Towards Adaptation and Evolution of Domain-specific Knowledge for Maintaining Secure Systems[*]

Thomas Ruhroth[1], Stefan Gärtner[2], Jens Bürger[1], Jan Jürjens[3], and Kurt Schneider[2]

[1] TU Dortmund, Germany; {thomas.ruhroth,jens.buerger}@cs.tu-dortmund.de
[2] Leibniz Universität Hannover, Germany
{stefan.gaertner,kurt.schneider}@inf.uni-hannover.de
[3] TU Dortmund and Fraunhofer ISST, Germany; http://jan.jurjens.de

Abstract. Creating and maintaining secure software require a good understanding of the system and its environment. Knowledge management is therefore one of the key factors to maintain secure software successfully. However, acquiring and modeling knowledge is a labor-intensive and time-consuming task. Thus, knowledge ought to be shared among different projects and must be adapted to their specific needs. In this paper, we present an approach allowing the stepwise adaptation from domain- to project-specific knowledge based on OWL ontologies. For this purpose, we define a basic set of adaptation operators which allows effective and frugal changes. Moreover, we discuss how our approach can be integrated into common software process models in order to adapt knowledge required for maintenance. Since domain- and project-specific knowledge changes over time, we show how our approach copes with changes efficiently, so that the affected knowledge remains consistent. The shared use of knowledge significantly reduces the complexity and effort to model required knowledge in various projects. Our case study and tool implementation shows the benefits for maintaining secure systems.

Keywords: ontology adaptation, domain-specific adaptation, maintaining secure systems, (co-)evolution

1 Introduction

Knowledge is part of a software development process at many points. While it often seems easy to take knowledge about security and compliance into account in the design phase of a project, it becomes more difficult during later maintenance phase. Overall, maintaining software is a knowledge-intensive activity [28]. Thus, it is important to have access to the knowledge that has already been used

[*] Funded by the DFG project SecVolution (JU 2734/2-1, SCHN 1072/4-1), part of the priority programe SPP 1593 "Design For Future - Managed Software Evolution".

for security analysis as well as additional knowledge gained while the development took place. Regarding security and compliance, required knowledge differs according to the domain and system context and also includes domain-specific knowledge as well as assumptions about the environment. Thus, some issues are common to a wide range of systems (e.g. encryption, privacy) and others need only to be considered for a specific domain or system (e.g. biometric, malware). Moreover, most required knowledge is nowadays documented in natural language and contains references to further natural language documents. Typical examples are regulations, laws, and best practices.

To support knowledge intensive tasks in an at least semiautomatic manner [6], a formal representation of knowledge is inevitable. Since collecting, formalizing, and maintaining required knowledge is a laborious task, a formal representation of knowledge should be shared among various projects and adapted to the specific needs and requirements. For example, privacy should be fulfilled by an organization according to given security standards and regulations. Therefore, the European Union defines privacy rules [5] which have to be refined by each member state (compare Germany: *Bundesdatenschutzgesetz* BDSG [2], France: *Data Protection Act* DPA98 [1]). Additionally, each organization has its own privacy guidelines extending and re-defining the respective country regulation.

To model such knowledge, ontologies are a commonly used technique [9,20]. They represent knowledge in a formal manner by using a set of types, properties, relationships, individuals, and axioms.

In this paper, we present a knowledge management approach based on ontologies which allows sharing common knowledge between projects for the integration in the maintenance process of the development cycle. The idea is to provide a simple and straight set of adaptation operators to support maintenance of domain- and project-specific knowledge. Moreover, knowledge management activities (e.g. use of ontologies) need to fit into the developers' workflow since rigid integration typically results in retarding instead of supporting the development as well as the maintenance process [17]. On this account, our approach is designed to fit well in existing development process models.

As knowledge evolves regularly, the corresponding links between domain- as well as project-specific knowledge must be re-adapted. This task, however, is laborious and error-prone and should be supported to cope with knowledge changes efficiently. Since the proposed set of adaptation operators is simple and straight, we show that our approach is suitable for this task.

The remainder of this paper is structured as follows: In Sec. 2 we sketch our approach and define the scope of our research as well as relevant research questions. The set of adaptation operators is proposed in Sec. 3 and it is shown in Sec. 4 that our approach is suitable to cope with knowledge changes efficiently. Afterwards, in Sec. 5 we introduce the case study and evaluate our approach. In Sec. 6, we discuss how to integrate our approach with common process models. Related research in the field of knowledge adaptation and evolution is listed in Sec. 7. Finally, in Sec. 8 our results and insights are discussed and future research is outlined.

2 Proposed Approach and Research Objective

In this paper, we focus on the distribution of domain-specific knowledge between different software projects. The aim of our research is to provide knowledge for evaluating the compliance and security of a software product. Thus, it is important to be compatible with different development process models and various domains.

