Design: the Quintessential Business Transaction

C. Douglas R. Muir

Department of Mechanical Engineering
James Watt Building
University of Glasgow
Scotland

Submitted to the University of Glasgow for the degree of Doctor of Philosophy

January 1999

In memory of Calum Muir who was an inspiration both as a world class scientist and as an outstanding father.

The fear of the Lord is the beginning of knowledge.

Proverbs 1:7

Abstract

The fundamental structures that underpin business activities must evolve and change in order to equip companies to thrive in a market whose characteristics are increasing competition and instability. The incremental advances in applied computing technology and business methodologies which focus on improving one aspect of company operations ignore the need for an underlying structure and model through which to engage any and all functions in a consistent and integrated fashion. Indeed, many exacerbate the problem through closed architectures, isolationist views of entity data storage and rigid methodologies imposed on the company that employs them.

The Product Model proposed fulfils that role. It is a model of the processes and entities that a company uses to conduct its business, at all levels and across all departments. Two other concepts are exposed: product model data and the design history record. Product model data are the values of instances of product model entities and relations, created to represent a particular design, artefact or object. The design history record captures the data and functions used in a transaction and the order and context in which they are used.

To exercise these concepts, a software suite was written, the Glasgow utility for Integrated Design, Guide. It supports the definition of a proud model and its subsequent use in the creation of product model data. Each interaction with the system is recorded, thus capturing the design history record, which can subsequently be processes to various advantageous ends. The major such uses are for re-use of part information in other designs and the extraction of design best practice with which to augment the company's design methodology. It is a comprehensive record, since all business processes are supported by, and can be transacted through Guide.

Guide has been used to validate the adequacy of the product model and has established many benefits through its use. Applications in many spheres are possible; engineering has been the primary focus for exemplars and case studies. The development was carried out under the scrutiny of constant validation and testing in live situations with several industrial partners. Guide is built on industry standard tools and uses relational database technology to store frame-based representations of entities, methods and relationships.

The design of project plans is carried out on the same platform used to support the project itself; the design data are not dissociated from the project controlling mechanism. Resources, including staff, are engaged according to requirements and audit mechanisms allow for constant re-evaluation of the project development. Control and

communication mechanisms support applications in an extended enterprise environment and the distribution of resources that this entails.

The system user is supported by methods that offer processing and data retrieval capacities within a single environment. Constraints safeguard the integrity of a design solution and bound the design domain. Process recursion is encouraged and flexible.

The system allows for legacy systems and data to be accessed. Its implementation can be rolled out across the company and it can work as a standalone utility for a particular function or as a whole business support tool.

Several test cases were constructed. Two are described here,

- A utility for design of ceramic-reinforced ceramic components
- Management of product lifecycle and information, engineering change and interface with sub-contractors.

The concepts expounded in this work represent a departure from traditional methods that try to expand the knowledge over which a particular entity has control. Instead the approach to the problem with a fundamental solution, which allows a unitary approach to modelling and supporting a company's business entity and process requirements. This is done without trying to impose a grand definition schema, such as PDES; users are free to define their own entities, although they might find the formalism and definitions of the PDES standard useful to them in this task.

Guide provides a methodology for design without prescribing the path to take through the design domain. It assists the designer's creative functions by removing some of the burdens of processing and information gathering.

Acknowledgements

Thanks to Professor Brian Scott for his rapid and helpful review of my work, his exacting standards and continuous encouragement.

This work was funded through a number of IBM grants. Two grants were awarded from IBM ISMD, Warwick. The first of these projects ran in the Engineering Design Research Centre in Glasgow. These were followed by a Shared University Research programme, jointly funded by IBM Poughkeepsie and IBM Greenock. This was the first such programme to be awarded to an academic institution outside the United States.

IBM was a model sponsor, providing research freedom, participation of management to a high level and a beneficial industrial view of the problems addressed. They opened their processes and data to scrutiny without precondition or qualification; this provided invaluable insights into the reality of engineering practice at IBM. Particular credit is due to Tony Munton of IBM ISMD and Alan Crombie of IBM Greenock for their constant support and applications for funding.

Within the department, Andreas Tsiotsias managed these projects, a task that he approached with professionalism and energy. Walter Robinson provided programming and systems support to a high level. Ian Hopper developed the Ceramics Design System and made the results of his research available.

My wife Fiona has provided constant support, encouragement and goading, each in good measure. She has also washed more dishes that I could count and has probably done another thousand things to make my life easier that I did not even notice. Thank you.

Contents

Figures

Tables

Plates

Definitions

The terms defined below are highlighted in bold italic typeface where they first occur in the text. They are organised according to the topic to which they appertain.

Guide-related:

Structure:	Entity representation knowledge construct.
Child structure:	A structure closely bound to its parent, used to characterise and simplify the parent structure.
Atom:	Attribute values of a structure.
Activity:	A particular type of structure; in addition to its own definition it is a container for a unit of work.
Element:	Any Guide component – structure, atom or method.
Method	Executable software program, which may act on its own or as a constraint.
Instance:	A particular occurrence of the generic definition, ascribed with attribute values relevant to the context of its use. Instances may be made of any Guide element.
Product model:	The set of Guide elements that model the business processes and entities used by a company.
Product model data:	The instances of Guide elements that result from exercise of the product model to prosecute business aims.
Design history record:	A chronological log of all of the interactions with the product model.
Compound variable:	An aggregate of values of pre-defined data types. The programming term for a compound variable is a structure; this term is not used to avoid confusion with Guide structures.

Variable stack: A short-term storage repository for compound variables that control the flow of Guide operations ensuring that transactions are properly completed. Values added to and removed from the stack on a last in, first out basis.

Design:

Function: The outcome of engaging one or more processes to meet a business operational requirement.

Process: A definition of the outcomes which must be achieved to implement a function. It does not prescribe the method of execution.

Procedure: An ordered set of tasks which defines the mechanism for accomplishing a process within a particular organisation.

Task: A unit of work.

Design domain: The sphere of knowledge pertaining to the artefact which is the object of the design process

PDES: Product data exchange using STEP (STandard for the Exchange of Product model data). An international initiative to define a set of entities to permit the communication of any product attribute in an agreed neutral format.

IBIS: Issue-Based information system. A system where problems are resolved by choosing one of the proposed solutions to the issue under consideration.

Logical schema: The representational model for knowledge, independent of storage medium considerations.

Knowledge: Data and their interrelationships.

Information: The value of knowledge.

IBM information system:

PIE: Product Information Exchange, a classified list of products previously designed by the company. Its purpose is to avoid re-design of products that are already available.

DPRS: Development and Product Record System, a product data repository that holds bill of materials and engineering change level information.

Computing:

API: An Application Programming Interface is a set of routines that allow a third party developer to access and utilise the functionality of a software package.

GraPhigs: A language for the definition and manipulation of geometric elements and the creation of simple user interfaces.

OLE: Object Linking and Embedding, a Microsoft protocol allowing certain objects to be shared between applications.

CORBA: Common Object Request Broker Architecture, a common object definition model, independent of application software that employs it.

1. The Product Model and its application using Guide

1.1 Introduction

The primary concepts dealt with in this work are the ***product model***, ***product model data***, business ***processes*** and the ***design history record***. Though the context in which these concepts are developed is engineering design, they can be applied to any business. Design is taken in its broadest sense; Dixon [1.1] proposes a useful definition, although the word ***knowledge*** should be substituted for ***information***.

> *A design (noun) is a state of information. The information may be in one or more forms: words, graphics, electronic data, and/or others. It may be partial or complete, it ranges from a small amount of highly abstract information early in the design process to a very large amount of detailed information later in the process sufficient to perform manufacturing. It may include, but is not limited to, information about size and shape, function, materials, marketing, simulated performance, manufacturing processes, tolerances, and more. Indeed, any and all information relevant to the physical or economic life of a designed object is a part of its design.*

The design process is the generation, use and maintenance of design knowledge. All disciplines in the company that have an impact on the design process and the resultant artefacts must be included. Accordingly, the environment in which designers operate is multidisciplinary [1.2]; the necessity to be inclusive when defining design is further recognised by Shaw et al. [1.3].

This chapter introduces the product model and explains its ability to support a company's entire design activity. Guide, the software utility written to perform these activities is described. Chapter two establishes the business requirements for the product model. Chapter three details the operation of Guide utilised in design and the benefits gained by its exercise. Chapter four describes its architecture and method of operation. Two examples of applications developed using Guide services are described in chapter five. The work concludes with indications for future directions and research topics.

This work was conducted as part of a larger research project in the Design and Information Systems Group of the Department of Mechanical Engineering at the University of Glasgow. It builds upon the work described by Tsiotsias in his PhD thesis [1.4], which describes the underpinning enabling technologies and introduces the design methodologies which are further developed and applied in this work.

1.2 Design and the product model

The product model is the template a company employs to represent its products, enabling their design, manufacture and maintenance. It serves the company's business needs and aspirations and supports its strategic policies. It expresses the methodology with which a company engages in design in terms of systems, rules, entities and relationships. It is equally applicable to service and manufacturing organisations. For a particular company, it will operate in one market area, which will determine the technological area or ***design domain*** in which design activities take place.

It should constitute a formalisation of the natural and logical relationships that exist within the product and describe it, together with the activities that are instrumental in its specification, definition and production. In other words, the structure of the company as well as the results of the actions it performs should be represented. The product model is dynamic in nature and evolves; changes to its structure and content must be managed.

The elements of the product model are (Figure 1.1):

- the design processes engaged,
- constraints applicable to the design,
- the definitions of entities used in the processes,
- relationships which exist in the design domain.

Design processes are all the actions taken in the research, specification, design, manufacture, documentation and maintenance of the product. They may be directly related to the particular product, such as engineering design calculations on a part or piece of geometry. They may also be related to the design process in general, such as project management. A further category of process is that of support functions, such as data storage and retrieval routines.

Constraints define the bounds of the design domain. The origins of the constraints may be company policy, relevant international standards or the laws of nature. Constraints act in a positive way to secure the validity of the design solution and are not an impediment to design, as the word constraint would suggest in common English usage.

The large number of parameters which must be dealt with during design are beyond the capabilities of one person to remember and manage at any one time. Decisions are often made based on the engineer's limited experience and prejudice. Design decisions are then revisited only if there is a downstream failure [1.4]. Effectively, designers take a chance that their educated guess will work out correctly. Guide enforces constraints

early after any design decision, enables evaluations and offers utilities to supply valid values into entity parameters.

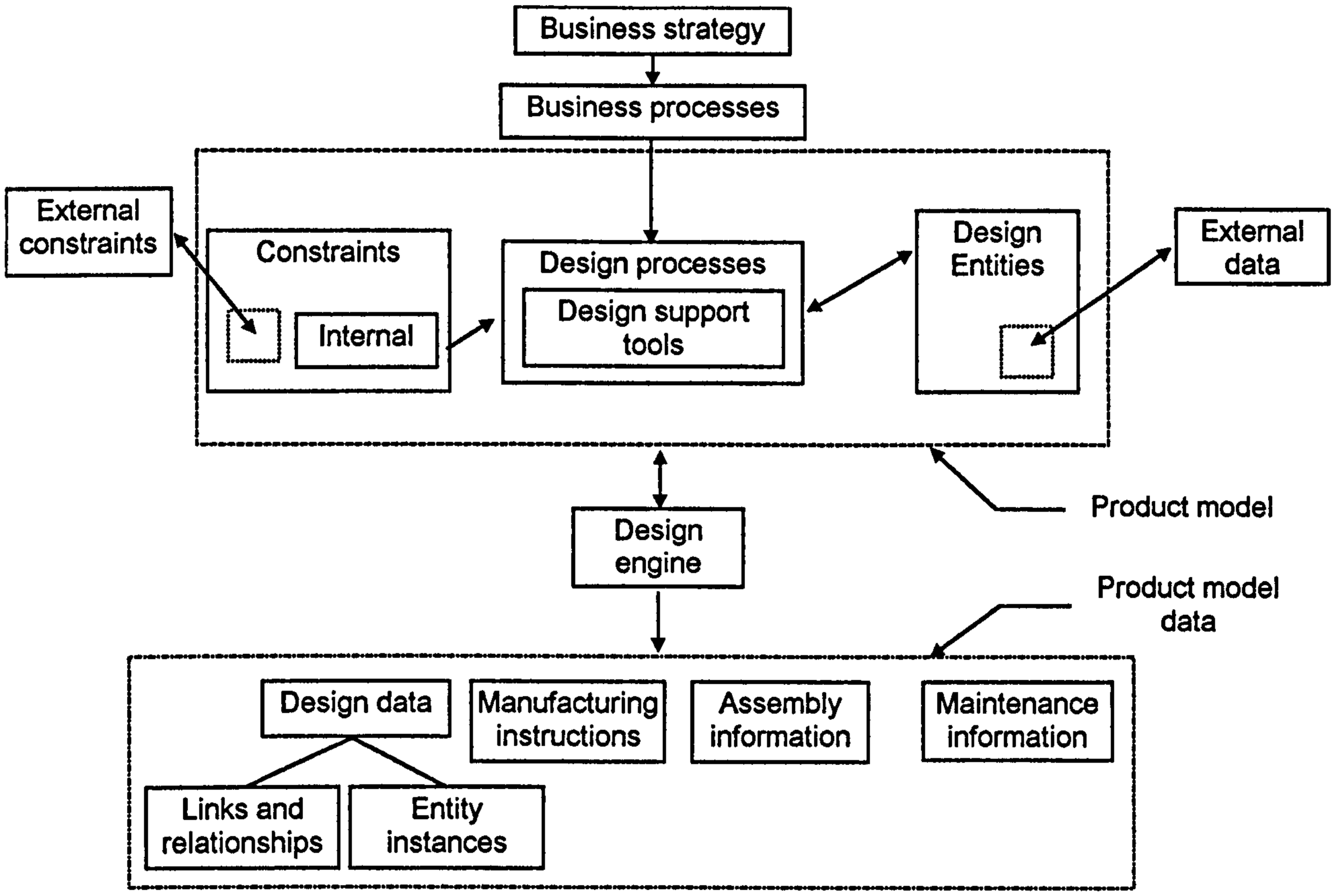


Figure 1.1 - Product model structure and dependent product model data

Product model entities are used as templates for the storage of information relating to a design project. The entities can represent abstract concepts such as a bill of materials or an invoice, or physical entities such as a part or cutting tool. The structure and definition of certain entities is affected by the software utilities which support design processes and which impose their representation structure on the entity definition.

Natural relationships exist between entities in the design domain. For example, the manufacture of a hole requires a drill of the same diameter. If a reamer is used for finishing, then the drill diameter is smaller and is calculated by a formula. The relationship may be simple or depend upon a complex algorithm that includes constraints.

The external data sources and constraints shown in Figure 1.1 are defined in the product model as a reference to a resource. These sources are not physically stored within the product model, but are located, wherever they are distributed, by a pointer.

Software utilities such as CAD systems and project management programs are used to support some of the processes. The particular utilities employed impose their structure on the product model locally to their area of influence, according to their specific problem solving approach. The danger is that the company adopts mechanisms imposed by the software which are sub-optimal, on the basis that it cannot change the software tool, but it can adapt its ***procedures*** to accommodate it.

Eastman [1.6] offers a description of the application of the product model:

> *Design, in the information modelling sense, consists of defining variables and recognising and applying constraints between the variables, including equality constraints, e.g. bindings, until a consistent set of variables, constraints and values have been defined, corresponding to the problem at hand.*

This fails to mention explicitly the processes that drive the design and ensure its completion. Cleetus [1.7] similarly fails to extend his proposed Unified Product Data Model (UPDM) to process definition and modelling, as do Majumder and Fulton with their design model [1.8]. The lack of agreed terminology in this area of research is marked and every author uses the term product model to mean different things, or calls it by different names.

