ISSN 1744-1986

***Technical Report N° 2004/24***

# *An Example of Domain Decomposition through Application of the Problem Frames Approach to a Complex Problem*

***Jon G. Hall,***
***Lucia Rapanotti,***
***Karl Cox,***
***Steven Bleistein,***
***June Verner***

*14th September 2004*

---

***Department of Computing***
**Faculty of Mathematics and Computing**
**The Open University**
**Walton Hall,**
**Milton Keynes**
**MK7 6AA**
**United Kingdom**

***http://computing.open.ac.uk***

The Open University

# An Example of Domain Decomposition through Application of the Problem Frames Approach to a Complex Problem

Jon G. Hall[1], Lucia Rapanotti[1], Karl Cox[2], Steven Bleistein[2], June Verner[2]
[1]Computing Department, The Open University, UK
{J.G.Hall, L.Rapanotti}@open.ac.uk
[2]Empirical Software Engineering, National ICT Australia Ltd., Sydney, Australia
{karl.cox, steven.bleistein, june.verner}@nicta.com.au

## Abstract

*It is critical to decompose problem context of real world problems in order to understand requirements and specifications for IT systems. However, this is far from trivial. In this paper, we present an example problem domain contextual decomposition that provides a means of ensuring high-level requirements, such as business requirements, are met by lower-level requirements that are closer to the software to be delivered. We use Jackson's Problem Frames approach to describe organisational context, and apply Architectural Frames (AFrames) to simultaneously decompose and simplify the problem context and hence requirements. Through AFrame application, and problem decomposition and projection, we show how socio-technical elements can also be described in our solution. We validate our approach through a case study taken from the literature of Seven-Eleven Japan.*

## 1. Introduction

Ideally, the software process has two components: the part that ends with the identification of the requirements that a system should meet – requirements engineering – and the development of software from those requirements – software development. A perennial problem in software engineering is the linking of these two components, i.e., in making the move from the discovery of requirements to the development of software. As identified elsewhere [1] there are a number of factors that prevent easy linkage.

1. Requirements will change during the lifetime of a software development project; this makes the process much more complex than it would be if there was a simple linkage.
2. Requirements engineering is typically constrained by domains that are very far from the technology used to develop the software. In the words of [2] requirements are often very far from the machine. Some way must be found to translate them so that they will be able to be applied at a technological level; however, for consistency, we must always be able to trace between the various levels of requirements expression.
3. Requirements engineering provides choices that the form of solutions may take and which need to be resolved both to explicate the solution, and to determine lower-level requirements. This point is related to point 2 above.
4. There is often a need to consider the design or redesign of other, non-computing components of a solution; for example, solutions to organisational problems often involve changes in the roles and competences of employees. In software engineering terms, the solution has both human and technological parts: the solution is socio-technical in nature. Treatment of the socio-technical factors better links requirements engineering and software development to provide an optimal solution.

In this paper we show how an extended problem frames framework can be used to address the second, third and fourth problems of software engineering as described above. We do not describe here work that extends the problem frames framework to address issue (1). We describe in detail, in section 2 below, our proposed extensions to the problem frames framework; these extensions remain faithful to the original foundations of this framework.

There are many published studies of problem frames. As noted in section 2 and also as reported in [3] these studies, with the exception of [4-6], either describe simple theoretical case studies, or small, well understood, classical software engineering exemplars. The contribution of this paper is to show how problem frames can be usefully applied in describing complex real-world problems. We exercise the extensions we have defined by showing, through a case study, how IT in support of an organisation's strategy for supply chain management may be developed. The case study is based on Seven-Eleven Japan's e-business system (for example, see [4-6]).

The paper is organized as follows. Section 2 provides background information on Problem Frames. Section 3 discusses Architectural Frames. Section 4 presents the

case example. Section 5 provides an evaluation and section 6 presents some conclusions.

## 2. Problem Frames

Problem Frames capture and classify software development problems [2, 7]. A Problem Frame structures the analysis of the problem within its problem space. It describes what is in the Real World and how the software is intended to change or guarantee real-world conditions in accordance with the requirements. The Problem Frames approach uses an understanding of a problem class to allow the "problem owner" with his or her domain knowledge to drive the requirements engineering process by selecting the appropriate development method to solve the problem type. As such Problem Frames are akin to design patterns [8] in that they provide a recognised problem pattern that has a known solution method. However, Problem Frames differ from design patterns in that they represent real world phenomena, as opposed to software solution phenomena.

Fig. 1 illustrates some essential elements of the original Problem Frames model. The Real World Problem Context provides us with information about the structure, processes and tasks that are already true of the problem domain. The Requirement states which properties we wish to be true in a software solution to be built, *the Machine* which will work within its real world context. The connection between the real world problem context and the machine is represented by the shared phenomena at the boundary between the problem and the machine. Shared phenomena can be data, events, commands and states. For most software problems there will be a number of domains of interest. The requirement can connect to a domain of interest in two ways. An arrowhead indicates that the domain is constrained by the requirement. That is, the machine must guarantee that the state or behaviour of that domain satisfies the requirement. A requirement reference, with no arrowhead from requirement to domain, indicates that the requirement refers to some phenomena in that domain [2].


