Proceedings of the ACM/IEEE International Workshop on Applications and Advances of Problem Orientation

Jon G. Hall and James Naish and Lucia Rapanotti and Liping Zhao (eds)

07 March, 2011

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Dear friend and colleague,

Thank you for your contribution to and participation in the IWAAPO 2010 workshop! It is good to know that the community has you at its heart.

As you will see from the enclosed programme there is a very good balance of new to experienced academics, covering a wide range of interesting topics. We hope that the interactions will be lively and that experience, insight and enthusiasm will flow both ways. We have tried to group similar topics together in the programme to facilitate the discussion.

There have been some logistical difficulties in organising the workshop - mainly down to the expense of attending ICSE, and we hope that this explains the participation of many students in ‘distributed mode’. Distributed means we will be using elluminate (elluminate.com, or Skype if all else fails). Indeed, this will provide an opportunity to build the mechanisms by which a distributed problem oriented community can thrive!

Unfortunately, because of a health related issue, Professor Alistair Sutcliffe is unable to attend ICSE. Fortunately, we have our very own Jon Hall who has very kindly accepted our invitation to replace Alistair. As many of you know, for many years, Jon and Lucia have been working side-by-side with Michael Jackson on problem-frames and problem oriented software engineering. Jon will be talking to us about “Tangled Problems”, a very stimulating topic indeed!

This is the promised technical report for the workshop and after the workshop, we will invite papers for a special issue of Wiley-Blackwell’s Expert Systems: The Journal of Knowledge Engineering.

We would like to thank the organisers of the Workshops for their time and boundless energy, and IARIA, the International Academy, Research, and Industry Association, for their sponsorship.

Thank you for attending IWAAPO 2010. I do hope you will enjoy it!

Jon, Lucia, Liping, James
IWAAPO Programme

9.00 Workshop registration
9.20 Welcome and Introduction
9.30 Invited Talk: Jon Hall
10.30 Coffee Break
11.00 Presentations (Open Mode)

- Capturing Design Rationale with Problem Oriented Engineering
  Ann A Nkwocha, Bank of America
  page 39

- A POSE Process with Alloy and Perfect Developer
  Derek Mannering, General Dynamics UK
  page 37

- Towards Tool Support for Problem-Oriented Software Engineering using Knowledge-Based Modelling
  Zhi Li, Guangxi Normal University
  page 42

- Problem Oriented Engineering Process Algebra
  Dariusz W. Kaminski, Marine Lab, Fisheries Research Services
  page 35

- A Second Life learning space for Problem Oriented Engineering
  Andy Moore, The Open University
  page 45

1.00 Lunch
2.00 Presentations (Co-located Mode)

- Using Problem Frame Concern to Drive Layered Requirements Analysis
  Jie Sun, Ye Wang, Xinyu Wang, Xiaohu Yang, and Aleksander J. Kavs
  page 24

- User Needs vs. User Requirements: a Problem Frame-based View
  Luigi Lavazza
  page 7

- Non-functional Attributes for Problem Frames Approach
  Bin Yin, Zhi Jin
  page 30

- Structuring Problem Analysis for Embedded Systems Modelling
  Jelena Marincic, Angelika Mader, Roel Wieringa, and Yan Lucas
  page 14

4.00 Coffee Break
4.20 Plenary: The Way Forward---Roadmap and Action Plan
5.00 Close
7.00 Dinner (not included in registration fees)
Abstract—It is well known that the analysis of requirements involves several perspectives and stakeholders. Very often several points of view at different abstraction levels have to be taken into account. All these features make requirements analysis a complex task. Such intrinsic complexity makes it difficult to understand several of the basic concepts that underlie requirements engineering. Actually, there is a bit of confusion—especially in industry—about what really is a user requirement, what are the differences between user requirements and user needs, and what are the relations with business processes. This paper discusses these issues and provides some clarifications, using the problem frames concepts and notation. Some indications about an analysis process that is aware of the differences between user needs and requirements are also given.

Index Terms—User needs, user requirements, problem frames, requirements analysis.

I. INTRODUCTION

In the analysis and documentation of user requirements several perspectives are often involved. The multiplicity of scopes and points of view is reflected in the terminology: it is quite usual to hear about ‘business processes’, ‘user needs’, ‘user requirements’, etc. Actually, there is a bit of confusion—especially in industry—about what really is a user requirement, what are the differences (if any) between a user requirement and a user need, and what are the relations with business processes. Moreover, how to effectively and correctly model all these aspects is not always clear.

The result is that user requirements tend to encompass multiple different aspects of the systems to be developed, and to be represented in very different manners, often mixing requirements with other issues.

In this paper the differences between user needs and user requirements are highlighted; the relation of user needs to business processes is also pointed out; finally, the usage of problem frames to model the mentioned issues is illustrated, using a case study.

The paper is organized as follows: the problem of requirements modeling is introduced in Section II. The involved issues are presented in Section III with the help of a case study. The distinction between needs and requirements is further discussed in Section V, and an analysis process aware of this distinction is sketched. Section VI reports about related work, while Section VII draws some conclusions and sketches future work.

Throughout this paper, the concept and notation of problem frames [1] are used. The reader is expected to know them.

II. THE PROBLEM OF REQUIREMENT MODELING

This section addresses the problem of establishing the precise meaning of requirements modeling.

First of all, we have to notice that there is an Environment in which the problems exist, and where the machine (i.e., the hardware/software solution to be developed) will operate. The problem domain is the portion of the Environment that is visible by the machine. Of course, the Environment interacts with the machine; in fact, a machine that does not get any input and does not provide any output would be hardly useful. The relationship between the Environment and the machine (which is seen as a black box in this phase) is described in terms of “phenomena” that are shared between the Problem Domain and the Machine.

The requirements are defined as properties expressed over the Environment and the Machine, while specifications are defined as properties expressed over the Environment-Interface interface only. An example is given in Section III to illustrate the concepts.

A. Focusing on the problem

In order to develop a software solution it is necessary to study and understand the problem. In other words, we need to understand where the problem is and where the solution is. Fig. 1 schematically represents the world (i.e., the environment) where the problem exists, and the machine (i.e., the hardware/software solution we have to provide to solve the problem). The relationships between the machine and the world are highlighted. In fact, it is fundamental that the machine gets the information it needs about the world, and that is provides outputs (either information or control) to the surrounding environment.
Fig. 1 The problem and solution spaces.

B. Describing problems

How can we define and describe a problem in order to support the development of a software-based solution? We have to keep in mind that developers have to be given all the relevant information concerning the desired behavior of the system: this information constitutes the requirements of the software to be developed. Therefore, Requirements are the criteria, conditions or constraint that a software artifact has to satisfy in order to be acceptable as a solution of the given problem.

C. Requirements engineering: what is it?

Some misbelieve that requirements simply represent what the users asks. This is generally not the case, because the user tends to talk about what it the problem in the real world, not about the role of the machine in solving it. In fact, the user knows the problem well, but often he/she hasn’t the slightest cue of how to solve it, or, even worse, has completely wrong ideas of how to solve the problem, because he/she does not know what is easy or hard to achieve by means of a computer system. This situation has been observed quite often, and it has generally been described saying that “the user does not know what he/she wants”. Actually, the user knows well what he/she wants in the problem domain, while he/she does not know what this should imply at the machine level, i.e., how should the machine behave in order to solve the problem.

In conclusion, requirements have to be proactively defined on the basis of the description of the problem provided by the user and on the knowledge of the environment where the machine will have to operate.

Requirements engineering is thus the creative activity that leads to defining the effects that the HW/SW machine will have on the application domain so that the problem is solved. To this end, it is necessary to remember that it is the behavior of the system combining the domain and the machine that has to solve (or at least to reduce) the problems that induced the user to ask for a computer-based solution.

The analyst has a double role in describing requirements:
- He/she must understand the needs of the user and the nature and behavior of the environment.
- He/she must (pro)actively cooperate with the user in defining requirements that are effective (i.e., the machine actually contributes to solve the problem), technically feasible and reasonable with respect to economic constraints.

The final product of the analyst is the requirement specifications. The goals of the requirement specifications are:
- To describe what are the responsibilities of the system in satisfying the user’s needs.
- To define what are the responsibilities of the environment in supporting and using the HW/SW system. For instance, the environment has to provide input data and to understand the machine outputs.

Therefore, the analyst cannot be a passive recorder of the user’s wishes. Instead, he/she has to be active, make proposals, and explore alternatives. In fact, as described in next section, the responsibilities of the system and the interaction between the machine and environment are not always determined univocally by the user needs.

III. Case Study

The user needs that the information provided by subsidiary agencies via email is permanently stored at the central agency, in order to be available for subsequent elaborations when required.

Some analysis shows that we can specify different machines that solve the given problem:

a) Email messages are read by a (human) operator that feeds manually the information to the system via suitable forms.

b) Email messages are read directly by the system, which extracts the relevant information and stores it in the system. Of course in this case the messages have to be suitably formatted (this is environment responsibility). The system checks for format errors and an operator fixes errors “manually” (as in point a).

c) Email messages –suitably formatted– are read by the system, which extracts information, etc. In case of error, the system sends automatically to the sender an email message containing a diagnostic and an invitation to resend a correct message.

The characteristics of the environment and the knowledge of software systems suggest that the three possible solutions have quite different properties:

Solution a)
It minimizes the software development cost, since it does not involve the elaboration of the email messages. Instead, the operating cost could grow excessively high if the number of messages is big. Several people could be necessary just to read the email and perform data entry. This observation suggests that the cost of the solution is not equal to the cost of the software; actually, a lot of work and resources are often necessary to let a computer-based system work properly.

Solution b)
It requires fewer operators, since all the correct messages are managed without manual processing. It requires that the system is equipped with both automatic and manual input management; thus, it is more expensive to develop than solution a). It is viable only if it is possible to assure that the subsidiary agencies actually adopt the required email format. Also this solution implies additional costs: if emails are generated automatically by the agencies, updating their systems could be required; operators of the subsidiary agencies could need proper training on the new formats; etc.

Solution c)
It involves no manual input management. Instead, it
requires automatic error management, which is not present in solutions a and b. The machine development is therefore more expensive than in solutions a) and b). Messages not correctly formatted can cause delays, for instance if the subsidiary is in a different time zone, or if the diagnostics are not clearly understood, etc.

Given these differences, the analyst and the user have to evaluate together the different possibilities, in order to identify the best solution.

IV. DISCUSSION

A. Specifying requirements is a design activity

The example reported in Section III shows that defining the requirements is a design activity, as it involves discovering alternatives, making decisions and defining how the environment has to behave (i.e., what has to be done in the environment) so that the combined effects of the machine and the environment solve the user’s problems.

With respect to the development of the machine, the design activities are just in a different context: the development of the machine (e.g., the “design” phase) concerns the internals of the machine, while the definition of the requirements concerns the responsibility of the system and its interaction with the environment, so that the sum of the system, the environment and the interaction results in a situation that suits the user.

In several software engineering books it is written that Requirements specifications indicate what has to be done, while Design documents specify how to do it. This is true only for the HW/SW machine. On the contrary, if one considers the union of the machine and the environment, also the requirements specifications contain elements of the solution: how the environment has to behave, how the machine and the environment interact, how the machine behaves at the interface.

The concepts informally expressed so far are described in a precise way in a reference model for requirements and specifications [5] and in the literature on Problem-Oriented Software Engineering (see for instance [12]).

B. Email storage: user needs

In our example the user needs are that “information provided by subsidiary agencies via email is permanently stored at the central agency, in order to be available for subsequent elaboration, when required”.

Such needs can be expressed by means of a problem diagram as illustrated in Fig. 2. Actually, the figure does not represent a proper problem diagram: in fact the cloud represents a part of the diagram which is left unspecified.

Note that also the Central agency information processor could be part of the IT solution to be developed. However, in this paper we concentrate only on the email storage problem.

It is possible to note that user needs have a set of specific characteristics:

- They concern a given part of the problem (in this case, the subsidiary agency email client), which are given (i.e., they cannot be modified).
- They concern a designed domain (the Email repository): this is the part of the solution the user is really interested into.
- They do not directly concern any machine behavior: the user does not actually care about how the goal is achieved.

The latter point explains the presence of the unspecified part of the diagram: there needs to be something (or someone) that feeds the repository, but the description of such role is not part of the user needs.

In order to move from user needs to user requirements, several details have to be clarified. In our case, the following facts are known (some are just simplifications that allow limiting the size and complexity of the example):

- All the email servers and clients work well (i.e., without failures).
- Every message contains a issuing time, sender, recipient, subject, information content. The information content is structured.
- Every valid message sent to the central agency is stored as a record <issuing time, sender, recipient, subject, information content> in an email repository. The message has to be stored no later than one working day after it has been received by the server.
- A message is valid if some known completeness and coherence rules are satisfied.
- Invalid messages sent to the central agency must be corrected (possibly with the intervention of the sender) and then stored. The message has to be stored no later than one working day after its correction.

A more detailed representation of the user needs is given in Fig. 3. Here the elaboration requirements are specified; in fact, the user knows what elaboration is needed, and the repository has to comply with these requirements (since the email repository is designed, the data that is stored in it must comply with the requirements posed by the elaborations to be performed, which is the very reason why the repository is needed).
C. Email storage: requirements

As discussed in Section III, there are several possible organizations of the environment and machine that satisfy the user needs. The alternative (at least a few of the possible alternatives) are explored in the following paragraphs.

1) User requirements a)

Here we describe a solution based on the fact that Email messages are read by a (human) operator that feeds manually the information to the system via suitable forms. Therefore, the responsibility of the machine is just getting the information that is supplied through the forms and storing it in the repository.

This situation is described by the diagram in Fig. 4. Note that the cloud that was present in the diagrams representing the user needs has given place to the machine (the Data input processor) that has to perform the work. The data input requirements specify that it is responsibility of the Data input operator to get the email messages from the Email server via the Central agency email client with a given frequency and then feed the messages to the Data input processor. The data input requirements specify also that the data provided by the operator are stored in the repository in a format suitable for the subsequent elaborations.

The specification of the user requirements and the given domains should be detailed and precise enough to support correctness arguments, according to the problem frames methodology [1]. In practice, it should be possible to show (in a more or less formal way, depending also on the notation used) that the combination of the user requirements and problem domains imply the satisfaction of the user needs.

If the original need is something like “a message sent is stored in the email repository”, a correctness argument could be organized as the following sequence:
1. A valid message is sent to the server.
2. The server stores the message (behavior of the server domain).
3. The operator periodically checks for new messages (designed behavior of the Data input operator).
4. Upon request by the operator the message is retrieved by the operator (behavior of the Central agency email client).
5. The message is valid (required check by the operator) and is therefore stored in the repository (user requirement).

Thus, the behavior achieved according to the user requirements is equivalent to the original needs.

Note that correctness arguments are usually used – according to the problem frames methodology [1] – to show that the combination of the problem domains and machine specification imply the satisfaction of the user requirements. Here we are interested in showing that a correct specification of the requirements and domain behavior imply the user needs. No machine is (yet) involved! Of course we know that the requirements (point 5 above) will be satisfied by means of a machine (as shown in Fig. 4), but we do not really need to involve the machine in the demonstration that the requirements imply the user needs.

2) User requirements b)

Email messages are read directly by the system, which extracts the relevant information and stores it in the system. Of course in this case the messages have to be suitably formatted (this is responsibility of the environment). The system checks for format errors and an operator fixes errors “manually” (as in point a).

