INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 05 MELBOURNE, AUGUST 15 – 18, 2005

EXTENDING THE DESIGN OPPORTUNITIES AFFORDED BY MODELLING FRAMEWORKS

J A Dalton, P W Norman, P Sen, S Whittle

Abstract

Modelling frameworks are applications which can be used to visualise complex systems during the design stage. They can be used to derive emergent properties of systems as well as providing valuable high-level system metrics. Frameworks of models facilitate the tracing of system properties and aspects such as design decisions. The Newcastle Engineering Design Centre has developed and constructed a modelling framework. Developed with early design conceptualisation in mind it soon became apparent that frameworks could be extended for use in other areas of the design processes. This paper describes the modelling framework and its capabilities. This scope is extended to include requirements traceability, which necessitate a level of documentation attached to a framework representation of a system. For future work we consider extending the information system to encompass the product lifecycle and suggest some advantages this will bring to understanding lifecycle issues such as manufacturing, implementation, validation and operational requirements. This direction will allow an increase in the transparency of the product lifecycle. The work to date has extended the uses of modelling frameworks to many aspects of the design process and shows that frameworks can be deployed throughout the lifecycle of products related to complex systems.

Keywords: modelling frameworks, traceability, design rationale, design reuse.

1 Introduction

Over the last few decades, the design processes related to complex systems have become more difficult. This problem mainly arises from the uncertainty associated with the behaviour of complex systems under changing conditions. For example during the design stage of a system, important decisions will be made and system parameters changed. Often designers will attempt to optimise according to their own sets of constraints without acknowledgement of global consequences. When a designer is part of a large team developing complex systems, global performance targets will depend more on the exchange of information between the relevant teams of designers. The generic problem then becomes one of communication.

A consistent approach to data management can facilitate this communication and one way of achieving this is through an integrated approach to modelling. Organisations involved in the design of complex systems often utilise many models. These models may be of varying age and fidelity and are intended to help design parts of a system. Often large organisations will have limited knowledge of the scope and capability of the models it owns. These may have been created a number of years beforehand by personnel who have since moved on. Information relating to any assumptions or the actual fidelity of the model may be uncertain, in some cases the original source code may be unobtainable.

A potential approach to integration is the use of modelling frameworks. As well as enabling a form of system simulation, such frameworks can be used to monitor emergent system properties by providing a meta-modelling environment. Emergent properties also provide a useful means of conducting trade-off analyses, which can optimise and make best use of design and manufacturing resources at the conceptual stage in the design process. This can also be a means of reducing development costs by using system simulation to reduce testing. A subsequent expectation is improved quality and reduced lead-time of designs, achieved by ensuring all necessary information is available early in the design process. System properties and other aspects such as design decisions and recorded design rationales can be traced and recorded; thus enabling the tracking and analysis of performance and design change. In conjunction with our sponsor, BAE SYSTEMS, this work has used an object-oriented approach to develop a modelling framework with a view to providing solutions to the problems in the design of complex systems. Unified Modelling Language (UML) has been used to describe and construct the framework, realised as a cross platform software application written using Java.

1.1 Objectives and benefits

The objectives of this work have been to obtain an understanding of areas in which modelling is used by developing an integrated modelling environment that can be used by designers of complex systems. The benefits of using such an approach is that it will assist designers to optimise designs and allow measurement of overall performance by considering system properties throughout the design hierarchy. The resulting objectives are generalised as: implementing a more efficient and effective use of modelling capability, permitting a consistent record of how systems function, creating a traceable record of the design decisions used, and encouraging reuse of previous design decisions in new systems. Example problems and solutions highlight where this approach is aimed.

- New aircraft design are invariably more advanced than previous ones. The result is that new designs involve ever more interacting sub-systems and disciplines – they are more complex.
- Major new projects are becoming fewer and subsequent time intervals between projects longer. The result is valuable design experience is becoming less available and in some cases lost completely.

An integrated approach to modelling can provide the following.

- To become a central communication tool between design disciplines in the early stages of design work, such that new design information is available to all parties.
- The effects on the system of a proposed design change can be evaluated immediately and used to guide system synthesis.
- New designs can benefit from a consistent record of previous designs. New designs rarely start from scratch, more often using previous designs as a basis. A fully accessible record of the design process is an obvious benefit.

1.2 Structure of this paper

In the next section we discuss the way in which the basic structure of the modelling framework has been designed and implemented as an information system. Traceability (an important aspect of information systems) and its implementation is discussed in Section 3. Using traceable information systems as a basis, Section 4 describes how these can be used to implement a form of design reuse implementing knowledge management (KM) techniques.

Future enhancements to the KM work (Section 5) includes a look at more formal means of recording design rationale. This will allow the capture and reuse of more advanced system design decisions. Section 6 is predominantly speculative, but based on work completed to date, and investigates the prospect of looking at lifecycle issues using the information system. Initially we have only been concerned with the earliest part of a product lifecycle (concept and design), we now look at the scope for extending this approach to embrace the overall lifecycle. The objective of doing this will be to improve lifecycle transparency and to utilise the KM techniques we have developed so far to facilitate designing for upgradeability.

2 Modelling framework

The modelling with which we are associated is mainly computer based, it is therefore reasonable to assume that a software application would be the best approach. This does not preclude the integration of information from other sources, for example test bed data. This section outlines the basic structure of the modelling framework.

