Editorial: A roadmap of Problem Frames research

Karl Cox[a], Jon G. Hall[b], Lucia Rapanotti[b]

[a]National ICT Australia Ltd., Australian Technology Park, Eveleigh, Sydney, NSW 1430, Australia

[b]Computing Research Centre, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK

It has been a decade since Michael Jackson introduced problem frames to the software engineering community. Since then, he has published further work addressing problem frames as well as presenting several keynote addresses. Other authors have researched problem frames, have written about their experiences and have expressed their opinions. It was not until 2004 that an opportunity presented itself for researchers in the field to gather as a community. The first International Workshop on Advances and Applications of Problem Frames (IWAAPF'04) was held at the International Conference on Software Engineering in Edinburgh on 24th May 2004. This event attracted over 30 participants: Jackson delivered a keynote address, researchers presented their paper presentations and an expert panel discussed the challenges of problem frames.

Featuring in this special issue are two extended papers from the workshop, an invited contribution from Jackson in which he positions problem frames in the context of the software engineering discipline, and this article where we provide a review of the literature.

1. Introduction

For the past decade, researchers in early lifecycle software engineering have explored and expanded upon the problem frames approach. Problem frames were introduced in 1995 in the book *Software Requirements & Specifications* [44]. They provide expression for the ideas that Jackson has developed (with others, notably Pamela Zave), based on the experience within the software industry. Problem frames can capture in a simple way much that is complex.

Problem frames provide an approach to understanding and describing real world software-intensive problems, for example control, information, business, military or medical systems.

Problem frames provides software requirements[1] engineering with an approach to requirements analysis that has sparked interest in the requirements community. In addition to Jackson's own contribution to this special issue, this editorial provides a review of the research literature that has developed, applied, advanced or discussed the problem frames

[1]In the following, unless specifically stated, we will mean software requirements when we speak of requirements.

idea.

1.1. Structure of the paper

In this editorial, we provide a roadmap of problem frames research. We address four core areas of research in problem frames. These are:

Principles: Section 2 discusses the principles embodied in the problem frames approach, and the criteria for correctness of the approach in the context of requirements engineering.

Techniques: Section 3 presents the basic techniques of problem frames for the representation, classification and transformation of software problems.

Processes: Section 4 looks at analysis processes based on problem frames and how problem frames fit in the wider context of software development methods.

Semantics: Section 5 focuses on problem frames semantics.

In Section 6 we draw some conclusions from our analysis of the literature and suggest possible future research.

2. An overview of the problem frames approach

In this section we lay out the basic principles of the problem frames approach and discuss its position within requirements engineering.

2.1. Problem frames in principle

Problem frames appear to have been motivated by Jackson's many long-standing concerns for software development. These include:

- that software development is not an engineering discipline (nor will it become one for some time) because of the failure of its practitioners to specialise;
- that software developers have found many reasons or 'denials' [45] for paying more attention to the solution than to the world that forms the problem's context; these are: denial by prior knowledge, denial by hacking, denial by abstraction and denial by vagueness;
- that requirements engineering has an 'irksomely informal subject matter' that 'tempts us to exclude it from software engineering' [49].

2.1.1. Specialism

Reflecting the specialism of engineers in traditional engineering disciples, in [52] it is asserted that some parts of software engineering lack an 'essential ingredient': the 'thick layer of knowledge specialised to particular product [software] groups' that, for instance, would distinguish a bridge engineer from a naval engineer in traditional engineering. As noted earlier, the way forward for software development is through increasing specialisation by requirement type and product class.

The problem frames approach provides what might be called hooks for specialisation that allow the templated application of software development expertise through the recognition and subsequent (specialist) solution of problems. A problem frame is precisely an opportunity to apply specialist skills and knowledge of what constitutes a good solution for a particular type of problem.

Jackson, in his article for this special issue, considers the problem frames approach to software development and how it contributes to 'software development['s long-standing aspiration] to deserve recognition as an engineering discipline'. He argues that the problem frames approach contributes by separating normal and radical engineering [85], and allowing specialism in software development.

2.1.2. Denials

To counteract the temptation to denial, Jackson gives a (seemingly) eclectic collection of principles [45] that feed directly into the principled basis of problem frames. The first is:

von Neumann's principle: there is no point in being precise when you don't know what you are talking about.

In the same paper, it is asserted that:

> 'Our very first obligation is to clarify the concepts and issues with which a system is concerned.'

Indeed, the need for precision in software development, but only if it is not at the expense of misunderstanding the problem to be addressed, is a recurrent observation [40, 43, 44, 47, 48, 51, 55]. Understanding the problem is a dominant theme in the problem frames approach.

The second principle is:

The principle of reductionism: to choose the simplest phenomena and define — where appropriate — more complex constructs in terms of them.

In software development one has considerable freedom to choose the ground terms and one must be careful always to choose those phenomena (those things that appear to exist or to be the case) which are most exactly and reliably recognised, and that this provides the best starting point for understanding and describing the world [44]. Problem frames place great emphasis on the choice and use of phenomena, particularly those that are shared between two real world domains, or a real world domain and the machine.

The third principle is:

The Shanley principle: (paraphrased in [45]) to characterise the complexity and interrelatedness of the world as 'every part of the world may play many roles, and perform many functions'.

Accordingly, the result for software development is that, at the level of domain description and problem analysis, the Shanley principle demands parallel consideration of different

views of the domains and their related sub-problems. Indeed, the primary mode for solving a problem in the problem frames approach is to decompose it into its principal parts, and produce solutions for each part which are then critically recomposed to form a solution for the problem.
The fourth and final principle is:

Montaigne's principle: the greater part of this world's troubles are due to questions of grammar (Montaigne, 16th century).

The problem frames approach places great store on the importance of the separation of 'asserted true' from 'desired true'; in grammatical terms, these are, respectively, the *indicative* and *optative* moods (of which more in Section 2.2).

2.1.3. 'Irksome informality'

It is argued in [49] that requirements engineering is hard because:

> 'It demands descriptions of an informal world precise enough to be intelligibly related to the formal behavior of the computer and its software, and reliable enough to guarantee that the formal behavior will evoke the required results in the informal world.

As a solution, 'A Discipline of Description' [50] is proposed in which proper emphasis is placed on traditional engineering notions such as description, designations, definitions and error analyses; indeed, Jackson and Zave's proposal for 'multi-paradigm' domain descriptions [60] is now widely accepted.

The problem frames approach provides tools and notations for the description of domains (in various languages), and places great emphasis on getting descriptions and definitions correct and designations accurate. It also has a focus on error analysis through the development of correctness arguments.