We explain our research objective using the running case study of the paper that addresses privacy in software and its legal foundation. The European Union (EU) passed a privacy directive (Directive 95/46/EC [5]) which had to be refined into a local law of all EU member states. In Germany, the *Bundesdatenschutzgesetz* (BDSG) [2] resembles the local refinement of this directive. Some parts of Directive 95/46/EC are optional to implement and the BDSG does not implement all optional parts (e.g. regulations about safety of privacy data). In contrast to the EU directive, the BDSG was amended several times during the past years. In Fig. 1, the adaptation of the above-mentioned directives is shown. As Directive 95/46/EC is implemented differently in the member states, the first adaptation is required to refine EU privacy rules. The second adaptation reflects modifications necessary to fit the development process and different compliance analysis techniques. The resulting structure builds a hierarchy upon the privacy knowledge. However, adaptation of ontologies is not supported sufficiently by existing ontology frameworks at this moment.

Regarding our case study, we identified following research questions in software maintenance and knowledge management. The first question is **RQ1**: How can common or domain-specific knowledge efficiently be shared among different projects? Here, we assume that among different projects the domain-specific

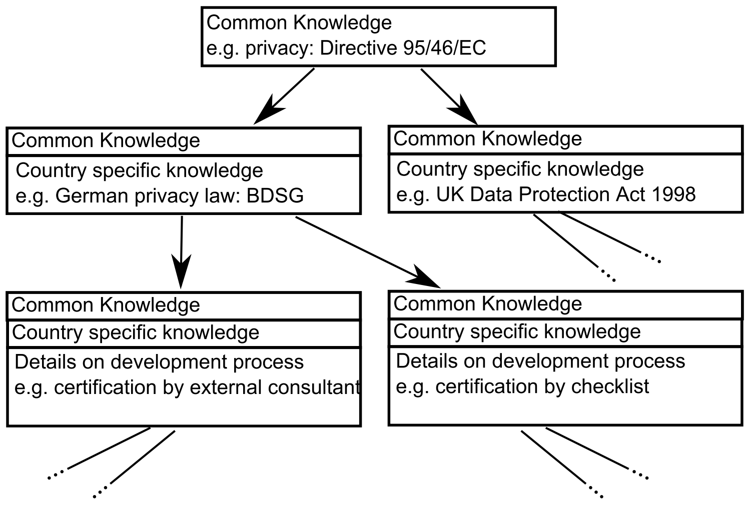
Common Knowledge
e.g. privacy: Directive 95/46/EC
Common Knowledge
Country specific knowledge
e.g. German privacy law: BDSG
Common Knowledge
Country specific knowledge
e.g. UK Data Protection Act 1998
Common Knowledge
Country specific knowledge
Details on development process
e.g. certification by external consultant
Common Knowledge
Country specific knowledge
Details on development process
e.g. certification by checklist

Fig. 1. Adaptation of knowledge

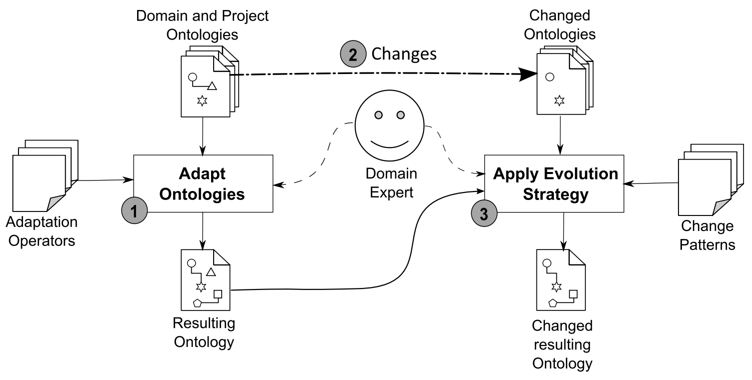

Fig. 2. Workflow of our proposed ontology adaptation approach.

knowledge only differs in details. This leads to the second research question **RQ2**: Which operations can adapt the common or domain-specific knowledge to fit the needs of different projects in a straight and sound manner?

To answer the given research questions, we propose an approach as illustrated in Fig. 2. Our approach consists of two activities: *Adapt Ontologies* (see 1) and *Apply Evolution Strategy* (see 3). To adapt ontologies to certain projects as well as process-specific needs, we establish a straight set of adaptation operators which are used by domain experts. To cope with changes (depicted as 2) of the original ontologies, we combine these operators with pre-defined knowledge changes. They are used to determine co-evolution strategies for the adapted ontologies semiautomatically. For ambiguous cases, the domain expert must decide which strategy fits best.

The knowledge adaptation presented in this paper facilitates the SecVolution [22,3] approach, which aims to maintain security properties in the software development process. For this purpose, SecVolution continuously monitors security knowledge and its evolution to determine required changes to the system semiautomatically. By applying our SecVolution approach, a long-living system is wrapped into a layer of relevant knowledge to maintain an appropriate level of security over a long period of time.