This situation is described by the diagram in Fig. 5. Here, the cloud that was present in the diagrams representing the user needs has given place to the machine (the Validator and input processor) that has to perform the work. The Validation and input requirements specify that it is responsibility of the Operator and error handler to periodically activate the validation and input processor (it could be replaced by a timer or a similar automatic device, but here we assume the presence of an operator, for the sake of simplicity). The requirements specify as well the rules –that the machine has to check– that determine whether an email message is correct, that the correct messages have to be stored in the email repository (in a
format suitable for the subsequent elaborations), and that the rule violations have to be notified to the operator.

Also in this case we can show, by means of a correctness argument, that the requirements satisfy the user needs. In fact, if the original need is something like “a correct message sent is stored in the email repository”, a correctness argument could be organized as the following sequence:

1. A correct message is sent to the server.
2. The server stores the message (behavior of the server domain).
3. The operator periodically checks for new messages (designed behavior of the Operator and error handler); the message is retrieved by the email client (behavior of the Central agency email client).
4. The operator activates the machine (designed behavior of the Operator and error handler).
5. The message is checked and found correct and is therefore stored in the repository (user requirement).

Thus, the behavior achieved according to the user requirements is equivalent to the original needs.

An alternative scenario could imply that the operator corrects the message and reactivates the Validator and input processor, which results in the message being stored in the Email repository, thus again satisfying the user needs.

3) User requirements c)

Email messages –suitably formatted– are read by the system, which extracts information, etc. In case of error, the system sends automatically to the sender an email message containing a diagnostic (which is also an invitation to resend a correct message).

![Diagram of email storage solution c)](image)

This situation is described by the diagram in Fig. 6. With respect to Fig. 5, there is no longer an operator; instead, the Validator, corrector and input processor is in charge, according to the Validation, correction and input requirements, of:

- Periodically retrieving messages via the email client;
- Checking the messages for correctness;
- Trying to correct the incorrect messages according to the given correction rules;
- Storing the correct(ed) messages in the Email repository;
- Sending diagnostics to the originators of incorrect messages.

It is easy to see that correctness arguments for the requirements of solution c) can be built as in the preceding cases, to show that the user needs are satisfied if the requirements are satisfied.

V. NEEDS VS. REQUIREMENTS

A. The relationship between needs and requirements

According to [5], the relationship that involves the machine, the environment and the requirements is:

\[ W, S \vdash R \]

where W indicates the problem world properties; R indicates the requirements (what the customer needs from the system, described in terms of its effect on the environment); S are the specifications that provide enough information for a programmer to build a machine that satisfies the requirements.

With this description, it is not immediate to distinguish the user needs, which lay completely in the environment, and the requirements, which involve, to some extent, the responsibility of the machine.

In order to make the difference between needs and requirements explicit, we suggest the following model:

\[
\begin{align*}
R, W & \vdash N \\
W, S & \vdash R
\end{align*}
\]

where N indicates the user needs.

Highlighting the user needs is useful to stress that in general there are several formulations of the requirements that satisfy the user needs, in the same way as there are several different machines that implement the same specification, hence satisfying the requirements. In fact, separating the notions of user needs and requirements makes it clear that writing requirements is a design activity, since one has to choose how to define proper requirements that satisfy the user needs.

Now, we can reason about the origins of user needs. It is easy to observe that user needs are generally conceived in some sort of business process; more specifically, the user needs are usually functional to achieving some goal in the context of a business process. For instance, in our case study the need for having the email messages stored in the email repository could be determined by the goal of letting the central agency process the customers’ requests addressed to the subsidiary agencies. In conclusion, we may say that

\[ BP, N \vdash G \]

where BP is the knowledge of the business process and G represents some business goal.

It is interesting to note that also the relationship between business goals and needs is one-to-many; in fact, the goal of informing the central agency of the requests collected by the subsidiary agencies could be achieved in different ways: the need expressed in Section IV.B satisfies the goal if the knowledge of the business process includes the fact that customers’ requests addressed to the subsidiary agencies are sent to the central agency via email, but other ways of communicating the requests are clearly possible.

In practice we are reproducing the hierarchical nature of requirements: the business process has some goals, which determine some needs, which can be achieved if some
requirements are satisfied. On their turn, requirements are satisfied if a suitable interaction between a machine and the environment is achieved. This hierarchy corresponds to different abstraction levels in the knowledge of the environment:

- At the topmost level the formulation of the goal depends only on the knowledge of the business process (transmitting requests is functional to the business);
- At the intermediate level the mechanisms supporting the process are known: an email system is used, therefore we need to save emails in a repository of the central agency.
- At the lowest level we finally take into consideration the local organization of the process (i.e., the fragment of the environment that is involved) and the responsibilities of the machine. Therefore, knowledge of the availability and cost of the personnel, as well as the cost of the machine development is required.

The hierarchical organization described above is also important to properly define monitoring and measurement activities. For instance, at the topmost level one is interested to know how long it takes to a request to reach the central agency; at the intermediate level one is interested in knowing the speed and throughput of the email system; at the lowest level several different measures can be defined depending on the chosen requirements (e.g., if the situation described in Section IV.C.2 holds, interesting measures could be the percentage of messages considered incorrect by the machine and the percentage corrected locally by the operator).

We may conclude that requirements are contextualized needs. Since the contextualization may proceed by progressively refining the abstractness of our descriptions, we can derive a hierarchy of requirements.

**B. An analysis process driven by needs**

The observations reported above can also affect the analysis process: being able to tell the difference between user needs and user requirements can help structuring the development process more conveniently and effectively. In fact, when the user needs are known, we face two possibilities:

a) To devise a machine and a domain behavior such that needs are satisfied, or
b) To devise user requirements that satisfy the needs, and then devise a machine that satisfies the requirements (and, hence, the needs).

Option a) corresponds to defining the machine specifications S and W such that

\[ W, S \vdash N \]

Although possible, this operation may be difficult, especially if the needs are expressed at a high abstraction level.

Option b) is expected to be simpler, because it allows splitting the analysis phase into two steps:

1) Defining R such that \( R, W \vdash N \)
2) Defining the machine specification so that  \( W, S \vdash R \)

However, it must be noted that during the first step one should take into account the cost of implementing the requirements, i.e., one should consider the effectiveness and cost of the machine that satisfies the requirements. We can thus conclude that the analysis process needs to be organized into several activities. In a first step the various possible R and W pairs that satisfy N should be explored; note that in this phase we consider the possibility of modifying the environment as needed, thus W is not just a representation of the environment “as is”, but it could involve some costs (e.g., for hiring or training people). The second step consists in choosing one of the requirements definitions produced in step 1 and defining the specifications of a machine that satisfies such requirements. Also in this case the cost of implementing the machine and possibly modifying the environment to correctly operate the machine should be evaluated, in order to assess the viability of the solution. If the costs are too high, it may be the case of going back to step 1 and look for further alternative requirements definitions.

**VI. RELATED WORK**

Several authors have published requirements acquisition strategies that start from an analysis of goals (for example, [9] [10] [8]). In [6] Potts proposes a schema similar to story schemata to help analysts develop a limited set of representative scenarios.

Kujala et al. propose an approach to representing user needs and translating them into user requirements in industrial product development cases [7]. They used users’ task sequence diagrams and use cases to model the user requirements. Although Kujala et al. do not provide a precise set of definitions or a methodology, they highlight the importance of documenting requirements in a simple and representative way.

None of the papers mentioned above used problem frames for the representation of requirements. Cañete-Valdeón et al. propose a characterization of the concept of “value” of use cases based on the catalogue of frames for real software problems proposed by Jackson [1]. For each frame they discuss which results from the system could be regarded as valuable for the user. The proposed approach is possibly helpful in integrating the usage of problem frames with use case based requirements engineering, but does not help in the identification and management of user needs, or even higher level goals.

The idea of progressing from the problem to the solution via transformations is at the base of Problem-Oriented Software Engineering (see for instance [12]). Here a similar approach is suggested, but mainly focused on the very high level descriptions of problems, when the users’ goals and needs are described entirely in the problem world, and the machine specification can be seen as the target of the process.

**VII. CONCLUSION**

A few concepts to properly identify and describe user needs and requirements have been proposed.

User needs lie entirely in the user’s problem domain space. No machine responsibility needs to be specified. This appears quite evident when a problem diagram is used to model the
user needs: in fact, the part of the diagram that includes the machine can be left unspecified, as in Fig. 2 and Fig. 3.

There are several requirements definitions that can satisfy the users’ needs. The analysis process has to take this issue into account: requirements engineering is a design activity, since it must lead to choosing one of the several possible requirements definitions that satisfy the user needs, and one of the several possible machine specifications that satisfy the requirements. Moreover, the problem domains are deeply involved in the analysis: it is often possible (as in our case study) that part of the solution depends on the behavior of the domains (often humans) that interact with the machine.

User needs are themselves determined by the goals of a business process. This observation suggests that there are not just two levels (the needs and requirements levels); rather, we may have a hierarchy of requirements, where each level of the hierarchy corresponds to an abstraction level in the description of the problems and solutions. Dealing with this hierarchy calls for an articulated analysis process. In this process the problems frames and related methodology play an important role. First of all, problem diagrams can be used to model both needs and requirements, as illustrated in Section III. Then, in order to prove the equivalence of user needs and requirements, one can build correctness arguments [1], as also shown in Section III.

Future work involves the definition of a methodology for organizing the description of needs in the aforementioned hierarchy and a consistent process. In particular, since it has been shown that the methodology of problem frames can be successfully used in combination with UML [2][3][4], it will be useful to define UML-based representations of the needs and requirements, and a UML-compliant analysis process. Finally, the transition from goals to needs and from needs to requirements can be considered similar to the transformations at the base of Problem-Oriented Software Engineering [12]: formalizing the proposed concepts in the context of Problem-Oriented Software Engineering is thus another goal of future activities.

Finally, it can be noted that requirements models based on problem frames can be successfully used for measuring the size of requirements (some initial work in this sense is reported in [13] and [14]). The work reported here suggests the possibility of measuring the size of needs, thus getting an idea of the effort required to satisfy the needs. To this end, the possibility of establishing traceability relations between needs and requirements is also interesting, since in principle these relations could allow assessing the consequences of changes in needs in terms of changes in requirements, and, consequently, on the implementation. All these issues will be subject to further investigations in the future.

REFERENCES

ABSTRACT

Our interest is embedded systems validation as part of the model-driven approach. To design a model, the modeller needs to obtain knowledge about the system and decide what is relevant to model and how. A part of the modelling activities is inherently informal - it cannot be formalised in such a way to constitute a basis for automated model design. This does not mean that modelling has to be chaotic. We therefore propose an informal method that structures modelling activities. In this paper we will focus on one of the method ingredients - modelling guidelines. In the industrial case study we performed, we captured modelling steps and elements in a form of a modelling handbook. The goal was to make modelling more efficient by preventing next modellers re-inventing things, but also to preserve a modelling style recognized within company’s context. We show in detail what these re-usable modelling elements are, and how identifying them can be generalised for designing modelling guidelines in general. Finally, we compare our work with work of researchers that formalise problem analysis.

Keywords

model-driven validation; modelling steps; embedded systems;

1. INTRODUCTION

Modelling embedded systems as part of model-driven design is intended to increase our confidence that the system will behave as required. During formal verification we design a mathematical model of the system and formalize requirements for the system’s behaviour in terms of this model. It is possible that the control software and the plant (the rest of the system) already exist, or we verify the system while, at the same time, designing its control.

There are many languages and tools for formal verification, but one is left without a technique how to use them [7]. In other words, there is no technique that guides us while designing a model. This results in having models of different quality depending on modeller’s experience and talent.

Modelling itself cannot be formalised in such a way to constitute a basis for automated model design. While designing the model, the modeller has to obtain knowledge about the system and about its requirements. Furthermore, she has to identify what is relevant to describe with the model and how to describe it. Also, we cannot formally prove that the model is an accurate representation of the system (and vice versa).

Our research is focused on designing an informal modelling method that improves confidence in the model and efficiency of modelling. Even though parts of modelling are informal, they need not be chaotic. By making them explicit and providing guidelines we make them accessible and controllable so that we can evaluate them. We identified three informal modelling aspects important to address and that usually stay implicit while modelling. They are: (1) modelling assumptions that are not part of the model [11], (2) steps that guide modelling and make it more uniform and structured, in other words make it an engineering process; (3) the argument that the model correctly represents the system [13]. We are building techniques to explicitly address these aspects.

In this paper we will focus on the second modelling aspect - steps and guidelines that structure modelling process. Our research question is: What can be generalised and extracted from modelling steps in one particular case and later reused in form of guidelines, checklists or useful techniques?

Through the description of one of our industrial case studies, we will show re-usable modelling steps and elements we captured in a form of a modelling handbook. The company where we performed the case study designs inserters - machines that automatically fold papers and insert them into envelopes. We analysed an inserter emulator - a digital circuit model of inserter’s mechanical part (or “plant” as it is called). The specification of the emulator is the plant’s
model as well. It is designed by the modeller and then automatically transformed to emulator execution. We designed a handbook for specifying emulators of future generations of inserters. The goal of the handbook was to make modelling more efficient, but also to preserve modelling style recognized among the model stakeholders. To be able to answer our research question, we generalised the results from the case study in form of classes of reusable modelling elements and decisions.

The rest of the paper is organized as follows. Section 2.2 explains the terms and concepts we are using as well as problems we tackle and research questions we start from in our research. Section 3 describes the method we propose. In Sections 4 and 5 we describe the case study we performed. Section 6 gives the relation with other researchers work in this area and builds on Sect. 7.

2. PROBLEM DESCRIPTION

2.1 Basic Terms and Concepts

2.1.1 Embedded systems

As shown on Fig. 1, an embedded system consists of a controller and the rest of the system, called the plant by embedded systems designers. The controller consists of the control software running on a special purpose hardware or a PC computer. The controller observes its environment via sensors, and enforces the plant’s required behaviour through actuators. We will abstract away the controller hardware and focus on the interaction of the control software with the plant.

For embedded system modeling, it is important to determine where the boundary of the system we are analysing is [19]. The environment of the controller, relevant for the model and the requirement, usually goes beyond the plant. There are users and operators that for example press buttons, products manipulated by the plant like paper in the printer, bottles in a bottle filling machine or chemicals mixed or extracted in a chemical plant. In wireless devices signals are transmitted through air, so the signal’s frequencies and intensity can be relevant to analyse when verifying these systems. One of the problems our method deals with, is the description of the controller environment in a verification model.

2.1.2 Verification Argument

To verify the system means to give a correctness argument that the plant (P) and the control (C) together satisfy the requirement (R).

\[
P \land C \implies R \tag{1}
\]

Using formal methods to build the correctness argument increases our confidence in the system. If the control is not designed yet, verification models can be used to help the control specification design. In that case the modeller’s task is to examine the problem, requirements, and the plant, and then create the solution in form of control specification. If, on the other hand, the control specification already exists, then the modeller has to additionally learn about the control, too.

2.2 Making Modelling Systematic

To design a model, it is not enough to learn a modelling language and get familiar with the tool that supports it. Before creating the solution (the model), the modeller has to analyse the modelling problem. She has to obtain knowledge about the system and the requirement, to decide what is relevant to describe with the model with respect to the requirement, and to decide how to model it.