The problems frames notation also emphasises problems over solutions. With its emphasis on providing tools that support the understanding of a problem, the problem frames approach supports the contribution of a 'problem owner' with specific domain knowledge in the requirements engineering process.

2.2. Problem frames in engineering

In the problem frames approach, a software development problem is 'a need for a useful machine, and the solution to the problem is the construction of a suitable machine to satisfy the need' [43]. A need consists of the environment of the useful machine described in terms of what is given and what is desired, in separate descriptions. In the approach, the useful machine is a specialisation of a general-purpose computer, the specialisation being achieved by software. The end-product of the requirements analysis task is a specification for that software. Software problems are modelled in diagrams complemented by textual, and other, descriptions (see Section 3.1). The context is formed from what exists in the environment of the useful machine: the givens. The requirement is what is desired. The two descriptions are:

- The 'givens' (indicative descriptions) are real-world (i.e., physical) domains that stand in a fixed structure with respect to each other, and communicate with each

other using shared phenomena (which can be control phenomena, event projections and event classification [46]). The given descriptions are statements that are true irrespective of the machine's presence or behaviour.

- The 'desires' (optative descriptions) are constraints on the environment's behaviour that a correct machine should bring about when deployed.

The givens are described in the indicative mood whereas the desires are expressed in the optative mood; in line with Montaigne's grammatical principle, combining indicative and optative descriptions would be 'seriously confusing' [58]. Problem frames take care to separate, and keep distinct, indicative and optative descriptions.

The correctness of a problem frame development hinges on the five criteria of [89], which proposed a split of Software Development into Requirements Engineering and Software Implementation. The split is motivated in general by the need to better define requirements 'engineering' in terms of the precise nature of requirements, specifications, and domain knowledge, and their interrelation, and specifically to determine the 'inputs' and 'outputs' of a requirements engineering process. One of the five criteria deals only with a pathological case in the logical description of the world and the machine which would mean that they cannot co-exist (see [28, 32] for more details). A constructive approach, such as that of problem frames, will not encounter situations in which this criterion does not hold. The other four criteria of [89] can be restated informally as:

1. There is a validated customer requirement that expresses all of the customer's desires. 'Validated' in this and other uses means 'informally checked'; informal, in this sense, is all that can be achieved in the (informally) described world;

2. There is a collection of validated descriptions that represent real world (i.e., problem domain) knowledge;

3. There is a machine specification that is not stated in terms of any unshared actions or states, and that does not refer to the future.

4. The contribution of the specification to its environment is one that satisfies the requirement.

Because problem frames embody such broad principles and ideas they have stimulated developments in many and different directions; in this paper we discuss many of them.

2.3. Problem frames and other approaches to Requirements Engineering

In object-oriented software development, domain modelling [20] produces a description of an application domain. An analysis model can then be derived from the domain model by making assumptions on the boundary between system and environment, and replicating domain model structure within the analysis model [64]. Analysis patterns [25] provide a particular technique for building domain and analysis models for business applications. It is not an approach to problem analysis *per se*, but provides a catalogue of recurrent analysis problems within the business domain. The main difference between these approaches and problem frames is the emphasis that problem frames place on separating

problem and solution, and indicative and optative descriptions. In addition, problem frames take a multi-paradigm approach to problem and solution descriptions, and do not assume, for instance, an object-oriented approach.

Domain theory [80, 81] places a principled foundation beneath domain modelling, so that the description and analysis of a problem can be developed via analogical reasoning, knowledge reuse, and developer expertise. In intention, problem frames and domain theory share much of their motivation. The major difference between the two approaches is the emphasis that problem frames place on the correctness of a specification with respect to a detailed problem description, rather than building the problem description. Indeed, domain theory could be seen as a useful precursor to a problem frame development.

Goal-oriented approaches [84] focus on the use of goals and their refinement for the elicitation, analysis and specification of a software system's requirements. A goal-oriented approach could be integrated within a problem frames development, allowing a detailed derivation of a system's requirements from a collection of high-level goals. We review some work which achieves this later on in this paper.

Use cases [18] focus on interactions at the boundary of the software system, allowing system operations to be specified from identified goals of actors external to the system. As such, and in contrast to the problem frames approach, use cases focus on that part of the problem context that is at the boundary of the system.

The relationships we have identified between problem frames and these approaches suggest that their techniques may be of use to a specialist developer who wants to develop (or maintain) a particular problem description that might begin a problem frames development. In addition, object-oriented domain modelling and analysis patterns could help in the subsequent development of a solution.

Problem frames may thus provide a framework in which many other techniques may work together, with the problem frame notion of correctness being an additional bonus.

3. Problem frames techniques

In this section we review some of the main concepts of problem frames, and survey related contributions from the literature.

3.1. Problem representation

In this article, we use the graphical notation for problem frames that appears in [56].

A *context diagram* defines the context of a problem by capturing the characteristics and interconnections of the parts of the world the problem is concerned with. Context diagrams typically include domain descriptions that describe the environment of the system to be built, and descriptions of their shared phenomena. In a context diagram, domains are not necessarily in direct contact with the machine to be built, and shared phenomena may exist between domains as well as between domains and the machine.

A problem is captured in the notation of *problem diagrams* that, in turn, are formed from context diagrams by the addition of a requirement. Figure 1 gives a simple example of a problem diagram. In the figure, the software problem is to specify a machine (the solution) to control a device (the problem context, consisting of a single domain in this case) so that a certain work regime (the requirement) is satisfied. The link between device and machine in the example indicates the phenomena that are shared between

them. Phenomena can be, for instance, events or states or commands. In this example, the shared phenomena are commands that the controller machine can issue to switch the device on and off. (That such phenomena are controlled by the controller machine is indicated by a ! after the abbreviation *CM*). Provided separately from the diagram are the designations of phenomena, that is descriptions which ground the problem's vocabulary in the physical world.

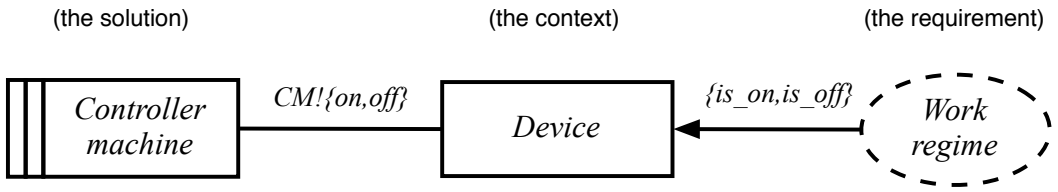

Figure 1. A simple problem diagram

Not included in the problem diagram, but an essential part of the problem definition, are also descriptions of all given domains and the details of the requirement. An important point is that the problem frames approach does not prescribe the notation to be used for these descriptions. Formal and informal descriptions are equally acceptable, provided they are precise enough to capture the characteristics of interest of the various respective domains. For instance, in our example, descriptions can be as simple as:

Device: a device that can assume one of the following states: *is_on* and *is_off*. Commands *on* and *off* can be issued to effect state changes in the device.

