Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 79 (2006) 273–289
The Journal of Systems and Software
www.elsevier.com/locate/jss

# Comparing requirements analysis methods for developing reusable component libraries

Alistair Sutcliffe [a,*], George Papamargaritis [b], Liping Zhao [a]

[a] Centre for HCI Design, School of Informatics, University of Manchester, P.O. Box 88, Manchester M60 1QD, UK
[b] BT Exact Technology, Orion Building MLB1/pp12, BT Adastral Park, Martlesham Heath, Ipswich IP5 3RE, UK

Received 5 May 2005; accepted 23 June 2005
Available online 15 August 2005

**Abstract**

Two approaches to requirements modelling are compared—the Domain Theory [Sutcliffe, A.G., 2002. The Domain Theory: Patterns for Knowledge and Software Reuse. Lawrence Erlbaum Associates, Mahwah, NJ.] and Problem Frames [Jackson, M., 2001. Problem Frames: Analysing and Structuring Software Development Problems, Pearson Education, Harlow.]—as a means of domain analysis for creating a reusable library of software components for constructing telemedicine applications. Experience of applying each approach as a domain analysis method to specify abstract components (object system models and Problem Frames) is reported. The two approaches produced detailed specifications although at different levels of abstraction: problems frames were better for monitoring, updating and data integrity requirements whereas the Domain Theory proved more useful for task support and user interface requirements. The lessons learned from using model-based approaches to requirements specification, and their merits for creating consistent specifications for reuse libraries, are discussed.


*Keywords:* Requirements specification; Reuse; Domain analysis; Generic models; Problem frames

## 1. Introduction

Few methods exist for requirements analysis and specification of reusable software, in spite of over 20 years' interest in software reuse. The current practice of developing application frameworks (Fayad and Johnson, 2000), product lines, or families of reusable software components is frequently not specified, so development of new reuse libraries proceeds on an ad hoc basis. Even when processes for developing reusable components are given (e.g., Weiss and Lai, 1999; Clements and Northrop, 2001) these supply little guidance about the level of abstraction to adopt when specifying reusable components. Domain analysis and engineering analysis methods such as FODA (Simos and Anthony, 1998; Vici et al., 1998) tend to be extensions of "vanilla" systems analysis and approach requirements specification simply as an exhaustive exercise in identifying and cataloguing functions within a particular domain. Little advice is given to help the software engineer determine the granularity or abstraction of components; furthermore, the scope of a domain is left to the individual's experience.

The Domain Theory (Sutcliffe and Maiden, 1998; Sutcliffe, 2000, 2002) proposed a more systematic approach to deriving requirements specifications for reuse. It provided a library of generic models and a design for reuse process that have been applied to creating reusable libraries of components for information retrieval applications (Sutcliffe and Carroll, 1999; Sutcliffe and Dimitrova, 1999; Sutcliffe and Ennis, 2000). However,

* Corresponding author. Tel.: +44 0 161 200 3315/306 3315; fax: +44 0 161 306 3324.
*E-mail addresses:* a.g.sutcliffe@manchester.ac.uk (A. Sutcliffe), george.papamargaritis@bt.com (G. Papamargaritis).

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.06.027

the process relied on expert knowledge of the application area and the Domain Theory models to identify many components. One motivation for this paper is to subject the Domain Theory to further testing while also developing an improved method of domain analysis and component identification.

Problem Frames (Jackson, 2001) provide another set of abstract models by which reusable specifications can be developed. However, there are few reports of the application of Problem Frames in practice (Bray, 2002), apart from a recent case study (Hall et al., 2002). Problem Frames share a concern with the Domain Theory over the nature of dependencies between the real world and the machine which is to be designed. Tracing dependency may therefore provide a good starting point for an analytic process by which components might be discovered. A second motivation for this research was to compare Jackson's Problem Frames and concept of investigating the boundaries between the environment and designed components with the Domain Theory requirements specification process.

The paper follows a case study application of the Domain Theory and Problem Frames in a commercial project to create a reuse library in the CSCW (computer supported collaborative work) area. The paper is structured in four sections. First the Domain Theory and its analytic process are described, followed by an introduction to the case study domain. This is followed by a report of the case study experience of applying the Domain Theory process and Problem Frames, and the resulting reuse library. Problem Frames and their application are then described. The second author carried out the case study as the user of the Domain Theory, while the first author was the user of Problem Frames, so the exercise carried out a limited test of the comprehensibility and utility of both theories with novice users. The paper concludes with a section on the lessons learned and a discussion of the research contributions.

### 1.1. Related work

Domain analysis methods are the conventional approach to developing reuse libraries (Weiss and Lai, 1999; Clements and Northrop, 2001) which advocate functional decomposition and modelling product line specification composed of common components and variation points where specialisations will be introduced during design by reuse. Jarzabek et al. (2003) employ use cases to specify semi-generic requirements with customisation scripts for reservation applications in a domain analysis method. The approach to separating specification of stable common parts of a system from the changeable components was first introduced by Jackson in his distinction between entities (stable core processes) and functions in Jackson Systems Development (Jackson, 1983). However, domain analysis methods give little advice on partitioning systems into components, so reuse libraries with different component granularities and boundaries can arise by application of the same method to a single problem by different development teams. To address the consistency of the components problem, several reuse libraries have been posited at the requirements specification level. For instance, conceptual modelling patterns (Fowler, 1997) describe business organisational structures with transaction patterns for accounting applications (sales, money transfer). Taxonomies of Generic Tasks have been described by Zhou and Feiner (1998) who focus on tasks associated with information visualisation such as comparing, evaluating, identifying and classifying objects; while Wehrend and Lewis' (1990) taxonomy covers generic information processing tasks with mappings to suitable visualisations. In knowledge engineering, Generic Tasks have been proposed, such as diagnosis and analysis (Breuker and Van Der Velde, 1994); however, these tasks record problem-solving processes as templates for building expert systems rather than specifying requirements to support human activity.

In requirements engineering, reusable requirements were described by Lam et al. (1997), who proposed a process for generalising requirements by parameterisation, and abstraction of functions and data structures. Generic functional requirements were specified as semi-formal structured narratives with parameters that could be specialised by adding constraints, objects, or details of methods. Tool support for this approach was developed that provided requirements documentation management with traceability between abstract requirements and rationale justifying the need in the domain (Lam et al., 1997). The MRAM method (Mannion et al., 1999) structures requirements for product families using hierarchical decomposition techniques with heuristics for identifying dependencies between requirements that indicate alternative designs and optional requirements for different versions. The ERP (Enterprise Resource Plans) literature describes reuse libraries in some detail, and although methods for their application in reuse are specified, little detail is given of the process by which ERPs were created in the first place (Keller and Teufel, 1998), although some clues to the origins of the SAP library can be found in Scheer (1994). Models at the design level of abstraction have been proposed by Shaw (1991), and architectures and components for constructing generalised software architectures at the system level by Harandi and Lee (1991); and in more detail by Smith (1992), whose reusable components for planning and scheduling algorithms were derived from air traffic control domains. The object-oriented patterns in the GOF library (Gamaa et al., 1995) provide solutions for design problems with collections of objects; for example, interfacing (Façade), handling polymorphism (Factory), encapsulation (Proxy), etc.

Component-based development (Levi and Arsanjani, 2002) describes goal-oriented processes with responsibility allocation of services in conceptual architecture to requirements. This method for domain analysis and composing services was illustrated by creating a framework for web e-commerce domains (e.g., sales, brokerage).

## 2. Overview of the Domain Theory

This section gives a brief description of the Domain Theory. For a more complete description of the theory, the reuse library of models and processes, see Sutcliffe (2002). The models pertinent to the case study in this paper are illustrated when the case study is explained, and given in the book's appendix A. The theory proposes a component-based ontology composed of families of generic models describing transaction-oriented problems, and generic tasks which describe human goal-oriented activity. In object-oriented parlance, the Domain Theory posits generic models organised in a class hierarchy of object collaborations, i.e., it is a collection of objects that transform initial states into a single desired goal state.

The top levels of the tree have been pruned, because models at such high levels of abstraction do not contain sufficient knowledge to be informative. The OSM (Object System Model) hierarchy starts at a meaningful level with nine separate trees, or OSM families: Allocation describes matching- and broking-style problems: Object Containment models transaction processing problems, such as sales order processing, loans, and inventory management; Composition models assembly and manufacture, with its inverse abstraction Decomposition that describes disaggregation; Sensing monitors and detects problems; Construction models manufacturing and processes that change objects; Logistics models movement of goods and messages; Agent Control is for command and control applications; and Simulation represents decision support and other interactive modelling applications.

