Expert Systems

Article

DOI: 10.1111/j.1468-0394.2012.00648.x

Business goals, user needs, and requirements: A problem frame-based view

Luigi Lavazza

Università degli Studi dell'Insubria, Dipartimento di Scienze Teoriche e Applicate, Varese, Italy
Email: luigi.Lavazza@uninsubria.it

Abstract: *It is well known that the analysis of requirements involves several stakeholders and perspectives. 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 some confusion – especially in industry – about what really a user requirement is, what are the differences between user requirements and user needs, and what are their relationships with business processes. The paper aims at clarifying the aforementioned issues, by providing a systematic and clear method for establishing requirements hierarchies. The problem of describing requirements hierarchies is tackled using the problem frames concepts and notation. A case study is used throughout the paper to illustrate the proposed approach. The description of requirements at different levels of abstractions and requirements hierarchies are illustrated. The resulting models are coherent with the reference model for requirements specifications and the problem frames. An analysis process that is aware of the differences between user needs and requirements is also provided, to illustrate the process of refining high-level goals into requirements that can be satisfied by a hardware/software machine. The proposed method appears promising to model, study, and evaluate the relationships between business processes and the strategies for achieving business goals based on the usage of information technology.*

Keywords: user needs, user requirements, problem frames, requirements analysis, knowledge elicitation

1. 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 goals', 'user needs', 'user requirements', etc. Actually, there is some confusion – especially in industry – about what a user requirement really is, what the differences (if any) are between a user requirement and a user need, and what is their relationship with business processes. Moreover, how effectively and correctly to 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.

The confusion of goals, needs, and requirements is problematic because the same goal can give place to different – often incompatible – sets of needs, and the same need can originate different requirements. Therefore, in proceeding from goals to requirements, evaluations have to be performed and choices made. Moreover, the 'distance' that separates goals from requirements is often great, thus 'jumping' directly from goals to requirements is generally infeasible in practice: intermediate refinement steps are required. Examples of these issues are given in Section 3.

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

More specifically, the paper aims on the one hand at clarifying the nature of requirements that are conceived and described at different levels (from the highest business-related goals to software requirements specifications), on the other hand at providing a description of requirements that is homogeneous through the various levels. This description takes the form of a hierarchy of requirements and has a set of desirable characteristics:

1. It is coherent with previous work on goal-oriented requirements engineering (RE).
2. It is based on the reference model for requirements and specifications (Gunter *et al.*, 2000).
3. It uses problem frames, thus enabling the usage of the concepts and methods described in Jackson (2001).
4. The definition of a homogeneous hierarchy of requirements makes clear that the nature of requirements definition is basically the same at different levels.
5. It connects high-level business-oriented analysis with Problem-Oriented Software Engineering (POSE) (Hall *et al.*, 2008; Hall & Rapanotti, 2011), which typically focuses on software requirements.

A process is defined to provide guidelines to analysts in applying the proposed concepts, and to make explicit that the process involves (a) refinements and (b) feasibility and cost evaluations, since problems often have several possible solutions, of which some could be very hard or expensive to achieve.

The paper is organized as follows: the problem of requirements modelling is introduced in Section 2. The involved

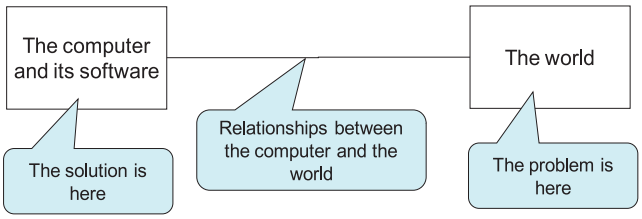

Figure 1: *The problem and solution spaces.*

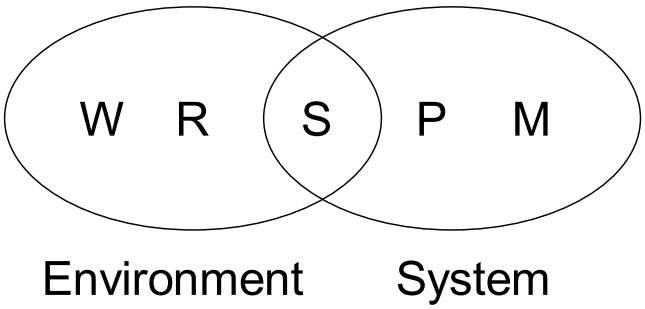

Figure 2: *The elements of the reference model for requirements and specifications.*

issues are presented in Section 3 with the help of a case study. The distinction between needs and requirements is further discussed in Section 4, and an analysis process aware of this distinction is sketched in Section 5. Section 6 reports related work, while Section 7 draws some conclusions and provides an outline of future work.

Throughout this paper, the concept and notation of problem frames (Jackson, 2001) are used. The reader is expected to know them.

2. Requirements modelling

Requirements modelling is of fundamental importance in the software development process. Among others, Jackson *et al.,* have contributed a problem-oriented view of RE (Gunter *et al.,* 2000; Jackson, 2001; Hall *et al.,* 2008; Hall & Rapanotti, 2011). The material reported in this section is mainly based on such work.

2.1. The nature of requirements

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. Figure 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 build 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.

In order to define precisely the meaning of requirements modelling it is fundamental 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, since 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.

These concepts are properly defined in the 'Reference Model for Requirements and Specifications' (Gunter *et al.,* 2000). The model is based on five artefacts, some of which pertain mostly to the system, while others pertain mostly to the environment (Figure 2 (Gunter *et al.,* 2000)). These artefacts are:

1. Domain knowledge that describes environment facts; this artefact is denoted as W (for the World, which includes the environment).
2. Requirements that indicate the desired situation that must be achieved in the environment as an effect of the introduction of the system in the environment; this artefact is denoted as R.
3. Specifications of the system, which provide enough information for a programmer to build the system: specifications do not need to communicate to the developer unnecessary information about the environment; this artefact is denoted as S.
4. A program that implements the specification using the programming platform; this artefact is denoted as P.
5. A programming platform that provides the basis for programming the system; this artefact is denoted as M (for Machine, which is the programmable platform).

Of course, we should guarantee that the requirements are satisfied (i.e. we build the right system) and the program correctly implements the specification (i.e. we build the system right).

Concerning P and M, it is useful to note that in case of embedded software the distinction between P and M on one side and W and R on the other side is quite clear, as the interaction between the environment and the system occurs via sensors and actuators; thus, people belong to the environment. On the contrary, when traditional 'business' applications are concerned, the 'system' usually includes people who use the software applications and execute 'manual' procedures. In these cases, the 'machine' is actually composed by both a hardware/software platform and people, and the 'programs' are partly software and partly rules and procedures that people have to follow. In 2005, Hall and Rapanotti extend the reference model to explicitly represent the 'social component', that is the people, who can be 'instructed' – as machines are programmed – for the purpose of satisfying the requirements. The result is that a third ellipse is added to the model in Figure 2.

Figure 2 (Gunter *et al.,* 2000) shows that W and R belong to the environment, while M and P belong to the system. The specifications S, instead, characterize the intersection of the environment and the system, that is the elements that are shared by the environment and the system. In the reference model, designations identify classes of phenomena (states, events, individuals, etc.) in the system and the environment and assign names to them. Some phenomena (denoted as e in Figure 3, Gunter *et al.,* 2000) belong to the environment,

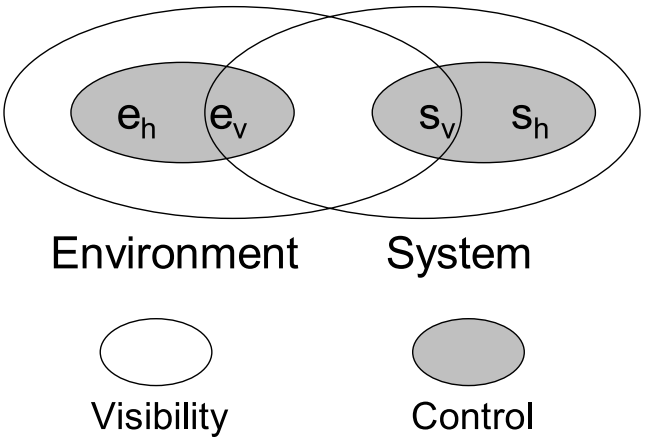

Figure 3: *Environment's and system's phenomena: control and visibility.*

which controls them. Phenomena denoted as s belong to the system. Some of the e phenomena are visible to the system, while others are not: the former are denoted as e_v, while the latter are denoted as e_h ($e = e_h \cup e_v$). The s phenomena are similarly classified as s_v and s_h. Phenomena e_h, e_v, and s_v are visible to the environment and used in W and R. Phenomena s_h, s_v, and e_v are visible to the system and used in P and M. Only e_v and s_v are visible to both the environment and the system: specifications S are defined only in terms of e_v and s_v.

How can we define and describe a problem in order to enable the development of a software-based solution (or just to evaluate if a software-based solution is at all possible)? We have to keep in mind that developers have to be given all the relevant information concerning the desired behaviour of the system: this information constitutes the requirements of the software to be developed. Therefore, requirements are the criteria, conditions, or constraints that a software artefact has to satisfy in order to be acceptable as a solution of the given problem. This conception of requirements implies that:

1. Requirements are usually expressed with reference to the environment phenomena only. Users know well their environment, while they often do not have a clear idea of what computers and software can provide. Therefore they tend to describe the situation they want to achieve, independently of the fact that the result is obtained via a computer-based solution or not.
2. The effects of the system on the environment depend not only on s_v, but also on the reactions that these phenomena generate. It is thus necessary that the rules that determine the behaviour of the environment are described. These descriptions belong typically to W and are expressed in terms of e_h.
3. The goal of the whole software development effort is the construction of a machine having specifications S such that its interaction with the environment causes the requirements to be satisfied.

The latter issue is typically summarized in the formula:

$$W, S \vdash R$$

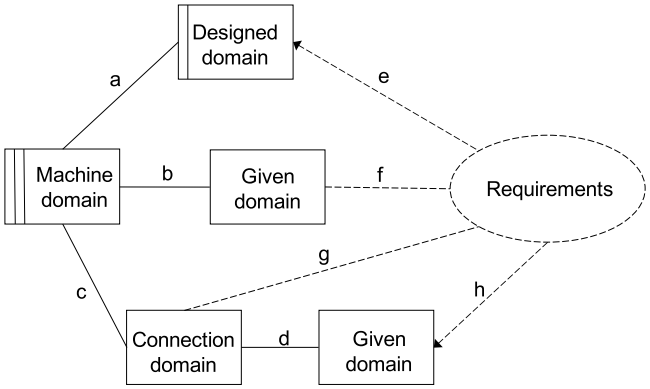

Figure 4: *A problem diagram.*

which states that the satisfaction of the requirements follows from the combination of the world and machine's behaviour.