3 Operations for Modified Import of Ontologies

As described in Sec. 2, ontologies do not always match their intended use exactly. Hence, we must be able to *adapt* ontologies because during creation of a refined ontology, some knowledge may be missing, needs to be adapted or parts are not used in an adapted version. Furthermore, there may be *local* changes that are unique to an ontology of a specific layer.

OWL already features a mechanism to support the integration of knowledge called *import*. With this it becomes possible to simply insert elements from another ontology without any constraint. However, the import mechanism is

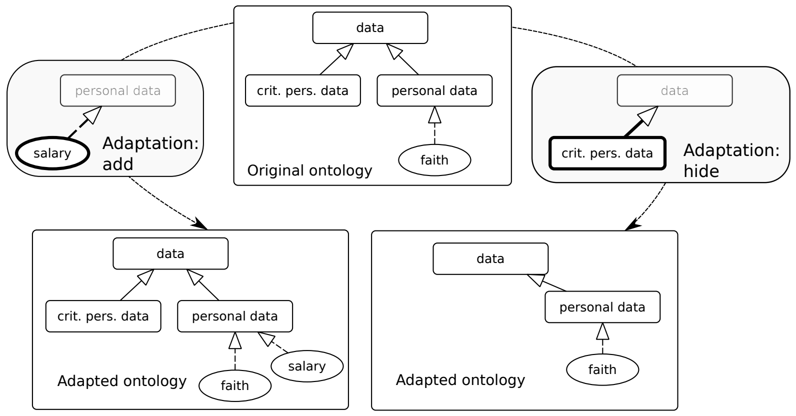

Fig. 3. Example depicting the hide and implicit add operation

insufficient because importing an element of an ontology may enforce a cascade of element additions. They may become necessary if a change needs to be applied to an element that has been imported beforehand. For example, regarding the EU directive for privacy of 2001, a number of classes have to be redefined when creating a refined ontology to reflect its German implementation. The standard import mechanism does not support redefinition of elements, so new concepts ought to be created. Hence, every role restriction and concept depending on these concepts needs to be changed as well so that the new (redefined) concept can use it.

In summary, it can be stated that not only the replaced parts of an ontology need to be changed but also every element that is directly or indirectly related to the elements that have been changed in the first place, needs to be altered as well.

To overcome the above-mentioned limitations, we propose a set of ontology modification operators extending OWL's import mechanism. The extension consists of two pairs of operators: *hide / unhide* and *change / reset*. These operators are used to realize the activity *Adapt Ontologies* (1) of our workflow presented in Fig. 2. By this means, it becomes possible to organize (i.e. *layer*) ontologies hierarchically in order to adapt them to project-specific needs. Table 1 gives an overview about the operators and their relation to each other.

Since there is currently no common graphical notation for ontologies, we use the representation as depicted in Fig. 3 throughout this paper. Individuals (rectangles) are the basic elements of ontologies and represent the actual knowledge. Individuals can share properties and are assigned to concepts (or classes) which are depicted as ovals. Axioms and properties are omitted in our representation due to simplicity and clearness of the example. Fig. 3 sketches an exemplary ontology that is adapted in two ways using two different adaptation operators.

The operator *add* (cf. Table 1) is implicitly defined by the existing import mechanism of OWL. It is easy to import a small ontology by featuring the elements that need to be added from the original ontology. Fig. 3 depicts the effect of the add operation. Here, the individual *salary* is added and thus is part of the adapted ontology.

The operation pair *hide / unhide* (cf. Table 1) allows to remove and to re-include elements from the original ontology. As Fig. 3 shows, the subconcept *critical personal data* is *hidden* by the adapted ontology or ontologies on lower layers. But in contrast to simply importing or ignoring elements, the unhide operator is useful to revert applied hidings. In comparison to simply adding the respective element, the unhide operation creates a link to the original element of the upper layer. This allows more sophisticated possibilities to react on the evolution of re-included elements, because original elements can be re-used and relations to further elements are preserved instead of simply creating a copy that just has the same name.

The operation pair *change / reset* (cf. Table 1) allows direct modification of elements, thus we can use straight adaptations avoiding many add and remove adaptations (straightness). For the reset operation the same arguments as for the unhide operator apply here.

As shown in Table 1, every adaptation operation is guarded by a precondition to ensure that the resulting ontology is syntactically and semantically sound. For example, a concept should only be hidden if it is not used by elements that are still visible. To ensure this, a concept can only be hidden if there is no unhidden reference pointing at it in the ontology. We assume that all adaptation operations between two ontologies take place in an atomic step and therefore avoid a problem with cyclic hide and unhide operations.

