Clarifying the semantics of value in use cases through Jackson's Problem Frames

José M. Cañete-Valdeón *, Fernando Enríquez, Javier Ortega, Ernesto Veláquez

Department of Computer Languages and Systems, University of Sevilla, Spain

Abstract

Use cases constitute a popular technique to problem analysis, partly due to their focus on thinking in terms of the user needs. However this is not a guarantee for discovering all the subproblems that compose the structure of a given software problem. Moreover, a rigorous application of the technique requires a previous consensus about the meaning of I. Jacobson's statement "a use case must give a measurable value to a particular actor" (The Rational Edge, March 2003). This paper proposes a particular characterisation of the concept of "value" with the purpose of problem structuring. To this aim we base on the catalogue of frames for real software problems proposed by M. Jackson (Problem Frames, 2001) and we reason about what could be valuable for the user on each problem class. We illustrate our technique with the analysis of a web auction problem.

Keywords: Software engineering; Problem analysis; Use cases; Problem frames

1. Introduction

Use cases [6–8] are a popular technique to problem analysis. The main hypothesis held by the approach is that software systems provide *values* to people, which are delivered in discrete episodes of uninterrupted, intense interaction. In such episodes people who receive value can play one or several roles called "primary actors". A use case groups all the interactions aimed at delivering a certain value (including those that eventually fail in the purpose).

* Corresponding author. Tel.: +34 954 553 873. Postal address: Departamento de Lenguajes y Sistemas Informáticos, ETS de Ingeniería Informática, Avenida Reina Mercedes, S/N, 41012 Sevilla, Spain.

E-mail address: jmcv@us.es (J.M. Cañete-Valdeón).

Focusing on the user needs is regarded as a useful approach to the analysis of a problem [11]. However, in general, such a focus is not enough in order to develop a complete understanding of the problem concerns and difficulties. Consider the analysis of a fire alarm problem [1]. The value that the system delivers to the residents of a building is safety against fires: to be informed of any evidence of a fire. However such a benefit is not obtained as the result of a discrete episode of interaction between the user and the system, but rather *during* the *continuous* functioning of the system since the very moment in which it was connected. The obvious user-system interaction is "connect system", but this is not a use case in the classic sense as nothing valuable is *returned*.

Of course one may argue that the former example corresponds to an autonomous control system with

a minimum user participation and hence the use-case technique is simply not suitable. However the information system that illustrates this paper, with plenty of user interactions, is another example where use cases are not enough to analyse the problem as it will be shown. As Jackson argues [4], problem analysis cannot be restricted to the interface between the user and the computer since the problem is generally deeper into the world.

Another important difficulty, which this paper focuses on, is that the use-case approach lays on the concept of "value" but there is no bibliographic reference where this concept is rigorously addressed (to the best of our knowledge). The cause of this situation is that the approach has not been developed to the extent where there exists a clear understanding of the main types of value that are commonly delivered by real-world software systems. Note that the origin of the technique was the observation of telecommunication systems, which led to an initial conception of "value". The problem lies in that such a specialised meaning does not fit other categories of systems. Consider a web auction system, which will be our running example. A brief description follows.

Web auction system. A system is needed to automate English-type auctions on the web. People access the system through a standard browser. Access is public but a previous registration is required, which grants a credential. The latter must be presented to the system in subsequent interactions. Sellers request auctions for items to sell, indicating the minimum admissible bid as well as initial and end times. Bids are accepted until the end time is reached; when this happens, the auction winner is the bidder who has issued the highest price. Bids are processed as received.

Contrary to the fire alarm system, many user-system interactions can be identified in this problem; for example:

- Register in the system.
- Place a bid on an auction.
- Browse all active auctions.
- Request an auction for an item.

Which interactions can be regarded as valid use cases? This depends on what the analyst understands as "measurable value" [6]. Each analyst in a team will probably have her opinion in this respect for each one of the former points. If the team is only seeking an agreement, the solution is easy: discuss until a consensus is reached. However, use-case models are not just mere project documentation but a tool to engineer the system. Therefore choosing the "right" use cases is important. How can the analysts know that they have chosen the "right" use cases?

