Designing Explanation Aware Systems: The Quest for Explanation Patterns

Jörg Cassens and **Anders Kofod-Petersen**
Department of Computer and Information Science
Norwegian University of Science and Technology
7048 Trondheim, Norway
{cassens,anderpe}@idi.ntnu.no

Abstract

Systems in general, and intelligent systems in particular, need to be able to explain their behaviour to their users or partners. Previously, a number of different user goals that explanations can support have been identified. Likewise, different kinds of explanations have been proposed. The problem remains how these abstract concepts can be made fruitful for the design of intelligent systems – they must be connected to software engineering methodologies. The work presented here builds on the concept of patterns and suggests using problem frames as a tool for requirements engineering. We further on propose to connect these problem frames with other design patterns as a tool supporting the implementation process.

Introduction

Although a wide variety of knowledge engineering methodologies exists (see the following section), there seems to be a lack of methods focusing on the peculiarities of explanatory knowledge. These special features of explanations include in particular their role in enhancing the user experience by adding a level of self reflection about the actions of the system and the importance of explanations to gain the user's trust into the system's capabilities.

In human to human interaction, the ability to explain its own behaviour and course of action is a prerequisite for a meaningful interchange, therefore a truly intelligent system has to provide comparable capacities. In order to make sure that the system has sufficient knowledge about itself and that potential interests a user can have towards explanations can be satisfied, the design of explanatory capabilities should be an integral component of the system's design process.

Additionally, if we have an interest in the widespread adoption of explanation aware systems, we have to integrate the methods focussing on knowledge aspects with other software development methodologies to make the design of intelligent system accessible for a large group of software engineers. To this end, it is our aim to develop a formal notation to support a design methodology which is based on experiences both in the intelligent systems and software development communities. We therefore propose the use of patterns, especially the use of problem frames, in order to start the discussion of such methodologies.

The structure of this article is as follows: first, we give a short overview about the notion of explanations as used in this paper. Second, we refer to some related work both from the knowledge engineering and the software engineering disciplines. In the following step, we propose a description of user goals following Jackson's notion of problem frames. Thereafter, we present a simplified example of explanation aware re-engineering of an existing ambient intelligent solution before concluding with some remarks about further work.

Explanations

The ability to generate explanations is an important aspect of any intelligent system (Roth-Berghofer & Cassens 2005; Sørmo, Cassens, & Aamodt 2005). We deem it necessary to investigate the explanatory capabilities from an early point onward in the design process to assure that the finished system can sufficiently explain itself. Therefore, an analysis of explanatory needs of both user and system should be part of the requirements engineering process. It is equally important to provide the system designers with methods to assure that explanatory knowledge and methods can be integrated into the application.

We have previously presented a framework for explanations in intelligent systems with a special focus on case-based reasoning (Sørmo, Cassens, & Aamodt 2005). Specifically, we have identified five goals that explanations can satisfy. The goal of *transparency* is concerned with the system's ability to explain how an answer was reached. *Justification* deals with the ability to explain why the answer is good. When dealing with the importance of a question asked, *relevance* is the goal that must be satisfied. *Conceptualisation* is the goal that handles the meaning of concepts. Finally, *learning* is in itself a goal, as it teaches us about the domain in question. These goals are defined from the perspective of a human user. His expectations on what constitutes a good explanation is situation dependend and has a historic dimension (Leake 1995).

Roth-Berghofer has explored some fundamental issues with different useful kinds of explanation and their connection to the different knowledge containers of a case-based reasoning system (Roth-Berghofer 2004). Based on earlier findings from natural language explanations in expert systems, five different kinds of explanation are identified: *conceptual explanations*, which map unknown new concepts to known ones, *why-explanations* describing causes

or justifications, *how-explanations* depicting causal chains for an event, *purpose-explanations* describing the purpose or use of something, and *cognitive explanations* predicting the behaviour of intelligent systems. Roth-Berghofer further on ties these different kinds of explanation to the different knowledge containers of case-based reasoning systems (Richter 1995), namely *case base*, *similarity measure*, *adaptation knowledge*, and *vocabulary*.

Building on the last two works, we have earlier started to investigate a combined framework of user goals and explanation kinds (Roth-Berghofer & Cassens 2005). The goal of this work was to outline a design methodology that starts from an analysis of usage scenarios in order to be able to identify possible expectations a user might have towards the explanatory capabilities of an intelligent system. The requirements recognised can further on be used to identify which kind of knowledge has to be represented in the system, and which knowledge containers are best suited for this task. In that work, we have also identified the need for a socio-psychological analysis of workplaces in order to be able to design systems which can meaningful engage in socio-technical interactions.

We have also previously proposed the use of activity theory, a theory of human interaction with other humans and technical artefacts from industrial and organisational psychology, to investigate when explanations are important to the user (Cassens 2004). The same theory has shown its usefulness in designing a case-based reasoning system geared towards ambient intelligence (Kofod-Petersen & Cassens 2006). This work has recently been extended to explicitly take explanatory capabilities into account.