The product model is differentiated from a product description structure that might, for example, comprise a bill of materials, part and assembly drawings and a set of manufacturing instructions. Such a product description template is constructed so that it contains enough information about any specific project to enable the manufacture of the product by any company competent in the particular manufacturing area. It does not include the product design information, design project management constructs or constraints.

1.2.1 Product model data

The product model data are those data created as a result of the exercise, by designers and other business functions, of the product model in the creation of a particular product. They are ***instances*** of the entities defined to the product model. They include the description of the product or service, and the information necessary to produce and maintain it.

The product model data will, by virtue of their parentage in the product model, represent information from all stages of the design process. The effect of storing product model data according to the disciplined structure of the product model is to make them accessible to concurrent and subsequent processes, including recursion of these

processes. Information need no longer be confined to sketches and notes in the workbook of a designer, accessible only to them. The type of change and version control usually applied to part data after release to production can be applied to all product model data [1.9]. Consistency and accuracy of the data can be enforced to an appropriate level on everything from assembly instructions to initial concept sketches.

The product model is dynamic and accommodates changes in design and business processes a company will introduce as part of its own evolution. Since the product model data are subservient to the product model, uncontrolled and undocumented changes to the product model may result in them being orphaned and left without semantics. Just as there exist change control procedures for the component parts of a product, so the evolution of the product model must be controlled and recorded [1.10]. The change control procedure is more rigorous than for components, in that every successive version of the product model must be recorded, not just the last one; this ensures the persistence of and preserves the value of those product model data created under each version. The match of product model data to the product model version under which they were created is achieved through the timestamp that all entities and their instances carry.

Cleetus [1.7] does not make this distinction between the product model and its consequent data. This increases the complexity of his schema and results in him proposing several UPDMs over the course of the product development lifecycle (e.g. requirements, visualisation of the product, architectural) as well as a final full and complete model. In effect, he advocates the freezing of the UPDM at different stages and identifying each stage separately. The resulting system does not achieve his stated intention of having a global and unifying model that is able to record changes over time. The re-use of entities from one design project to the next is also left unclear.

Product model data are often considered in isolation from the product model [1.3]. This is a mistake, as it divorces the specific from the underlying generic definition. The full definition of the entity must then be carried with each instance, since there is no link back to the template from which the instance was created. Information on the particular set of processes and constraints active at the time of instantiation is also unavailable.

1.3 Guide and the product model

The Glasgow Utility for Integrated DEsign (Guide) provides a framework and discipline for the construction and use of the product model. It constitutes a vehicle to facilitate the conduct and administration of design and allied business functions.

The facilities offered by Guide to support the use of the product model include an environment for the execution of design processes and *tasks*, storage of the resulting product model data and communications necessary during the design process. It is not a design automation engine, although it does support the automation of design processes.

Guide sustains design processes within the constraints of the companies in which it is utilised. It does not impose on the company a design methodology or prescription of data and knowledge representation formats. The unpredictable actions that occur between designers and between disciplines [1.11] are fully supported. The degree of constraint applied to the activities of the designer is a business or project management decision, not a system limitation: where necessary and desirable, it promotes the evolution of the company's design practices in a disciplined fashion.

Every company utilises a product model, by definition. Few, however, are aware of it as a concept or maintain it in a comprehensive and documented manner. An initial benefit of a Guide implementation is that it will cause the company to define what is best practice in their circumstance. This is in contrast to practices predicated by the capabilities of their current software panoply. Guide will then support whatever schema the company establishes for its new design process, including changes made to extend and improve their design procedures and capabilities. One of the advantages of IT implementations generally is their disruptive potential, by opening previously impossible ways of working [1.12]. This can be used as a powerful vehicle for system and cultural changes in a company.

1.3.1 Guide components

A schematic representation of the interactions between the product model, product model data and Guide is shown in Figure 1.2; its elements are described below.

The Guide engine provides those processing and control functions necessary for the system operation. It has two main categories of function: performing system core functions and executing any design processes invoked by the user.

The user interface allows for the display of and interaction with the product model data and the underpinning product model. These may be of any form - textual, numeric or graphical. Since all Guide functions are available from every workstation, the user requires access to one workstation alone. The contrast is with dependence on disparate systems, some of which may reside only on particular workstations: engineers must go to the data, rather than receive the data on demand.

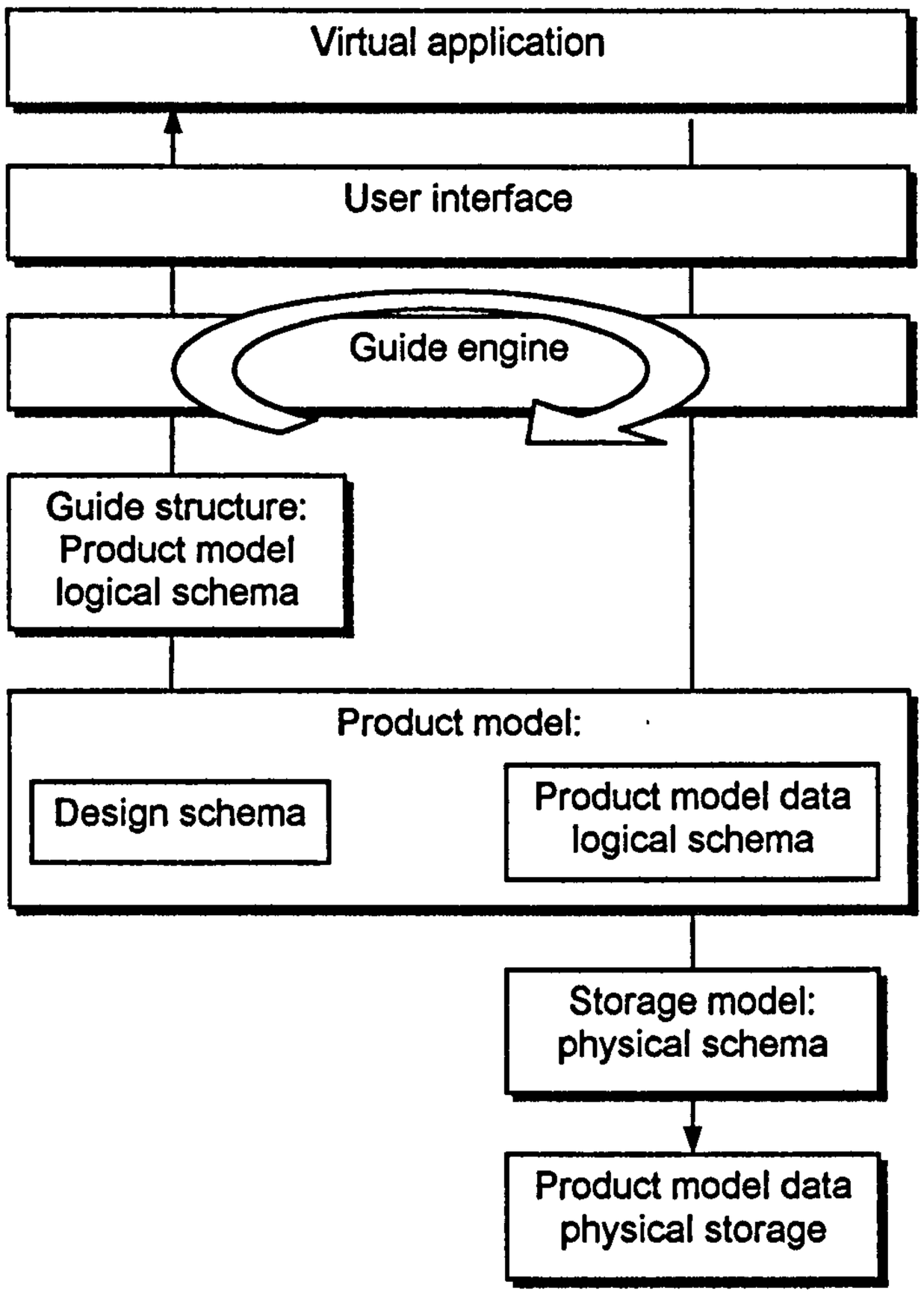


Figure 1.2 - Guide operation model

The "virtual application" is so called because the product model entities and data on which it operates define it. There is not a conventional user interface, with a limited set of pre-defined functions, each of which will cause the execution of particular processes. The appearance and function of the user interface are controlled by the product model entities and processes accessed by the user while using the system. Any valid process can be invoked at any stage of the interaction with Guide. It is analogous to a web browser, which is a window onto the data and processes fed to it. This results in applications that are intimately tailored to the user's requirements, the direct expressions of the company's process definitions.

The product model defined to and stored by Guide acts in two capacities. The first is to provide the basis for the virtual application in the supply of design processes and the entities with which the users operate. The second is as a template for the product model data arising from and reflecting the exercise of the design process. The product model data have a physical location controlled by a physical schema. The storage locations may be, and most probably are, distributed amongst systems internal and external to the company.

The Guide structure is the ***logical schema*** for the software, containing the internal architecture model on which the engine operates. It is the core of Guide, and contains the mechanisms that allow it to operate (Figure 1.3).

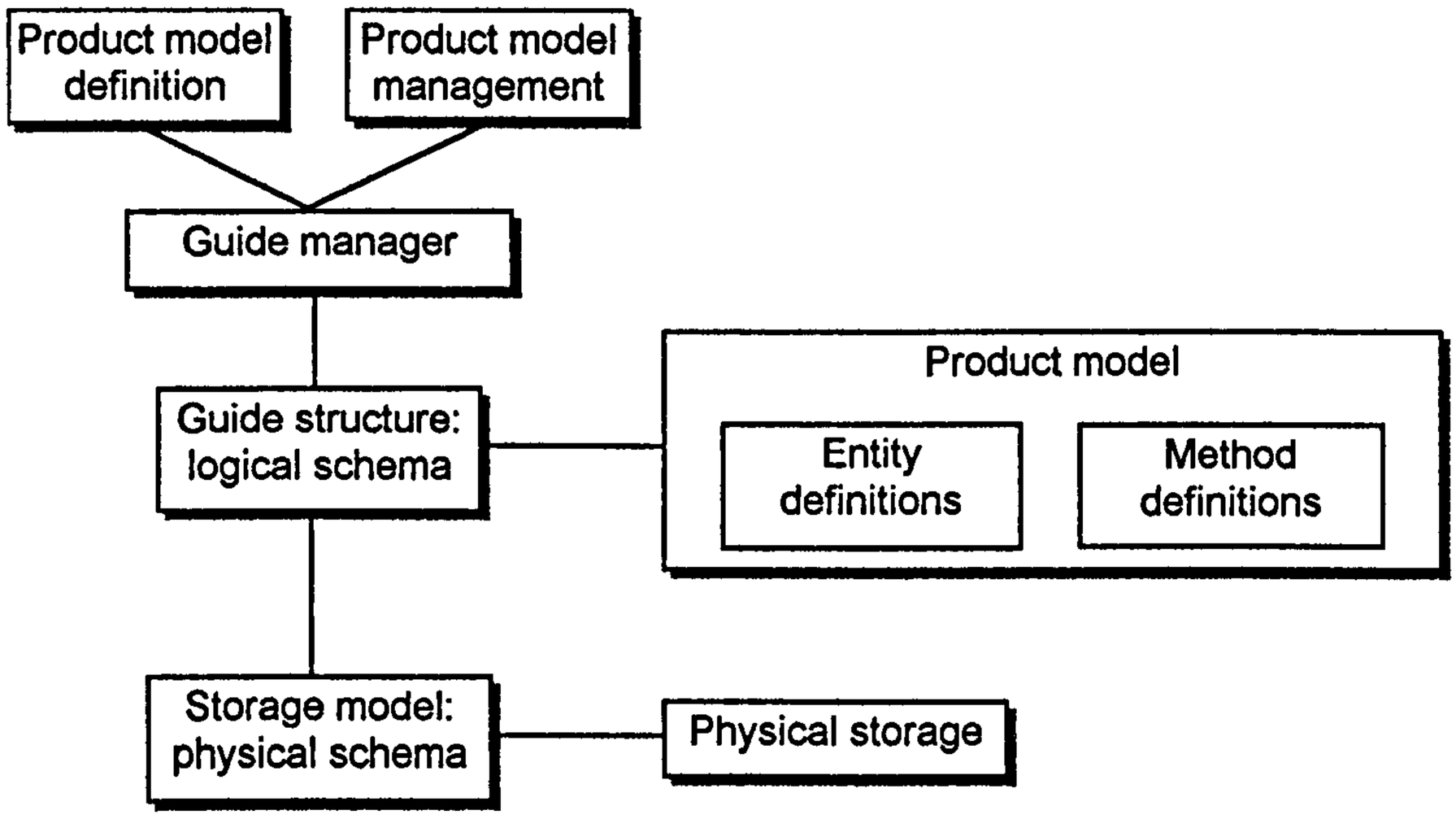


Figure 1.3 - Guide product model definition and maintenance

Guide employs frame-based representations [1.13] to store product model entities, ***methods*** and relationships. The Guide Manager utility proffers an interface via which users can define and maintain the product model. Frames and frame representation are acknowledged to advance considerably the ability to model objects and knowledge in a fashion open to design actors and agents [1.14]. They provide Guide with a platform powerful enough to represent the range and variety of entities that comprise the product model.

The physical storage of the product model is independent of the product model data and is controlled by a separate physical schema. There is also a requirement for the ability to distribute the product model storage. Though the information may be kept in one place, different sectors of the company may need to access the product model as a remote storage location.

1.4 Guide activities and design management

Part of the significance of the product model is that the same methodology used to design products is used to design the design project. The design project is a service to the company and a process like any other. As such, it requires comprehensive support if it is to function in an optimal manner. Finger, Gardner and Subrahmanian [1.15] recognise the need for a design system able to support concurrent engineering (CE) to

manage both the agents (people, computer programs) active in the design process and data, information and knowledge. Guide activities provide this facility and are the key elements in the management of the design process.

It is important that designers are able to set their work in the context of the overall business objectives of the company, since, in large measure, they affect profitability. When evaluating concepts, they must be able to make decisions in the context of the company's business aspirations [1.16]. Guide's ability to represent information and knowledge at all levels according to a common framework enables this to happen.

1.4.1 Activities

A Guide ***activity*** bounds a unit of work. It is a container for the product model data resulting from work done in the context of the activity. There are no *a priori* prescriptions as to the nature of the work, or the extent and size of the activity.

An activity is a particular type of entity, with attributes that characterise and describe it. Additionally, and in contrast to other Guide entities, it has a repository linked to it, which contains all the product model data generated while the designer is working in that activity. In Figure 1.4, a design project to design a bracket is initiated. This spawns an activity tree, the purpose of which is to provide a solution to the top-level activity.

Activities can be linked in three ways:

1. One of the attributes of an activity can be a child activity. In order to complete the parent activity, the child activity must be completed. Because in this example the bracket is in a safety-critical load bearing application, finite element analysis is mandatory under the relevant safety standard and is therefore defined as a child activity.

2. The association of a linked activity also occurs at the entity attribute level. A pointer type attribute can accept links to one or more other entities, including activities. The type of the linked entity is not pre-defined. In order to determine the manufacturing instructions, the capabilities of a number of potential sites are evaluated. The specific sites are not pre-defined, nor are the evaluation criteria.

3. A sub-activity is one that is initiated from within another. This is the most common type of linkage. In Figure 1.4, Part geometry, Manufacturing instructions, Material selection, Geometry definition and Geometry meshing are all sub-activities.