The control software and the plant work together to satisfy the system requirements, but the requirement refers to the plant [8]. To decide what is relevant to model, the modeller needs to obtain and integrate knowledge from different domains, like for example electrical, mechanical etc.

Both analysis of the modelling problem and model construction are informal activities. This is in contrast to formal techniques that manipulate the model. Using them, we get a mathematical proof that the model has (or not) a certain property. If it does, we conclude that the system has it, too. But we do not have proof for this conclusion, we can only be confident in this. Also, we cannot automate the modelling process. This makes model quality and efficiency of modelling dependent on the modeller’s talent and experience. We want to improve this situation by making modelling a structured, systematic activity.

Our goal is to make the modelling process less dependent on the modeller, her experience, talent and creativity. We also want to increase confidence in the model by having the modelling process more structured. We are looking for steps and principles that can be captured and reused to guide modelling.

To build our method, we need to answer the following research question: What are the elements of the modelling process that can be generalized and (1) used to explain the existing model and argue its correctness, (2) transformed into guidelines that teach others how to model and (3) exploited when maintaining or redesigning the model? In our research, we focus on the problem of the plant description (modelling).

3. TAXONOMY OF MODELLING DECISIONS

In her earlier work, one of the authors proposed a taxonomy of modelling decisions [10]. Its purpose is to provide understanding of modelling as informal activity that leads to a formal verification model. Having better understanding of modelling, being aware of its structure, already structures the way we think and work while modelling. But, the elements of the taxonomy can be also articulated as questions to answer while modelling, or as a checklist of activities to perform. At the same time, it is the list of questions to explicitly address and further refine. The elements of the tax-
The purpose of the model is in our case to verify the requirements, but verifying the requirement is, as we will show later on the example of our case study, part of broader verification process. Also, at a different level, there are a number of requirements (quality criteria) for the model. For example, we might require that the model is traceable in a sense that it follows the existing decomposition of the system as much as possible. Additional limitations and requirements on how the system will be modelled are imposed by pragmatical issues like time, money and resources present. This is not addressed in the academical environment, but in practice it is not possible to neglect it.

Before identifying relevant parts, it is necessary to define what the object of modelling is. Is it a specification of the control, or the system requirement? Is it the algorithm that we have to verify or the control that implements it? There are differences between modelling these, and often they are not clearly stated.

While analysing the problem, the modeller also decomposes and structures it, to better understand it. Further decomposition is needed to find out what aspects of the system are relevant for the model and what parts and their properties should be represented in the model.

The modeller decides on tools and languages to use and the mathematical domain in which the system is described. Also, the modeller structures (decomposes) the model which can coincide with the system decomposition he identified. Representing the system with the model means that different abstractions, simplifications and idealizations are done. They, of course, should be done in a way that the model still represents the system with respect to the requirement. For example, if we do not need to prove a timing requirement, we can model plant’s actions as the actions that do not take any time.

The model does not contain only the elements that represent the system. Often, to prove the requirement, additional elements are needed and they depend on the modelling solution, and languages and tools used. For example, in a model of a printer we possibly need an extra variable to count number of papers printed, although there is no any counter in the printer.

As we will show on the example of the work on our case study, the taxonomy can be used while modelling, but it is even more useful to apply it to the context and to refine the modelling steps given.

Applying the steps from taxonomy is part of the method we propose. As we will show on the example of our case study, these elements are helpful, but it is even better if we further refine these steps. Further refinement produces steps that are depending on the actual problem.

4. DESIGNING A MODELLING HANDBOOK - ACTION CASE

4.1 The System Description

We performed field research (action case) in a company [14][15] that develops, produces and distributes different types of mailroom equipment and document systems. The users of document systems are companies that send a lot of paper mail on a daily basis, like insurance companies, post offices, banks etc. One such machine, called inserter, is shown on Fig. 3. The inserter automatically folds paper sheets and inserts them in envelopes.

The inserter works as follows. The operator places documents (paper sheets) and envelopes in appropriate feeders. He also specifies recipes for manipulating documents, through the user interface. A recipe defines how many documents will be inserted in each envelope, whether the paper sheets will be folded and how (for example, a document can be folded along the half or along the third of its length). An example of a recipe is: “Select three documents from the first feeder, make a “Z” fold and insert them into the envelope.” Recipes form scenarios, e.g. one scenario would be to repeat the previous recipe hundred times.

The diagram in Fig. 4 describes inserter modules where...
different processes take place. The arrows in the diagram represent the flow of the documents and envelopes through the system. In the feeder, rollers (small plastic wheels) rotate along the surface of the document on the top, which pulls it out of the feeder. The document is then moved toward the collator, again with rotating wheels. The wheels are placed in pairs on each side of the document path. In the collator, the documents that form a batch to be inserted in one envelope are collected and straighten out (collated). The batch is then moved to the folding position. It is folded with a stroke of a long, thin, sharp-edged arm. In the meantime, the top positioned envelope in the envelope feeder is pulled out of the feeder, and brought to the flap moistener. Its flap is first turned up and then moistened with a stroke of a brush. The upper side of the envelope is lifted, so that document batch can be inserted. Folded documents are then inserted in the envelope, the flap is closed and the envelope with documents is moved toward the exit of the machine.

Every process performed on documents and envelopes is the result of controlled movements of mechatronic parts. Rollers, folding arm, brush and many other system parts are connected to actuators that move them. Along the system, sensors are placed to signal the presence or absence of material in front of them. Based on sensor readings, the control software sends signals to actuators.

The high level system architecture is shown on the left side of Fig. 5. The System Controller handles operator requests and recipes and forwards them to the Embedded Controller. Embedded Controller controls the plant.

4.1.1 Plant and Control Software Integration

When a new generation of inserters is designed, some mechanical parts, or even whole modules, are re-used from the old inserter generation. Some parts, on the other hand, are completely re-designed. Radical design results in significant improvements of functionality or performance which is important for competitiveness on the market. At the same time, it brings unanticipated problems and issues that did not exist before [18]. Some of these problems are related to the integration of the control software and the plant. If the integration comes in a later stage of the project, solving these problems might need rework of parts, whole modules or concepts which causes delays.

The plant and controller together deliver the inserter's desired behaviour. It is therefore important to enable their concurrent development. When the control software and plant are designed at the same time, mechanical and software engineers communicate from early development stages, and problems related to plant and control software integration arise early enough to be resolved on time.

The problem is that in the early development stages the plant exists only in sketches and CAD drawings. The interaction with the plant via sensors and actuators is in the essence of the control software, therefore without the plant control software testing is not possible.

4.2 The Plant Model

To enable concurrent engineering of the plant and the control software from early development phases, the company designed and built the plant emulator for the purpose of control software testing. Figure 5 shows the comparison of the inserter architecture and the architecture of the testing setup. The borders of the control hardware are the interfaces that exchange signals with sensors and actuators. The emulator is connected to these interfaces. On the emulator side, the interface connected to the controller mimics sensors and actuators behaviour.

The emulator is a programmable hardware device, which means that it consists of digital circuits that a user can configure and combine. These digital elements are connected and they together mimic the intended future plant behaviour - the flow of documents and envelopes, rolling of rollers etc. We can say that the emulator (M1) is a digital signal model of the plant (P).

There are two more models that play a role here - the specification of the emulator (Φ) and the run-time visualisation of emulator’s signals (M2).

The specification of the emulator’s digital signals behaviour is at the same time the specification of the plant behaviour. The specification is written in a graphical language, called G-language, using the LabView tool. LabView diagrams describe the flow of material, the sensor reaction to the material and movement of different physical parts. The LabView compiles the diagram-based specification to the program for
the hardware. We consider this transformation correct, and deal with the graphical model of the system.

For visualisation purposes, the diagram that shows the states of emulator’s digital elements is designed (part of the LabView package are also run-time diagrams for hardware monitoring). To be more precise, only those digital elements representing sensors are collected. Their order in the diagram corresponds to the spatial order of the sensors in the inserter. This is convenient for the testing engineer who monitors signal flow in the emulator as if it were document flow in the plant.

We can therefore argue that we have three models. One is the physical, digital signal model of the plant. The other is the graphical model, the diagram that is designed to represent the plant. The third is the run-time visualisation model. We can say that: \( M_1 \models \Phi \), \( P \models \Phi \) and \( M_2 \models \Phi \).

The model we analysed is the second one - the graphical specification of the emulator (\( \Phi \)). The reason for this is that we are interested in designing the model that represents the plant for the purpose of control software testing. Our questions are related to gathering knowledge about the plant, and deciding how to describe it. In this case, the model designed to represent the plant is the emulator’s specification diagram. The programmable hardware (emulator) is just the execution of the specification, in a way similar to the computer hardware’s execution of a computer program. The model could have been just as well implemented as a simulator, describing the plant with software; for our research question this would not make any difference.

4.3 Design Task
As we already explained, the plant model already exists. Designing it for the first time took significant time and effort. The modeller spent a lot of time learning about the plant, trying out different modelling solutions and finding the optimal one. He did not document his insights and knowledge progression, due to time-pressure. The model-based testing improved communication between departments and shortened the integration time, so it will be used in the future.

How the new, future generation of the system will look like, when it will be developed, which parts will be incrementally and which radically designed, is not known yet. The modeller will most probably be another software architect as the original modeller has already left the company. In fact, the modeller may be someone from outside the company. Finally, the tools and languages used now, may not be the choice next time.

To make modelling efficient next time the model is designed, and to preserve existing modelling style, we designed a modelling handbook.

5. THE HANDBOOK DESIGN

5.1 Solution Approach
We designed the modelling handbook, or in other words, we designed a modelling method suitable for the concrete modelling problem. The left hand column in the table shown in Fig. 6 shows the modelling steps we propose in the handbook. The right column in the same table shows our solution approach for this case or how we designed this method.

For modelling handbook design, we had to find what elements of the model and modelling process can be extracted and turned into clear, relatively short modelling guidelines.

<table>
<thead>
<tr>
<th>Modelling method suited for the problem</th>
<th>Steps to design this method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform refined taxonomy steps to understand the problem</td>
<td>Apply the taxonomy on the existing models</td>
</tr>
<tr>
<td>Apply re-usable modelling solutions</td>
<td>Look for categories of modelling solutions in the existing models</td>
</tr>
<tr>
<td>Re-use previous model parts</td>
<td>Identify what parts of the system will not change</td>
</tr>
</tbody>
</table>

Figure 6: Modelling method and steps to design it

- We applied the taxonomy of modelling decisions [10] developed in our earlier work.
- We interviewed the modellers about their modelling decisions. More precisely, we asked for the rationale of their modelling choices and decisions.
- We interviewed model stakeholders about requirements for the model and about the system.
- We analysed the model and asked model stakeholders to estimate which parts will not change in the future.

Due to lack of space, we will show only a couple of elements of the handbook elements as illustration of our points. The complete handbook (without the confidential parts) can be found in the appendix of our case study report [12].

5.2 Applying the Taxonomy
We performed the analysis of the existing model using our taxonomy. When applied to concrete case, we were able to

- refine modelling decisions listed in the taxonomy
- examine if and how these steps showed while designing the model and
- examine if and how these steps could be seen in the model.

Some of our findings are the things that are not changing, they are the part of the whole modelling context in the company, and by context we mean the way the things are viewed, designed, documented and organised. Most likely they will stay the same next time the model is designed. So, documenting them may be useful to start from next time the modelling task for emulator is defined.

5.2.1 Identify the Purpose of the Model
The high-level purpose of the emulator is to facilitate testing of the control. The initial idea was to use it until the plant is made, but the emulator turned out to be useful for continuing some of the tests even when the plant is already there.

We classified verification (testing) purposes as shown in table on Fig. 7. We also listed the faults and irregularities that should be taken into account while modelling. They are: broken sensor, paper sheet slipping (not moving) under the roller, and lengths of paper sheets different then specified. The broken sensor can be always on or always off. Finally, the last thing to address is: what kind of system limitation do we want to test with the model.
<table>
<thead>
<tr>
<th>Control testing</th>
<th>Plant testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing all possible recipes</td>
<td>Testing limitations of the specification</td>
</tr>
<tr>
<td>Regression testing</td>
<td>Testing impact of changes in the software behaviour</td>
</tr>
<tr>
<td>Endurance testing</td>
<td>Testing how small changes reflect in system behaviour (e.g., moving sensor)</td>
</tr>
<tr>
<td>Fault and irregularities tolerance</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 7: Purpose of the model](image)

Different purposes mean that model describe different plants - one with a certain combination of faults, or one without it. Or a plant with a sensors placed on different distances between each other. Instead of having different, but very similar models for each of these purposes, the model was parametrised. Parameters reflect different faults, different position of sensors and different irregularities in documents length.

5.2.2 Identify the System Requirements to Verify/Test

The system requirement we are interested in is the correct system behaviour - inserting of documents into envelopes, according to the recipes that operator defines. What was verified was that the emulator and control software were correct wrt to the requirement.

There are different modes of work (one recipe and optimisation of many recipes) and for these two, the control behaves differently. For this, the simulator of the system controller is designed to provide the control with different recipes and scenarios.

5.2.3 Identify Quality Criteria for the Model

We explored if the quality criteria we identified were relevant when designing the model and if there were other criteria we did not mention before.

Our basic criterion is that the model should be **truthful**. It means that it has to represent the system correctly with respect to the requirement verified or tested. In the emulator case, the modeller was aware that the model was not truthful, and that there are some things that will show up once the plant is produced. The emulator does not eliminate the need to test and refine the system in the integration phase, it just makes it shorter. It would be good to know if the model is ‘truthful enough’ to prevent discovering major design errors in the later development phase, but this cannot be guaranteed. Another source of deviating from truthfulness is more benign. Some parts of the model are oversimplified because of lack of hardware resources. This causes some of the tests giving wrong results about the system. However, these modelling decisions were chosen in such a way that further analysis of testing results leads to valid results.

If we want the model to be **understandable** we also have to specify for whom it has to be understandable. For example, the model might be completely clear to another modeller and unclear to a software engineer. The main users of the emulator are testing engineers. They used not the emulator directly, but the LabView model. As we explained earlier, the LabView model is at the same time specification of the emulator hardware and the representation of the emulator during run-time. The model consists of a number of diagrams, each representing one part or one process of the system (emulator). The modeller designed a visualisation model that shows only the main signals in the emulator.

This model is at the same time a simulation model. Testing engineers use this model, and in case of problems they examine further a diagram where the source of the problem is. The modeller was always present and available to assist analysis and changes of the model. This may not be the case in the future. The role of the modeller and testing engineer might be separated. So, the more of the modelling style is used in the next model, the less time testing engineers have to spend analysing what is what in the model.

A model is **traceable** if the structure of the system can be found in the structure of the model. It is traceable in the other direction if the fault in the model can be traced to the specific part of the system. The emulator specification followed the decomposition of the system into modules. This way, it was also possible to design a visualisation model, to be able to understand the testing results. Everything we said about testing, when talking about understandability holds here, too. The traceability was in conflict with the model simplicity in the part of the model describing document merging together after they left different feeders. When looking at the model, we may think of different layout of feeders. At the same time, the document flow still describes the flow in the existing plant.

When specifying that the model has to be **small** in case of model-checking, this usually means, simple enough to be manipulated with computer-aided tools and avoiding state explosion. In the emulator case, it was not processing time that had to be considered, but the number of integrated circuits that constitute the emulator. They are placed on integrated circuits boards, and they are expensive. Unnecessary complexity increases modelling time, maintenance time and also makes the model less understandable. For example, the calculation that involve multiplications need a lot of digital elements. That was one of the reason why the motor in the system was not modelled, but kept as part of the system. It was cheaper to have it there then to model it.