The task of software analysts is to define S such that, given W and R, $W, S \vdash R$ holds.

The task of designers, programmers, testers, etc., is to build a program P such that, given M and S, $M, P \vdash S$ holds.

If both the analysts and the implementers accomplish such tasks, then both $W, S \vdash R$ and $M, P \vdash S$ hold, thus we get that also

$$W, M, P \vdash R$$

holds, that is the goal of the development is achieved.

Problems are generally described by means of problem diagrams (Jackson, 2001), like the one in Figure 4. Problem diagrams show the domains involved in the problem (i.e. the domains belonging to the environment), the machine that is expected to solve the problem, possible designed domains (i.e. domains whose structure and content can be – to some extent – designed, as opposed to environment domains, that are given and cannot be modified). Domains are shown as boxes in the diagrams. Interfaces between domains are represented as lines connecting the domain boxes. Interfaces indicate that some 'phenomena' (e.g. values, states, signals, etc.) are shared between the connected domains. Sets of shared phenomena are labelled with letters, whose meaning is explained separately: for instance, in Figure 10 c: SP!{store_operation} indicates that the domain whose initials are SP (storage processor) controls a phenomenon 'store operation' that is shared with domain Repository. Problem diagrams also show the requirements in terms of properties that are not given, but have to be achieved by means of a proper machine. Requirements are given in terms of relationships between problem domains.

2.2. *The role of analysts*

As mentioned above, the user tends to talk about what the problem is in the real world, not about the role of the machine in solving it. In fact, the user knows the problem well at a quite high, business-related level. This level of understanding of the business problem provides a starting point for business analysis, which leads to a better understanding of the user's problem and to a more complete and detailed description of the problem, especially concerning the problem environment. Usually the user does not have a clear idea of the role of the machine in solving his/her problem. In fact, the user often

has not the slightest clue of how to solve the problem, 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.

In general, even business analysis does not yield a final indication about the role of software in providing a solution to the given problem. This is due to the fact that business problems allow for several possible solutions, characterized by different costs and effectiveness. Therefore, requirements have to be proactively defined on the basis of the description – provided by business analysis – of the problem and the environment where the machine will have to operate.

RE is thus the creative activity that leads to defining the effects that the hardware/software machine will have on the problem domain so that the problem is solved. To this end, it is necessary to remember that it is the behaviour of the system encompassing 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. Accordingly, the analyst has a double role in describing requirements:

1. He/she must understand the needs of the user and the nature and behaviour of the environment.
2. 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:

1. to describe what are the responsibilities of the system in satisfying the user's needs.
2. To define what are the responsibilities of the environment in supporting and using the hardware/software 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 Section 3, the responsibilities of the system and the interaction between the machine and environment are not always determined univocally by the user needs.

These observations partly contradict what was written in Section 2.1, since R are not just given, but are defined by the user and the analyst together. However, this conclusion leads to a new question, since the user surely has some needs or goals that are not negotiable (e.g. because they are fundamental for achieving a competitive advantage), as opposed to the concrete formulation of requirements, which is actually established with the analyst. In fact, we said that users have 'computer-independent' needs that are typically related to their business and that cannot (or at least should not) be changed by the introduction of computer-based systems. Therefore, we would like to be able to answer questions like the following:

1. What are the relationships between non-negotiable user needs and goals on one side and requirements on the other?

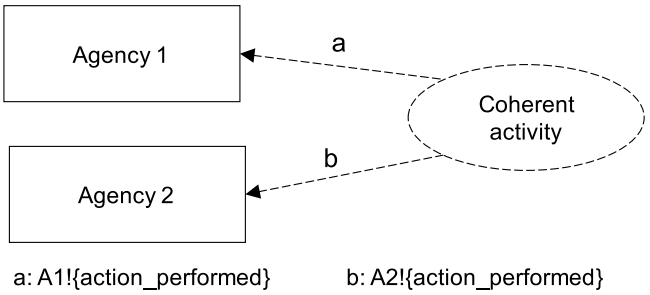

Figure 5: *Problem diagram representing requirements at the business goal level.*

2. Is the knowledge about the world that appears in W, $S \vdash R$ the same that leads to the formulation of non-negotiable user needs and goals?
3. How should we deal with the part of the world whose behaviour is not fixed once and for all, but can be to some extent adapted to the user needs?

3. Case study

In the rest of the paper, the proposed concepts are illustrated by means of an example, described in this section.

Let us start with the high-level needs of the 'user'. In our case study, a company needs that its agencies act in a coordinated way, in order to maximize efficiency, avoid work duplications, avoid inconsistencies, etc. The case study reported here is a simplification of a real-world problem, where a logistic company had to keep track of container movements managed by local agencies in order to optimize freight planning.

We can classify these needs as part of the 'business goals' (*BG*) of the company (throughout the paper the term 'business goal' is used as a shorthand for 'business-originated user goals').

Figure 5 represents the business goal by means of a problem diagram. Since the goal of the company is clearly independent of the existence of a computer-based system, a machineless problem diagram illustrates the goal. The 'requirement' Coherent activity is actually the *BG* statement that agencies have to act in a coordinated way, that is a and b must be coherent: for instance, the action performed at agency 1 – denoted by phenomenon a – must be coherent with the actions – denoted by phenomenon b – performed at agency 2. Of course, the domain must be capable of the expected coherent behaviour among all others possible behaviours; the solution is thus to make sure that the expected behaviour actually occurs. Phenomena a and b denote the information needed to specify the coherency of activities. What does actually mean that a and b must be coherent depends on the detailed specification of the properties of a and b, and on the definition of 'behaviour coherency': these details are not relevant for our purposes.

Now we have to understand *how* the business goal can be actually achieved. To this end, we rewrite the diagram introducing a machine (there must be a machine, otherwise the goals could only be achieved by a 'spontaneous' behaviour of the environment, but if such spontaneous behaviour existed, no needs and goals would arise . . .).

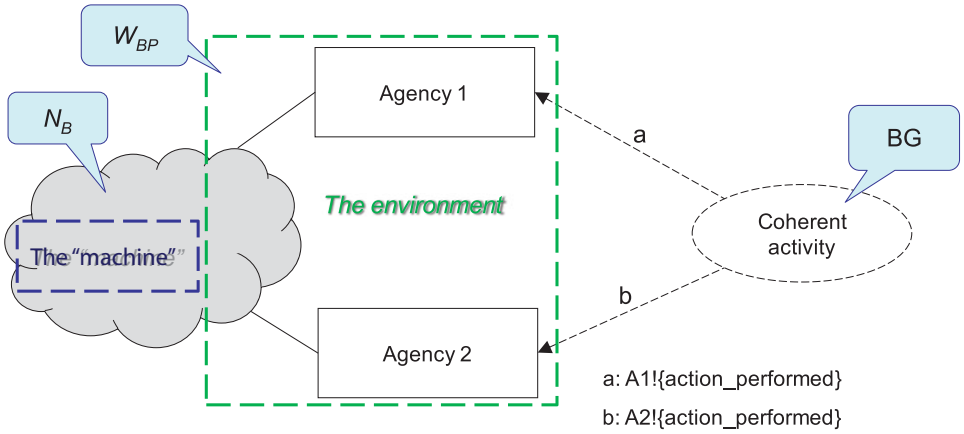

Figure 6: *Problem diagram representing business goal-level requirements.*

Figure 6 describes the same problem as Figure 5, except that we have introduced the part of the system that is responsible for the achievement of the business goals. Since at the moment it is not yet known whether the solution will be based on a computer, the solution is represented in a cloud, which possibly contains some sort of machine (no reference to cloud computing here!). In fact, the stakeholder(s) – typically belonging to top management – who expressed the business goal may have no clue about this part of the system.

In order to make clear that the part of the system in the cloud is actually part of the solution, we can write the usual formula:

$$W_{BP}, N_B \vdash BG \tag{1}$$

where BG is the business goal, W_{BP} describes the business process environment, rules, etc., and N_B specifies the behaviour of the part of the system (possibly including a machine) that will cause the BG to be satisfied. Formula (1) above states that the business goal BG (agencies have to act in a coordinated way) is satisfied, in the environment described by W_{BP}, by some 'machine' described by N_B. Note that at this level, the 'machine' is not necessarily a hardware machine equipped with suitable software. Rather, it can be a sort of mechanism – typically involving people – which 'enacts' the business processes. In our case study, W_{BP} specifies that the agencies store a set of business-relevant information, upon which they take decisions and operate. Moreover, W_{BP} guarantees that agencies act in a coordinated way if they operate according to the same information concerning the business. It is easy to conclude that the business need (N_B) is that information concerning the business has to be shared. In other words: if we perform some actions in the business environment that guarantees that the business information is shared, the coordination goal is achieved. Of course, this brings to a new problem: now we have to decide *how* the information sharing need can be actually achieved. To this end, we redraw the diagram of Figure 6 as reported in Figure 7.

The diagram in Figure 7 still features the cloud, since the nature of the solution is still not known. The requirements N_B state that globally interesting information available at agency 1 (a) is available also at agency 2 (b), and vice versa. For the sake of simplicity, in the rest of the paper we always consider agency 1 as the originator of the relevant information and agency 2 as the place where the relevant information must be reproduced. Of course, dealing with the case when agency 2 originates the relevant information, or when multiple agencies are involved does not pose conceptual difficulties.

The problem described in Figure 7 can be formalized as follows (note that N_B, which was the 'solution' in (1) is the goal in (2)):

$$\begin{aligned} W', R &\vdash N_B \\ W' &= W \cup W_{BP} \end{aligned} \tag{2}$$

where R describes the role of the solution, while W' is an extension of W_{BP}, since it is likely that in dealing with this new problem we need additional information concerning the environment than was provided for the original problem (1).