We are now in the process of investigating how these different aspects – a socio-technical analysis, user goals with explanations, and the different kinds of explanations – can fit into a design methodology which can be handled by knowledge and system engineers.

Related Work

The use of patterns (Alexander *et al.* 1977) is common for different software engineering approaches. Patterns can be used in different software development phases and they can have different foci. We can also identify knowledge engineering approaches making use of patterns.

When we look towards the software engineering world, we can see that patterns are used in different phases of the design process.

Early on in the requirements engineering process, *problem frames* (Jackson 2001) are a method to classify software development problems. Problem frames look out into the world and attempt to describe the problem and its solution in the real world. Problem frames introduce concepts like 'Information Display' and 'Commanded Behaviour'.

Jackson's set of basic problem frames can be extended to be better able to model domain specific aspects. For example, (Hatebur & Heisel 2005) introduce new problem frames for security problems. Their proposal includes problem frames for issues like 'Accept Authentication' and 'Secure Data Transmission'. They also provide architectural patterns connected to these problem frames.

On the software architecture level, we find *architecture patterns* (Avgeriou & Zdun 2005). At this level, we encounter concepts like 'Blackboards', 'Model-View-Controller', or 'Pipes and Filters'.

For finer grained software development close to the actual implementation, one can make use of design patterns which look inside towards the computer and its software (Gamma *et al.* 1995). Design patterns deal with concepts like 'Factories', 'Facade', and 'Decorater'.

Patterns can also be useful for modelling non-functional requirements. HCI design patterns are such a special class of patterns. Rossi, Schwabe, & Lyardet (2000) introduce HCI patterns for hypermedia applications (like 'Information on Demand' and 'Process Feedback'). Another collection of HCI patterns can be found in (van Welie & Trætteberg 2000), covering aspects like 'Wizards' or 'Preferences'.

Some research has been done on the issue of how patterns on different levels are related with each other. For example, (Wentzlaff & Specker 2006) apply case-based reasoning to construct design patterns from developed problem frames. The problem part of the cases consist of problem frames, and the solution part is a corresponding HCI pattern.

Choppy, Hatebur, & Heisel (2006) relate architectural patterns to problem frames. The design problem at hand can be divided into multiple frames, and the authors offer a modular approach to refining the problem frames into architectural patterns.

Methods and languages which use patterns and focus explicitly on the knowledge aspects of system design exist as well. There are for example efforts to provide reusable architectures by describing the abilities of (a library of) generic problem solving methods. An example for a component model is the Unified Problem-Solving Method Development Language UPML, cf. (Fensel *et al.* 1999).

Plaza and Arcos (1999) describe an application of the UPML model to case-based reasoning (CBR). They propose the ABC software architecture, based on the three components *task description*, *domain model*, and *adaptors*. The authors focus on the reuse part of the CBR cycle (Aamodt & Plaza March 1994) and interpret problem-solving as constructing a (case-specific) model of the input problem.

The INRECA (Bergmann *et al.* 2003) methodology is aimed at developing (industrial) case-based reasoning applications. Software process models from existing CBR applications are stored in an experience base which is structured at three levels. The *common generic level* is a collection of very generic processes, products, and methods for CBR applications. At the *cookbook level*, we find software models for particular classes of applications (so called recipes). At the *specific project level*, experiences from particular projects are stored. We can identify the recipes at the cookbook level as patterns.

Another well-known approach can be found with the CommonKADS methodology (Schreiber *et al.* 2000). It encompasses both a *result perspective* with a set of models of different aspects of the knowledge based system and its environment, and a *project management perspective* starting from a spiral life-cycle model that can be adapted to the particular project.

The CommonKADS template knowledge model provides a way of (partially) reusing knowledge models in new applications and can be understood as patterns in the software engineering sense of the word.

Building on the KADS model and extending it explicitly towards software engineering, (Gardner *et al.* 1998) introduce the notion of KADS Objects. The KADS Object framework allows direct support for object-oriented decomposition and utilises research on human cognition. The authors also supply a library of generic problem-solving templates, which themselves can be seen as software engineering patterns.

Unfortunately, despite the fact that a lot of work has been done on knowledge engineering methodologies and in particular the reuse of experience gained, it seems that little attention has been paid to the specifics of explanatory knowledge outlined above.

Explanation Problem Frames

The main purpose of any problem frame (Jackson 2001) is to propose a machine which improves the combined performance of itself and its environment by describing the machine's behaviour in a specification. The most important approach is to address the frame concern. To explain ones behaviour a problem frame must be constructed that relates the behaviour the system shows to different parts of knowledge used by the system to support the chosen course of action in a specification.

Jackson (2001) originally described five different basic frames, each of which comes in different *flavours* and *variants*: 'required behaviour', 'commanded behaviour', 'information display', 'simple workpieces' and 'transformation'. Each basic frame has its own requirements, domain characteristics, domain involvement, and frame concern.