OSM families can be organised into groups according to their interaction with entities in the system environment. System input arrives via an Object Sensing Model that detects and interprets events in the world. These can range from simple validation routines for data entry to complex event monitoring and interpretation. Object Inventory, Accounting Object Transfer, Object Returning and Object Allocation all model transactions on conceptual entities that correspond to real world entities as they are sold, purchased, hired, serviced, etc. The output from these models is management information. Another group of OSMs, Object Construction, Composition and Decomposition, model manufacturing systems and are usually associated with an Agent Control OSM that organises and plans the operations. The Object Logistics family has a specific purpose in transporting objects through space. Agent Control OSMs directly affect the world through a device, while Object Simulation represents a modelled world to the user on a VDU or other device.

In a similar manner to OSMs, Generalised Tasks are organised in 11 families: Information Acquisition, Analysis/modelling, Diagnosis, Information Retrieval, Validation/testing, Progress tracking, Planning/scheduling, Navigation, Judgement/decision-making, Explanation/advising, and Matching. During the specification

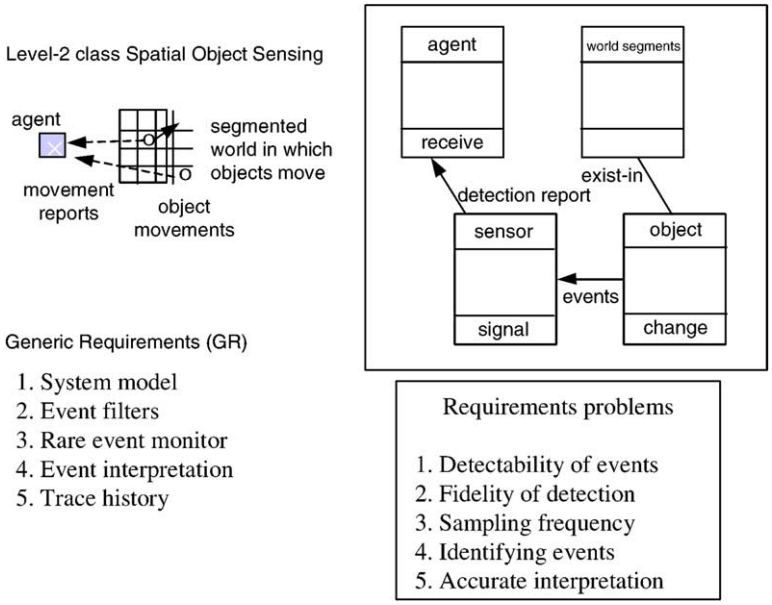


Fig. 1. Object Sensing OSM with associated generic requirements.

process, Generalised Tasks are specialised to more concrete examples; for instance, the generalised model of Diagnosis determines the cause of some malfunction in a system, locating its cause and proposing remedial treatment, whereas instantiations may vary from medical diagnosis of a patient's illness to the technical diagnosis of a fault in a photocopier.

Design rationale and generic requirements are attached to each model to provide reusable knowledge that can be specialised into functional requirements in the new application, and a checklist of issues that may require in-depth analysis. An example of an OSM with attached generic requirements and design rationale is illustrated in Fig. 1.

The requirements problem draws attention to issues that need to be resolved, e.g., *detectability of events*, such as the properties of the hardware sensor device that captures events, the range of physical spectrum it is sensitive to, how it is connected to the external world and the software machine. *Fidelity of detection* draws attention to calibrating sensors so they detect significant events without false alarms, while *sample frequency* explains the problem of how often the device needs to sample, given the anticipated frequency distribution of environmental events. The requirements problems are linked either to generic requirements or design rationale that provides trade-off advice related to the problem. So fidelity of detection is linked to GR2, event filters that propose a functional requirement to screen out unwanted noise; while rare events monitor (GR3) advises on the trade-off of detecting rare events which may be significant or so infrequent that they can be discounted.

## 3. Case study: Component engineering for collaborative telemedicine applications

We illustrate the process of discovering appropriate reusable components by a case study specification of a reuse library for middleware components to support CSCW applications with a telemedicine reuse library built on top of it as an application layer. The case study was part of a research project in BTexaCT Technologies in the component-based service provision area (Rudkin and Smith, 2000; Rudkin et al., 2001). The business motivation was to create a reuse library for CSCW applications, and also to create a reusable application framework for telemedicine. Hence part of the reuse is in the area of middleware network services and part belongs in an end-user application domain. The problem was how to partition this problem space when the components interacted, e.g., end-user interaction in the medical domain may have implications for CSCW support. The monitoring service consisted of a set of diagnostic sensors (hardware components) and methods that interpreted critical patient properties (blood pressure, ECG signals and temperature) supplied from the sensors. Two scenarios of use were employed to motivate the analysis as follows:

### 3.1. Scenario 1. Patient monitoring

Mr. Jones has chronic diabetes and kidney problems. He is in a nursing home waiting for a transplant operation, but his condition needs to be monitored to ensure that he takes insulin to correct fluctuations in his blood sugar, and diuretic drugs for his kidney complaint. At two-hourly intervals he has to place a small blood sample in a blood-sugar analyser. While he is in bed he is connected to a heart-rate monitor that also detects his GSR (Galvanic Skin Response), providing data on his diuretic condition. When the monitoring system detects any deviation from acceptable limits in his blood sugar, blood osmotic pressure or heart rate, the resident nurse is alerted to take appropriate action.

### 3.2. Scenario 2. Collaborative diagnosis

Three consultants are discussing Mr. Jones' case at different locations. They can inspect X-rays of his kidneys and pancreas, MR scans of the same, recordings of ECG electrocardiograms, and his history of blood sugar and osmotic pressure readings over several months. In addition there is a simulation of various drug treatments which allows doctors to test different drug combinations and observe the effects on blood sugar and osmotic pressure over time. The doctors are connected by a video conferencing system and can view each other and/or the medical data; however, they cannot view all of it at once, so they have to agree which items are on a shared display. The consultants review Mr. Jones' history, then view the MR and X-ray data to decide whether a transplant is necessary in the near future. Some disagreement on this point leads them to investigate treatment options. They review a range of drug treatment options before agreeing the appropriate combination to test in the simulation. Once they have viewed the simulation results they record a common diagnosis and treatment plan.

## 4. Domain analysis method

The Domain Theory design process starts by identifying the requirements of a new application in order to match them to Generic Tasks and object models. The process follows a walkthrough-style analysis and starts by identifying any sub-systems in the application. These may be suggested by geographical distribution, ownership by different parts of the organisation, or by locating who operates the system. A dual track approach is used which first decomposes the application to discover

OSMs and Generalised Tasks, and secondly traces dependencies between components following event threads. Combination of the top-down and more bottom-up event analysis identifies the appropriate set of abstract models.

The following questions help to discover OSMs or Generalised Tasks inherent in the system.

- Who or what are the agents that carry out activity and tasks within the system? List the responsibilities of the agents. Agent responsibility relationships point to tasks. Doctors, consultants and nurses are the principal agents in the system, with responsibilities for monitoring the patients' health and diagnosing illness, and deciding on treatments.
- Describe goal-oriented activities and the states they produce once complete. Activities either attain or maintain goal states. Most tasks have achievement goals but monitoring tasks might have maintenance goals. The system goals are to ensure that the patient remains healthy, or at least that the treatment is successful and the patient's condition does not deteriorate. The goals indicate tasks of monitoring, diagnosis and treatment.
- Describe the agents or objects acted on by the system, and the nature of the state change they undergo. The patient agent is changed by the system when treatments are carried out. However, doctors and nurses may act on patients in the real world, so the Agent Control OSM may be appropriate; alternatively, the treatment may in some sense improve patients' health so an Object Repair OSM could also be indicated.
- Describe the life histories of the objects changed by the system and the nature of their change. A decision tree guides object model identification (see Fig. 2). If the composition of the object is changed in any way, it belongs to the Object Construction family; otherwise, any change in ownership is evaluated. If the object life history leads to a permanent change then Object Supply abstraction is indicated; however, if ownership change is temporary then a loan (Object Hiring) is appropriate; but if a relationship is formed that could be a precursor of ownership change, then Object Allocation models reservation, booking and matching applications. Other possibilities for conceptual objects are movement across some physical space in Object Messaging, or being represented to the external world in Object Simulation. The patient agent is physical and the initial state can be assumed to be unsatisfactory, so Object Repair is indicated. However, representations of the patient are also displayed to users (doctors, nurses) in the real world so Object Simulation may be involved. If these conceptual representations are changed in some way the conceptual Object Construction is implicated.
- Define the boundaries of the real world that needs to be modelled in the system. The system will interact with the physical world through sensors and devices;

