for organising a dialogue if needed. The user applies the explanation in some way. With this application, in principle, a utility is connected, either for the user or for the originator. It depends on the application which utility dominates. Sometimes the user is mainly interested in reacting properly and sometimes the primary interest is on the side of the originator.

4 Multi-Layered Explanation Model

We propose an abstract architecture and a multi-layered explanation model for providing explanations as depicted in Fig. 1. The mentioned scenario implies that the originator has to provide detailed information about to its behaviour

and solutions regarding the fidelity desideratum. Therefore it is necessary that the originator prepares some kind of log or trace representing the initial starting point for the explainer to generate explanations. Regarding (potential) user questions, this information is step-by-step transformed into an adequate explanation including representational information.



(a) Architecture (b) Explanation Model

Fig. 1. Explanation-Aware Software Design

Depending on the coupling, originator and explainer share information sources that are required for explanation generation. However, the originator can have access to information which is hidden from the explainer and vice versa. At least, they have to share the tracing information.

As its name implies, the *tracing process* collects all information with respect to the behaviour of the originator for constructing the *trace model*. As it is not useful to collect virtually everything (efficiency desideratum), the information need must be predefined in the system design with respect to the explanation need of users. However, this contradicts the low construction overhead desideratum which should also be taken into account.

Any kind of explanation concerns only certain parts of the entire trace and thus, these parts must be identified. The respective transformation process, called *selection process*, extracts information from the trace with respect to user questions and dialogue situation representing basic explanation information. Hence, the selection represents a filter for the trace model.

This information does not necessarily represent complete explanation information. For this purpose, the selection model must be adapted to the current user requirements and context. The corresponding process is called *construction process* which is similar to the concept of reconstructive explanations as presented in Sect. 2. Here, supporting information can be added or confusing information can be removed. The construction process is used to span the entire explanation space comprising several possible explanation alternatives. Hence, the *construction model* contains the content of the explanation including domain knowledge and context depended information.

Explanation is information that is communicated by text, charts, tables, *etc.* Each communication form has different application possibilities in an explanation scenario. Text can describe complicated conceptions whereas charts can reveal qualitative connections in a simple way [12]. The *externalisation process* transforms the content model into a formal description for communicating explanations which is named as *externalisation model*. In this work, we put a special emphasis on semantic networks based on mathematical graphs for depicting explanations. However, we will consider further formal descriptions of tables, lists or natural language in the future.

The result of that process is not intended to represent the final explanation information because presentation information is missing. For instance, a mathematical graph model does not contain any layout information or written natural language uses a certain font. However, layout and style are important aspects of explanations influencing readability and understandability. For instance, highlighting important concepts in a textual explanation can serve as substantial and helpful support. The final transformation process transforms the externalisation model into the *presentation model* that can be leveraged by rendering engines to depict the explanation.

The main difference between this and existing approaches as presented in Sect. 2 is strict separation between content and form of an explanation. This allows depicting one and the same explanation in different forms which plays an important role with respect to user preferences [13].

A further difference is the explicit integration of the externalisation process. Each kind of externalisation offers various kinds of user interactions. For instance, lists can be sorted by means of a certain criterion or hyperlinks in texts can be followed. Regarding mathematical graphs, different exploration methods are conceivable such as closing of certain explanation paths. However, the most interesting aspect may be the application of various graph algorithms, such as shortest path algorithms. In case the construction model contains several possible explanation alternatives in form of paths the *shortest* explanation can be presented [14]. If users doubt that one, the next shortest path can be offered. Hence, the externalisation and presentation model play an important role to realise the understandability desideratum.

A further difference to other approaches is the concept of explanation-awareness at runtime of the system. For being *explanation-aware* the explainer has to trace his own transformation process including the results of the single process steps. Consequently, the explainer is aware of how content and presentation correspond to each other. In addition, it is aware of why an explanation is given in the current explanation scenario enabling explanatory dialogues and analysis of given explanations. This kind of information is stored in the *mediation model* addressing the sufficiency desideratum. For instance, if the explainer knows that a particular node of a graph-based explanation represents a domain concept, it can offer a respective concept explanation.

