A NEW APPROACH TO EXPERT SYSTEM EXPLANATIONS

Regina Barzilay[†], Daryl McCullough[*], Owen Rambow[*]
Jonathan DeCristofaro[‡], Tanya Korelsky[*], Benoit Lavoie[*]

[*] CoGenTex. Inc.
[†] Department of Computer Science, Columbia University
[‡] Department of Computer Science, University of Delaware
Contact: owen@cogentex.com

1 Expert System Explanation

Expert systems were one of the first applications to emerge from initial research in artificial intelligence, and the explanation of expert system reasoning was one of the first applications of natural language generation.[1] This is because the need for explanations is obvious, and generation from a knowledge-based application such as reasoning should be relatively straightforward. However, while explanation has been universally acknowledged as a desirable functionality in expert systems, natural language generation has not taken a central place in contemporary expert system development. For example, a popular text book about expert systems such as (Giarratano and Riley, 1994) stresses twice in the introduction the importance of explanation, but provides no further mention of explanation in the remaining 600 pages. (The book is based on the popular CLIPS framework.) In this paper, we present a new approach to providing an expert system with an explanation facility. The approach comprises both software components and a methodology for assembling the components. The methodology is minimally intrusive into existing expert system development practice.

This paper is structured as follows. In Section 2, we discuss previous work and identify shortcomings. We present our analysis of knowledge types in Section 3. In Section 4 present the Securioty Assistant and its explanation facility. Finally, we sketch a general methodology for explainable expert system engineering in Section 5.

[1] The work reported in this paper was carried out while all authors were at CoGenTex, Inc., and is in part supported by contract F30602-96-C-0076 awarded by the Information Directorate of the Air Force Research Laboratory at the Rome Research Site. We would like to thank Rob Flo, project engineer, for his support and feedback. We would also like to thank Joe McEnerney for help in integrating the explanation facility with the SA, and Mike White and two anonymous reviewers for useful comments.

2 Previous Work

A very important early result (based on experiences with explanation[2] in systems such as MYCIN (Shortliffe, 1976)) was the finding that "reasoning strategies employed by programs do not form a good basis for understandable explanations" (Moore, 1994, p.31). Specifically, simply paraphrasing the chain of reasoning of the expert system does not let a human user easily understand that reasoning.

Two separate approaches have been proposed to address this problem:

- In the **Explainable Expert System** (EES) approach (Swartout et al., 1991; Swartout and Moore, 1993), the knowledge representation used by the expert system is enriched to include explicit "strategic" knowledge, i.e., knowledge about how to reason, and domain-specific knowledge. From this knowledge, the rules used by the expert system are compiled, and this knowledge is also used to provide more abstract explanations of the system's reasoning.
- In the **Reconstructive Explainer** (Rex) approach (Wick, 1993), the expert system is unchanged, but after it has performed its reasoning, a causal chain for explanation is constructed from the input data to the conclusion reached previously by the expert system as a separate process. The work of (Tanner et al., 1993) can also be seen as falling in this paradigm, since a separate representation of knowledge (the "functional representation") is used only for explanation, and the explanation must be specially derived from this.

These approaches have in common a preoccupation with a categorization of knowledge used in the system into different types. The Explainable Expert System concentrates on an abstract representation of strategic knowledge (how does a particular action of the system relate to the overall goal?) and on the representation of design rationale (why are actions reasonable in view of domain goals?). In addition, there is terminological domain knowledge (definitions of terms). The Reconstructive Explainer and related approaches have a representation of domain knowledge, along with domain rule knowledge (mainly, causality), which is completely separate from that used by the expert system itself. This knowledge is used to derive an "explanation path" through the domain knowledge representation.

There are problems with both approaches. EES has not proven to be a fully satisfactory solution to the problem of expert system explanation. The problem is that the writers of expert systems have not been too quick or too eager to adopt frameworks such as EES. The requirement for a more abstract representation of knowledge (from which the actual expert system rules are compiled) that EES imposes may be considered onerous by the expert system developer, appearing unmotivated from the point of view of the core functionality of the system, namely reasoning (as opposed to explanation). Presumably, it is difficult for one and the same person to be a domain expert and a expert on communication in the domain.