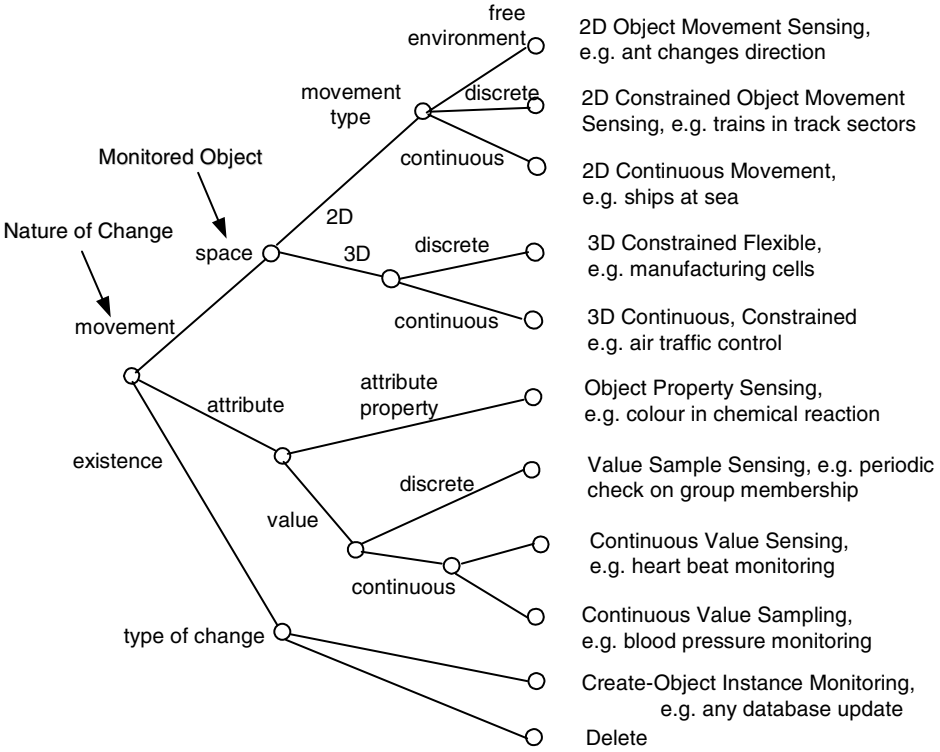


Fig. 2. Decision tree for discovering sub-classes in the Object Sensing family.

however, many information systems simply process conceptual representations of their physical counterparts. The telemedicine domain clearly has a physical presence in the patient; however, it also needs a representation of the patient that is shared between the doctors, and this information has to be transported, so an Object Messaging model is indicated with Object Simulation and conceptual Object Construction because changes will be made to the model.

- Trace events from their origin in the system environment through the system components to their destination, ending in either a data structure update or an event being passed back to the system environment. Event tracing flows from patients to monitoring devices and displays; it also flows between doctors and information displays during consultation.

### 4.1. Domain analysis: Patient monitoring

Patient monitoring involves detecting events, so Object Sensing models are required. Object Repair records the treatment events that help the patient's condition and these events will originate from doctors or nurses. Patients will be assigned to nurses/doctors, hence an Object Allocation model will process input lists of patients and medical staff. Devices connected to the patient to administer drugs or other treatments indicate Agent Control models. Information about the patient has to be transmitted between doctors, indicating Object Messaging.

Input events to most systems are processed by an Object Sensing Model that detects events emanating from the real world. In the telemedicine application there are two possible sources: the patient who is being monitored, and one or more doctors who will be diagnosing problems and initiating treatments. Tracing the Object Sensing tree sub-classes (see Fig. 2), the best fit is Object/Agent property sensing since we wish to detect the properties of the patient such as blood sugar, osmotic pressure, heart rate, etc.

Sensing will be passive as the machine will be responding to events created by the patient. Requirements problems indicated by the Object Property Sensing Model are: fidelity of detecting events to avoid false alarms; sampling rate and frequency of state change in the world, to avoid missing rapid changes; interpretation of events and the context of change; and false events caused by oversensitive detecting devices.

Interpreting events depends on the computer system possessing a model of the observed phenomenon (i.e., the patient); however, the accuracy of event interpretation depends on the fidelity and detail contained in the patient model. This creates further requirements for an accurate and up-to-date model of the patient, which in turn raises the question of how such a model is updated. This indicates Object Sensing to capture the inputs and Object Construction to update the model.

Generic requirements associated with the OSM are imported into the specification and specialised as follows:

1. Reliable event detector and interpreter: a non-functional requirement which needs to be specialised as a quantifiable target, e.g. the event sensor will detect all patient events within 5 ms and report significant changes with less that 0.001% errors.
2. Sampling rate control: user interface features to set the monitoring interval for each variable.
3. Tuning sensing mechanism to eliminate false alarms: controls on the hardware-software interface so device event detection can be customised.
4. Interpreters/filters to eliminate false alarms: ability to change ranges and pattern recognisers, e.g., heart beat irregularities.
5. Log file for history analysis of events: raises the question of how much data is stored and for how long, thus implying a requirement for an archive process.

Once incoming events have been detected and interpreted, the information needs to be distributed to one or more medical staff who are responsible for the patient's care. This implies an Object Messaging model to transport messages to several destinations. In this case the associated requirements problems are: bandwidth and network constraints on throughput; detection of network congestion; and messages that may get lost or corrupted, indicating requirements to counteract message delivery problems. Furthermore, information within the message may be confidential and it certainly needs to be secure, so this adds requirements for secure transmission. The generic requirements list for this part of the system is:

1. Access protocol for message transmission, e.g., token ring, CSMA/CA (Carrier Sense, Multiple Access/Collision Avoidance: the Ethernet protocol).
2. Adaptable route planning to avoid network congestion.
3. Error recovery protocols (acknowledge, resend).
4. Message preparation and formatting (packet protocols).
5. Message assembly.
6. Detecting message loss (packet headers, sequence codes).

Many of these requirements may be beyond our immediate control; however, we can specify a quality of service that the network provider should supply.

Once the information has been transported to a doctor's or nurse's workstation it has to be displayed in a comprehensible manner to the user. In Domain Theory terms this points to an Object Simulation model which represents patient monitoring information in the context of the patient model, i.e., the danger zone for heart rate is interpreted by reference to an individual patient model to account for age, general health, etc. So far the domain analysis has only traced one causal chain of events. The telemedicine domain also needs to support the doctor's activity in analysing patients' problems from monitored data, diagnosing any causes of change and carrying out effective treatment.

### 4.2. Domain analysis: diagnostic support

This analysis uses the Domain Theory's Generalised Task models. A decision tree (see Fig. 3) guides the analyst towards appropriate Generalised Tasks, in this case Diagnostic and Matching to control allocation of users to collaboration sessions and control for access to shared artefacts.

The generalised model of Diagnosis, derived from Rasmussen (1986), contains sub-goals for monitoring and interpreting events, before arriving at a causal diagnosis for those events (see Fig. 4). The Generalised Task specialises in models for causal diagnosis with living things, designed artefacts and complex systems.

Requirements problems include: gathering information for diagnosis when the signs might not be immediately accessible; structuring facts to create a model of the problem; and determining the degree of automation, which is dependent on the degree of predictability in the domain.

Functional allocation in diagnosis may vary from near-complete automation in deterministic domains (e.g., electrical fault finding) to collaborative models in less deterministic domains (e.g., medical diagnosis). In the latter, the computer supplies information on the faulty system, lists of potential faults, signs and symptoms, possible causes, with repair strategies for fault types. Given the poor track record of medical diagnostic expert systems, only partial support for the users' task is advisable by pre-processing information, e.g. screening out hypotheses that do not match known observations. Functional requirements associated with partial automation are:

1. Pre-processors to sort and rank symptoms and observed problems in order of severity, importance, time, etc.
2. Question checklist dialogues to ensure full capture of diagnostic information.
3. System model simulations to locate problems and support diagnostic reasoning.
4. Expert system inference to diagnose possible causes from observed signs.