The adaptation operators are chosen in a way that they can build a set of simple, straight and revertable adaptation operations of ontologies. *Simple* means that the goal is to have a small set of basic operations that is sufficient to describe all possible modifications. The *straight*ness property describes that we can use the operations without having to resort to complex sequences which include pathological states. For example, the operators *add* and *remove* are as basic operations to describe all possible modifications, but the only use of them leads to complex adaptations. In the worst case, nearly or even all elements of

Operation	Description	Reverted by	Precondition
add	Addition of new OWL 2 elements. The add operation is defined by the usual means of the import.	hide	
hide	Removal of OWL 2 element.	unhide	There are only hidden references to this element
unhide	Reinclude a previously hidden element	(hide)	The element is not referencing any hidden element and the hide is in scope.
change	Change a property of an element	reset	
reset	Revert a change	(change)	

Table 1. Adaptation operators for include operations

an ontology have to be deleted until it contains only a few (or even no) elements and the whole ontology has to be rebuild in the modified structure from scratch. The third property *revertable* is used in import chains of ontologies when a former state should be reinstated. Regarding the running case study, reverting can be necessary if a system is implemented in Germany but is also involved in international (i.e. *EU-wide*) relationships. Therefore, the system inherently is related to the German adaptations of the privacy regulations but needs to revert some of them to get parts of the EU directive hold directly.

4 Application to Evolving Domain-Specific Knowledge

In Sec. 3 we explained the first part of our approach (see Fig. 2) that enables ontologies to import knowledge from one another. In this section, we show the second part dealing with changes of the ontology (evolution) and the needed modifications of adapted ontologies (co-evolution). Based on a formalism that describes possible patterns of ontology changes, we apply our approach exemplary and show how to determine strategies to solve co-evolution problems semi-automatically.

Figure 4 shows a typical adaptation as well as evolution process, featuring two ontologies on different layers. The boxes shows two states of the adaption where the left one is an evolved version of the right one. The question mark reflects an issue coming from the German privacy law BDSG as described in our running case study. In a former version of the BDSG, data is distinguished in two subcategories: anonymized data and personal data. During the adaptation process it was decided that, considering the specific system, data is needed to be distinguished into three categories: anonymized data, data of 1st / 2nd party and 3rd party data. As an evolution of the domain knowledge, we con-

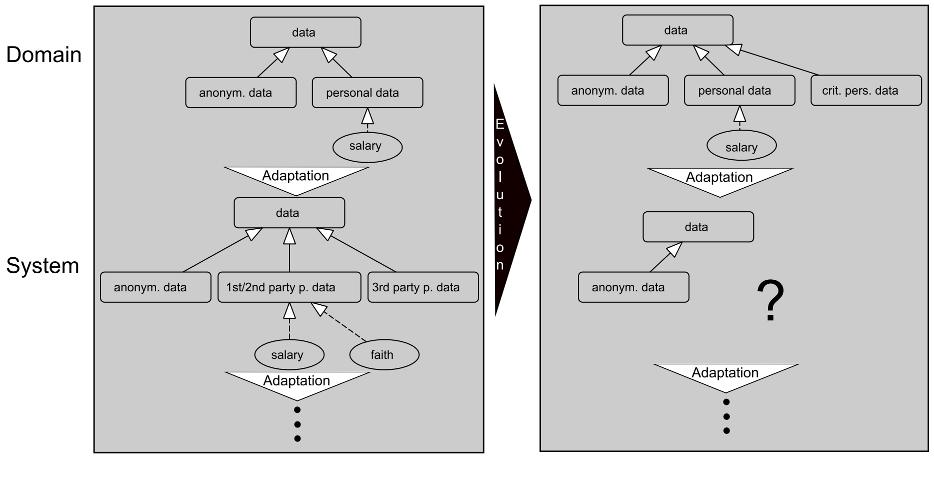

Fig. 4. Example of layered ontologies and the reaction to knowledge evolution

Change Pattern Parameters	Description	Constraint
Split Concept (X,($Y_1, \dots, Y_n$))	A class X is split into two or more classes $Y_1, \dots, Y_n$	Concept X deleted $\wedge$ Concept $Y_1, \dots, Y_n$ created $\wedge$ $\forall i$ superclass(X) = superclass(Y_i)
Merge Concepts (($Y_1, \dots, Y_n$), X)	One ore more classes $Y_1, \dots, Y_n$ are merged into one class X	Concept $Y_1, \dots, Y_n$ deleted $\wedge$ Concept X created $\wedge$ $\forall i$ superclass(Y_i) = superclass(X)
Pull up Concept (X, Y)	Concept X is pulled up to class Y	before: superclass(X) = Y, after: superclass(X) = superclass(Y)
Pull down Concept (X, Y)	Concept X is pulled down to class Y	before: superclass(X) = superclass(Y) after: superclass(X) = Y

Table 2. Examples for occurring change patterns (compare [12])

sider a change of the BDSG so that a new category is introduced to represent critical personal data. Thus, the domain ontology evolves and all adapted ontologies need to inherit these changes. Therefore, the changes that are unique to the system layer and the changes of the domain ontology have to be combined to update the whole ontology and to preserve consistency between the layers. Ontologies may become complex very fast, so avoiding inconsistencies is an important and non-trivial problem. Firstly, syntactic or structural inconsistencies arise when constraints are violated or entities cannot be referenced. Secondly, semantic inconsistencies arise when the meaning of an entity changes or gets ambiguous. Thirdly, adaptations themselves need to be evolved if the adapted elements evolve. Hence, the evolution process requires a detailed analysis of side effects as consequence of change operations [26].