2.1 Basic structure

The fundamental structure of the framework is an abstract container, which holds components representing the models being integrated. The purpose of the framework container is to: represent models, their associations, and properties relating to system configuration, hold and exchange information about a system defined by the models it represents, identify emerging properties, and provide a platform upon which prototype designs can be assessed.

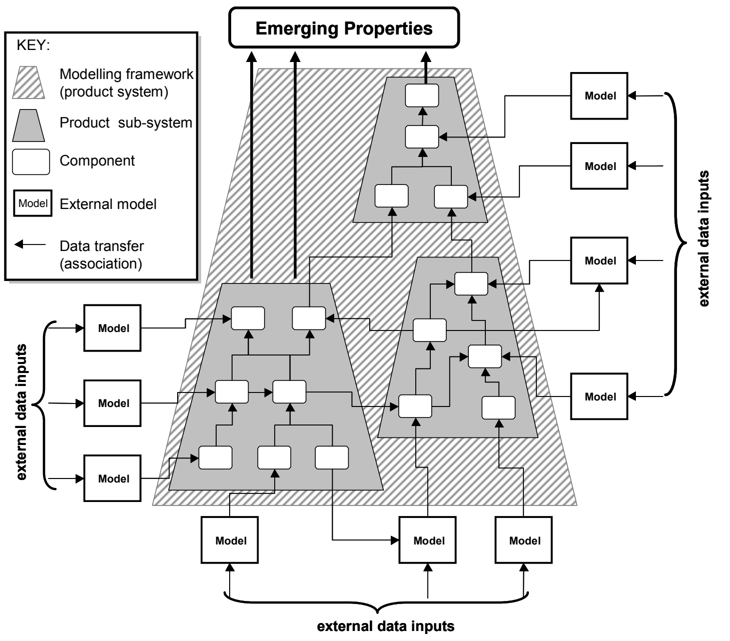

Figure 1. Conceptual view of a modelling framework.

A conceptualised view of this is shown in Figure 1. The figure represents a number of sub-systems forming an overall system meta-model. Each sub-system contains an arbitrary number of components, which each represent a model.

Components are place-holders for objects which deal with the modelling representation (these can also act as aggregations of models). These hold and transfer data, maintain links, evaluate models, obtain data from external sources. A view of a component and its relationship with the modelling domain is shown in Figure 2.

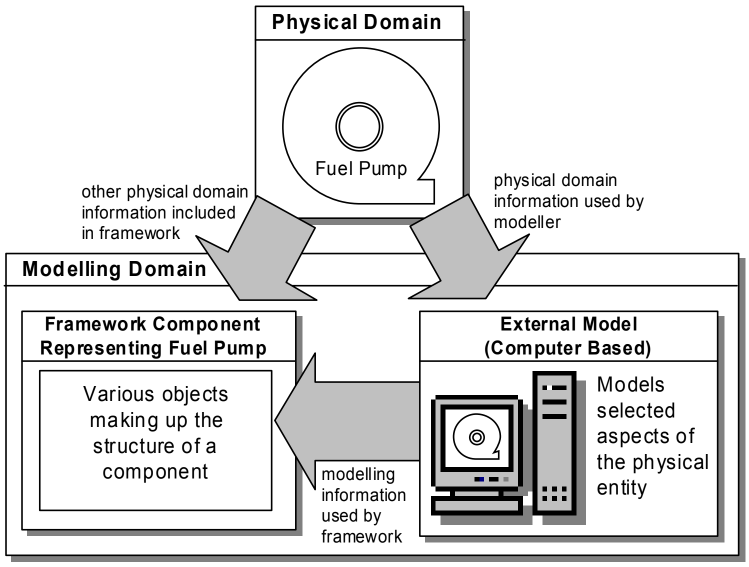
Physical Domain
Fuel Pump
other physical domain information included in framework
physical domain information used by modeller
Modelling Domain
Framework Component Representing Fuel Pump
Various objects making up the structure of a component
External Model (Computer Based)
Models selected aspects of the physical entity
modelling information used by framework

Figure 2. Relationship between the physical modelling domain, highlighting a framework component.

An engineering entity such as say, a heat exchanger may exist physically or as a proposed design. The important properties of either are extracted and modelled to expose the properties of interest. Defining the representation objects of a component requires an understanding of the type of information propagated through a system by the associated models. This includes:

- functional behaviour,
- design definitions, goals, or constraints the designer is working within,
- structural information about an entity being modelled, and
- any other additional information that may be deemed necessary to model the system.

The terms behaviour and functionality are ill defined [1,2], yet a consistent definition is required to represent them. We therefore need to decide on terms of reference. A common view of behaviour and functionality is that behaviours achieve functions [3-5]. For example, Modarres [4] argues that complex systems can be best described by (a number of) hierarchical frameworks. One of these is defined as a functional hierarchy, which: ... represents the role of the relevant parts of the system described by its structure. If components are the relevant parts of a system then the functionality of a component is its role or purpose. Behaviour is defined as the way that a purpose (or a function) is achieved [3,4], i.e. behaviour is how an object acts or reacts, in terms of its state changes, so as to attain its intended function [4]. These views are utilised within the framework such that: function describes the purpose of a component, and behaviour is the way that purpose is achieved [3]. Design definitions may be regarded as goals, i.e. as a subjective motive of a system or its components [4]. The relationship between functions and goals are described by the following: To attain a goal, one needs a collection of functions to be realised [4]. These constraints may then be taken as a measure of success or failure for a function. Models of an entity will often consist of behaviour and functionality. However the system properties may require further information. For example, the behaviour and functionality of a heat exchanger, say, may be successfully modelled without any reference to other attributes, such as size, mass or cost. Yet these may be required to derive the emerging properties of a complete system. Conceptually the modelling framework is an information meta-model, designed to support prototype system designs described by means of system properties. In support of this we have constructed this as an object oriented information model.

