Fuzzy Extensions for Reverse Engineering Repository Models

Ulrike Kölsch
T-Systems International GmbH
Fasanenweg 5
70771 Leinfelden-Echterdingen, Germany
Email: koelsch@acm.org

René Witte
Concordia University,
Department of Computer Science
Montréal, Québec, Canada
Email: me@rene-witte.net

Abstract

Reverse Engineering is a process fraught with imperfections. The importance of dealing with non-precise, possibly inconsistent data explicitly when interacting with the reverse engineer has been pointed out before.

In this paper, we go one step further: we argue that the complete *reverse engineering process must be augmented with a formal representation model capable of modeling imperfections. This includes automatic as well as human-centered tools.*

We show how this can be achieved by merging a fuzzy set-theory based knowledge representation model with a reverse engineering repository. Our approach is not only capable of modeling a wide range of different kinds of imperfections (uncertain as well as vague information), but also admits robust processing models by defining explicit degrees *of certainty and their modification through fuzzy belief revision operators.*

The repository-centered approch is proposed as the foundation for a new generation of reverse engineering tools. We show how various RE tasks can benefit from our approach and state first design ideas for fuzzy reverse engineering tools.

1. Introduction

Reverse engineering (RE) is a process permeated by imprecisions. It has already been suggested [10] that imperfect knowledge must be modeled explicitly, in order to exploit it for automatic and human-centered RE tasks.

Reverse engineering, especially of large, industrial-sized legacy systems, can be described as a cyclic and iterative discovery process as well as a cooperative task, performed by a group of reverse engineers with the goal to gain a more abstract and semantically richer description of the system examined [18]. The process evolves as a sequence of bottom-up and top-down process steps using multiple sources, methods, heuristics, and tools. Their choice depends on the current situation and the intention of the performing reengineer. The goal is to acquire as much knowledge about the examined system as possible from the information available at a certain point in time. The sequence of steps performed is neither predefined nor precisely formalizable, but is determined by the individual situation of each RE project.

This behaviour leads to a vast and diverse set of heterogeneous results. Usually, these results are neither stored in a homogenous way nor are they categorized according to origin, quality, and reliability. Additionally, due to the very nature of reverse engineering, most of the information obtained is permeated by imperfections, which typically cannot be represented explicitly. As a consequence, the knowledge stored does not result in an accurate representation of the actual system, leading to a loss of important information that could improve further analyses.

To overcome this situation we proposed the adoption of the repository concept [16, 17]. Already, it has proven to be useful in forward software engineering. Successful CASE tools like *Together, Rational Rose,* or *ArcStyler* contain a repository based on a domain model, which allows to define and represent a system's design and implementation artefacts in an appropriate way.

We recommend the use of a model-based repository to consolidate and aggregate all results, information, and knowledge gathered while reverse engineering a legacy system. Integration of diverse knowledge is needed in order to achieve an abstract target representation of the legacy system. Here, the repository's conceptual model determines its representation and level of abstraction.

Since imperfection is an intrinsic characteristic of reverse engineering [10], the repositories used within reenginering frameworks must be enhanced in order to adequatly capture and process imperfect information.

The remainder of this paper is structured as follows: in the next section we analyse the requirements for such a repository model. Additionally, we illustrate these requirements with a number of deficiencies we detected through our TAO repository model [17]. We propose a fuzzy set-theory based representation and processing model that satis-

Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03)
1095-1350/03 $ 17.00 © 2003 IEEE

fies these requirements. The integration of this fuzzy model with our repository and its deployment in a fuzzy RE framework is presented in section 4. Finally, we discuss related work, followed by conclusions and ideas for future research.

2. Analysis — Imperfect information in RE

The importance of incorporating imperfect information into the reverse engineering process has been pointed out before [10]. We agree with Jahnke and Walenstein insofar as the impedance mismatch between the quality and characteristics of reverse engineering knowledge and its representation within repositories and tools leads to major problems.

In contrast to Jahnke and Walenstein, we are of the opinion that imperfection should be regarded as an intrinsic part of the reverse engineering domain, and not as a feature of specific items, methods, or tools. Consequently, these characteristics should be incorporated into the modeling approach used for defining the domain model, in such a way that *all* reengineers, tools, and methods can make use of an adequate representation of imperfect knowledge.

In this section, we analyse the requirements concerning *imperfect information* within the reverse engineering domain. We extract three major requirements accompanied by a major constraint. These requirements manifest the features that must be fulfilled by a model in order to support the proper handling of imperfect knowledge.

2.1. Illustration of the requirements using the TAO framework

By definition, reverse engineering is described as "a process of analyzing a subject system to *(i)* identify the system's components and their interrelationships and *(ii)* create representations of the system in another form or a higher level of abstraction" [7].

But the process of knowledge extraction as well as the methods and tools used are ambiguous and error-prone. In order to support reverse modeling, a repository is employed to accumulate data and support further processing. The repository domain models represent the systems under examination on different levels of abstraction. At least three levels — implementation, design, and requirements — are widely accepted.

Equally known is the *impedance mismatch*, which describes the problem that the assignment of semantically rich design concepts to more constrained implementation solutions (and vice versa) can not always be decided unambiguously. Therefore, assignment decisions cannot be determined automatically but rather need human intervention [8].

When we regard the assignment problem from a reverse engineering perspective the problem becomes even more severe. The RE process produces a lot of isolated results by different methods, tools, analyses, inspections, etc. Such an abundance of disjoint information leads to the problem of collecting them in a homogenous and contradiction-free way into a *single* repository that represents the legacy system's reverse model. The information obtained during RE must be derived from the implementation level, which is usually not precise enough to determine the original design concepts that have been implemented by the extracted implementation artefacts.

Thus, the target model must be built gradually and iteratively based on the evidence found, developing more abstract concepts in a step-by-step fashion, which allows for the pursuit of different possible reverse design concepts in parallel.

Reverse engineering example: In order to illustrate the dilemma that arises from the deterministic modeling approach of RE domain models consider the following simple example of an RE process: A legacy system is examined regarding a specific variable *i*. For the scope of this example we are especially interested in the variable's *type*, which in turn is needed to derive its *usage* within the legacy system. Code fragments as well as data and available documentation are scanned. Several results are found by different methods and tools:

Data Flow Analysis: The result of a data flow analysis is that variable *i* is defined as *char*.