The definition of "value" depends on the methodological context as it will condition the use cases that the analysts will identify. For example, the Rational Unified Process (RUP) methodology is use-case driven. RUP lays on the basis that engineers will identify the right use cases at the requirements workflow, which are claimed to determine the system architecture to a great extent [7]. Such use cases are those that provide an "observable result of value" [10]. The authors of RUP may have a precise idea of what this means, but it is not described in the bibliography. Identifying "wrong" interactions as use cases will make the method not succeed.

Evidences of this lack of rigorous semantics for "value" can be found by examining the use-case bibliography. Thus, in a recent work, Bittner and Spence [1] employ the term without providing a clear characterisation of the concept. They simply state that individual functions of the system, such as "validating a password", are *not* use cases [1, pp. 106–108]. All we can deduce from this claim is that the results of such functions should not be regarded as valuable, but why not?

A different approach is taken by Cockburn who argues that systems help primary actors to reach *goals* [3, p. 27]; therefore we can infer that the obtained value is to get some goals accomplished. The author identifies three categories of typical goals according to their granularity: summaries, user goals, and subfunctions. Any of them may refer to varying scopes of "system", such as the whole company or the software system only. In Cockburn's approach, *every* user goal is unfolded into a use case, as well as certain summaries and subfunctions [3, pp. 66–67]. The author provides a simple characterisation of what should be understood as a user goal, including a time span (it typically takes from two to twenty minutes), satisfaction (the primary actor is claimed to go away happy after the interaction with the system), and an increment of the user's performance (when iterating the goal).

The aim of this paper is to propose some categories of values that help the engineer to identify use cases in the analysis of a problem. While many alternative characterisations of the concept might be considered, the one reported here will be guided by the purpose of *structuring the software problem*. For this reason we have based on a well-validated theory: Michael Jackson's Problem Frames [4]. Such a work is a thorough study on software problem analysis. The author identifies a num-

ber of classes (frames) of software problems that occur with frequency in practise, often in combination. Our strategy consists of specialising each frame in order to accommodate the special features of the use-case view. For each particularised frame we discuss which results from the system could be regarded as valuable for the user. This leads to the obtention of a catalogue of types of values for frequent classes of problems. The validity of our approach is supported by its roots on a sound theory about software systems.

The rest of the paper is organised as follows. In Section 2 we report the obtention of the catalogue of value types and suggest some guidelines for its application. Section 3 illustrates the usefulness of the proposed catalogue with the analysis of the web auction problem. The paper closes with conclusions in Section 4.

2. A catalogue of frequent values delivered by software systems

The Problem Frames approach [4] thoroughly characterises five classes of problems which are claimed to *commonly* occur as components of real-world software problems. They are named "basic frames". Each one presents a physical layout, i.e. a particular distribution of the real-world domains that take part in the problem, including the computer system itself (which is referred to as the "machine domain"). A requirement determines conditions on the domains that must be satisfied in order to solve each problem class. We first present a very brief overview of the basic frames and next we particularise each one to the scheme of the use-case approach. At the end of the section we propose some guidelines for discovering use cases.

2.1. The basic problem frames

The layout of the "required behaviour frame" consists of a machine domain and a controlled domain; communication takes place over an interface of shared phenomena. The requirement is to achieve that the controlled domain behaves in a certain, preestablished way, so such a domain is assumed to be causal (i.e. its behaviour can be predicted) [4, pp. 85–86].

A variant of the former frame includes an explicit (human) operator who communicates with the machine and can issue commands. The requirement "constrains the behaviour of the controlled domain by describing general rules for its behaviour and specific rules for how it must be controlled in response to the operator's commands" [4, p. 90]. This variant is called the "commanded behaviour frame".

The "simple workpieces frame" is somewhat similar to the former one in that there is also a user who issues commands to the machine [4, p. 125]. The main difference is in the characteristics of the central domain (named "workpieces"): although it has a causal aspect supporting its operations and their effects, its main significance is lexical [4, p. 127]. In consequence it is inert: it may change its state in response to an externally controlled event, but it initiates no state changes and no events [4, p. 97]. The problem requirement stipulates what effects the commands issued by the user to the machine should have on the symbolic values and states of the workpieces [4, p. 97].