2.2 Class diagram

The fundamental notion in object-oriented design is that of an object, where "an object is an abstraction of the real world that captures a set of variables that correspond to actual real world behaviour" [6]; a definition, which captures the essential principles we are attempting to emulate. As Rumbaugh [7] states "the essence of object-oriented development is the identification and organization of application-domain concepts ...". In keeping with this ideal, the structure of the framework meta-model and its constituent parts should be concisely described and explained. An information modelling language such as UML is a good candidate for this task [7]. To do this successfully the framework model as we have described it so far has been described as a series of classes.

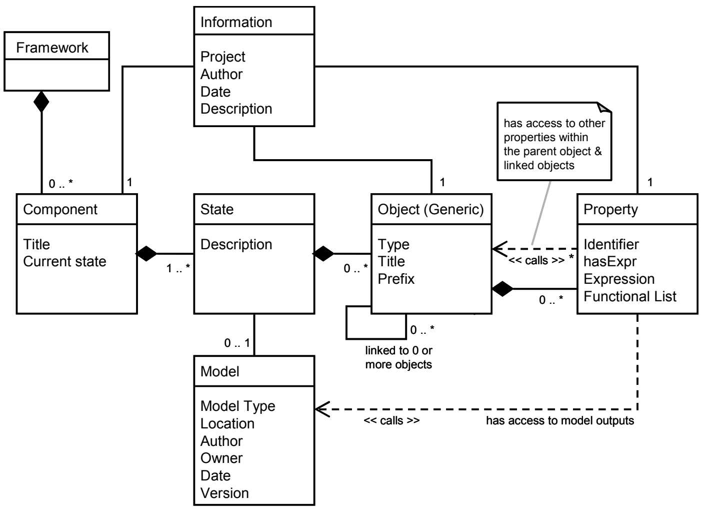

Figure 3. The main class diagram for the framework.

The class diagram for the framework (Figure 3) shows a component can contain zero or more abstract objects (representing the structural, goal, functional and behavioural objects). In turn each of these objects can hold zero or more property objects. The inclusion of a state class allows a component to be dynamically changed so that it can represent different models. The main attributes of the Property class are the property identifier and any expression or functional list associated with a property. This is so that a property may be expressed in terms of its functional relationships. For example:

$$\text{force} = \text{mass} * \text{acceleration, as: force} = \Phi(\text{mass, acceleration}).$$

The class diagram shows that as well as being referenced from a represented model, a property may also refer to other property objects from any associated component.

2.3 Data transfer

A view of data transfer between a represented model and the framework is shown in Figure 4. This figure highlights three distinctive forms of transfer. The first is between objects held by a component. The second is between components, but within a framework (which represents the integration between models). Finally the most significant is between the framework and the modelling domain. Information interaction between objects and the modelling domain requires formally defined associations. In particular this type of data transfer requires a neutral data exchange format. An emerging standard is Application Protocol (AP) 233, Systems engineering data representation developed within the ISO STEP context (ISO 10303). This standard is specifically of interest since as well as being firmly aimed at avionics and airframe systems, it sets out to define within the scope of ISO 10303 such discipline views as:

- definitions between interacting systems,
- support of hierarchical break down and object-oriented modelling techniques,
- functional and non functional requirements of a system in each lifecycle phase, and
- the definition of static and dynamic behaviour of a system.

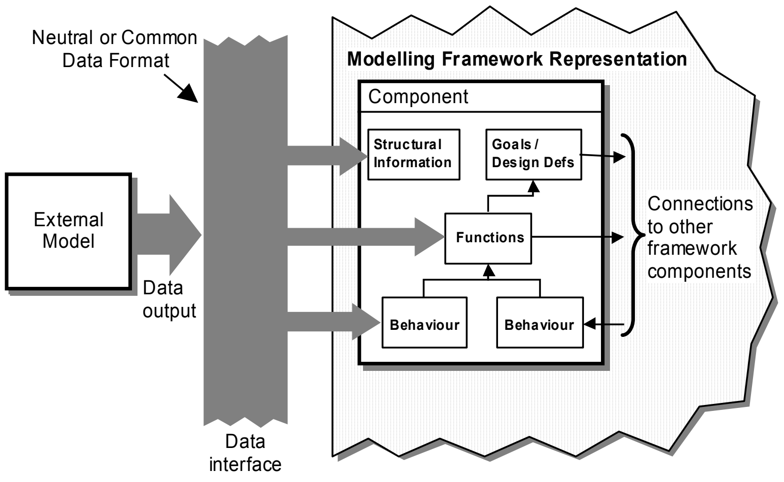

Figure 4. View of data transfer between domains.

3 Traceability

The framework permits tracing of represented entities, such as properties. For example the mass of an engine may be critical to a system design. By tracing a property through a represented system, low level contributing properties can be determined. This information can allow designers to target their efforts and can also be used to test the sensitivity of properties and estimate the required fidelity of integrated models. A typical sensitivity trace may show that a low fidelity model (i.e. a cheaper model) is all that is required to simulate a system. Whereas another trace may identify where an increase in model fidelity is needed.