Control Flow Analysis: The result of a control flow analysis shows that variable *i* is used in certain control flow statements *(if, case)*. This indicates that it might serve as a control flow variable — but since it is also used in I/O operations, we cannot be certain of that result. We therefore store this information, annotated with an assessment of its relevance, in a free-form textfield.

Domain Expert: A domain expert discovered a document stating that variable *i* represents the seven days of the week. However, since the documentation is somewhat outdated, the domain expert stores the information as an uncertain result and adds her assessment as being *"quite certain"* in a free-form textfield.

Developer: A developer opines that variable *i* is of type *integer*, effectively storing a one-byte integer representation of the seven days of the week including a default value zero for the case "no day assigned." But an *integer* type definition cannot be derived from the source code, only control statements imply that the value of *i* lies between 0 and 7. So, his information is valuable but again *"not certain."*

Data Mining: The data mining tool extracts all instances of the variable *i* stored in a database. It reports a result indicating a domain range between -1 and 31, but showing a significant density for the values 0–8. Hence, the reengineer takes this to be a strong indication for a data range between 0 and 8. The other values could be the result of old

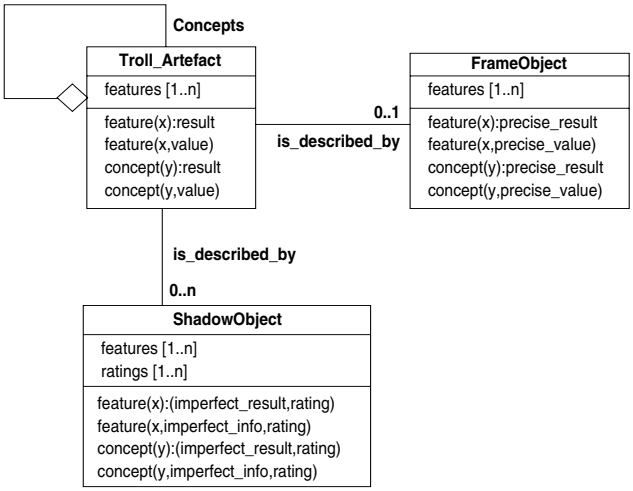

Figure 1. The generic TAO model

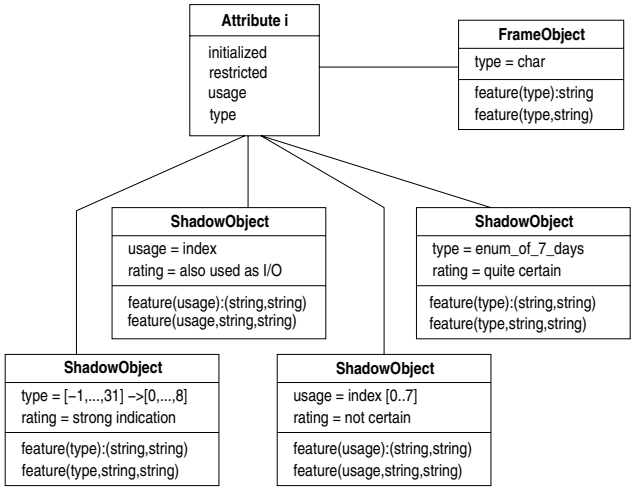

Figure 2. Instance model of an ATTRIBUTE *i*

or dirty data, which are negligible. This result is stored denoting variance and deviation of the data mining result in a free-form textfield.

Generally, domain models of RE repositories are based on proven modeling concepts, such as UML, OMT, or ER as meta modeling languages. The domain models have been designed in order to describe legacy systems in terms of their target model. E.g., the domain models of COREM, Varlet, or TAO are object-oriented and describe a legacy system in terms of classes, associations, attributes, and other OO concepts.

However, by using a repository based on crisp conceptual modeling languages we are severely restricted by the fact that we cannot store all results gained concerning an artefact, like the attribute *i*. This limitation of repository models is due to the underlying conceptualization, which is not able to deal with imperfections like vagueness, multiple options, or uncertainty.

So far, repositories offer two alternatives for the storage of artefact-result pairs:

1. Every artefact-result instance is stored separately. No interdependency between the different result items can be supported by this approach. Particularly the confirmation of knowledge through similar and supportive results is not possible.
2. Only the most certain result concerning one artefact is stored in the repository. All other information has to be neglected as less valuable and ignored, even though it might be useful or become more certain during the ongoing process.

In our approach, the TAO framework [16,17], we already tried to overcome this constraint by extending the domain model, including the notion of what we called hypothetical (not finally validated) knowledge. We therefore decided to split the repository model into two parts: one part representing the precise and final knowledge (the FRAME classes), and the other part representing the hypothetic knowledge (the SHADOW classes). All artefacts in the TAO model are defined as an aggregation of $0 \ldots 1$ FRAME and $0 \ldots n$ sibling SHADOW objects as shown in Figure 1. The model of the TAO framework is based on the formal object-oriented specification language TROLL, providing us with all concepts needed to describe a system in an object-oriented way [6]. Figure 2 shows how the RE results of an artefact ATTRIBUTE *i* can be encoded as a TAO repository instance. Although the TAO model can store imperfect information that arises during the RE process, we detected several shortcomings with this approach:

Insufficient expressiveness of the representation formalism. The degree of *certainty* or *correctness* of a result cannot be represented through a formally defined concept. Since it is just a textfield, all kinds of representations can be found, single adjectives as well as short phrases and statistical measures. These results form an uncomparable heap of data that cannot be automatically processed for further analyses.

No management and interpretation of joined imperfect data instances. The TAO framework lacks functionality to evolve the knowledge base according to information gathered during RE. The number of uncertain but valid results collected about attribute *i* illustrates this issue. After obtaining more and more evidence the uncertain belief that attribute *i* is of type *integer*, representing the seven days of the week plus zero as default, should be more certain than every single result alone. And this fact should be reflected as nearly certain knowledge about the artefact *i* within the repository.