**Fig. 1. Elements of Problem Frames**

Jackson describes two moods, in the grammatical sense, to represent the problem context and the requirement [2]. Indicative mood represents everything in the problem context that is given and will remain unaffected by the solution. Optative mood represents the way we would like everything to be, given the construction of the solution. This is the requirement. A requirement can change the properties, states and behaviours of domains of interest but cannot affect indicative properties. In this way, we can get three descriptions: the problem context; the requirement, i.e., all that we would like the solution to bring about in the problem context; and the specification, which describes the shared phenomena between the problem context and the solution that achieves the requirement.

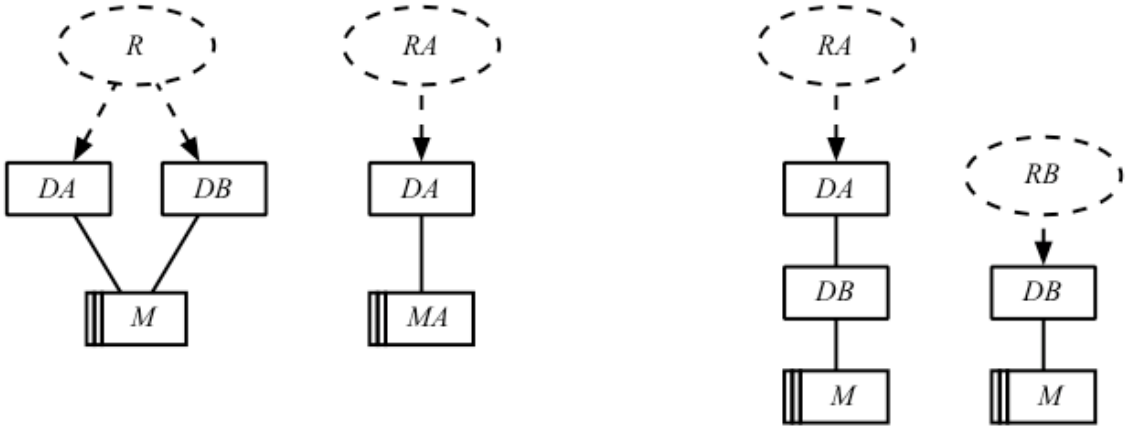
Relating problem context, requirement and solution specification is a correctness argument that is constructed during the development of a solution. The correctness argument ensures that, to some level of certainty, when placed in the problem context, the specification will work to ensure the requirements.

A problem diagram, containing the same elements as in Fig. 1, describes a software problem showing the problem parts consisting of problem context and the requirement. Problem Frames are derived through decomposition of problem diagrams. Even though the software/hardware system may consist of multiple devices or computers, for the purpose of a problem diagram these are represented as a single machine. Decomposing problem diagrams reveals an increasing level of detail, including separate distinct machines.

The original Problem Frames framework uses two problem transformations, illustrated in figure 2:

a. A divide and conquer approach to solve problems: a large problem diagram is projected onto simpler sub-problems, under a correctness preserving recomposition constraint to ensure that sub-problem solutions can be recomposed in non-antagonistic ways.
b. The progression of problems transformation in which high-level requirements – those far from the solution – are refined to levels of abstraction closer to the machine. Detail of the induced requirement refinement is not given by Jackson in [2]. However, it is addressed by Bleistein et al in [4-6]. Note that we do not address this concern here due to page length limitations.

Each of these transformations preserves the problem to be solved – more plainly, any solution that can be arrived at through application of the transformations to a problem will remain a solution to that problem. Although complete for solving problems, projection and progression of problems transformations are not, by themselves, complete for modern software development. Neither transformation allows the construction of partial solutions, such as are the basis of the architectural business cycle (ABC) [9], and that lead to negotiation with the customer about trade-offs of non-functional or quality requirements. ABC would require an exploration of the high-level *structure* of the machine, i.e., the introduction of new domains to represent possible solution components that have no place in a customer's view of a problem.




a) b)

**Figure 2 –Transformations: a) Projection; b) Problem progression (adapted from [2])**

The original problem frame framework works with software problems. To this end, it contains a single target for design – the machine domain – the decorated rectangle of Figure 1 that 'stands' for the solution.

In the extended problem frame framework of this paper, we work with three problem-preserving transformations. Alongside projection and problem progression we add AFrame expansion and we expand projection to include guided sub-problem discovery, as introduced in [10, 11]. These are detailed in section 3. In addition, we expand the types of targets for design that to which we can refer in a problem diagram. In addition to machine domains, we introduce knowledge domains [12] and organisational domains.

A knowledge domain represents a solution (part) that requires some form of human flexibility – training, for instance, extra competences or knowledge – to be designed as part of the solution. An example that we use in this paper is the training that a store clerk will need to be able to operate a Point of Sales system. An organisational domain represents parts of an organisation that will need to be designed to solve an organisational problem. We show how the business problem of supply chain management has an organisational solution. Organisational solutions typically consist of human resources and machines working together, and we will show how AFrames can be used to transform organisational problems into problems involving machines and people.