3.1 Methods and approaches

To perform a trace requires some relationship between at least two entities, and this relationship may take the form of a syntactic or semantic nature [8]. A number of types of models within which tracing process can operate have been identified:

- information models,
- process models,
- documentation models, and
- enterprise models.

The model we use is an information model. The potential to include other models, such as the documentation model, remains an option. In relation to syntactic traceability the rigour referred to by Pearson [9] is that of a functional nature, which is how the relationship between properties is described in the framework meta-model, however given other matching criteria, other objects can be traced. Using the above example of the mass of an engine sub-system, tracing the contributing parts of that mass may be considered a downward trace. An alternative is to perform a trace from a low level property upwards through a system to identify which emerging properties it influences.

Figure 5 shows a simple example of a downward trace. The 'Result' object, shown in the top left of the figure, contains an emerging property 'force'. This property is the product of other properties distributed within connected objects. The result of the trace on this property is displayed using a tree structure (right of the figure). This displays all contributing properties down to the lowest level. The tree root is the title of the property being traced. Each branch in the tree is indicative of some functional relationship, i.e. the syntactic relationship between the properties. The tree in Figure 5 shows that in 'Result' the property 'force' is a function of 'o1.m' and 'o2.a'. In turn in 'Object o1', 'o1.m' is a function of 'rho' and 'vol'. The rightmost leaves represent the low level properties, that contribute to the traced property. This example shows the low level properties, which make up 'force' in 'Result' are 'o1.rho, o1.vol, o2.v2, o2.v1 and o3.time'.

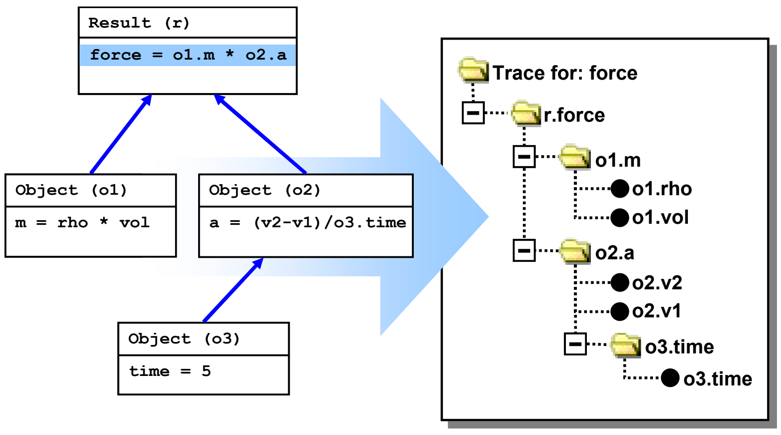

Figure 5. Downward trace tree.

3.2 Implementing traceability within a framework

The overall tracing process can be described using three stages [10]:

- formulation of a data structure,
- population of the data structure, and
- the tracing process itself, which searches the populated data structure.

These follow the Executable UML approach advocated by Mellor [11]. The class diagram describing the tracing data structure is shown in Figure 6. This is the template populated by properties during a trace. The search processes then operate on this to produce the trace tree.

The primary data container is a hashtable utilising two main methods, put() and get(). The put method requires two parameters, which reference objects, these are a key and some content. The method uses a unique key to place a content object within the container. Once stored the get() method when given the unique key will retrieve a reference to the stored content object from the container. The two classes which inherit the hash table characteristics (shown in the figure) are PropertyTrace and TraceRecord. PropertyTrace is the fundamental class used in the tracing process and contains two main methods: constructData() which actually assembles the data and doSearch() which performs the searching process. As a container the data objects placed in this are an aggregation of a key and some content. The key object is a string that uniquely identifies an object in the form of:

component Title+":"+objectTitle+":"+prefix.

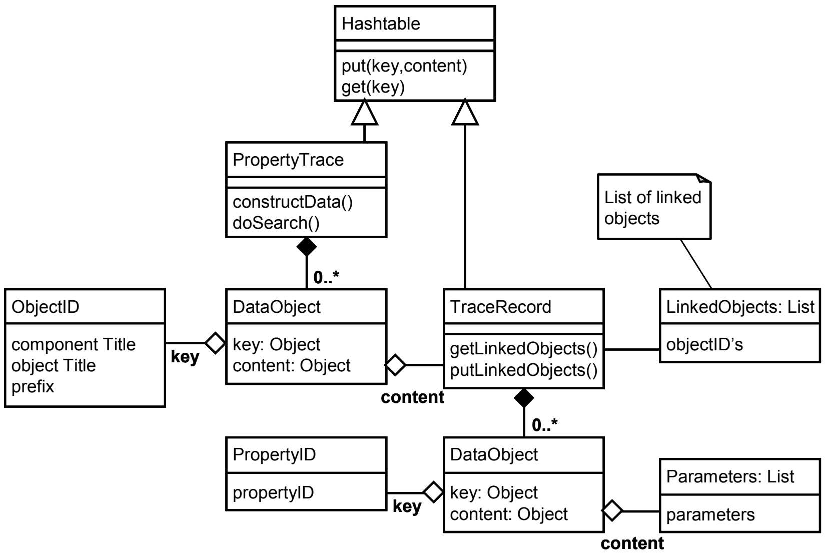

Figure 6. Class diagram describing the data structure.