[2]We do not consider explanation generation from data bases (for example, (McKeown, 1985; Paris, 1988; Lester and Porter, 1997)) to be the same problem as expert system reasoning explanation (even though we may use some similar techniques). In data base explanations, the knowledge is static and its representation is given *a priori* as part of the problem statement. In expert system explanations, the knowledge to be explained is generated dynamically, and the proper representation for this knowledge is part of the solution to the problem of expert system explanation, not its statement.

In the Rex approach, the obvious problem is that in order to generate an explanation, additional reasoning must be performed which in some sense is very similar to that done by the expert system itself (e.g., finding causal chains). This is redundant, and does not result in a clear separation between reasoning and explanation. While Wick (1993) argues against such a separation on philosophical grounds, practical constraints suggest, as indicated above, that the domain expert responsible for implementing the reasoning system should not also be responsible for implementing the explanation capability, and that the communication engineer (responsible for implementing the explanation facility) should not need to replicate domain reasoning.

In this paper, we present a new approach (system and methodology) to expert system explanation which does not require the expert system writer to take into account the needs of the explanation while writing the rules. At the same time, we avoid the necessity of having a separate domain reasoning component for the explanation generation. Instead, the expert system is largely considered a stand-alone application, onto which explanation is added. However, this is done by having a communication expert design a second knowledge representation (separate from the expert system's domain knowledge representation) specifically for the purpose of communicating explanations. This representation is populated by the expert system as it reasons, not post-hoc. Thus, no separate reasoning facility is needed.

3 Types of Knowledge in Explanation

We follow previous work in distinguishing different types of knowledge. However, we classify knowledge by what it is used for and who is responsible for its engineering, not by its structure or contents. Specifically, we distinguish three types of knowledge:

- **Reasoning domain knowledge** (RDK). This is the domain knowledge encoded by the domain expert in the expert system proper. Typically, it includes terminological knowledge, instance knowledge, and rules.
- **Communication domain knowledge** (CDK). This is knowldge about the domain which is needed for communication about the domain.
- **Domain communication knowledge** (DCK). This is knowledge about how to communicate in the domain.

The distinctions may at first seem overly fine-grained. However, each type of knowledge is distinguished from the other types. CDK is domain knowledge, but it is only domain knowledge that is needed for communication, not for reasoning. RDK and CDK of course overlap, but they are not identical. This is in fact the lesson from much previous work in expert system explanation, for example the work of Paris et al. (1988) contrasting "the line of reasoning" and "the line of explanation", and the claim of Swartout et al. (1991) that the domain representation must be augmented with additional knowledge about the domain and about reasoning in the domain.

CDK is different from DCK in that CDK is *knowledge about the domain as it is needed for communication*, but DCK is *knowledge about how to communicate in that domain*, and in a specific communicative setting (characterized by factors as diverse as communication type or genre, hearer

needs, communication medium, or cultural context).[3] DCK is not knowledge about the domain, but about texts (or whatever the communicative medium is). For example, it may be expressed in communicative plan operators which achieve goals related to the hearer's cognitive state, while domain knowledge would never include plan operators related to the hearer's cognitive state because the hearer is not part of the domain.

CDK is not a new concept. Many researchers have identified the need for packaging domain knowledge differently for communication. For example, the "views" of Lester and Porter (1997) can be seen as a form of CDK, though they are not a declarative representation. What is new in our work, however, is the proposal that CDK should be represented explicitly in a distinct representation from the domain knowledge. At CoGenTex, we have used an "Intermediate Knowledge Representation Scheme" (IKRS) for representing CDK in several applications (Korelsky et al., 1993). The IKRS, like the CDK used for explanation generation, is a declarative representation, and it was motivated mainly from methodological and practical software engineering considerations.[4]

4 The Security Assistant

The Security Assistant or SA (Webber et al., 1998) is part of the DesignExpert tool (Ehrhart et al., 1998), which helps software engineers analyze system-wide (or "non-functional") requirements such as security, fault-tolerance, and human-computer interaction. The SA aids a software engineer in choosing security measures to protect valuable system assets (e.g. important data) against likely threats (e.g. disclosure or corruption). In the following three subsections, we discuss how the three types of knowledge discussed in the previous section – RDK, CDK, and DCK, are represented and used in the SA.