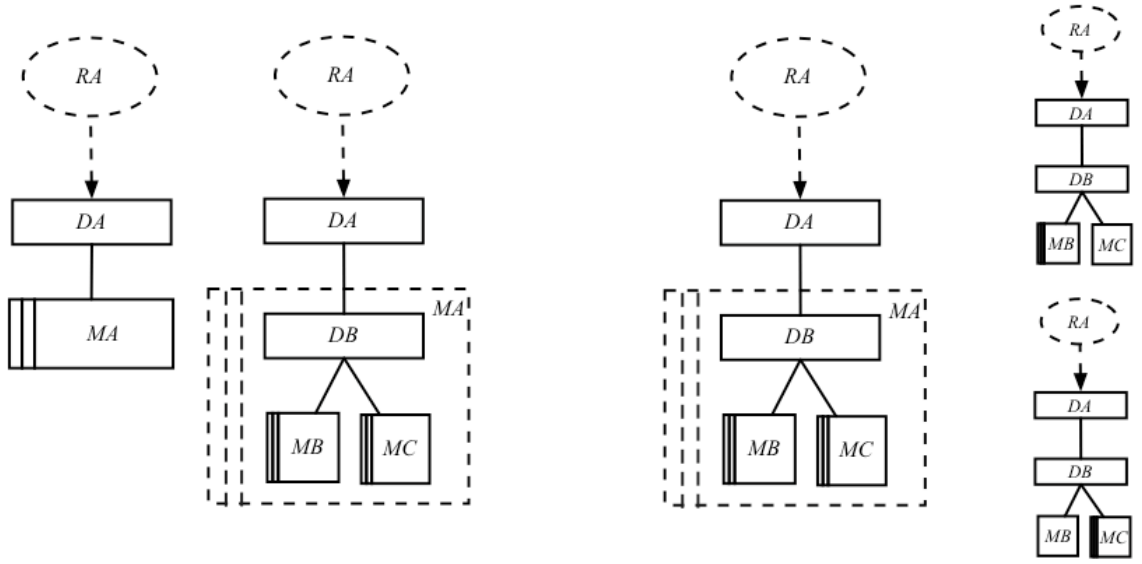
## 3. AFrames and Socio-Technical Solutions

Problem frames on their own are neither completely adequate nor appropriate for addressing concerns of socio-technical and organizational IT systems. Related work on Problem Frames has focussed on identifying what techniques are most useful to eliciting and documenting requirements and specifications once the Problem Frame is known [13, 14] and in exploring the role Problem Frames have with aspects of software architecture [11, 15]. Research has extended and expanded problem frames to address different types of concerns including those for geographic applications [16], system security [17], simulator systems [18], extreme programming [19] and business process modelling [20]. None of this research presents particularly complex examples that require a high degree of decomposition to get to a problem frame i.e. the problems described are very close to the machine or refer to the machine itself.

Architectural Frames (AFrame) guide sub-problem discovery, the third transformation in our extended Problem Frames framework. As such, they form a new element of the Problem Frames framework [11]. The intention of AFrames is to provide a practical tool for sub-problem discovery (and subsequent recomposition of solutions) that allows the Problem Frames practitioner to separate and address in a systematic fashion, the concerns arising from the intertwining of problems and solutions. The rationale behind AFrames is the recognition that known solution structures can be usefully employed to inform problem analysis.

An AFrame captures the combination of a class of problems and a class of solution structures. We focus here on the combination of solutions to a class of problems that arise from observed good practice; other possibilities for guiding solution structures include software architectures, for instance. There are many ways in which good practice is captured in the literature; many international standards, for instance, can be seen as the documentation of good practice. The characterisation of good practice is achieved through decomposition templates, which form an integral part of every AFrame definition. Decomposition templates capture a way of decomposing a problem into sub-problems; they complement classical Problem Frames decomposition by providing guidance and decomposition rules. In Figure 3,

we illustrate AFrame expansion and guided sub-problem discovery.




**a)** **b)**

**Figure 3: a) AFrame expansion and b) Guided Sub-problem discovery**

AFrame expansion works by allowing known (or postulated) structure to be added as solution. Figure 3a shows how the domain MA is restructured through AFrame expansion as a given (solution) domain DB, together with solution component MB and MC that remain to be designed. Such structure often arises with the use of software architectures that, through AFrame expansion, can be chosen for application when known to be appropriate as a solution in a problem that fits the general form of RA and DA. Note that the original machine domain MA remains in the transformed diagram; this occurs because a problem diagram is required to have only a single designable solution domain, and AFrame transformation may introduce any number of such domains – the dotted original reminds us of the original, single design target.

In contrast, guided sub-problem discovery identifies sub-problems in an AFrame expansion that are proper problem diagrams. It does this by producing as many sub-problems from an AFrame expansion as there are designable solution domains therein. In Figure 3b, for instance, MB and MC are each given their 'own' problem diagram in which each is the sole target for design, the other being assumed part of a refined problem context. Solutions to these separate problem diagrams will later be recomposed (under a correctness argument that forms part of the original problem frames projection transformation) to produce the whole solution. In adding domains to the original problem context, guided sub-problem discovery may appear to introduce domains about which nothing is known, and in the worst case this will, indeed, be the case – in Figure 3b, for instance, all we know about domain MC (in the top sub-problem) which forms part of the context of the solution domain MB is that it is named MC. However, the context of guided sub-problem decomposition is AFrame expansion, and it is likely that, either:

- MB and MC are components that can be designed separately; or
- There are further AFrame transformations to be applied in the second sub-problem that will inform the structure of MC in the top sub-problem.