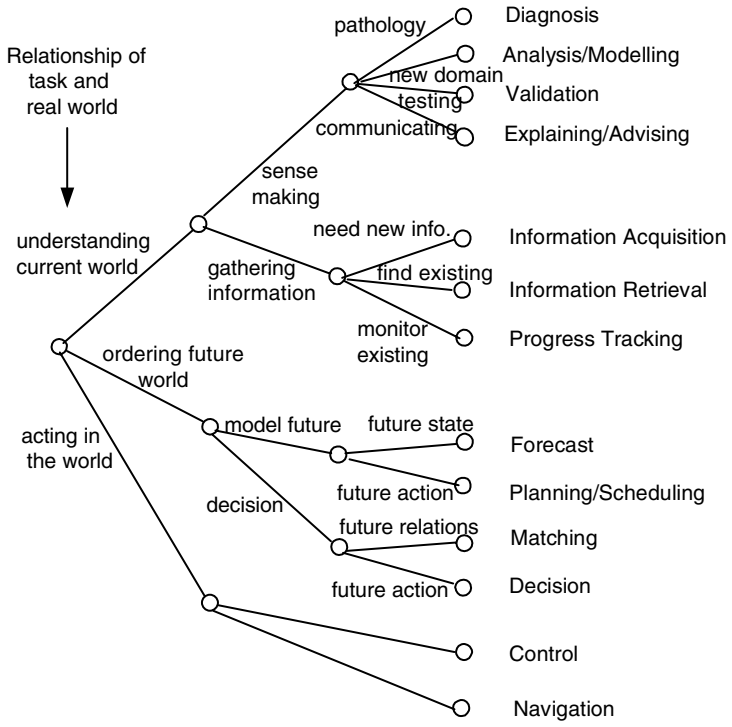


Fig. 3. Decision tree to identify Generalised Tasks.

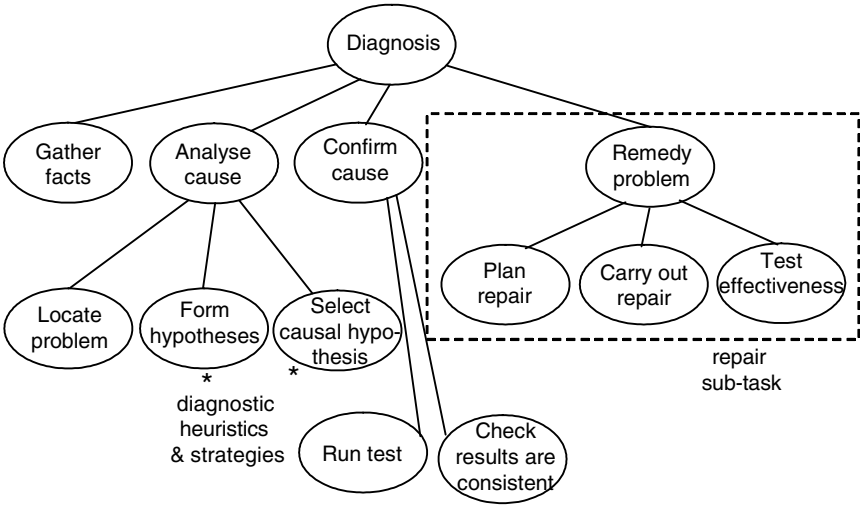


Fig. 4. Goal hierarchy for the Diagnosis task.

5. Interactive microworld system models with "what-if?" testing facilities.
6. Automatic implementation of treatments for identified problems.
7. Suggestions for potential cures/treatments.
8. Facilities for follow-up tests to confirm diagnosis.
9. Guided assistance for treatment/repair strategies and procedures.

The reusable knowledge associated with the task model helps to elaborate the requirements specification as well as indicating architectural components for system models, simulations for check treatments, and advisory sub-systems.

### 4.3. Domain analysis: CSCW layer

The collaboration sub-system needs to deliver messages to several users as well as controlling interaction with shared objects. The goal of delivering messages points towards an Object Messaging OSM. Apart from transmission of ordinary messages (interaction requests, audio/video communication among users), transmission of real-time audio or video content may be necessary for telemedicine purposes (e.g., cardiogram audio samples, endoscope images, etc.). The system must ensure that the appropriate communication quality and reliability is associated with critical tasks (i.e., patient monitoring, examination).

Following the event trace walkthrough, the first need is for the system to register users to ensure only authorised users can gain access. The goal state in this problem is for all the system usage requests to be identified and authenticated, which points to an Object Allocation model. Once users have been registered, the problem of shared object control arises. The assumption is that only one user can interact with a shared object at once, hence a prioritisation process is necessary. An Object Allocation model is employed for this purpose. A user will be allocated control until it is relinquished or the system enforces a timeout to ensure that access is distributed more democratically.

The composite architecture of both sub-systems is illustrated in Fig. 5.

Space precludes exhaustive description of all the generic requirements, so we have selected an example of design rationale for allocation of access control in Object Messaging systems which indicates trade-offs among three options for shared object control (Fig. 6), as follows:

1. System-based: if the client requests the floor, the system assigns it to that client if the floor is free; if not, then the client's request is queued. As soon as the floor becomes free because the previous owner releases it, the system assigns it to the longest-waiting client.
2. Manager-based: a user is given authority for the resource allocation. The user-manager assigns the floor to a participant or releases it on another request. If the manager releases control it returns to a system mode.
3. Token-based: in this case, the user who owns the floor may choose the next owner of the floor. If the current owner releases the floor then the control switches to a system mode. A specialisation of this option is round robin-based token control.

Generally one option would be chosen for implementation; however, to preserve flexibility when constructing a reuse library we intended to provide the reuser with the choice of all three options.

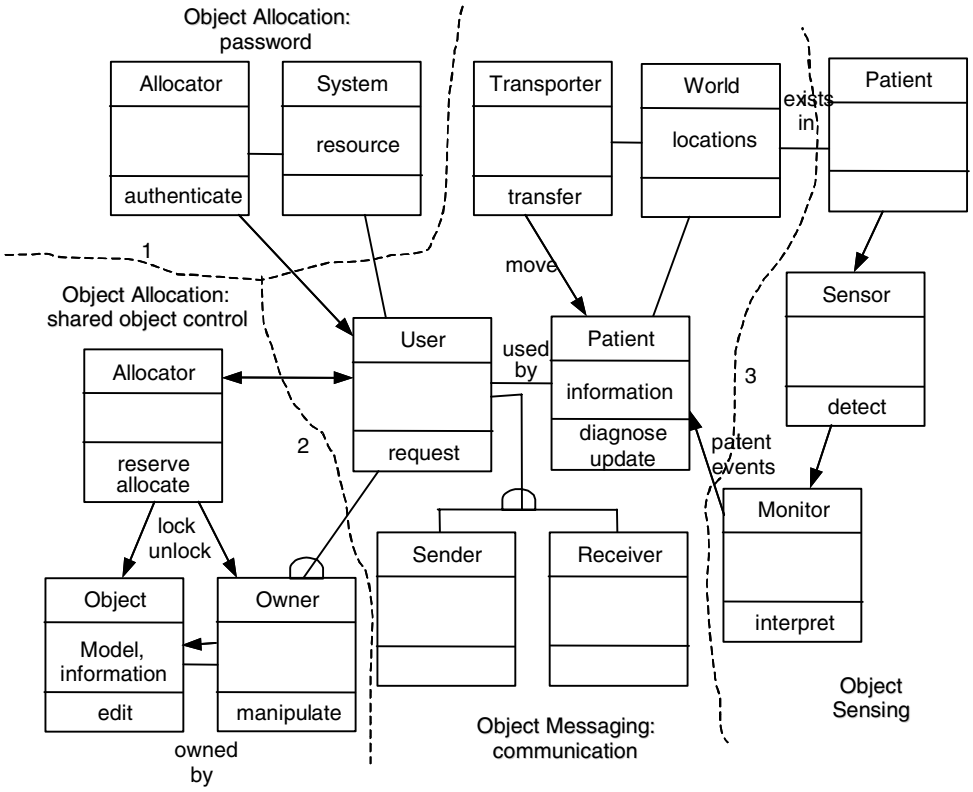


Fig. 5. Composite system architecture after the modelling phase, showing the CSCW and patient monitoring components with interfaces between sub-systems: (1) logon authorisation; (2) shared object allocation; (3) patient monitoring data. The Diagnosis task and Object Construction OSMs have been omitted for simplicity.