For avoiding inconsistencies, it is important to perceive the ontology changes and their actual meaning. To keep track of what changes on a syntactic level mean on a semantic level, we use the concept of *change patterns* as proposed by Javed et al. [13,12]. These change patterns are used to provide the semantic part of the ontology evolution process. To resolve occurring inconsistencies and to assist the domain expert during the update process, the detected change patterns are further used to compute additional change sets to realize co-evolution. More precisely, there may be several strategies to resolve a specific inconsistency detected by the cause of an ontology evolution, and change patterns are the base for computing the possible alternatives. Note that, when considering a manual adaptation of changes, individuals or properties may not be visible to the domain expert during the process of applying modifications because they are not properly linked. This is an error-prone situation because if the ontology to be revised is complex, it is difficult to keep track of everything and information about how objects are linked can get lost.

An example of a typical ontology evolution is to split a class into two or more classes at the same hierarchical level (basically the same as in Fig. 4) Thus, the pattern *Split Concept* consists at least of three activities. First, a class is deleted. After that, at least two classes are inserted on the same hierarchical level.

These activities can surely be performed intermittently and in an arbitrary order. Hence, a given evolution resembling *Split Concept* can be realized in numerous ways. The discovery of relevant change patterns as well as finding a complete set of change patterns is a key task to make our approach able to react on evolution. It can be achieved by mining frequent patterns from change logs or by simply defining them as done in [13]. Some change patterns related to our running case study are shown in Table 2 according to [12]. Every pattern is defined by a function with a parameter set and a description. A list of constraints for each pattern shows which detected actions in the course of ontology evolution indicate that a given change pattern is present.

After all occurred changes are characterized by instances of change patterns, the consecutive step is to determine appropriate evolution strategies. Since every evolution has its own precondition, not all possible strategies are applicable in certain cases [23]. For example, if a class is split into two classes that are disjoint, it is not possible to re-attach the individuals to all new classes. If more than one evolution strategy is enabled for a given change pattern occurrence, different evolution strategies can also be combined arbitrarily.

In the following, we give an example how adaptation operators can be combined with change patterns and co-evolution strategies to realize the integration ontology evolution and co-evolution. Fig. 5 shows an evolution of a simple ontology with the split concept pattern and a possible evolution strategy following and extending the example of Fig. 4. On the left, the ontology in its unmodified state is shown. Now, the domain expert decides to refine the structure of the ontology such that normal and critical personal data can be distinguished. Thus, the *Split Concept* pattern occurs: The class *personal data* is deleted and two new classes are introduced, both also being subclasses of *data*. The resulting ontology is shown in the middle. The problem to be solved is how the individuals *salary* and *faith* should be linked now. To react upon the change pattern, a possible evolution strategy is to involve the domain expert who decides how every individual is re-attached to one of the new classes. More precisely, the *salary* individual will be re-attached to the *personal data* class and *faith* to the *crit. personal data* class. By applying this strategy of re-attaching to the two individuals, the ontology is now in a consistent state again.

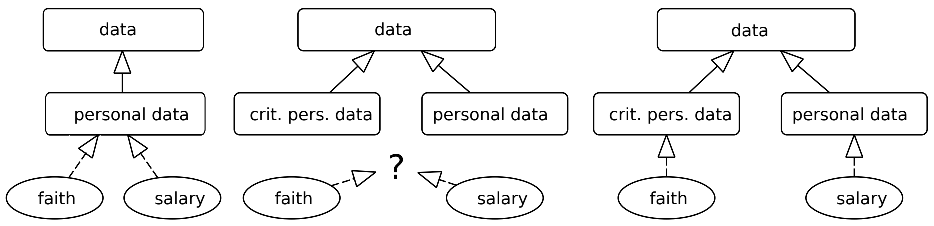

Fig. 5. Split Concept Pattern

5 Case Study and Implementation

To evaluate the feasibility of our approach, we implemented the adaptation stage in Java and OWL. To be compatible with standard OWL tools such as Protégé [25], we include the adaptation operators as annotations in the OWL files.

We use our prototype for our case study to show that our approach is capable of adapting knowledge in real world scenarios. Therefore, we use the BDSG to define the privacy properties in an extended version of the CoCoME case study [11]. CoCoME represents a trading system as it can be found in most supermarkets. The system consists of a number of cash desk PCs connected to a store server. A number of store servers again is connected to a central enterprise server. As the communication paths between these systems are used to transmit business as well as personal data (e.g. when processing electronic cash (EC) transactions), communication between the systems has to satisfy given security requirements. Moreover, a lot of additional hardware is plugged into the cash desk PC providing various entry points to the whole trading system.