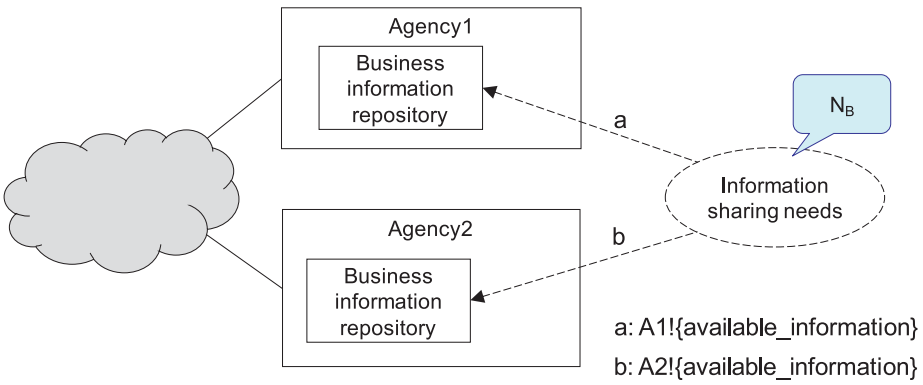

Figure 7: *Problem diagram representing business needs.*

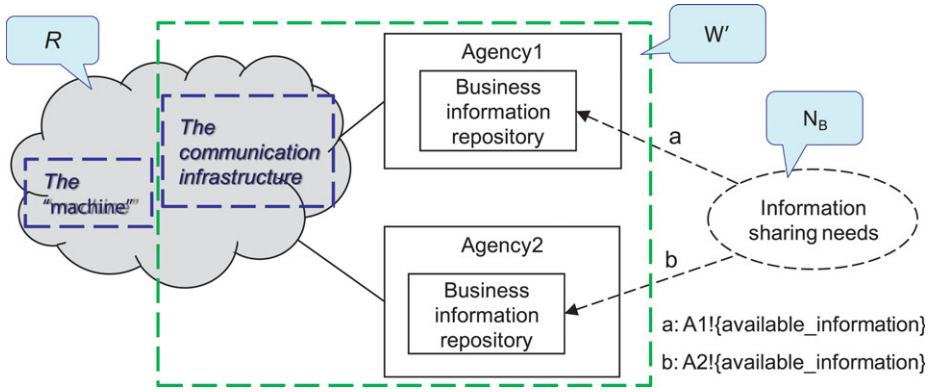

Figure 8: *Problem diagram representing business needs according to (2).*

It is clear that in order to have the globally interesting information available at all agencies, it is necessary to communicate the information. Thus, W specifies that the agencies are connected via a communication infrastructure. Accordingly, a possible specification of R states that the information generated at agency 1 is communicated – via the communication infrastructure – to agency 2, and that the received information is stored locally. The fact that W' includes a communication infrastructure can be taken into account as in Figure 8.

In order to tackle the new problem described in Figure 8 (W', $R \vdash N_B$) we need to define W' and R. To this end, we need to say what is in the cloud, that is what communication infrastructure is available. Of course, several possibilities exist:

1. The agencies are already connected, maybe using relatively old technology (e.g. plain email) and we are constrained to achieve the goal using the available connection. In this case, the communication infrastructure is part of the given environment.
2. The agencies are not connected, or the management decided that a new connection can be built anyway. In this case, the infrastructure is part of the solution.

At this stage of the analysis process, the analyst should:

1. Evaluate the environment, in order to highlight its relevant characteristics. In our case, the communication infrastructure is the relevant characteristic to be identified and described.
2. Sketch possible solutions, some of which will be based on the environment as is, while others will involve changing the environment, possibly introducing new elements. For instance, if the environment does not contain communication infrastructures, the analyst can think of introducing one (maybe sketching different new scenarios, each characterized by a different communication infrastructure).
3. Evaluate which ones of the identified solutions (and related environment) are viable.

In our case study we assume that the analyst has performed the tasks mentioned above, and has found that the agencies are connected via plain email. This situation is described in Figure 9 (from now on, the domains are no longer labelled as R, W, N, etc., to avoid overloading the pictures).

We also suppose that the analysis of the possible solutions has led to the decision that the business goal should be achieved using the available connections. In this case, the communication infrastructure is part of the given environment.

Having chosen the email-based infrastructure, W' specifies the usual functions of the email system (server and clients), while R states that:

1. Any interesting information stored at an agency is sent to the other agencies (W' assures that the communication is carried out properly).
2. Any received information is stored in the repository.

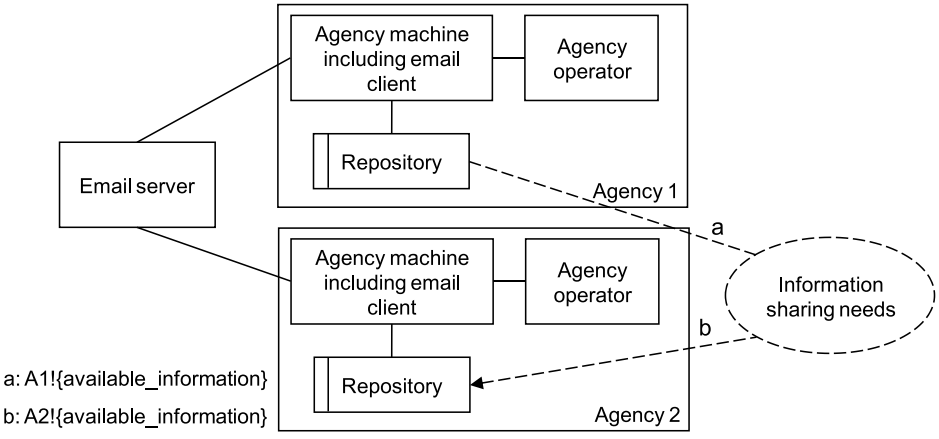

Figure 9: *Problem diagram representing business needs and details of the environment.*

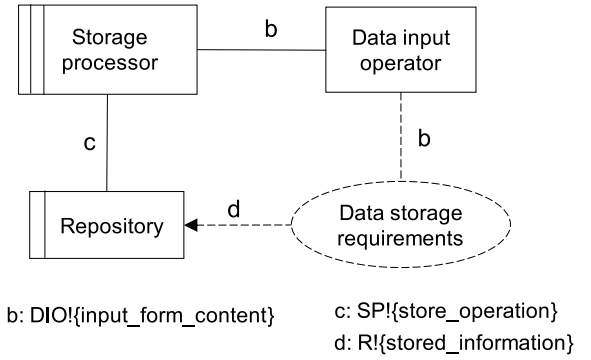

Figure 10: *Problem diagram representing the work done at agency 1.*

These specifications of W' and R assure that the business need N_B involving information sharing is satisfied. Therefore, the problem is now how to satisfy R. As is often the case, there are several ways for satisfying R. In the next sections three possible ways of satisfying R are illustrated. Before proceeding to see how R can be satisfied, it is useful to note that the nature of the problem and the availability of the communication infrastructure suggest that the problem is split into two sub-problems:

1. The sub-problem at agency 1 requires that the locally generated (and locally stored) information is sent to agency 2.
2. The sub-problem at agency 2 requires that the information received from agency 1 is stored locally.

In principle, both sub-problems can be solved in different ways. However, for the sake of simplicity, here we explore only the alternatives for agency 2.

3.1. Email-based case: sub-problem 1 (information generation)

The main work done at agency 1 (as far as we are concerned) consists of generating and storing information. This activity is described by the problem diagram in Figure 10. The data storage requirements simply state that the stored information (d) is coherent with the data provided by the operator (b).

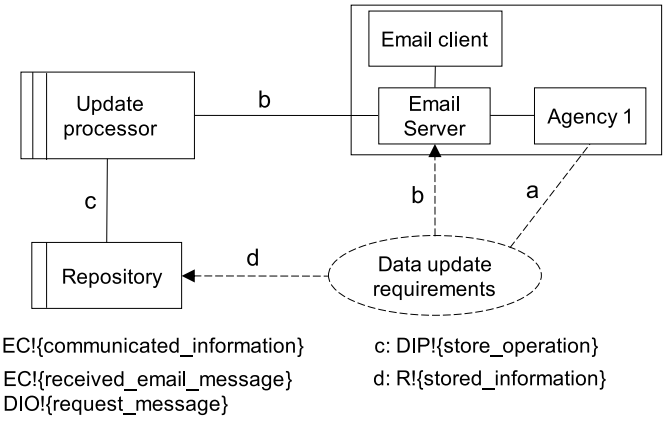

Figure 12: *Problem diagram representing the problem at agency 2.*

The information stored locally has to be transmitted to agency 2. This requirement is modelled in the problem diagram reported in Figure 11.

The data transmission requirements state that whenever a new piece of information (c) appears in the repository, an email message (a) containing the same information is sent to agency 2. As already mentioned, there are several machines (the communication processor in Figure 11) that can satisfy these requirements. For instance:

1. The operator sends manually an email whenever he/she inputs new data in the repository;
2. the data input processor, upon receiving data from the operator, both stores it in the repository and sends an email to agency 2 containing the same information;
3. a specific program periodically checks the repository, detects new information and emails such information to agency 2.

We do not explore in detail the solutions for agency 1. Instead, we concentrate on the problem at agency 2, described in Figure 12. This problem diagram finally introduces the machine (the update processor) that is responsible for satisfying the data transmission requirements. The satisfaction of these requirements, together with the satisfaction of the requirements for the problem at agency 1, guarantee that needs N_B in equation (2) are satisfied.

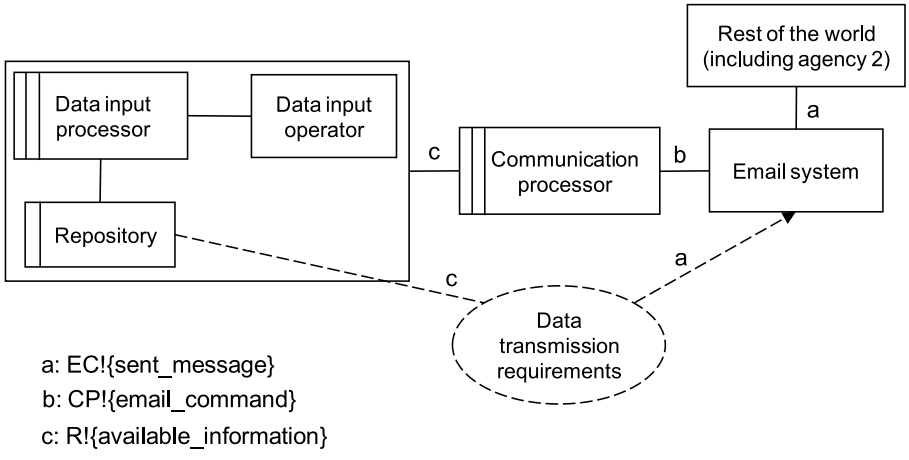

Figure 11: *Problem diagram representing the communication requirements for agency 1.*

Here we have to take into account that the environment is not perfect. The description of the environment (W) states that:

1. Email messages (b) contain the communicated information (a).
2. However, it is possible that – because of some error – the message (b) does not contain comprehensible information.

The requirements data update requirements (R) state that:

1. The communicated information (a) is stored in the repository; that is d = a.
2. Exception: if the content of the received email message (EC!{received_email_message}) is unclear, a clarification request (DIO!{request_message}) is sent to the originating agency.