The content object of this class, titled TraceRecord is itself an extension of the hash table class and is a container for data objects that have a unique key, which references the properties held by a framework object. The content held within this data object is a simple array of parameters attributed to the property, for example:

$$y = a * b / c, \text{ can be referred to as: } y = \Phi(a, b, c),$$

where y is the property and the parameters are a, b, and c. The class diagram in effect describes a nested hash table, the basic PropertyTrace class holds objects where the key is a unique identifier for all the objects in a framework and these hold a content object TraceRecord. This provides a very generic tracing structure, which can be applied to any linked entities.

4 Knowledge capture and the reuse of designs

A desirable aspect of designing systems using a framework is to be able to trace and potentially reuse stored design decisions. To achieve this requires a view of how design decisions are made and recorded so that knowledge within a tested and validated design can be stored, traced and potentially reused. We begin by discussing how designs can be represented, followed by a consideration of how the ontology of a framework may be defined and how this can utilise Problem Solving Methods (PSM's).

4.1 Capturing the design process

The design process involves mappings from the design space specifications to a space of devices or components [12]. This is conducted by means of a search within the space of possible subassemblies of components for those that satisfy a set of constraints. To achieve a solution the design problem solving process can be subdivided into a number of subtasks. A task structure lays out the relationship between a task, methods for solving the task and knowledge requirements for the methods [12]. A design task can be defined as [12]:

- a set of functions to be delivered by an artefact, including a set of constraints which must be satisfied, and
- a relevant technology, that is a repertoire of components (assumed to be available) and a vocabulary of relations between the components.

A design solution consists of a complete specification of a set of components and their relations that together describe an artefact that delivers the required functionality and satisfies the constraints. The most relevant object held by a component in relation to a design is the functional object; as previously stated functions describe what an object does and this represents the role of the relevant part of a structure. Therefore a framework using functional representation satisfies the requirement that a design can be specified by a set of functions. The constraints can be achieved using the goal object, also included within a framework component. As we have stated: goals may also be described as design definitions in that a designer may specify some limit or value that a function must reach [13]. The technology, or the vocabulary of the system representation is already present in the form of an ontology (described below). Central to the ontology are that the components, objects and associations are a representation of the system design. This also implies that the system being represented is either a conceptual or existing system being portrayed as the designer intends it to be, i.e. the framework contains a representation of the design decisions that have gone into the creation of the system.

4.2 Ontology

The purpose of an ontology is to capture domain knowledge in a generic way and provide a commonly agreed understanding of a domain [14], which may be reused and shared across a number of applications and groups. In practice an ontology is a hierarchically structured set of terms for describing a domain that can be used as a foundation for a knowledge base [15]. This ontological requirement is met by the structure of the modelling framework as an information system. Although we do not have a formal language for the framework, its storage as an XML file defined by a DTD file fulfils this requirement. Swartout [15] generalises that: "Large scale knowledge based systems are difficult and expensive to construct. If we could share knowledge across systems, costs would be reduced. However, because knowledge bases are typically constructed from scratch, each with their own idiosyncratic structure, sharing is difficult. Recent research has focused on the use of ontologies to promote sharing". The use of XML with this application clearly solves the issues of sharing information and compatibility with other applications [15] and is based on the design of the underlying structure of the modelling framework.

4.3 Problem solving methods

The term Problem Solving Method (PSM) describes the reasoning process of a knowledge based system (KBS) [14]. A PSM defines and encapsulates how to achieve the goal of a task. The form of a PSM may differ between applications, however in general a PSM describes which reasoning steps have to be performed and the type of domain knowledge needed to perform a task. PSM's provide the abstract procedures required for solving tasks. They are arranged to utilise the available ontology which describes how a problem will be solved. In other words, a PSM encapsulates a solution to a problem (that has already been defined and solved) in the method ontology. A potential architecture for a PSM [16] is shown in Figure 7.

This shows that the heart of a PSM consists of a hierarchical structure of inference steps. Each step can consist of solutions to sub-tasks which themselves are solved by (sub) PSM's. Referring to Figure 7, a PSM can be described by the following three parts [16]:

- functional specifications,
- operational specifications, and
- requirements.

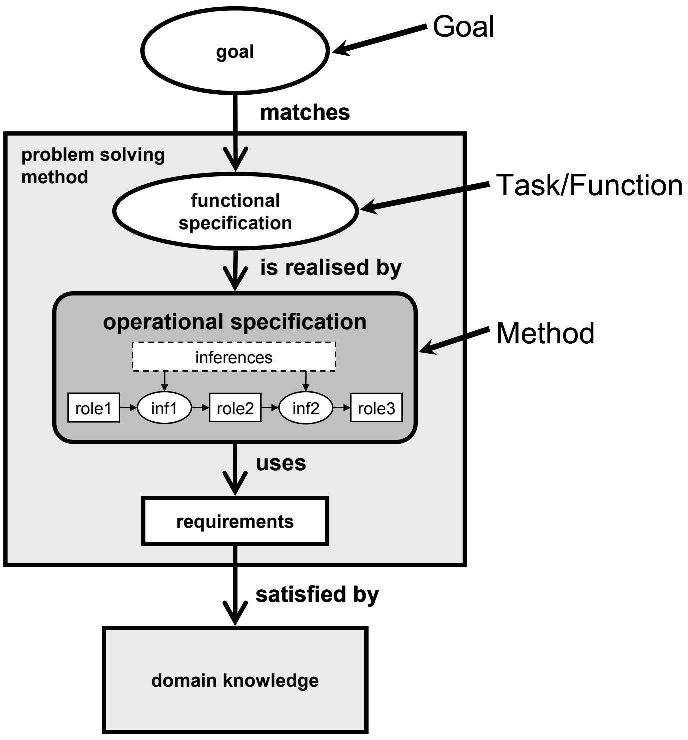

Figure 7. An architecture for a Problem Solving Method [16].