Different representation of certain and imperfect information. Having different representations for imperfect and crisp knowledge leads to problems when evolving knowledge. Knowledge discovery often starts with a guess. Throughout the discovery process supporting or opposing facts are found, leading towards a valid assumption. The knowledge itself, as well as its change, must be representable within a repository model. But strict models like TAO refer to imperfect and precise knowledge in a different way. Therefore the change of a knowledge instance from uncertain to certain during its lifetime cannot be modeled. Knowledge discovery in RE processes lacks adequate support for imperfect information, as already stated by Jahnke and Walenstein [10].

2.2. Detected requirements for handling imperfect information in RE repository models

All issues detected in the last subsection share one key feature: strict modeling concepts lack the ability to adequately support the reverse engineering domain. Issues that force a change in repository models are essential to reverse engineering: results are fraught with imperfections, since the derivation of an abstract concept from implementation sources is usually not precisely determined. Results can be supported or invalidated during the ongoing RE process by additional, possibly conflicting results. Numerous sources of different quality and certainty supply information about a specific artefact. Their integration and consistency-preservation must be performed automatically, especially with regard to the amount of data produced by reverse engineering an industrial-strength legacy system.

At present, repository models are neither able to support the explicit representation of imperfection nor do they enable the processing of such knowledge instances.

We therefore propose to overcome these constraints of reverse engineering domain models by extending their abilities to handle imperfections. However, contrary to the approach of the TAO model, we do not extend the domain model but rather aim for an extension of the underlying conceptual model, i.e., a meta modeling approach.

Thus, an extension of conceptual models has the following requirements:

Imperfection. Information gathered during the different RE process steps need to be rated concerning quality and credibility. This should be based on a strict and formal method in order to ensure a homogeneous rating of all results in the repository and to guarantee comparability. It is useful to base the rating method on a known and proven concept covering the handling of several kinds of imperfections, both uncertain and vague knowledge. This will allow us to describe the quality and credibility of information in a neutral way admitting a homogeneous, comparable, and automatically processable representation of imperfect knowledge.

Figure 2 emphasizes the importance of this requirement. In the TAO repository all results concerning artefact *Attribute i* are rated differently and informally, leading to uncomparable pieces of information.

Automatic management of imperfect data. As suggested by [10], it is necessary to explicitly model imperfect knowledge, to be able to use it for further (automatic) analyses. But it is not sufficient to model only the static aspect, the management of imperfect data must be dealt with as well. This concerns the most basic data management problems like inserting, updating, or deleting imperfect data instances. Moreover, the aspects of knowledge propagation and consolidation must be addressed. Some imperfect data instances may form interdependencies, one enforcing or suppressing the other. It is therefore necessary to develop operations based on imperfect data models.

The following scenario based on the prior example gives a motivation for this requirement. Storing the crisp result — *attribute i is implemented as a char* — should lead to a (partial) inconsistency warning when another crisp result is stored stating that *i* is implemented as an integer. On the other hand, a crisp information *i is integer* and an uncertain information *i is enumeration* are at least partially compatible. Attribute *i* as char might as well be the implementation solution for the design type integer (storing small numerical values in a single byte). During the ongoing RE process, more results that indicate an integer type as the correct RE result arise. In the end this fact should be reflected in the repository in such a way that it contains the knowledge of *i* being of type integer encoding the days of the week as the most certain and semantically richest result.

Uniform knowledge representation. An RE repository describes a legacy system in terms of artefacts defined by a specific target model. The necessary extensions for handling imperfect data should not change the domain model, as not the domain artefacts are vague or uncertain but the RE knowledge about those artefacts. Therefore the representation of knowledge ought to be modeled in a way that reflects its quality, origin, and credibility. A semantically rich representation model should be able to handle both precise and imperfect information in a common framework.

Additionally, it is necessary to solve the assignment problem. All RE data must be stored closely with their artefact. There should be no limit as to how many different or even contradicting results can be assigned to a specific artefact.

The points mentioned above represent a set of requirements which can not be fulfilled by conventional strict information modeling concepts like UML, TROLL, ER, or relational modeling. These features are orthogonal to the familar concepts of data and information modeling.

In order to avoid "reinventing the wheel," a key feature of an enabling and holistic concept must be the following mandatory constraint: *the ability to integrate the approach*

within already existing solutions. It must be possible to enhace existing RE frameworks and tools through an orthogonal extension, without the need for large restructuring or reimplementation.

3. The fuzzy model

We now present the formal representation model we developed for the discovered requirements.

The central requirement for such a model is the ability to handle several kinds of imperfections: *uncertain* and *vague* information, as well as the ability to handle *inconsistencies*. These requirements match the ones stated in [10].

Fuzzy-set theory is a well-known representation formalism that can handle both kinds of imperfections [3, 13]. It has also proven successful within a wide range of industrial applications, most notably through the area of *fuzzy control*.

For the application within an RE repository we deploy a fuzzy-set based model that has been developed especially for the use within information systems [22]. It provides the necessary features for supporting the requirements of the RE process, especially the ability to backtrack information and modify a knowledge base through non-monotonic belief revision operators. We first review the basic features of the model and then present some formal enhancements that were necessary to capture the semantically rich structure of RE information.

3.1. Representing imperfect information

The basic representational unit with our model is a *fuzzy set*. A fuzzy set μ_A extends the characteristic function of a set A from the binary $\{0, 1\}$ *(not member, member)* to a continuous interval, typically in [0,1]. Thus, each element ω from a domain Ω is assigned a corresponding *membership degree* $\mu(\omega)$ with which this element belongs to the fuzzy set $\mu_A \in F(\Omega)$.