Work regime: the device can be turned on at any time, but should not operate for longer than 2 hours uninterrupted. After 2 hours of operation, the device should be turned off.

3.2. Problem classification

Problem frames are used to classify problems. *Problem frame diagrams* (or *frame diagrams* for short) are diagrams used to classify problems and to relate to them known solutions. Together with a generic problem description, a problem frame diagram acts as a template for recognising problems in its class. In form, a problem frame diagram is simply a problem diagram, annotated with domain and phenomena markings. The markings state characteristics of the domains and phenomena. The characteristics of causal, lexical and biddable domains—and event, causal and symbolic (and other types of) phenomena—are succinctly described in [54]. The frame diagram is also associated with a correctness argument template (referred to as the *frame concern*) that is used to determine whether a proposed solution specification stands in the correct relationship to the world description and requirements, as required by criterion 4 of [89] (see Section 2.2). The combination of generic problem description, problem frame diagram and associated

correctness argument template, constitute a *problem frame*, that is the characterisation in the abstract of a software problem class.

An example of problem frame is the Required Behaviour frame [56]. A required behaviour problem is one in which a machine has to be specified to impose some control on some part of the world, so that it behaves in a way which satisfies certain conditions. (Indeed, the problem discussed in Section 3.1 is an instance of such a type.) Figure 2 gives the Required Behaviour frame diagram.

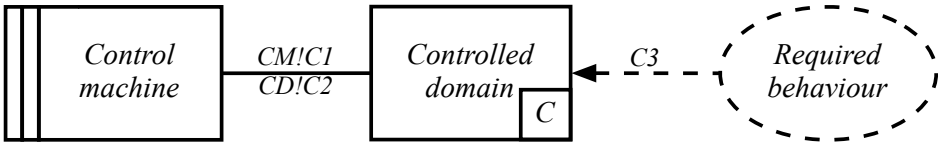

Figure 2. Required Behaviour frame diagram

The annotation C on the Controlled domain expresses the fact that such a domain is causal, that is, a domain whose properties include predictable relationships between its phenomena. The annotation on the links between domain, machine and requirement is a reminder that causal phenomena must be identified for the specific problem to be analysed and for a correctness argument to be made (again, ! indicates control). The correctness argument for this class of problems follows the template indicated in Figure 3. The template tells us how the various descriptions (specification, domain properties and requirement) and shared phenomena can be linked together to demonstrate that the specification indeed meets the requirement within the given context.

To match a problem to a problem class as represented by a problem frame, the problem must match the generic problem description of the problem frame. The problem diagram and its parts must:

- match the problem topology — as captured in the problem frame diagram;
- match the domain characteristics — as captured by the domain markings in the problem frame diagram;
- match the characteristics of the shared phenomena — as captured by the markings on the connections in the problem frame diagram.

When matched, problem frames are used to identify sub-problems, as will be discussed in Section 3.3.

The primary source of problem classes characterised as problem frames is [56], which presents five basic problem frames: the Workpieces frame (also introduced in [41, 42]), the Required Behaviour frame (explained above), the Commanded Behaviour frame, the Information Display frame and the Transformation frame. Jackson has defined other

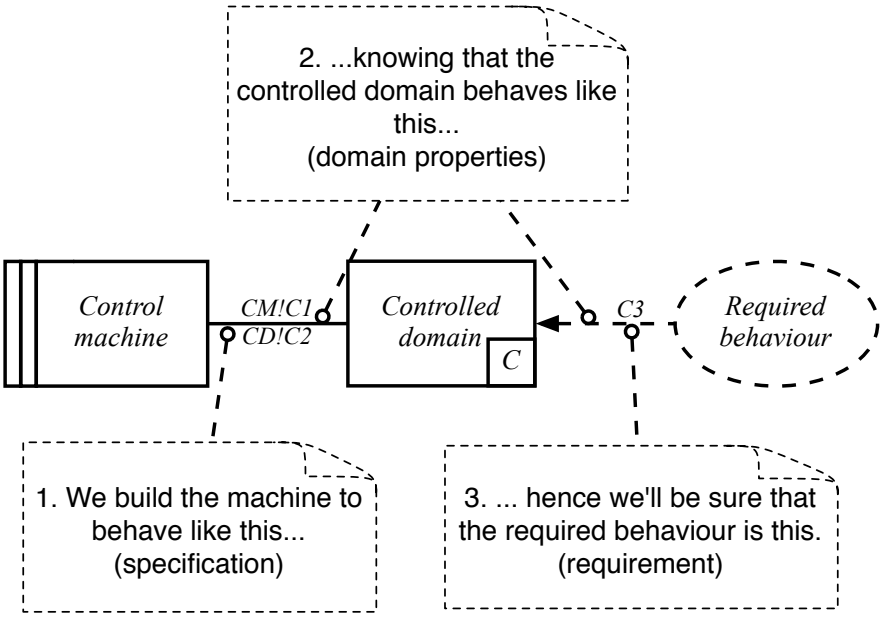

Figure 3. Required Behaviour frame concern

problem frames elsewhere, including the JSD and Environment-Effect frame [41, 42], the Simple Control frame and Simple Enquiry frame [53], the Simple Information Answers frame [54], and the Four Variable Model problem frame [59].

In addition, other proposed problem frames have appeared in the literature. Wieringa [87] introduces a Declarative problem frame which captures problems in reactive systems; inputs into the system force certain outputs based on rules and laws of the physical domain and so shape that domain. This problem frame is introduced with no graphical representation. Nelson *et al.* [72] introduce a suite of twelve problem frames that address problems in the area of geographic applications. Konrad and Cheng [62] describe a problem frame for the Actuator-Sensor pattern used in specification for an automotive embedded system. Bray and Cox [14] propose the Simulator frame to address recognised simulation problems. Hall and Rapanotti [33] define the User Interaction frame in order to capture the basic problem of defining a user-machine dialogue.

Nelson *et al.* [73] show how formal analysis techniques can be used to encode and reason about frame concerns. In particular, they use the Alloy language [36] for domain, requirement and machine descriptions, which allows them to discharge correctness arguments in a (semi-)automatic fashion. They provide a more detailed example of this in their paper in this special issue.