In this case study, which is also part of a larger scenario of the SecVolution project, we describe the privacy regulations using three layers of ontologies: The global layer is defined by the EU directive (directive 95/46/EC), the domain is reflected by the German law (BDSG) and the system layer contains information specific to the CoCoME system and its development process model. In the global and domain layer, some concepts resembling regulations from the laws are defined. Moreover, various objects and subjects are introduced, e.g. different stakeholders like data subject or processor. Roles are defined so that they can be used in the system layer to define the relationship of different individuals. For example, we need to know if an operation requires the *the data subject's consent* (technically also called OptIn) before it is performed. The system layer includes information that is specific to the CoCoME, e.g. which data is to be considered as personal data. Using reasoning techniques, we can derive a number of concepts that allow to determine easily if the system is compliant to the laws.

An interesting point of this case study is that the BDSG failed to be a rigorous implementation of EU directive in the beginning, and thus needed some adaptations regarding concepts and roles. Later on, Germany was forced to change the BDSG to fully implement the directive and so raised the need for adaptation. In this case study, we therefore consider two significantly different versions of the BDSG.

The results of applying our approach to the case study are shown in Table 3. On the left, the number of elements is presented that are included in the basic ontology (EU Directive). The two other ontologies (BDSG before 2003 and BDSG 2009) are examples of adaptations of the basic ontology. Since both BDSG versions differ significantly, they are analyzed separately. For each BDSG version, the number of ontology elements are given that have been used to adapt the elements from the basic ontology. As presented in Table 3, the use of our adaptation operators in contrast to the standard OWL features reduces the number of elements up to 34%. Thus, the OWL import feature is not sufficient to build up an ontology hierarchy because the BDSG extends the meaning of derived concepts

EU Directive		BDSG (before 2003)(*)				BDSG (actual - 2009)			
	Size	Size		Difference		Size		Difference	
		without	with	# diffs	reduction	without	with	# diffs	reduction
		Adaptation				Adaptation			
Concepts	31	52	40	12	**23%**	46	39	7	**15%**
Individuals	4	4	4	0	**0%**	4	4	0	**0%**
Axioms	20	44	29	15	**34%**	41	29	12	**29%**
Roles	6	12	8	4	**33%**	12	8	4	**33%**
Assertions	0	0	0	0	**0%**	0	0	0	**0%**

Table 3. Sizes of privacy ontologies with/without using adaptation operators. (*) Not fully compliant with EU Directive.

that cannot be expressed without redefining parts of the ontology structure. For example, the EU directive does not incorporate the notion of anonymization which is only part of the BDSG. Anonymization means that if data is changed in a way that it cannot be associated with the subject anymore, the data can be used without the consent of the subject. Thus, the derived concepts needs to be redefined to include the new regulations. In summary, the use of adaptation operators reduces the number of newly defined elements in the ontologies due to the reuse of existing (changed) elements in contrast to additionally defined specializations.

6 Process Model Integration

In this section, we discuss how our approach as presented in Fig. 2 can be integrated with existing development and maintenance processes. To identify reasonable connection points, we therefore reviewed common development process models in software engineering. They reviewed models are: Waterfall Model (with iterations), Rapid Prototyping, iterative development, agile development, incremental development, and Boehm's spiral model (cmp. [18]). Moreover, the adapted knowledge must be used, re-adapted, co-evolved, and extended in the maintenance process. Here, we considered following models: Quick-Fix Model, Boehm's Model, Osborne's Model, and Iterative Enhancement Model (cmp. [7]).

During initial development, the required knowledge is adapted to the domain and project requirements as described above for activity *Adapt Ontologies* (see Fig. 2). For this purpose, the domain expert or requirements engineer chooses an appropriate knowledge base in the system analysis or requirements engineering phase that is provided by all abovementioned process models (part of the planning stage). The knowledge is adapted to the actual development project based on the corresponding project management, assumptions about the environment, and system as well as software requirements. In the development stages (e.g. design, coding, testing), the adapted knowledge can then be utilized by various assessment techniques to analyze software artifacts (cmp. [6]). Feedback cycles and iterations can be used to re-adapt and extend the knowledge during development. This is important because domain experts or requirements engineers are not able to adapt the knowledge at once, as they usually have to deal with vague requirements and constraints. Thus, process models which contain feedback cycles and iterations (e.g. Rapid Prototyping, agile development) increase the efficiency of our approach.

Compared with new development, maintenance process models have similar stages. But in software maintenance more effort is required in the early stages in order to understand the system at hand as well as to analyze and classify incoming change request [7]. Moreover, documentation is of particular importance in software maintenance to discover dependencies between development artifacts and to avoid ripple effects. Thus, knowledge that has been adapted during the initial development can be used to support these tasks. To incorporate knowledge changes properly, the analysis, re-design, or documentation stage and their corresponding feedback cycles of the maintenance process can be used. This includes to re-adapt, co-evolve, and extend the knowledge as required for activity *Apply Evolution Strategy* (see Fig. 2).