In the next sections we show how we realised the presented approach in the semantic search engine *KOIOS*.

5 The Semantic Search Engine *KOIOS*

KOIOS is a semantic search engine that enables keyword-based search on graph-shaped RDF data. *KOIOS* first computes a set of relevant SPARQL queries allowing users to select the most appropriate ones. Consequently, a selected query is used to search in a respective triple store. The main advantage is that users do not need explicit knowledge about the query syntax or the underlying ontologies. The presented approach is based on the work of [15] and [16].

For computing queries, the keywords are mapped to elements of input RDF data in a first step. Based on these *keyword elements*, a search is performed on the graph data to determine a *connecting element*. As its name implies, it is a particular graph element connecting the keyword elements. The paths between the keyword elements and the connecting elements are analysed to create a *matching subgraph*. For each subgraph, a conjunctive query is constructed by mapping the graph elements to query elements. To achieve a scalable search, the input data is preprocessed obtaining two data indices. The *keyword index* is used to realise the keyword mapping. For searching, a *graph index* is derived which represents a kind of summary of the input data containing structural patterns only. At runtime, this index is extended with the keyword elements obtained by means of the keyword index. Therefore, the extended graph comprises sufficient information to compute the SPARQL query.

The presentation of search results in common search engines always includes some hint why a particular result is relevant with respect to the keywords. Google, for instance, shows title, text snippets and URI of retrieved documents and highlights keywords contained therein. This information represents some kind of explanation or justification which describes a connection between search and result. In the end, it can help the user to open only relevant documents.

In case of *KOIOS*, explaining search results for end-users is bit more complicated. The justification has to contain the information how keywords are mapped to keyword elements and how the elements are connected. Finally, the interpretation as query has to be explained. In general, the same applies for developers when they debug the system. However, they need more detailed information and further intermediate results.

In the next section we describe how we used Semantic Web technology to realise some basic explanations.

6 Explanation Generation in *KOIOS*

In this section we present the realisation of the explanation facility of *KOIOS* using Semantic Web technology. For this purpose, we developed three RDFS ontologies which are used to describe the models of the explanation component. In addition, we implemented for each process a set of rules specifying how to transform a certain model.

As presented in Sect. 3 the starting point of the explanation generation is a kind of log provided by the originator. For that purpose, we developed a simple

process ontology in RDFS which comprises primitive concepts and relations for representing the *KOIOS* process called KOIOS *Process Language (KPL)*. It provides different specialisations of the basic class *Process* such as *KeywordMapping*, *GraphSearch* or *QueryConstruction*. In addition, KPL offers several classes for input and output of these processes. For instance, the class *IndexHit* is used to capture the mapping form keywords to a specific element of the input graph. As *KOIOS* is implemented in JAVA we used AspectJ[5] to design a logging module. The module instantiates the trace model of a particular search and provides it for the explainer for further processing. Here, the benefit of using aspect-oriented programming is that the logging module can be completely decoupled from the originator *KOIOS*.

For realising a graph based view on the process model we also developed the Mathematical Graph Language (MGL). In a nutshell, the MGL ontology includes primitive concepts and relations for representing knowledge about simple mathematical graphs and is described using RDFS. It contains three main constructs, namely *Graph*, *Node* and *Edge* indicating the corresponding mathematical concepts. In addition, the class *Markup* and its two differentiations *Labelling* and *Weighting* can be used to annotate graph elements. All related markups are part of a certain *Marker* which also has two subclasses, i.e., *Labeller* and *Weighter*.

MGL is used to describe the externalisation model but it does not contain any information of visualising a graph. Although an intuitive visualisation of a mathematical graph and labelling information is possible, the layout of the graph is important. Especially the sequence of *KOIOS* processes should be arranged properly for clarity reasons. Thus, we conceived a third ontology for visualising graph based information called VGL. The ontology provides elementary constructs for structuring graph nodes in a grid or circle layout. In addition, VGL offers a variety of node visualisations by means of symbols or icons including labels. Hence, the VGL is used to describe the visualisation model.