Lin *et al.* [69, 70] introduce 'abuse frames' as an extension to problem frames, in order to deal with security risks. An abuse frame captures classes of security problems expressed as problem diagrams. Abuse frames introduce a new form of correctness argument. Because it is from the attacker's perspective, in form the argument is existential (i.e., the attacker

is looking for the existence of at least one possible abuse) rather than universal (i.e., for all possible behaviours of the machine, an attack is possible).

3.3. Problem transformation

Most problems are too complex to fit basic problem frames. Indeed most problems have complex contexts, with requirements which are initially abstract and far removed from any machine phenomena. During any process of requirements analysis, requirements become refined and new requirements, hence new problems, discovered. In order to support this form of analysis, problem frames includes two forms of problem transformation [56]: problem decomposition and problem progression. We now focus on the characteristics of the two techniques. In Section 4 we will discuss how they have been applied in the literature within a variety of analysis processes and frameworks.

3.3.1. Problem decomposition

Problem decomposition (which is not the same as hierarchical decomposition [37]) allows for a divide-and-conquer approach to a problem solution: from an initial complex problem, simpler and smaller sub-problems are derived each of whose solution will contribute to the solution of the original problem. This decomposition can be accomplished either by matching problem frames or by applying generic decomposition heuristics (a number are given in [54, 56]), or based on specific knowledge of the problem (this requires particular skills of the analyst).

By the matching of problem frames within a problem diagram, sub-problems are identified corresponding to known classes of problems. The conditions under which this matching can be achieved were described in Section 3.2. From the initial complex problem diagram, via a frame diagram, a 'projection' of the problem into a simpler sub-problem diagram can be achieved. As the sub-problem is identified, so too is a simpler requirement that is relevant to the projected sub-problem context. In practice this means that within the original problem requirement, we can identify a sub-requirement for the sub-problem. Through the repeated identification of sub-problems in this way, a large problem finds expression as a collection of sub-problems, with requirements that are decomposed over the projections.

Once sub-problems are identified and solved, their solutions need be recomposed to contribute to the solution to the initial complex problem. This gives rise to a number of concerns; in [48] it is stated that:

> 'Problem complexity arises from the interactions of [a problem's] subproblems and of their solutions, both as a problem structure and as a solution structure.'

This issue is also addressed in [57]. Such interaction can only occur in sub-problems when they share either a problem domain or a machine domain, in which case care must be taken in their composition as a solution to the complex problem. The correctness of the composition is the subject of *composition concerns*, such as: Consistency, Precedence, Interference and Scheduling [56], Synchronisation and Contradiction [54]. Composition concerns are motivated by the fact that, in problem projection, the division of requirements loses information concerning the sub-problem interactions. The interference and

scheduling concerns are notable in that they involve sub-problems that share machine domains, and so are to do with solution composition; all other concerns (considered here) address the composition of domains shared among the sub-problems.

Problem decomposition and solution recomposition has been addressed in a number of recent works. Laney *et al.* [65] propose Composition Frames to address some composition concerns. In particular, Composition Frames include the specification of a Composition Controller to resolve conflicting requirements, akin to the notion of an architectural connector [2].

Rapanotti *et al.* [77] and [33] propose Architectural Frames (AFrames) as the combination of a problem class and a class of software architectures. AFrames regularise problem decomposition and subsequent solution recomposition through a characterisation of the solution architecture. This has the dual benefit that problem decomposition is guided by the architecture (hence made easier from an analyst's viewpoint) and many composition concerns become trivial. Incidentally, AFrames decomposition is achieved through an expansion of the solution machine architecture, which can be regarded as a further type of problem transformation. One of the benefits claimed of AFrames is that they allow for quality trade-off analysis (in the style of Bass *et al.* [4]) in a problem frames development.

The decomposition of complex problems into sub-problems is also addressed by Bleistein *et al.* [7–11], where goal decomposition complements problem decomposition and is used to drive the decomposition process.

Sub-problem decomposition in the context of case studies is addressed in Haley [29] and Haley *et al.* [30, 31].

It should be stressed that, despite these efforts to ease the task of problem decomposition and matching basic problem frames, little is known of how the problem frames approach performs in the analysis of complex software problems. In an empirical study, Phalp and Cox [76] compare students' and practitioners' ability to match problem frames to a variety of realistic software problems. The five basic problem frames, and some derived from their composition (so-called multi-frames), were used in the experiment. It was found that single frame problems were readily identifiable, but multi-frame problems were not, and the authors raise the question of whether problem frames are scalable to industrial software problems.

3.3.2. Problem progression

Problem progression allows the simplification of a problem context (by removing one or more domains from it) whilst simultaneously re-expressing the requirements to apply in the new context. With the removal of context, a requirement is re-expressed in terms of the remaining phenomena, hence moves *closer to the machine*. The removal of a domain from a problem's context and the re-expression of the requirement are, of course, related by the need to preserve the solution, which remains invariant throughout the transformation. Note that problem progression is mentioned in [56], but is not described in any great detail. It remains an under-explored area of problem frames at present.

4. Problem analysis and solution synthesis with problem frames

Problem frame analysis begins by understanding a problem context through the development of a context diagram [56]. Amongst other things, a context diagram for a problem

bounds a problem's scope by identifying what is and what is not relevant in a problem statement; as part of this, the context diagram identifies the in-scope domains. A problem diagram is developed from a context diagram by associating requirements with subsets of the domains. The process of deriving problem diagrams is thus — in skilled hands — relatively straightforward. Jackson [56] provides numerous illustrative examples of the step-by-step transition from context diagrams to problem diagrams, and from problem diagrams to sub-problem diagrams, that involve the application of problem transformation tools. Problem frames, however, are best regarded not as an analysis process *per se*, but primarily as a collection of techniques that can be used flexibly within a chosen analysis and development process. In this section we review approaches which have developed analysis processes based on problem frames (in Section 4.1), or located problem frames within wider software development processes (in Section 4.2).

4.1. Analysis processes based on problem frames

Bray [12] proposes a loose process framework based on problem frames, which he labels Problem Domain Oriented Analysis (PDOA). In the PDOA framework, Bray suggests these analysis steps:

1. Abstract the problem domain:
   (a) identify the sub-domains;
   (b) identify sub-domain interactions (in terms of shared phenomena);
   (c) characterise each sub-domain;
   (d) generate a(n extended) context diagram.
2. Recognise relevant, standard frame(s).
3. Fit frames to the problem (as best we can).
4. Use Kovitz's tables [63] (see also Section 4.2) for the relevant frames to guide further analysis and documentation.