Having the model **simple** is closely related to its understandability. Simplicity was not quantified here in form of number of maximal elements in a diagram or maximal number of states. It was the subjective decision of the modeller to dismiss modelling solutions that might have a smaller model, but it would make the diagram difficult to read.

Finally, the new criteria we found while analysing the model was the **degree of software independence**. Even though the plant model should be as much as possible independent from the software specification, this in practice would mean that the model would be too complex, too big and incomprehensible. Certain mechanical parts were designed knowing that, for example, that the control will not make the full rotation of the folding arms, which would cause it crashing into another part. The software dependence might be seen as another source of model’s untruthfulness. But, where the modules are reused, and where also the control is reused, it makes sense to have this dependence. For example, if the same feeders are used and the same control that takes two documents from the feeders is used, and if the control and feeders behave in such a way that documents either overlap at least on one third of their length, or do not overlap at all - if this was already tested in previous systems, and if it
would be too complicated to model all possible overlaps, it is reasonable to make the assumption that the control will behave in the way we explained.

5.2.4 Find Out Pragmatic Aspects of the Model

The most obvious pragmatic aspect is that the model has to be as cheap as possible, and be designed in the shortest possible time. This was reflected in the previous criteria we talked about, like not modelling the motor but having it as part of the testing setup, oversimplifying some model elements etc.

Other pragmatic aspects were the knowledge and experience of software engineers. This influenced the choice of the modelling technique and tool. The software engineers were not experienced in using hardware programming language like VHDL, but had experience in LabView programming. This was one of the reasons to choose the second tool.

Another reason to choose LabView was that it was a diagram based. The goal was to avoid a language that looked like programming, because software engineers would start to think of events in the model as the events in the control execution, rather than events in the plant.

5.2.5 Decide What the Object of Modelling Is

The object of modelling are physical parts of the plant, processes performed on documents and envelopes, and the flow of documents. At the first glance, it may look like the emulator is the model of sensors and actuators behaviour only. However, the emulator is more than that. Its signals represent the flow of documents and envelopes, movement of different physical parts and faults in them.

Where do we stop decomposing elements, what is the granularity of the model? In the emulator model, the actuators and physical parts performing functions on the paper sheet were described. Where no faults of physical parts were reflected, the model stopped decomposing on a previously defined functional decomposition.

Another object of modelling that we do not analyse here are user recipes and scenarios.

5.2.6 Decompose the System to Model it

One of the model purposes, besides representing the plant for control testing, is the communication between domain experts. The purpose of the testing is to integrate control and mechanical plant, and modelling is the way to give feedback to plant and control designers.

While modelling, it is sometimes necessary for the modeller to check with different domain experts if the model is correct. If the model follows commonly accepted decomposition of the system, this makes the task easier.

One of the wide-spread decompositions is the one to physical modules and the processes they perform on documents. This decomposition is reflected in how the design, development and testing tasks and responsibilities are assigned, how the teams are organized and how communicate with each other. This decomposition is also reflected in project documentation available to everyone.

The model follows this decomposition whenever possible. The fact that the system is highly modularised helps here. The model is decomposed to diagrams that describe these modules. There are parts that perform different functions and belong to two modules, in this case it was up to the modeller to place them in one of the model components.

<table>
<thead>
<tr>
<th>Path element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Papers</td>
<td>Length, thickness and deviation of the defined length are relevant</td>
</tr>
<tr>
<td>Segment</td>
<td>The path of the paper sheets in the inserter is divided in segments. The segment begins at the sensor position. The segment ends before the next sensor. There are one or more rollers on each segment.</td>
</tr>
<tr>
<td>Merging point</td>
<td>This is the place where documents coming from different feeders cross each other and merge.</td>
</tr>
<tr>
<td>Selector</td>
<td>From this point, two different paths are possible.</td>
</tr>
<tr>
<td>Sensor</td>
<td>A sensor is on when there is a material in front of it. It is off when there is nothing in front of it.</td>
</tr>
</tbody>
</table>

Figure 8: Paper sheet path elements described in the model

5.2.7 Decide on Mathematical Domain, Language and Tools

We addressed this issue when talking about the model’s pragmatic aspects. We only did not mention that the plant behaviour is described with hierarchical state machines. This is a common way of describing reactive, embedded systems and is natural way of looking at systems and describing them.

5.3 Re-usable Model Components

As we shown in Sect. 5.2, we identified context dependent elements relevant when analysing the problem and that are less likely to change in the future.

We analysed the model to find re-usable modelling components. We did not document them in form of a pattern in LabView, because it is not sure that LabView will be used next time. Instead, we used state charts that both mechanical and software engineers understand.

The model has two very different components, one describes the flow of documents and envelopes through the inserter. It may happen that something change in the future, maybe different sensors will be used, or rollers that have to be described in more details. But certain aspects of the material movement will not change. Due to space limitation we only list some of these elements in the table on Fig. 9; all of them can found in the Appendix of our technical report [12]. We documented them in a form of short explanation, diagram that helps understanding their position in the real system (for this we used the diagrams that the modeller used) and as state machines. The state machines are accompanied with the vocabulary explaining what events and states mean in the system. As these things were written informally, they are not the absolute source of information. For sure, the modeller will have to talk about them with domain experts in the future. They are there to structure issues and elements that the modeller will have to address and as an instruction how to model them, rather then a ready modelling solution, or a pattern to initialise.

Besides these elements we documented possible faults. For example, it is assumed that the broken sensor will either be always on or off (it will not for example be off when it should be on and vice versa.)

The other model components describe the rest of the plant - mechanical modules performing different processes on pa-
Questions to answer when modelling the rest of the plant

| For a given function, is there more than one physical element that performs it? |
| Does examined physical element perform more than one function? |
| Does the physical component work in synchronisation with another physical component? |
| Is the duration of performing the function relevant? |
| What is the interface of a component with other components? |
| Is the material brought to a component by another part of the system or the component takes it? |
| Is the initial position of a component relevant? |
| Is the ending position of a component relevant? |

Figure 9: Purpose of the model

per sheets and envelopes, which are: folding and inserting documents, opening and sealing envelopes and wetting envelope flaps. Here, radical changes are possible, so we cannot assume for any of the design solution that it will stay the same. This is the part of the system where new design ideas are allowed and welcomed, because they make the system performing much better and give the inserter advantage on the market.

However, the processes performed on paper sheets will not change. They will stay the same, as well as their order. If we look at the rest of the model, not as the description of physical parts, but the processes they perform, we can extract model parts that do not depend on the physical characteristics and the position of mechanical parts.

One such thing is the algorithm of folding, which is already documented. It specifies the order of actions to perform on the paper sheet and the positions of folding movement across the paper sheet. To decide further if there are relevant physical characteristics to put in the model or to take into account, the modeller will have to know more about the concrete solution for folding module and the parts surrounding it.

Analysing the existing model, we generalised from modelling decisions and document questions that could steer modelling in the future. These questions are the result of the analysis what aspects of physical parts and their position were relevant in the current model.

5.4 Rationale of Modelling Decisions

We interviewed the modeller about the rationale of his modelling decisions. Some of them we already described when talking about the modelling decisions in the light of our taxonomy. It is more focused on problem analysis than on designing the solution, although sometimes it is difficult to draw the line between these two. For example, decomposition of the system can be seen as investigation of possible system decompositions or deciding about the model structure.

The taxonomy-based refinement of modelling elements, discovered us the things that do not change. Some of the modelling decisions that belong to the classes we list below will probably change. But, those classes give some directions how to model.

5.4.1 Following the System Decomposition

As we explained earlier, the model followed the existing system decomposition, wherever that was possible. We explained this in more detail in Sect. 5.2.6.

5.4.2 Exemption from Following the System Structure

Some parts of the model did not follow the system structure. More precisely, the part that describes movement of the material from the feeder is implemented differently. The feeders will not change in the future, so we documented this solution as relevant for the future, too.

5.4.3 Model Elements That Do not Exist in the System

A models does not consist only of the elements that describe parts of the system. There are additional elements that are, for example, needed for initialisation of the model or they are tool-related modelling tricks needed to draw conclusions about the model. Some of these elements may look like the description of the system part. In the model we analysed, we observed that there were 'sensors' in the model where they do not exist in the plant. The control software extrapolates position of the paper without a sensor in this particular case, but here the modeller needed the plant to be independent of the control software. The non-existing sensor shows the testing engineer the position of the paper, even though the control only calculates it.

5.4.4 Alternative Modelling Decisions

Some of the alternative modelling decisions were useful to document, because thinking of them does not reveal modelling difficulties. Only when trying these ideas out, it turns out that that model component becomes too complex, or not easy to maintain or not easy to follow the signals in the run-time. To prevent re-invention of modelling solutions, we sketched these alternative solutions and explained the difficulties with them. These were all decisions about the material path, namely they were about the length of the model segment and of the positions of sensors and rollers on them.

5.4.5 Knowledge and Insights of the Modeller

Finding out what plant properties are relevant for the model is a process of discovery taking place while the model is designed. We documented some of the modeller's insights. We also documented the properties that were not relevant for the current model, but were recognized by the modeller as subtle details that might be important in the future. For example, the rollers that move the paper flatten on the places that touch a paper, and this influences the peripheral speed of the roller, and therefore the paper.

5.4.6 Assumption on the control

Ideally, the emulator would describe all the possible plant behaviours. However, this would lead to a model that is too complex to understand, maintain and that will require a lot of hardware sources and modelling time. So, the task of the modeller was to find out those control solutions that will never change. For example, the folding knife will never rotate if the paper is not ready for folding. This is quite a strong assumption on the control, but it was good enough for the modelling purposes. We collected these assumptions,
even though some of them might be different in the future.

5.5 Solution Validation

5.5.1 Internal Validation

We designed the handbook in close collaboration with the modeller. We adopted industry as laboratory’ approach [16] which means that we do not invent a method and then look for the problems solved by our method. Instead, we perform action research and adopt our method to the problems we face. Therefore, we were performing shorter, iterative cycles of problem investigation, solution design, design validation, design implementation and implementation evaluation [20].

After the modelling handbook was designed we presented it to model stakeholders - heads of software and system departments, testing engineers and software architects. We interviewed them and asked them to validate the handbook. Software architects found it useful, the head of software department said they would use it for the future emulator. The testing engineer who was using the existing emulator said that for understanding the current model using the actual emulator was much more useful than reading the handbook. This is related to another approach of guiding the modelling, which would be using the existing model and maintaining it. This was possible at the moment, but in the future it might not be.

5.5.2 External Validation

How is our handbook useful for modelling in general? It showed that the taxonomy we developed earlier was a good starting point to analyse an existing model. The classification of concrete modelling decisions is classification of solutions that possibly can be applied in practice.

With this example we argue that it is possible to extract guidelines from the existing plant model of mechatronic systems. To give an answer to the question of how much the guidelines would be useful when starting modelling from scratch - this requires further research.

Another threat to validity is that the emulator is not a formal verification model in a sense we define a formal verification model. But, to build it, it was necessary to be precise and accurate the same way we need to be when designing, for example, an automata based model for model checking.

6. RELATED WORK

6.1 Techniques To Reuse Design Decisions

Our work has common elements with the idea of software product lines and software design patterns. Software product lines are a technique to reuse one model for a product that evolves or changes. Except that this is the technique to design the software and not the model of the plant or control, it also pays a great deal to maintenance and evolvability. For us this was not an issue, although it can be seen as a quality criteria for the model.

Software design patterns [3] [17] capture the solutions to common software design problems. A pattern is not a solution, it is a template, a rough-structure that has to be shaped to the concrete design problem. Software design patterns are invented by object-oriented community. A pattern consists of the problem description, context of a problem, specification of the behaviour (software requirement), solution description, resulting context and behaviour and what it does not do, and the rationale of the design. The reusable model elements we identified have a lot in common with modelling patterns, but we did not design a LabView pattern that has to be initialised with parameters that describe a future problem.

6.2 Problem Orientation

As we already mentioned earlier, our work has a lot in common with problem frames technique [8]. Starting from its views and principles we continued further in designing an informal modelling method. Other authors combined problem frames and UML to design control [2], combined patterns and problem frames technique for architectural decisions [1], or developed guidelines for designing a formal control model [6].

Another group of researches, including Jackson himself worked on formalisation of the technique [4] or on formalisation of recomposition of the sub-problems found after problem decomposition [9]. A community of researchers are working on formalising problem analysis and combining problem frames technique with formal techniques [5]. Formalising modelling steps give valuable insights on modelling and relationships between different modelling elements. Our interest is more on identifying these steps before formalising them. These steps are an engineer’s own way of finding a solution, but they usually stay implicit. They are usually not recognised by academia, because they are not (or cannot be) formalised. Our goal is to find these engineering steps and bring them up in order to cope with them in adequate way.

7. CONCLUSION

To summarise, we are developing a modelling method in order to improve efficiency of modelling and the quality of models. Our method is informal, it structures modelling activity and makes implicit modelling decisions explicit. In fact, in this paper we described modelling method on two levels. This is illustrated in table on Fig. 6. One is the modelling method for designing a model, suited for the problem in hand (shown in the left column of the table). In the case study we performed, we designed guidelines to model future inserters, by reflecting from the existing model.

On another level, we identified steps to design the first modelling method. They are

- Apply the taxonomy to analyse the existing model
- Identify re-usable model components
- For the system parts that will change, write down questions to which current modelling decisions were the answer
- Find out what model components do not follow the system structure
- Identify model elements that do not exist in the system
- Document alternative modelling decisions that might be relevant in the future
- Write down insights that were not relevant for to model, but are hidden in the domain expertise, and might become relevant
Write down the assumptions on the control.

It is not possible to separate these two, as they together form our approach for dealing with modelling decisions.

In comparison to the techniques to formalise problem analysis, we might be not precise as they are, or we might be missing a relationship between elements of our taxonomy, but this does not make our informal method wrong. For future work, we plan to compare the results of formalisation of problem analysis and if the two can complement each other.

7.1 Acknowledgements

We would like to thank Dusko Jovanovic from Neopost for useful comments on this paper and Peter Tjerdsma for insights about modelling the inserter plant.

8. REFERENCES


Using Problem Frames to Drive Layered Requirements Analysis

Jie Sun, Ye Wang, Xinyu Wang, Xiaohu Yang, and Aleksander J. Kavs

Abstract—The premise of Jackson’s Problem Frames approach is the previously collected knowledge of the problem context. Nevertheless, the identification and the appropriate description of this knowledge is not an easy job. Business process modelling is considered as a useful way to elicit and describe the problem context. However, existing requirements analysis approaches that map business process models to problem frames only deal with simple business processes that can be represented in one layer. This paper proposes an approach to layered requirements analysis that decomposes problems by mapping layered business process models to problem frames. A set of examples of requirements analysis for stock trading systems are adopted to illustrate our approach.

Index Terms— problem frame, frame concern, requirement analysis, layered business process model

1 INTRODUCTION

The Problem frame approach [1], as a useful way to capture requirements, is attracting an increasingly attention in the research community [2]. The approach is employed to analyze the requirements under the presumption of the previously collected knowledge of the problem context. However, the identification of problem context is recognized as a difficult task [3]. In order to alleviate this difficulty, Cox and Phalp pointed out that business process modeling is a good way to capture the problem context. They also developed a requirements analysis approach that derives problem frames from Role Activity Diagrams (RAD) [4] [5]. We believe that mapping business process models to problem frames would be particularly helpful for requirement analysis. However, there is one limitation of Cox and Phalp’s approach. It only deals with one layer RAD. Some large scale information systems, for instance stock trading systems, always have complex business processes that need to be decomposed into multiple layers. How to use problem frames to derive and model requirements from layered business process models is an unsolved problem.