So far, we described the necessary ontologies to describe the different layers of the explanation model. For constructing the layers we used *HEPHAISTOS* which is a transformation language for the Semantic Web. It is based on Horn logic but is especially designed for querying and transforming RDF models. Although the formal semantics of RDF (open world) and Horn logic (closed world) are not conforming, *HEPHAISTOS* is a useful tool particularly with respect to explanation generation. The name of the language is inspired by the Greek god of forge, Hephaistos, who 'transformed' workpieces into other forms. Similar to such approaches as TRIPLE [17] or OntoBroker[6], the core language of *HEPHAISTOS* supports RDF primitives, *i. e.*, name spaces, resources, and statements. It also includes the representation of contexts of statements which extends statements by a fourth dimension to quadruples. The core language can be compiled into Horn logic programs and executed by Prolog systems. For integrating the Java framework RDF2Go[7] with *HEPHAISTOS* we implemented

[5] `http://www.eclipse.org/aspectj/`
[6] `http://www.ontoprise.de/en/home/products/ontobroker/`
[7] `http://semanticweb.org/wiki/RDF2Go`

a JNI interface to the Prolog system XSB.[8] The rule in Fig. 2 illustrates a simplified rule of the externalisation process which maps instances of the KPL class 'IndexHit' to instances of the MGL class 'Node' including labels.

```
01: <r1>
02:
03: true(H,uri('rdf','type'),uri('kpl','IndexHit'),
    uri('kpl','ConstContext')),
04: true(H,uri('kpl','element'),E,uri('kpl','ConstContext')),
05: X = true(H,uri('kpl','keywords'),K,uri('kpl','ConstContext')),
06: map([H],N),
07: map([X],L)
08:
09: ->
10:
11: true(N,uri('rdf','type'),uri('mgl','Node'),uri('kpl','GraphContext')),
12: true(L,uri('mgl','about'),N,uri('kpl','GraphContext')),
13: true(L,uri('mgl','label'),K,uri('kpl','GraphContext')),
14: true(L,uri('rdf','type'),uri('mgl','Labeling'),uri('kpl',
    'GraphContext')),
15: true(H,uri('kpl','isMappedTo'),N,uri('kpl','MediationContext')).
```

Fig. 2. Simplified rule of the externalisation process

The rule comprises three sections, namely rule declaration, input pattern and output pattern. The first section *<r1>* simply names the rule and has no further function. The following section describes the input pattern with respect to the construction model. It also includes instructions how elements of the construction model are mapped to elements of the externalisation model. In addition, it is also possible to add further constraints for separating similar rules from one another. The last section only contains the output pattern of the rule. It comprises quads for the externalisation and the mediation model as well (line 15). The rule contains three important predicates. The *true* predicate is used to represent either triples or quads depending on the number of arguments. The example only uses quads whereas the fourth argument represents the context or the URI of the explanation model. An URI is expressed by means of the *uri* predicate. In case the predicate has two arguments, the first argument represents an abbreviation of a namespace and the second argument a local name. Finally, the *map* predicate is used to 'construct' new resources out of existing ones (lines 6-7). The first argument is always a list of resources whereas the new resource is represented by the second argument. For mapping triples or quads, the equal operator has to be used (line 5). Every fact is assigned a unique ID. The equal operator employs the ID to construct an URI and assigns it to a variable. As a consequence, it is possible to map a set of triples to one resource.

[8] `http://xsb.sourceforge.net/`

For explaining search results of *KOIOS* we provided for each explanation need of end-users and developers a set of rules that transforms step-by-step the trace model into a visual model. The information of the visual model is leveraged by a particular renderer returning a Java JPanel with the entire explanation.

Fig. 3 shows the graphical user interface of *KOIOS*. In this case, the underlying graph-shaped data corresponds to YAGO[9] representing a huge knowledge base that is derived from Wikipedia[10] and WordNet.[11] The figure depicts a search for *Barack Obama* and everything this person *wrote*. The white left panel contains one of five computed SPARQL queries whereas the right white panel presents the respective result of the query. The interaction panel below is used to browse through the SPARQL queries, to change the interaction with the SPARQL queries, and to request certain kinds of explanations. In this version of *KOIOS*we realised only two kinds of explanation. Clicking on the earth symbol above the question mark symbol the user can obtain conceptual explanations of the instances and classes contained in a query. While maintaining the same externalisation mode the connections to other elements are revealed.