7 Related Research

Ontology reuse is an important issue in many areas such as knowledge management and software engineering. For this purpose, global ontologies must be adapted to a specific domain or system which is known as adaptation, integration, or matching. The fundamental problem is to specify the mapping between the global ontology and the local ontologies. To cope with this problem, Calvanese et al. [4] proposed a framework for ontology integration. Another approach addressing ontology integration based on integration operations is given by Pinto et al. [21]. Integration operators specify how elements from a global ontology are included and combined with elements in the local ontology. They comprise composing, combining, modifying, and assembling operators. Additionally, ontology integration is closely related to ontology alignment. There, relationships between two ontologies must be determined based on a pre-defined similarity measure. Udera et al. [29] proposed an approach leveraging the data and structure contained in ontologies. Alignment of ontologies can be used to identify knowledge elements which need to be added in an adaptation.

To cope with changes, Ruhroth et al. [23,24] described a basic set of evolutions and co-evolutions. It represents a formal foundation of (co-)evolutions which can be utilized in a wide scope of applications. Heflin et al. [10] pointed out the problem of changing ontologies in distributed environments. They argue that a changed ontology can be used without any problems if it is backwards compatible to the original ontology. An ontology is backwards compatible if only concepts and relations have been added. Otherwise, the ontology or query needs to be modified manually to reflect changes. Ontology evolution allows access to the elements through the most recent ontology, while ontology versioning allows access through different versions. For that reason, ontology evolution can be treated as part of ontology versioning [16]. Noy et al. [19] emphasized that existing versioning systems are not able to compare and represent structural changes. Thus, they present an ontology versioning environment to address the problem of maintaining versions of ontologies. Based on this approach, the domain expert is able to analyze changes in order to accept or reject them. They extended their approach in [20] and proposed a framework for ontology evolution in collaborative environments based on their own experiences. Stojanovic et al. [27] proposed a well-structured ontology evolution process which provides the domain expert capabilities to control it. Therefore, the domain expert can select one of the proposed co-evolution strategies. Moreover, Stojanovic [26] also presented an approach to solve inconsistencies by using co-evolution strategies. For this purpose, they used a model for the semantic of changes in OWL ontologies as well as resolution strategies to ensure consistency [8]. Javed et al. [13,12] presented a graph based approach of pattern matching and the discovery of frequent changes in ontologies.

In summary, most approaches propose a rather large number of adaptation operators which makes adaptation of ontologies complex and error-prone. Thus, to support maintenance of knowledge more efficiently we presented a simple and straight set of operators. Hence, our operators are more suitable to cope with evolution and co-evolution of knowledge. In contrast, approaches regarding ontology changes in response to certain needs do not cope with integrated ontologies sufficiently or even address co-evolution strategies.

8 Discussion and Conclusion

We introduced a set of adaptation operators that allow one to modify an ontology on special needs of different projects. Thus, we can share common knowledge between projects without ignoring the different needs and also reducing the workload for the elicitation and maintenance of knowledge. The case study as introduced in Sec. 5 shows that the use of adaptation operators can reduce the size of modifications significantly. The effect can be explained by the straightness property. Since modifications can be made locally, the changes are smaller compared to the case where only a minimal set of operations (add/remove) are used.

We discuss the achieved results with our research questions: *RQ1: How can common or domain-specific knowledge efficiently be shared among different projects?* Knowledge given by an ontology can be shared among projects by using stepwise adaptation to different domains and in the last step to the system itself. As presented in Fig. 1, knowledge is therefore hierarchically structured. This means that the most common knowledge is stored on top of the hierarchy. Thus, a maximal amount of the knowledge can be reused. Additionally, maintaining widespread knowledge becomes more straightforward.

RQ2: Which operations can adapt the common or domain-specific knowledge to fit the needs of different projects in a straight and sound manner? The defined adaptation operators (add, hide/unhide, change/reset) give an answer to RQ2. Adaptation operators can be used to add, change or remove knowledge. They are complete in the sense that they can be used to adapt an ontology to any other ontology. To show this, we can use the sometimes used argument by destructing the ontology by removing (here hiding) all elements, such that we get an empty ontology. Afterwards we can build any new ontology. The strength of our adaptation set is to be compatible with evolution, such that the evolution of common knowledge can be co-evolved in the adapted versions. Since many co-evolutions need a manual interaction, the straightness and the reversibility reduces the amount of manual interaction. This is also a result of the small adaptations between ontologies.

The presented approach is a foundation of the central part of the approach developed in the project SecVolution, namely the Security Maintenance Model. The sharing of global knowledge as well as the adaptation and evolution of the Security Maintenance Model is used to trigger semiautomatic evolutions of the software itself and thus forms an important base for a security aware evolution approach of long-living software systems. The advantages of the presented approach are used to integrate our project work with other projects in the joint case study CoCoME of the SPP "Design for Future - Managed Software Evolution". In future work, we furthermore plan to use the results presented here in the context of model-based development of secure software [15,14].

In summary, software maintenance of long-living software can benefit from domain-specific knowledge which is gathered and adapted during development. Our approach aims to decrease the effort to adapt required knowledge and thus increases the return of investment.