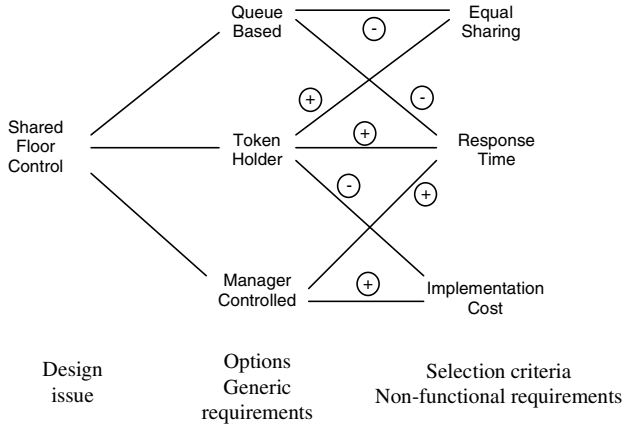


Fig. 6. Design rationale for shared object control. Positive influences are marked with a plus sign, negative or decreasing influences with a minus sign; thus the manager-controlled option has a positive influence on cost and response time.

## 5. Problem Frames

Problem Frames were intended as a formal approach to requirements specification rather than a method for design for reuse so this case study applies Jackson's theory in a new way. Problem Frames (Jackson, 2001) are a set of generic models that describe recurring software engineering problems, although at a higher level of abstraction than the Domain Theory. Jackson draws attention to the connection between the designed system and the real world in which it is or will be embedded. He sees requirements engineering as the process of precise specification of the dependencies between designed systems and the external world. Domains are divided into three types: lexical domains, which are symbol systems such as text, diagrams and models; causal domains that model laws and predictable behaviours in the real world; and biddable domains that exhibit behaviour but have no guarantee of reliability, e.g., human users are biddable in the sense that they may or may not respond. Applications are decomposed using five abstract Problem Frames, described by Jackson (2001) as follows.

### 5.1. Required Behaviour

"There is some part of the physical world whose behaviour is to be controlled so that it satisfies certain conditions. The problem is to build a machine that will impose that control."

There is no direct equivalent of this frame in the Domain Theory, as it refers to compliance with physical laws or conditions that exist in the real world. Instead, the Domain Theory specifies abstract models for

Required Behaviour in problem contexts of control, construction, and transaction processing.

### 5.2. Commanded Behaviour

"There is some part of the physical world whose behaviour is to be controlled in accordance with commands issued by the operator. The problem is to build a machine that will accept the operator's commands and impose control accordingly."

The nearest analogue in the Domain Theory is the Agent Control OSM, which models the behaviour of the controller and controlled agent.

### 5.3. Information Display

"There is some part of the physical world about whose states and behaviour information is continuously needed. The problem is to build a machine that will obtain this information from the world and present it at the required place in the required form."

Mapping to the Domain Theory for this frame in one to many, as the problem is decomposed into Object Sensing which obtains information about the physical world, Object Messaging if the information has to be transported to the required place, and then an Object Simulation OSM to represent it in the required form.

### 5.4. Workpieces

"A tool is needed to allow the user to create and edit a certain class of computer processable text or graphic objects, or similar structures, so that they can be subsequently copied, printed, analysed or used in other ways. The problem is to build a machine that can act as this tool." Workpieces maps to a Conceptual Object Construction OSM.

### 5.5. Transformation

"There are some given computer readable input files whose data must be transformed to give certain required output files. The output data must be in a particular format and it must be derived from the input data according to certain rules. The problem is to build a machine that will produce the required outputs from the inputs."

This Problem Frame maps to several OSM transaction families that require data transformation, e.g., Object Inventory, Object Hiring, Object Allocation.

Each frame encapsulates objects in the real world, objects in the required system, and connections between them that formally model the assumptions made about the system. For example, the Workpiece frame describes the interaction between events and their effect on a model held in the machine. Problem Frames have "frame concerns" which record issues that need to be addressed in a class of problems, in a similar manner to the Domain Theory's requirements problems; in addition, more general concerns are described, such as issues of overrun, initialisation, reliability, identity of individuals, and completeness. Problem Frames also indicate requirements for correct behaviour; for example, in Workpieces one frame concern draws attention to the need to validate that commands are syntactically correct and appropriate for a context, thereby preventing errors such as trying to edit a record before it has been created. A limited number of composite Problem Frames are described: for instance, the Model View Controller generic architecture can be mapped to Workpiece frames for updating the model and the display, Commanded Behaviour for user editing controls, and Required Behaviour for maintaining consistency between the model, edits and the view on the model. Composite frames are associated with concerns such as precedence, consistency between domain descriptions, interference of interactions between frames, and scheduling of machines that share a common domain.

### 5.6. Domain analysis using Problem Frames

Identification of Problem Frames relies on investigating the connection between the real world and designed machine, and inquiring about the dependencies between "domain properties": facts and laws which are known to be true in the real world, and specifications of the designed machine that meet the stated system requirements. Problems are decomposed and mapped to Problem Frames using the following heuristics (Jackson, 2001):

- Identifying the core problem. This is similar to the essential system model (McMenamin and Palmer, 1984) or the entity model in JSD (Jackson, 1983), and represents the objects involved in the process that achieves the major system goal.
- Identifying the ancillary problems that surround the core, especially information processing sub-problems. These map to functions in JSD, also error handling routines.
- Standard decomposition of sub-problems, using combinations of Problem Frames to build composite frames, also investigating the dependency of static or dynamic models on the physical world.
- Identifying concerns and difficulties. Frames can be discovered from frame concerns.
- Different tempi: more than one temporal pace in an application, where one part changes quickly while the other changes over a longer period of time. This suggests two dynamic and static model sub-problems.
- More than two moods. If the frame has more than one indicative mood (domain properties, facts, laws) and more than two optative statements (i.e., require-

ments which state the goal to be achieved and the behaviour specification of the machine that achieves the goals) then there are probably more frames in the application.
- Complex domains or requirements. These usually have several Problem Frames, although no criteria for assessing complexity are stated by Jackson. Subsets of heuristics are given for combinations of problems frames such as Required and Commanded Behaviour.
- Modelling the user: a separate sub-problem.

Space precludes a more complete description of Jackson's theory for which the reader is referred to Jackson (2001).

### 5.7. Problem Frame analysis: patient monitoring

In Problem Frame terms, patient monitoring is a causal domain. The accuracy of event interpretation depends on the fidelity and detail contained in the patient model. An Information Display Problem Frame models monitoring events in the external world, with a Transformation frame interpreting them with reference to the system's internal model of the external entity (i.e., the patient) and representing the events for the users (nurses/doctors). The frame models the change from analogue values to a representation of the patient's blood pressure, EEG, etc. The monitoring and interpreting part of the patient sub-system is modelled by two interconnected Problem Frames: Information Display and Transformation. The Transformation frame concern is the need to interpret data as quickly as possible to make sure it is accurate, because if it is too slow, the monitored world will have changed so the interpretation of the patient's state will be inaccurate. The Domain Theory uses one OSM (Object Sensing) for monitoring and one Generic Task for interpreting. Both theories draw attention to the need to interpret external events by reference to the system model of external entities; however, the Domain Theory requirements problems concern detecting events, whereas Problem Frames tend to focus more on transformation of representations.

The patient monitoring and data display for nurses is modelled by an Information Display frame (see Fig. 7). Implicit in this frame is a patient model sub-system (thresholds/settings) which is composed of a Workpiece Problem Frame for updating the model either from user-initiated edits or automatic updates collected by the monitoring system (illustrated in Fig. 8). Many Information Display frames will be necessary to model monitoring of different sensor devices, e.g., (blood pressure, temperature, ECG, etc.), and data from these frames has to be integrated. Mapping data integration to Problem Frames is not direct; either a Connection frame or more usefully a Transformation frame can be used. The patient monitoring Information Display frame is associated with an Identity frame concern, which draws attention to the need to specify the integrity of model updating to preserve the individual's identity, hence an update to Mr. Jones' blood pressure alert threshold entered by the doctor has to be faithfully registered with Mr. Jones' record in the Patient database. There is another identity concern in the patient monitoring sub-system which points out that the connection between the individual patient, the hardware sensor, and the event stream processed in the software machine all need to