In general, a problem frame assumes a user driven perspective. Except for the 'required behaviour' basic frame, each frame assumes that the user is in control and dictates the behaviour of the machine. Since intelligent systems (ideally) take a much more pro active approach and mixed initiative issues become relevant, new problem frames addressing these topics have to be developed. For the course of this paper, we will focus exclusively on frames targeting explanatory aspects and will not discuss other types of problem frames.

Problem frames can be described by problem frame diagrams. These diagrams consist basically of dashed ovals, representing the requirements, plain rectangles, denoting application domains, and a rectangle with a double vertical stripe, standing for the machine (or software machine) domain to be developed. These entities become the nodes of the frame diagram. They are connected by edges, representing shared phenomena and denoting an interface. Dashed edges refer to requirement references. Dashed arrows designate constraining requirement references.

The domains can be of different types, indicated by a letter in the lower right corner. Here, a 'C' stands for a *causal* domain whose properties include predictable causal relationships among its phenomena. A 'B' denotes a *biddable*

domain which lacks positive predictable internal behaviour. Biddable domains are usually associated with user actions. Finally, an 'X' marks a *lexical* domain. Such a domain is a physical representation of data and combines causal and symbolic phenomena.

In the software development process, problem frames are used in the following way. First, we start with a *context diagram*, which consists of domain nodes and their relations, but without the requirements. Afterwards, the context diagram is divided into sub problems. The resulting sub problems should, whenever possible, relate to existing generic *problem frames*. These generic problem frames are hereby instantiated do describe the particular sub problem at hand.

In the remainder of this section, we propose a set of new generic problem frames to capture aspects of explanations connected to the aforementioned different user goals identified in (Sørmo, Cassens, & Aamodt 2005).

Transparency Explanation
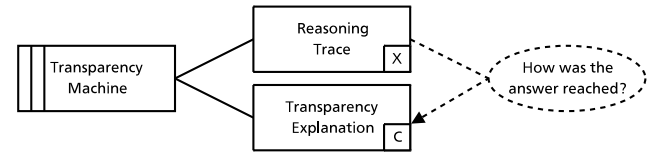
Figure 1: Transparency Explanation. An explanation supporting this goal gives the user some insight into the inner working of the system. To this end, the system inspects its own reasoning trace when formulating the explanation.

This goal is concerned with an explanation of how the system reached the answer.
"I had the same problem with my car yesterday, and charging the battery fixed it."
The goal of an explanation of this kind is to impart an understanding of how the system found an answer. This allows the users to check the system by examining the way it reasons and allows them to look for explanations for why the system has reached a surprising or anomalous result.

The frame diagram depicted in Figure 1 highlights that in order to support the transparency goal, the software system has to inspect its reasoning trace and represent the relevant facts of its reasoning process to the user. We expect a transparency explanation usually to be given after the reasoning process has terminated.
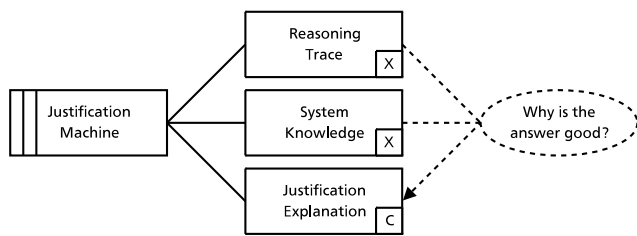
Justification Explanation

When supporting the justification goal, we want to explain why the answer given is a good answer.
"You should eat more fish - your heart needs it!"
"My predictions have been 80% correct up until now."
This is the goal of increasing the confidence in the advice or solution offered by the system by giving some kind of support for the conclusion suggested by the system. This goal allows for a simplification of the explanation compared to the actual process the system goes through to find a solution. Potentially, this kind of explanation can be completely

Figure 2: *Justification Explanation. In contrast to the transparency explanation, the user is here not primarily interested in why the system exposes a particular behaviour, but wants to have evidence supporting that this behaviour is correct. Therefore, other knowledge has to be taken into account besides the reasoning trace.*

decoupled from the reasoning process, but it may also be achieved by using additional background knowledge or reformulation and simplification of knowledge that is used in the reasoning process.

A explanation supporting the justification goal, as shown in Figure 2, has not only to take the reasoning of the machine into account, but it will also make use of other parts of the system's knowledge in order to generate after the fact explanations supporting its actions or decisions. Since justification explanations complement transparency explanations, we expect it to be given usually after the reasoning process has terminated.