An important class of problems is represented as the "information display frame". The layout consists of a machine connected, on the one hand, to some (causal) domain of the real world and, on the other, to a domain with display capabilities. The requirement is that certain information about the real world is continually needed, and it must be presented at the display; thus a correspondence is stipulated between the symbolic requirement phenomena of the display and the causal requirement phenomena of the real world [4, p. 93].

A last basic frame is called the "transformation frame". The machine has access to some computer-readable input files. The requirement is that some output files must be derived by the machine from the given data, and their contents and format must follow certain rules [4, pp. 99–100].

2.2. Identifying values in the basic problem frames

As we explained at the introduction, the theory behind the use-case technique assumes a simple layout consisting on a user interacting with the system, as well as a simple requirement: such an interaction must deliver some value to the user playing the role of primary actor. Therefore we have to specialise the different frames in order to fit such conditions.

Only two basic frames explicitly include a human domain interacting with the software system, namely "commanded behaviour" and "simple workpieces". We specialise the remaining three by inserting an explicit human domain that interacts with the machine. This is allowed by the Problem Frames approach (an "operator variant"; see [4, pp. 214–219]).

The "required behaviour frame" with the addition of an operator is equivalent to the "commanded behaviour frame". Note that the problem requirement on the behaviour of the controlled domain upon a command is not restricted to discrete action courses but the specification of a continuous activity is also allowed. For example,

a security system may be responsible for activating the alarms of a building whenever a security violation is detected, once the operator has issued a "connect system" command. While classic (Jacobson's) use cases seem to be restricted to the achievement of a discrete behaviour at the end of the interaction, our characterisation will also embrace continuous behaviour as a valuable result in a control problem: when the interaction finishes, the user can "go away happy" expecting that the system is ensuring a continuous, desired behaviour of some part of the world.

The beneficiary may be the operator herself, or she may act on behalf of another user. The latter layout can be obtained by making the operator be a "connection domain" between the machine and another domain representing the real beneficiary. This is allowed in Problem Frames under the topic "connection variants" [4, pp. 219–229].

What is the profit that the user can obtain? The execution of one particular command may or may not be a valuable achievement. We would need more information about the concrete problem to decide this. For example, the value delivered by a painting robot on a car manufacturing plant is to get a car painted; this may require more than one command from the operator. At the abstraction level of the "commanded behaviour frame" all we can assert is that the value obtained by the user is twofold: some part of the world will behave in a desired way[1] *and* the guarantee that this is achieved with the available set of commands at the interface with the machine.

In the specialised "information display frame", the user (directly or through a connection domain) issues enquires to the system, requesting some information about a certain physical domain. Such a variant of the basic frame is so common that Jackson gives it a name: the "commanded information frame" [4, pp. 215–216]. The information is the value that the user obtains.

The "simple workpieces frame" allows the user to create and edit a certain class of computer-processable entities. Therefore the value for the user is constituted by the workpieces obtained according to the commands issued by the user herself.[2] The symbolic phenomena of such workpieces will often have some meaning to the human user or to other people [4, p. 97].

The "transformation frame" deals with lexical domains. The value for the user is twofold: the output *and* the guarantee that such an output conforms to the conversion rules. The interaction between the operator and the system includes commands such as "start", which instructs the machine to begin the process, as well as any other data required by the system during the transformation. For example, LaTeX transforms TEX files into DVI files according to certain rules. The system issues a prompt when some error is found during compilation; an operator command is then required to resume the transformation.

Table 1 summarises our conclusions. As we have based on the Problem Frames theory, we can be confident that these values commonly happen in real-world software problems.

2.3. Guidelines for use-case identification

Next we suggest some guidelines in order to help the reader to apply the values of Table 1.