Although the steps are presented here as a sequence, the process is iterative; hence the steps should be revisited a number of time during analysis. PDOA does not to make use of problem diagrams as an intermediary step from context diagrams to sub-problem diagrams through problem frames decomposition. There are many examples of application of PDOA in [12], where problem analysis is followed through to requirements and specification documentation. The work also gives a number of heuristics for defining problem diagrams and related descriptions, which complement those of [44].

Other process frameworks have been proposed. Bleistein *et al.* [7–11] show how the context of a problem and its requirements (expressed as goals) can be developed in parallel. They apply the problem frames approach in the context of explicitly modelling and tracing from business strategy objectives, described through goal modelling [27, 88], and business model context, described through Jackson's context diagrams. Combined, they represent problem diagrams that capture the requirements problem from business strategy thorough to concrete system requirements. To add detail to the connections between goals and

context, in [9, 10] they introduce process models in the form of Role Activity Diagrams, extending the work described in [21–23].

Weiringa *et al.* [86] consider e-commerce software problems beginning with the value a software product might generate for an organisation, and showing how software requirements can be defined to be consistent with higher level requirements. In problem frame terms, their domains and shared phenomena are described in terms of manipulations of business value and the way value is transferred between domains. A key observation in the Weiringa approach is that the manipulation of value by the organisation should be aligned with business goals; similarly, business processes must be aligned with value manipulation, and software aligned with business processes.

Also in the context of business, Cox and Phalp [21] outline an approach that takes business process modelling as a starting point for problem frame analysis, showing its application in an industrial study of an on-line financial system in [23]. They suggest the mapping from process model to problem frames as six steps:

1. Explore the problem context.
2. Produce or revisit process models.
3. Identify outcomes of interactions between roles in the process model.
4. Identify domains from outcomes using Bray's domain taxonomy [12].
5. Identify potential rules that govern interactions (the requirements).
6. Match problem frames.

Their analysis process does not always lead to a decomposition into sub-problems that match Jackson's basic problem frames [56], suggesting that another set of problem frames may be needed for this domain. Cox *et al.* [22] describe an alternative analysis approach for the same case study, which also comes to the same conclusion.

Jeary and Phalp [61] argue that, for web-based development, a business model is a potential starting point for problem frames analysis. Moreover, they argue that web development is not too different from traditional software development; hence that Jackson's five basic problem frames should be sufficient in this application domain, drawing different conclusions to those of [23].

Hall and Rapanotti [33] propose an extension to problem frames that allows for both machines *and* humans to be explicitly considered as part of the solution of a socio-technical problem. The authors introduce a new solution domain, the *knowledge domain*, to represent the human for which some instruction has to be designed (this by analogy to the machine domain in the original problem frames approach for which a specification is sought). They propose both an extension of the problem frames notation and its foundation in the reference model of Gunter *et al.* [28], in order to allow for the introduction of knowledge domains.

Following up from that work, Brier *et al.* [15] apply this socio-technical approach to illustrate the covariance of requirements and solution domain in socio-technical problems. They argue that the State Change Model of Stacey [79] in which change is described

as open (i.e. totally unpredictable), contained, or closed (i.e. totally predictable), is usefully correlated to the flexibility needed of any humans in the solution space. The use of problem diagrams, with the explicit representation of the problem context, allows such covariance.

4.2. Problem frames in software development

Researchers view problem frames differently in terms of what happens after a complete problem diagram has been identified. Aligned with the four criteria of [89], Jackson [56] identifies the outcomes of a problem frame analysis as three documents: the given domain descriptions, a requirements description and a machine specification. These three descriptions can then be used to determine whether criterion 4 in [89] is met, i.e., the specification provides a solution to the problem.

One view of problem frames and where they fit in relation to subsequent development work is presented by Kovitz [63] and then expanded upon by Bray [12]. In [63], Kovitz views the derivation of problem frames as a precursor to conducting a full requirements analysis and subsequent specification. He suggests that the fitting of problems to problem frames allows the engineer to select the appropriate requirements analysis techniques in order to explore the requirements in finer detail. He presents tables that list the techniques and tools that can be used to explore and document requirements and specifications for each case. Kovitz notes that the problem frames approach, as it stood from Jackson's first work introducing problem frames [44], was only about problem description and understanding where requirements had an effect in the real world, not the behavioural rules that describe the specification of the machine. Bray [12] expands Kovitz's tables [63], but neither author suggests a follow-up design or implementation approach.

A proposal to fit problem frames within a development method is that of Tomayko [83], who suggests that problem frames would provide the ideal requirements/problem description metaphor in eXtreme Programming (XP) [5]. Although the two approaches, XP and problem frames, may appear at first to be somewhat contradictory (XP values code over any other artefacts, such as models or documentation), Tomayko argues that there is no reason not to conduct a lightweight problem frames analysis as part of an agile development process [1].

Ways of connecting problem frames and the UML have been proposed by Choppy and Reggio [17], where they show how the Commanded Behaviour problem frame can be integrated with UML through class diagrams, use cases and statecharts, in such a way to make further UML analysis and design viable. In this special issue, they expand on this work to bring within the remit of UML each of the remaining basic problem frames defined in [56]. Note that this is the only work which provides solution methods for all of Jackson's five basic problem frames. Before that, only two of the frames had recognised solution methods. One is the Transformation frame, solved using Jackson Structured Programming (JSP) [38]; indeed, this frame was originally named the JSP frame [44]. The other is a variant of the Information Display frame, solved using Jackson System Development method (JSD) [39].

Lavazza and Del Bianco [66] have also combined problem frames with the UML. Their approach uses object temporal logic (OTL, [67]) and Real-time UML (UML-RT, based on ROOM [78]) to express problem diagrams.

Taylor [82] discusses an extension of problem frames into the object-oriented solution domain. His idea is to associate two artefacts with a problem frame:

- a list of required characteristics of each principal part (or domain in a frame diagram); and
- a class diagram of the necessary parts of an object-oriented solution that reflects in the machine the problem frame components.

The development of an object-oriented solution is completed by filling in the detail of each class's features, and by adding code to implement the behaviour specified in each feature's contract. Taylor's approach might be seen as choosing design patterns appropriate for a particular problem frame.

Within the context of the twin-peaks process model [74], Hall *et al.* [34] describe how problem and solutions can be related using problem frames (see also their work on AFrames described in Section 3.3.1).

Barroca *et al.* [3] have also researched architectures in order to support compositional and incremental problem analysis and design. Their approach is based on the combination of problem frames and coordination-based techniques, and allows for dynamic sub-problem composition, in the sense that composition is not constrained to follow pre-established decomposition structures. Their intention is to provide a basis for run-time system reconfiguration in the face of changing requirements.