Relevance Explanation
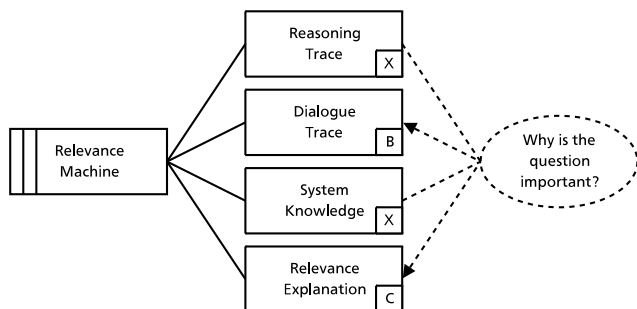
Figure 3: *Relevance Explanation. An explanation supporting this goal should instil confidence by indicating that the system's behaviour is connected to the task at hand. Consequently, the reasoning and dialogue traces should be taken into account as well as other (domain) knowledge.*

An explanation targeting this goal gives hints about why a question asked is relevant.
"I ask about the more common failures first, and many users do forget to connect the power cable."
An explanation of this type would have to justify the strategy pursued by the system. This is in contrast to the previous two goals that focus on the solution. The reasoning trace type of explanations may display the strategy of the system implicitly, but it does not argue why it is a good strategy. In conversational systems, the user may wish to know why a question asked by the system is relevant to the task at hand.

It can also be relevant in other kinds of systems where a user would like to verify that the approach used by the system is valid.

Since this goal, depicted by the frame diagram in Figure 3, is of particular interest for mixed initiative systems, the explaining machine has to relate its explanation both to its own dialogue with the user (and here in particular the questions asked by the system or the actions performed), the reasoning trace (in oder to relate to the situation the system assumes it is in) and the system knowledge relevant. In contrast to the first two goals, an explanation supporting this goal is important to be given during the reasoning process of the system.

Conceptualisation Explanation
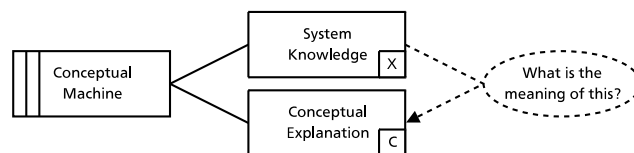
Figure 4: *Conceptualisation Explanation. By giving a conceptualisation explanation, the system explicates its own conceptualisation of the domain or the task at hand to the user. Hence, it will connect the concept to be explained with its own knowledge components.*

The conceptualisation goal deals with the need to clarify the meaning of concepts.
"By 'conceptualisation' we mean the process of forming concepts and relations between concepts."
One of the lessons learned after the first wave of expert systems had been analysed was that the users did not always understand the terms used by a system. This may be because the user is a novice in the domain, but also because different people can use terms differently or organise the knowledge in different ways. It may not be clear, even to an expert, what the system means when using a specific term, and he may want to get an explanation of what the system means when using it.

This explanation machine, represented with the frame diagram depicted in Figure 4, builds on its own system knowledge. This highlights the fact that explanations supporting this goal should set unknown concepts in the context of the other knowledge the system has, and which is expected to be shared with the user already. Conceptualisation explanations are important both during the reasoning process (e.g. in addition to a relevance explanation) and after the reasoning process has terminated (e.g. in addition to a justification explanation).

Learning Explanation

The learning goal focuses on the interest of the user to learn something about the application domain.
"When the headlights won't work, the battery may be flat as it is supposed to deliver power to the lights."
This goal is of specific interest for educational applications, which have learning as the primary goal of the whole system. In these systems, we cannot assume that the user will
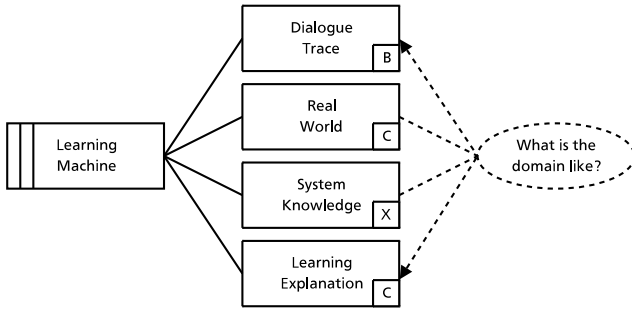
Figure 5: Learning Explanation. This goal is special, since it focuses on the user's interest in the application domain (hence the real world), and not on some particular behaviour of the system.

understand even definitions of terms, and may need to provide explanations at different levels of expertise. The goal of the system is typically not only to find a good solution to a problem, but to explain the solution process to the user in a way that will increase his understanding of the domain. The goal can be to teach more general domain theory or to train the user in solving problems similar to those solved by the system. In other words, the explanation is often more important than the answer itself.

The Figure 5 highlights this fact by pointing out that the explanatory machine has to connect its own system knowledge with the real world (representing the application domain) in order to generate explanations supporting the user in gaining a better understanding of the application domain. The explanation given should also relate to the system's assumptions about its user, which is influenced by the dialogue trace. Because of the nature of this goal, it will usually be important during the system's reasoning process.

Example

Like mentioned in the introduction, we have designed and implemented an ambient intelligent application (Cassens & Kofod-Petersen 2006) where the main purpose is to identify ongoing situations and proactively acquire digital information required by the persons present.