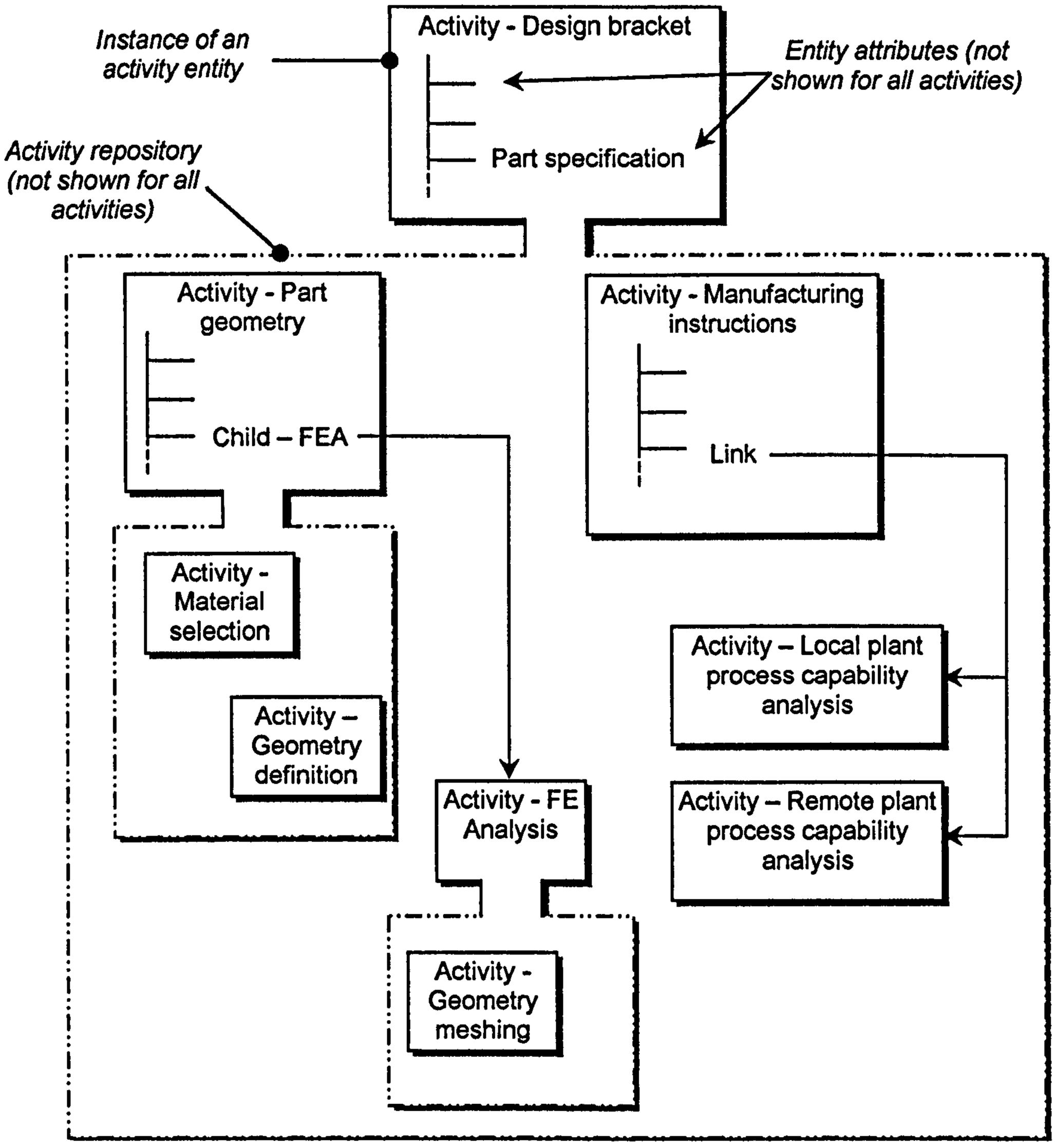


Figure 1.4 - Fragment of design project activity tree

Child activities can themselves spawn linked activities. This is another illustration of the flexibility of Guide: whilst it does not impose a method on the finite element analysis, it offers the facility explicitly to constrain the user to a particular solution method if desired.

When an activity is initiated (Plate 1, Plate 2), the initiator becomes the activity owner (Figure 1.5). The activity is assigned to an actor responsible for the execution of the tasks necessary to complete that activity (Plate 3). The actor may be another individual, a group or, indeed, the initiator. As part of the work on the activity, the actor may spawn sub-activities which, in effect, sub-contract aspects of the problem solving. The originator of such a new activity may assign it back to himself, so helping him to manage

the task so assigned. The completion of the activity requires the entire completion of the dependant activities. When the activity together with any child or linked activities is complete, it is returned to the owner. The owner then inspects the activity, and either accepts it, changing its status from complete to closed, or re-issues it for further work.

The activity may be re-issued to the same person, or assigned to another person for further work. The owner of an activity may examine its contents in read-only mode at any time and request changes of the responsible engineer. The owner may not intervene whilst the activity is in progress but must wait until the responsible person has marked the activity as complete. Activities may be suspended at any stage: the person might log off the system at the end of the day, or may require to work on another activity for a time. When a user needs to work on a child activity, the parent activity must be temporarily suspended.

Activities allow a task to be organised into sub-activities of manageable size. One or more tasks, encapsulated in activities, can be initiated from within their parent activity. Specific aspects of parent tasks can thereby to be granted individual attention. This does not involve a separation of the sub-task from the rest of the design project; the activity remains linked to the rest of the design project through its parent. The relationship between parent and child activities is maintained explicitly, and may therefore be used to navigate between activities. This renders the contents of one activity accessible to processes invoked in a separate activity.

1.4.1.1 Complex task management

Children activities can be issued to an authority appropriate for operation on the particular problem, irrespective of the domain of the parent task. The higher levels of the activity tree act as specification for lower levels, against which the sub-activities work, either independently or in co-operation. This construction provides control of the process and imparts the ability to deploy appropriate expertise at the detail level.

The term decomposition, taken in the context of design activity management, implies a splitting and division of the work. Decomposition carries the risk that data, relationships or meaning derived from the original task may be lost in the transmission to sub-tasks. Guide, through the structure it imposes, does not acquire these risks.

The model in Figure 1.4 shows the interaction between activities. It is important that a system for management of complex tasks does not incur losses in the information content through task decomposition. This is important, since any sub-contracting of a task must not result in decoupling and losses in information content; instead, the individual tasks must relate to each other [1.17], [1.11].

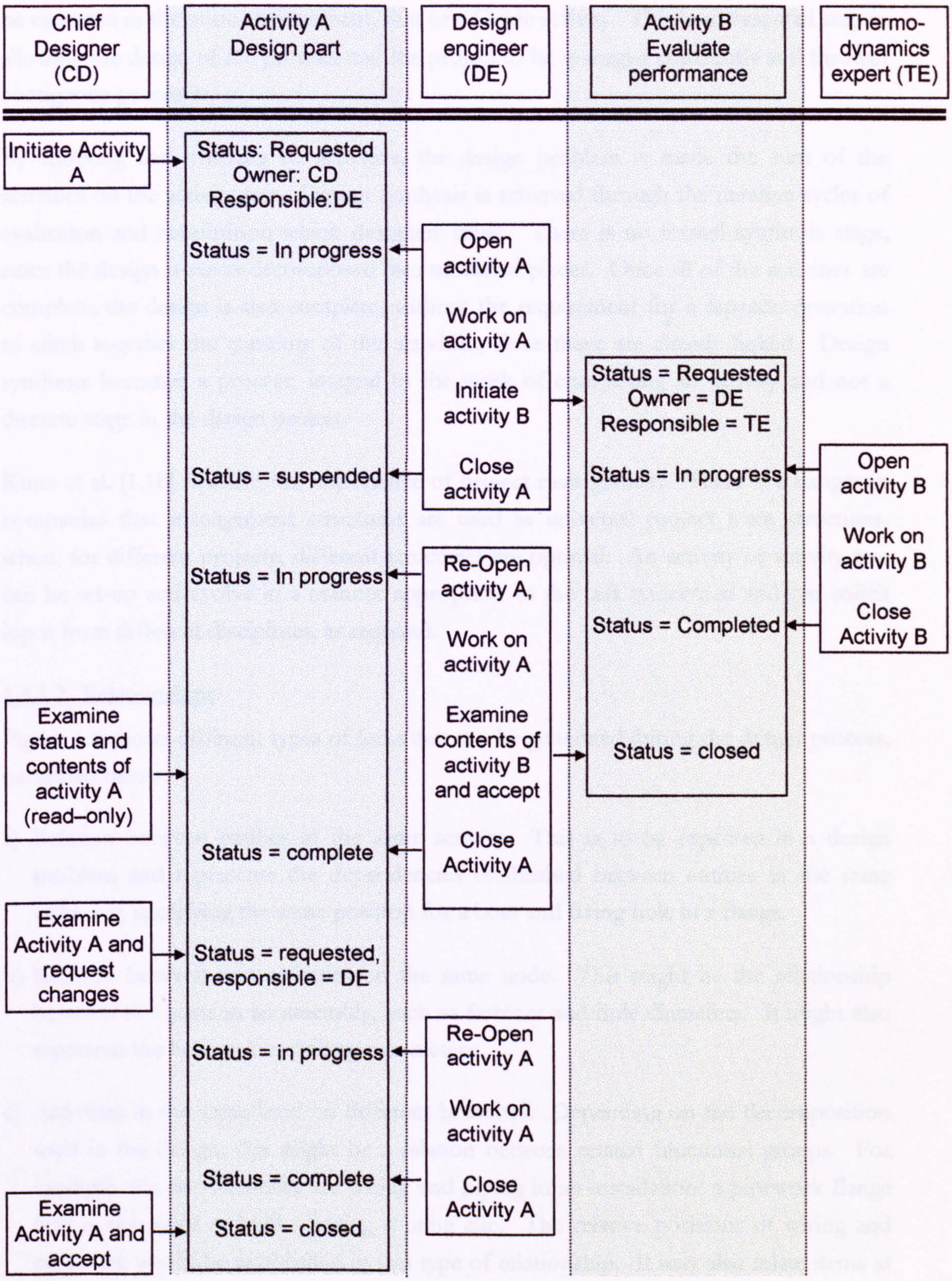


Figure 1.5 - Activity ownership and control during a project

The process of spawning sub-activities can take place to any number of levels. By offering this possibility of recursion, all the processes involved in the design project can

be executed in the same environment, that of a Guide activity. This is an essential step in allowing the design of the product and the project to be managed coherently and through contiguous processes.

By adopting the structure of activities, the design problem is made the sum of the activities on the activity tree. Design synthesis is achieved through the iterative cycles of evaluation and redefinition which designers cause. There is no formal synthesis stage, since the design is never decomposed into unrelated pieces. Once all of the activities are complete, the design is also complete, without the requirement for a separate operation to stitch together the contents of the activities, since these are already linked. Design synthesis becomes a process, integral to the work of completing an activity and not a discrete stage in the design project.

Kunz et al. [1.18] reaffirm the importance of project management. There is a danger in companies that management structures are used as universal project team structures, when, for different projects, different structures are optimal. An activity or activity tree can be set-up and evolve in a manner appropriate to the task concerned and can solicit input from different disciplines, as required.

1.4.1.2 Relationships

Figure 1.6 shows different types of links that can be generated during the design process, described below.

a) Relation between entities in the same activity. This is to be expected in a design problem and represents the dependencies established between entities in the same tasks, e.g. specifying the same position for a boss and fixing hole in a flange.

b) Relation between two activities on the same node. This might be the relationship between two parts in an assembly, such as fastener and hole diameters. It might also represent the hole and its drilling parameters.

c) Activities at the same level on different branches. Depending on the decomposition used in the design, this might be a relation between related functional groups. For example, the two branches are wiring and piping in an installation; a pipework flange bolt is also used to hold a wiring routing clip. The relative positions of wiring and pipework would be established in this type of relationship. It may also relate items at different levels of granularity: in an activity branch representing a simple object, the detail levels will be reached much sooner than in the case of a complex object.

d) Activities on different branches and at different levels. This is the same as c), and may relate entities at similar or different levels of granularity.

e) Link to an entity in the parent activity. Links to the specification, or design entities in the parent entity, such as would be the case if, say, the wiring and fuel lines for a car were established, based on the chassis and bodywork geometries held in the parent activity. Negotiation as to the specification is also possible and a parent activity can adapt and evolve based on feedback from its dependants.

f) Link to a different activity tree. For example, this could relate a machining process to a machine tool capability, or link to a standard part dimension.

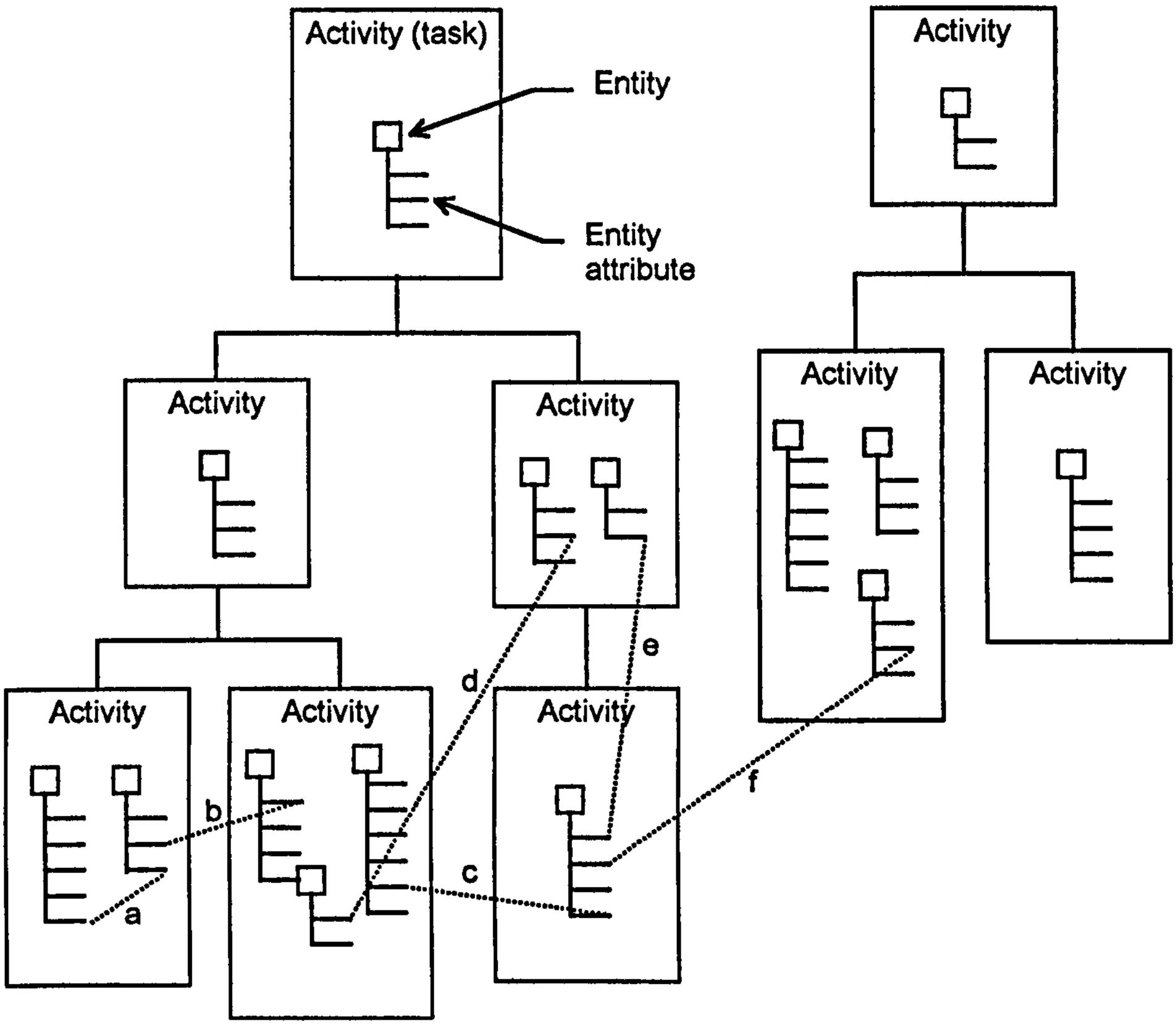


Figure 1.6 - Relationship types between design entities

Solid lines are, in general, control links; dotted lines are problem-solving. It is also possible to refer to a whole activity with all its dependants, to adopt it into the current activity tree. In this way, previously executed work can be integrated seamlessly into the current task.