For distributed systems, Haley [29] proposes an extension to problem diagrams so that cardinaility on arcs can be specified which would otherwise have to be expressed as a cardinality relationship on shared phenomena between domains. In other work, Haley *et al.* [31] introduce connection pseudo-domains to combine problem projection with the need to work at different levels of abstractions in the architectural design of components.

Lin [71] considers the use of problem frames in conceptual modelling of a problem domain. It is claimed that their method simplifies the process of modelling 'abstraction' concepts that are often composed of complex combinations of physical entities.

5. Semantics of problem frames

Work on problem frames semantics remains sparse. Work by Bjørner *et al.* [6] characterises the properties of some early frames (Translation frame, Control frame, Information Systems frame, Workpiece frame and Connection frame [44]) in terms of typical types of domains and requirements, and for two of them, the design of the solution. This might be used to determine the solution form. Choppy and Reggio [16] take this approach further, providing a formalisation of the Translation frame and Information System frame using CASL (the Common Algebraic Specification Language, [19]).

Delannay [24] proposes a semantic approach which uses UML as a means of describing a meta-model as a semantics for problem frames. The meta-model describes context diagrams, problem diagrams and problem frames, but does not capture the meaning of other elements, such as correctness of a machine specification with respect to its environment and requirements.

Although problem frames are closely related to the reference model for requirements engineering of Gunter *et al.* [28], a formal relation that underpins correctness has been

established only very recently. Working from that reference model, Hall *et al.* [35] propose a 'framework for understanding problem frames', which describe how a problem diagram can be considered as a 'challenge' to find an element of a set of solution specifications that satisfy the requirement in the given problem context and phenomena. The semantics is not constructive in the definition of such a set, and works at a level about that of description languages for domain, requirement and phenomena. The goal of the semantics, explicitly stated by Li *et al.* [68], is to allow:

- the representation of problem diagrams;
- the definition of equivalent problems;
- the underpinning of problem transformations; and
- the underpinning of correctness arguments.

6. Conclusions and future directions

This article has provided an analysis of problem frames research. It has described four core areas of problem frames (principles, techniques, processes and semantics) by discussing Jackson's original contributions and related work from other authors. We have found that techniques and processes are the areas on which researchers have focused most. Our reading of the literature has highlighted that, although much research is in progress, much work remains to be done in many areas. In particular, the following areas represent possible foci for future research.

New problem frames: the five basic problem frames [56] are only a starting point. Many more software problems could be captured as problem frames, paralleling the development of design patterns [26]. More research is required for the characterisation and validation of such new frames.

Realistic case studies: problem frames techniques are exemplified in the literature mainly through exemplars and small case studies, while their applications to realistic complex software problems is still lacking. Therefore there is little indication of how scalable problem frames are. Empirical evidence is needed in order to ascertain the true potential of problem frames for application in industry settings.

Teaching and dissemination: although the intellectual contribution of problem frames to requirements engineering has received some recognition in the research community, problem frames have yet to achieve the popularity of other analysis techniques, such as goal-based approaches [84], or use cases [18]. They are not usually taught in software engineering curricula (one exception is reported by Bray [13]) or in tutorials at conferences or workshops. The dissemination of the approach is then primarily through research papers, which are limited in scope and not widely accessed by students and practitioners.

Tool support: the complexity of current software problems demands support in terms of automatic tools which may ease the burden of development. Although many

tools exist to support other approaches (UML tools are notable examples), no such tools exist for problem frames. Ourusoff [75] argues that a tool for problem frames is 'urged and justified', and an essential ingredient in the uptake of the approach by practitioners. The maturity of problem frames and the development of supporting tools is something problem frames researchers need to come to terms with.

Acknowledgements

We are very grateful to Michael Jackson, Martin Shepperd, Bashar Nuseibeh, June Verner, the reviewers of the IWAAPF workshop and this special issue, and all others who were involved in the organisation and running of the workshop, without whom this special issue would not have been possible.

REFERENCES

1. Agility (2004). Agile alliance. http://www.agilealliance.org/home. Last accessed November 2004.
2. Allen, R. and Garlan, D. (1997). A formal basis for architectural connectors. *ACM Transactions On Software Engineering and Methodology*, **6**(3), 213–249.
3. Barroca, L., Fiadeiro, J., Jackson, M. A., Laney, R., and Nuseibeh, B. (2004). Problem frames: A case for coordination. In *Proceedings of the 6th International Conference on Coordination Models and Languages*, volume 2949 of *Lecture Notes in Computer Science*, pages 5–19.
4. Bass, L., Clements, P., and Kazman, R. (1999). *Software Architecture in Practice*. Addison Wesley Longman, Inc.
5. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.
6. Bjørner, D., Koussobe, S., Noussi, R., and Satchok, G. (1997). Michael Jackson's Problem Frames: Towards methodological principles of selecting and applying formal software development techniques and tools. In L. ShaoQi and M. Hinchley, editors, *Proceedings of the International Conference on Formal Engineering Methods (ICFEM'97)*. IEEE Computer Society Press.
7. Bleistein, S., Cox, K., and Verner, J. (2004a). Modeling Business Strategy in e-Business Systems Requirements Engineering. In *Proceedings of the 5th International Workshop on Conceptual Modeling Approaches for e-Business*, Lecture Notes in Computer Science, Shanghai, China. 9th November 2004.
8. Bleistein, S., Cox, K., and Verner, J. (2004b). Problem Frames Approach for e-Business Systems. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 7–15, Edinburgh. IEE. 24 May 2004.
9. Bleistein, S., Cox, K., and Verner, J. (2004c). RE Approach for e-Business Advantage. In *Proceedings of Requirements Engineering: Foundations of Software Quality (REFSQ'04)*, Riga, Latvia. 6–7 June 2004.
10. Bleistein, S., Cox, K., and Verner, J. (2004d). Requirements Engineering for e-Business Systems: Intergrating Jackson Context Diagrams with Goal Modelling and BPM. In *Proceedings of the 11th International Asia-Pacific Software Engineering*