The functional specifications of a PSM describe the tasks it can accomplish. Whether these tasks are achieved or not is measured by an appropriate set of goals. In relation to the modelling framework we have functional and goal objects, which are used to describe these aspects and functionality can be tested against one or more goal objects. Both the functional and goal objects within the framework can also be formed into a hierarchical structure from other functional and goal objects. If we take a closer look at an atomic inference this may be considered a singular reasoning step, which has input information supplied by an input role and output information (based on whatever that reasoning step did) supplied to an output role. An atomic inference (or reasoning step) can be represented by a component as used in the modelling framework. As part of a system simulated by a number of interconnected components a single component may be considered a reasoning step. The input and output roles also form a similarity with the inputs and outputs from a component. Although components do not exchange data, the objects they contain do and this data can be taken from (or sent to) models or other objects within a framework. Combining components together then replicates a hierarchy of inferences, which form the operational specification of a PSM.

4.4 Implementing design reuse with a framework

Designs represented by frameworks may be viewed as successful cases. If so, the information they hold is useful when designing a new system from scratch. The basis of this is that once a particular problem has been solved, this solution serves as a potential solution to a new problem, which approximates the original problem. The normal approach to the storage of PSM's for design reuse is to arrange these in libraries and this raises a number of issues [14]. An alternate approach that we use is to regard the storage of frameworks as a virtual library of PSM's. This makes use of the tracing capabilities developed with this framework. If we consider a system being described by an electronic store of a number of frameworks, then for a given problem (defined by a set of functional requirements) we can search this store for matches. The search takes place on the functional objects held by the frameworks. Once found, the tracing capability is deployed to find all components (inference steps), which contribute to this functionality. When this process is completed we have a set of matched functional capabilities, that includes the contributing inferences. In other words a PSM, as described by the architecture in Section 4.3. This then becomes a potential solution to a new design problem.

5 Recording design rationale

The modelling framework representation has been designed and built as a means of integrating modelling capability. Its primary purpose is to demonstrate (at the conceptual stage of the design cycle) the way a system operates. Properties can be exchanged with other model representations via editable links and it is this which helps to simulate a systems capability. An extensive amount of other pieces of information can also be held within this arrangement. This can be in the form of textual descriptions, links to external documentation, unit data, version information and authorship as well as many other types. Two methods of manipulating the data held by the framework have been added, that is bi-directional tracing and a means of retrieving "tacit" knowledge stored in the frameworks. As described in previous sections, this forms a traceable (and searchable) design knowledge store, as well as an effective modelling database and repository. To make better use of traceability and knowledge management, a more formal record of design is required in some standardised form of documented design rationale.

To achieve this requires the capture of design rationale, which can be accomplished using requirements management methods. Since the design process is heavily dependent on experience it is this experience that the design rationale will have to capture. This will therefore act as a record that can be used by other designers in the future or by inexperienced designer as part of a learning process. Many of the ideas related to design rationale also include traceability, since it is in the interest of designers to be able to trace the reasons why a particular choice of design has been made. These approaches consider what should be recorded in such a way that these types of traces can be implemented. A number of approaches have been identified, including: a method based on rich traceability and one based on requirements management. In each some form of documentation is associated with the method. The first method also utilises a flowchart of design rationales between a users requirement and what the requirements of the system are. The rationales simply plot the designers' path between these. This also exists in parallel with a large document base which records all relevant information.

Currently we have not implemented a way of recording a flowchart of design rationales. Essentially these items would be the system requirements. Therefore to implement such an approach a flowchart facility would be needed. Another view is that goal objects, contained by a component can be used to specify system requirements. The functional and behavioural objects can contain the relevant information required to attain these goals. If the parts of a represented system contain references to relevant documentation then it will be relatively easy to extend the concept of this framework toward the documentation model and also the responsibility model and change control. To investigate design rationale it is probably most important to regard what a design engineer does and thinks. Research in this field has identified how experienced designers think and record what they do, prior to arranging this in the form of a design framework that novice designers could adhere to [17]. In theory this helps novice designers to think and act more like experienced designers. The main aspects exhibited by experienced designers are, they: will try to identify many more of the issues than a novice designer, be aware of the reason why something was done, will refer to past designs as starting points for new designs, question if something is worth pursuing, question the validity of data, keep options open, be aware of trade-offs, and be aware of limitations. A good record of design rationale will therefore have to be able to demonstrate these points and this will give a definite direction for the progression of the MFR as a design tool in this area. What an experienced designer has is experience and if a novice designer can be supplied with the benefit of someone else's experience then this will address this issue. This is therefore the type of functionality that an extended MFR would be expected to fulfil.

6 Extending the range of application

Having looked at the basic modelling framework, created for the early design stage of systems, this section considers the implications of extending these ideas further. Many aspects of design now attempt to fully consider the entire lifecycle of a product. Addressing these lifecycle issues can be beneficial from the point of view of increased design flexibility and allow insight into the potential for increasing the upgradeability of designs. To help achieve this, it is proposed that the approach we have used so far can be extended further into the lifecycle of a product. This section considers some of the issues relevant to this and discusses the implications.