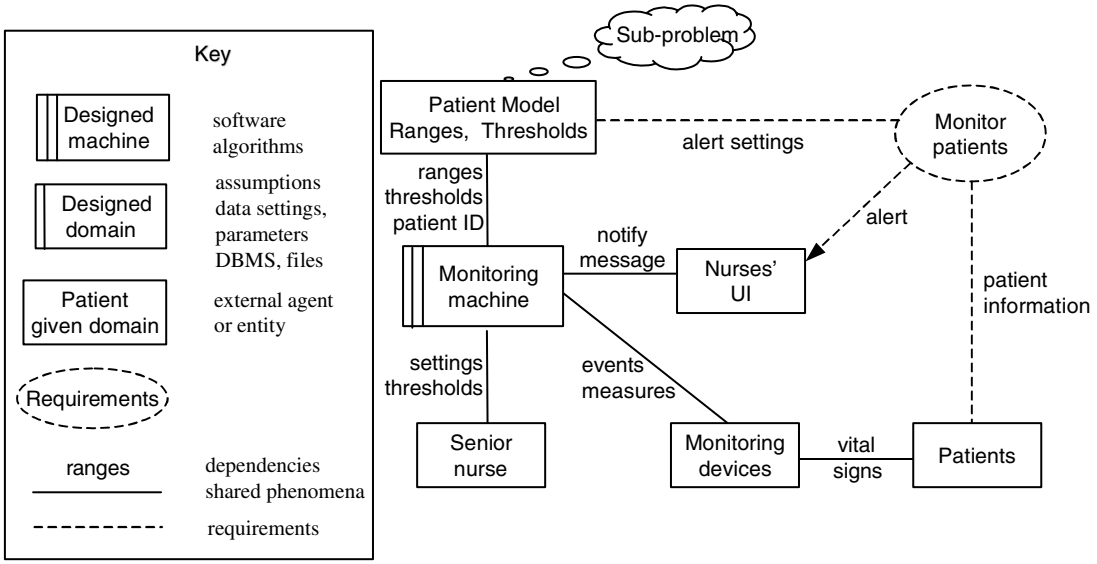


Fig. 7. Information Display Problem Frame diagram for the patient monitoring sub-system. The Transformation frame for event interpretation has not been illustrated.

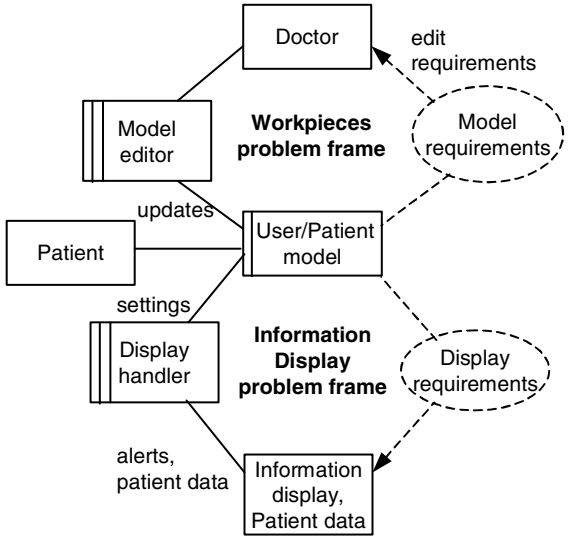


Fig. 8. Problem diagram in the patient model sub-system. Not illustrated are other instances of Information Display frames which specify editing controls and feedback on the display.

be checked to ensure they monitor the same individual. In Domain Theory terms this sub-system contains Object Construction to build and update the model, and Object Simulation to represent it. The Domain Theory and Problem Frames propose similar abstractions and draw attention to the similar problems expressed as frame concerns or requirements issues (e.g., update integrity problem in Workpieces).

Problem Frames are difficult to map in detail to task-based activity. An Information frame models the representation of the patient monitoring data, while Commanded Behaviour frames can be applied to matching symptoms to possible causes in diagnosis and then matching the diagnosed causes to treatments (see Fig. 9). In contrast, the diagnostic part of the application is modelled in considerable detail by the Domain Theory as a single Generalised Task and by three OSMs, which describe requirements for task support and information requirements as well as drawing attention to requirements trade-offs when automating diagnosis according to the accuracy of information and determinism of diagnostic rules.

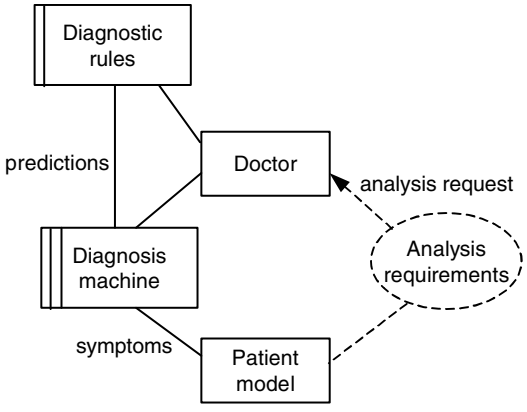


Fig. 9. Commanded Behaviour Problem diagram for the diagnosis support sub-system. This frame is associated with an Information Display frame for representing the results of the diagnosis.

### 5.8. Problem Frame analysis: CSCW sub-system

The CSCW component contains four sub-systems. First is message passing, which involves encoding and decoding packets, modelled respectively as a Transform frame for encoding, followed by a Commanded Behaviour frame for addressing, and two Required Behaviour frames, one for dispatching and controlling the routing, and another for handling the communication protocol acknowledgements. Error checking for lost packets adds more Required Behaviour frames. Allocating users to collaborative sessions by a logon facility suggests a Commanded Behaviour to assigned valid users (see Fig. 10). Control of shared access to a common artefact, i.e., editing the patient model, is a Workpiece frame with Commanded Behaviour to control access of users to sessions; the same is true for viewing/editing the rights over a shared artefact. Integrating Problem Frames with shared phenomena (in this case, users) raises composition concerns of consistency, precedence, interference and scheduling. For instance, a request to edit an artefact from a user who has not joined the current session is a nonsense. Specifying requirements to handle these concerns necessitates modelling permissible event sequences at the state transition level, as well as handling identity concerns for authorised users. Access rights will ultimately depend on domain-specific requirements such as constraints on the number of nurses and doctors who can join a particular session. To handle these functions the generic CSCW modules need parameterised controls

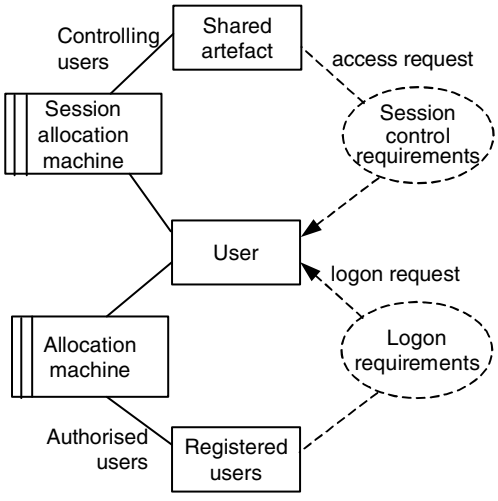


Fig. 10. Commanded Behaviour Problem Frames diagram in the CSCW sub-system for allocating users to sessions, and controlled access to shared artefacts.

and editors to set such parameters (more Workpiece frames).

Message handling in the CSCW sub-system involves several decompositions covered in Jackson's book in the packet-router problem. In the Domain Theory, the CSCW is composed of message-passing OSMs. Turn-taking for conversation management in CSCW is modelled by another Commanded Behaviour Problem Frame, or in the Domain Theory by an Object Allocation OSM, with an Associate Generic Task. Space precludes further description of specification; however, it should be noted that the preceding description only covers the Problem Diagram level; the Jackson approach progresses to more formal specification of requirements for the designed system and assumptions about the domain at the Problem Frame diagram level.

## 6. Lessons learned

### 6.1. Identifying abstractions

Problem Frames were identified using Jackson's problem decomposition heuristics and following dependency links from event output from one frame to another. The small number of Problem Frames made identification relatively easy; for example the CSCW problems involved access control and session allocation, which strongly indicated Commanded Behaviour frames. Similarly, information monitoring indicated Information Display frames, and any need to change representations of input events into output suggested a Transformation frame. Editing operations and user-initiated changes to internal system models were easily identified as Workpiece problems. Difficulties with Problem Frames arose when connecting and integrating them. The models rapidly grew into multi-frame representations. Although some reduction in complexity was possible by using frame context diagrams, there was no escape from the need to specify detailed inter-frame connections and formally model the dependencies between the domain properties (indicative requirements) and required system behaviour (optative requirements).

Domain Theory models were identified by several strategies, such as following event dependencies, links between models in the Domain Theory library, identification heuristics and lexical identification using synonym tables. Since the Domain Theory did not explicitly allocate methods (i.e., tasks) to objects, some confusion arose because of the redundancy between OSMs and Generic and Generalised Tasks. For instance, Object Allocation modelled the problem in an object-oriented view, while Matching (Generalised Task) described a process-oriented view. Some system components mapped naturally to Generic Tasks, e.g., diagnosis and interpreting event patterns, whereas others mapped more easily to OSMs, such as Monitoring and Object Sensing. Object Repair was not an obvious abstraction for the diagnosis and treatment sub-system, and the multiple instances of Object Allocation encountered the same difficulty as the Commanded Behaviour Problem Frame: namely, it was hard to integrate turn-taking control and shared-object control in the CSCW layer.