## 4. Example: Seven-Eleven Japan

In this section we will show how knowledge of good practice in the retail domain leads us through an exploration of a possible solution for this problem. Good practice is coded as a sequence of AFrames; it derives from the very successful Seven-Eleven Japan (SEJ) retail convenience store, which we represent as a collection of AFrames, applied in sequence, that – in this case study – lead to simpler problems. We rely on a number of literature sources that describe this case in detail [21-26].

### 4.1. Seven-Eleven Japan as an AFrame

SEJ manages a national network of convenience stores and generates value by leveraging and controlling ownership of information to optimize efficiency across a value chain. SEJ actually owns very few physical assets. The company positions itself in the centre of a value chain that includes manufacturers, distributors, third-party logistics partners, and franchise shops, all of whom are

independently-owned companies, yet all of whose objectives are maximizing throughput of products ultimately sold to franchise shop end-customers. SEJ's macro-level business model includes the participants mentioned above and their shared phenomena in terms of transactional flows of money, information, and products, based on the description of e-business models appearing in [24]. Figure 4 gives a representation of SEJ as an organisational AFrame: it is organisational, as the solution is the application of a business strategy to an organisation.

Both Franchise Store and Head Office are organisational (sub-)domains within the SEJ organisation. Although SEJ does not own the franchise stores, and as such has no direct contact with the customers, the SEJ business model prescribes very precisely how franchise stores should conduct their business. In particular, SEJ bases its strategy for competitive advantage on an extremely high level of competency at anticipating consumer purchases store-by-store, item-by-item, hour-by-hour, and then providing customers with products they want when they want them. SEJ's strategy leverages information technology to accomplish its strategic objectives.

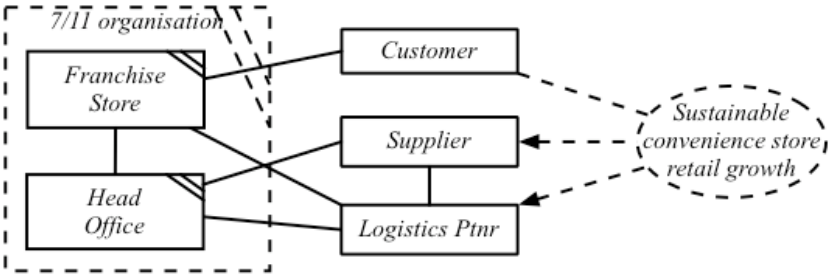
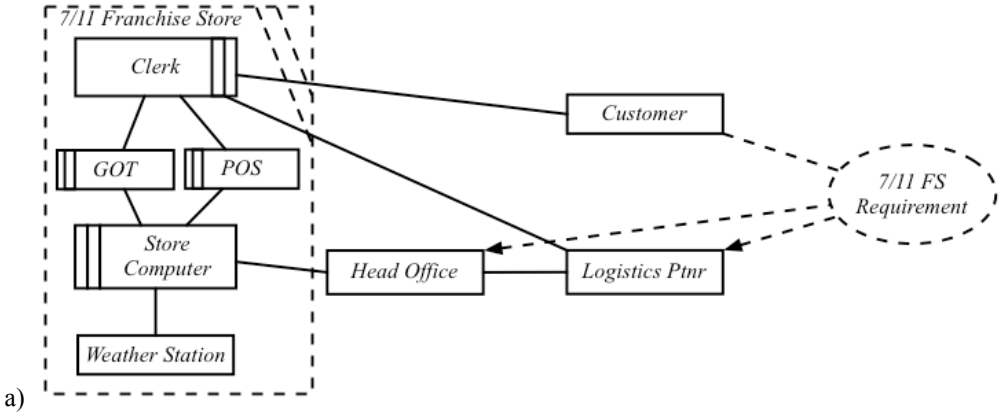


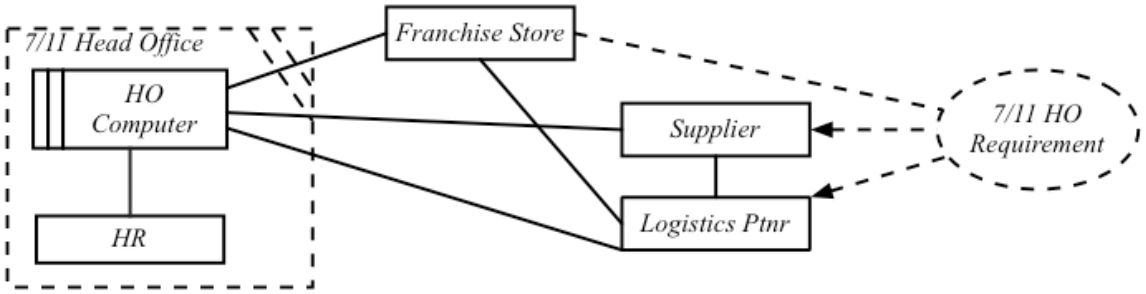

**Figure 4. The SEJ AFrame**

SEJ's ownership of information enables sophisticated supply chain management to reduce inventories, lower costs, and increase sales. SEJ moves information between itself and its partner companies via an ISDN network. To better understand customer demand, SEJ actively gathers and analyses purchasing information in real time, and correlates this with other social and environmental factors, including neighbourhood demographics, planned local events like festivals, and the weather. SEJ then uses a highly acute just-in-time delivery system to meet that demand generating remarkable value.