Conference (APSEC 2004), pages 410–417, Busan, Korea. IEEE. 30th November–3rd December 2004.
11. Bleistein, S., Cox, K., and Verner, J. (2005). Strategic alignment in requirements analysis for organizational IT: an integrated approach. In *Proceedings of the 20th ACM Symposium on Applied Computing*, Santa Fe, USA. 13–17 March 2005, to appear.
12. Bray, I. (2002). *An Introduction to Requirements Engineering*. Pearson Addison Wesley, 1st edition.
13. Bray, I. (2004). Experiences of teaching Problem Frame based requirements engineering to undergraduates. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 17–20, Edinburgh. IEE. 24 May 2004.
14. Bray, I. and Cox, K. (2003). The Simulator: Another elementary problem frame? In B. Regnell and E. Kamsties, editors, *Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, pages 121–4, Velden, Austria. Essener Informatik Beitrage.
15. Brier, J., Rapanotti, L., and Hall, J. G. (2004). Problem Frames for socio-technical systems: predictability and change. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 21–25, Edinburgh. IEE. 24 May 2004.
16. Choppy, C. and Reggio, G. (1999). Using CASL to specify the requirements: a problem specific approach. In *Proceedings of the 14th International Workshop on Algebraic Development Techniques (WADT'99)*, Bonas, France. 15–18 September 1999.
17. Choppy, C. and Reggio, G. (2004). A UML-Based method for the Commanded Behaviour frame. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 27–34, Edinburgh. IEE. 24 May 2004.
18. Cockburn, A. (2001). *Writing Effective Use Cases*. Addison-Wesley.
19. CoFI (1999). CASL the common algebraic specification language summary, version 1. Technical report, The Common Framework Initiative for Algebraic Specification and Development. http://www.brics.dk/Projects/CoFI/.
20. Cook, S. and Daniels, J. (1994). *Designing Object Systems: Object-Oriented Modeling with Syntropy*. Prentice-Hall.
21. Cox, K. and Phalp, K. (2003). From process model to problem frame - a position paper. In B. Regnell and E. Kamsties, editors, *Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, pages 113–6, Velden, Austria. Essener Informatik Beitrage.
22. Cox, K., Phalp, K., Aurum, A., Bleistein, S., and Verner, J. (2004a). Connecting Role Activity Diagrams to the Problem Frames approach. In *Proceedings of the 9th Australian Workshop on Requirements Engineering (AWRE'04)*, Adelaide. 6–7 December 2004.
23. Cox, K., Phalp, K., Bleistein, S., and Verner, J. (2004b). Deriving requirements from process models via the Problem Frames approach. *Information and Software Technology*. to appear.

24. Delannay, G. (2002). A meta-model of Jackson's Problem Frames. Technical report, University of Namur.
25. Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Object Technology Series. Addison-Wesley.
26. Gamma, E., Johnson, R., Vlissides, J., and Helm, R. (1995). *DesignPatterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
27. GRL (2004). Goal-oriented requirements language. http://www.cs.toronto.edu/km/grl/. Last accessed November, 2004.
28. Gunter, C. A., Gunter, E. L., Jackson, M., and Zave, P. (2000). A reference model for requirements and specifications. *IEEE Software*, **17**(3), 37–43.
29. Haley, C. (2003). Using problem frames with distributed architectures: A case for cardinality on interfaces. In *Proceedings of the 2nd International Workshop from Software Requirements to Architectures (STRAW'03)*, Portland, Oregon. 9 May 2003.
30. Haley, C., Jackson, M. A., Laney, R., and Nuseibeh, B. (2003). An example using problem frames: Analysis of a lighting control system. Technical Report 2003/18, Department of Computing, The Open University.
31. Haley, C., Laney, R., and Nuseibeh, B. (2004). Using problem frames and projections to analyze requirements for distributed systems. In B. Regnell, E. Kamsties, and V. Gervasi, editors, *Proceedings of the 10th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'04)*, Riga, Latvia.
32. Hall, J. G. and Rapanotti, L. (2003). A reference model for requirements *Engineering*. In *Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE'03)*, pages 181–187, Monteray, California. IEEE.
33. Hall, J. G. and Rapanotti, L. (2004). Problem Frames for Socio-Technical Systems. In J. Mate and A. Silva, editors, *Requirements Engineering for Socio-Technical Systems*. Idea Group, Inc.
34. Hall, J. G., Jackson, M. A., Laney, R., Nuseibeh, B., and Rapanotti, L. (2002). Relating software requirements and architectures using problem frames. In *Proceedings of the 10th IEEE Joint International Conference of Requirements Engineering (RE'02)*, pages 137–144, Essen, Germany. IEEE Computer Society Press.
35. Hall, J. G., Rapanotti, L., and Jackson, M. A. (2005). Problem frame semantics for software development. *Journal of Software and Systems Modelling*. To appear.
36. Jackson, D. (2001a). Micromodels of software: Lightweight modelling and analysis with alloy. Software Design Group MIT Lab for Computer Science. http://alloy.mit.edu/reference-manual.pdf.
37. Jackson, D. and Jackson, M. A. (1996). Problem decomposition for reuse. *Software Engineering Journal*, **11**(1), 19–30.
38. Jackson, M. A. (1975). *Principles of Program Design*. Academic Press.
39. Jackson, M. A. (1983). *System Development*. Prentice-Hall.
40. Jackson, M. A. (1991). Description is our business. In *Proceedings of the VDM Conference*, Nordwijkerhout. 21–25 October 1991.
41. Jackson, M. A. (1994a). Problems, methods and specialisation. *IEEE Softw.*, **11**(6), 57–62.
42. Jackson, M. A. (1994b). Problems, methods and specialization. *Software Engineering Journal*, **9**(6), 249–255.