A summary of the modules identified from both the Domain Theory and Problem Frames analysis is given in Table 1.

Table 1
Abstract models identified in case study sub-systems by the Domain Theory and Problem Frames approaches

| Modules | OSM | Problem Frame |
|---|---|---|
| *TeleMedicine* | | |
| 1. Patient monitor | Object Sensing | Information Display |
| 2. Patient state interpreter | Interpret & Judgement GTs | Transformation |
| | | Required Behaviour |
| 3. Diagnostic assistant | Diagnosis GT | Transformation |
| | Object Simulation | Information Display |
| | Object Allocation | Commanded Behaviour |
| 4. Patient model | Object Construction | Workpiece |
| | Object Simulation | Information Display |
| 5. Treatment control | Agent Control | Commanded Behaviour |
| | Object Repair | |
| *CSCW* | | |
| 6. Message passing | Message Transfer | Transform—encoding |
| | | Command—dispatch |
| 7. Shared access to object control | Object Allocation | Workpiece (model) |
| | Object Construction—Agent Control | Commanded Behaviour—Allocate |
| 8. Common object display | Object Simulation | Workpiece |
| | Object Construction | Information Display |
| 9. Turn-taking access control | Object Allocation | Workpiece |
| | Object Construction | Information Display |

Fifteen OSMs were necessary for the core application but adding user interfaces for editing the registered users database and three Object Sensing Models for other monitoring devices took the total to 19 plus 3 Generic Tasks. The core model needed 17 Problem Frames; however, each interactive user interface is modelled by at least two or three Information Display frames, and more Information Display instances were required for modelling monitoring devices, bringing the total to 23 which would be increased when more monitoring devices were attached. Both the Domain Theory and Problem Frames contributed requirements and insights into the specification of the patient monitoring and nurse alert functions, e.g., identity concern and fidelity of event capture Generic Requirement. Both approaches were weak on integration of several data sources. Problem Frames modelled the diagnostic support component as Transformation and Commanded Behaviour for treatment, but gave little guidance about the functional requirements. The Domain Theory's generalised task models provided more specification detail, as well as guidance on functional allocation. However, both theories provided little advice about the user interface requirements, e.g., alert function modality, message format, etc. for the nurses or doctors.

In the CSCW layer both theories provided reusable requirements and specification advice for the shared artefact and session control components, but were less forthcoming on the generic dialogue control and presentation layer. Problem Frames dealt with these issues in the composite Model View Controller frame and indirectly addressed CSCW concerns of shared viewpoints and constructing different presentations on one shared object for each user (Rodden, 1991) by Interaction and Consistency frame concerns.

### 6.2. Lessons learned: reusing knowledge

Once the abstractions had been identified, both theories guided the software engineer towards identifying important problems inherent in the application domain. Problem Frames do so partly by encouraging more formal specification of the dependencies between the real world domain properties and machine requirements and partly through frame concerns that draw attention to typical pitfalls in specification. The Domain Theory presents advice explicitly in the form of requirements problems that point to pitfalls, associated with generic requirements that present solutions as functional requirements.

Frame concerns were useful in pointing out several potential pitfalls, such as the need to specify identity of individuals (in patient records, patient monitoring data); in update integrity of patient data, accuracy and recency of data; and in synchronisation of control in CSCW conversation and access to shared artefacts (e.g., the individual who edits can also talk about the edits). However, composite frame concerns are not attached to particular Problem Frames so reuse of this knowledge was almost a separate process from specification. Informal inspection of dependencies between Problem Frames uncovered some potential flaws in specification in handling error conditions, such as lost connections between individuals and maintaining conversational threads for recovery.

Problem Frames partitioned the system into more, lower-level components, while drawing attention to several specification problems as frame concerns for update integrity, access to shared data, synchronising queues and controlling execution sequences. Identification of Problem Frames was more difficult because Jackson's method relies on examples and a few heuristics to help the user discover appropriate abstractions. Furthermore, the detailed level of specification implied by Problem Frames made for a complex specification requiring a large number of interfaces between individual frames. This specification proved to be time consuming. The value in Problem Frame analysis lay in identification of frame concerns which pointed out aspects of the specification that required detailed design to ensure correct system behaviour. Problem Frames also encouraged formalisation of the dependencies between domain properties, requirements and specifications. However, a complete formal specification was not attempted, first because of the effort required and secondly because interfaces between frames on which formalism depended were difficult to define precisely. In particular this problem arose in biddable domains involving the human-computer interface where unpredictable user behaviour was difficult to anticipate.

Both theories helped to develop component specifications; however, the reuse library needed to be designed and implemented as software components. Domain Theory models and Problem Frames were mapped as object-oriented (OO) patterns (Gamaa et al., 1995); however, this exercise showed only a weak dependency between the requirements specification models and OO patterns. For instance, Object Sensing mapped to the Observer patterns which provides an object collaboration design for monitoring, but other patterns such as use of Façade for hiding different implementation variations, Factory for flexible implementation of different methods (such as the shared access protocols) and Proxy to control client server communication in the CSCW layer, had few dependencies with requirements models.

### 6.3. Comparison of the approaches

The Domain Theory and Problem Frames both provide improved analytic techniques compared to standard domain analysis methods because they draw attention to the nature of abstractions inherent in all

Table 2
Summary of comparison between the Domain Theory and Problem Frame approaches

| Feature | Domain Theory | Problem Frames |
|---|---|---|
| Generic models provided | 12 families, 56 OSM models, 12 Generalised and 22 Generic Tasks | 5 Problem Frames |
| Source of models | Elicited from experts + theoretical analysis | Theoretical analysis |
| Analysis process | Identify abstraction, reuse generic models and associated requirements | Use Problem Frames to reason about requirements dependencies and constraints |
| Analysis heuristics | No, but implicit in OSM and task model families | Frame concerns |
| Formal specification | No, UML modelling language | Possible FOL expression of dependencies and constraints |

applications at a deep level. They also provide more process guidance and heuristics than Fowler's conceptual patterns. However, the two approaches function at different levels of abstraction. The differences between the approaches are summarised in Table 2.

The Domain Theory provides a more comprehensive library of reusable models, and this drives the analysis approach in which abstractions in the domain are identified by matching to generic models; the models and reusable requirements can then be used to structure the component library. In contrast, Problems Frames is a reasoning-intensive approach where the models (frames) act as cognitive probes to promote reasoning about the dependencies between the domain and the specified solution. The focus is on understanding the dependencies between the real world and the designed machine (indicative and optative requirements in Jackson terminology), then specifying the solution to deal with the constraints and desired system behaviour. Frame concerns draw attention to general specification issues and more formal specification is possible. However, the number of problem types addressed by Jackson's method is restricted; for instance, active software agents, or problems of spatial movement and planning are not dealt with directly. In contrast, the Domain Theory level of abstraction is closer to specific types of problems, e.g., OSM families are given for active agents (Agent Control) and spatial problems (Object Logistics).

## 7. Discussion

Development of reusable software will necessitate advances beyond the current generation of domain analysis methods. These provide process guidance for problem decomposition and indexing of reusable components, but little else. The contribution of this paper has been to apply two theories to design for reuse and evaluate their potential for improving domain analysis and specification of reuse libraries.

The Domain Theory (Sutcliffe, 2002) and Problem Frames (Jackson, 2001) contributed in different ways. The Domain Theory provides a library of generalised models that act as templates for conceptual modelling. These abstractions proved reasonably easy to identify using decision trees and heuristics. The generic problems pointed to issues that required further analysis; however, the Domain Theory did not provide detailed problem analysis advice, whereas frame concerns did draw attention to specification issues more precisely. The generic requirements and functional allocation advice in the Domain Theory were more useful for developing the requirements specification, e.g., in the diagnosis subsystem where generalised task models advised on functional allocation. Problem Frames, in contrast, were less useful for specification of task support functions. To an extent the approaches are complementary; the Domain Theory is a knowledge-intensive approach providing many reusable abstractions to guide thought, whereas Problem Frames provide few abstractions but more analysis guidance. The danger of the Domain Theory approach is that the models could be reused "as is" without much thought; however, the models are more accessible. The converse may be true of Problems Frames; since the models and approach are less accessible, more reasoning may be applied.