As one of the main advantages of AFrames is guided decomposition, based on the SEJ business model, the SEJ AFrame of Figure 4 leads to the two organisational (sub)problems of Figure 5, corresponding to the design of a 7/11 Franchise Store (part a) and the 7/11 Head Office (part b). In Figure 5a, POS stands for Point Of Sale, a system that monitors customers' purchases, while GOT for Graphical Order Terminal, a device which allows the Clerk to track and report on sales and shop stock levels; the weather station broadcasts information about weather conditions within a 20km range. Note how the Head Office becomes an (ordinary) given domain in the Franchise Store sub-problem, and the Franchise Store becomes an (ordinary) given domain in the Head Office sub-problem.




a)




b)

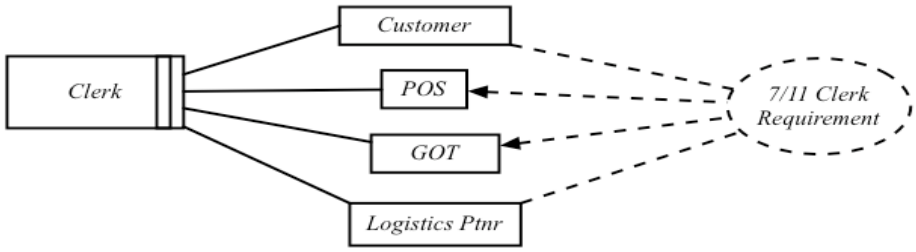
**Figure 5. The two SEJ organisational sub-problems: a) the Franchise Store sub-problem; b) the Head Office sub-problem.**

The Franchise Store sub-problem consists of four targets for design: the Clerk, a knowledge domain, the Store Computer, the Graphical Order Terminal (GOT) and the Point of Sale System (POS). Through sub-problem projection we reach the problem diagrams of Figure 6.

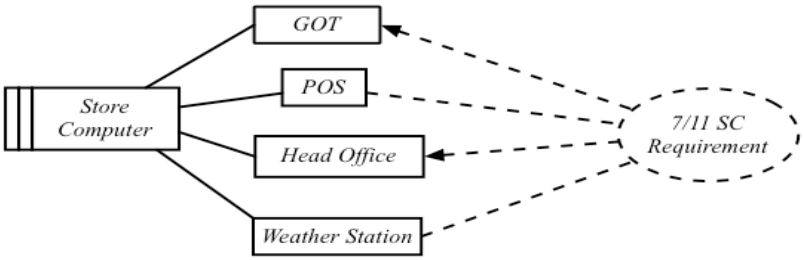
### 4.2 Problem transformations

The relation between the problems described in the previous section is illustrated in Figure 7. The relation between Problem 1 and Problem 2 is through the application of the SEJ AFrame to the original business problem, top left-hand diagram in Figure 8. From this follows a decomposition into the Franchise Store problem (Figure 7, problem 2a, and described in the detail in Figure 6) and the Head Office problem (Figure 7, problem 2b): these provide a structure to the organisational solutions derived from the SEJ business model. Finally, solution structures at this level lead to further problem decomposition of the Franchise Store problem into a number of constituent sub-problems.

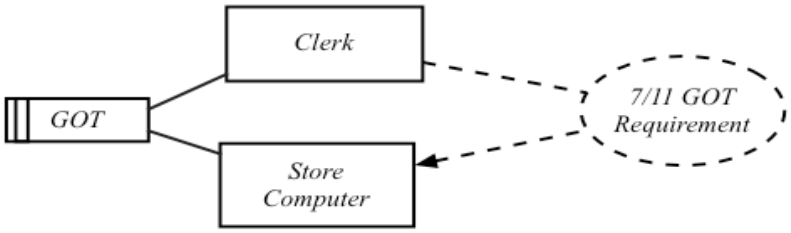
We describe how the two transformation types apply within the SEJ AFrame, from the original problem down to, say, the sub-problem of designing the GOT of the franchise store. In Figure 8, from the original problem an AFrame expansion adds structure to the Organisational domain. A further AFrame expansion is then applied to the Franchise Store, which now becomes the subject of design. At the same time the Head Office domain goes out of scope of design and becomes a given domain in this subproblem. A subsequent problem progression allows us to eliminate the Supplier (still relevant, but far from the machine in this sub-problem) and factor it within the new requirement. The transformation continues to where GOT is now the object of design, so all other solution domains become given in the sub-problem. At the lowest level in Fig. 8, the Clerk operates the GOT, which sends inventory data to the Store Computer to pass onto SEJ in real time. We can, though, decompose this problem diagram further to reach a problem frame.