- Study the (physical) domains in the problem context, as indicated by Jackson in the Problem Frames approach [4, Ch. 2]. Pay attention to the different kinds of users and the roles they play in the problem. Note that the users of the system are not necessarily those who directly interact with the computer but one or more connection domains may exist (e.g., in a library administration problem, a librarian may use the machine on behalf of the library members who are users in their own right).
- Focus on the users' high-level goals in the problem context, i.e. goals related to the problem but far away from the computer interface. A good technique for determining whether you are overgeneralising the goals is to anchor yourself in your cus-

Table 1
Values obtained by the user from each basic problem frame, in the particular case that he benefits from the system through discrete episodes of interaction

Basic problem frame	Obtained value
Required behaviour + operator (commanded behaviour)	Some part of the world will behave in a desired way with the guarantee that this is achieved with the available set of commands at the interface with the machine.
Information display + operator	Information about the world.
Simple workpieces	The workpieces obtained according to the commands issued by the user.
Transformation + operator	The output and the guarantee that such an output conforms to the conversion rules.

[1] Despite the non-applicable or even dangerous commands that the operator might issue.

[2] And, as in the commanded behaviour frame, despite the non-applicable or dangerous commands that the operator might issue.

Table 2
Some subproblems of the web auction problem. They are named according to the names of their machines

Subproblem	PF	Domains	Trace
Register machine-1	WP	Credentials (workpieces), User (operator)	UC11
Info. machine-1	ID	Auctions* (real world), Web browser (display)	UC12
Contract machine-1	WP	Contracts for bidders (workpieces), Current bidder (operator)	UC13
Control machine-1	CB	Bidders (controlled domain), Current bidder (operator)	UC13
Register machine-2	WP	Declarations of payment (workpieces), Winning bidder (operator)	UC2
Contract machine-2	WP	Auction contracts (workpieces), Seller (operator)	UC31
Info. machine-2	ID	Declarations of payment (real world), Web browser (display)	UC32
Register machine-3	WP	Declarations of shipment (workpieces), Seller (operator)	UC33
Model builder	ID	Auction contracts & clock (real world), Auctions Model (display)	–
Control machine-2	CB	Bidders (controlled domain), Current bidder (operator)	–
Control machine-3	RB	Users (controlled domain)	–

The last column indicates the use case that motivated each subproblem. Abbreviations: PF (problem frame), RB (required behaviour), CB (commanded behaviour), ID (information display), and WP (workpieces).

* As explained in the text, web auctions are (socio-technical-legal) processes. The only way the system can observe auctions is through a model, but this will be discovered later (with the introduction of the model builder subproblem). Therefore the Auctions domain should be replaced for the Auctions Model domain.

tomer: do not go beyond the customer's authority in the problem [4, pp. 29–33].
- Identify the value that can be obtained from each goal by referring to Table 1. If no value can be identified, the goal might be hiding a composite problem with multiple values for the user.[3] This may be the case even if a value has been successfully identified. An approach is to find a refinement of the goal such that the obtained subgoals are sufficient for satisfying the higher-level one, and try to identify their values in the table. If some value(s) has been identified for a goal, allocate a use case for such a goal.
- Operationalise each goal by means of a success scenario. Then try to identify which values in the table can be obtained from each step. This process can be repeated until the steps are so simple that they do not provide any value to the user (in Cockburn's model such steps correspond to low-level subfunctions). If some value has been identified for a step, allocate a use case for such a step. Allocate one additional use case for composing those that have been obtained from the scenario, unless such a use case has previously been allocated for the goal.
- The identified values indicate the existence of subproblems. Therefore a preliminary decomposition in frames has been obtained. Additional frames can be discovered by considering the concerns of the already identified ones as well as by studying the problem domains [4]. The decomposition heuristics proposed by Jackson will be specially useful.

[3] It is noteworthy that the basic frames identified by Jackson are typical but they do not constitute an exhaustive catalogue [4, p. 351]. Therefore the values of Table 1 cover many but not all the possible goals that one can potentially find in a given problem.

3. Applying the catalogue of values to the web auction problem

This section presents a partial analysis of the web auction problem (introduced at Section 1) according to the proposed technique and explains how the use cases and the subproblems are discovered. Table 2 summarises the resulting problem analysis.

We distinguish two roles that people play in their interactions with the auction system: bidder and seller. They constitute our primary actors. We begin by studying their goals in the world, far away from the computer interface.

3.1. Bidders

A bidder is a person ultimately interested in *acquiring an item in a web auction*, at least as far as the problem concerns. In a first approach, the benefit obtained from this goal seems to be the item itself, but this value is beyond those of Table 1 and therefore we still have not got a criterium for allocating a use case related to the goal. Let us try to operationalise the goal with a scenario:

(i) Win the item in a web auction.
(ii) Pay for the item.
(iii) Pick up the item when it arrives.