The different possible ways of specifying the update processor machine are described in Sections 3.2–3.4. In fact, for the problem in Figure 12 we have to assure that $W, S \vdash R$ is true. There are several possible machines whose specifications make $W, S \vdash R$ true. In the following sections we shall see three machine specifications that cause R to be satisfied.

3.2. Email-based case: scenario 1

In this scenario, the update processor is made of the same data input processor available at agency 1 (i.e. the machine that is used to store useful information in the local repository) and a data input operator. The latter actually behaves as the 'program' of the update processor, that is he/she determines the behaviour of the 'machine' that guarantees the satisfaction of the data input requirements.

In practice, this means that no specific software has to be written; instead, a manual procedure has to be established.

R: the requirements data input requirements state that:

1. The communicated information (a) is stored in the repository, i.e. f = a.
2. Exception: if the content of the received email message (EC!{received_email_message}) is unclear, a clarification request (DIO!{request_message}) is sent to the originating agency.

W: the environment description states that:

1. Email message (EC!{received_email_message}) contains the communicated information (a). However, it is possible that – because of some error – the message does not contain comprehensible information.
2. The data input processor and Repository guarantee that the stored information (f) is coherent with the input information (d).

S: specifications

1. Email messages (EC!{received_email_message}) are read by the operator, who feeds manually the information (c) to the system via suitable forms (d).
2. Exception: if the content of the received email message is unclear, the operator sends a clarification request (DIO!{request_message}) to the originating agency.

It is easy to see that a 'machine' implementing specifications S satisfies requirements R. The data input operator is essential: he/she is part of the 'machine'.

This solution is clearly suitable if the information to be shared is generated rather infrequently. In this case, the work of the operator avoids the need of building an ad-hoc computer-based system.

However, if the frequency of incoming email messages becomes sufficiently high, the operator could have to perform a large amount of work, thus making this solution excessively expensive. Moreover, being the solution completely demanded to the operator, it is error prone. These considerations suggest that other solutions are explored.

3.3. Email-based case: scenario 2

In this scenario we consider the automation of the work that in scenario 1 was done manually by the operator. Accordingly, in Figure 14, the machine (still named data input processor) is connected directly to the email server, from which it retrieves the incoming email messages. In case the content of the message is not clear, the data input processor issues a warning on the console of the data input operator, who can either provide the correct interpretation of the received data to the processor, or send a clarification request to agency 1.

The specification S of the update processor is as follows (note that the 'machine' involves also a manual procedure, as in scenario 1, although in this case it is just for solving unclear messages):

1. Email messages (b2) are read by the data input processor, which automatically stores the communicated information (a) to the repository.
2. Diagnostics (d) are issued when the informative content of email messages is unclear.
3. The operator reads diagnostics (d) and, if able to interpret the message, provides to the data input processor the information (d) to be stored, otherwise it sends an email message (b1) requiring a correct input.

The machine specified in this scenario is clearly more suitable for dealing with big volumes of email messages. However, the solution is effective if only a few messages cannot be understood correctly by the machine; otherwise, most of the work would still have to be done manually.

3.4. Email-based case: scenario 3

In this scenario we consider the full automation of the procedure; that is also the handling of unclear messages is performed automatically.

Accordingly, in Figure 15 the machine update processor is connected directly to the email server, from which it retrieves the incoming email messages. In case the content of the message is not clear, the update processor sends a clarification request to agency 1 via email.

The specification S of the update processor is as follows (note that the 'machine' no longer involves a manual procedure):

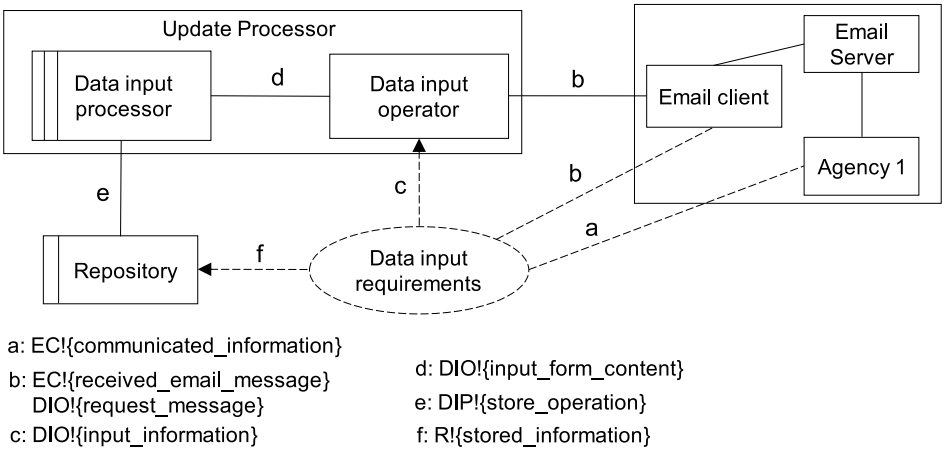

Figure 13: *Problem diagram for agency 2: scenario 1.*

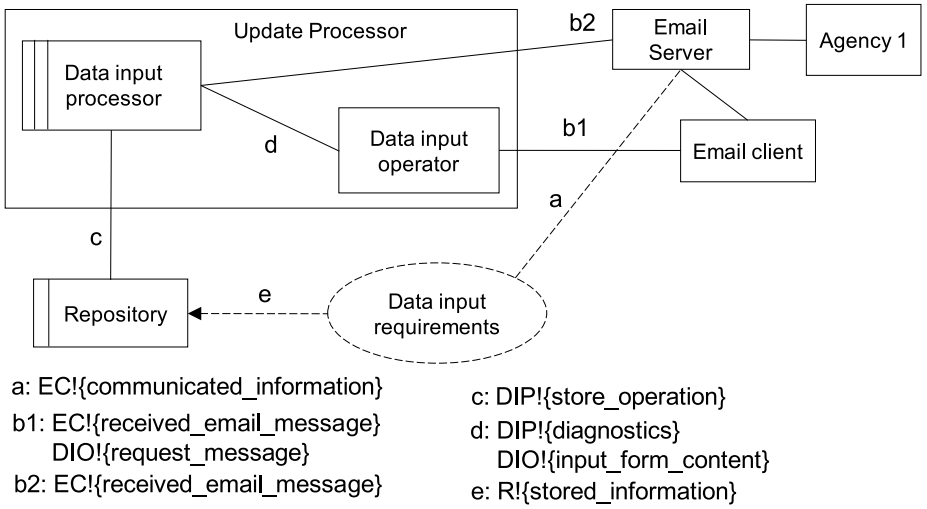

Figure 14: *Problem diagram for agency 2: scenario 2.*

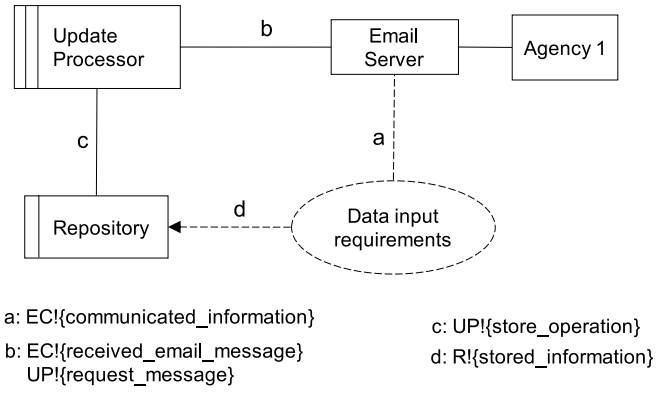

Figure 15: *Problem diagram for agency 2: scenario 3.*

1. Email messages (EC!{received_email_message}) are read by the processor, which automatically stores the communicated information (a) into the repository.
2. If unable to interpret the message the processor sends an email message (UP!{request_message}) requiring a correct input.

The machine specified in this scenario is clearly suitable for dealing with big volumes of email messages, both correct and incorrect.

4. The relationship between needs and requirements

As already mentioned, according to the reference model (Gunter *et al.,* 2000) 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. This description allows for several machine specifications that satisfy 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 fact, the very satisfaction of requirements depends upon the machine).

4.1. Needs versus requirements

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

$$W, R \vdash N$$

$$W, S \vdash R$$

where N indicates the user needs. This new description stresses that the user requirements concerning a (usually) computer-based solution are functional to the satisfaction of the user needs. Highlighting the user needs is also 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 specifications that satisfy 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 functional to achieving some goal in the context of a business process. For instance, in our case study the requirement of having the email messages stored in the email repository was determined by the need of sharing information, which in its turn was determined by the goal of letting the agencies act in a coordinated way. In conclusion, we may say that

$$W, N \vdash BG$$

where W is the knowledge of the business process and BG 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 making agencies act in a coordinated manner could be achieved in different ways: for instance, a coordination board could collect action proposals from agencies and control that no conflicts exist.

Similarly, information sharing could be translated into different requirements, according to the available communication infrastructure.

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:

1. At the topmost level, the formulation of the goal depends only on the knowledge of the business process, and needs are functional to the business.
2. At the intermediate level, the mechanisms supporting the process are known: an email system is used; therefore, we need to save emails in repositories at the agencies.
3. 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.

Quite interestingly, while the knowledge of the world that is needed increases in extent and detail while we proceed from goals to requirements, it is also true that at the lower levels several details of the business knowledge can be safely ignored: for instance, when dealing with email message storing we could ignore that this is due for work coordination.

The hierarchical organization described above could be beneficial to properly define monitoring and measurement activities. For instance, at the topmost level one is interested in knowing how long it takes to a piece of information to reach an 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: for example if the situation described in Section 3.3 (scenario 2) holds, interesting measures could be the percentage of messages considered incorrect by the machine and the percentage corrected locally by the operator.

4.2. Requirements as transformations of needs and goals

The proposed approach can be regarded as an application of software problem transformations (Hall *et al.,* 2008; Hall & Rapanotti, 2011). In these transformations, problem P2 is related to problem P1 in that, in it, there is a detailed description of the solution for P1. This relationship is also justified by an argument of the formal correctness of the solution, so that we can say that a solution of P2 is actually a solution of P1. A software problem transformation transforms a single problem – the conclusion – to a set of problems – the premises – and records an argument – the adequacy argument step – that justifies the relationship of the premises to the conclusion.