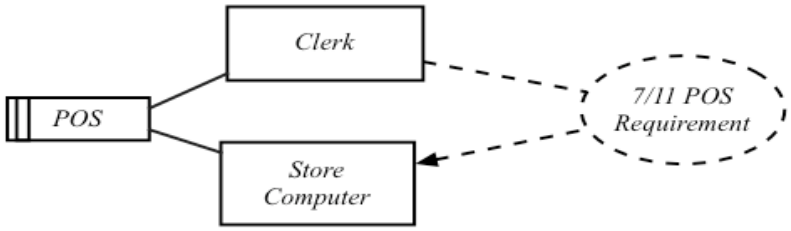

a) the Clerk sub-problem (Problem 2a.i)



b) the Store Computer sub-problem (Problem 2a.ii)




c) the GOT sub-problem (Problem 2a.iii)




d) the POS sub-problem (Problem 2a.iv)

**Figure 6. Further sub-problem decomposition**

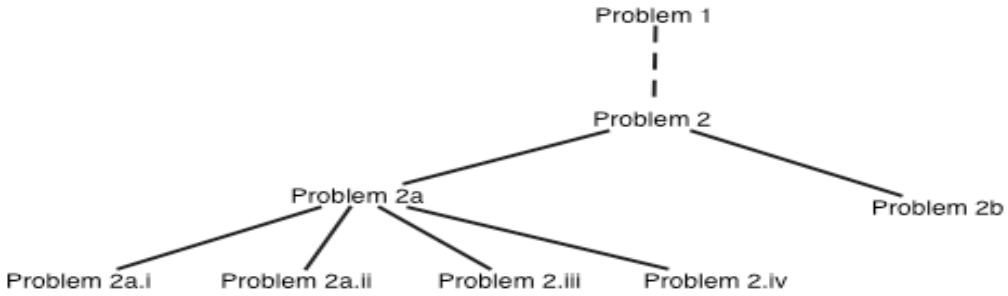



**Figure 7. Relations between problems**

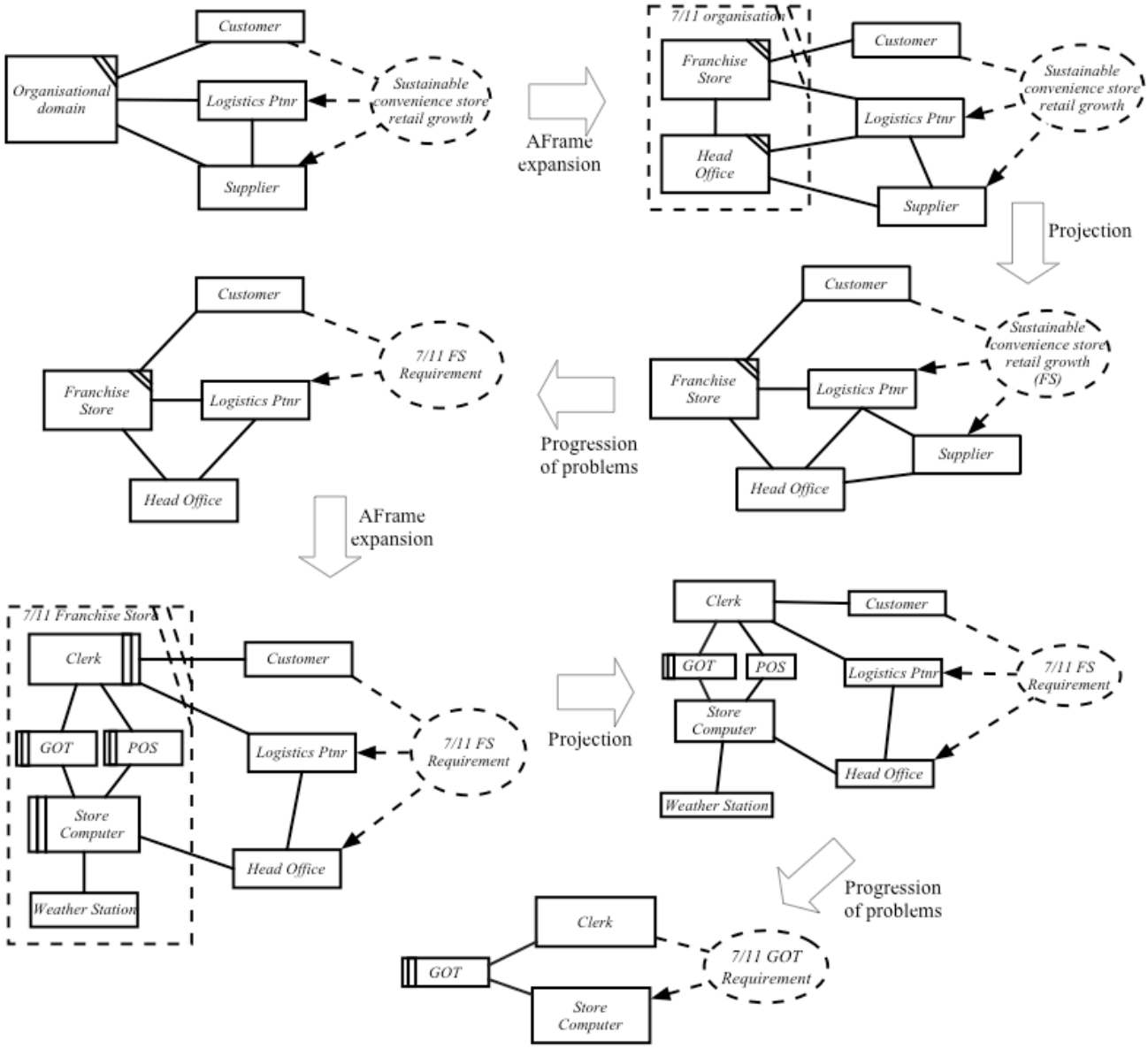



**Figure 8. A complete problem transformation from the original problem diagram to that for the Graphics Order Terminal (GOT)**

Once we have reached the lowest decomposition in Figure 8 we can progress from there to problem frames. Figure 9 provides an example of a variant frame, a Commanded Information Display frame [2]. The requirement stipulates that the Clerk be able to interrogate the GOT about the current inventory available in the store and also to update inventory when a delivery might arrive. This is a variant frame because we found it made little sense to describe an Information Display frame without an operator, in this case the Clerk. Such abstraction in this problem would remove a degree of context that brings understanding in addressing SEJ's requirements.