43. Jackson, M. A. (1995a). Problems and requirements. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering (RE'95)*, pages 2–8, York, UK. ACM Press. 27–29 March 1995.
44. Jackson, M. A. (1995b). *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley Publishing Company.
45. Jackson, M. A. (1995c). The world and the machine. In *Proceedings of the 17th International Conference on Software Engineering (ICSE'95)*, pages 282–292, Seatle, USA. IEEE/ACM. 23–30 April 1995.
46. Jackson, M. A. (1996). Connecting viewpoints by shared phenomena. In *Viewpoints'96: International Workshop on Multiple Perspectives in Software Development*, pages 180–183, San Francisco, USA. ACM. 14–15 October 1996.
47. Jackson, M. A. (1997a). The meaning of requirements. *Annals of Software Engineering*, **3**, 5–21.
48. Jackson, M. A. (1997b). Problem complexity. In *Proceedings of the 3rd International Conference on Engineering of Complex Computer Systems (ICECSS'97)*, pages 239–248, Lake Como, Italy. IEEE Comp Soc Press.
49. Jackson, M. A. (1998a). Defining a discipline of description. *IEEE Software*, **15**(5), 14–17.
50. Jackson, M. A. (1998b). A discipline of description. *Proceedings of CEIRE'98, Special Issue of Requirements Engineering Journal*, **3**(2), 73–78.
51. Jackson, M. A. (1998c). Formal methods and traditional engineering. *Journal of Systems and Software, special issue on Formal Methods Technology Transfer*, **40**(3), 191–194.
52. Jackson, M. A. (1998d). Will there ever be software engineering? *IEEE Software*, pages 36–39.
53. Jackson, M. A. (1999). Problem analysis using small problem frames. *Proceedings of WOFACS'98, Special Issue of the South African Computer Journal*, **22**, 47–60.
54. Jackson, M. A. (2000a). Problem analysis and structure. In *Proceedings of NATO Summer School on Engineering Theories of Software Construction*, Marktoberdorf, Munich. 25 July–6 August 2000.
55. Jackson, M. A. (2000b). The real world; in millennial perspectives in computer science. In J. Davies, B. Roscoe, and J. Woodcock, editors, *Proceedings of the 1999 Oxford-Microsoft symposium in honour of Sir Antony Hoare*, pages 157–173. Palgrave.
56. Jackson, M. A. (2001b). *Problem Frames: Analyzing and Structuring Software Development Problem*. Addison-Wesley Publishing Company, 1st edition.
57. Jackson, M. A. (2001c). Problem structures and solution structures: a position paper. In *Proceedings of the International Workshop on Requirements Engineering (IWRE'01)*, Imperial College, London. 25 April 2001.
58. Jackson, M. A. (2002). Some basic tenets of description. *Software and Systems Modeling*, **1**(1), 5–9.
59. Jackson, M. A. (2003). Why software writing is difficult and will remain so. *Information Processing Letters*, **88**(1-2), 13–15.
60. Jackson, M. A. and Zave, P. (1993). Domain descriptions. In *Proceedings of the 1st IEEE International Symposium on Requirements Engineering*, pages 56–64. IEEE CS

Press.
61. Jeary, S. and Phalp, K. (2004). On the applicability of problem frames to web-based business applications. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 35–38, Edinburgh. IEE. 24 May 2004.
62. Konrad, S. and Cheng, B. H. C. (2002). Requirements patterns for embedded systems. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 127–136. IEEE Computer Society.
63. Kovitz, B. (1999). *Practical Software Requirements; A Manual of Content and Style*. Manning.
64. Kruchten, P. (1999). *The Rational Unified Process: An Introduction*. Object Technology Series. Addison-Wesley.
65. Laney, R., Barroca, L., Jackson, M. A., and Nuseibeh, B. (2004). Composing requirements using problem frames. In *Proceedings of the 12th Int. Conf. on Requirements Engineering (RE'04)*, pages 113–122, Kyoto, Japan. IEEE Computer Society Press.
66. Lavazza, L. and Del Bianco, V. (2004). A UML-based approach for representing problem frames. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 39–48, Edinburgh. IEE. 24 May 2004.
67. Lavazza, L., Morasca, S., and Morzenti, A. (2004). A dual language approach to the development of TimeCritical systems with UML. In *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS)*, Barcelona, Spain. March 2004.
68. Li, Z., Hall, J. G., and Rapanotti, L. (2004). Reasoning about decomposing and recomposing problem frames developments: a case study. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 49–53, Edinburgh. IEE. 24 May 2004.
69. Lin, L., Nuseibeh, B., Ince, D., Jackson, M. A., and Moffett, J. (2003). Introducing abuse frames for analysing security requirements. In *Proceedings of the 11th International Conference on Requirements Engineering (RE'03)*, pages 371–2, Monterey, California.
70. Lin, L., Nuseibeh, B., Ince, D., and Jackson, M. A. (2004). Using abuse frames to bound the scope of security problems. In *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, pages 354–355. IEEE Computer Society.
71. Lin, Y. (2004). Applying problem frames to modeling 'abstraction' concepts. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 55–59, Edinburgh. IEE. 24 May 2004.
72. Nelson, M., Cowan, D., and Alencar, P. (2001). Geographic problem frames. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01)*, pages 306–7, Toronto, Canada.
73. Nelson, M., Nelson, T., Alencar, P., and Cowan, C. (2004). Exploring problem frame concerns using formal analysis. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem*

Frames, pages 61–68, Edinburgh. IEE. 24 May 2004.
74. Nuseibeh, B. (2001). Weaving together requirements and architectures. *IEEE Computer*, **34**(3), 115–117.
75. Ourusoff, N. (2004). Towards a CASE tool for Jackson's JSP, JSD and Problem Frames. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 69–73, Edinburgh. IEE. 24 May 2004.
76. Phalp, K. and Cox, K. (2000). Picking the right problem frame - an empirical study. *Empirical Software Engineering Journal*, **5**(3), 215–228.
77. Rapanotti, L., Hall, J. G., Jackson, M. A., and Nuseibeh, B. (2004). Architecture-driven problem decomposition. In *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, pages 73–82, Kyoto, Japan. IEEE.
78. Selic, B., Gullekson, G., and Ward, P. (1994). *Real-Time Object-Oriented Modeling*. Wiley.
79. Stacey, R. D. (1991). *Dynamic Strategic Management for the 1990s*. Kogan Page.
80. Sutcliffe, A. (2002). *The Domain Theory: Patterns for Knowledge and Software Reuse*. Lawrence Erlbaum Associates.
81. Sutcliffe, A. and Maiden, N. (1998). The domain theory for requirements engineering. *IEEE Transactions on Software Engineering*, **24**(3), 174–196.
82. Taylor, P. (2000). Problem frames and object-oriented software architecture. In *Proceedings of 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-37'00)*, pages 70–81, Sydney, Australia. IEEE. 20–23 November 2000.
83. Tomayko, J. (2001). Adapting problem frames to extreme programming. In *Proceedings of the XP Universe Conference*, Raleigh, NC.
84. van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE2001)*, pages 249–263, Toronto. 27–31 August 2001.
85. Vincenti, W. G. (1990). *What Engineers Know and how they know it: Analytical studies from Aeronautical History*. The Johns Hopkins University Press.
86. Weiringa, R., Gordijn, J., and van Eck, P. (2004). Value framing: A prelude to software problem framing. In K. Cox, J. G. Hall, and L. Rapanotti, editors, *Proceedings of the 1st International Workshop on Applications and Advances of Problem Frames*, pages 75–84, Edinburgh. IEE. 24 May 2004.
87. Wieringa, R. J. (2000). The declarative problem frame: Designing systems that create and use norms. In *Proceedings of the 10th International Workshop on Software Specification and Design*, pages 75–85, San Diego, USA. IEEE Computer Society. 5–7 November 2000.
88. Yu, E. S. (1993). Modeling organizations for information systems requirements engineering. In *Proceedings 1st IEEE International Symposium on Requirements Engineering*, pages 34–41.
89. Zave, P. and Jackson, M. A. (1997). Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, **6**(1).