So, for instance, we could write equation (3) to formalize that problem $W', R \vdash N_B$ (the information sharing problem) is solved if problems $W_{a1}, S_{a1} \vdash R_{a1}$ (information communication at agency 1) and $W_{a2}, S_{a2} \vdash R_{a2}$ (information updating at agency 2) are solved. The adequacy argument J (not detailed here) exploits the properties of the environment, namely of the agencies' environments and the email connection that guarantees that messages sent from agency 1 are received at agency 2.

$$\frac{W_{a1}, S_{a1} \vdash R_{a1} W_{a2}, S_{a2} \vdash R_{a2}}{W', R \vdash N_B} \ll J(W' \cup W_{a1} \cup W_{a2}) \gg \tag{3}$$

Another example of the use of transformations in our approach is given by the application of domain interpretation, 'the problem transformation schemata by which a non-solution domain description is interpreted, that is re-expressed in some way to make it more suitable for problem solving' (Hall *et al.,* 2008; Hall & Rapanotti, 2011): in our case study, the introduction of the description of the email system connecting agencies into the business domain description can be seen as a domain interpretation transformation.

Another construct proposed in Hall *et al.* (2008) and Hall and Rapanotti (2011) that can be effectively used in our approach is context expansion (the combination of a number of problem domains in a problem's context).

In general, it seems possible to say that the construction of a hierarchy of requirements can be built and manipulated according to the principles of Problem Oriented Software Engineering illustrated in Hall and Rapanotti (2011) and Hall *et al.* (2008).

4.3. Generalization of the approach

Although in the case study we had requirements on three levels (the business goals, the user needs, and the requirements) we can easily generalize the proposed approach to deal with requirements hierarchies encompassing any number of levels. In such case, the description is given by a set of formulae like the following:

$$\begin{gathered} W_0, R_1 \vdash R_0 \\ \dots \\ W_i, R_{i+1} \vdash R_i \\ \dots \\ W_n, S \vdash R_n \end{gathered} \tag{4}$$

In practice, we start with a high-level requirement R_0 in an environment described by W_0 and – through a sequence of refinement steps (or transformations, if you like better the terminology of POSE (Hall *et al.,* 2008; Hall & Rapanotti, 2011) – we reach the usual formulation of a problem (W_n, $S \vdash R_n$) as in the reference model (Gunter *et al.,* 2000).

The number of refinement steps needed to reach an implementable machine specification depends on several factors, including how abstract was the specification of R_0 and how complex is the environment.

A constraint that has to be satisfied for the validity of (4) is that W_{i+1} must be compatible with W_i, that is all the statements that were true in W_i are still true in W_{i+1}. However, it is possible that W_{i+1} provides a more extended and/or detailed description of the environment than W_i, thus there can be statements that are true in W_{i+1} and were simply not present in W_i.

Note that the descriptions in (4) are expected to be consistent with the properties expressed in the reference model (Gunter *et al.,* 2000). So, for instance, relative consistency should be assured, that is 'any choice of values for the environment W_i variables visible to the system should be consistent with R_{i+1} if it is consistent with assumptions about the environment' (Gunter *et al.,* 2000).

The requirements hierarchy described by (4) can also be useful to classify requirements analysis activities. For instance, system-level analysis and component-level analysis are easily distinguished as it is possible to note that W_i contains a description of the system with little or no details about the system's components, while W_{i+1} contains more details about the components. As an example, in the description of an intensive care unit (such as the one widely discussed in Jackson (2001) at a given level the specification of the problem might not mention sensors at all, while the refined description could introduce them.

The relationship between W_i and W_{i+1} can also deal with the unspecified parts that may appear in problem descriptions (such parts are represented as grey clouds in Figures 6–8). In general, the lack of details is justified by the fact that the description is at a high level in the hierarchy. Sometimes, however, it may be that we have some degrees of freedom in shaping (and then describing) the environment: hence we can introduce in W_{i+1} elements not present in W_i. Consider again the case of the intensive care unit: if the original description of the environment does not specify the presence of any alarm device, we can introduce a device of our choice in the next level description. These considerations provide an answer to the last question reported in chapter 2: when it is clear that a description of the environment does not support the requirements (e.g. it is not feasible that the patients themselves introduce their data into the intensive care unit system) then you can explore the possibility of extending or modifying the environment, for example by introducing sensors that connect patients with the machine.

As far as terminology is concerned, the hierarchy of refinements described by (4) suggests some possible definitions:

1. Business goals are the highest R_i: for instance $\forall\ i,\ 0 \leq i \leq k \rightarrow R_i$ is a *BG*. However, the hierarchy could start with a R_0 that is not a *BG*, but some lower level requirement; therefore, we need more stringent conditions for defining a *BG*: a possibility is to say that R_i is a *BG* if the corresponding environment description W_i does not include a computer machine as a necessary element.
2. Similarly, $\forall\ i,\ k < i \leq m \rightarrow R_i$ is a need. Needs are characterized by the fact that they do not include any reference to the final goal that led to their definition. In other words, given a need, it is usually not evident *why* that need arose. The description of the environment can involve a computer system, but the latter is usually described in an abstract way, that leaves several degrees of freedom to the analyst for outlining a solution.
3. Finally, $\forall\ i,\ m < i \leq n \rightarrow R_i$ is a requirement. This is the case most often described in literature: requirements are about the responsibility of the machine, whose detailed description is explicitly given (of course, only in terms of shared phenomena, the definition of the machine internals being left to designers).

According to this terminology the higher steps dealing with *BG* can be classified as 'business analysis', while lower steps are 'requirements analysis'.

Finally, it is necessary to stress that (4) actually describes a hierarchy, rather than a sequence. In fact, at each step there are generally multiple definitions of R_{i+1} that satisfy R_i.

4.4. Validation and limitations

This paper is meant to clarify the nature of concepts (such as user requirement, user needs, and business goals) and activities (business analysis, requirements analysis) of the 'higher phases' of the software development process. Another objective of the paper is to put the concepts and activities mentioned above in the context of well-established approaches (the reference model for requirements and specifications (Gunter *et al.,* 2000), problem frames (Jackson, 2001), POSE (Hall *et al.,* 2008; Hall & Rapanotti, 2011)).

These clarifications can be used in several ways. At the two extremes of the range of options are the following possibilities:

1. Analysts can use the suggested approach implicitly, as a discipline to keep order in their ideas while they proceed through the different phases of the analysis.
2. Analysts shape the analysis activities according to a process that explicitly takes into account the proposed concepts. So, for instance, artefacts classified as business goals, user needs and requirements are produced, and refinement/transformation steps are also planned and performed.

In both cases, Section 5 provides guidelines about the organization of the analysis process.

Of the two possible ways of using the proposed approach, only the former was actually used by the author. Up to now, the argumentation of the validity of the proposed approach rests on the fact that it is coherent with the techniques involved (the reference model for requirements and specifications (Gunter *et al.,* 2000), problem frames (Jackson, 2001), POSE (Hall *et al.,* 2008; Hall & Rapanotti, 2011)), and the previous research on goal-based RE.

Concerning the possible limitations of the approach, it is expected that it is applicable as long as the underlying techniques – mainly problem frames – are applicable. A possible hindrance to the full-fledged application could be that the need to produce several classes of documents (business goals, user needs, and requirements) may be excessively expensive for small developments. On the contrary, in bigger development efforts, the possibility of exploring and assessing the cost of alternatives (see Section 5.3) could be a decisive advantage.

5. An analysis process driven by user 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:

1. Devising a machine and a domain behaviour such that needs are satisfied, or
2. devising user requirements that satisfy the needs, and then specifying a machine that satisfies the requirements (and, hence, the needs).

Option (1) 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. In fact, in such cases the 'distance' between needs and machine is so large that the figuring out the features of the machine that satisfy the requirements involves building a quite complex argument. Moreover, the degrees of freedom in choosing the machine features are many, and call for criteria to make choices that are often not at the 'machine level'.

Option (2) 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 $W, S \vdash R$.

In this way, one does not have to 'jump' from needs directly to the machine specifications: the distance between needs and machine is covered in two shorter steps: from needs to requirements and from requirements to machine specifications. However, it must be noted that during the first step one should take into account the cost of implementing the requirements, that is one should estimate the effectiveness and cost of the machine(s) that can satisfy 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 that – together with W – satisfy N should be explored; in this phase we consider the possibility of modifying the environment as needed to define more easily achievable requirements. The possibility of modifying the environment is justified by the fact that the higher is the level of needs, the more suitable are hybrid solutions, composed of both hardware/software machines and people. Of course, the costs involved in modifying the environment (e.g. for hiring or training people) have to be taken into account when evaluating the most convenient definition of R.

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.

5.1. A process model

The considerations reported in Sections 3 and 4 may provide something of interest to analysts. However, it may not be obvious how effectively to use in practice the concepts embedded in the proposed hierarchical requirements model. In order to enable analysts to take advantage of such model, we propose a specific problem analysis process. The main merit of the proposed process model is that it highlights that analysis has to proceed through a sequence of refinements, and that feasibility and cost evaluation activities are necessary, since some solutions could be hardly feasible or excessively expensive.

Figure 16 shows a UML activity diagram representing the process of dealing with a business-level problem. The first activity (*Define problem*) involves describing the environment and the business goal, thus resulting in providing the definitions of W and BG. The rest of the process is devoted – as expected – to defining a 'machine' N such that $W, N \vdash BG$. This is clearly the most difficult part of the process, which requires both creativity and a systematic and rigorous approach. Activity *Specify N* involves defining a specification N that not only lets achieving the business goal BG, but allows for feasible and reasonably expensive solutions. Activity *Specify N* can fail altogether, thus leading to a failure of the whole analysis. If a specification is produced, the following activity (*Feasibility and cost evaluation*) involves estimating the cost of the specification N. Three types of results are envisaged:

1. The specification is feasible and the estimated cost for implementing it is acceptable;
2. the specification is unfeasible or too expensive;
3. the specification is too complex to get reliable evaluations about its feasibility or the implementation cost.

In the first case, we can either consider the analysis complete, or we can explore different definitions of N, searching for cheaper specifications.

In the second case, we go back to activity *Specify N*, in order to look for different – more feasible – definitions of N.