A single fuzzy set can capture a single piece of imperfect information, both vague and uncertain. However, within an RE repository a multitude of information from different sources has to be managed, stored, and combined. As pointed out in [23], mixing the semantics of different sources into a single fuzzy set is not a suitable approach for information systems, because it is important to keep track of the source of each individual piece of (imperfect) information. This can be achieved with a model that combines the semantics of fuzzy sets with the syntactical features from propositional logic [23]. Classical propositional logic allows the construction of complex propositions from primitives, the *atoms*, and their logical combination to *literals*, *clauses*, and *formulas*. By substituting precise atoms with fuzzy atoms, we can construct fuzzy propositions where each logical constituent has a semantic interpretation in form of a fuzzy set. Modifications can be done on the syntactic level, e.g. by adding a new imprecise fact or vague information with a logical "and" operation, which leads to a new semantic interpretation in form of a fuzzy set. The basic unit in this model is a *fuzzy atom*, which formalizes the notion of imprecise concepts introduced above:

DEFINITION 1 (FUZZY CONCEPT VOCABULARY, FUZZY ATOM) *Let a domain Ω be given. A fuzzy concept vocabulary over Ω is a triple $(\mathfrak{A}(\Omega), \mu., \mathscr{A}.)$, where $\mathfrak{A}(\Omega)$ is a countable set of symbols and $\mu., \mathscr{A}.$ are mappings*

$$\begin{aligned} \mu. &: \mathfrak{A}(\Omega) \to F(\Omega), \mathscr{A} \mapsto \mu_{\mathscr{A}} \\ \mathscr{A}. &: F(\Omega) \to \mathfrak{A}(\Omega), \mu \mapsto \mathscr{A}_\mu \end{aligned}$$

satisfying $\mu. \circ \mathscr{A}. = \mathrm{id}_{F(\Omega)}$. The elements of $\mathfrak{A}(\Omega)$ are called fuzzy atomic concepts (or fuzzy atoms for short) over Ω. The fuzzy set $\mu_{\mathscr{A}}$ is called the interpretation of $\mathscr{A}$ and the atom $\mathscr{A}_\mu$ is called the (distinguished) name of μ.

Example (imperfect information) We construct an example for a fuzzy clause. Here, three fuzzy atoms represent different imperfect information for the type of a variable, obtained from different sources (e.g., source code analysis, reverse engineer, documentation). Each fuzzy atom indicates how certain a given type can be attributed to the variable *i*. The membership degree is interpreted in a possibilistic fashion: a value of 0.0 *("impossible")* indicates that the variable cannot be of this type, a value of 1.0 *("certain")* means that none of the available information opposes the variable from having this type (*not* that it must be of this type!), and values in between indicate varying degrees of compatibility of the variable with the type. The fuzzy clause $\mathscr{K}$ formed from these atoms represents the combined knowledge about the variable *i* within an RE repository:

$\mu_{\mathscr{A}_1}$: *type of i according to reverse engineer* =
$\{\mathsf{int}/0.8, \mathsf{string}/0.1, \mathsf{char}/0.1, \mathsf{bool}/0, \mathsf{enum}/0.0\}$
$\mu_{\mathscr{A}_2}$: *type of i from source code analysis* =
$\{\mathsf{int}/0.2, \mathsf{string}/0.0, \mathsf{char}/1, \mathsf{bool}/0, \mathsf{enum}/0.1\}$
$\mu_{\mathscr{A}_3}$: *type of i according to documentation* =
$\{\mathsf{int}/0.5, \mathsf{string}/0.0, \mathsf{char}/0.5, \mathsf{bool}/0, \mathsf{enum}/0.5\}$
$\mu_{\mathscr{K}} = \mu_{\{\mathscr{A}_1,\mathscr{A}_2,\mathscr{A}_3\}} = \mu_{\mathscr{A}_1 \vee \mathscr{A}_2 \vee \mathscr{A}_3}$: *type of i (combined)* =
$\{\mathsf{int}/0.8, \mathsf{string}/0.1, \mathsf{char}/1, \mathsf{bool}/0, \mathsf{enum}/0.5\}$

Similarly, different fuzzy clauses can be combined to a *fuzzy formula* by computing the logical *and* operation on the individual fuzzy clauses.[1] Thus, all imperfect information regarding an artefact is stored in fuzzy conjunctive normal form (FCNF), which helps to simplify operations.

The language of fuzzy atoms, fuzzy clauses, and fuzzy formulas permits the aggregation of arbitrary pieces of imperfect information, however inconsistent they may be. This is an important feature, because an RE system does not have

1 We use the standard functions for possibilistic fuzzy sets: *min* for intersection, *max* for union, and $1 - \mu$ for computing the complement.

to reject inconsistent information out of hand (as is the case with conventional approaches), but can rather attempt to restore consistency by giving preference to some pieces over others. This will be discussed later on in more detail when we examine how imperfect information are processed.

3.2. Enhancements for RE repositories

This model has been enhanced to capture the semantics needed for building fuzzy reverse engineering repositories. These enhancements are based on the observation that the various imperfect states representing the obtained knowledge about a system cannot be changed independently, but are rather tightly coupled. Consider for example information obtained about the *type* of a variable, as illustrated in the example above. If the knowledge about the variable changes, e.g. because an additional source provides more information, this change influences other information collected in junction with the analyzed variable, like its *usage*. It would be possible to represent these interactions implicitly by programming them into a fuzzy reverse engineering system. However, their frequent appearance suggests an underlying phenomenon, which we decided to model explicitly by enhancing the fuzzy representation model discussed above.We call these interactions *dependencies*, they are modeled by introducing a *dependency graph* into the model:[2]

DEFINITION 2 (DEPENDENCY GRAPH) *A (fuzzy) dependency graph is a directed, acyclic graph* $\mathfrak{G} = (V, E, \xi)$ *with the set of vertices* V*, the set of directed edges* E *and a function* ξ *for associating every vertex* $v \in V$ *with a fuzzy formula* $\mathscr{F} \in \mathfrak{F}$ *(*$\mathfrak{F}$ *being the set of all fuzzy formulas):* $\xi : V \rightarrow \mathfrak{F}$*. Dependencies are transitive and acyclic.*

Note that the dependency graph, by itself, only captures the structure of dependencies between different types of information within an RE repository. The automatic *update* of dependent information is handled by special fuzzy operators that percolate changes through the graph. We will discuss these in the next section.

However, computations across these dependencies become complicated by the fact that different fuzzy formulas, attached to nodes in the graph, can be defined on different domains (e.g., information regarding a variable's *type* vs. its *usage*). We therefore have to introduce *transformation functions* Θ for mapping a concrete fuzzy set from one domain Ω (e.g., the type) to another domain Ω' (e.g., the usage): $\Theta_{\Omega,\Omega'} : F(\Omega) \rightarrow F(\Omega')$. These functions are highly application specific (we give an example below) and therefore have to be defined in a separate data dictionary.

2 Note that Jahnke and Walenstein also state the requirement that *dependent beliefs* must be modeled explicitly and processed automatically when updating the knowledge base [10].