4.1 The Expert System: Reasoning Domain Knowledge

The SA first queries the user for information about entities of the system to be analyzed, such as system assets, system components, and system sites, and the damage types that are of concern for these entities. Additional damage types are inferred for each important asset of a system (e.g. data can suffer from disclosure or corruption). The system then searches for defenses against these damage types. If none can be found, the SA identifies attack methods which can cause the damage,

[3] While DCK is domain- and genre-specific knowledge about how to communicate, we do not claim that the same type of reasoner with different domains (say, a expert system for car repair and a expert system for helicopter repair) would necessarily require different DCK. However, the type of expert system in the two cases might be very similar, and it is this fact that would allow us to re-use the same DCK. Thus, from the point of view of the explanation system, the "domain" is not the domain of the expert system, but the expert system itself. For a discussion of the distinction between domain communication knowledge and domain-independent communication knowledge, and for an argument in favor of the need for DCK, see (Kittredge et al., 1991).

[4] While CDK is closely related to *content selection*, it should not be reduced equated with content selection, which is often seen as the first task in text planning (followed by content ordering). Content selection is entirely oriented towards the anticipated act of communication, and hence defined by its parameters: what the communicative goal is, what medium, who the hearer is, and other constraints (length of communication, and so on). CDK is a representation of knowledge needed for content selection, but excludes all choices that depend on knowledge of the intended act of communication. For example, CDK might include relative salience between domain objects, but does not include information about how salient an object needs to be in order to interest the hearer. However, we admit that the distinction may be less than cracklingly crisp, especially in implementations.

and identifies enabling conditions for such attacks. It then attempts to find defenses that prevent such situations. This reasoning can then be iterated. The result of the SA's reasoning is a list of recommended defenses.

For example, suppose direct modification by a malicious user has been identified as a possible damage to a system asset (say, a database), and that no immediate defense against direct modifictaion is known (it is impossible to disable all editors). If the system has network connections, then modification is only possible after the user has gained illegal access to the system. In this case, we would say that illegal access *enables* modifictaion. A defense against illegal access is therefore also a defense against modification.

The knowledge needed for reasoning is expressed in the usual manner as production rules which, if the conditions are met, assert the existence of new damages, defenses, enabling conditions, and so on.