In the last case, we need to tackle the complexity of the specifications by refining the business goal. This is done by

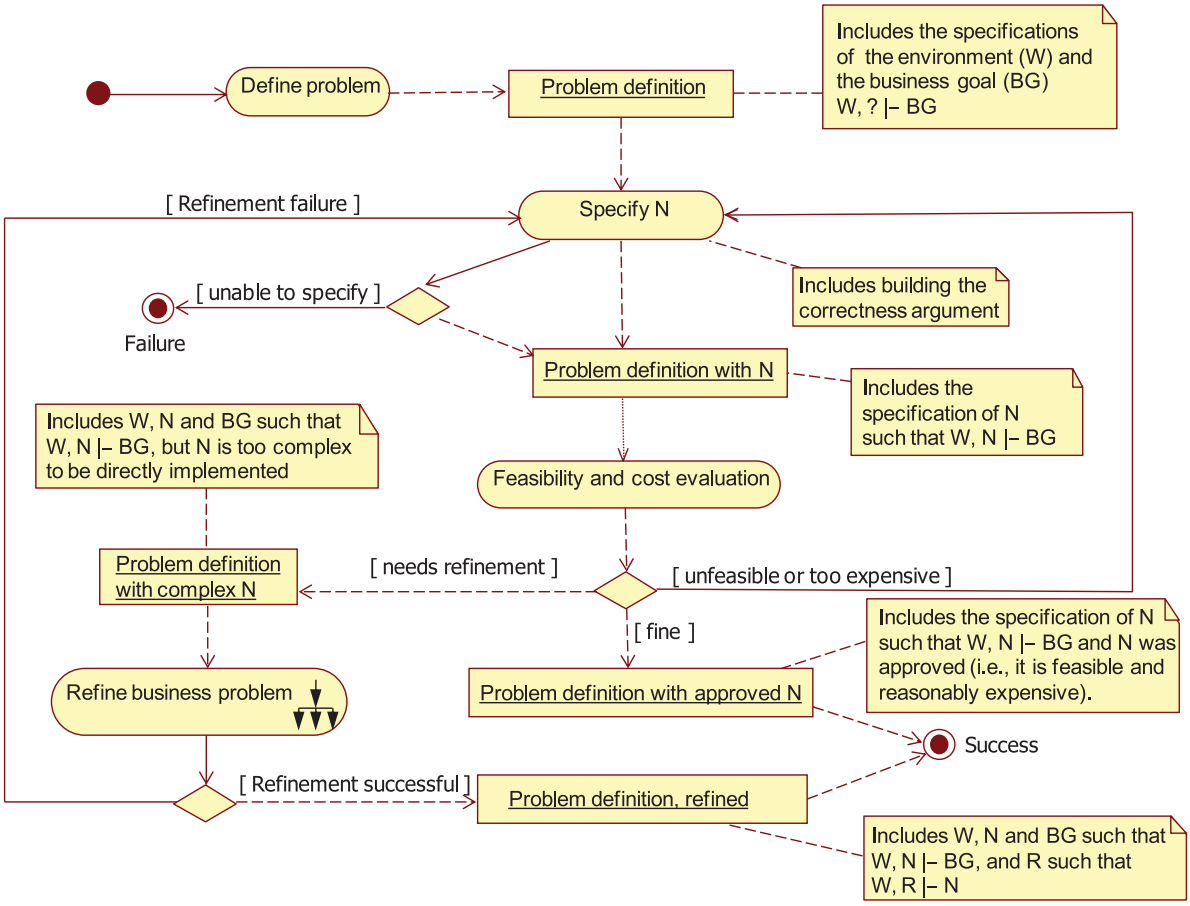

Figure 16: *The process of defining how to achieve a business goal.*

means of the activity *Refine business problem*, which – if successful – produces a specification R that is feasible and reasonably expensive. Such R makes $R, W \vdash N$ true, so that also $W, N \vdash BG$ becomes true. Therefore, the cost of achieving BG is the cost of implementing R. If the activity *Refine business problem* fails, either we are able to devise an alternative definition of N, or we must give up, and declare BG not achievable, at least at reasonable costs.

By applying the process in Figure 16 to our case study, we would get as a problem definition the specification described in Figure 6. Activity *Specify N* would bring to the definition of N_B as described in Figure 7 (i.e. N_B specifies the need for information sharing as a solution of the coordination problem). The cost evaluation of N_B would reveal the need for refining the problem; in fact – as discussed in Section 3 – information sharing can be achieved in different ways and at different costs, depending on the available communication infrastructure and how it is used.

Activity *Specify N* includes the construction of a 'correctness argument'. This means that the resulting definition of N is not just stated, but it is shown (convincingly, if not formally) to be true (Jackson, 2001).

Activity *Refine business problem* – whose goal is to define R such that $R, W \vdash N$ and R is feasible and reasonably expensive – is described in Figure 17. It is easy to see that the activity is organized very like the process of dealing with business-level problems. A noticeable difference is that before proceeding to devise any definition of R, the given description of the environment (W) is properly extended, in order to include details that are not relevant at the business level, but become essential when dealing with the processes that support the business. It could also be possible to consider the extensions of the environment description as strictly connected with the specification of R, thus activity *Refine problem definition* could be made part of the loop that involves defining alternative specifications of R and evaluating them.

In our case study, *Refine problem definition* would bring to the extended representation of the environment as in Figure 9. Finally, *Specify R* produces the specifications of R for the two sub-problems (information creation and transmission at agency 1 and information receipt and storage at agency 2). More precisely, the first execution of *Specify R* would produce the specifications reported in Figures 11 and 13. If this set of requirements is considered feasible and reasonably expensive the process proceeds to the identification of a machine that satisfies the requirements. Otherwise, a new iteration of *Specify R* is performed, which produces the specifications reported in Figure 14. This new set of requirements is evaluated with respect to feasibility and cost, and so on, until either a satisfactory set of requirements is found, or the process fails, having explored all the possible requirements that lead to satisfying N_B.

Of course, we may recognize the need for refining the specifications of R (as is often the case, as discussed in Section 3) in order to test with several specification S of the actual machine. In such cases, activity *Refine needs* is executed. Its definition (not reported here) could be very similar to that of activity *Refine business problem*.

As a final note, even though three levels of requirements were often mentioned (i.e. goals, needs, and requirements),

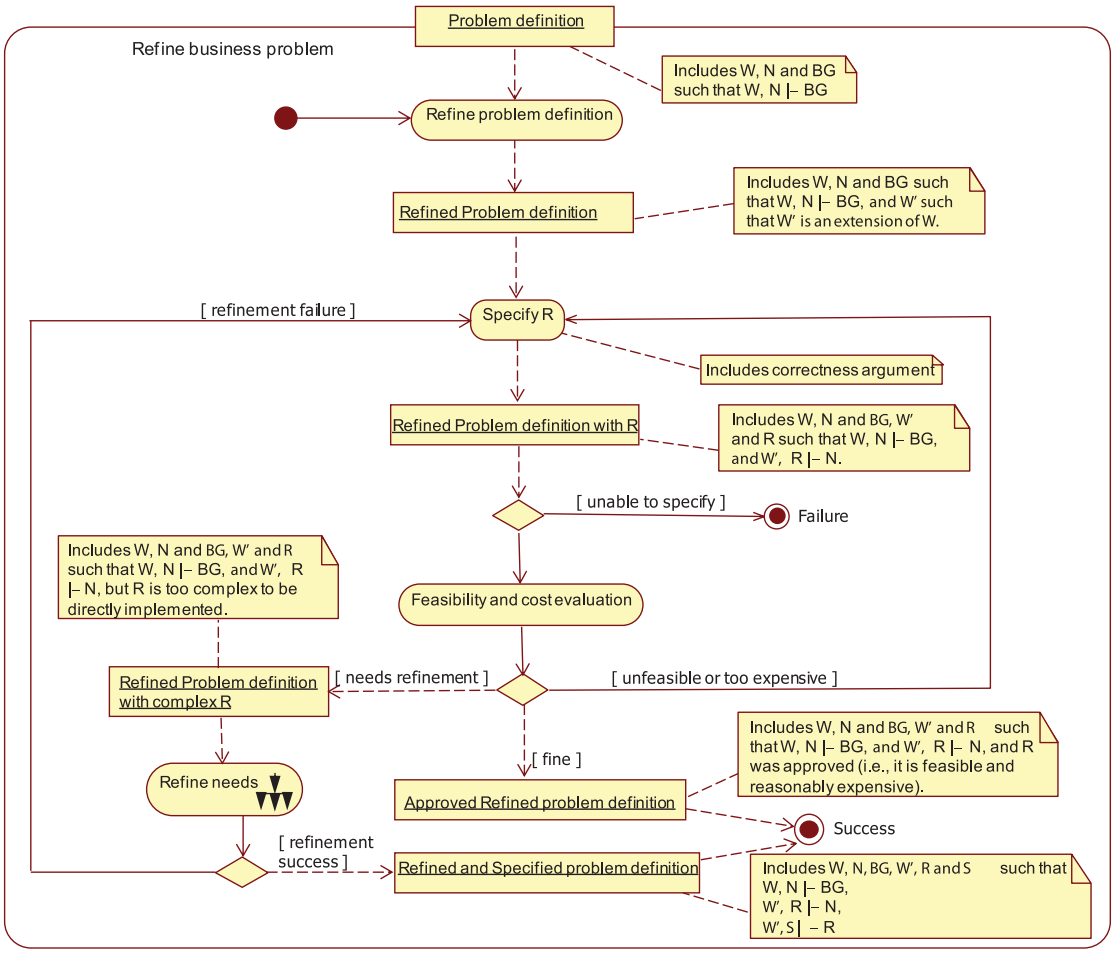

Figure 17: *The process of refining a business problem.*

the requirements hierarchy can be further expanded towards higher and lower levels, for example considering that several machines can implement the same specification, etc.

5.2. *Dealing with failures*

The process described in the section above acknowledges the possibility that refinements fail. When a failure occurs, it is often possible to 'backtrack' and look for an alternative refinement. However, when no refinement appears possible you have to modify the problem, for example by negotiating the needs to be satisfied. In this section, the possibilities of failures and how to deal with them are discussed.

The most critical case is probably when different business goals are somehow contradictory. In the formula $W, N \vdash BG$, we have a set of new goals BG and an environment W that already satisfies the 'older' business goals. In fact, W is generally suited to support the current business: typically, W describes an organization (and its environment) as it is at the moment, and BG represents the improvements in the business the organization wants to achieve. Now, it is possible that BG contain some goals that *may* lead to contradictory requirements: in this case by considering the business goals together, either we find a set of requirements that satisfies all the goals, or we come to the conclusion that some goal has to be changed or dropped.

It is also possible that some goal in BG is in contrast with W, that is it cannot be achieved as long as W is maintained as it is. This is the case when a new goal is in contrast with some decision that has been previously taken to satisfy older goals. In this case we are confronted with the choice between a conservative behaviour – that is keep W and drop part of BG – or a more innovative behaviour: critically analyse W to understand if the older goals are still valid and if they can be achieved differently; in such case drop the features or W that contrast with the achievement of BG, and rebuild it so that it can support both the new and the older goals.

Schematically, the procedure to deal with this issue is as follows:

The initial situation is $W_{old}, N_{old} \vdash BG_{old}$.

The new problem is $W_{old}, N_{old}, N_{toBeDefined} \vdash BG_{new}$.