Guide is therefore able to support all types of tasks - dependent down the depth of the tree, independent as nodes on a branch between which there are no links, and interdependent as links between nodes on a branch.

This is important from a task completion standpoint and allows an overall task to be completed by a combination of problem solving, sub-contracting and collaboration. It also provides a vehicle for the application of local expertise to solve a part of a larger problem. This approach is also utilised in the ACME system of D'Ambrosio et al., which concentrates on the protocols necessary to optimise a solution, given the preferences and constraints of each of the actors [1.19].

1.4.1.3 Navigation of product model data

Navigation across the product model data is made possible by the way in which links are stored explicitly as an integral part of the product model data. These links can be followed across the product model data, subject to the user having authority and clearance to do so. This facility can be used for the retrieval of sets of information, and renders separately maintained bills of materials obsolete. It is now possible to have a bill of materials structure that is a filtered image of the rest of the product model data. Links can be followed for the purposes of finding a related value, in propagating change through the system and in retrieving data. The benefits arising from the ability to navigate the design data in this way are recognised by Goodwin and Chung [1.20] and Urban et al. [1.2].

1.5 Product model data access

The product model and product model data offer the company a cohesive representation of the design project and the product emerging therefrom. Different agents participating in the project will have information requirements peculiar to their needs and areas of influence. They need rapid and easy access to the product model entities and product model data that they use in their transactions within the project. The information must be made available to them in a coherent and intelligible manner [1.8]. A product model and product model data access filter mechanism allows the relevant information to be presented to the user, without affecting the integrity of the product model and its data (Figure 1.7). While only the data of interest and relevance to the user will be visible to them, these remain part of the product model data and maintain all the links and relationships of that structure. The access filter presents a view of the product model and product model data, rather than a segregation of one section of information. Hardwick and Downie [1.21] agree on the requirement for access to product model data by different agents, but take the concept only as far as allowing various computing tools access to it.

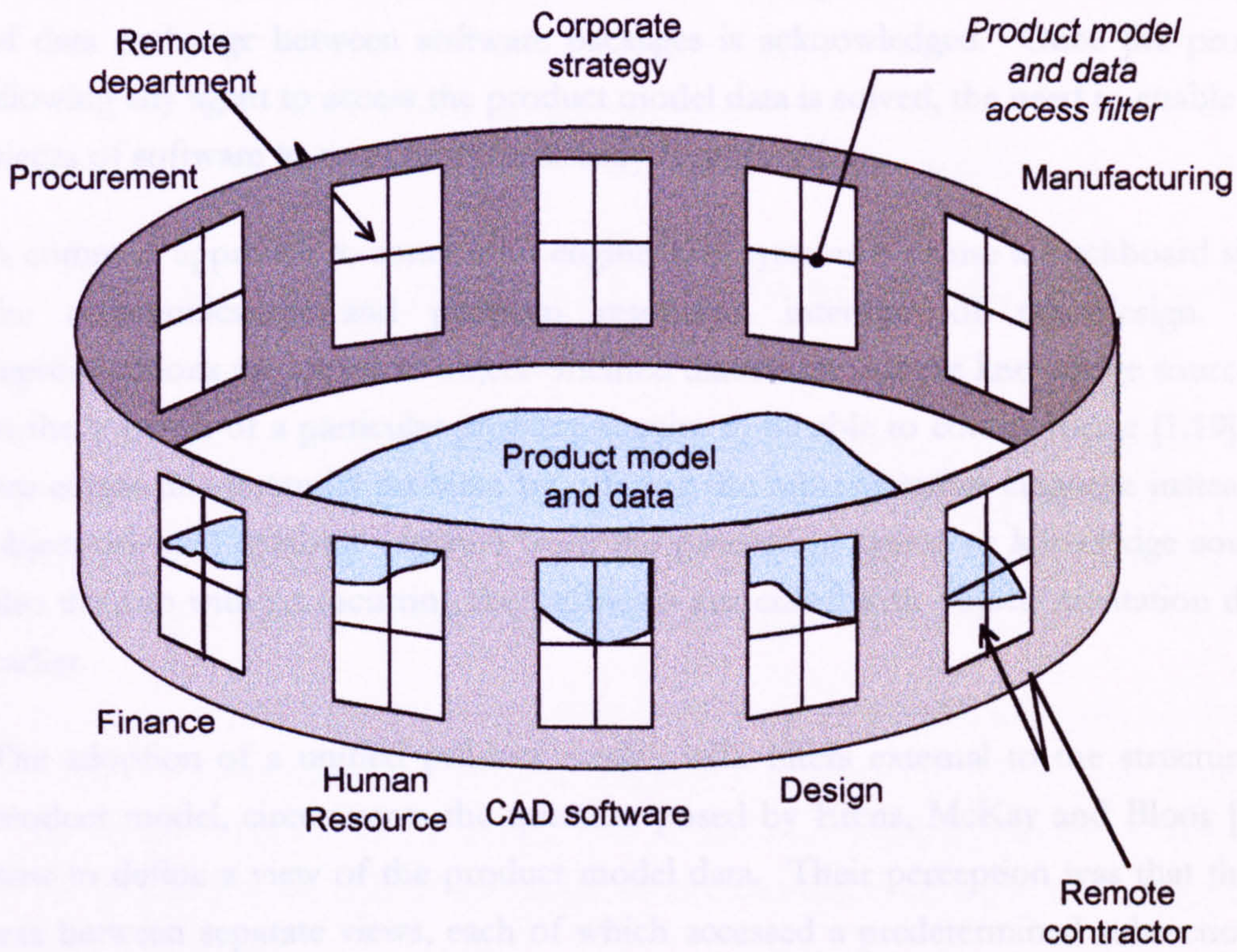


Figure 1.7 - Filtered perspectives of the product model

Users occasionally require information beyond their normal scope. This is accessed by following the explicit links to related entities in the rest of the product model. Navigation through the wider entity set is via the common interface and does not require additional procedures. This is in contrast to product models that are distributed amongst collections of systems and repositories, without explicit links or navigation paths between related items. Retrieval of related information then requires searches in all affected repositories to find a set of information bound by an identification key, such as a part or reference number.

Transparency of access to data from other areas is important, and allows the design to be built up from its foundations by all parties concerned. The structure can be solidified as constraints are applied, unsettled matters resolved and details finalised. Brissaud and Garro [1.22] express it as: *a common-sense* (co-ordinated solution) *to the different modules develops through the information exchanges,* which reinforces the necessity for collaboration and interactive design operations.

Software utilities deployed in support of the design process also demand access to part of the product model data, and invariably in a proprietary format. If the data are stored in a different form, the access filter for an application can include the translation function. This allows data stored in any format to be accessed by several different applications. The

data, meantime, remain connected to the rest of the product model data. The problem of data exchange between software packages is acknowledged. Once the problem of allowing any agent to access the product model data is solved, the need to enable any two pieces of software to communicate directly is removed.

A common approach to concurrent engineering systems is to use a blackboard system as the communication and problem resolution interface of the design. Entity representations are stored in object-oriented databases. All the knowledge sources active in the solution of a particular problem require to be able to communicate [1.19]. Guide pre-empts this potential problem by offering the representation language instead of the object-oriented database separate from the processing agents or knowledge sources. It also does so without incurring the problems associated with object orientation discussed earlier.

The adoption of a unified product model, with filters external to the structure of the product model, circumvents the question posed by Erens, McKay and Bloor [1.23] on how to define a view of the product model data. Their perception was that the choice was between separate views, each of which accessed a predetermined subsection of the product model data, and a structure in which each node contained information identifying those views able to access it.

1.5.1 Extended enterprise considerations

Large national and multinational companies employ complex design processes and process management structures, in which design tasks are carried out by several parties and across locations. These agents active in the design process form part of the extended enterprise - the entire body that carries out the design process. Different company departments in several locations together with sub-contractors will be involved.

The current economic and business environment predicates that companies are able to have their suppliers respond rapidly and flexibly to their requirements; this can be achieved in a stable fashion only through greater integration between the businesses [1.24].

The integrity of the product model is an essential factor in enabling the extended enterprise to converge on design solutions. Care must be taken in several areas to safeguard this integrity:

- the communication of product model data must not isolate them by breaking their links to related entities; once received by the vendor they must remain synchronised with the rest of the product model data;

- authority over parts of the product model must be devolved unambiguously and the eventual completion of commissions be recorded;
- the communication protocols implemented must be powerful enough to support these functions and transmit the fine detail required for unambiguous communication;
- data format translation is necessary to accommodate different company-specific data formats.

The Guide activity structure supports the above conditions. In the example in Figure 1.8, the design of one of the components of the product has been sub-contracted. Depending on the level of integration desired with the sub-contractor and whether they have a Guide installation, several operational scenarios are possible.

1. Full integration. The sub-contractor has a Guide installation and picks up the activity. Because Guide does not prescribe either the method of carrying out design or the tools used to perform the design, it is possible for a sub-contractor to contribute to the product model data of the design of its customer, even when the two do not share computing tools.

2. Read-only access. The sub-contractor can access the parent company's Guide system and interrogate it for information required in the design of the component. For example, if the geometry of component 1 affects the interface with component 2, the sub-contractor can interrogate Guide to acquire information as the design of component 1 progresses. The design work is done independently by the sub-contractor. Once the design task is complete, the results can be imported into the company's product model data as an activity that is usable elsewhere in the design process.

3. Separate systems. The sub-contractor has no access to the parent company's Guide installation and pursues the task independently, using their own design methods. There is still a record in the parent company that the design of component 2 has been tasked to the sub-contractor. Since any change to a particular aspect of the design can be evaluated for its impact on the rest of the product, should component 2 be one of the affected parts, the updated information can be communicated immediately to the sub-contractor. As with case 2, there exists a receptacle in the form of an activity to receive the results of the design.

In the first case, the product model data can reside with either the sub-contractor or the parent company. In the other two cases, a replication of the information into the parent company's system will be necessary. For this reason, the design component 2 activity box in Figure 1.8 is shown crossing the line between the two parties involved.

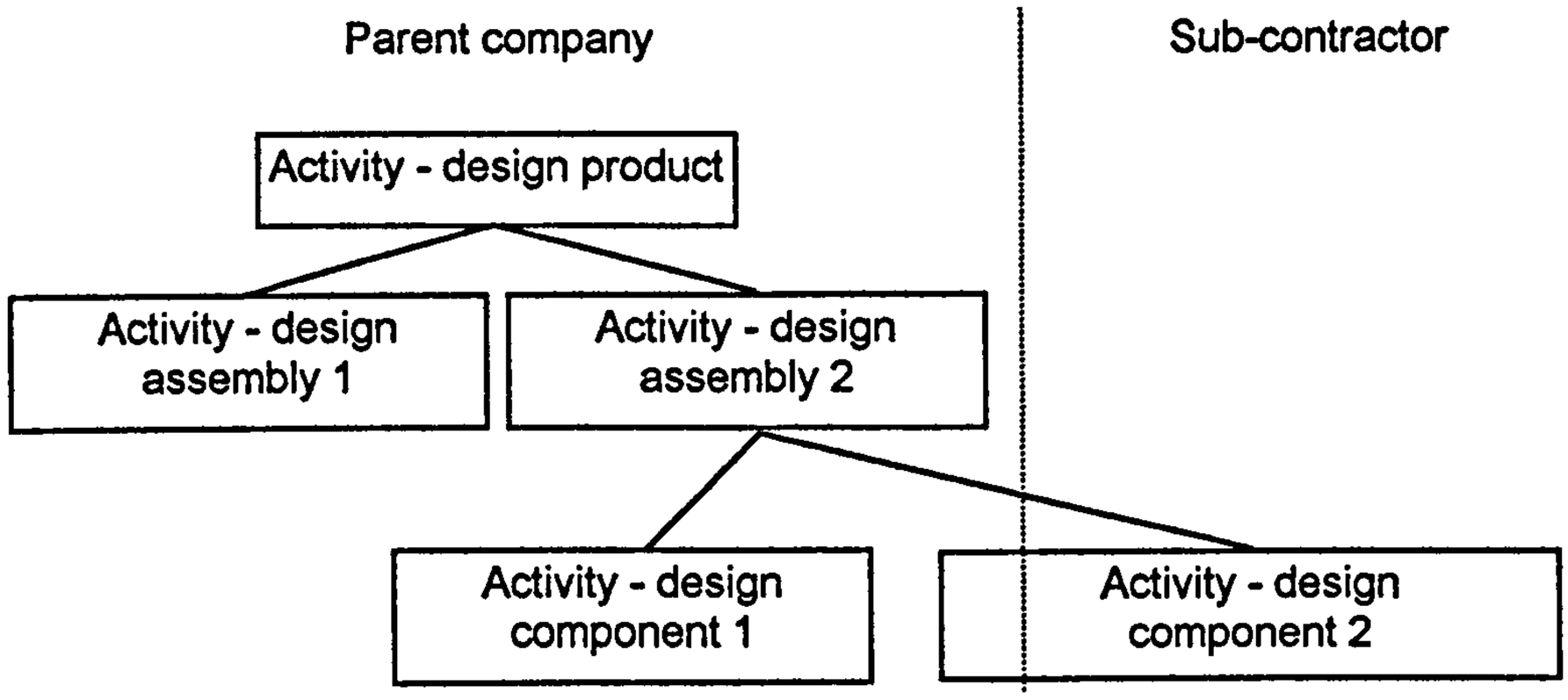


Figure 1.8 - Activity distribution across the extended enterprise

1.6 Design history

A company must be innovative and evolve its design capabilities constantly to remain competitive. This applies to all industry sectors, even in those that are well established such as paper making or valve manufacture. One method of achieving this is to ensure that knowledge acquired during a design project is not lost from the company. Knowledge loss is rapid: presented with a set of decisions alone, an expert in the field would have difficulties in determining the design rationale behind the decisions [1.20]. A record provides a method of securing the expertise deployed during the course of a design for future re-use. Sinclair et al. [1.25] describe it thus:

> *"The maintenance of design histories is critical to the success of future design; typically, these are accessed by human-to-human communication, with experienced designers acting as repositories of the organisation's design knowledge, even when documentation is adequate. Loss of these experienced designers can be critical"*

Shah et al [1.9] further define the design history as:

> *A design history provides a step-by-step account of the events and states through which a design project proceeded to produce the final design of a product. A design history contains designed product data, design process data and the relationship between them, at various stages of the design.*

At any instant, the product model data give a snapshot of the project status. They do not however contain any information on the steps taken in the design process that led to the current state. The design history record chronicles the evolution of the product model data and the operations performed to transform one product model state to another (Figure 1.9). The utility of the design history record and areas of its application are discussed in section 3.2. The capture of the history is covered in section 4.5.