The system consists of three main components: one component acquiring data from the environment relevant for classifying the situation (Kofod-Petersen & Mikalsen 2005); one component assessing the ongoing situation through the use of case-based reasoning, which we understand as being context aware (Kofod-Petersen & Aamodt 2006); and finally one component conducting a task decomposition to solve the problem in the ongoing situation, which we understand as being context sensitive (Gundersen & Kofod-Petersen 2005).

To exemplify how problem frames can assist us in building explanation aware applications, let us look at a typical engineering task where we start with the existing system and want to re-engineer it to include explanations.

As an example, we have the instance where the system correctly classifies an ongoing situation as a pre-ward round. A pre-ward round is a particular type of meeting that occurs

every morning. Here the physician in charge and the nurse in charge go over the status of the patients on the ward, and decide on the treatment plan. The current condition of the patients in question is reviewed in light of any changes, test results, and the like.

We know from the knowledge acquisition and modelling process (Cassens & Kofod-Petersen 2006) that the goal of this type of situation can be decomposed into the following sequence of tasks:

1. Acquire name of patient;
2. Acquire changes in patient's condition since yesterday;
3. Acquire any new results from tests;
4. Examine, and possible change, medication scheme;
5. Note changes in treatment.

If we focus on the context sensitive part of the system, its main purpose is to examine the artefacts, represented by agents, in the environment and find those that can supply relevant information. If we examine a particular pre-ward round situation, here one occurring at the cardiology ward, the problem can be decomposed as depicted in Figure 10.

Figure 10 demonstrates how the initial problem of finding the name of the patient can be facilitated by the *Patient List Agent*. Further on, the 'Acquire Information' task is decomposed into one task that acquires changes which are supplied by the *Electronic Patient Record*, the *WiseW* application and the *Patient Chart*, and another task that acquires results which can be delivered by the *Patient Chart* and the *WiseW* application. So far this application only supplies information without any explanation of its behaviour.

In order to demonstrate how the explanation goal problem frames can be used to model explanatory needs in the problem domain, we will start with a simplified problem diagram for our application (Figure 6). We have modified Jackson's *information display* problem frame (Figure 7) and used it as a starting point for the diagram. You can see three domains representing (groups of) the agents mentioned in Figure 10 and explained above.
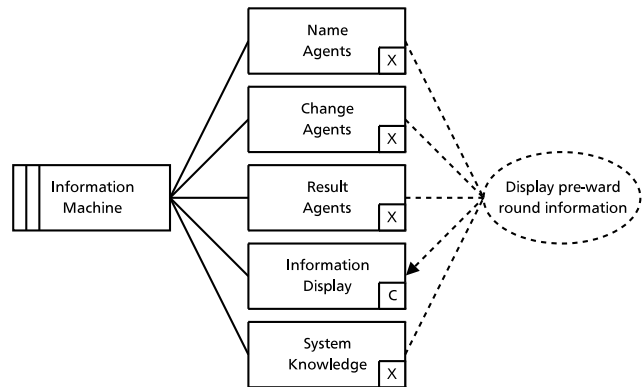
Figure 6: Simplified problem diagram for an ambient intelligent system for displaying medical information in pre-ward rounds.

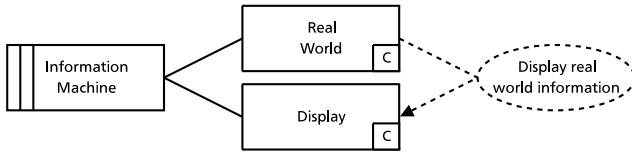Additionally, you see the 'Display' domain which stands for the information display of the system and 'System

Figure 7: *Jackson's information display problem frame diagram.*

Knowledge' for deciding which data sources to use. For the sake of clarity and simplicity, we have abstracted away the sensor parts of as well as the context aware parts of our example application and focus solely on the information gathering and display parts.

Let us now assume that the results displayed by the system are of such a nature that the physician using the system requires an explanation. Let us further focus on the transparency and justification explanations.