A requirement is an end-to-end constraint on phenomena from the problem world, which are not necessarily controlled or observed by the machine [1]. Therefore, in order to completely analyze requirements, it is quite important to ensure that the machine will indeed enforce the requirement. Frame concerns describe the correctness argument of a problem frame by combining together the requirement, the specification and domain properties.

Analyzing the frame concern can structure the way in which the machine specification will work to ensure the requirement. Hence, we believe that frame concerns analysis is an essential part of requirements analysis, although Cox and Phalp’s approach does not involve this aspect.

This paper, therefore, aims at guiding requirements decomposition by mapping layered business process models onto problem frames. Meanwhile, frame concerns are analyzed to completely analyze and model requirements in the context of different problem frames. In order to illustrate the utilization of this approach, we show some examples of requirements analysis for stock trading systems. The stock trading processes are modeled via an integrated business process modeling approach composed of two sub-models – Process Flow Model (PFM) and Role Cooperation Model (RCM) [6]. Both of the above sub-models can have multiple layers.

This paper is organized as follows: section 2 provides the background knowledge of PFM and RCM; section 3 describes our approach through a set of examples of the stock trading requirements analysis; section 4 discusses some implications for our requirement analysis approach; section 5 briefly introduces others’ work about problem frame and section 6 concludes the paper.

2 OVERVIEW OF PFM AND RCM

To represent business processes, we adopt the integrated business process model with two sub-models – RCM and PFM, instead of RAD that is chosen by Cox and Phalp’s approach. The reason is that RAD mixes roles within process flows. This results in a cobweb of each process model and makes processes incomprehensible. On the contrary, RCM and PFM respectively represent the two complementary process views: the role-based view and the flow-based views. As implied by names of the two sub-models, PFM models the aspect of process flows ordered by process steps involved in a business process, while RCM models cooperations and responsibilities performed by roles involved in the same business process. Both PFM and RCM
can have multiple layers, and each layer is refined by the one directly below. In order to relate the two sub-models, there’s a mapping from RCM to PFM at each layer.

A three-layer PFM example is shown in fig. 1, which describes a part of the stock trading process. PFM notations are on the basis of UML2 Activity Diagram [7]. The ProcessFlow stereotypes extend from the Activity metaclass of UML. A process flow consists of actions, data object, control nodes and other process elements. An action is denoted by a rectangle. A data object produced during a process flow is denoted by a rectangle with round corners.

Actions of a process flow can be refined by process flows of the layer directly below. The tagged value – refinedNodes – refers to the action that is refined by the current process flow. In the following example, the top-layer process flow consists of four actions – Order Entry, Order Matching, Trade Printing and Trade Allocation. Order Matching is decomposed to the second layer, followed by a further decomposition of Price Order to the bottom level.

RM notations are defined by extending the stereotypes of UML2 Class Diagram. There are five stereotypes – RoleCooperation, CompositeRole, Role, Interaction and Responsibility. Responsibility is represented as a function of a class, which is owned by Role. RoleCooperation consists of Role and Interaction. CompositeRole is a conceptual role and inherits from Role, which actually includes two or more physical roles. The tagged value – includedRoles – specifies the roles included by RoleCooperation. CompositeRole is refined by RoleCooperation in the layer directly below and the cooperation is carried out between the included roles. The tagged value – refinedCompositeRoles – refers to CompositeRole refined by the current role cooperation.

To relate RCM with PFM, the tagged value – referencedProcessFlow – is used to map the current RoleCooperation to ProcessFlow. Besides, each role’s responsibilities are one-to-one mapped to actions of the corresponding process flow.

Fig. 2 A two layer example RCM of stock trading processes.

3 REQUIREMENT ANALYSIS APPROACH

In this section, we illustrate our requirement analysis approach through the examples of stock trading systems in section 2. Layered requirement problems are derived from PFM and RCM, and then are mapped to problem frames with frame concerns.

3.1 Starting from PFM and RCM

As in section 2, the mapping from RCM to PFM starts from the second layer process flow, namely Order Matching. Correspondingly, we derive and analyze requirements from the two business processes – Order Matching and Price Order.

In the Order Matching process, a newly entered order is firstly given a price by Order Pricer – a composite role play by the stock trading system and market data feeder. Only if the order is priced, its state will be changed from “New” to “Active”. Then the active order can be matched in the subsequent processes. The stock trading system will try to match the order with the best contra-side order of the same stock symbol (e.g. IBM). The stock trading system searches for contra-side orders from an order book (a list of unmatched orders). If no best contra-side order can be found, i.e., the order cannot be matched, it will be added into the order book for further matching. Otherwise, the order and the selected contra-side order will be matched into a matched-order. The matched-order will be further executed into a trade (i.e. create a new trade). Finally the information of both orders needs to be updated, e.g. the remaining order size and the order state. A trade is created based on the information of the matched-order.
The created trade will not be printed (i.e. Trade Printing) until the order is completely traded.

In the Price Order process, the stock trading system firstly subscribes the price information to the market data feeder. The market data feeder will then process the subscription request (e.g. obtain price information from the market) and provide price information to the stock trading system. Finally the stock trading system calculates the price and applied it to the new order according to the price rule. This way, this order becomes an active order that can participate in the subsequent process of Match Order.

3.2 Modelling Requirements from PFM and RCM

Given PFM and RCM as the source of requirement analysis, a set of problem diagrams can be derived and then matched to problem frames. The frame concern analysis of each problem frame is also presented in this section. A mapping from the elements of PFM and RCM to that of problem frame is shown in Table 1.

<table>
<thead>
<tr>
<th>PFM</th>
<th>RCM</th>
<th>Problem Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessFlow</td>
<td>RoleCooperation</td>
<td>Problem Diagram</td>
</tr>
<tr>
<td>ProcessElement</td>
<td>Interaction/Responsibility</td>
<td>Interface/Requirement</td>
</tr>
</tbody>
</table>

Based on a process flow and its relevant role cooperation on each layer of a process model, a problem diagram can be constructed. This way, we derive a set of layered problem diagrams that model requirements from coarse grain to fine grain. In each layer, problem diagrams may be either directly mapped to a problem frame or projected onto smaller problem diagrams that can be mapped to problem frames.

Role and CompositeRole are one-to-one mapped to the domains of problem diagrams. A CompositeRole may be composed of both human and the machine. The type of its mapped domain is difficult to determine. We consider the domain type as changeable under different problem context. The domain can be treated as any type of which its included roles belong to, according to the choice of an appropriate problem frame that a problem diagram may fit with. Besides Roles and CompositeRoles, the outcomes of process models are considered as potential domains, if they will be used by subsequent processes for a number of times, not simply transient outcomes.

Both interfaces and requirements are identified from Interactions, Responsibilities and ProcessElements. Some requirements may also be identified from potential rules that govern the execution of processes.

Finally, as an essential part of requirements analysis, frame concern analysis is performed to show that the machine specification satisfies the requirements. It ensures that if the machine obeys the derived specification, the requirement will hold under a set of assumptions made by domain properties.

In the following subsections, we will derive and analyze layered requirements from layered processes of Order Matching and Price Order. The stock trading system is the machine to be built, which is called ST Machine in below problem diagrams.

3.2.1 Order Matching

In this business process, both of the order and the trade are outcomes. They are not transient and will be used in the subsequent process of Trade Printing. Order Picker is a CompositeRole. Therefore, four domains are obtained, which are ST Machine, Order Picker, Order and Trade. Both Order and Trade are treated as lexical domain here. From the process flow and the role cooperation of Order Matching, we can know the requirement is to get a priced order (i.e., active order) from Order Picker and then match the order with contra-side orders to a trade(s). Fig. 3 shows the derived problem diagram of Order Matching. Both interfaces of the specification phenomena and the requirement phenomena are omitted, which will be listed in the sub-problems.

In this layer, the Order Matching problem can be further projected onto two sub-problems – get priced order and match order to trade. We select the latter one as an example to illustrate the requirement analysis by using frame concerns. Fig. 4 describes the transformation frame that this sub-problem matches, attached with the relevant frame concern. Order is the input domain, while Trade is the output domain. At the interface “a”, ST Machine gets values of the order’s attributes. At the interface “b”, ST Machine determines values of the trade’s attributes. The matching rule describes how the order information (i.e., the phenomena “d”) is associated with the trade information (i.e., the phenomena “e”).

By analyzing the frame concern of the transformation frame, we can know how ST Machine traverses the OrderInfo and TradeInfo, and correctly transforms OrderInfo to TradeInfo under the guidance of the matching rule.

- Both the input domain (Order) and the output domain (Trade) properties can be described by the class diagram. Fig. 5 shows a part of the domain descriptions. Each order is marked with a stock symbol, size, side and maxMatch at the time it enters the ST Machine. The maxMatch indicates the maximum size of the order that can be matched with other contra-side orders. An order’s price is given by the Order Picker. An order’s state will be changed according to different operations performed on it. All these attributes
belong to OrderInfo. TradeInfo includes symbol, price and size.

![Diagram](image_url)

**Fig. 4 Transformation frame for match order to trade**

- At this layer, the requirement derived from the process model is of coarse grain, as listed below. We may only know which information of the Trade is calculated based on that of the Order, without understanding in detail how Order is transformed to Trade, for instance, how to give a price to an order and how an order’s state is changed. All these detailed requirements will be analyzed in the problem diagrams of the lower layer. The requirements of this layer can be described as:
  - Only if Order is at a certain state, it can be matched to Trade.
  - The symbol of Trade is determined based on that of Order;
  - The price of Trade is calculated based on the price and the side of Order;
  - The size of Trade is decided together by the size and maxMatch of Order.

### 3.2.2 Price Order

There are three domains identified from this business process—ST Machine, Market Data Feeder, Order. Order is the outcome of Price Order, and will always be used in subsequent processes (e.g. order will be used by Match Order and Update Order). Hence, it is permanent and is treated as a domain. The requirement is to subscribe price information to Market Data Feeder and then give the new order a price to activate it. For this process, we can draw a problem diagram that looks like fig. 6.

![Diagram](image_url)

**Fig. 6 The problem diagram of Price Order**

In this layer, the Price Order problem can be projected onto two simpler sub-problems - price subscription and order state changing. As an example, we will analyze the latter one in detail. It can match the required behaviour frame, as shown in Fig. 7. Although the Order is a lexical domain in Order Matching, here we consider it as a causal domain, because in this problem its property includes predicted relationships between its phenomena. For instance, if a new order becomes an active order, it can be matched with other contra-side orders. Otherwise, it cannot be matched. The ST Machine is responsible for CalculatePrice and ApplyPrice. The Order has two states – New and Active. For this problem frame, we will analyze the frame concern as follows.

![Diagram](image_url)

**Fig. 7 Required behaviour frame for order state changing**

- The traverse specification of this problem is described as follows: ST Machine traverses attributes of Order and Trade simultaneously, and produce values of Trade based on values of Order. First, the symbol of Order is found and a symbol value of Trade is produced. Then the price of Order is found and a price value of Trade is produced. Finally, the size and maxMatch of Order are found and a size value of Trade is produced.

<table>
<thead>
<tr>
<th>Order</th>
<th>Trade</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol: string</td>
<td>symbol: string</td>
</tr>
<tr>
<td>price: float</td>
<td>price: float</td>
</tr>
<tr>
<td>size: int/long</td>
<td>size: int/long</td>
</tr>
<tr>
<td>side: enum</td>
<td>maxMatch: int/long</td>
</tr>
</tbody>
</table>

- The requirement is direct and simple. A newly entered order will be given a “New” state. Before it can be matched with other contra-side orders, its state must be changed to “Active”. This requirement can be described like this:

```plaintext
Forever {
    Change an order’s state from “New” to “Active”;
}
```
- Domain properties show how the states of the order is changed based on different operations performed by the ST Machine (i.e. specification phe-
nomena). A simple state machine that describes the change of an order’s statues is given in Fig. 8. Here a new state - New’ - is introduced to denote the conceptual state between the operation of CalculatePrice and ApplyPrice, though physically an order does not have such a state.

![State transitions of an order](image)

- The specification shows how the ST Machine is ought to perform the operations at its interface with the Order, as given below:

```java
Forever {
    CalculatePrice();
    ApplyPrice();
}
```

4. Discussion

This paper tries to carry out layered requirements analysis based on layered business process models. Business processes of complex systems are probably not able to be described in one layer. Hence, the layering approach can clearly represent complicated processes in different layers. Requirements derived from layered business processes as well are under different levels of abstraction. Therefore, to model these requirements, we proposed the layered problem frame approach. The same problem frame can be used to model requirements under different layers. Even if high-level requirements are of coarse grain, based on the frame concern, it can be shown that requirements are satisfied by high-level machine specification together with domain properties, as illustrated by our example in section 3.1.

Jackson [1] proposed projection to decompose a large problem to several smaller ones. Here we want to clarify that the concepts of “Projection” and “Layer” are entirely different. In this paper, layering a requirement means refining it from the coarse grain (i.e. abstract requirement) to the fine grain (i.e. detailed requirement). Therefore, they are on different levels. Projecting a problem indicates the decomposition into sub-problems that are of the same granularity. In this paper, both the problem of Order Matching and Price Order are projected onto two sub-problems at the same layer.

To our knowledge, there is no approach that provides a strict guideline to decompose a complex business process. No published research can provide an unambiguous definition for each process layer. During the requirement modeling by problem frame concerns, we found that for some terribly decomposed processes, it is hard to match an identified problem to a problem frame. Even if matched by somehow, the frame concern analysis is still difficult to perform. We hold the opinion that problem frames and frame concerns may be helpful to the business process decomposition. At least they can be a useful feedback of whether certain decomposition is a good one.

5. Related Work

As the application of Jackson’s problem frame approach, a number of researches have been carried out. Colombo, Bianco and Lavazza [2] report the experience of applying problem frames to model the requirements of a system for monitoring the transportation of dangerous goods. Bleistein, Cox and Verner [8] describe the e-business problem domain by incorporating a business strategy dimension to the problem frames approach. Halley, Jackson, Laney and Nuseibeh describe the decomposition of a lighting control system by using problem frames. To our knowledge, about the stock trading problem domain, there is no research currently being conducted.

Besides the application of problem frame, a lot of work has focused on the subsequent development (e.g. software architecture) assisted by problem frames with the assumption that the problem frame has already existed [9] [10]. There are also other researches about proposing variations of problem frames [11] or entirely new problem frames [12] [13].

Cox and Phalp firstly attempt to derive appropriate problem frames from business process models for an e-business system [8]. Their work [3] [5] is quite relevant to ours. They map a one layer RAD to problem frames to derive requirements, but for currently complex processes, single layer is not enough to represent a complete process. Hence, this paper proposed layered problem diagrams mapping to layered process models. In their approach, problem frames derived from RAD without involving frame concerns actually are not complete. We argue that each PF is ought to be accompanied by a frame concern, as it is important to show that the machine specification together with the domain properties satisfies or implies the requirement [1]. Beside, their approach only identifies requirements and problems from the interactions between roles. Some problem frames, for instance, transformation frame can be easily omitted, as no role in RAD is identified to match the input domain and output domain. Our approach not only takes into account interactions between roles in RCM, but also the process elements in PFM.