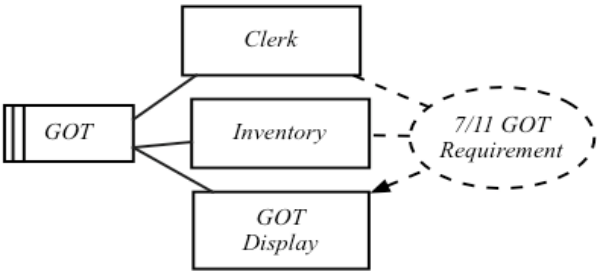



**Fig 9. Commanded Information Display Frame**

## 5. Evaluation

We have expanded the collection of problem transformation tools that exist within the problem frames framework. The new tool is that of guided problem decomposition that allows known solution structures to be used to guide problem exploration, decomposition and subsequent recomposition. Structurally, the original problem frames tools are incomplete in that they do not provide for the use of known solution structures – a technique known to aid problem analysis. Guided AFrame expansion, on the other hand, does allow known solution structures to be used, and so properly extends the problem transformation tools available to the problem frame practitioner.

Whether the new tools are complete in some sense is unknown, but worthy of investigation. The syntactic nature of problem transformation could point to completeness being characterised in that all possible manipulations are possible; indeed, we may postulate that it is complete simply because an AFrame can represent arbitrary solution structures.

There are, however, some validity threats with our work. One threat is the realism or otherwise of the example. We argue that this is a well-documented case, taken from numerous sources in the literature. Hence, our view of the SEJ problem is taken from that of industrialists, including SEJ's CEO, as well as of leading academics. We can thus draw on a number of informed views on aspects of the case. It is also the case that providing examples of our approach through, for example, an ATM, a defacto standard of the requirements engineering research community, a sluice gate or simple traffic lights controller, regularly cited 'exemplars' of the problem frames community, would defeat the point of our paper, that of providing a realistic example of problem domain decomposition using the problem frames approach.

The whole approach to problem description can be compared to standard object-oriented analysis (OOA). However, there are some differences. The most obvious one is that we do not describe concepts or programming concerns in our analysis. OOA does this as a matter of course. We only describe physical, real-world entities that have nothing to do with *how* we design the system. This is difficult to avoid when doing OOA. Though some proponents suggest that OOA really does describe the real world problem domain as it is, they typically provide examples that are about programming. For instance, Kaidl presents what he describes as an entirely problem domain OOA diagram of an ATM, which includes a class entitled Cash Notes with program-oriented attributes of on_hand and dispensed [27] – how does a cash note know whether it is on hand or it has been dispensed? That's something for the ATM itself or the cashier to decide. Thus, Problem Frames distinguishes itself by focussing on real problem domain elements that bear no relationship to programming.

We understand that it makes no sense to 'design' a human user of a machine and we are well aware that employees receive training to use the machines to do their work. However, we assert that it is critical for employees to be aware of exactly what their job entails and, even more significant for us, as engineers, that developers be very aware of the role of employees like Clerks have in the success or failure of the *entire* business. As an example, the Clerk is expected to record each Customer's age and gender at each transaction. This information is used by SEJ to implement its strategy for competitive advantage – providing the Stores customers with the goods they want when they want them. SEJ examines each sales transaction and customer profile on a real-time basis to understand the demographics and customer purchase patterns of the catchment area for each store. If this requirement is not implemented correctly on the machine, or the Clerk does not do his job properly, then the risk of failure is high. Without a detailed domain decomposition of the problem, in terms of domains and requirements, providing traceability from the business strategy to low-level requirements, this potential show-stopper might get overlooked. An example of the ability to overlook critical business requirements is presented by Arlow, who reports that a multi-million dollar business requirement of plane sharing between airlines on booking systems was reduced to a multiplicity asterisk in a UML class diagram [28].

We make a number of claims about requirements in this paper without providing much information about requirements de- and re-composition. Unfortunately, space does not allow anything but a cursory and over-simplified discussion. Since we do not wish to fall foul of over-zealous abstraction, we direct readers to other papers that provide full details on requirements decomposition [4-6].

## 6. Conclusions

We show how the Problem Frames approach can be used to describe complex, real world problem contexts. It is possible to capture high-level requirements and context for business problems and through problem domain decomposition, trace the business problem to the software problem. We do this through Architectural Frame decomposition, combining the ideas of projection and divide-and-conquer of problem domain decomposition. We show how to describe socio-technical domains and highlight the importance of describing this domain explicitly.