4.2 The Content Representation Graph: Communication Domain Knowledge

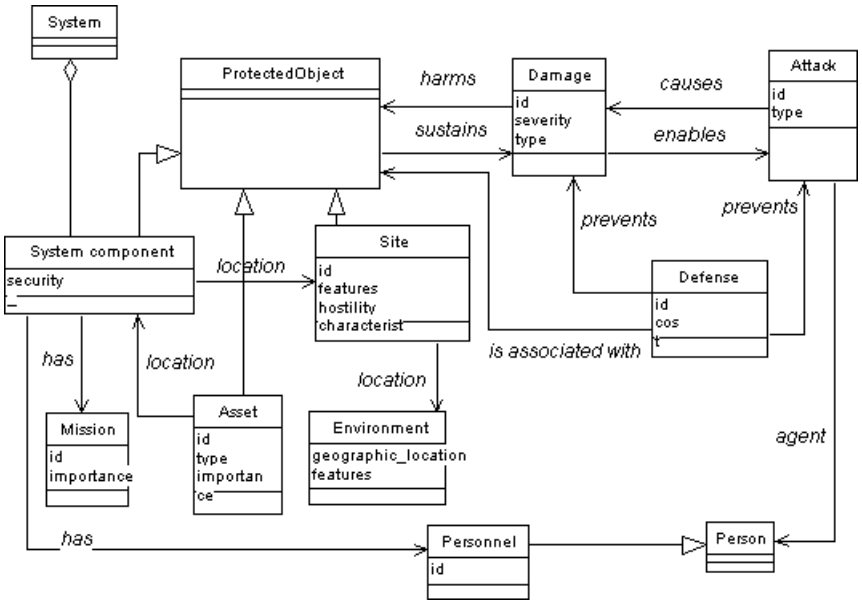

Figure 1: The domain model

In SA, the starting point for expressing CDK is a domain model of the type that is used in object-oriented design and analysis. Our domain model (Figure 1) represents security domain concepts, various attributes and concept relationships, as they are used in explanation. The domain Model was created by analyzing a small corpus of explanations of reasoning performed by the SA. The corpus had been written by a domain expert, and was analyzed by a text engineer. Each of the boxes in the model stands for a concept in the security domain, and inside these boxes are attributes associated with the concept. Arrow-tipped edges represent relations between concepts in the Knowledge Database, triangle-tipped edges represent *is-a* relations and diamond-tipped edges are *has-a* relations. Some examples:

- *Defense* objects have *id* (name) and *cost* attributes;
- *Damage* objects have *id*, *severity* and *type* attributes;
- *prevent* is a relation that holds between a *Defense* instance and a *Damage* instance;
- *Site*, *Asset* and *Location* are different sub-classes of *ProtectedObject*;
- A *System* consists of one or more *System components*.

This domain model has no role in the expert system reasoning. In fact, during the reasoning process, the expert system models the relations as primary objects, and the concepts of our domain model are merely slots of the relations in the expert system. As a result, the relations typically are not binary, but n-ary. In contrast, the domain model contains only binary relations. This reflects, we claim, the difference between the optimal way of representing knowledge for machine reasoning, and the way in which humans model the world (which is what the data model captures). As an example of the difference in relations, the relation that corresponds to the data model's *prevent* relation between *Defense* and *Damage* corresponds to, in the reasoning component, a quintary relation between the defense, the location of the defense, the damage it prevents, the locations at which it prevents the damage, and the damages that negate the defense. Another example is the *likely_attack_method* relation (and its structural clone, the *possible_attack_method* relation) of the reasoning component, which is a ternary relation between an asset, a location, and an attack method. As can be seen from the domain model diagram, this relation is not modeled in the data model at all.

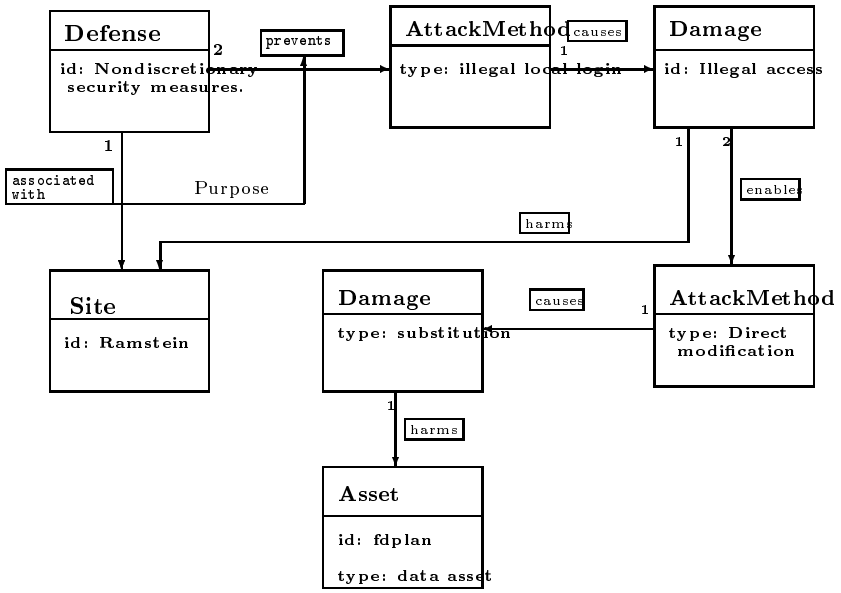

Figure 2: The CRG for the example

Knowledge about domain concepts and relationships is not sufficient for generating an expressive explanation. Additional CDK is required in order to select and organize content according to ex-

planation type and length limitation. The domain model is therefore augmented with the following information.