The transparency and justification explanations are related in the sense that they to some degree serve the same purpose. Namely, to persuade the user of the validity of the proposed solution and/or the validity of the problem solving approach chosen by the system. In the work presented here, we decide upon which of the two explanations to present as a function of the user's level of competence. That is, expert users are subject to transparency explanations and novice users to justification explanations (Mao & Benbasat 2000).
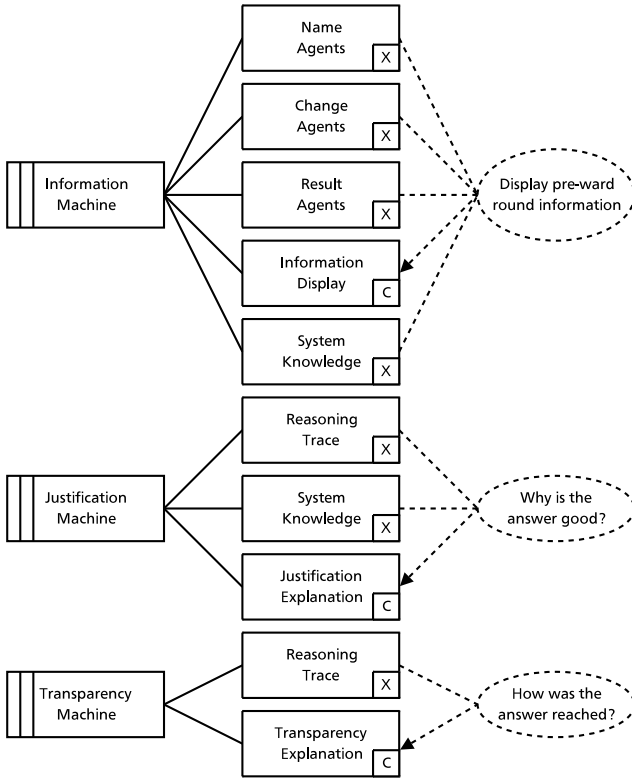

Figure 8: *Simplified problem diagram with explanation problem frames added.*

To model the explanatory capabilities of the system, we want to integrate the explanation sub problems described by the two problem frame diagrams for the *Transparency* and the *Justification* goal with the original application problem diagram. The goal is to compose a single problem diagram for the three sub problems depicted in Figure 8.
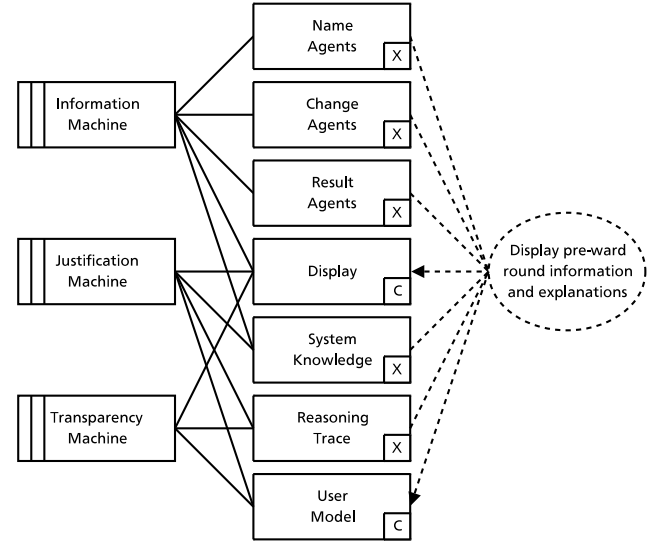

Figure 9: *Simplified problem diagram with explanation problem frames integrated and user model added.*

Figure 9 shows one possible problem diagram for an explanation aware patient information display system. In order to be able to chose the right type of explanation – transparency or justification – we have included a user model component in this diagram. The need for this domain became clear when we tried to integrate the two different explanation machines. Please note also that we now share one common display for both the information delivery and explanation delivery.

The problem diagram depicted in Figure 9 is a simplified version of a real world diagram. The solution shown is probably not the best one possible, but it can be seen that different sub problems can be composed into a larger problem diagram. Some of the domains of the problems can be shared, whereas others might only be used by one or some sub problems (please note the user model in our example).

After we have included the explanatory machine in our problem diagram, we can re-visit the problem described above. The expert user physician might wish to know how the particular combination of information displayed was reached. According to the transparency explanation problem frame, this explanation can be achieved by displaying the reasoning trace. This can for example be done by showing that the top task 'Pre-ward round' was selected as a function of the classification, by displaying how the decomposition tree looks like, and by supplying information about the agents which were selected.

For the justification explanation, the novice user physician would like to know why this combination of information is
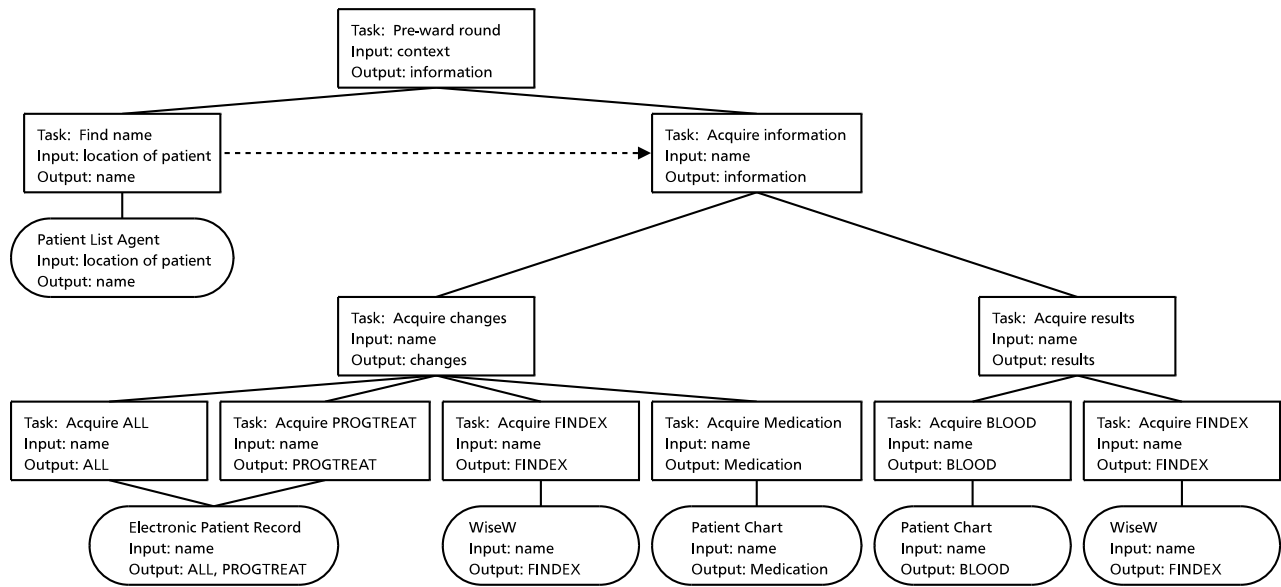
Figure 10: *Decomposition tree for a pre-ward round situation, as constructed by our context sensitive component.*