If $N_{toBeDefined}$ cannot be defined:

> If BG_{old} is no longer important, then restate the problem as $W_{old}, N_{toBeDefined} \vdash BG_{new}$
>
> Else, consider restating the problem as $W_{old}, N_{toBeDefined} \vdash BG_{new}, BG_{old}$ (this involves dropping the implementation of N_{old}, which may be quite expensive).

5.3. *Estimating the cost of implementing specifications*

In the discussions reported above it has been stressed that being able to evaluate specifications is very important. The cost of achieving a business goal is made of two components: the cost of building (or buying) a machine that implements the specifications, and the cost of running it. In order to

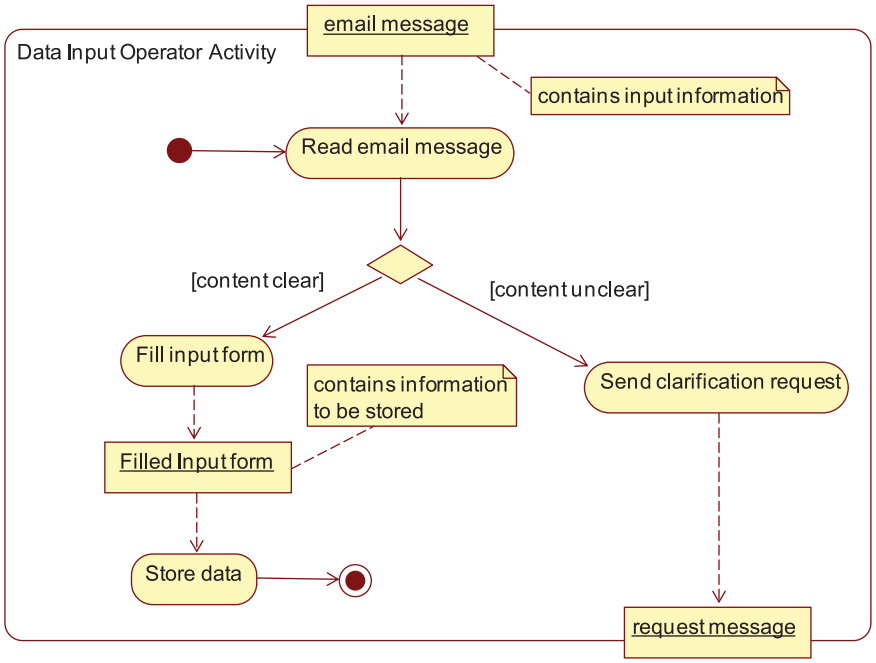

Figure 18: *Activity diagram specifying the behaviour of the data input operator in scenario 1.*

evaluate such costs it is necessary to remember that the machine is usually made of both a computer-based application and people who execute procedures that are part of the specifications (as in scenario 1 in Section 3.2, where an operator reads emails and fills database forms with the received information).

Therefore, in order to fully evaluate the cost of a specification, it is necessary to consider and estimate the following factors:

1. The cost of implementing the hardware/software machine;
2. The cost of setting up a human-based "machine" (i.e. defining "manual" procedures, hiring people that are able to carry out such procedures, training people, etc.);
3. The cost of execution of the software applications;
4. The cost of running the manual procedures, which includes the cost of the involved people and of the devices and resources that people have to use in order to carry out the specified procedures.

Costs (1) and (2) can be estimated by means of widely available techniques. Concerning cost (1) we should be able to apply the techniques that have been proposed for the cost estimation of software development. These techniques usually require that the size of the software to be developed is measured (or estimated). In particular, modern techniques and tools (like for instance COCOMO II (Boehm *et al.*, 2009) and SEER-SEM (Galorath & Evans, 2006)) accept that the size is given in function points (FP) (Albrecht, 1979) or – sometimes – in COSMIC FP (CFP) (COSMIC, 2007). Both these functional size measures are applied to user requirements and define the size measure as a function of a set of 'base functional components': it is thus necessary to identify base functional components by examining the requirements specifications. Such issue has been addressed in Lavazza and del Bianco (2008) and del Bianco and Lavazza (2009), where mappings between problem frame modelling elements (like shared phenomena, for instance) and both FP and CFP are defined: the methods proposed in Lavazza and del Bianco (2008) and del Bianco and Lavazza (2009) allow for measuring the functional size of machine specifications, and then to use consolidated techniques and tools to get estimates of the cost of building the specified machines.

As far as costs (4) are concerned, it is possible – as in the case of costs (1) – to exploit the information contained in problem frames. As an example, let us consider the specification of the data input operator, which is an essential part of the update processor in scenario 1. These specifications can be represented by means of a UML activity diagram, as in Figure 18.

By examining the diagram, it is easy to express the cost of executing the specifications as follows:

$$C_{DIO} = KN_{RM}(T_{Rem} + P_{um}T_{Scr} + (1 - P_{um})T_{Fif}) \qquad (5)$$

Equation (5) expresses C_{DIO} the operating cost of the data input operator with respect to:

N_{RM}: the number of received email messages;

T_{Rem}: the average time taken to read and interpret a message;

T_{Scr}: the average time taken to write and send a clarification request message;

T_{Fif}: the average time taken to fill the input form with the information from the email message (the time taken to store the data is assumed to be practically nil);

P_{um}: the probability that a received email message cannot be reliably interpreted;

K: a constant that transforms the time employed into a cost.

6. Related work

The RE community has produced a large amount of work on the role of user goals in RE.

Ross and Schoman stated that requirements must specify 'why a system is needed, based on current or foreseen conditions, which may be internal operations or an external market', 'what system features will serve and satisfy this context', and 'how the system is to be constructed' (Ross & Schoman, 1977). Goals were identified as the originators of requirements (Lee, 1991). Conversely, requirements emerge because of some underlying goal (Ross, Schoman, 1977; Dardenne *et al.,* 1991; Sommerville & Sawyer, 1997).

Mylopulos suggested the transition from more traditional forms of analysis to goal-oriented analysis, which involves the description and evaluation of alternatives (Mylopoulos *et al.,* 1999). He also pointed out that goals and the way they are tackled are related to 'the organisztional objectives behind a software development project' (which correspond quite closely to the 'business goals' mentioned above).

Goal refinement leads from high-level goals to low-level technical requirements, according to business-related considerations (Yu, 1993).

Goal refinement involves choosing among different alternatives; actually it is fundamental for detecting the existence of alternatives (Lamsweerde, 2000).

The identification of requirements from goals has been extensively studied (Dardenne *et al.,* 1991; Rubin & Goldberg, 1992; Anton & Potts, 1998; Dubois *et al.,* 1998; Kaindl, 2000; Lamsweerde, 2000).

A fundamental step in goal-based RE was the definition of goals as properties to be achieved, described via optative statements as opposed to indicative ones (Jackson, 1995; Zave & Jackson, 1997). A quite comprehensive review of the research carried out in goal-oriented RE can be found in Lamsweerde (2001).

Several authors have published requirements acquisition strategies that start from an analysis of goals (e.g. Dardenne *et al.,* 1993; Yu & Mylopoulos, 1994; Moffett & McDermid, 1995). In 1995, Potts proposes a schema similar to story schemata to help analysts develop a limited set of representative scenarios. In 1996, Darimont and van Lamsweerde presented an approach to goal refinement that provides formal constructive support while hiding the underlying mathematics.

Kujala *et al.* propose an approach to representing user needs and translating them into user requirements in industrial product development cases (Kujala *et al.,* 2001). 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.* (2008) propose a characterization of the concept of 'value' of use cases based on the catalogue of frames for real software problems proposed by Jackson (2001). 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 RE, but does not help in the identification and management of user needs, or even higher level goals.

Some work aimed at transforming requirements into specifications in the context of problem frames appears in the literature (Rapanotti *et al.,* 2006; Seater & Jackson, 2006; Li, 2007; Seater *et al.,* 2007). With respect to the work reported here, all the mentioned papers focus on the lower section of the hierarchy described by (4); that is they tend to disregard the connection of requirements with higher level business goals. On the contrary, they focus on the process of deriving specifications from requirements.

The idea of progressing from the problem to the solution via transformations is at the base of POSE: see, for instance, Hall *et al.* (2008) and Hall and Rapanotti (2011). 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. In other words, while POSE aims – using very similar principles – at finding solutions to given problems, our approach is more concerned on defining problems themselves, given the high-level user goals.

In 2009, Hall and Rapanotti propose assurance-driven design in the context of POSE. They also propose a process model that addresses the issues of assurance and trade-offs. Such process stresses the importance of problem and solution validations. An important aspect of development that is made explicit in Hall and Rapanotti (2009) is that of backtracking: design steps that come to a dead end because they cannot be validated cause to backtrack to a point where a different approach can be taken. This also applies in the process proposed here: for instance, in Figure 17 when a refinement fails we go back to specifying requirements in a different manner (but always in a way that satisfies W', $R \vdash N$).

In Section 1, it was stated that it would be useful to clarify what are the differences between a user requirement and a user need, and what are the relations with business processes. The growingly strong interconnection between IT and business processes has made clear the need to link higher level (i.e. business related) and lower level (i.e. at the software development level) goals. This need appears especially often when it comes to evaluating IT-supported business strategies (Becker & Bostelman, 1999; Buglione & Abran, 2000; Bianchi, 2001; Card, 2003). The relation of higher with lower level goals has been recently described – in connection with the problem of evaluating the effectiveness of business strategies and IT solutions – by Basili *et al.* (2010). In IT solutions are viewed as (part of) the strategies that in a given context and under given assumptions lead to achieving given business goals (Basili *et al.,* 2010). However – also because they focus on the evaluation problem – Basili *et al.* do not propose a clear and systematic way of modelling the requirements hierarchy. The proposal contained herein of systematic refinement of business goals in a given environment into needs to be achieved could help clarify the concepts of 'strategy' and 'context' proposed in Basili *et al.* (2010).

7. Conclusion

This paper proposes the definitions of a few concepts that are needed to properly identify and describe user needs and requirements.

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 Figures 5–7.

There are several requirements definitions that can satisfy the users' needs. The analysis process has to take this issue into account: RE 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, 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 behaviour 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 which the problem frames and related methodology play an important role. In Section 5, a process for dealing with requirements hierarchies has been outlined, using UML activity diagrams. Since it has been shown that the methodology of problem frames can be successfully used in combination with UML (Lavazza & Bianco, 2004, 2006; Bianco & Lavazza, 2008), it will be useful to define UML-based representations of the needs and requirements, and a UML-compliant analysis process. Future work includes the definition of a methodology for building UML-based descriptions of goals, needs, and requirements and using problem frames methodology to carry out in the most seamless and integrated way refinements, correctness argument constructions, cost analysis, and any other useful activity.