References

1. British Parliament: Data Protection Act 1998
2. Bundesministerium des Inneren: Bundesdatenschutzgesetz. Bundesgesetzblatt
3. Bürger, J., Jürjens, J., Ruhroth, T., Gärtner, S., Schneider, K.: Model-based security engineering with UML: Managed co-evolution of security knowledge and software models. In Aldini, A., Lopez, J., Martinelli, F., eds.: Foundations of Security Analysis and Desing VII: FOSAD Tutorial Lectures. Volume 8604 of Lecture Notes in Computer Science. (2014) 34–53
4. Calvanese, D., De Giacomo, G., Lenzerini, M.: A Framework for Ontology Integration. In: The Emerging Semantic Web. IOS Press (2002)
5. EU Parliament: Directive 95/46/EC of the european parliament and of the council of 24 october 1995. Official Journal of the European Union **L 281** (1995) 0031–0050
6. Gärtner, S., Ruhroth, T., Bürger, J., Schneider, K., Jürjens, J.: Maintaining Requirements for Long-Living Software Systems by Incorporating Security Knowledge. In: 22nd IEEE International Requirements Engineering Conference, IEEE (2014) 103–112
7. Grubb, P., Takang, A.: Software Maintenance: Concepts and Practice. World Scientific (2003)
8. Haase, P., Stojanovic, L.: Consistent evolution of OWL ontologies. In: Proc. of ESWC, Springer (2005) 182–197
9. Happel, H., Seedorf, S.: Applications of ontologies in software engineering. In: Proc. of Workshop on Sematic Web Enabled Software Engineering (SWESE). (2006)
10. Heflin, J., Hendler, J., Luke, S.: Coping with changing ontologies in a distributed environment. In: AAAI-99 Workshop on Ontology Management. (1999)

11. Herold, S., Klus, H., Welsch, Y., Deiters, C., Rausch, A., Reussner, R., Krogmann, K., Koziolek, H., Mirandola, R., Hummel, B., Meisinger, M., Pfaller, C.: CoCoME. In: The Common Component Modeling Example. Volume 5153 of LNCS. (2007) 16–53
12. Javed, M.: Operational Change Management and Change Pattern Identification for Ontology Evolution. PhD thesis, Dublin City University (May 2013)
13. Javed, M., Abgaz, Y.M., Pahl, C.: Ontology change management and identification of change patterns. J. Data Semantics **2**(2-3) (2013) 119–143
14. Jürjens, J.: Secure Systems Development with UML. Springer (2005)
15. Jürjens, J., Wimmel, G.: Security modelling for electronic commerce: The Common Electronic Purse Specifications. In: First IFIP conference on e-commerce, e-business, and e-government (I3E), Kluwer (2001) 489–506
16. Klein, M., Fensel, D.: Ontology versioning on the Semantic Web. In: SWWS. (2001) 75–91
17. Meyer, S., Averbakh, A., Ronneberger, T., Schneider, K.: Experiences from Establishing Knowledge Management in a Joint Research Project. In: Product-Focused Software Process Improvement. Volume 7343 of LNCS., Springer (2012)
18. Münch, J., Armbrust, O., Kowalczyk, M., Soto, M.: Software Process Definition and Management. Springer (2012)
19. Noy, N.F., Kunnatur, S., Klein, M., Musen, M.A.: Tracking changes during ontology evolution. In: Proc. of the 3rd (ISWC), Springer (2004) 259–273
20. Noy, N., Chugh, A., Liu, W., Musen, M.: A framework for ontology evolution in collaborative environments. In: Proc. of the 5th ISWC. (2006) 544–558
21. Pinto, H.S., Martins, J.a.P.: A methodology for ontology integration. In: Proc. of K-CAP, ACM (2001) 131–138
22. Ruhroth, T., Gärtner, S., Bürger, J., Jürjens, J., Schneider, K.: Versioning and evolution requirements for model-based system development. In: International Workshop on Comparison and Versioning of Software Models (CVSM). (2014)
23. Ruhroth, T., Wehrheim, H.: Refinement-preserving co-evolution. In Breitman, K., Cavalcanti, A., eds.: Formal Methods and Software Engineering, ICFEM 2009. Volume 5885 of LNCS., Springer (2009) 620–638
24. Ruhroth, T., Wehrheim, H.: Model evolution and refinement. Science of Computer Programming **77**(3) (2012) 270 – 289
25. Stanford Center for Biomedical Informatics Research (BMIR): Protege - homepage. `http://protege.stanford.edu`
26. Stojanovic, L.: Methods and tools for ontology evolution. PhD thesis, Karlsruhe Institute of Technology (2004)
27. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: Proc. of the 13th EKAW, Springer (2002) 285–300
28. Tiwana, A.: An empirical study of the effect of knowledge integration on software development performance. Information and Software Technology **46**(13) (October 2004) 899–906
29. Udrea, O., Getoor, L., Miller, R.J.: Leveraging data and structure in ontology integration. In: Proc. of SIGMOD, ACM (2007) 449–460