6.1 Lifecycle issues

Historically, traditional designs have paid little attention to the later parts of the lifecycle. This is understandable because such issues are not of immediate importance to most designers. One exception to this is manufacturing, but other issues that are often related to maintenance and end of life issues are frequently neglected. It is generally accepted that in future most design engineers will need to consider all aspects of the design lifecycle. This is not just a case of optimising a design or making a design more cost effective but means a more holistic view of the cost of a product throughout its lifecycle, involving the total cost of ownership. In future, aspects such as this will more often be a matter of legislation [18]. In general what is required is an increase in the transparency to the designer of the:

- manufacturing costs
- maintenance costs, and
- termination or end of life (scrap) costs.

What we have developed so far can be extended such that a deeper insight into these lifecycle issues can be achieved. Currently what we have is intended for use in the earliest stage in the lifecycle, that is the conceptual design. To increase this further will require a more complete understanding of the following stages: manufacturing, supportability, cost, implementation, validation, and operational requirements. In effect this means all aspects of the lifecycle from initial concept to final disposal. The result will be an increase in the overall transparency of the product lifecycle. The benefits that would accrue from this approach would include:

improvement of product quality, concurrency of product and process phases and an overall reduction in development time.

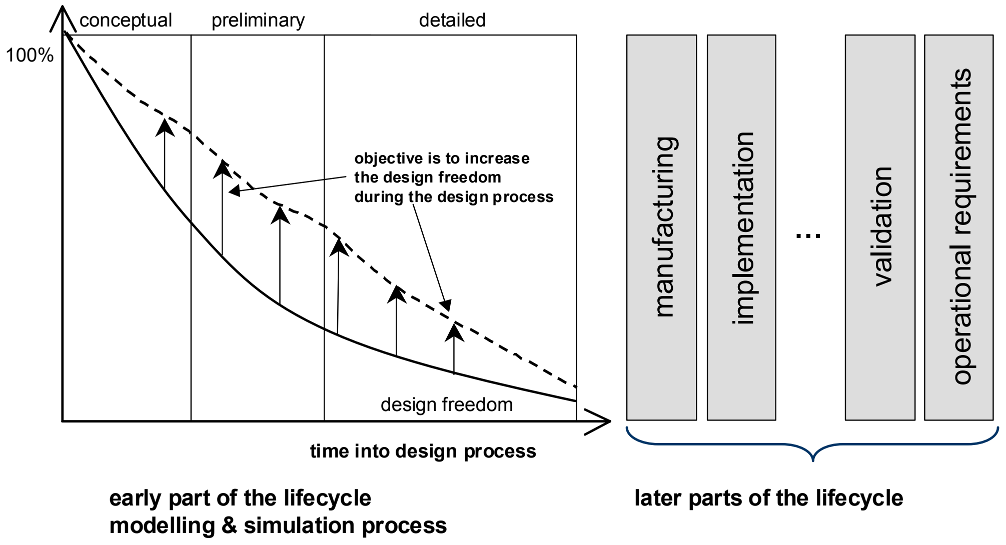
conceptual
preliminary
detailed
100%
objective is to increase the design freedom during the design process
design freedom
time into design process
manufacturing
implementation
...
validation
operational requirements
early part of the lifecycle modelling & simulation process
later parts of the lifecycle

Figure 8. Design freedom in the early stage of design.

If we consider Figure 8, we see that in the initial design stage design freedom starts from an arbitrary 100%, but quickly falls as the design progresses into the detailed stages. The objective of the modelling framework, is to lift this freedom thus allowing more flexibility in design. Extension of the modelling framework into the later stages of the lifecycle will produce an even greater flexibility into the design stage by increasing the transparency. Thus allowing system designers more extended views of their design consequences.

6.2 Prospects of an all encompassing design tool

One of the aspects discussed earlier was that of storing design rationale and making the knowledge accrued by the system designer available for reuse. Another view of this is that of using the information system as a legacy utility. This is also linked to upgradeability which requires flexibility in the design of a system. As we saw in Figure 8, an increase in flexibility is one of the aims of the current approach.

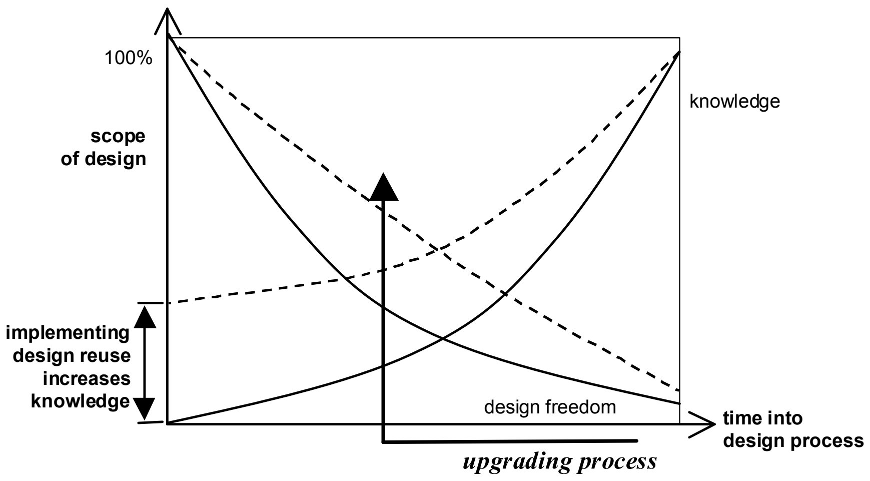

Figure 9. Facilitating the upgradeability of a system design.