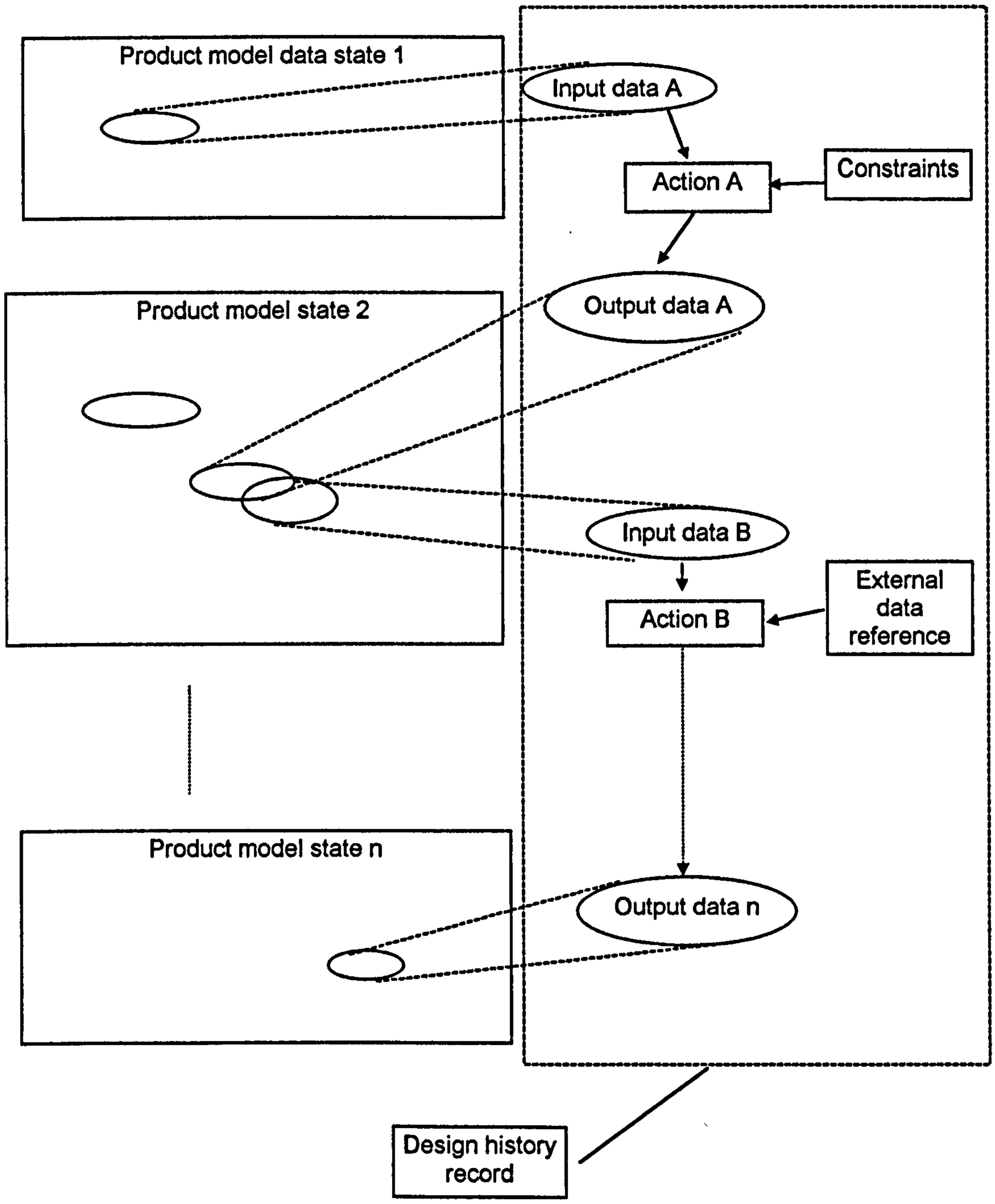


Figure 1.9 - Design history record contents

Every time the user interacts with the product model, the transaction is recorded by Guide. All transactions are recorded, be they originated by the user or by system-driven

functions. The details recorded are the nature of the transaction (the process invoked), the entities and data which it operated upon, the time at which the operation occurred and the context in which it took place. The context of a transaction describes the current activity and the particular operation in progress at the time that the transaction is executed.

The product model includes entity and process definitions in a secure form and underpins the ability to create a design history record. To record a history without these entity and process representations is impossible [1.26]. The history records only the entity identifier and the values of its characteristics. It is not necessary to record the entire entity definition, for this is available in the product model.

The design history record is primarily an action-based record. The design rationale and intent can also be recorded, but this requires extra work [1.26]. The user must be prompted and constrained at key decision points to explain the basis on which their decision is made. This rationale is captured as an entity, like any other. The approach taken is recognised as being pragmatic, rather than theoretically complete. It meets the criteria proposed by Shah et al [1.9]:

> *...Incorporation of process modelling within an object-oriented database, modelling of design constraints and rationale, incorporation of versioning concepts, and the development of a dynamic data definition language facility that can specify a Design History completely as well as represent the dynamic evolution of the design process.*

Urban et al. [1.2] adopt an approach similar to the author's and describe representations for data, processes, constraints and rationale in their model. They do not give any details of their method of history capture, or of its subsequent use.

Garcia and Howard [1.27] set out three different approaches to capturing design rationale:

1. Argumentation-based rationale. In this type, the focus is on the problems or issues that must be solved to progress the process. Each issue is defended or condemned by arguments attached to it and which will influence the decision on the issue. The archetypal example of this is the Issue-Based Information System (***IBIS***). The argumentation in such systems is often expressed in text form, with associated problems of later interpretation and the difficulty of machine parsing to establish content.

   Garcia and Howard [1.27] use a system of a design apprentice, Augmenting Design Documentation (ADD). It operates on a knowledge base of heating, ventilation and air conditioning (HVAC) system design rules. It is effectively a parametric design

system for HVAC installations, which forces the user to justify deviations from the design proposed by the ADD system.

2. Action-based rationale: in which all the actions that took place during the design process are recorded. The requirements for such a system forwarded by Garcia and Howard are for an integrated design environment that would allow designers to access all the information and tools they need. The environment should record the designer's entire log, i.e. all actions the user takes during a design session and so allow the design process to be reproduced later. Guide fulfils these pre-condition criteria and operates in this way. The capacity to price ratio of available storage media means that the potentially large volumes of data generated in this type of record are not a problem.

3. Model-based rationale. A deep knowledge base is used to support decisions – decisions need to be related to the concepts in the knowledge base, in effect explained from first principles. The flaw in this approach is that designers often use heuristic methods or even personal preferences that have no foundation in first principles and cannot be explained through this model.

1.7 Implementation of Guide

Guide provides the company with a base structure from which to support its product model. Guide Manager proffers a set of tools with which to describe that product model to the system. Once the product model is defined, the system provides the utilities necessary for its use via its engine and interface. An implementation will therefore have two stages: definition and use.

It is not necessary to define the whole of the product model to Guide *ab initio* - a partial product model will suffice. Guide can be implemented in a progressive fashion, assuming new areas of operation over time. Indeed, in some instances it may only support a single function in the company. Once installed however, the utility and flexibility of Guide will engender the demand for its deployment across other parts of the enterprise.

Guide is provided, in base form, with its core and processing engine. Its entire functionality is fully available when it is delivered. It is the definition of the company's product model to Guide that allows the company to exploit the functionality which the system offers in the context of its own design processes. The tool is developed and evolved by the company; it does not inherit an inflexible tool that will only partially satisfy its needs [1.11].

Cleetus [1.7] comments on the importance of companies building their own data structures. Eventually, it may be that the entire product model is defined through the ***PDES*** standard; this is not going to happen for a long time and companies still need to operate in the interval. In disagreement with Cleetus, there is already evidence of companies moving to common, or at least commonly interchangeable representations e.g. ***OLE, CORBA*** or native PDES-based representations.

In software terms, Guide sits between the two extremes of standard and bespoke applications. Standard applications force the company to align their systems to the software, and accept at least some of the methodologies prescribed by it. Their advantages are that they tend to be inexpensive and upgraded on a regular basis.

At the other end of the spectrum are bespoke software systems, written to the specification of a company. While these provide exactly what the company asks for in terms of business process support, it fixes these processes to that point in time. The software initial cost is high, as are any subsequent updates, the company being the sole customer for that particular software utility.

1.7.1 Implementation resources

There is a requirement for computing support staff to manage a Guide implementation, as with any software package installation. Their efforts will be directed to defining the product model to Guide and maintaining it. This is not an extra task on top of their existing ones, since many of their activities will be made redundant. The company will not need anyone to manage the metadata - relations between CAD drawings, bills of materials, documents, standard parts and any other data employed to specify and define the design project in some way. This will be handled through Guide, which offers powerful tools to assist in this process and operates directly on the product model data, rather than on metadata. It is the responsibility of the support staff to provide for the users entities and processes which have meaning and utility in their application domain. [1.28]

It is possible to provide collections of standard entities and processes that the company could then use, discard or adapt as necessary. Such collections could be delivered with the Guide installation, or be supplied later. For example, a milling machine could be supplied with the set of entities and methods that describe its capabilities and operating parameters. This would decrease the burden on the administrator.

Like any piece of software or tool, there will be users conversant with different levels of complexity of the software. The normal user will employ Guide as they would a word processor, television or car. It is a tool with which to perform operations; functions and

methods are offered to them for pursuing their task; they do not need to have prior knowledge of the system architecture. The specialist user will know how to develop entities in Guide and possibly methods for their particular sphere of activity. They may be entrusted to carry out the tasks themselves, or have to refer them to the system administrator. The administrator can make changes to the product model, which are then available through the system to all users.

Willcox and Sheldon [1.16] working on "Design for X" (DfX) tools stated that from their case studies their observation is that the real benefit of such tools is in terms of project management and cultural change. The company implementing Guide must be aware of this outcomes plan for it and ensure that the benefits are maximised [1.29].

1.8 Review

Mäntylä [1.30] gives a statement of the intention of using a product model:

> *The bold ultimate goal of product modelling is to be able to represent all this information in a way which makes it possible to capture and access the relevant information through the whole design-planning-manufacturing sequence, with no loss of information in any stage.*

The product model is a common framework on which all operations of the company can be built. These encompass, among others, the company strategy, a design project plan, product design, manufacturing and support to the end of the product life cycle. The definition is robust enough to support the requirements of the extended enterprise, where design teams are scattered among several locations. Processes can take place concurrently and dependency links can be established early in the design process, even before details are finalised. Links in the product model data are maintained explicitly; this obviates the necessity for metadata to track related items.

Guide provides utilities with which users can define, maintain and apply the product model. It is by nature configurable by each company and allows business processes to be conducted without prescription. In this respect, it differs from the work of Beitz and Feldhusen [1.31]. They propose a construction whose elements are a product-defining model, product-defining data and operational principles, analogous to the product model and product model data. The implementation however is a form of parametric design, where their elements are all fixed and pre-defined. For each industry and design type, entities, processes and design phases are imposed to which the designer must adhere.

A record of actions taken through the system is made and stored in a format that facilitates its subsequent use.

Guide is implemented on industry-standard utilities and resources. It allows for legacy systems and data to be accessed from or incorporated into the product model. Its implementation in the company can be global or selective, to support a particular function or department.

The effect of the implementation of a product model and the associated methodologies demolishes one of the fundamental problems in design and business, that of communication of information. The problem is not a new one [1.32]:

> *"Come, let us build ourselves a city, with a tower that reaches to the heavens, so that we may make a name for ourselves and not be scattered over the face of the whole earth." But the Lord came down to see the city and the tower that the men were building. The lord said "If as one people speaking the same language they have begun to do this, then nothing they plan to do will be impossible for them. Come, let us go down and confuse their language so they will not understand each other." So the Lord scattered them from there over all the earth, and they stopped building the city.*

Those building the city and tower of Babel were confounded when their common language was confused and translation became an essential part of communication. It was not their engineering ability that was affected – they were still as capable architects, builders and planners as before. An analogous situation exists in companies, where communication difficulties impede the progress of engineering and business functions. Guide provides a common language for company transactions, thus removing the translation issue from communication.

2. The Product Model in a Business Context

2.1 The business case for Guide and the product model

The product model affects the operation of business at every level. Its implementation is not a matter to be undertaken lightly, and must be done for sound business reasons. This chapter sets the product model in context at every level of company operation and expounds its relevance.

2.2 Business Processes - the Cultural Revolution

The nature of the challenges which businesses face changes constantly. Those of the recent past have been the increasingly global nature of the economy, with competition coming from countries which a few decades ago hardly had a manufacturing industry. Some industries are protected from newcomers because they are highly capitalised, such as semiconductors and automotive mass manufacture. In others, the hurdle to entry is very low - in the software market, they are a personal computer, a good idea and lots of programming time. In this market, an upstart can unseat an established player. The circumstances external to companies also influence their policies, including those directing design work in the company [2.1].

The nature and structure of companies and hence competitive pressures in the marketplace have evolved over time, from when craftsmen served a small geographical area and often would have little real competition. During the industrial revolution, companies increased the volume of goods they produced. Automation was a powerful tool in this growth phase, making previously unimaginable productivity levels possible.

Following that came technological improvements. Two world wars in this century have accelerated these advances, through the development of disciplines such as metallurgy and computing and their application.

Companies needed to become more flexible in their response to fluctuations in market demand [2.2]. This is reflected in the push towards JIT supplies and the concomitant reduction in stock levels, to allow a more rapid change in production without incurring heavy penalties through the processing or loss of high stock levels.

Subsequently, there was a focus on quality. The effects of the media and education were that consumers became increasingly sophisticated in their choices, and poor quality and reliability were more widely publicised - consumers had found their voices. Allied to this, far eastern companies were manufacturing goods to high quality standards which were gaining a reputation and following on that basis.

Currently, functionality and efficiency are agents of change. Customers are demanding more for their money, while environmental awareness causes them to look for efficient products. In the IT sector especially, companies are being forced to deliver new models that offer more functionality for less cost, year after year. So the change agent evolved, and any company that did not adapt to keep pace with the latest trend went out of business. For example, quality was an optional extra twenty years ago, now it is an essential pre-requisite for operation.

Changed consumer attitudes is another factor that influences the ways in which companies operate. Many markets now operate on a quasi-fashion basis, for example computing, cars, and leisure equipment. Henry Ford would have great difficulties today in persuading all his customers that black was the perfect colour for their new car. Furthermore, for many items, there is a recognition of future obsolescence or designed disposability. A scythe might have lasted a lifetime, and been expected so to do, but a computer will be obsolete in three years at most and many people throw out their razor or contact lenses every day. This means that companies must be able to respond quickly with new designs at regular, short intervals and keep up with the latest in technology, so that their product is, at its launch at least, ahead of the competition. Technological pre-eminence also enables price top-slicing until competitors catch up.

Industry has gone through all these changes, mostly by responding with better or different products and optimising one aspect of their operation, in isolation from others. Now that affordable, reliable, efficient, high performance goods are expected as the norm, the pressures on companies are different to any experienced before, they are for internal reform. Companies need to be able to implement new business procedures rapidly and effectively. Hanson and Voss [2.3] find that:

> *The biggest inhibitor to achieving business objectives is not lack of funding, government policies or interest rate change - but the ability to implement change quickly enough.*

and

> *There are no 'quick fixes' - the leading companies are better than the laggers because they have adopted better practices and improved their performance at <u>every level</u>.*

These sentiments are reflected by Mattinsley, who states that improvements in individual areas are not going to lead to business breakthroughs [1.29]. This then is the cultural revolution which must take place in industry to meet the challenges of today. The emphasis on 'every level' in the quote above is most significant. The product model provides a platform for this to happen in a structured and flexible manner by addressing

and enabling every aspect of the business in a cohesive manner. It supplies both a vehicle and a challenge to change.

To understand the nature of changes required, company structure and operation must first be examined.

2.3 Company structure and operation

The business process may be defined as

> *A set of interrelated activities and their relationships which, when executed, achieve a declared business outcome [1.29]*

The emphasis here is on the relations between the activities, both before and after execution; put otherwise, nothing in a company happens in isolation. Any change or action will have implications elsewhere in the company. The structure that supports these activities and allows them to happen in a controlled manner is described below.

2.3.1 Business structure

Businesses operate through application of the following levels of administration and action. The list does not imply a strict hierarchy.