Example (dependency graph and transformation function) In our example we have two domains, representing the *type* and the *usage* of a variable, respectively. Hence, $\Omega_1 = \text{type} = \{\textit{int, string, char, bool, enum}\}$ and $\Omega_2 = \text{usage} = \{\textit{control, output, index, temp, input}\}$. The corresponding dependency graph $\mathfrak{G} = (V, E, \xi)$ is quite simple, since it contains only two vertices connected by a single directed edge:

$$V = \{v_1, v_2\}, E = \{(v_1, v_2)\}, \xi = \{[v_1, \mathscr{F}_1], [v_2, \mathscr{F}_2]\}$$

A possible transformation function $\Theta_{\text{type,usage}}$ that converts a fuzzy set μ_1 from Ω_1 to Ω_2 is defined by:

$$\Theta_{\text{type,usage}}(\mu_1)(\omega_2) = \begin{cases} \mu_1(\text{int}) & \textbf{if } \omega_2 = \text{index} \\ \mu_1(\text{char}) & \textbf{if } \omega_2 = \text{temp} \\ \mu_1(\text{bool}) & \textbf{if } \omega_2 = \text{control} \\ \mu_1(\text{string}) & \textbf{else.} \end{cases}$$

A concrete example for the transformation of a fuzzy set $\mu_1 \in F(\textit{type})$ to a fuzzy set $\mu_2 \in F(\textit{usage})$ is given by:

$\mu_{\mathscr{A}} = \mu_{\xi(v_1)} : \textit{type of } i =$
$\{\text{int}/0.5,\ \text{string}/0,\ \text{char}/1,\ \text{bool}/0.15,\ \text{enum}/0.15\}$
$\mu_{\Theta_{\text{type,usage}}(\mathscr{A})} = \mu_{\xi(v_2)} : \textit{usage of } i$
$\{\text{control}/0.15,\ \text{output}/0,\ \text{index}/0.5,\ \text{temp}/1,\ \text{input}/0\}$

3.3. Processing imperfect knowledge

The challenge we address next is the *modification* of imperfect information stored in our fuzzy repository. Modifications within the RE domain are not restricted to simple updates (in the sense of Katsuno and Mendelzon [11]), but rather revisions, since we collect more and more information about an already existing world (i.e., the legacy system). Thus, new information concerning an artefact cannot simply replace all existing information — we rather want to combine it with the knowledge obtained so far.

As fuzzy clauses and formulas are sets, modifications could be done by simply adding or removing fuzzy literals or clauses. Simple set operators, however, are not concerned with semantics — clauses added to a fuzzy formula in this way can easily lead to inconsistencies in the fuzzy set interpretation. But maintaining consistency was one of our prime requirements for an RE repository: information coming from different sources must be combined in such a way that minor inconsistencies (within a specified degree) can be tolerated, but larger contradictions must be resolved in an automatic, consistency-preserving way.

Our solution to this problem is to define fuzzy operators based on Gärdenfors-style (AGM) belief revision [2, 4, 5]. These operators are called γ-expansion, γ-revision, and γ-contraction, they allow to modify a complex fuzzy formula while maintaining a specified degree of consistency γ [23]. In the case of γ-revision for example, a new piece of information is either rejected, merged with all of the existing

information, or added while removing some fuzzy clauses that are inconsistent to the new fuzzy formula within a specified degree γ. If existing information have to be removed in order to restore consistency, only the minimally required changes are performed. These operators have the following semantics (for details and their formal definition please refer to [23]):

Expansion: A *γ-expansion* adds new information (expressed as a fuzzy formula) to an existing set of information (again a fuzzy formula), without removing any existing clauses while ensuring that the resulting fuzzy formula reaches a consistency degree of at least γ. If this is not possible, the new information is *rejected*.

Revision: The *γ-revision* operation always adds new information, if its consistency degree reaches at least γ. Existing information (fuzzy clauses) may be removed in case there is a partial or complete inconsistency.

Contraction: The *γ-contraction* operation removes information from a knowledge set, in a way that preserves as much of the existing information as possible while ensuring that the result's consistency degree reaches at least γ.

Additionally, these operators had to be extended for the management of dependent information in dependency graphs as defined above [22]. The semantics of the operations remain unchanged, but additionally each change of a fuzzy formula now checks for dependencies in the graph, applies transformation functions if necessary, and then computes the operation on all dependent fuzzy formulas, until no more edges lead out of a given node.

4. Fuzzy reverse engineering

In this section we show how the concepts of the fuzzy model, the processing operators, and the RE repository introduced above can be combined to a new fuzzy RE framework. We first show how the fuzzy representation and processing model can be integrated into an object-oriented repository framework. This extension allows the cooperation and coexistence of both classical and fuzzy knowledge within a single repository, as well as the RE tools operating on it. Next, we illustrate our concepts on a concrete example that shows how our fuzzy model and fuzzy repository interact within the RE process. Finally, we state first requirements for fuzzy RE tools as a basis for future work.

4.1. The fuzzy extended TAO framework model

The first step is the fusion of the fuzzy representation model and the RE repository model. As an example, we demonstrate this step by blending our TAO framework and the fuzzy model. Since the TAO repository is built on object-oriented technology, we show how an OO model can be enhanced to handle fuzzy information (but the same extension principle can be equally applied to other data models).

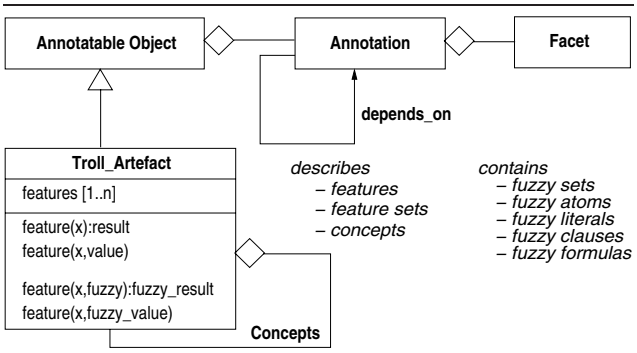

Figure 3. The generic TAO fuzzy model

The main design idea is to add imperfect information as an orthogonal extension through a frame-like concept, which we call *annotations*. Each object (or part of an object, like an attribute) holding RE information and meta-information can thus be *annotated* with container objects referring to one or multiple meta-information objects, called *facets*. Within our model, facets can store fuzzy information in form of *fuzzy components*, like fuzzy sets, fuzzy literals, fuzzy clauses, or fuzzy formulas.