As we have shown in this paper, AFrames can represent good practice. The application of good practice does at least three things:

- it shortens development times
- it leads to confidence that a solution is fit-for-purpose; other fit-for-purpose solutions have, presumably, been based on such good practice, by definition;
- it would allow the developer to focus on the analysis of the unique characteristics of the problem – those elements that require bespoke solutions – and to solve the associated sub-problems having properly established the structures into which those bespoke solutions fit.

The risk of missing what may appear to be inconsequential requirements, in fact, may lead to serious financial problems and even outright failure of organisations. The ability to explicitly show the connection from high-level business models to low-level problem frames through problem domain decomposition is critical for businesses like that described in this paper to succeed.

## References


1. Nuseibeh, B., *Weaving Together Requirements and Architectures.* IEEE Computer, 2001. **34**(3): p. 115-117.
2. Jackson, M., *Problem Frames*. 1st ed. 2001: Addison-Wesley Publishing Company. 368.
3. Cox, K., J. Hall, and L. Rapanotti, *1st International Workshop on Advances and Applications of Problem Frames - Summary.* Software Engineering Notes, 2004. **29**(5): p. To Appear.
4. Bleistein, S., K. Cox, and J. Verner. *Problem Frames Approach for e-Business Systems*. in *1st International Workshop on Advances and Applications of Problem Frames*. 2004. Edinburgh: IEE.
5. Bleistein, S., K. Cox, and J. Verner. *Requirements Engineering for e-Business Systems: Integrating Jackson Context Diagrams with Goal Modelling and BPM*. in *APSEC 2004,11th Int. Asia-Pacific Software Engineering Conference*. 2004. Busan, Korea: IEEE Comp. Soc.
6. Bleistein, S., K. Cox, and J. Verner. *Modelling Business Strategy in e-Business Requirements Engineering*. in *eCOMO'2004, 5th International Workshop on Conceptual Modelling Approaches for e-Business*. 2004. Shanghai, China: LNCS.
7. Jackson, M., *Software Requirements and Specifications*. 1995, Harlow: Addison-Wesley.
8. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995: Addison-Wesley.
9. Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*. 2003: Addison Wesley.
10. Hall, J. and L. Rapanotti, *Problem Frames for Socio-Technical Systems*, in *Requirements Engineering for Sociotechnical Systems*, J. Mate, Silva, A., Editor. 2004, Idea Group, Inc.
11. Rapanotti, L., et al. *Architecture-driven Problem Decomposition*. in *The 12th IEEE International Requirements Engineering Conference (RE'04)*. 2004. Kyoto, Japan: IEEE.
12. Brier, J., L. Rapanotti, and J. Hall. *Problem Frames for Socio-Technical Systems: predictability and change*. in *1st International Workshop on Advances and Applications of Problem Frames*. 2004. Edinburgh: IEE.
13. Bray, I., *An Introduction to Requirements Engineering*. 1 ed. 2002: Pearson Addison Wesley.
14. Kovitz, B., *Practical Software Requirements; A Manual of Content and Style*. 1999: Manning.
15. Hall, J., Jackson, M., Laney, R., Nuseibeh, B., Rapanotti, L. *Relating Software Requirements and Architectures using Problem Frames*. in *RE'02*. 2002. Essen, Germany: IEEE Computer Society Press.
16. Nelson, M., D. Cowan, and P. Alencar. *Geographic Problem Frames*. in *Symposium on Requirements Engineering*. 2001. Toronto, Canada.
17. Lin, L., et al., *Analysing Security Threats and Vulnerabilities Using Abuse Frames*. 2003, Open University Computing Dept.
18. Bray, I. and K. Cox. *The Simulator: Another Elementary Problem Frame?* in *9th International Workshop on Requirements Engineering: Foundation for Software Quality - REFSQ'03*. 2003. Velden, Austria: Essener Informatik Beitrage.
19. Tomayko, J. *Adapting Problem Frames to Extreme Programming*. in *XP Universe Conference*. 2001. Raleigh, NC.
20. Cox, K. and K. Phalp. *From Process Model to Problem Frame*. in *9th International Workshop on Requirements Engineering: Foundation for Software Quality - REFSQ'03*. 2003. Velden, Austria: Essener Informatik Beitrage.
21. Bensaou, M., *Seven-Eleven Japan: Managing a Networked Organization*. 1997, INSEAD Euro-Asia Centre.
22. Rapp, W.V., *Retailing: Ito-Yokado Seven-Eleven Japan*, in *Information technology strategies : how leading firms use IT to gain an advantage*. 2002, Oxford University Press: New York. p. 163-186.
23. Whang, S., et al., *Seven Eleven Japan (GS18)*. 1997, Stanford University Graduate School of Business.
24. Weill, P. and M. Vitale, *Place to Space: Moving to eBusiness Models*. 2001, Boston: Harvard Business School Publishing Corporation.
25. Kunitomo, R., *Seven-Eleven is Revolutionising Grocery Distribution in Japan.* Long Range Planning, 1997. **30**(6): p. 887-89.
26. Makino, N. and T. Suzuki, *Convenience Stores and the Information Revolution.* Japan Echo, 1997(Spring): p. 44-9.
27. Kaindl, H., *Is object-oriented requirements engineering of interest?* Requirements Engineering Journal, 2004. **to appear**.
28. Arlow, J., *Use Cases, UML, Visual Modelling, and the Trivialisation of Business Requirements.* Requirements Engineering Journal, 1998. **3**(2): p. 150-2.