- Importance level, which is defined for every relation and attribute. This information about relative importance of attributes and relations enables us to produce explanations of different length. For example, the relation *prevent* between *Defense* and *Damage* has higher importance level than the relation *have* between *SystemComponent* and *Mission*. In our domain model, we use a two-valued scale.
- A key attribute for each concept which is required in instances of the cocept and which identifies an instance of the concept. For example, *id* is a key attributes for *Site* but *HostilityCharacteristic* is not a key attribute.
- Mutual dependencies among concept relations and attributes. This information covers cases in which a particular relation or attribute can be presented only if some other relations or attributes are presented. For example, the relation *prevent* between *Defense* and *AttackMethod* should be included only if the relation *cause* between *AttackMethod* and *Damage* is included as well.
- Order among relations and order among attributes of the same concept, namely in what order should relations of the concept be presented, e.g. for concept *damage* arc *goal* is ordered before arc *enable*.
- Meta-relations between relations of the same concept. For example, there is a meta-relation purpose between (*Defense prevent damage*) and (*Defense required ProtectedObject*).

To derive the CDK needed for a specific explanation task, the augmented domain model is instantiated. While the reasoning component performs the reasoning proper, it also populates the concepts of the augmented domain model with instances. The result is an instantiated model that contains concept instances, and attributes bound to their values. We called this instantiated model the "Content Representation Graph" (CRG) of the explanation. The CRG contains all the information that is needed for the generation of the text. An example of a CRG is shown in Figure 2.

4.3 Text Planning: Domain Communication Planning

As already mentioned, the CRG does not determine the form of the text, but only restricts its content. We implemented two different planners that build different texts from the same CRG. The first plan is intended to be used in an interactive setting, where the user can request more information if he or she is interested in it, by clicking on hyperlinks. An example is shown in Figure 3, where hyperlinks are shown by underlining.

In this application we used a planner with a declarative formalism for plan specification, which directly expresses the DCK (Lavoie and Rambow, 1998). Other representations of domain-specifc text planning knowledge could also have been used, and we omit details of the formalism we used.

However, for the DesignExpert application it is also necessary to generate explanations that are free of hypertext for inclusion in printed documents. These texts must include the entire explanation at

Nondiscretionary security measures are required on the Ramstein site.

- Which damage do nondiscretionary security measures prevent?
- Which assets do nondiscretionary security measures protect?

Figure 3: The interactive hypertext

Nondiscretionary security measures are required on the Ramstein site in order to prevent substitution of data asset "ftdplan". They prevent substitution because nondiscretionary security measures prevent illegal local login, which may enable illegal access to the Ramstein system. Illegal access may enable direct modification and direct modification may cause substitution to data asset "ftdplan".

Figure 4: The fluent, hyperlink-free text

a level of detail appropriate for the kind of expected reader. An example is shown in Figure 4. In order to create hyperlink-free explanation text, the CRG must be traversed according to constraints at every nodes: which attributes to use to describe the object, which relations of this object with other object must be presented in explanation, in what order to present the relations and what are meta-relationship between them. The planner processes every graph edge according to specified order, and structures resulting phrases with respect to meta-relations.

5 Methodology

We propose the following methodology for developing an explainable expert system. We assume three roles, that of the domain expert (where "domain" refers to the domain of the expert system, such as computer security or infectious diseases), knowledge engineer (a specialist in eliciting and representing domain models, specifically in the form of expert systems), and a communications engineer (a specialist in analyzing and representing the knowledge needed for efficient communication).

1. The knowledge engineer creates the expert system in consultation with the domain expert, using any sort of tool or shell and any sort of methodology that are convenient.
2. The domain expert writes several instances of (textual) explanations of the type needed for the application in question, based on scenarios that the expert system can handle.
3. The communication engineer analyzes the corpus of hand-written explanations along two lines:

- The domain concepts that are reported in the text are analyzed and recorded using an object-oriented modeling technique, perhaps augmented by more expressive constructs, such as meta-relations (relations between relations). This structure, called the *content representation graph*, represents the communication domain knowledge (both terminological and instances).
- The structure of the text is recorded using some standard notation for discourse structure (say, RST (Mann and Thompson, 1987)).

4. Using the communication domain knowledge representation, the communication engineer consults with the domain expert and the knowledge engineer to define a mapping from the domain representation used by the expert system to the communication domain knowledge representation devised by the communication engineer. The communication domain knowledge representation may be modified as a result.