This approach allows for a significant simplification of the TAO framework, since it is no longer necessary to split knowledge-keeping objects into FRAME and SHADOW objects. Rather than maintaining certain (crisp) data seperately from imperfect (fuzzy) information in two disparate knowledge hierarchies, the annotation model offers the possibility to fuzzify individually all parts of TAO artefacts — concepts, features, and feature sets. As shown in Figure 3, every TAO artefact inherits from the generic meta-object ANNOTATABLE OBJECT. Through inheritance all properties and operations of the fuzzy model now become available within the TAO framework.

As an additional side-effect, both reverse engineers and automatic tools benefit from this new framework because access and storage of different types of knowledge become now seamless and adaptable.[3]

The annotatable object provides the main entry point to the fuzzy knowledge. It is defined as an aggregation of annotation objects. Each ANNOTATION is defined as a specific aspect of an artefact that is fuzzified. The information gained about this specific aspect (may it be vague or precise) is then represented in a set of FACET objects, together they form the available domain knowledge. The interdependency between specific information that is supported by the fuzzy dependency graph is modeled as a reflexive association of the annotation object. Each of these objects defines a node, the association set represents the edges within instantiated

3 Note that the same annotation model can simultaniously hold additional types of meta-information for an artefact by adding a facet of a different type, e.g., statistical data, pictures, or speech.

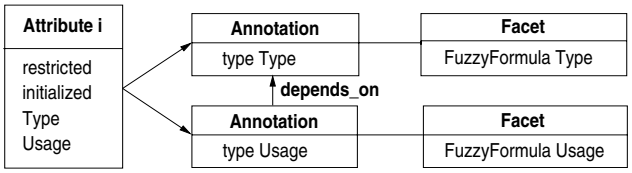

Figure 4. Fuzzy instance model of variable *i*

fuzzy dependency graphs. The fuzzy operations defined on FUZZYCOMPONENT objects and dependency graphs may be used to output a fuzzy or de-fuzzified strict result. This offers the possibility to deal both with precise and fuzzy information during RE when using automatic tools.

In Figure 4 the instance of a fuzzy artefact ATTRIBUTE is shown. The attribute object is annotated with two annotation objects describing the TYPE and USAGE features. One annotation keeps the RE knowledge about the type, the other annotation stores the RE knowledge regarding the usage in form of fuzzy formulas. Their fuzzy set interpretations are equivalent to the ones shown in the transformation example before. During the RE process the information collected about usage and type are added as additional facets. The process of adding information modifies the result with respect to the system's model being reengineered. When asking for e.g. the type of artefact *i* the answer evolves over time, changing in a non-monotonic way as new evidence becomes available. At each point in time the RE system supports a certain interpretation of a system, similar to classical, crisp repositories. But we are now able to give an explicit evaluation of each result, showing its certainty and internal degree of consistency through the fuzzy set interpretation. In the example below, we show how the interpretation for *i* changes from the inital assumption of a *char* variable to an index for the days of the week coded from 1–7 plus 0 as default. But all information gathered is kept and still evaluated as long as the RE process continues. This is a significant advantage of the fuzzy extended domain model and helps to perform RE tasks more efficiently and successfully.

4.2. Fuzzy reverse engineering process example

In this section we give a complete example that illustrates how the concepts of the fuzzy RE repository, the fuzzy representation model, and the fuzzy operators interact. Here we go back to the analysis of a program variable *i*, examining its implicit (and uncertain) type and usage information.

To begin, we assume a certain repository state representing information collected about variable *i* so far (collected e.g. through source code analysis tools or information given by the software reengineer). A fuzzy formula $\mathscr{F}_1$ holds the available information about the variable's type, represented by a single fuzzy clause $\mathscr{K}_1$, which has the following fuzzy set interpretation:

$$\mu_{\mathscr{K}_1} = \{\mathsf{int}/0.9,\ \mathsf{string}/0.5,\ \mathsf{char}/0.75,\ \mathsf{bool}/0,\ \mathsf{enum}/.15\}$$

As can be seen from the fuzzy interpretation, there is a high certainty for the variable being of type *int* or *char*, while types *bool* or *enum* are rather unlikely. Moreover, we have some information about the usage of *i*; here, two pieces of information have been collected into a fuzzy formula, represented by two fuzzy clauses $\mathscr{F}_2 = \{\mathscr{K}_2, \mathscr{K}_3\}$:

$$\mu_{\mathscr{K}_2} : \textit{usage of } i = \{\mathsf{control}/0.8,\ \mathsf{output}/0.5,\ \mathsf{index}/0.2,\ \mathsf{temp}/1.00,\ \mathsf{input}/0.1\}$$
$$\mu_{\mathscr{K}_3} : \textit{usage of } i = \{\mathsf{control}/0.2,\ \mathsf{output}/0.5,\ \mathsf{index}/1.0,\ \mathsf{temp}/0.75,\ \mathsf{input}/0.1\}$$
$$\mu_{\mathscr{F}_2} : \textit{usage of } i = \mu_{\{\mathscr{K}_2,\mathscr{K}_3\}} = \mu_{\mathscr{K}_2 \wedge \mathscr{K}_3} = \{\mathsf{control}/0.2,\ \mathsf{output}/0.5,\ \mathsf{index}/0.2,\ \mathsf{temp}/0.75,\ \mathsf{input}/0.1\}$$

The first source clearly assumes *i* is a *temp* variable used for computations, while the second source favours the interpretation that *i* is an *index* variable, though it also indicates a high certainty for it being a *temp* variable. When combined, these two pieces of information result in the fuzzy interpretation $\mu_{\mathscr{F}_2}$ shown at the bottom. Note that even though the certainty of *i* being an *index* variable has dropped significantly because of the contradicting information, the original clause $\mathscr{K}_3$ is still stored in the RE repository and will be used when revising the information under new evidence, as we will show below.