Only the first two steps seem to require an interaction with the machine. Let us try to identify their associated values.

The value of winning an item in a web auction is the obtention of a legal right (plus an obligation) to buy the item. In other words: the establishment of a contract. Contracts are conceptual entities, physically represented in paper or, in this case, as workpieces. The creation of such workpieces is valuable for the bidders. This agrees with Table 1, which reveals a workpieces subproblem. Therefore we can allocate a use case (UC1) for this goal.

However, on the one hand, it may not be obvious for the analyst to discover this value at the outset. On the other hand, winning an item is a complex goal and we do not know whether it hides additional values. For these reasons we recommend to operationalise the goal with a scenario. One possibility is:

(i) Register in the system.
(ii) Locate an auction where the desired item is being sold.
(iii) Place bids on the auction.

Register is related to the creation of a credential that enables access to the system services. A credential is a computer-processable entity and, just like a workpiece, it can be created, modified, and possibly deleted. Therefore this can be regarded as a workpieces subproblem, where the observable value is the credential itself. According to the problem statement, such an object must accompany all the subsequent events that the user issues to the system. In Problem Frames this can be represented as roles, which describe the participants in an event [4, pp. 80–81]. For example, assume that *PlaceBid*(e) denotes that e is an event of class *PlaceBid*. Then we define some roles: *Credential*(e, c) denotes that c plays the role of the credential in event e, *Bidder*(e, b) denotes that b plays the role of the bidder in event e, and *Amount*(e, q) denotes that q plays the role of the amount in event e.

The value of locating an auction is obtaining information about a reality: the active auctions. Just like loans in a library system are better regarded as processes than as entities [4, pp. 171–172], an auction can be thought of as a process too, i.e. a collection of events that are ordered in time: a seller requests to auction an item, the system indicates that the auction begins, bidders place bids, the system indicates that the auction ends. It is important to note that these events are associated to (temporary) legal relationships. We will explain them throughout this section. Therefore this is an information display subproblem: certain information about the auctions is needed.

Finally, placing a bid on an auction pursues two goals. On the one hand, to establish a contractual relationship between the bidder and the seller. This contract determines rights and obligations. For example, the bidder commits herself to pay an amount equal to the placed bid for the item if she eventually becomes the auction winner. Shipment costs can be included in the contract. In a web auction, contracts are represented by computer-processable entities. According to Table 1, the creation of such lexical entities is the obtained value in a workpieces subproblem. This was the value that we originally identified for the goal *Win the item in a web auction*.

On the other hand, an auction is a social process where placing a bid has an effect on the other bidders. From the viewpoint of the user who places the new bid, a (secondary) goal is to achieve that the user who held the maximum bid looses her right to buy the item. This is a kind of control on the behaviour of the competing bidder: she will not be able to buy the item when the auction ends (unless she bids again before this happens). According to Table 1 a desired behaviour is a value in itself and it corresponds to a commanded behaviour subproblem[4] with the user who places the current maximum bid as operator. The requirement for this subproblem can be extended to the whole Bidders domain and formulated as: bidders different from the one who places the current maximum bid will not be able to buy the item when the auction ends.

In summary, we have identified three use cases included in *Win the item in a web auction* (UC1):

- UC11: *Register in the system*. Value: creation of a workpiece (the credential).
- UC12: *Locate an auction*. Value: information about a reality (an auction where the desired item is currently being sold).
- UC13: *Place a bid*. The value is composite:
  - Create a contractual relationship between bidder and seller. Value: creation of a workpiece (the contract).

[4] The commanded behaviour frame regards the controlled domain as *causal*. In the particular case of this subproblem, the Bidders domain is modelled as a causal domain. This makes sense if we regard that a bidder needs to issue a *PaymentComplete* event to the system in order to buy an item and being its legal owner; if such an event is inhibited for a particular bidder, then it is impossible for him to legally acquire the item. This causality grants the machine a certain control over the Bidders domain.

- Avoid that competing bidders acquire the desired item. Value: to control a domain (all the bidders except the currently winning bidder).