As briefly discussed at the end of Section 4, the transition from goals to needs and from needs to requirements can be considered similar to the transformations at the base of POSE (Hall *et al.,* 2008; Hall & Rapanotti, 2011): formalizing the proposed concepts in the context of POSE is thus another goal of future activities. Actually, the proposed hierarchy of requirements can be seen as the base for defining methodological guidelines for using POSE transformations in dealing with high-level user goals and needs.

Finally, we can note that even though in industry there is still some confusion about the nature or requirements and their relationships with business goals, the situation is improving: for instance, some companies are starting to follow consistent guidelines, such as the Business Analysis Body of Knowledge (International Institute of Business Analysis), a collection of guidelines that reflects current generally accepted practices of business analysis.

Acknowledgements

The research presented in this paper has been partially funded by the project 'Metodi e tecniche per l'analisi, l'implementazione e la valutazione di sistemi software' funded by Università degli Studi dell'Insubria.

References

Albrecht, A. (1979) Measuring application development productivity. *IBM Application Development Symposium*, IBM Press, 83–92.

Anton, A.I. and C. Potts (1998) The use of goals to surface requirements for evolving systems, *20th International Conference on Software Engineering*, 19–25 April 1998, Kyoto.

Basili, V.R., M. Lindvall, M. Regardie, C. Seaman, J. Heidrich, J. Munch, D. Rombach and A. Trendowicz (2010) Linking software development and business strategy through measurement, *IEEE Computer*, **43**, 57–65.

Becker, S.A. and M.L. Bostelman (1999) Aligning strategic and project measurement systems, *IEEE Software*, **16**, 46–51.

Bianchi, A.J. (2001) Management indicators model to evaluate performance of IT organisations, *International Conference on Management of Engineering and Technology*, Vol. **2**, IEEE Press, 217–229.

Boehm, B.W., C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer and B. Steece (2009) *Software Cost Estimation with COCOMO II.* Upper Saddle River, NJ, USA: Prentice Hall Press.

Buglione, L. and A. Abran (2000) Balanced scorecards and GQM: what are the differences? *3rd European Software Measurement Conference*, Federation of European Software Measurement Association, 18–20 October 2000, Madrid, Spain.

Cañete-Valdeón, J.M., F. Enríquez, J. Ortega and E. Veláquez (2008) Clarifying the semantics of value in use cases through Jackson's Problem Frames, *Information Processing Letters*, **107**, 221–229.

Card, D. (2003) Integrating practical software measurement and the balanced scorecard, *27th Annual International Computer Software and Applications Conference*, IEEE CS Press.

COSMIC – Common Software Measurement International Consortium (2007) *The COSMIC Functional Size Measurement Method – version 3.0 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003)*. Available at http://www.cosmicon.com/dl_manager4.asp?id=73 (accessed August 22, 2012).

Dardenne, A., A. van Lamsweerde and S. Fickas (1993) Goal-oriented requirements acquisition, *Science of Computer Programming*, **20**, 3–50.

Dardenne, A., S. Fickas and A. van Lamsweerde (1991) Goal-directed concept acquisition in requirements elicitation, *6th International Workshop on Software Specification and Design*, Como, 14–21.

Darimont, R. and A. van Lamsweerde (1996) Formal refinement patterns for goal-driven requirements elaboration, *4th ACM Symposium on the Foundations of Software Engineering*, October 1996, San Francisco, 179–190.

Del Bianco, V. and L. Lavazza (2008) Enhancing problem frames with scenarios and histories in UML-based software development, *Expert Systems – The Journal of Knowledge Engineering*, Vol. **25**, no. 1, Blackwell Publishing.

Del Bianco, V. and L. Lavazza (2009) Applying the COSMIC functional size measurement to problem frames, *14th IEEE International Conference on Engineering of Complex Computer Systems – ICECCS '09*, 2–4 June 2009, Potsdam (Germany): IEEE Computer Society.

Dubois, E., E. Yu and M. Petit (1998) From early to late formal requirements: a process-control case study, *9th International Workshop on Software Specification and Design*, 16–18 April 1998, Mie, Japan: IEEE CS Press, 34–42.

Galorath, D.D. and M.W. Evans (2006) *Software Sizing, Estimation, and Risk Management*, Auerbach Publications.

Gunter, C., E. Gunter, M. Jackson and P. Zave (2000) A reference model for requirements and specifications. *IEEE Software*, **3**, 37–43.

Hall, J.G. and L. Rapanotti (2005) Problem frames for socio-technical systems, In *Requirements Engineering for Socio-Technical Systems*, A. Silva and J.L. Mate (eds), chapter 19, Idea Publishing Group, 318–339.

Hall, J.G. and L. Rapanotti (2009) Assurance-driven design in problem oriented engineering, *International Journal on Advances in Systems and Measurements*, **2**, 119–130.

Hall, J.G. and L. Rapanotti (2011) Software engineering as the design theoretic transformation of software problems. *Innovations in Systems and Software Engineering*, London: Springer.

HALL, J.G., L. RAPANOTTI and M. JACKSON (2008) Problem Oriented Software Engineering: solving the package router control problem, *IEEE Transactions on Software Engineering*, **34**, 226–241.
INTERNATIONAL INSTITUTE OF BUSINESS ANALYSIS, *The Guide to the Business Analysis Book of Knowledge, version 2*. http://www.theiiba.org.
JACKSON, M. (1995) *Software Requirements & Specifications – A Lexicon of Practice, Principles and Prejudices*. ACM Press, Addison-Wesley.
JACKSON, M. (2001) *Problem Frames – Analysing and Structuring Software Development Problems*, Addison-Wesley, ACM Press.
KAINDL, H. (2000) A design process based on a model combining scenarios with goals and functions, *IEEE Transactions on Systems, Man and Cybernetic*, **30**, 537–551.
KUJALA, S., M. KAUPPINEN and S. REKOLA (2001) Bridging the gap between user needs and user requirements, in *Advances in Human-Computer Interaction* I, N. Avouris and N. Fakotakis (eds), Rio Patras, Greece: Typorama publications, 45–50.
LAVAZZA, L. (2010) User needs vs. user requirements: a problem frame-based view, *International Workshop on Advances and Applications of Problem Orientation (IWAAPO'10)*, 8 May 2010, Cape Town, South Africa.
LAVAZZA, L. and V. DEL BIANCO (2004) A UML-based approach for representing problem frames, *1st International Workshop on Advances and Applications of Problem Frames (IWAAPF)*, 24 May 2004, Edinburgh: IEE.
LAVAZZA, L. and V. DEL BIANCO (2006) Combining problem frames and UML in the description of software requirements, *Fundamental Approaches to Software Engineering (FASE06)*, 25 March–2 April 2006, Vienna: Springer.
LAVAZZA, L., V. DEL BIANCO (2008) Functional size measurement based on problem frames: a case study, *3rd International Workshop on Advances and Applications of Problem Frames (IWAAPF)*, 10 May 2008, Leipzig: ACM press.
LEE, J. (1991) Extending the Potts and Bruns Model for recording design rationale, *13th International Conference on Software Engineering*, 13–17 May 1991, Austin, Texas, USA: IEEE-ACM, 114–125.
LI, Z. (2007) Progressing problems from requirements to specifications in problem frames, PhD Thesis, Department of Computing, The Open University, Walton Hall, Milton Keynes, UK.
MOFFETT, J.D. and J.A. MCDERMID (1995) Goals as a focus for conflict management in requirements engineering. *RE '95: Second International Symposium on Requirement Engineering*, 28–30 March 1995, York, UK: IEEE Computer Society.
MYLOPOULOS, J., L. CHUNG and E. YU (1999) From object-oriented to goal-oriented requirements analysis, *Communications of the ACM*, **42**, 31–37.
POTTS, C. (1995) Using schematic scenarios to understand user needs, *1st Conference on Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, 23–25 August 1995, Ann Arbor, Michigan: ACM.
RAPANOTTI, L., J.G. HALL and Z. LI (2006) Deriving specifications from requirements through problem reduction, *IEE Proceedings: Software*, **153**, 183–198.
ROSS, D.T. and K.E. SCHOMAN (1977) Structured analysis for requirements definition, *IEEE Transactions on Software Engineering*, **3**, 6–15.
RUBIN, K.S. and A. GOLDBERG (1992) Object behavior analysis, *Communications of the ACM*, **35**, 48–62.
SEATER, R. and D. JACKSON (2006) Problem frame transformations: deriving specifications from requirements, *2nd International Workshop on Advances and Applications of Problem Frames*, 23 May 2006, Shanghai, China.
SEATER, R., D. JACKSON and R. GHEYI (2007) Requirement progression in problem frames: deriving specifications from requirements, *Requirements Engineering Journal (REJ'07)*, **12**, 77–102.
SOMMERVILLE, I. and P. SAWYER (1997) *Requirements Engineering: A Good Practice Guide*. Chichester: Wiley.
VAN LAMSWEERDE, A. (2000) Requirements engineering in the year 00: a research perspective, keynote, *22nd International Conference on Software Engineering*, 4–11 June 2000, Limerick, Ireland: ACM Press, 5–19.
VAN LAMSWEERDE, A. (2001) Goal-oriented requirements engineering: a guided tour, *5th IEEE International Symposium on Requirements Engineering*, 27–31 August 2001, Toronto, Canada: IEEE Computer Society, 249–263.
YU, E.S.K. (1993) Modelling organisations for information systems requirements engineering, *1st International Symposium on Requirements Engineering*, January 1993, San Diego, CA, USA: IEEE, 34–41.
YU, E.S.K. and J. MYLOPOULOS (1994) From E-R to 'A-R': modelling strategic actor relationships for business process reengineering, *13th International Conference on the Entity-Relationship Approach*, Manchester, UK.
ZAVE, P. and M. JACKSON (1997) Four dark corners of requirements engineering, *ACM Transactions on Software Engineering and Methodology*, **6**, 1–30.

The author

Luigi Lavazza

Luigi Lavazza is associate professor at the University of Insubria at Varese. He also cooperates with CEFRIEL (ICT Center of Excellence For Research, Innovation, Education & Industrial Labs partnership). His research interests include: software process modelling, measurement, and improvement; the measurement of software products, especially concerning quality and the functional size; model-based development, especially concerning real-time and embedded software; requirements engineering and software development environments and tools. He participated in several research projects co-funded by the European Union or by the Italian government, often guiding the research unit of CEFRIEL or of the researchers of the University of Insubria at Varese. He is co-author of over 100 scientific articles, published in international journals, or in the proceedings of international conferences or in books. He serves on the program committees of a several international conferences. Luigi Lavazza is an IARIA fellow and member of the IEEE computer society.