any good. This can be achieved by relating the reasoning trace to the domain model of the system. For example, according to the domain model, the 'Acquire Medication' task could be satisfied not only by the *Patient Chart* but also by the *Electronic Patient Record*. However, as the *Electronic Patient Record* agent was busy serving other requests only the *Patient Chart* could respond to this request.

This example shows how the use of explanation goal problem frames can explicate which kind of explanatory knowledge is necessary to support the explanatory needs of the users of the system. It can help identifying the structure of the problem at hand and the knowledge components required. It can further on be used as a means of communication between prospective users, software engineers, and knowledge engineers.

Conclusion and Future Work

This is ongoing work, but we have sketched how a set of problem frames targeted specifically towards explanatory capabilities of knowledge based systems can support the engineering process of explanation aware systems. With the explicit use of patterns, we have started to formalise the previously introduced notions of explanation goals and explanation kinds.

Until now, we have only looked out into the environment in which the intelligent system has to function, and proposed a formal notation for the description of user goals. These outward looking descriptions should further be connected with another view looking inwards towards the implementation of the system.

There are two directions of research we want to explore. One direction is to amend the explanation problem frames and to further analyse the relation between explanation goals and explanation kinds. To this end, we have to formalise the previously proposed stepwise refinement process from goals to kinds (Roth-Berghofer & Cassens 2005) so that we can construct combined patterns for goals and kinds. By this, we are going to populate the proposed model with examples for how the outward directed view on the non functional user requirements for explanation aware systems can be combined with the inward directed view of necessary explanatory knowledge.

The other direction is aimed at relating the proposed explanation goal problem frames with architectural patterns. Ideally, this would enable us to discuss explanation issues, knowledge aspects, HCI aspects, and functional requirements at a very early stage of the development process without any a priori assumptions about the problem solving methods used.

Further on, it is necessary to extend the formalism at "both ends", meaning that we on one hand have to revisit our analysis of how the necessary knowledge to support the different explanation kinds can be represented in the actual system, and that we on the other hand have to refine our socio-technical analysis to end up with a psychologically plausible model for elucidating the explanatory needs of the prospected users.

Acknowledgements

References

Aamodt, A., and Plaza, E. March 1994. Case-Based Reasoning; Foundational Issues, Methodological Variations, and System Approaches. *AI Communications* 7(1):39–59.

Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.;

Fiksdahl-King, I.; and Angel, S. 1977. *A Pattern Language*. New York: Oxford University Press.

Avgeriou, P., and Zdun, U. 2005. Architectural Patterns Revisited – A Pattern Language. In *Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPlop 2005)*, 1–39.

Bergmann, R.; Althoff, K.-D.; Breen, S.; Göker, M.; Manago, M.; Traphöner, R.; and Wess, S. 2003. *Developing Industrial Case-Based Resoning Applications: The INRECA Methodology*, volume 1612 of *LNCS*. Berlin: Springer-Verlag, second edition.

Cassens, J., and Kofod-Petersen, A. 2006. Using Activity Theory to Model Context Awareness: a Qualitative Case Study. In *Proceedings of the 19th International Florida Artificial Intelligence Research Society Conference*, 619–624. Menlo Park, CA: AAAI Press.

Cassens, J. 2004. Knowing What to Explain and When. In Gervás, P., and Gupta, K. M., eds., *Proceedings of the EC-CBR 2004 Workshops*, number 142-04 in Technical Report of the Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 97–104.

Choppy, C.; Hatebur, D.; and Heisel, M. 2006. Component Composition through Architectural Patterns for Problem Frames. In *APSEC '06: Proceedings of the XIII Asia Pacific Software Engineering Conference*, 27–36. Washington, DC: IEEE Computer Society.