If we consider this further we can see that knowledge of a design is also linked to flexibility (Figure 9). That is the decrease in design freedom sees a corresponding increase in knowledge. The overall aim is to lift these levels using tools such as the modelling framework, and Figure 9 also indicates that by implementing design reuse we can lift the level of knowledge at an earlier stage in the design.

Upgradeability is closely linked to flexibility and knowledge levels. Often a design will be in operation for a number of years before an upgrade is considered. It is also extremely difficult to design with upgradeability in mind, since technology will advance and the scope of any future upgrade must be generally unknown to the designer. Therefore by increasing the levels of design freedom and knowledge this will give greater scope to the issue of upgradeability. This will also be facilitated by using the modelling framework as a fully integrated information system, which can track and monitor changes as well as giving some indication of the knock-on effects as far as later lifecycle issues are concerned.

7 Conclusion and discussion

Modelling frameworks are being used to help provide solutions to the ever growing problems of designing complex systems. We have presented our approach to this in the form of a basic modelling framework, which can represent a wide variety of computer based models, trace properties produced by those models and be used as a basis for design reuse. This is in a form designers of systems can use to view the relevant systems information they need.

The design of the modelling framework application (written using the Java programming language) has been modelled using an object oriented approach. Designing the framework using UML has provided a flexible approach that can be easily adapted and extended into other areas of design use. This flexibility is important to extending the current version to enable it to capture and record design rationale. More importantly this versatility can be utilised to expand use of the modelling framework into areas relevant to the product lifecycle. The view of the lifecycle issues discussed with respect to the capabilities of our approach with frameworks, show that the scope exists for such expansions. When implemented, this will give a more transparent view of the lifecycle to the designer and will help enable future designs to be made with upgradeability in mind.

References

[1] Herzog E., and Törne A., "Support for representation of functional behaviour specifications in AP-233", 7th International Conference and Workshop on Engineering of Computer Based Systems, pp 351-358, 2000.

[2] Salustri F., "Function modelling for an integrated framework: A progress report", 11th Florida Artificial Intelligence Research Symposium, pp 339-343, May 17-20, 1998.

[3] Keuneke A., "Device representation: the significance of functional knowledge", IEEE Expert, April 1991: 22-5.

[4] Modarres M., and Cheon S.W., "Function-centered modeling of engineering systems using goal tree-success tree technique and functional primitives", Reliability Engineering and System Safety 64 (1999) 181-200.

[5] Pegah M., Sticklen J., and Bond W., "Functional representation and reasoning about the F/A-18 aircraft fuel system", IEEE Expert, Vol. 8, No. 2, April 1993.

[6] Sage P., "Systems engineering", J Wiley, 1992.

[7] Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W., “Object-oriented modeling and design”, Prentice- Hall, 1991.

[8] Pearson S., and Saeed A., “Information structures for traceability for dependable avionic systems”, Technical Report number 567 Department of Computing Science, University of Newcastle, 1997.

[9] Pearson S., Riddle S., and Saeed A., “Traceability for the development and assessment of safe avionic systems” Proc. 8th Int. Symposium International Council on Systems Engineering (INCOSE 98), pp 445-452, Vancouver BC. Canada, Jul 1998.

[10] Dalton J.A., Norman P.W., Whittle S., and Rajabally T.E., “Using an MDA approach to model traceability within a modelling framework”, Metamodelling for MDA, Kings Manor, York, England, November 24-25 2003.

[11] Mellor S.J., and Balcer M.J., “Executable UML: A foundation for model-driven architecture”, Addison-Wesley, 2002.

[12] Chandrasekaran B., “Design problem solving: a task analysis. American Association for Artificial Intelligence”, 1990.

[13] Dalton J.A., Norman P.W., Sen P., and Whittle S., “Use of modelling frameworks in the design of complex engineering systems”, Engineering Design Conference (EDC 02), King's College, London, UK, 2002.

[14] Pérez A.G., and Benjamins V.R., “Overview of knowledge sharing and reuse components: ontologies and problem solving methods”, The IJCAI-99 workshop on ontologies and PSM’s (KRR5) Stockholm, Sweden, August 2,1999.

[15] Swartout B., Patil R., Knight K., and Russ T., “Toward distributed use of large-scale ontologies”, 10th Knowledge Acquisition for Knowledge Based Systems Workshop (KAW 96), Nov 9-14, Banff, Alberta, Canada, 1996.

[16] Benjamins V.R., Fensel D., and Straatman R., “Assumptions of problem-solving methods and their role in knowledge engineering”, ECAI 96. 12th European Conference on Artificial Intelligence, 1996.

[17] Ahmed S., and Wallace K.M., ”Identifying and supporting knowledge needs of novice designers within the aerospace industry”, Journal of Engineering Design, 15, 3, 2004.

[18] Jarratt T.A.W., Eckert C.M., Weeks R., and Clarkson P.J., “Environmental legislation as a driver of design”, International Conference on Engineering Design, ICED 03, Stockholm, August 19-21, 2003.

Corresponding author:
John Dalton
Newcastle Engineering Design Centre
Stephenson Building
University of Newcastle
Newcastle upon Tyne NE1 7RU
United Kingdom
Tel. Int +44(0)191 222 8556
Fax Int. +44(0)191 261 6059
E-mail: John.Dalton@ncl.ac.uk
URL: www.edc.ncl.ac.uk