6. Conclusions

This paper presents a means of analyzing requirements from layered business process models by using problem frame concerns, combined with the example requirements modeling for stock trading systems. This approach uses layered problem diagrams to model requirements under different levels of abstraction, from highly abstracted re-
requirements to detailed ones. During the requirements analysis, the correctness argument of a problem frame is explored by analyzing the frame concern to combine the requirement, the specification and domain properties.

In this paper, we did not pay much attention to the problem projection in a certain layer. Combining both projection and layering for requirements analysis may be a possible future work. During the layered requirements analysis, a large number of problem diagrams will be derived. Therefore, developing a software tool to manage these problem diagrams of different layers might also be considered as a future work.

ACKNOWLEDGMENT

The work reported in this paper was supported in part by the State Street Cooperation in USA. The authors acknowledge the domain experts for providing the knowledge of stock trading process models. The authors appreciate the College of Computer Science at Zhejiang University for their support of this collaborative project. This work was also supported by Chinese Universities Scientific Fund and Major State Basic Research Development Program.

REFERENCES

Non-functional Attributes for Problem Frames Approach

Bin Yin, Zhi Jin

Abstract—The Problem Frames approach is a new and prospective tool for classifying, analyzing and structuring software development problems. However, it didn’t pay enough attention to non-functional requirements and fails to facilitate the capturing of non-functional requirements. In this paper, we propose a set of non-functional attributes according to characteristics of different kinds of domain. The aim is using these attributes to help system analysts to capture non-functional requirements. A case study will show the reasonability of this approach.

Keywords—non-functional requirement; Problem Frames approach; Requirements elicitation

1 INTRODUCTION

In requirement engineering, functional requirements describe the behavior of system, while non-functional requirements define a set of criteria about how the system will behave, rather than the special behaviors. Non-functional requirements state constraints to the system as well as particular notions of qualities a system might have [1] and directly express quality of software. But there are few good policies for dealing with non-functional requirements. The main reason is that non-fictional requirement has a wide range and is always dealt with in a special way according to domain characteristics [2]. Therefore, how to help system analysts to capture non-functional requirements has become an important research issue.

Many efforts have been made on the issue. And some classifications of non-functional requirements have been proposed. Boehm developed a definitive hierarchy of well-defined, well-differentiated characteristics of software quality [3] in 1976. Sommerville proposed another classification of non-functional requirements in the book of “Software engineering” [4]. He classifies non-functional requirements into product requirements, organizational requirements and external requirements. Recently, more attentions have been paid on the modeling and analysis of the non-functional requirements. For example, Kotona in [5] focused on the relationship between user concerns and non-functional requirements. The second one is based on goal-oriented modeling method proposed by Loucopoulos in 1995 [6]. Lawrence Chung and other people have proposed another goal-oriented approach to deal with the non-functional requirements [11]. The concept of soft goal has been used to capture the features of non-functional requirements. Most of the approaches for modeling and analyzing non-functional requirements are developed based on some particular requirement engineering methodology. The Problem Frames (PF) approach [7] is a new and prospective tool for classifying, analyzing and structuring software development problems [8]. In PF approach, requirements, in the sense in which we are using the word, are located in the environment, which is distinguished from the machine to be built [9]. That principle can help people to recognize problems of real world. The PF approach also defines five basic problem frames and their variants to capture the functional requirements.

In terms of the principle of the PF approach, we argue, in this paper, non-functional requirements are also located in the environment—that is, the domains in the real world, just like the functional requirements. For example, in a stock exchanging system, there are some non-functional requirements such as immediacy of stock exchanging, accuracy of data, and security of user’s password. These non-functional requirements come from the environment, from the system’s potential users.

In this sense, based on the PF approach we propose to capture non-functional requirements by attaching some non-functional attributes on the domains and interactions between the machine and the domains. System analysts can be guided to capture relevant non-functional requirements by answering the questions about these attributes. In this paper, we define a set of non-functional attributes for three kinds of domains according to their different domain characteristics and use an example to show how these attributes can be used to guide the capturing of the non-functional requirements.

Section 2 is a brief introduction to the PF approach. In section 3, we will present the definitions of sixteen non-functional attributes designed for three kinds of domains and explain how these attributes are related to the non-functional requirements. In section 4, we will take a banking system as an example to explain how to use these attributes to capture the nonfunctional requirements. Section 5 is about the related work. Section 6 concludes the

---

- B. Yin is with the MADIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China. Email: byin@amss.ac.cn.
- Z. Jin is with Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University, China. Email: zhijin@amss.ac.cn.

Manuscript received 15 March 2010
2 INTRODUCTION TO PROBLEM FRAMES APPROACH

The PF approach is a problem domain oriented analysis approach which addresses the real world problems, and provides a means of analyzing and structuring problems [10]. Using the PF approach to elaborate the software development problem, we get a series of diagrams. First, by developing the context diagram, the world is structured into the problem domains and the machine domain. Three kinds of domains have been differentiated in the PF approach. They are the biddable domain, the causal domain and the lexicon domain. The next step is identifying the interactions between the machine and the domains. The context diagram is used to represent the machine, the domains and the interactions between them. Then by adding the requirements to the context diagram, the problem diagram is obtained. The problem diagram is a starting point for problem analysis. A composite problem diagram needs to be decomposed, i.e. be projected, progressively into some sub-problem diagrams. If each of these sub-problem diagrams can be fit with a problem frame, the solution of the original problem can be figured out as the solution of each problem frame has been well-defined.

3 NON-FUNCTIONAL ATTRIBUTES ON THREE KIND OF DOMAIN IN PF

As we argued above, like the functional requirements, the non-functional requirements are also located in environment – domains in the real world. They are the constraints arisen from the domains. Thus, for capturing the non-functional requirements, we need to identify which kinds of constraints can be arisen from each domain that the machine will interact with.

Based on some available standards of the non-functional requirements properties [3] [4], we collect a set of non-functional properties and attach them with the domains as their non-functional attributes. The meanings of these attributes may vary with the domain characteristics. We describe the meanings of the non-functional attributes. These non-functional attributes can be used to help system analysts to identify the potential constraints from the domains efficiently.

3.1 Attributes on biddable domain

A biddable domain usually consists of people. The most important characteristic of a biddable domain is that it’s physical but lacks positive predictable internal causality [7]. It is of autonomy and may have some wishes on how the system will work. System analysts can not compel a person to initiate an event. These characteristics imply that the following attributes need to be considered for a biddable domain.

- Usability

  The meaning: This attribute describes whether the members of this domain have a higher expectation for usability.

  Interactions initiated by a domain with this attribute must be designed to be more convenient for users to operate and more easy for a novice to learn how to use it.

- Aesthetic

  The meaning: This attribute describes whether the members of this domain have a higher expectation for aesthetic of the interface of the system.

  In many systems, such as image processing system, great importance should be attached to aesthetics. System designers will take more considerations on how to make the interface of the software look good. Domain with this attributes can direct designer to be aware of aesthetic of the software. But a designer for a library system may not need to make such a great effort on the aesthetic considerations.

- Peak Load

  The meaning: This attribute describes how many people may operate this system simultaneously at most.

  For example, a requirement analyst of an online system should pay more attention to how many people are likely to log into the system at a time. This attribute relates to many aspects of system analysis, such as the size of memory applied by program, the CPU load, and the users’ net.

- Total number of user

  The meaning: This attribute describes how many people will register the system in total.

  Requirement analysts could estimate the scale of user information which should be stored in a database by multiplying total number of users by the number of byte for information of each user.

- Operational constraints

  The meaning: This attribute describes whether the members of this domain should be trained or learn something else in order to operate this system.

  Many systems are developed to achieve a particular function. As a result, this kind of system can only be operated by some expert operators. For example, the system developed to control space shuttle must be operated only by trained astronaut, instead of other people.

- Complexity of users’ identity

  The meaning: This attribute describes whether the identity of the members of this domain is complex.

  This attribute is also very useful in many online systems. In a system oriented wide area network, the identities of users are complex as every people who connect with net can use this system. By contrast, in a system oriented local area network, such as a communicating system in a company, identities of users are not so complex, i.e. only the employer of the company can use it. We will adopt different levels of security policy. The Designers for the first system should pay more attention to that whether some wicked people will attack this system by some illegal methods, while the designers for the second system may not do it.

- Reliability

  The meaning: This attribute describes whether the members of this domain have a higher expectation for services provided by the system.

  It consists of MTBF (Meantime between Failures), uptime (a period of time when the service is available for user), MTTR (Meantime to Repair), and robustness.

- Language
The meaning: This attribute describes whether the members of this domain use another kind of language. If a system can be used by people in many countries, there should be some versions in different kinds of language, apart from mother language.

3.2 Attributes on lexical domain
A lexical domain is a physical representation of data – that is, of symbolic phenomena. It combines causal and symbolic phenomena in a special way [7]. Some non-functional attributes have been chosen for it as follows.

- **Flexibility**
The meaning: This attribute describes whether the domain is likely to be changed or updated.
This attribute focuses on the changes of the physical device. If a physical device has been changed or updated, the interactions between the machine and this domain must change too. That asks the analysts of the system to focus on the compatibility of those interactions.

- **Safety**
The meaning: This attribute describes whether the domain has the ability to damage other things in the real world. Some physical devices are very dangerous, such as train, car, and so on. If there is something wrong with the control system of these devices, accidents may happen. That will cause heavy damages probably. The interactions which initiated by machine and received by the physical devices should take this attribute into consideration.

- **Minimum response time**
The meaning: This attribute describes the minimum response time of the physical device. Unit of this attribute is second in general case. This attribute is decided by hardware’s configuration of the storage device too. This attribute can help analyst to limit the interval between two interactions. One is initiated by the machine and received by the physical device, and the other is the response interaction.

4 Case study
Let’s use a banking system to illustrate the reasonability of some of the non-functional attributes. This is a composite problem which includes two basic problems. One is that users can withdraw cash from ATMs. The other is that users can inquire the amount of money in their account and update their personal information through web. Figure 1 shows the overall problem diagram and figure 2 and figure 3 show these two problem diagrams respectively.

![Overall Problem Diagram](image1.png)

**Figure1:** overall problem diagram
a: bs1{withdrawCash} ATM1{showTip}
b: user1{amountOfMoney, userInfor}
c: ATM1{login, waiting, logout}

Figure 2: problem diagram for withdrawing cash

Banking system

Database

User

Revise

Figure 3: problem diagram for inquiring the amount of money and updating personal information

Now we can use the attributes described in last section to capture the non-functional requirements from each problem diagrams. System analysts can be guided to capture relevant non-functional requirements by answering the questions about these attributes mentioned above and some non-functional requirement tables are generated.

The first sub-problem is a command behavior problem and fits to the commanded behavior problem frame. Users can use an ATM to achieve the goal of withdrawing cash. The following is the non-functional requirements captured by identifying those related attributes on the domains of this sub-problem.

Table 1: non-functional requirements on user

<table>
<thead>
<tr>
<th>Attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>The system must be convenient for user to achieve their goal of withdrawing money. Moreover, it must offer directions in order to help a person who has not used ATM beforehand.</td>
</tr>
<tr>
<td>Aesthetic</td>
<td>There are no more requirements for aesthetic. Users would not focus on whether the interface of this system is beautiful.</td>
</tr>
<tr>
<td>Peak Load</td>
<td>There are no more requirements for load. Because one ATM only can be used by one person.</td>
</tr>
<tr>
<td>Total number of user</td>
<td>The number of users of this bank is about one million.</td>
</tr>
<tr>
<td>Operational constraints</td>
<td>There are no more operational constraints. Everyone could use this system conveniently.</td>
</tr>
<tr>
<td>Complexity of users’ identity</td>
<td>Users’ identity is complex. The system should adopt a higher level of security policy.</td>
</tr>
<tr>
<td>Reliability</td>
<td>The ATM will work all day long. It is allowed to break down once a year.</td>
</tr>
</tbody>
</table>

Language

This ATM can also serve foreigners. So the system should offer an interface in English.

Table 2: non-functional requirements on ATM

<table>
<thead>
<tr>
<th>Attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>ATMs are likely to change. So the interface between machine and ATM should have a good compatibility.</td>
</tr>
<tr>
<td>Safety</td>
<td>There are no more requirements for safety. ATMs have not the ability to damage other things in the real world.</td>
</tr>
<tr>
<td>Minimum response time</td>
<td>The minimum response time of the ATM is 0.1 second. So the interval between the interaction of “withdraw cash” and the interaction of “pop cash” is 0.1 second at least.</td>
</tr>
</tbody>
</table>

The second basic problem is a simple work pieces problem and fits to the simple work pieces problem frame. Users can inquire the amount of money in their account and revise their own information through web, such as user name, password and so on. The following two tables contain the nonfunctional requirements captured by identifying those related attributes on the domains of this sub-problem.

Table 3: non-functional requirements on user

<table>
<thead>
<tr>
<th>Attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>The system must be convenient for users to achieve their goal of withdrawing money. Moreover, it must offer directions in order to help a person who has not used ATM before to use this system.</td>
</tr>
<tr>
<td>Aesthetic</td>
<td>There are no more requirements for aesthetic. Users would not focus on whether the interface of this system is beautiful.</td>
</tr>
<tr>
<td>Peak Load</td>
<td>The maximum number of people who can use this system simultaneously is 20 thousands.</td>
</tr>
<tr>
<td>Total number of user</td>
<td>The number of users of this bank is about one million.</td>
</tr>
<tr>
<td>Operational constraints</td>
<td>There are no more operational constraints. Everyone could use this system conveniently.</td>
</tr>
<tr>
<td>Complexity of users’ identity</td>
<td>Users’ identity is complex. The system should adopt a higher level of security policy.</td>
</tr>
<tr>
<td>Reliability</td>
<td>The system is designed to prohibit user from inputting wrong format of user’s information, such as a too long user name.</td>
</tr>
<tr>
<td>Language</td>
<td>There are no more requirements for language.</td>
</tr>
</tbody>
</table>
Table 4: non-functional requirements on database

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>2000GB</td>
</tr>
<tr>
<td>Flexibility</td>
<td>There are no more requirements for flexibility.</td>
</tr>
<tr>
<td>Confidence and importance of data</td>
<td>This domain has stored the information of users and their accounts. So it requires a higher level of confidence and importance.</td>
</tr>
<tr>
<td>Query speed</td>
<td>0.3 seconds. So the interval between the interaction of “userInform” and the interaction of “accountInform” is 0.3 seconds at least.</td>
</tr>
<tr>
<td>Maximum throughput</td>
<td>20MB per second</td>
</tr>
</tbody>
</table>

5 RELATED WORK

NFRs have been the subject of research for many years and we only review NFRs in the context of the RE literature.

Lawrence Chung and other people have proposed a goal-oriented approach in their book entitled “Non-Functional Requirements in Software Engineering” [11]. This approach uses the concept of soft goal to express non-functional requirements and regards non-functional requirements as a set of soft goals which the system should satisfy. By this means, system analysts can refine non-functional requirements by elaborating soft goals. But they didn’t supply the guidelines that can be followed in the process of eliciting non-functional requirements. On the contrary, we propose that non-functional requirements can be elicited from the environment constraints to the system. The non-functional attributes proposed in this paper help system analysts to consider systemically what non-functional constraints are likely to be brought into the system by different kinds of environment domains. In the paper of “non-functional requirements elicitation” [12], the authors propose an approach to elicit non-functional requirements using LEH [13]. System analysts need to iterate the knowledge base on NFRs once for each LEH symbol, because the approach doesn’t classify non-functional requirements. However, the non-functional attributes proposed in this paper are classified into three groups in terms of the characteristics of different domains. Therefore, system analysts can capture non-functional requirements more efficiently.