Fensel, D.; Benjamins, R.; Decker, S.; Gaspari, M.; Groenboom, R.; Grosso, W.; Musen, M.; Motta, E.; Plaza, E.; Schreiber, G.; Studer, R.; and Wielinga, B. 1999. The Component Model of UPML in a Nutshell. In *WWW Proceedings WICSA1, 1st Working IFIP Conference on Software Architectures*.

Gamma, E.; Helm, R.; Johnson, R.; and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley.

Gardner, K. M.; Rush, A.; Crist, M. K.; Konitzer, R.; and Teegarden, B. 1998. *Cognitive Patterns*. Cambridge, UK: Cambridge University Press.

Gundersen, O. E., and Kofod-Petersen, A. 2005. Multiagent Based Problem-solving in a Mobile Environment. In Coward, E., ed., *Norsk Informatikkonferanse 2005, NIK 2005*, 7–18. Institutt for Informatikk, Universitetet i Bergen.

Hatebur, D., and Heisel, M. 2005. Problem Frames and Architectures for Security Problems. In Winther, R.; Gran, B. A.; and Dahll, G., eds., *Proceedings of the 24th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, number 3688 in LNCS, 390–404. Berlin: Springer-Verlag.

Jackson, M. 2001. *Problem Frames – Analysing and Structuring Software Development Problems*. Boston, MA: Addison-Wesley.

Kofod-Petersen, A., and Aamodt, A. 2006. Contextualised Ambient Intelligence through Case-Based Reasoning. In Roth-Berghofer, T. R.; Göker, M. H.; and Güvenir, H. A., eds., *Proceedings of the Eighth European Conference on Case-Based Reasoning (ECCBR 2006)*, volume 4106 of *LNCS*, 211–225. Berlin: Springer-Verlag.

Kofod-Petersen, A., and Cassens, J. 2006. Using Activity Theory to Model Context Awareness. In Roth-Berghofer, T. R.; Schulz, S.; and Leake, D. B., eds., *Modeling and Retrieval of Context: Second International Workshop, MRC 2005, Revised Selected Papers*, volume 3946 of *LNCS*, 1–17. Berlin: Springer-Verlag.

Kofod-Petersen, A., and Mikalsen, M. 2005. Context: Representation and Reasoning – Representing and Reasoning about Context in a Mobile Environment. *Revue d'Intelligence Artificielle* 19(3):479–498.

Leake, D. B. 1995. Goal-Based Explanation Evaluation. In *Goal-Driven Learning*. Cambridge: MIT Press. 251–285.

Mao, J.-Y., and Benbasat, I. 2000. The Use of Explanations in Knowledge-Based Systems: Cognitive Perspectives and a Process-Tracing Analysis. *Journal of Management Information Systems* 17(2):153–179.

Plaza, E., and Arco, J. L. 1999. The ABC of Adaptation: Towards a Software Architecture for Adaptation-centered CBR Systems. Research Report 99-22, Artificial Intelligence Research Institute (IIIA), Bellaterra, Spain. http://www.iiia.csic.es/.

Richter, M. M. 1995. The Knowledge Contained in Similarity Measures. Invited Talk at the First International Conference on Case-Based Reasoning, ICCBR'95, Sesimbra, Portugal.

Rossi, G.; Schwabe, D.; and Lyardet, F. 2000. User Interface Patterns for Hypermedia Applications. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, 136–142. New York, NY: ACM Press.

Roth-Berghofer, T. R., and Cassens, J. 2005. Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems. In Muñoz-Avila, H., and Ricci, F., eds., *Case Based Reasoning Research and Development – ICCBR 2005*, volume 3630 of *LNCS*, 451–464. Berlin: Springer-Verlag.

Roth-Berghofer, T. R. 2004. Explanations and Case-Based Reasoning: Foundational Issues. In Funk, P., and Calero, P. A. G., eds., *Proceedings of the 7th European Conference (ECCBR 2004)*, volume 3155 of *LNCS*, 389–403. Berlin: Springer-Verlag.

Schreiber, G.; Akkermans, H.; Anjewierden, A.; de Hoog, R.; Shadbolt, N.; de Velde, W. V.; and Wielinga, B. 2000. *Knowledge Engineering and Management – The CommonKADS Methodology*. Cambridge, MA: MIT Press.

Sørmo, F.; Cassens, J.; and Aamodt, A. 2005. Explanation in Case-Based Reasoning – Perspectives and Goals. *Artificial Intelligence Review* 24(2):109–143.

van Welie, M., and Trætteberg, H. 2000. Interaction Patterns in User Interfaces. In *Proceedings of the 7th Pattern Languages of Programs Conference (PLoP)*.

Wentzlaff, I., and Specker, M. 2006. Pattern-Based Development of User-Friendly Web Applications. In *ICWE '06: Workshop Proceedings of the sixth International Conference on Web Engineering*. New York, NY: ACM Press.