- Strategy. This is the vision and mission statement. At the enterprise level, it includes policies on matters like market segments to be attacked. It represents the 'declared business outcome' above. The strategy leads to an operating plan, through which the strategy is implemented for the short and medium terms.
- Practice. The set of rules according to which the company operates, including aspects such as company structure and organisation. It is a formalisation of the procedures to be executed in order to achieve specific functions.
- Function. The outcome of engaging one or more processes, e.g. product manufacturing
- Process. A definition of the outcomes which must be achieved to implement a function. It does not prescribe the method of execution.
- Procedure. An ordered set of tasks which defines the mechanism for accomplishing a process within a particular organisation.
- Task. A unit of work.

Some or all of these elements are applied to successive layers within the company operation - just as the company has an overall strategy, so too a department will have a

departmental strategy, which is designed to undertake company processes and so perform company functions. The operations of the department are in effect the implementation of functions which the company carries out.

A simplified example of a business structure is shown in Figure 2.1. The relation of the different levels to supporting tools and data is also shown.

Supporting tools facilitate tasks at the level they are deployed and are generally tailored to those tasks. They may be computer based but may equally comprise manuals, procedures and handbooks. They may be of a general nature, or highly specialised to the department and activity to which they relate.

Applications which support a particular activity, and often do so very well, lead to what are often termed 'islands of automation' [2.4] - a specific task can happen very quickly, but the tool to enable its rapid execution and the resulting data are completely isolated from any other systems in the company [2.5]. These data can often only be read by the software that created them. The effect is represented in Figure 2.1 by the horizontal dotted lines; a fragmented information model results.

As the number of specialised computing tools for solving narrowly defined problems increased and they became more universally and widely used across departments, so their isolationist nature was both recognised and became a problem [2.6]. Shah and Rogers refer to this as 'islands of optimisation' [2.7].

The effect is compounded when it is considered that the figure shows only one branch of the tree. In practice, there are several departments carrying out many tasks to fulfil all of the company functions. Several departments will carry out work related to one product, so information about that product ends up distributed in a non-linked fashion across the company. These barriers can further be complicated by releases of new versions of the software that may not be entirely backwardly compatible with data generated under the earlier versions.

Urban et al. [2.8] propose a Shared Design Manager (SDM). This is an integrative tool, based on STEP entity definitions. It maintains a metadata dictionary that maps onto the entity sets of computing tools used and holds relationships between the entities contained therein. It also seeks to provide a common interaction user interface. This is an understandable but misguided view, using entity representations as glue between applications rather than as the foundation supporting them.

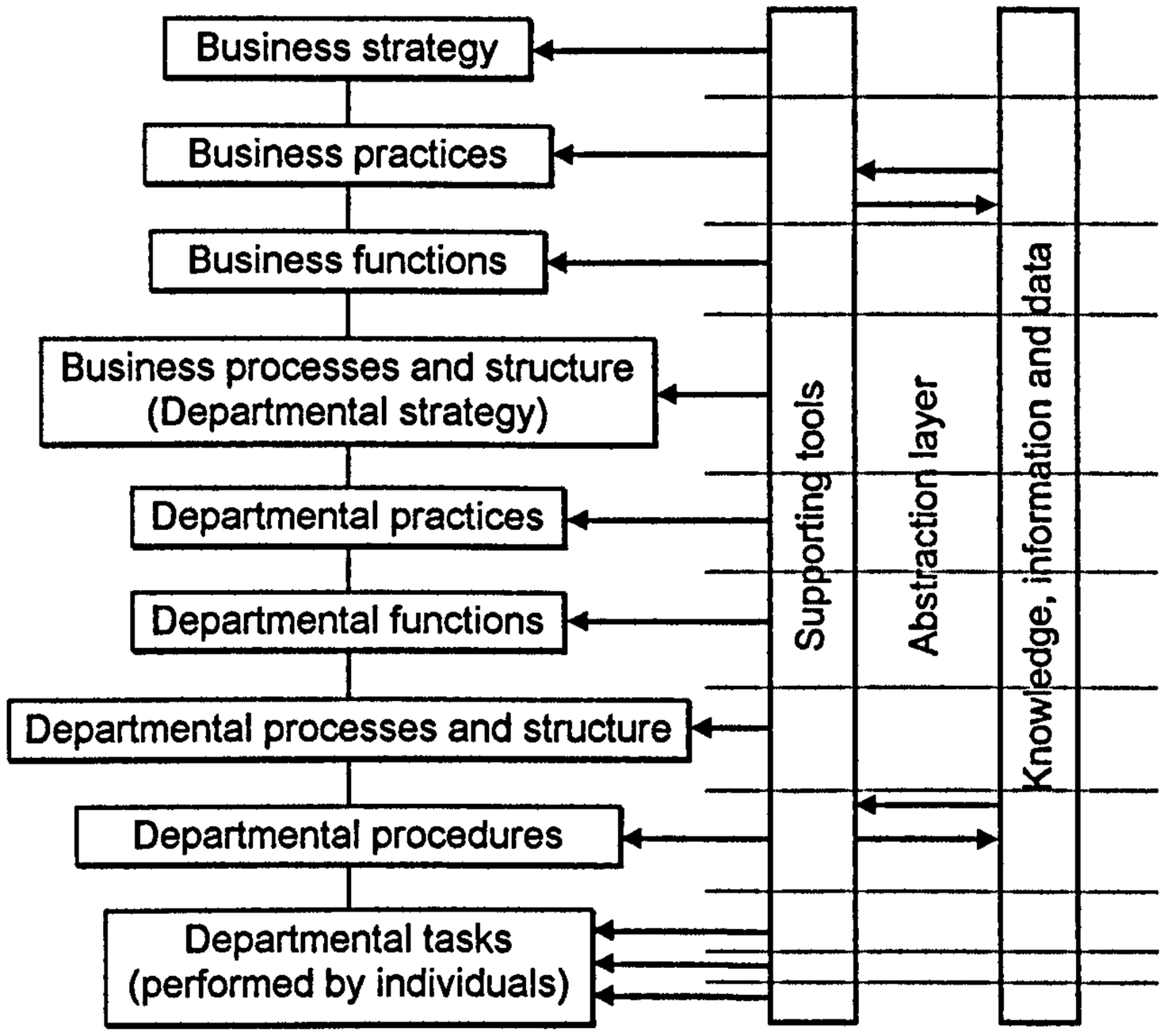


Figure 2.1 - Relationship of business activity to supporting data and information

The model in Figure 2.1 shows a template for activity. When it is exercised, instances of each of the stages are created; for example, a new strategy of entering the mobile phone market is formulated. Departments will be set up to implement the strategy by performing the tasks that complete their departmental processes. The volume of activity and rate of change increases as the project cascades down the tree - hours of market research and thousands of drawings will be realised to discharge that single company strategy. By considering the model this way, two new dependencies are introduced - the hierarchy dependency (actions are performed in a certain context) and a time dependency. The hierarchy is managed through Guide activities: each activity instance created is time-stamped, enabling precedence to be established.

The layer of abstraction between the supporting tools and the data is the interpretation that must be applied by either the application or the user of the application in order to make sense of the information presented to them. A simple example is that reading a piece of text requires the reader to be familiar with the language used. Another example is the conventions and rules that need to be known in order to understand an engineering two-dimensional drawing representing a three-dimensional object.

Over time, as supporting applications become more sophisticated, the depth of the abstraction layer is decreasing for the users. It is easier to understand a

three-dimensional CAD model that can dynamically be shaded, rotated, sliced and seen in relation to mating parts than a two-dimensional engineering drawing.

For computing applications, there also exists an abstraction between data and their electronic storage format. This is a separate issue, relating to the mechanics of the storage of information, rather than their content.

If, as the quote at the beginning of the section indicated, changes are to occur at all levels, then there is more to the issue than examining each of the levels separately and optimising it, since there are interrelations between levels, and implications up- and downstream of any change made. What is required is a holistic approach to the management, actions and information processing requirements of the company, leading to enterprise-wide solutions which can then support tailored, activity specific tools within a common framework. The solutions must ensure that the integrated information model is context and time sensitive.

2.3.2 Product development

This is one of the core functions in an engineering company. Product development ensures the future of the company and enables it to keep abreast or ahead of the competition. Like any other function it is a business process and conforms to the structure laid out in section 2.3.1.

Figure 2.2 shows the effect of the development time on potential profit. Area A, the cost of product development, needs to be smaller than area B, the profit from sales, for the product to produce some return to the company. Bringing a completely new product onto the market well ahead of any competition will yield a larger return. Conversely, being second to the market significantly reduces the exploitation potential.

Computer technology has been deployed to assist and improve the embodiment and manufacture stages of product realisation, in the quest for shorter lead times and lower costs. An example from manufacturing is the way in which the CAD solid model of a part can rapidly be processed to yield CNC code for its manufacture. CNC machine tools then generate the component in a repeatable, accurate and rapid manner. As discussed in the previous section, These advances have occurred mostly in a defined and narrow domain, aimed at improving one particular process, rather than being part of a holistic approach to improving company performance. This is not to denigrate the importance of these measures, for they have in the main succeeded in achieving or even surpassing their goals. The changes have also concentrated towards the detail design and manufacturing areas of product development.

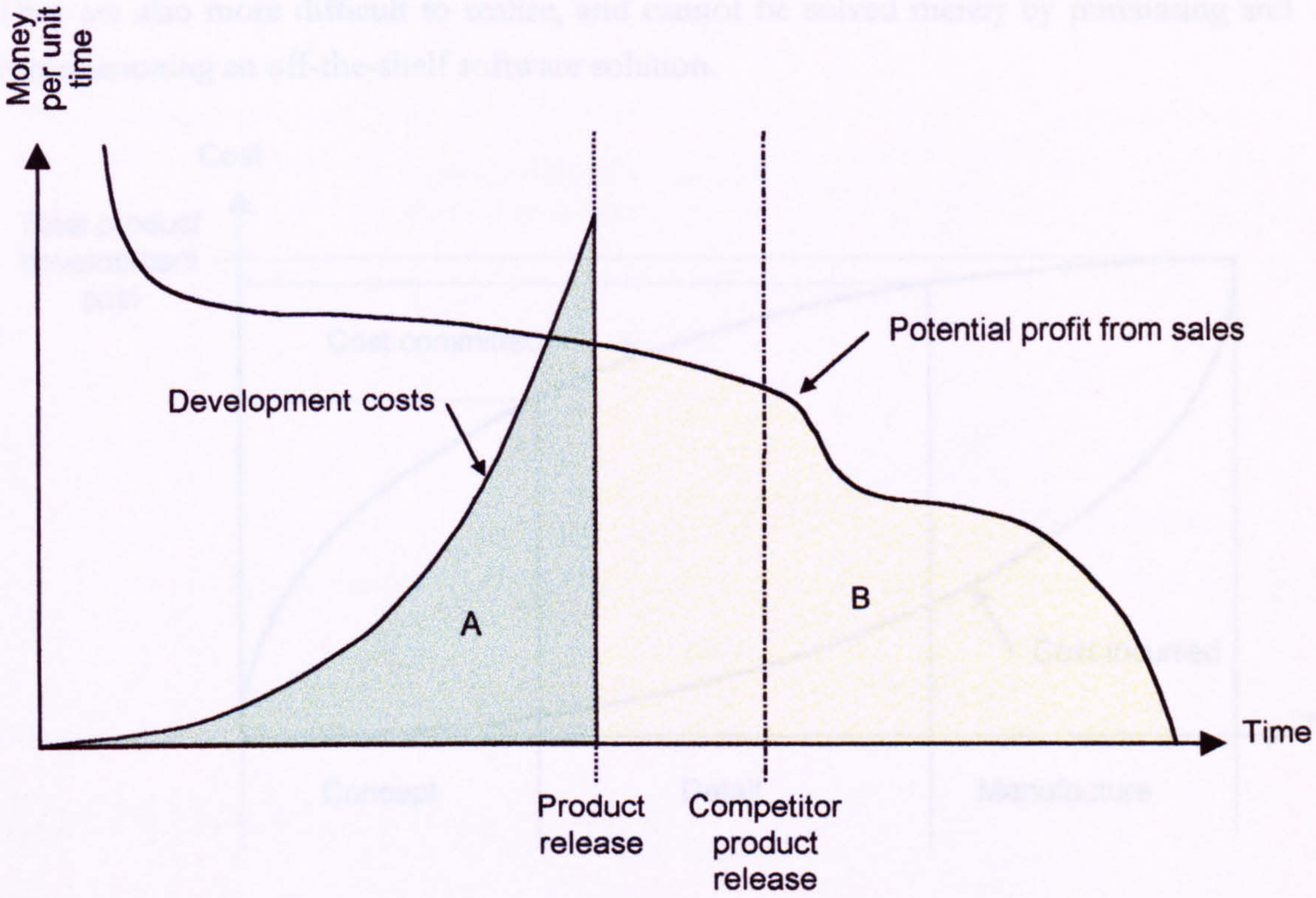


Figure 2.2 - Product lifecycle versus product development time

In recognition of the narrowness of the field of application of most computer assistance tools for the design domain [2.5], some of the initiatives taken in this area are generically branded Design for X (DfX), where X stands variously for Manufacture, Cost, Assembly or other parameter to be considered. As their name suggests, they still concentrate on that specific parameter and are not constructed to take a more holistic view of whole product lifecycle effect on cost [1.16].

From Figure 2.3, it can be seen that many of these improvements have occurred in the later stages in the product realisation process. Their principal effect has been to reduce product development times, meaning that investments can start to be recouped earlier. The desired reduction in development times are very large - companies aim to at least halve or third their development and manufacturing cycle, keeping their product range up to date, and thus maintaining a competitive edge [2.9]. This cannot be delivered exclusively from improvements applied to the later stages of the project.

Effort is now directed at avoiding cost *ab initio* - reducing the costs built in to the product and its manufacturing process at the early design stages [1.17]. This is also where the majority of costs are incurred [2.10]. The opportunities for controlling and reducing costs in this area are greater than addressing the later stages of product development.

They are also more difficult to realise, and cannot be solved merely by purchasing and commissioning an off-the-shelf software solution.