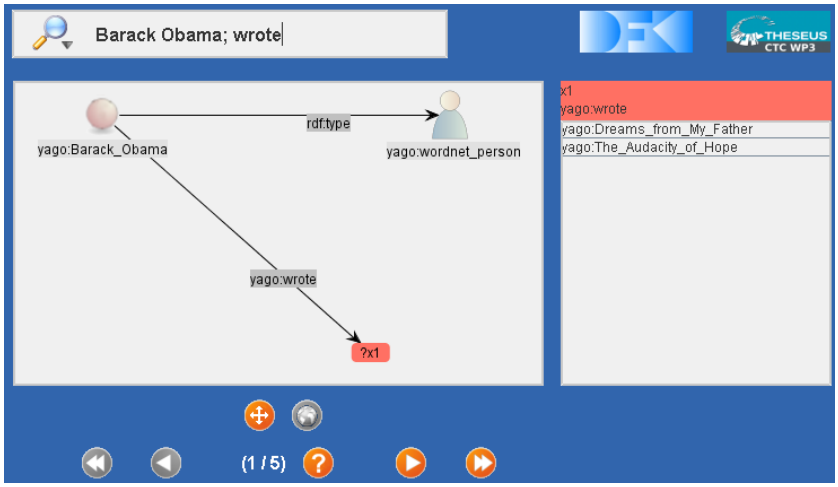


Fig. 3. Graphical User interface of *KOIOS*

The second kind of explanation is retrieved by clicking on the question mark symbol. Thereupon a new panel is opened containing action explanations. The explanations are intended for developers and try to justify the keyword-element mappings (justification goal). Regarding the example from Fig. 3 the developer may be interested in why the query contains a *yago:wordnet_person* instead of a *yago:wordnet_politician*. Fig. 4 provides the

[9] `http://www.mpi-inf.mpg.de/yago-naga/yago/`
[10] `http://en.wikipedia.org/`
[11] `http://wordnet.princeton.edu/`

hint that the resource *yago:Barack_ Obama* is typed with *yago:wordnet_ person* and *yago:wordnet_ politician* as well. However, the justification also provides the weights of both classes in the summary graph which is indicated by the float value in the node label. Hence, the class *yago:wordnet_ person* is considered most in the computation process addressing the transparency goal.