Ian Alexander proposes an approach to elicit non-functional requirements by using misuse cases. In his paper [14], he introduces how to elicit security and safety requirements. However, there are some other non-functional requirements which are difficult to elicit in this way, such as the attributes of aesthetic and minimum response time proposed in this paper.

6 CONCLUSIONS

In this paper, we have proposed a set of non-functional attributes for biddable, causal and lexical domains in the PF approach and presented detailed definitions of these non-functional attributes. These attributes should be taken into consideration by system analysts and can help analysts to capture the nonfunctional requirements when using the PF approach. But the list of these non-functional attributes could not be complete. In the next step, we are going to complete the list and validate the approach on real-world problems. After that, we will formalize these attributes as knowledge to build an ontology and integrate this ontology into the PF approach for helping analysts to elicit non-functional requirements systematically.

ACKNOWLEDGEMENT

This work was supported financially by the National Science Fund for Distinguished Young Scholars of China (Grant No. 60625204), the National Basic Research and Development 973 Program (Grant No. 2009CB320701), the Key Projects of National Science Foundation of China (Grant Nos. 90818026, 60736015).

REFERENCES

Abstract—Hall and Rapanotti define the POE Process Pattern in [4]. They state that it is a generator for processes being composable in three ways: in sequence, in parallel and fractally. Here we provide a first sketch of a process algebra for POE processes, and sketch definitions of the most interesting - the fractal composition - in CSP-M. Give a CSP-M semantics of POE process will allow us better to reason about their trace properties, perhaps identifying deadlock and the like in a modelled process.

I. INTRODUCTION

The POE Process Pattern (PPP) as demonstrated in [4] and [5], can be replicated and the results could be combined together, leading to a fractal design. This idea was further extended to design and expressed as sequential, parallel, and recursive (or fractal) design [6]. There are four main elements of the PPP: POE exploration process (for problem and solution exploration) and POE validation processes (for problem and solution validation) (Figure 1).

It is a long tradition that process algebras are illustrated or even fitted to graphical models of processes. Both Milner [9] and Hoare [7] illustrate their process algebras, and the Petri Box Calculus [1] forms a process algebraisation of Petri Nets. Here we continue this tradition based on our analysis of the POE Process Pattern.

This extended abstract discusses a Problem Oriented Engineering Process Algebra (POEPA) that allows for composing POE processes and their elements to build new POE-based processes, perhaps with new properties and characteristics. The benefits of creating an algebra for POE process are twofold. Firstly, we wrap up POE process definitions in a concise and abbreviated form, which simplifies syntax of process description. Secondly, the POEPA would enable manipulation of processes and their compositions that could be a basic language for expressing structural characteristics of arbitrarily complex POE processes. POEPA operations on POE processes could be useful to both human and non-human agents, and would make it possible to generate as well as to consume process descriptions encoded in POEPA syntax.

II. POE PROCESS ALGEBRA

A process in POEPA is formed under the following syntax:

\[ P = \beta \mid P ; P \mid P \parallel \mid P \parallel P \]

in which a basic POE Process, \( \beta \), combines to produce sequence, parallel, and fractal processes. The reader will note that, unlike traditional process algebras, no choice operator is defined for POEPA, this is because of the backtracking of POE Processes is thought to make it unnecessary.

These operations are treated as fundamental operators under POEPA[3]. Our semantics is over the domain of CSP-M expressions and so we must associate with each POE Process expression \(^1\) a CSP-M term.

\(^1\)At this point we model only very simple validation structures into which the POE processes will fit.
III. POE Process Modelling

Currently the work is underway that attempts to model the POEPA using machine readable CSP (CSP-M) [2], as implemented in ProB tool [8] that allows for animating and model checking of CSP models and traces [7]. Expressing the POEPA semantics of POE process in terms of a CSP model might allow for identifying deadlock and perhaps other properties of a POE process model.

A. Validation

Validation within POE Processes will, briefly, be achieved through composition with CSP-M terms representing validation processes. As yet, we do not have a precise model for these, but simple versions – with which we have come to this point in our modelling – are

\[
\text{VALIDATEP} = \text{askedVP} \to \\
\quad \Box x : \{\text{validP}, \text{notValidP}\} \bullet x \to \text{VALIDATEP}
\]

\[
\text{VALIDATES} = \text{askedVS} \to \\
\quad \Box x : \{\text{validS}, \text{notValidS}, \text{notValidSP}\} \bullet x \to \text{VALIDATEP}
\]

in which validP represents a successful problem validation, notValidP an unsuccessful one in the problem validation phase, and validPS represents a successful solution validation, notValidS an unsuccessful one and notValidSP an unsuccessful problem validation in the solution validation phase.

B. Basic POE Processes

A basic POE Process has embedded problem and solution exploration processes which, for this paper, we will assume have a CSP-M expression as, respectively, \(\text{PROBX} = \lambda x \bullet \text{exploreP} \to x\) and \(\text{SOLNX} = \lambda x \bullet \text{exploreS} \to x\). Given these definitions, define the basic process

\[
\text{PROBX} \bowtie \text{SOLNX} = \lambda y \bullet \text{PROBX}(\text{checkVP}) \to \\
\quad \text{SOLNX}(\text{checkVS} \lor y \lor \text{validS} \to \text{SKIP}) \lor \text{notValidS} \to \text{REDOSX}) \lor \\
\quad \text{validP} \to \text{SKIP}\} \lor \text{notValidP} \to \text{REDOPX} \\
\quad \Box \text{notValidSP} \to \text{REDOPX}
\]

We note that \(\text{PROBX} \bowtie \text{SOLNX}\) is a \(\lambda\) expression as is its operand, and so may be substituted within other \(\bowtie\) expressions. This is the basis of the fractal composition.

C. Sequence and Parallel Composition

That our semantics is in terms of \(\lambda\) expressions makes sequential composition modelable in terms of functional composition. Parallel is a slightly more complex operator which involves CSP-M parallel composition and renaming (so that its operands do not share a namespace). We do not have the details of parallel composition as yet.

IV. Conclusion

Introducing the POEPA onto POE processes is a first step into POE process modelling, and it should allow for capturing arbitrarily complex POE processes and expressing them in a simple syntax. Such process expressions would be useful for animating and simulation purposes, and particularly for investigating semantics and behaviour of complex POE processes. This would potentially be useful in the area of process evaluation and refactoring.

ACKNOWLEDGMENT

The author would like to thank Jon G. Hall in the Centre for Research in Computing at The Open University for the feedback and inspiration.

REFERENCES


A POSE Process with Alloy and Perfect Developer
Derek Mannering
General Dynamics UK Ltd.
Castleham Road
St. Leonards-On-Sea, UK, TN38 9NJ
+44 1424 798126
derek.mannering@generaldynamics.uk.com

ABSTRACT
The goal of this paper is to report on the industrial use of Problem Oriented Software Engineering (POSE), an extension and generalisation of Problem Frames, as an integral part of the early phases of a safety critical development process.

Categories and Subject Descriptors
D.2.1 [Requirements/Specifications]: Methodologies

General Terms
Design

Keywords
Problem Orientation, Safety, Requirements Transformation

1. Extended Abstract
The demonstration of adequate safety is a crucial factor in the deployment of many embedded systems, including those used in avionics applications. This concern has been captured in safety standards such as the UK Defence Standard 00-56 [17] and the international IEC 61508 [7]. These standards require hazard identification and preliminary hazard analysis to occur in the early phases of the development process (e.g. [15]). This is consistent with studies that have shown that a large proportion of anomalies occur at the requirements and specification stages of a system development [3, 9]. A study by Lutz concluded that safety-related software errors arose most often from inadequate or misunderstood requirements [11]. Other work has highlighted the need to conduct a safety analysis of the requirements [2, 4]. This implies that safety must be built into the design, and that the evolving design representations must be analysed to demonstrate that they have the desired safety properties [10].

System engineering processes to support critical system development by necessity must include the identification and clarification of system requirements, the understanding and structuring of the context into which the system will be deployed, the specification of a design for a solution that can ensure satisfaction of the requirements in context, and the construction of convincing arguments that the system will provide the functionality and qualities that are needed [5]. The research work reported in [12] demonstrated how the Problem Oriented Software Engineering (POSE) framework [6, 14] could be used to directly support the early phases of an appropriate system engineering process – specifically supporting the process of formulating a requirements model that can undergo hazard identification and preliminary hazard or safety analysis (PSA) as required by the safety standards (e.g. [17]). The successful result being a revised requirements model that is known to be able to satisfy its identified safety requirements and thus forms a good basis for the remainder of the development process. Further, the work in [12] showed how POSE, in conjunction with the Alloy formal development system [8] could be used to improve an existing safety critical development process used by the author’s company. The case study work presented in [12] was based on realistic examples but was retrospective. The success of these case studies encouraged the use of the approach (POSE/Alloy in combination) at the front end of a number of real critical system developments being undertaken by the author’s company. The process used was based on the work in Chapter 6 of [12] – particularly the POSE safety pattern, the method for using POSE to develop a formal (Alloy) model and how to further develop and validate this formal model (especially sections 6.3.1 to 6.3.6 and section 6.4). In particular, the real development work used the four part process for applying the formal modelling identified in section 6.4 of [12]:

1. use simulation to develop the model and gain confidence that it has the required behaviour;
2. perform formal proof to validate the model;
3. use the model to prove safety properties (if necessary);
4. investigate any specific problem areas using simulation.

The results were successful for the initial problem development where the problem model was implemented in stages. However, producing a full model was problematic as the Alloy tool ran out of resources due to the number of elements involved, especially where multi-arity functions were used. Simplifying the models to use lower-arity functions was successful, but this removed the parameterisation that was possible with the multi-arity representation – a highly undesirable outcome.

As detailed in [13], the original company safety critical development process used the Z notation [16] to specify the system. However, this required a costly and time consuming manual validation of conformance and consistency. To provide tool support the safety critical process was enhanced by the introduction of the Perfect Developer (PD) tool [1], which was used to specify the software in place of Z, allowing the consistency and conformance proof work to be automated. Initially, the POSE/Alloy work was used at the front end of this development to validate the early system models had the desired behaviour and satisfied the required safety properties. However, the inability of the Alloy tool to scale up prompted an investigation of whether POSE could be used directly with PD to derive appropriate representation models.

The structure used to form Alloy models from the output of the POSE safety pattern was adapted for use with PD. The proof tools operated fine, the only problem was a lack of simulation.
capability, which was addressed by using the code generation capabilities of PD to produce Java which could be run within a defined test harness. A further advantage of this test harness is that it allows the development of system level tests that can be used in the final design proving. The availability of this simulation capability, albeit with more effort than the Alloy system, allows the 4-step formal modelling process to be used with PD models, which means that the efficacy of the PD model can be checked early in the development process. The result is that the design, as represented by the PD model, can be shown to have the required behaviour and to satisfy its safety properties – thus forming a good basis for the rest of the development process.

2. CONCLUSIONS
In conclusion, the current thinking concerning the use of POSE, Alloy and PD is as follows. For concept exploration and initial proving work use POSE and Alloy, since Alloy has the simulation and proof capabilities readily available and these can be quickly deployed to test alternative architectures – the goal being to select a appropriate candidate architecture. Then proceed to full development of the selected candidate architecture using POSE in conjunction with PD, with the coincident development of a test harness framework that allows the PD specification developed from the POSE model to be simulated and design proving tests to be developed. This combination also has the advantage that it allows simulation and proof to be conducted on the PD model using the 4-step formal modelling process described above.

3. ACKNOWLEDGMENTS
My thanks to Jon Hall and Lucia Rapanotti at the Open University for providing much needed support for this work.

4. REFERENCES
Design Rationale Capture with Problem Oriented Engineering

Ann Nkwocha
CWTechSolutions LTD (Bank of America Corporation)

ABSTRACT
Design rationale in software engineering fills in the gaps between the original requirements of a system and the finished product encompassing decisions, constraints and other information that influenced the outcome. Existing research in this field corroborates the importance of design rationale for the evolution of existing systems and creation of new systems. Despite this, the practice of design rationale capture and reuse is not as extensive as could be expected due to reasons which include time and budget constraints and lack of standards and tools.

This capture of Design Rationale during software design activities carried out using Problem Oriented Engineering (POE) was demonstrated with the use of a case study. POE is a formal system for engineering design that provides a framework for the resolution of software problems in a stepwise manner.

1. BACKGROUND
Computers and software have become indispensable in western society, underpinning business, health, careers, welfare and day to day domestic activities.

The need to meet demands for new functionality, faster and more effective features and to exploit new technologies necessitate constant change to the way in which they are used. With so much invested in software, there is a necessity to guard the knowledge and experience which makes these changes possible.

Design rationale captures the reasoning and knowledge that justify the resulting designs (Tang et al., 2006) and its availability can expedite the creation of new and the evolution of existing systems. Research in this area has highlighted advantages of having and reusing design rationale. Despite this, it is still not always captured during design activities due to perceptions that it can be time consuming and expensive adding (Burge et al., 2000) and places extra burdens on resources facing tight deadlines without presenting any immediate benefits to them (Krutch et al., 2005).

Problem Oriented Engineering (POE) (Rapanotti and Hall, 2008) is a formal system for engineering design that provides a framework for the resolution of software problems in a stepwise manner resulting from research extending Problem Frames, aimed at analysing and structuring problems, to include activities addressing the solution domain.

Software problems are represented as sequents consisting of a real world context, a requirement and a solution. The framework was originally presented as Problem Oriented Software Engineering (POSE). Main features include:
- A stepwise process for solving solve software problems. Each step, referred to as a transformation, relates a problem and its justification to its derivative set of problems.
- Problem Transformation Schemas contain information on how a problem is transformed from its premise problem(s) to the conclusion problem.
- Justifications detailing the suitability of a solution for the problem are detailed during problem transformations.

A generalised POE process map has emerged from research facilitating its application to non-software engineering use resulting in the generic form of the framework – Problem Oriented Engineering POE (Hall et al., 2007).

1.1. REASONS FOR THE RESEARCH
This research aimed to show how design rationale capture can be achieved during the design process using the POE process. To accomplish this aim, the following question was addressed:

What information gathered during the application the POE framework to a software design problem makes it suitable to represent the design rationale of the problem in a form that facilitates future re-use?

The investigation considered:
- Whether the information captured was sufficient to represent design rationale
- Whether the information that is captured was in a structure that would allow it to be accessed for future reuse.

2. THE VALID AMOUNTS CASE STUDY
For this research a single case study was used to examine:
- If the information gathered during the execution of the case study could be characterised as design rationale
- If the structure of the information gathered could facilitate access for later reuse
If the use of the POE framework could affect costs and benefits of the design rationale capture activities.

The selected case study was set in a UK based subsidiary of an American Financial organisation with clients; business, systems and technical analysts based in the UK; technical architects in the US and development staff in India.