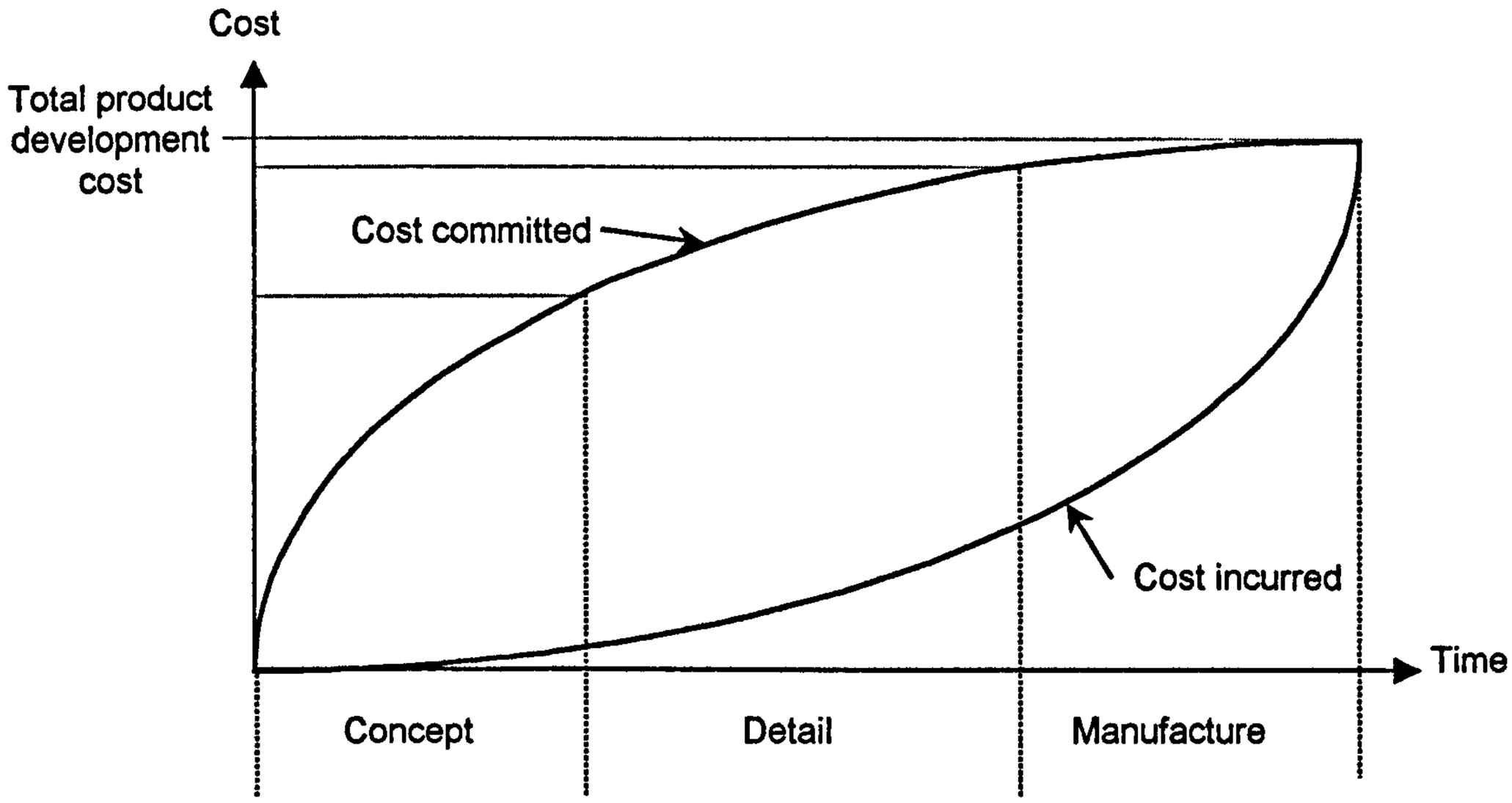


Figure 2.3 - Costs analysis of a project

The difficulties arise for many reasons. The information in the conceptual stages of design is difficult to represent in any form. It deals with intangibles and, as the name indicates, concepts. The consequences of decisions made at this stage appear only later in the process, including cost implications, as seen above. Solutions to managing the early stages of product development must address this issue. There are also issues of control over the information generated during this stage, which are discussed more fully in section 2.3.3.1.

Some of the benefits in better management come from later cost commitment. If alternatives can remain open for as long in the design cycle as possible, then issues can remain fluid until late in the project. Decisions can then be taken with as complete a set of information as possible (Figure 2.4).

One benefit of a structure that will support in a secure fashion functional and partial information is that designers will not seek the certainty of fixed geometry early in their deliberations. Ranta et al. [1.24] observed that by postponing decisions on geometric form, the designer is more likely to arrive at an innovative solution to the problem.

Guide enables relationships and dependencies to be established between entities and areas of a project before they are fully defined. This allows for early what-if analyses to be performed and for product details to be left fluid until late in the project. Meanwhile, constraints safeguard the eventual integrity of the solution.

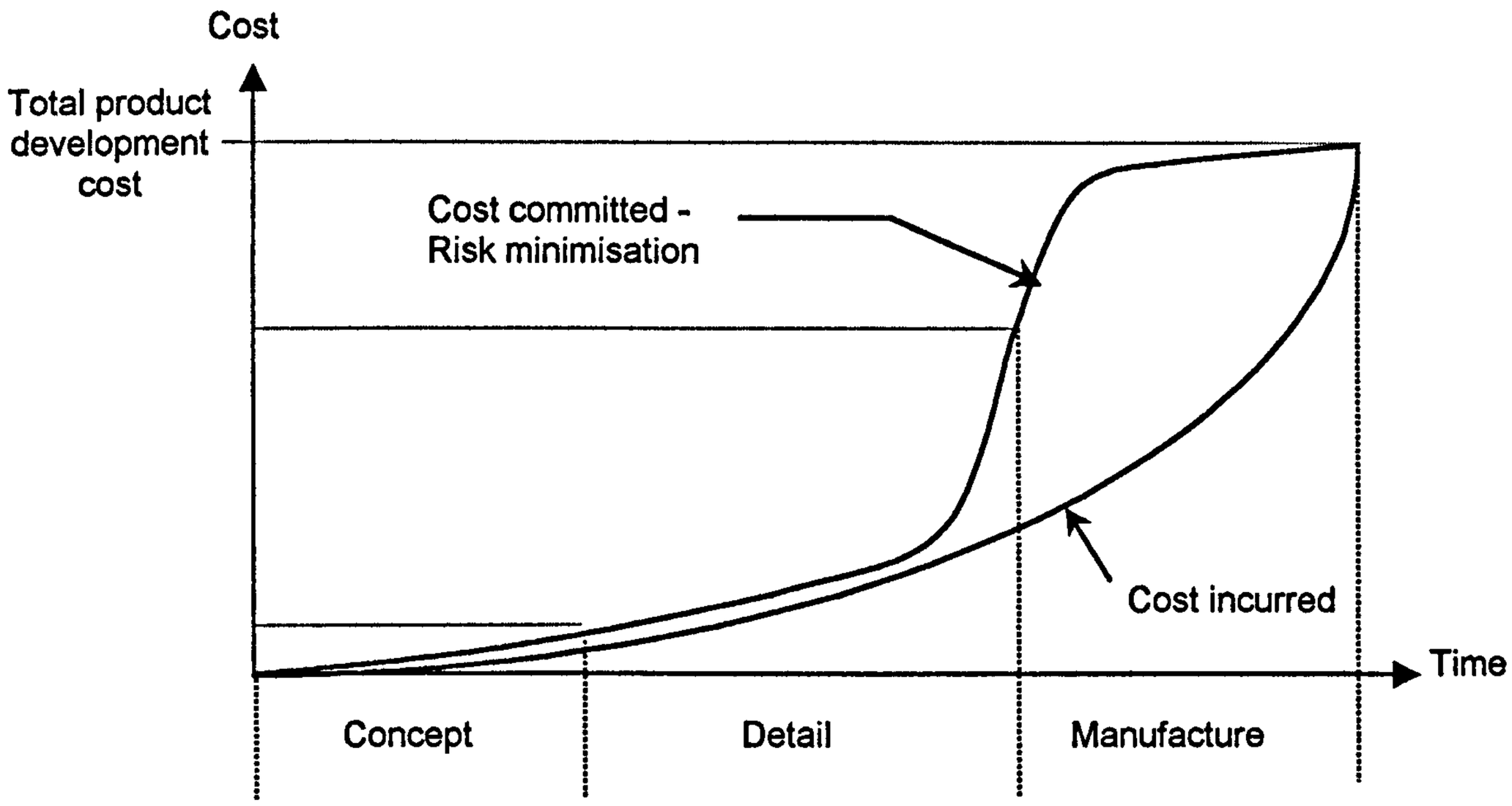


Figure 2.4 - Preferred cost profile of a project

2.3.3 Control and monitoring

There are several different aspects to this area. The first and most obvious is monitoring the integrity of the work in progress. This might be making sure that the design for the parts of an assembly will produce components that mate properly (section 2.3.3.1). Secondly, there is monitoring of the project generating the work in progress. This is measurement against parameters such as project cost and progress against deadlines (section 2.3.3.2). The third area of monitoring is measuring the effectiveness of the processes themselves, ascertaining their suitability and quality (section 2.3.3.3).

2.3.3.1 Monitoring work in progress.

The purposes of examining the work in progress are:

- to pre-empt errors or reduce their repercussions when they occur,
- to ensure that specifications for components are being adhered to,
- to ensure completion and compliance of design checklists
- to check for concurrency in the design and as 'sanity checks' to make sure that errors and false assumptions are not being built upon.

During a product development, products on which it is deemed that development has ceased are released to manufacturing. After that time, any changes to the part are carefully monitored, since decisions may have been made on the premise that the part would not change. Any effect of modifying the part must then be carefully examined.

This release process at the current time separates well-represented, stored and maintained data from that which is not.

A typical release cycle is shown in Figure 2.5. Standard parts are, by nature, fully and completely defined before the design commences. Their manufacturing processes and instructions are fixed, and are of no concern in the current design. A complete physical and functional interface definition is necessary to integrate the part into the evolving design.

Variant parts require modification from an existing design to adapt them to the required specification. This is less time-consuming than new design and variant parts and contributes to shorter product development cycles. In order to fully benefit, the manufacturing information and design history for the product must be available. The design history and variant design is described in section 3.2; the manufacturing information saves re-definition of operations unaffected by the changes to the original part. An essential element of the re-use of standard parts is that they be easy to find. This problem has been addressed by group technology systems.

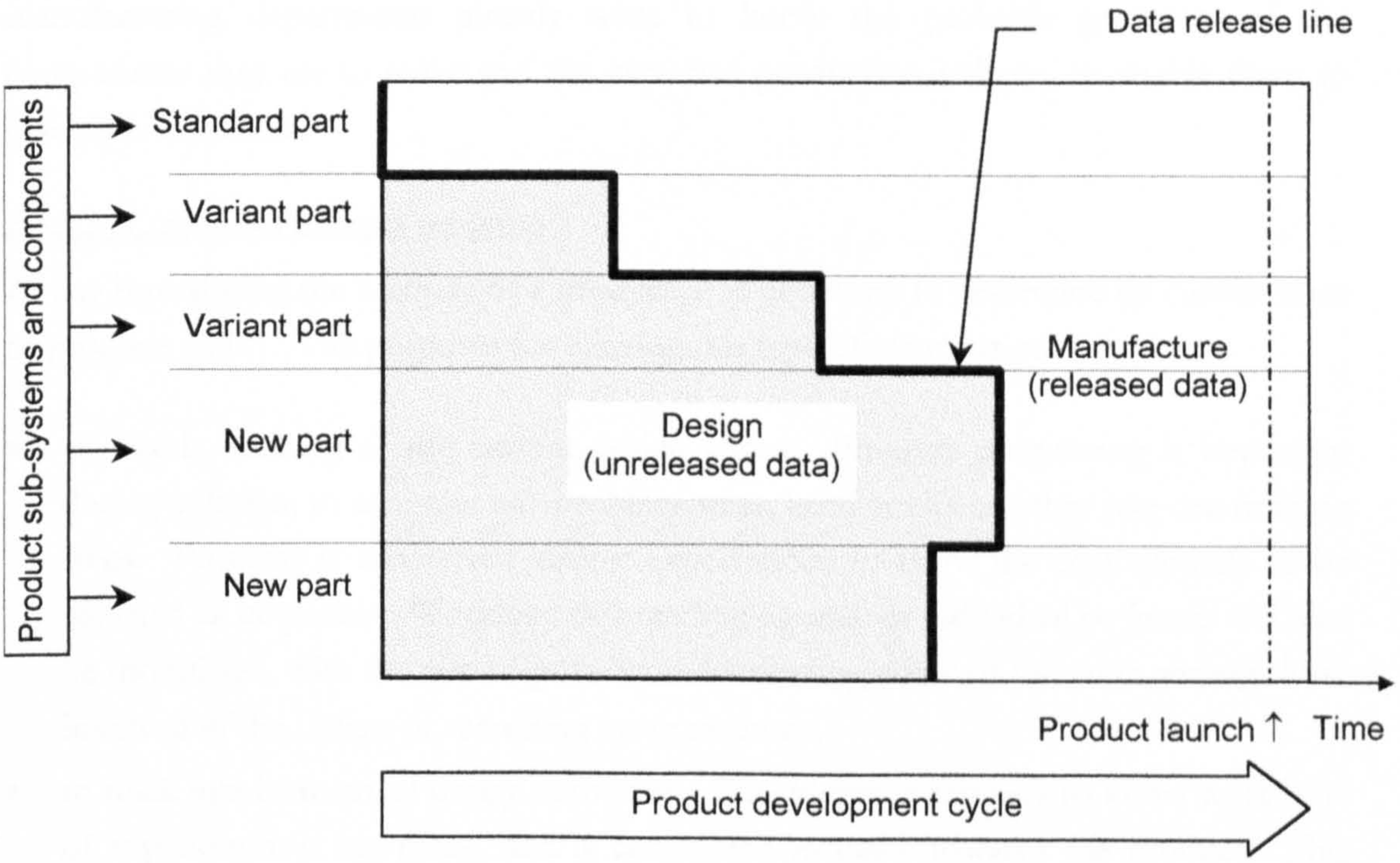


Figure 2.5 - Release cycle during product development

New design parts require the highest work input before they can be released for manufacture. The use of standard and variant parts offers a significant opportunity to reduce product development cycle times. Difficulties in leveraging this advantage are

initial awareness of the part and incomplete information. Extensive reverse engineering and guesswork can add significantly to the variant design timescale, reducing the advantage potential.

Before a part is released, the information held about it will be known only to the designer or design team. This is everything in the shaded area of the figure. There is no formal, traceable way of noting changes or of evaluating or tracking their effects. Because release happens at different times, changes may be made to a part under development which affect a released part, without this being apparent until the first batch of assembled products is tested.

After the release point, there is a strict mechanism for controlling change. This is essential, as the release action enables tooling and manufacturing capacity to be allocated to the product, and so any changes made to it will have direct and immediate consequences on the downstream operations. This philosophy does not however carry backwards to the early parts of the design process where, although changes do not have immediate effect on downstream processes, their effect, as has been seen above, is much wider-reaching. Furthermore, during the stages prior to the release process, manufacturing departments already want to know the probable geometry of the components they are to make and the expected production volume, to enable them to plan their resources.

2.3.3.2 Control of process progress

At any time during the lifecycle of a product, it is necessary to determine its current state of progress [2.11]. The purposes for eliciting this type of information may be:

- to enable tracking of the current design status. Progress monitoring is important during a design, to ascertain any problem areas, bottlenecks or other rate determining steps. Progress is monitored against expectations to allow the cost estimate to be adjusted as necessary. Workload outstanding against an individual or group will also be monitored, with the aim of possibly redistributing some of the tasks among those involved in the design or recruiting more resource.
- to track involvement of design resources. The company requires to know what level of expenditure it has made, and is committed to make towards the product. This may be, for example, in terms of capital investment, man-hours, database transactions or disk storage space used.
- to help in forecasting, including time to release and total projected cost.

Mechanisms are provided for these types of assessment procedures, like any other procedure in the company. These must be managed and updated as appropriate.

Wood [2.11] provides a similar list of uses for design process monitoring, viz.

- Compliance with the specification.
- Optimisation for reliability, operability, maintainability, safety and lifetime cost.
- Status of development and test programmes.
- Agreement of documentation with entries in the project information system.
- Control of interfaces with other work packages.
- Change control status.
- Resolution of actions from previous reviews.

2.3.3.3 Audit

Audit differs from the monitoring described in section 2.3.3.2 in that it is concerned not with the current project or design, but how effective the processes and tasks are at realising that project. The purposes of audit are:

- to provide persons responsible for the management of processes tools with which to evaluate changes in the process against different criteria. Highlighted in Hanson and Voss [2.3] was that:

  the adoption of BS5750 - promoted by many as the most important quality practice - does not in itself guarantee any improvement in quality performance.

  In other words, implementing processes does not guarantee that the objective of the process will be accomplished. A measure of the process' ability to achieve its stated goals is required.
- to provide tools with which to analyse the resources employed during the course of a project, such as a product development cycle. This will give an indication of over- and under-used company resources and provide predictive analysis for future requirements.
- to assess the effect that introducing new technologies or techniques has on the design process. Companies want to know what savings new techniques, management or design have on process parameters such as lead-time and cost.
- to identify risk and its repercussions. During the project, it is desirable to assess the impact that a change will have. Similarly, once a product is in service, it should be possible to track any failure back to the decision which incorporated the incompetent feature into the design, or failed to design it out of the product. Constraints on that part may subsequently be tightened, or a different set of parameters prescribed. In litigious cases, the company will want to demonstrate that all reasonable precautions were taken, and that the failure, and consequent damages, were not due to negligence.