5. The knowledge engineer adds rules to the expert system that instantiate the communication domain knowledge representation with instances generated during the reasoning process.

6. The communication engineer designs a text planner that draws on the knowledge in the CDK representation and produces text. This task involves the creation of an explicit representation of DCK for the domain and task (and genre) at hand.

The resulting system is modular in terms of software modules. The expert system is preserved as a stand-alone module (though its rule base has been somewhat extended to identify communication domain knowledge), as is the text planner. Both module can be off-the-shelf components. Only the CDK representation is designed in a task-specific manner, but of course standard knowledge representation tools can be used (Oracle, LOOM, etc.).

In addition, the methodology is modular in terms of tasks and expertise. The domain expert and knowledge engineer do not need to learn about communication, and the communication engineer does not need to understand the workings of the expert system (though she does need to understand the domain well enough in order to design communication strategies for it, of course).

6 Conclusion

We have presented an approach to expert system explanation which is based on a classification of types of knowledge into reasoning domain knowledge, communication domain knowledge, and domain communication knowledge. We have argued that this distinction, in addition to being theoretically appealing, allows us to better manage the software engineering aspect of explainable expert system development.

While we think that our approach is well suited to explaining the reasoning of expert systems to users after the fact, the approach does not appear to lend itself very well to answers to "Why are you asking?" type questions from the user (as opposed to "Why are you recommending this?", which is what the SA answers). This is because the CDK is not intended to mimic the system's reasoning. However, it may be possible to extend the CDK presentation to include a model (oriented towards explanations for the human user) of the expert system itself. Using this model and the domain

model, such questions could be answered in the presented framework. We intend to investigate this in future work.

Bibliography

Ehrhart, L. S., Korelsky, T., McCullough, D., McEnerney, J., Overmyer, S., Rambow, O., Webber, F., Flo, R., and White, D. (1998). **DesignExpert**: A knowledge-based tool for developing system-wide properties. Submitted.

Giarratano, J. and Riley (1994). *Expert Systems: Principles and Programming*. PWS Publishing Company, Boston.

Kittredge, R., Korelsky, T., and Rambow, O. (1991). On the need for domain communication knowledge. *Computational Intelligence*, 7(4).

Korelsky, T., McCullough, D., and Rambow, O. (1993). Knowledge requirements for the automatic generation of project management reports. In *Proceedings of the 6th Conference for Knowledge-Based Software Engineering (KBSE93)*. IEEE.

Lavoie, B. and Rambow, O. (1998). A framework for customizable generation of multi-modal presentations. In *36th Meeting of the Association for Computational Linguistics (ACL'98)*, Montréal, Canada. ACL.

Lester, J. C. and Porter, B. W. (1997). Developing and empirically evaluating robust explanation generators: The KNIGHT experiments. *Computational Linguistics*, 23(1):65–102.

Mann, W. C. and Thompson, S. A. (1987). Rhetorical Structure Theory: A theory of text organization. Technical Report ISI/RS-87-190, ISI.

McKeown, K. (1985). *Text Generation*. Cambridge University Press, Cambridge.

Moore, J. (1994). *Participating in Explanatory Dialogues*. MIT Press.

Paris, C., Wick, M., and Thompson, W. (1988). The line of reasoning versus the line of explanation. In *Proceedings of the 1988 AAAI Workshop on Explanation*, pages 4–7.

Paris, C. L. (1988). Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, 14(3):64–78.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: Mycin*. American Elsevier, New York.

Swartout, W. and Moore, J. (1993). Explanation in second generation expert systems. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 543–585. Springer Verlag.

Swartout, W., Paris, C., and Moore, J. (1991). Design for explainable expert systems. *IEEE Expert*, 6(3):59–64.

Tanner, M. C., Keunecke, A. M., and Chandrasekaran, B. (1993). Explanation using task structure and domain functional models. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 586–613. Springer Verlag.

Webber, F., McEnerney, J., and Kwiat, K. (1998). The DesignExpert approach to developing fault-tolerant and secure systems. In *4th Int'l Conf. on Reliability and Quality in Design*.

Wick, M. R. (1993). Second generation expert system explanation. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 614–640. Springer Verlag.