Regarding the second step in the scenario of acquiring an item in a web auction, *Pay for the item*, let us assume for simplicity that payments are made by means different from the system: bank transfers, postal sending, etc. The value of this goal is to fulfil the auction contract and therefore it does not appear in Table 1. In order to identify possible use cases we elaborate a success scenario for this goal:

(i) Make a bank transfer to the seller for the value that is specified in the contract.
(ii) Indicate to the system that the payment has been made.

The machine does not participate in the first step. The second one denotes a goal whose value is to physically *register* a legal declaration: the winner of the auction has made the payment and fulfilled the contract. This declaration has the form of a lexical unit, and its creation is valuable for the winning bidder. Therefore this is a workpieces subproblem. We allocate a use case (UC2) for the goal of the second step. Such a use case is initiated by a *PaymentComplete* event.

For completeness, it is useful to add one additional use case that includes the ones that have resulted from the scenario of the goal of acquiring an item in a web auction. We may denote it as UC0 (*Acquire an item in a web auction*). Its value is the composition of the values of the included use cases (UC1 and UC2).

3.2. Sellers

A seller is ultimately interested in *selling an item*. As far as the problem concerns, a possible way to satisfy this goal is by *auctioning the item on the web*, a more specific goal. In both cases the value is to have the item sold, which is measurable (e.g., as a relation of the economic gain and the time invested on the sale). However this value is beyond those of Table 1 so it is probably of little usefulness if our purpose is structuring the problem.

It is noteworthy that the goal *auctioning an item on the web* adds an additional value that is not necessarily present in the generic *selling an item*: the obtention of a legal contract by which someone commits himself to buy the item. We know that such a contract is represented in the problem as a workpiece, and its creation is regarded as valuable according to Table 1. Therefore we identify a use case (UC3) related to the goal *auctioning an item on the web*. In order to discover additional values we operationalise the goal as a success scenario:

(i) Register in the system.
(ii) Request auctioning the item at a certain date.
(iii) Wait for the payment from the auction winner.
(iv) Send the item to the winner and indicate this to the system.

Registry has already been covered as use case UC11. Request auctioning an item is an event associated to a legal contract, with rights and obligations. For example, the seller commits herself to sell the item to the auction winner for the same price as the maximum placed bid (shipment costs can also be included). The resulting value is that such a contract is created. As contracts are represented by lexical units, we have a workpieces subproblem. We identify a new use case (UC31).

Waiting for the payment requires information about the moment when the associated workpiece is created, so we have another information display subproblem. The part of the real world under observation is constituted by the declarations of payment. We tag the new use case as UC32.

Finally, indicating that the item has been sent is an event from the seller to the system. The value is to physically register a declaration: the seller has complied with his part of the contract. If such a declaration is regarded as a lexical unit, this is again a workpieces subproblem. We identify a new use case (UC33).

3.3. Additional subproblems

So far we have focused on the users' point of view, obtaining a number of subproblems. However this perspective does not ensure a complete problem analysis. There are additional concerns and difficulties that can only be discovered from other perspectives, by studying all the domains in the problem and not just the so-called primary actors. For brevity we shall only indicate some of the subproblems that have been overlooked.

Studying the Auctions domain uncovers new subproblems. We previously argued that auctions could be regarded as processes. In traditional English-type auctions the auctioneer determines the beginning and the end of the process. In web auctions this responsibility rests on the computer system. This can be regarded as a model-building subproblem [4, Ch. 7], which is a specialised information display frame. The part of the real world under observation is the domain constituted by the auction contracts and the clock. When the current

date and time are equal to the starting date and time stipulated at the contract of an auction, the model builder machine issues an *AuctionStarted* event, which is registered in the model. The end of an auction is similar. Such a model is a lexical domain which can be consulted by other subproblems, e.g., to determine the current active auctions (remember the information display subproblem that appeared at UC12).

An interesting event in the auction process is the placing of a bid. An overlooked problem concern is a property of English auctions [9]: once a bid b is placed for an amount q, no other bid b' can be placed for an amount q' if $q' \leqslant q$. This can be regarded as a commanded behaviour subproblem on the Bidders domain: when event *PlaceBid*(e) happens, no bidder will be able to participate in an event *PlaceBid*(e') if *Amount*(e, q) and *Amount*(e', q') and $q' \leqslant q$.