Information was gathered by applying POE process steps to transform a problem into a solution acceptable to pertinent stakeholders. The steps were documented using notation prescribed in existing POE literature and the results examined for design rationale elements identified during. The results were examined for design rationale elements identified from the literature survey which were:

Rationale for decision(s) taken, Alternatives, Suitability, Constraints, Assumptions, Status, Complexity of the design, Issue descriptions, Traceability and Contacts (stakeholders)

A questionnaire was designed and sent out to staff to sample perception about design rationale in the company.

The company, Software Supplier, licence software to a Client for use in servicing mortgage accounts. The Valid Amounts Problem is based on an issue where the Client identified instances where financial figures used to calculate loan balances and billed amounts were incorrect.

POE process steps were followed to solve the Valid Amounts Problem case study. The existing process for receiving and dealing with issues in the Software Supplier. The graphical notation, templates, terms and abbreviations used to document the case study were described. The POE steps applied to solve the problem were explained and documented. Figure 2 illustrates the topology of the problem using POE graphical notation.

3. DESIGN RATIONALE CAPTURE AND REUSE

Results of the questionnaire indicated that perceptions of staff in the case study organisation were in line with those encountered in existing Design Rationale research and that the setting of the Valid Amount Problem makes it a typical case study from which generalisations can be drawn.

Pattern matching evidence gathered in the case study with the elements of design rationale established from my literature survey, it was possible to show that POE captured information representing each of the identified Design Rationale Elements. The results are summarised in Table 1.

An important factor observed in this research was that the design rationale was recorded as a methodological by-product (Lee, 1994) of the using the POE process. Using POE for solving software problems with the required validation and justifications carried out for the steps captures the rationale with no additional effort needed. One could conclude from this that the use of POE would assist in addressing cost and time concerns that affect the practice of design rationale capture noted in existing research.

The potential for reuse of design rationale was discussed by examining the flow of information during the execution of POE for the case study (shown in Figure 3).

Formal hand-off point between each activity in the case study facilitated validation activities. The absence of such a hand-off artefact between the initial problem validation (1) and the Client validating the solution (5), was observed as a gap in the Software Suppliers process. In their normal day to day processes, communication following the acceptance

The research found that the ability to reuse of design rationale captured in the POE process depends on the effectiveness of the categorisation, storage and organisation of the information gathered. In Rapanotti and Hall (2008) where POE was used practically on a project for the Open University, categorisation and access were addressed by using a structured repository based on a naming convention for the problems examined.

5 CONCLUSION

A review of literature on Design Rationale, its capture and management yielded a list of elements used as the criteria for identifying design rationale in the information gathered during the case study. Examination of that information revealed that all the identified elements were recorded and led to the conclusion that Design Rationale is captured when solving a software problem using POE.

Examination of the flow of information that occurred during the execution of the case study led to the conjecture that Design Rationale recorded during the case study could be reused. Successful reuse would, however depend on the effectiveness of the categorisation, storage and organisation of the information gathered.

This research has shown that POE could be a valuable methodology for design rationale capture. The benefits to be gained from the practice of rationale capture have been highlighted in research. Time and budget constraints on software projects are cited as key factors in restricting its uptake. Having the ability to capture rationale without additional impact to those factors could encourage greater uptake of the practice. The results of this research have led to the conjecture that POE could be used to accomplish rationale capture through methodological by-product thus helping to address time and budget concerns.

Hu et al., (2000) surmise that the way in which design rationale will be stored and presented for retrieval is of
critical importance supporting the communication and reuse of design knowledge. A large amount of information can be generated from the use of POE and effective categorisation and storage for this information is an area that would benefit from further research.

A much fuller version of the whole report is currently reaching completion, for submission to The Journal of Software Quality. In the mean time it is accessible as http://computing-reports.open.ac.uk/2010/TR2010-06.pdf

ACKNOWLEDGEMENTS

Many thanks to Jon Hall and Lucia Rapanotti, whose enthusiasm, encouragement and hard work has played a major role in ensuring this research has been completed. I would also like to thank my manager Mr Bob Crafer and employers CWTech Solutions who have given me time and support to invest in my research and last but not least my husband Gus Nkwocha who has consistently supported my academic endeavours.

REFERENCES


Towards Tool Support for Problem-Oriented Software Engineering using Knowledge-Based Techniques

Zhi Li, Professional Member, ACM

Abstract—Problem orientation is gaining popularity among the software engineering community. Its development can be traced back to Jackson’s foundational work on problem frames (PF). As extensions to PF, problem-oriented software engineering (POSE), and recently problem-oriented engineering (POE) have been proposed. Here we discuss the tool needs of POSE, and their basis in knowledge engineering.

Index Terms—knowledge engineering, problem frames, problem orientation, tool support.

1 INTRODUCTION

Problem orientation is gaining popularity among the software engineering community [1-7]. Its development can be traced back to Jackson’s foundational work on problem frames (PF) [5, 8]. As extensions to PF, problem-oriented software engineering (POSE) [6, 9], and recently problem-oriented engineering (POE) [10] have been proposed.

POSE are based on the following building blocks: firstly, problems, which capture knowledge about the requirements, their context and the designed solutions; secondly, transformations, which capture discrete problem-solving steps; thirdly, justifications, which capture necessary knowledge for stakeholder validation (see [6, 10] for detailed examples).

Please note that both “problems” and “justifications” rely on “knowledge” captured from various stakeholders (e.g., requirements from customers, context from domain expert, designed solutions from software architect, etc). “Transformations” are performed based on such knowledge. This motivates us to develop tool support for POSE using knowledge engineering techniques.

According to Feigenbaum and McCorduck [11], knowledge engineering (KE) is “an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise”. Since we want to build computer-aided tool support for solving complex engineering problems under POSE, it is natural to look for applicable KE techniques.

2 RESEARCH QUESTIONS

Like many software development approaches, such as the scenario [12], or goal-oriented approaches [13], tool support is essential to their adoption and empirical evaluation in industry [14-16]. In order to provide support for the evaluation and assessment of POSE in a practical setting, we set the following research agenda aiming at providing tool support for problem-oriented software engineering:

1. Given that the knowledge about problems (i.e., requirements, context and designed solutions) is structurally bounded by problem diagrams with some textual descriptions, how can we ensure that sufficient knowledge is presented for the desired problem transformations?
2. Given that the problem-solving process can be represented by several problem transformations which can be made systematic by a rule-based technique [17], how and to what extend can problem transformations be automated or semi-automated?
3. Given that the rule-based transformation technique is based on knowledge about causal relationship, how can this type of knowledge be represented or deduced using knowledge-based techniques?

3 KNOWLEDGE-BASED TECHNIQUES

Many knowledge engineering techniques have been applied to requirements engineering (see [18-23] for some examples, and [23] for a survey on relevant knowledge-based tools). Particularly relevant to problem-oriented software engineering are ontology-based approach [24], domain-based approach [25], and AI techniques in problem-solving [26, 27].

Since the working definition of causality adopted in our problem transformation is targeted for practical engineering use, we aim for finding and applying KE definition of causality and associated techniques for POSE. In addition to eliciting and identifying causal relationships, the application conditions and strategies of those problem transformation rules [28], i.e., which can be mapped to “justifications” under the POSE framework, have ramifications to the choice and creativity in the problem-solving process. We believe KE may contribute to this aspect of POSE.

* Zhi Li is with the College of Computer Science and Information Technology, Guangxi Normal University, No. 15 Yu Cai Road, Guilin, Guangxi 541004, P. R. China. E-mail: zhili@mailbox.gxnu.edu.cn.

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.
4 RESEARCH METHODOLOGY

Firstly, we are going to do a systematic literature review [29] in the KE literature to identify suitable knowledge-based techniques for providing tool support for POSE.

Secondly, we are going to develop a software tool for creating and editing problem diagrams with associated texts, representing causal relationships, and recording and semi-automating various steps of problem transformations.

Thirdly, we are planning to evaluate the set of tools using empirical studies, e.g., observational studies, in which the tool users’ experience and results will be observed and evaluated by another set of researchers. Although we do not expect any conclusive evidence, still some feedback will be valuable for improving the tool set.

4 CONCLUSION

In this paper, we have set out a series of research agendas for problem-oriented software engineering. Although there may be many possible areas in which POSE can be supported, we believe that knowledge-based techniques may be a promising area to be explored.

ACKNOWLEDGMENT

The author wishes to thank Guangxi Normal University for funding this line of research.

REFERENCES


A Second Life learning space for Problem Oriented Engineering

Andrew Moore
The Open University, UK

Introduction

Over the past few years online environments accessible via the internet have allowed computer users from all over the world to interact in various flavours of online virtual worlds. These environments range from simple messaging services (Twitter, 2010) through social networking sites (Facebook, 2009) to sophisticated 3D online games (Blizzard Entertainment Inc., 2009). Increasingly, commercial (Linden Labs, 2009) and educational (Dieterle & Clarke, 2007) institutions are using these readily available online worlds to train, their staff and students using immersive “games” in these 3D worlds to provide an environment similar to the real world. Second Life (Linden Lab, 2009) is one of these 3D online virtual worlds, where users have the ability to interact in a social space, but where they also have the capability to design and build their own virtual worlds. As one of the market leaders in 3D virtual worlds, Second Life provides a good platform from which to study the design, construction and use of virtual educational environments.

One of the problems faced by practitioners wanting to use 3D virtual worlds for education is the lack of design guidelines or principles. In this paper, we report on an small scale investigation into the applicability of a set of 2D multimedia design principles (Nelson & Erlandson, 2008) to the design 3D educational environments. As proof of concept, the investigation focuses on the design of a Second Life environment to teach some of the basics of Problem Oriented Engineering, an emerging framework, being developed by The Open University (Hall & Rapanotti, 2009), which formalises engineering design by using a four stage process to explore and validate a problem and then its solution. The framework was chosen because it is an emerging engineering design framework with a limited knowledge base beyond The Open University, hence it provides an ideal topic to investigate how students learn by interacting with a 3D environment when they have limited knowledge of the subject matter.

Approach

We designed a 3D virtual building (the POE Lodge) which contained teaching elements covering the essential process and documentation aspects of Problem Oriented Engineering (POE). With a lack of specific guidelines or principles to guide the construction of 3D educational environments, the building and its contents were designed by applying and generalising principles developed for 2D multimedia applications. The POE Lodge was designed and built to the same
shape as the POE process diagram (Hall & Rapanotti, 2009), and a case study was included in the design, augmented with narratives text and audio narratives, structural elements from the POE process and web links.

The POE Lodge was evaluated with real users. As users of the simulation moved around and interacted with objects in the building, their interactions were recorded and as a result their activity was tracked through the POE Lodge. As users exited the building they were invited to complete a questionnaire to assess their knowledge assimilation, their perception of the environment, and to gather information on their previous experience in Second Life.

**Results**

The POE Lodge environment was visited a total of twenty times by ten distinct individuals over a period of two weeks. Following the visits six questionnaires were started, however some questionnaires were not fully completed. Of these only three of the respondents had no prior knowledge of POE. Two of these respondents scored 60% on their knowledge assimilation (although one did not complete all the questions), which could be regarded as a reasonable level of assimilation. The third respondent did not score so highly and remains an anomaly, as this user stayed in POE Lodge for longer than respondent number two, and had more interactions in total and with more distinct objects than both the other respondents.

![Figure 2 POE Lodge courtyard showing POE process & documentation elements](image)

![Figure 3 Knowledge assimilation in respondents with no prior experience of POE](image)

Users of the environment were also asked about how they viewed the ease of use of the various components within POE Lodge. The responses to this question varied depending on how much experience a user had of Second Life. In Figure 4, a lower score indicates a POE Lodge entity that was regarded as easier to use. From this it is clear that frequent users found...
the narratives and Second Life notecards easiest to use whilst regular and occasional users for the most part found the items of near equal value.

Figure 4 Ease of use of POE Lodge facilities

This difference between frequent users perceptions and those less familiar with Second Life is probably associated with a gradual change to use those parts of Second Life that are closer

Figure 5 Usefulness of POE Lodge facilities
to real life. This is reflected in the learners’ perceptions of the usefulness of the various parts of POE Lodge. Despite the fact that experienced users found the wall slides the least easy to use, from Figure 5 (again a high score indicates something regarded as less useful), it is possible to see that experienced users ranked them as equally useful alongside the narratives and notecards. Less experienced users tended to rank items as having different usefulness focusing on the slides and notecards; items that are more familiar in a real world educational setting.

In summary, based on the data collected on knowledge assimilation, the adoption and generalisation of 2D multimedia design principles to underpin the design of the POE Lodge appears to have been successful.

In general however whilst less experienced users found the facilities of the environment easy to use, those with more experience of Second Life ranked the facilities as less easy to use. The more experienced Second Life users also ranked the facilities of equal usefulness while those with less experience tended to have a preference for what can be regarded as the more traditional learning aids, namely the wall slides and notecards. This differentiation between experienced and less experienced Second Life users is expressed directly by one survey respondent who wrote

“\textit{I think cut down the information to read in the exhibit - the bits that it's good at getting across are the four stages and the avatar can walk those steps, but SL's not good for encouraging lots of reading.}”

Discussion

Although the use of multimedia design principles has in this study been at least a partial success, there are a number of aspects of design in virtual worlds that need to be investigated further.

There appears to be a small but noticeable difference in the way in which experienced users, and those with less experience, view the facilities available in POE Lodge and we would expect this to be the case with other environments. This difference can be explained at least in part, by the fact that experienced users are more familiar with how the Second Life works and how to manipulate and use the facilities available.

Rather like computer gamers learning a new game, those with less experience seem to favour real world analogues, due to the fact that they are familiar with how these work. As users gain experience, they tend to use the more novel facilities that may not exist in the real world. This view is shared by Minocha and Reeves (Minocha & Reeves, 2009), who find that the aesthetics of an environment can have a significant effect on how it is viewed and used.

Multimedia design principles do go some way towards informing the design of virtual world educational simulations. Further work is however needed to ensure that those with less experience in a 3D environment are not disadvantaged, whilst taking full advantage of novel teaching approaches possible in worlds where “game” world experiences such as flying and teleporting are possible.
References


# Index of authors

<table>
<thead>
<tr>
<th>Author</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhi Jin</td>
<td>30</td>
</tr>
<tr>
<td>Dariusz W. Kaminski</td>
<td>35</td>
</tr>
<tr>
<td>Aleksander J. Kavs</td>
<td>24</td>
</tr>
<tr>
<td>Luigi Lavazza</td>
<td>7</td>
</tr>
<tr>
<td>Zhi Li</td>
<td>42</td>
</tr>
<tr>
<td>Yan Lucas</td>
<td>14</td>
</tr>
<tr>
<td>Angelika Mader</td>
<td>14</td>
</tr>
<tr>
<td>Derek Mannering</td>
<td>37</td>
</tr>
<tr>
<td>Jelena Marincic</td>
<td>14</td>
</tr>
<tr>
<td>Andrew Moore</td>
<td>45</td>
</tr>
<tr>
<td>Ann Nkwocha</td>
<td>39</td>
</tr>
<tr>
<td>Jie Sun</td>
<td>24</td>
</tr>
<tr>
<td>Ye Wang</td>
<td>24</td>
</tr>
<tr>
<td>Xinyu Wang</td>
<td>24</td>
</tr>
<tr>
<td>Roel Wieringa</td>
<td>14</td>
</tr>
<tr>
<td>Xiaohu Yang</td>
<td>24</td>
</tr>
<tr>
<td>Bin Yin</td>
<td>30</td>
</tr>
</tbody>
</table>