Few other methods have addressed the problem of identifying a reusable set of abstractions for use in requirements specification. Reusable software architectures have been produced for system families in flexible manufacturing (Gomaa, 1995), but these are more domain specialised; for instance, Automatic Guided Vehicle components would be decomposed from the Domain Theory viewpoint into a collaboration of Object Sensing, Agent Control and Object Logistics abstractions. Design-level abstractions are well known from the GOF (Gang of Four) Object Oriented design patterns (Gamaa et al., 1995), and some design patterns map directly to Domain Theory models, such as Observer design pattern to Object Sensing models (Papamargaritis and Sutcliffe, 2004). However, most GOF patterns reflect design rather than requirements concerns. The KAOS language and GRAIL tool (Van Lamsweerde and Letier, 2000; Van Lamsweerde, 2001) have been used to create generic models, for instance in hospital reservation systems. KAOS does not propose a comprehensive set of reusable generic models; instead it gives a general process for formal specification of requirements and constraints using a goal decomposition method. However, Van Lamsweerde has specified generic types

of goals and obstacles building on the Inquiry Cycle concepts of attainment and maintenance goals. These can be seen as precursors of generic models. A large number of product lines have been proposed but each author creates new families on an ad hoc basis, with little generality beyond the scope of the original domain analysis. The case study we have reported is, we believe, the first attempt to develop reusable software at more than one level of abstraction so it can be reused more flexibly. Clearly we will have to wait for evidence of reuse of the telemedicine library to evaluate the utility of applying the Domain Theory and Problem Frames to design for reuse.

## Acknowledgements

The authors wish to thank BTexaCT Technologies who supported GP's research, and Chris Voudouris who provided access to the Telemedicine domain. AGS would like to thank Michael Jackson for answers to several queries on Problem Frames; however, any errors in the specification are the author's responsibility.

## References

Bray, I.K., 2002. Expertise: An Introduction to Requirements Engineering. Addison-Wesley, Reading, MA.

Breuker, J., Van Der Velde, W., 1994. CommonKADS Library for Modelling. IOS Press, Amsterdam.

Clements, P., Northrop, L.M., 2001. Software Product Lines: Practices and Patterns. Addison-Wesley, Reading, MA.

Fayad, M.E., Johnson, R.E., 2000. Domain-Specific Application Frameworks: Frameworks Experience by Industry. Wiley, New York.

Fowler, M., 1997. Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading, MA.

Gamaa, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.

Gomaa, H., 1995. Reusable software requirements and architectures for families of systems. Journal of Systems and Software 28, 189–202.

Hall, J.G., Jackson, M.J., Laney, R.C., Nusibeh, B., Rananotti, L., 2002. Relating software requirements and architectures using problem frames. In: Greenspan, S., Saddiqui, J., Pohl, K. (Eds.), Proceedings of RE 02, 1st International Conference on Requirements Engineering. IEEE Computer Society Press, Los Alamos, CA, pp. 137–145.

Harandi, M.T., Lee, M.Y., 1991. Acquiring software design schemas: a machine learning perspective. In: Proceedings of the 6th Conference on Knowledge Based Software Engineering (Syracuse NY). IEEE Computer Society Press, Los Alamos, CA, pp. 239–250.

Jackson, M., 1983. Systems Development. Prentice Hall, London.

Jackson, M., 2001. Problem Frames: Analysing and Structuring Software Development Problems. Pearson Education, Harlow.

Jarzabek, S., Ong, W.C., Zhang, H., 2003. Handling variant requirements in domain modeling. Journal of Systems and Software 68 (3), 171–182.

Keller, G., Teufel, T., 1998. SAP/R3 Process Oriented Implementation. Addison-Wesley, Longman, Reading, MA.

Lam, W., McDermid, J.A., Vickers, A.J., 1997. Ten steps towards systematic requirements reuse. In: Proceedings ISRE '97: 3rd IEEE International Symposium on Requirements Engineering (Annapolis MD). IEEE Computer Society Press, Los Alamitos CA, pp. 6–15.

Levi, K., Arsanjani, A., 2002. A goal-driven approach to enterprise component identification and specification. Communications of the ACM 45 (10), 45–52.

Mannion, M., Kaindl, H., Weadon, J., 1999. Reusing single system requirements from application family requirements. In: Proceedings of the International Conference on Software Engineering, ICSE 99. IEEE Computer Society Press, Los Alamitos CA.

McMenamin, S.M., Palmer, J.F., 1984. Essential Systems Analysis. Yourdon Press, Englewood Cliffs, NJ.

Papamargaritis, G., Sutcliffe, A.G., 2004. Applying the domain theory to design for reuse. BT Technology Journal 22 (2), 104–115.

Rasmussen, J., 1986. Information Processing in Human Computer Interaction: An Approach to Cognitive Engineering. North Holland, Amsterdam.

Rodden, T., 1991. A survey of CSCW systems. Interacting with Computers 3 (3), 319–353.

Rudkin, S., Alan, S., Papamargaritis, G., 2001. COMPOSE: A Component-Based Service Provision Architecture. BTexact Technologies, Martlesham.

Rudkin, S., Smith, A. 2000. A scheme for component-based service deployment. In: Proceedings: Conference on Universal Service Markets, Munich.

Scheer, A.W., 1994. Enterprise-Wide Data Modelling. Springer-Verlag, Berlin.

Shaw, M., 1991. Heterogeneous design idioms for software architecture. In: Proceedings 6th International Workshop on Software Specification and Design. IEEE Computer Society Press, Los Alamitos, CA, pp. 158–165.

Simos, M., Anthony, J., 1998. Weaving the model Web: a multi-modeling approach to concepts and features in domain engineering. In: Devanbu, P., Poulin, J. (Eds.), Proceedings of the Fifth International Conference on Software Reuse (Victoria BC). IEEE Computer Society Press, Los Alamitos, CA, pp. 94–102.

Smith, D.R., 1992. Track assignment in an airtraffic control system: a rational reconstruction of system design. In: Proceedings of the KBSE 92, Knowledge Based Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, pp. 60–68.

Sutcliffe, A.G., 2000. Domain analysis for software reuse. Journal of Systems and Software 50 (3), 175–199.

Sutcliffe, A.G., 2002. The Domain Theory: Patterns for Knowledge and Software Reuse. Lawrence Erlbaum Associates, Mahwah, NJ.

Sutcliffe, A.G., Carroll, J.M., 1999. Designing claims for reuse in interactive systems design. International Journal of Human–Computer Studies 50 (3), 213–241.

Sutcliffe, A.G., Dimitrova, M.T., 1999. Patterns, claims and multimedia. In: Sasse, A., Johnson, C. (Eds.), Proceedings of the INTERACT 99 IFIP TC.13 International Conference on Human–Computer Interaction. IFIP/IOS Press, Amsterdam, pp. 329–335.

Sutcliffe, A.G., Ennis, M., 2000. Designing intelligent assistance for end-user information retrieval. In: Paris, C., Ozkan, N., Howard, S., Lu, S. (Eds.), Proceedings of the OZCHI-2000 (Sydney). CSIRO/CHISIG, Canberra, pp. 202–210.

Sutcliffe, A.G., Maiden, N.A.M., 1998. The domain theory for requirements engineering. IEEE Transactions on Software Engineering 24 (3), 174–196.

Van Lamsweerde, A., 2001. Goal-oriented requirements engineering: a guided tour. In: Proceedings of the RE'01—5th IEEE International Symposium on Requirements Engineering, Toronto, August, 2001. IEEE Computer Society Press, Los Alamitos, CA, pp. 249–263.

Van Lamsweerde, A., Letier, E., 2000. Handling obstacles in goal-oriented requirements engineering. IEEE Transactions on Software Engineering 26 (10), 978–1005.

Vici, A.D., Argentieri, N., Mansour, A., d'Alessandro, M., Favaro, J., 1998. FODAcom: an experience with domain analysis in the Italian telecom industry. In: Devanbu, P., Poulin, J. (Eds.), Proceedings of the Fifth International Conference on Software Reuse, pp. 166—175.

Wehrend, R., Lewis, C., 1990. A problem-oriented classification of visualization techniques. In: Proceedings First IEEE Conference on Visualization: Visualization 90. IEEE Computer Society Press, Los Alamitos CA, pp. 139–143.

Weiss, D.M., Lai, C.T.R., 1999. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, Reading, MA.

Zhou, M.X., Feiner, S.K., 1998. Visual task characterization for automated visual discourse synthesis. In: Karat, C.M., Lund, A., Coutaz, J., Karat, J. (Eds.), Human Factors in Computing Systems CHI 98 Conference Proceedings (Los Angeles). ACM Press, New York, pp. 392–399.