Finally, studying the Credentials domain (which appeared at UC11) we realise that only users with a credential in such a domain are allowed to access the system services. This constraint determines a required behaviour subproblem on the users: they will not be able to make any service request e to the system if *Credential*(e, c) and c does not belong to the Credentials domain.

4. Conclusions

Focusing on the users and their needs is a useful approach to the analysis of a software problem [11]. The use-case technique naturally fits such an approach and it is broadly extended. Further, important software development methods such as RUP [7,10] are based on this focus on the user. While we do not question the benefits of the strategy, we argue that it is not enough to achieve a complete problem analysis and structuring. Jackson's Problem Frames [4] is a rigourous and validated approach to the same purpose, and it allows to achieve a complete understanding of the software problem. In this paper we have provided a simple road for analysts who feel comfortable with use cases towards the benefits of problem frames. The obtained use cases and frames constitute a sound basis for further problem analysis and solution development (e.g., architectural design [5,7]).

Analysts who address problems with the use-case approach often find themselves with the question of what should be regarded as "valuable" for the users of the system-to-be. Many characterisations of the concept may be proposed for different purposes. In this paper we have reported one with the purpose of employing the use-case technique to problem structuring. To this aim we have based on the catalogue of basic problem frames. We have also proposed some guidelines for applying the characterisation to problem analysis.

The so-called CRUD use cases [3, Ch. 14] are an example of the consequences of a fuzzy semantics for "value". The word is an acronym for typical operations on databases: create, retrieve, update, and delete. As Cockburn points out [3, p. 145]: "So far there is no consensus on how to organise all those little use cases". If they provide real value to the user, they should explicitly appear in the analysis. But, again, what is value? Our approach ensures that only those CRUD use cases which can be associated to a workpieces subproblem in the problem context are relevant to the analysis; the remaining ones are actually part of a particular system implementation (e.g., the manipulation of a database). There is one exception to this rule: the introduction of a model domain, which is actually part of the solution [4, p. 182]. However, in the Problem Frames approach the introduction of such a domain is controlled; it is justified solely in terms of understanding the problem [4, Ch. 7].

Problem frames are accompanied by a convenient graphical notation named "frame diagrams" [4, App. 1]. While we have not employed diagrams to illustrate the web auction example, they can be easily generated from the domains of Table 2 and the requirements described in Section 3.

We have not proposed a notation for linking use cases with their corresponding frames but we have denoted such associations in a tabular form. Another simple option could be to annotate use-case descriptions with the names of the related subproblems, which can also be shown as notes attached to the ellipses in use-case diagrams.

The work presented in this paper is the elaboration of an idea included in Cañete-Valdeón's PhD thesis [2]. We must remark that the proposed characterisation is an initial contribution to clarify the concept of value. It can be refined by further studying each problem class and by identifying new frames.

Acknowledgements

The authors want to explicitly express their gratitude to the anonymous reviewers. Their comments have greatly contributed to improve the clarity and quality of this work.

References

[1] K. Bittner, I. Spence, Use Case Modeling, Object Technology Series, Addison-Wesley, 2003.

[2] J.M. Cañete-Valdeón, A theory of languages and design methods in software engineering, PhD thesis, Universidad de Sevilla, 2006.

[3] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2001.

[4] M. Jackson, Problem Frames. Analyzing and Structuring Software Development Problems, Addison-Wesley, ACM Press, 2001.

[5] M. Jackson, Problem structure and dependable architecture, in: R. de Lemos, C. Gacek, A. Romanovsky (Eds.), Architecting Dependable Systems III, Springer-Verlag, 2005.

[6] I. Jacobson, Use cases—yesterday, today, and tomorrow, The Rational Edge, March 2003.

[7] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Object Technology Series, Addison-Wesley, 1999.

[8] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, ACM Press, 1992.

[9] P. Klemperer, Auctions: Theory and Practice, Princeton University Press, 2004.

[10] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley Professional, 2003.

[11] D. Leffingwell, D. Widrig, Managing Software Requirements: A Unified Approach, Addison-Wesley, 2000.