During the RE process, we receive further information about the type of our variable *i* (e.g., from a data analysis, or the program's original documentation); in this example represented by a fuzzy formula $\mathscr{F}_3 = \{\mathscr{K}_4\}$ and its transformation $(\mathscr{F}_3)^{\theta}_{v_2}$ onto node v_2 (usage):

$$\mu_{\mathscr{F}_3} = \mu_{\{\mathscr{K}_4\}} : \textit{type of } i = \{\mathsf{int}/0.9,\ \mathsf{string}/0.5,\ \mathsf{char}/0.1,\ \mathsf{bool}/0,\ \mathsf{enum}/0.1\}$$
$$\mu_{(\mathscr{F}_3)^{\theta}_{v_2}} = \mu_{\{\mathscr{K}_5\}} : \textit{usage of } i = \{\mathsf{control}/0,\ \mathsf{output}/0.5,\ \mathsf{index}/0.9,\ \mathsf{temp}/0.1,\ \mathsf{input}/0.5\}$$

This information gives a very high certainty that variable *i* is indeed of type *int* and therefore used as an *index* variable. However, this information is now partly inconsistent with the information in the RE repository, which currently favours the *temp* interpretation. Hence, we cannot simply add it without receiving a contradictory and therefore useless result. Indeed, the attempt to add this piece of information with the graph expansion operator introduced above would result in a rejection for any consistency degree > 0.5. The solution is to *revise* the RE repository through non-monotonic consistency-maintaining fuzzy belief revision. Thus, we attempt a graph revision, requesting a consistency degree of at least 0.6: $\mathfrak{G}' = \langle \mathfrak{G}, v_1 \rangle \oplus^{\mathfrak{G}}_{0.6} \mathscr{F}_3$. That is, we start revising the belief network stored in the repository at the node v_1, which is associated with the *type* information about

variable i. The effect of this operation in this example is twofold: Firstly, the fuzzy formula for the type itself is being revised, and secondly the new information is percolated through the graph, revising all dependent information on the path — in this example, the *usage* information attached to node v_2:

$$\xi'(v_1) = \mathscr{F}_4 = \xi(v_1) \oplus_{0.6} \mathscr{F}_3,$$
$$\xi'(v_2) = \mathscr{F}_5 = \xi(v_2) \oplus_{0.6} (\mathscr{F}_3)^{\theta}_{v_2}$$

The fuzzy formula $(\mathscr{F}_3)^{\theta}_{v_2}$ at vertex v_2 is also shown above. The transformation $(\mathscr{F}_3)^{\theta}_{v_2} = \Theta^{\mathscr{F}}_{\text{type,usage}}(\mathscr{F}_3)$ uses the same transformation function as in the example shown before.

This operation results in two revised fuzzy formulas, which have the following fuzzy set interpretations:

$\mu_{\mathscr{F}_4} = \mu\{\mathscr{K}_1, \mathscr{K}_4\} : \textit{type of } i =$
$\{\text{int}/0.9,\ \text{string}/0.5,\ \text{char}/0.1,\ \text{bool}/0,\ \text{enum}/.1\}$
$\mu_{\mathscr{F}_5} = \mu\{\mathscr{K}_3, \mathscr{K}_5\} : \textit{usage of } i =$
$\{\text{control}/0,\ \text{output}/0.5,\ \text{index}/0.9,\ \text{temp}/0.1,\ \text{input}/.1\}$

Thus, the fuzzy clause $\mathscr{K}_2$ has been removed by the graph revision operator from the fuzzy formula representing the usage of i. Note that the operator succeeded in restoring a consistency of at least 0.6 as requested (actually 0.9 for both type and usage), minimally changing the information along the way. As a consequence, the degree of certainty for the usage as an *index* variable has increased again.

In a real-world scenario, there will be thousands of artefacts that have to be examined, and possibly hundreds of information from many different sources. Note that the graph operators proposed here provide a powerful and flexible tool for maintaining a reverse engineering repository throughout the RE process, since they admit inconsistency upto certain prescribed degrees. Hence, new information that are collected throughout the RE process do not have to be rejected because of small inconsistencies, but can be stored in the repository. This is important, since the whole RE process now has more information available, stored in a formal and automatically evaluable representation.

Moreover, the internal consistency of the repository can be adjusted over time since the fuzzy graph operators allow for changing γ values: In the beginning of an RE process, where there is a high degree of uncertainty and internal inconsistency, this value can be set quite low (perhaps around 20%), allowing to store many beliefs that are inconsistent to a large degree, but are nevertheless all potentially correct and important pieces of information. Later in the process, the required consistency degree can be increased continuously, thereby taking all available information into account and finally converging to a consistent result.

5. Related work

The research areas of RE, program comprehension, and software maintenance deal with the problem of eliciting and processing imperfect knowledge. Von Mayrhauser, Vans, and Howe carried out extensive studies to examine the nature and process of program understanding, offering a precise model of the iterative and non-monotonic knowledge elicitation process of software understanding [20], [21].

Quilici and Chin [19] developed the tool DECODE, which offers an approach to externalize not only certain knowledge but also the hypothetic knowledge of *not* knowing a specific fact. DECODE already combines automatic and human-centered RE steps into a single repository. But its approach is limited insofar as the transition of knowledge states (from certain to uncertain to not-existing) is unsupported.

The need for handling imperfect information during RE processes is also stated by Jahnke and Walenstein [10]. The solution offered in that paper handles uncertain information within a single reverse engineering step, thereby resolving all imperfections within the VARLET tool [9]. They then come up with a precise result, effectively erasing all vaguenesses and contradictory interpretations of the gained knowledge. Contrary to this approach we provide a fuzzy meta model that enables *all* conceptual models of RE tools and repositories to deal explicitly with uncertainty, vagueness, and the integration of conflicting or inconsistent information sources or results throughout the whole RE process.

We support and extend the CORUM II approach by Kazman et al. [12] in the sense that we offer additional possibilities to mitigate the problems arising from the integration of different information sources and abstraction levels. Also Jahnke et al. [8] consider the problem of how to associate different views and perspectives in software maintenance. Even though the chosen approach is different, we are convinced that a fuzzy extension can offer support and benefit since it is necessary to integrate and consolidate knowledge from different sources concernig a single artefact in a homogenous way. By applying the fuzzy approach and the orthogonal extension of conceptual models, integration of RE information can be separated from the problem of consistency handling. All knowledge is represented equally throghout the integration process and can be processed automatically, even if a specific item is (temporarily or permanently) inconsistent. This is a significant advantage given the iterative and diverse nature of reverse engineering. Moreover, the automatic consistency handling through nonmonotonic revision operators is a major benefit in knowledge integration tasks that concern large amounts of data to be scanned and rated, with a high number of inconsistencies that have to be resolved.