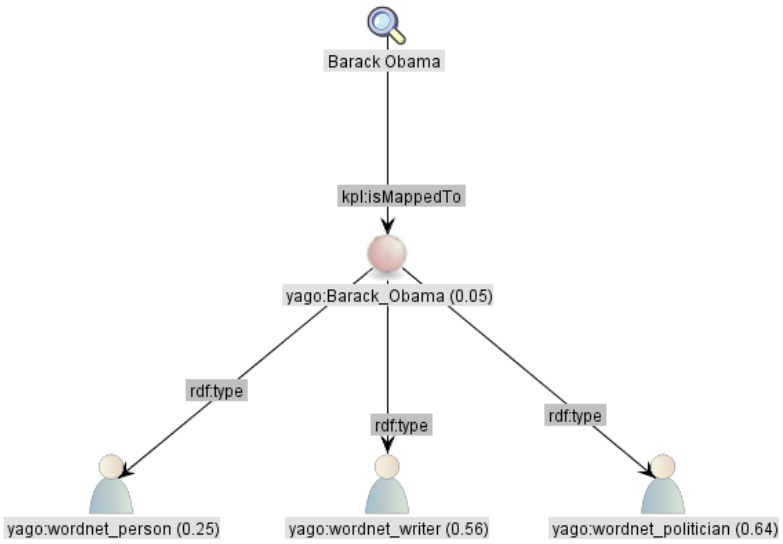


Fig. 4. Justification of keyword-element mapping

7 Summary and Outlook

In this paper we presented the explanation facility of the semantic search engine *KOIOS*. The facility was developed making use of Explanation-Aware Software Design (EASD) principles, which advocate the integration of explanation capability into the entire system development process. For that purpose, the explanation need of select users is collected implying particular explanation goals such as learning or transparency. Depending on such goals well defined explanation methods ensure that explanations fulfil user requirements. EASD explanation generation is based on a multi-layered explanation model and an iterative transformation process. In case of *KOIOS*, a process model is stepwise transformed into a visualisation model with respect to a particular user question. The constructed explanation model allows different views on explanations and enables various explanation methods such as sophisticated explanatory dialogues or the provision of alternative explanations.

The current process ontology of *KOIOS* is dedicated to its search algorithm. The next step of our work is to generalise the ontology for using it in other

knowledge-based systems. In addition, the provision of transformation rules for each explanation need is quite difficult due to syntax errors. Here, reusable rule patterns seem to be indicated.

References

1. Roth-Berghofer, T.: ExaCt manifesto: Explanation-aware computing (2009) `http://on-explanation.net/content/ExaCt_Manifesto.html`.
2. Richards, D.: Knowledge-based system explanation: The ripple-down rules alternative. In: Knowledge and Information Systems. Volume 5. (2003) 2–25
3. Wick, M.R., Thompson, W.B.: Reconstructive expert system explanation. Artif. Intell. **54**(1-2) (1992) 33–70
4. Ducassé, M., Noyé, J.: Logic programming environments: Dynamic program analysis and debugging. J. Log. Program. **19/20** (1994) 351–384 model of user-oriented program analysis.
5. Ishizaka Alessio, L.M.: How to program a domain independent tracer for explanations. Journal of Interactive Learning Research **17(1)** (2006) 57–69
6. McGuinness, D.L., Ding, L., Glass, A., Chang, C., Zeng, H., Furtado, V.: Explanation interfaces for the semantic web: Issues and models. In: Proceedings of the 3rd International Semantic Web User Interaction Workshop (SWUI'06). (2006)
7. Swartout, W.R., Moore, J.D.: Explanation in second generation expert systems. Second generation expert systems (1993) 543–585
8. Sørmo, F., Cassens, J.: Explanation Goals in Case-Based Reasoning. In Gervás, P., Gupta, K.M., eds.: Proceedings of the ECCBR 2004 Workshops. Number 142-04 in Technical Report of the Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Madrid (2004) 165–174
9. Spieker, P.: Natürlichsprachliche Erklärungen in technischen Expertensystemen. Dissertation, University of Kaiserslautern (1991)
10. Du, G., Richter, M.M., Ruhe, G.: An explanation oriented dialogue approach and its application to wicked planning problems. Computers and Artificial Intelligence **25**(2-3) (2006)
11. Roth-Berghofer, T.R., Richter, M.M.: On explanation. Künstliche Intelligenz **22**(2) (2008) 5–7
12. Wright, P., Reid, F.: Written information: Some alternatives to prose for expressing the outcomes of complex contingencies. Journal of Applied Psychology **57 (2)** (1973) 160–166
13. Kemp, E.A.: Communicating with a knowledge-based system. In Brezillon, P., ed.: Improving the Use of Knowledge-Based Systems with Explanation. (1992)
14. Roth-Berghofer, T., Forcher, B.: Improving the understandability of semantic search explanations. International Journal of Knowledge Engineering and Data Mining (IJKEDM) (2010) to appear.
15. Wang, H., Zhang, K., Liu, Q., Tran, T., Yu, Y.: Q2semantic: A lightweight keyword interface to semantic search. In: In Proceedings of the 5th International Semantic Web Conference (ESWC'08. (2008)
16. Tran, D.T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In: Proc. of the 25th Intern. Conference on Data Engineering (ICDE'09), Shanghai, China (2009)
17. Sintek, M., Decker, S.: TRIPLE – a query, inference, and transformation language for the semantic web. In: ISWC '02: Proc. of the First International Semantic Web Conference on The Semantic Web, London, UK, Springer-Verlag (2002) 364–378