Repository-based engineering tools and frameworks like COREM [14], Varlet [9], Fujaba [1], or commercial CASE tools like Rational Rose and Together already offer some kind of RE functionality. But as reported by Kollmann et al. [15] these tools offer a rather limited and idiosyn-

cratic approach to the requirements of reverse engineering. They are only able to include information provided by the source code (and even this information source can not be used to its fullest extent). These tools offer more or less a transformation from source code to class hierachies and sequence/collaboration diagrams. Such features are of course very helpful, but they do not meet the needs to integrate different kinds of imperfect information. Also, most models are closed in the sense that they do not allow for an extension of the conceptual reverse engineering model. Especially with regard to an open and flexible repository domain model our proposed fuzzy extension of the underlying metamodel offers a significant improvement.

6. Conclusions and further work

In this paper we developed a fuzzy extension that can be included within conceptual models of reverse engineering repositories in order to improve their abilities to store and process imperfect knowledge.

The features of this extension are *(i)* the formal definition of the term of imperfection (vagueness and uncertainty) by introducing the concept of fuzzy formulas, *(ii)* the uniform representation of all (precise and imperfect) knowledge concerning a specific reverse engineering artefact, and *(iii)* the automatic handling of inconsistencies within specific instances through fuzzy belief revision operators.

We show exemplary how to perform the fuzzy extension by applying our model to the TAO framework. The proposed annotation approach allows for an orthogonal extension of existing conventional models, without requiring expensive re-implementations. Available resources and tools can be incrementally enhanced to benefit from the semantically richer fuzzy model, gaining the ability to integrate different RE knowledge sources within a single, automatically evaluable representation.

A formal representation model capable of handling imperfect information also offers new perspectives for other research areas in RE. We envision the definition and development of algorithms, heuristics, tools, and methods that intrinsically use fuzzy representations and operations in order to provide results that are better suited to deal with the imperfect nature of reverse engineering.

References

[1] Fujaba homepage. `http://www.fujaba.de`.

[2] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 2(50):510–530, June 1985.

[3] E. Cox. *The Fuzzy Systems Handbook*. AP Professional, 2nd edition, 1999.

[4] P. Gärdenfors. *Knowledge in Flux*. MIT Press, 1988.

[5] P. Gärdenfors, editor. *Belief Revision*. Cambridge University Press, 1992.

[6] T. Hartmann, G. Saake, R. Jungclaus, R. Hartel, and J. Kusch. Revised version of the modeling language TROLL — Troll Version 2.0. Informatik-Berichte 94/03, Technische Universität Braunschweig, Abt. Datenbanken, April 1994.

[7] IEEE. IEEE Technical Council on Software Engineering. `http://www.tcse.org/revengr`.

[8] J. Jahnke, H. Müller, N. Mansurov, and K. Wong. Fused Data-Centric Visualizations for Software Evolution Environments. In *Proc. of the 10th IWPC*, pages 20–24, Paris, France, June 27–29 2002. IEEE Computer Society.

[9] J. Jahnke and J. Wadsack. Varlet: Human-Centered Tool Support for Database Reengineering. In *Workshop on Software Reengineering*, Bad Honnef, Germany, 1999. University Koblenz-Landau.

[10] J. H. Jahnke and A. Walenstein. Reverse Engineering Tools as Media for Imperfect Knowledge. In *Proc. of the 7th WCRE*, pages 22–31. IEEE Computer Society Press, 2000.

[11] H. Katsuno and A. O. Mendelzon. On the Difference between Updating a Knowledge Base and Revising it. In *KR*, pages 387–394. Cambridge, MA, April 22–25 1991.

[12] R. Kazman, S. G. Woods, and S. J. Carrierre. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In *Proc. of the 5th WCRE*, pages 154–163. IEEE Computer Society Press, 1998.

[13] G. J. Klir and T. A. Folger. *Fuzzy Sets, Uncertainty, and Information*. Prentice-Hall, 1988.

[14] R. Klösch and H. Gall. *Objektorientiertes Reverse Engineering*. Springer Verlag, 1995.

[15] R. Kollmann, P. Selonen, E. Stroulia, T. Systä, and A. Zündorf. A Study on the Current State of the Art in Tool Supported UML-Based Static Reverse Engineering. In *Proc. of the 9th WCRE*. IEEE Computer Society, 2002.

[16] U. Kölsch and J. Laschewski. Objectifying Legacy Application Systems Using a Specification Language Framework. In *Proc. of the ICSE'97 Workshop*, Boston, MA, USA, 1997.

[17] U. Kölsch and J. Laschewski. A Framework for Object-oriented Reverse Engineering of Legacy Information Systems. *International Journal of Software Engineering and Knowledge Engineering*, 9(1):27–54, 1999.

[18] U. Kölsch and M. Wallrath. A Process Model for Controlling and Performing Re-engineering Tasks. In *Proceedings of the first CSMR*, pages 20–24, Berlin, Germany, March 17–19 1997. Euromicro, IEEE Computer Society Press.

[19] A. Quilici and D. N. Chin. DECODE: A Cooperative Environment for Reverse-Engineering Legacy Software. In *Proc. of the 2nd WCRE*. IEEE Computer Society Press, 1995.

[20] A. von Mayrhauser and A. Vans. Program Understanding: Models and Experiments. In M. Yovits and M. Zelkowitz, editors, *Advances in Computers*, volume 40, pages 1–38, 1995.

[21] A. von Mayrhauser, A. Vans, and A. Howe. Program Understanding Behaviour During Enhancement of Large-Scale Software. In *Journal of Software Maintenance – Research and Practice*, volume 9, pages 299–327, 1997.

[22] R. Witte. *Architektur von Fuzzy-Informationssystemen*. BoD, Norderstedt, Germany, 2002. ISBN 3-8311-4149-5.

[23] R. Witte. Fuzzy Belief Revision. In *9th Intl. Workshop on Non-Monotonic Reasoning*, pages 311–320, Toulouse, France, April 19–21 2002. `http://rene-witte.net`.