2.3.4 Managing change

Changes happen to the elements of the business operation structure - imperatives change, new procedures are implemented and tasks updated as new computing tools are deployed. Some of these changes, at all the different levels, occur in an informal, unmanaged way in every company. Employees find ways round official company practices that are more suitable for them and their needs. While possibly improving the efficiency of that person or the section in which they work, it may have adverse effects on other operations within the company. There is a risk of dissociating data from the process that created them, because the latter is not documented. At some stage, the data generated by these processes need to be patched back in to the company product model, where downstream operations can access them. To resolve this problem, companies may endorse these processes as official fixes, even though they may not be well considered and developed replacements to the existing ones.

The ability to support and enable changes in company practices in a responsive but controlled manner is required. When considering change, it is also important to remember that it must still be easy to access and use data generated under previous practices. The common perception has been that information relating to products is of a dynamic nature, while that relating to company procedures is static. This is why managing change in procedures is not a mature discipline. In fact, all the business elements are subject to change, at different rates. In Figure 2.1, the rate of change increases from the top to the bottom of the figure. This applies equally to processes and data and does not imply that the upper levels can be considered permanently and eternally static.

The reason for considering the rate of change of processes, rather than taking snapshot views of their state at a particular time, is that evaluating the effect of the changes is of great importance in assessing their impact on the success of the business. Furthermore, to establish how changing a process affects the characteristics of data generated by it. It is important to understand the way that constraints applied to the process evolve, so that earlier work may be understood in the context in which it was created [1.7].

In effect, this amounts to applying version and change control to all types of information used by the company whatever their nature, including processes and standards[1]. In order

---

[1] The author is not suggesting that all items of data are subject to constant change, especially when considering those which are science-derived, such as material properties. By considering materials technology and properties as a whole, however, it can be seen that developments and enhancements in this field will mean that constraints previously

to do this, it is required to model information, including procedures themselves, in a suitable format, understand the relationships and dependencies between the pieces of information and the evolution of the data and their relationships so that version control is no longer limited to engineering drawings. This approach is a further development in the attitude to product model definitions. Initially, only models of static definitions were constructed, followed later by dynamic information [2.12].

2.3.5 Design and information

The rules and laws which govern the real world also constrain design: artefacts created as a result of the design process are part of the real world, and will therefore behave according to its laws. The artefact is the result of a creative process, represented by crossing the sea of Figure 2.6. The sea represents a variety of impediments to artefact creation - it hasn't yet been thought of, nobody knows how to make it or make it work, laws prohibit its creation to name a few examples. In order to bridge the gap between the mainland of reality and any possible artefact island, a bridge of knowledge must be built between the two. The support elements for the bridge spans are:

1. Science and technology. This is as good a model of reality and explanation for the way it works as is currently known and accepted. It will never be an exact model and representation for reality, but it is being refined as research uncovers more of creation's secrets over time.
2. Engineering knowledge. This represents information useful to the engineer. It is in part a subset of science and technology, but it also includes elements outside their scope, rules that the engineer can use even though the underlying technology is not understood. The Wright brothers were airborne before their understanding of aerodynamics was complete. Engineering knowledge is effectively the knowledge on how to apply science and technology [2.13].
3. Design rules. The designer is constrained by parameters not directly related to the artefact's physical properties and behaviour. These can be standards and specifications to which the design must conform, or matters of aesthetics.
4. Designer experience. The position of experience as overwhelmingly the most important aspect of the solution has diminished, as crafting has turned into engineering.

---

imposed by this technology will change when applied to a product. Thus version control over the state of materials technologies and their application is important if previous designs are to be understood within the context which they were designed.

5. Design data. These are the data produced as a result of the design process. They serve to describe the artefact so that it can be manufactured and provide information on the product cost, recyclability, or any other relevant attribute of the product.

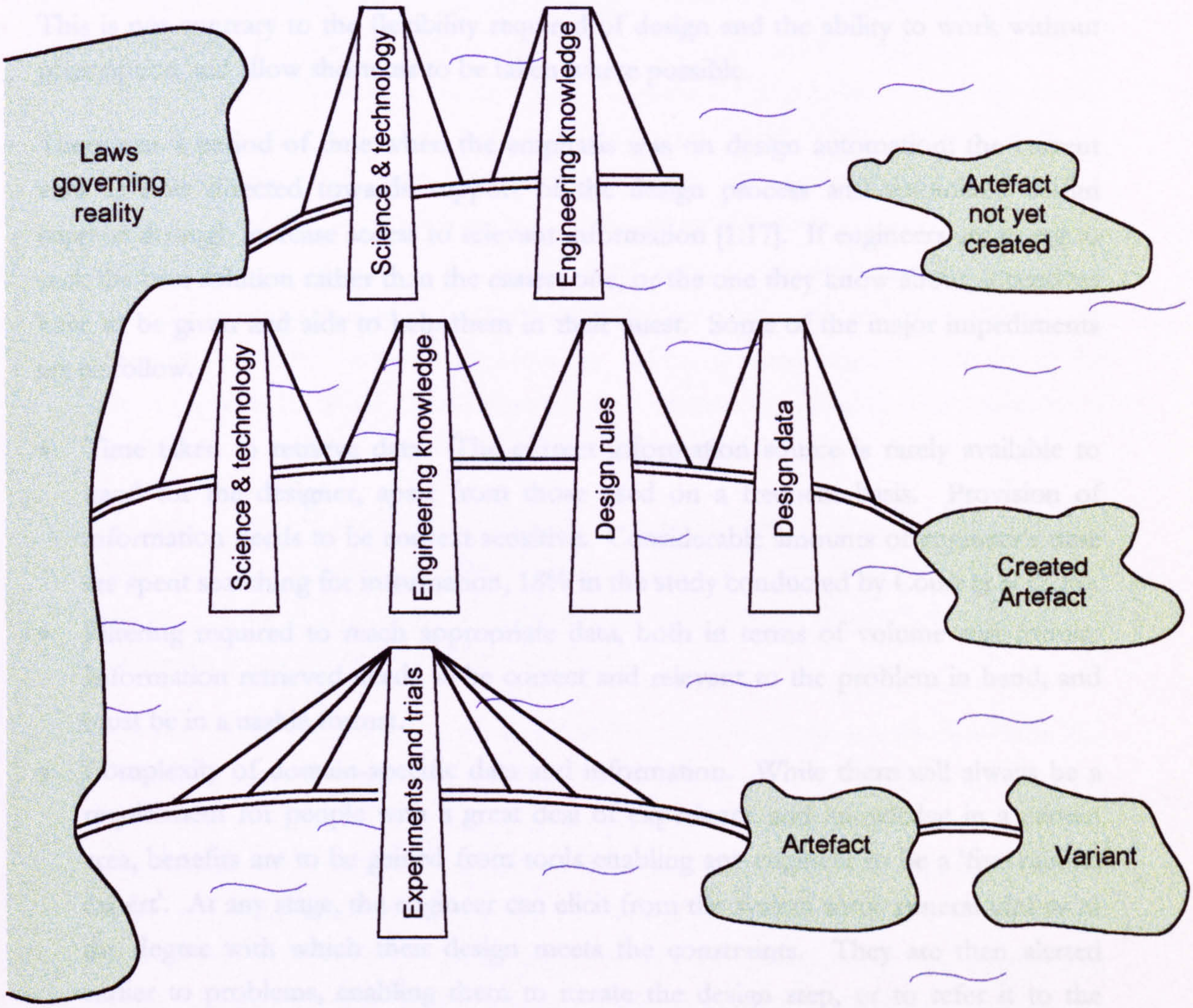


Figure 2.6 - Modelling path to creation of new artefacts

There are other possible contributors to spanning the gap, such as invention, discovery and the use of an existing artefact to spawn another. If any of the supports are missing, then the remaining ones need to bear the load.

The rate of change of information increases with distance from the mainland. On the assumption that 'God doesn't play marbles with the universe' [2.14], reality never changes and the universe continues to run according to the same rules. At one end of the bridge, our science and technology base evolves very slowly. To call anything absolutely static, however is imprudent - even well established and tested theories can prove not to be the

correct interpretation of the truth. At the other end of the bridge, the data for the artefact currently being defined change on a daily basis.

The bridge supports in themselves are not enough. For best effect, they must be constructed and linked in the correct order if the desired result is to be achieved. The design process, in its widest definition, is the facilitator allowing this bridge to be built. This is not contrary to the flexibility required of design and the ability to work without prescription and allow shortcuts to be taken where possible.

There was a period of time when the emphasis was on design automation; the current view is now directed towards support of the design process and automated design support through increase access to relevant information [1.17]. If engineers are going to seek the best solution rather than the easiest one, or the one they know about, incentives have to be given and aids to help them in their quest. Some of the major impediments are as follow.

- Time taken to retrieve data. The correct information source is rarely available to hand for the designer, apart from those used on a frequent basis. Provision of information needs to be context sensitive. Considerable amounts of engineer's time are spent searching for information, 18% in the study conducted by Court et al [2.15].
- Filtering required to reach appropriate data, both in terms of volume and format. Information retrieved needs to be correct and relevant to the problem in hand, and must be in a usable format.
- Complexity of domain-specific data and information. While there will always be a requirement for people with a great deal of experience and knowledge in a certain area, benefits are to be gained from tools enabling any engineer to be a 'five minute expert'. At any stage, the engineer can elicit from the system some general idea as to the degree with which their design meets the constraints. They are then alerted earlier to problems, enabling them to iterate the design step, or to refer it to the expert. Experts have thus had low-level routine workload lifted from them and are able more effectively to deal with more complex and demanding problems.

2.3.5.1 Design context

The design process is one of acquiring data, applying it or using it as input to generate more data, and storing the result. Throughout the development cycle, therefore, there is interaction with the knowledge base available to the designer or design team (Figure 2.7). There are three parts to this:

- Quasi-static - representing the mainland of Figure 2.6. This covers science, technology and other design influences that have not been understood, quantified or even recognised, such as a designer's personal beliefs or aesthetic preferences.

- Product data which are stored in some formal way and are available beyond the end of the project. Examples are written specifications, released drawings, bills of materials and procurement schedules. The volume of this information takes some time to increase, as a lot of early work on the project is conceptual and tentative in nature.
- Transient data, which are ephemeral in nature. Examples are information exchanged in conversations, rough sketches made on scrap paper and subsequently discarded, sometimes even the designer's notebook. In the latter case, either the designers can leave, taking the knowledge with them [1.26], or their notes are at a level of abstraction which only they can understand at the time, and which even they may forget, making the notes meaningless. The volume of such data increases rapidly from the start of the project. At the end of the project, as designers move on to new projects and the information in their notebooks becomes ever less intelligible, so the volume of transient data decreases rapidly. Some of these transient data are of no further value and should be discarded, but others are valuable and represent a significant loss to the company when they are no longer available.

The volume of data expands quickly initially, as concepts are generated. Later, choices are made and the volume of data shrinks rapidly, as some potential options are discounted. This can be a measure of the project progress; when the volume of active data decreases, the project is at the decision phase.

Towards the end of the project, there is a transfer of some of the information from the upper levels downwards, from the transient to the recorded. This happens as designers document their work, commit to paper their ideas and write up the project. There may also be generic lessons which can be learned from the results of some of the work carried out during the design, and which may be formalised and then enrich the technological data base. This process is represented by the step in the scientific and engineering knowledge section towards the end of the project. The recorded design data of the current design project and augmented body of scientific and engineering knowledge are available to subsequent projects (this step is not represented in the figure).

Occasionally, the required technology for the product does not exist, and so basic research and development must be carried out in order to augment the body of learning to enable the artefact to be created. An example is the way in which micro electronic circuits were developed to enable the creation of compact video cameras.

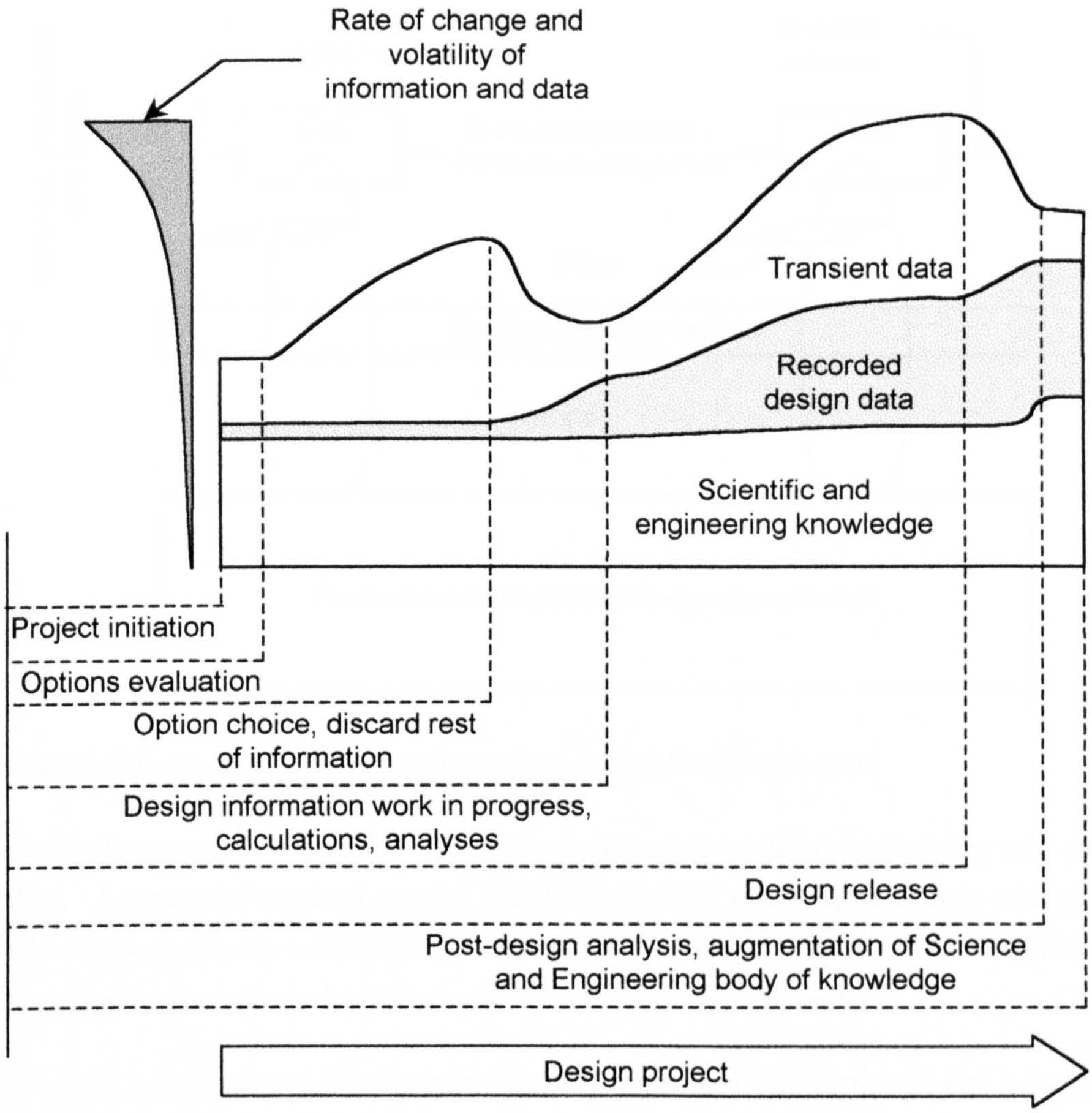


Figure 2.7 – Information profile during the design cycle

Design takes place in the context of the information and knowledge described above, and the final form of the artefact is defined and influenced by this information and knowledge. The sum of this information must be filtered to a usable and relevant level of granularity for use (filter in Figure 2.8). For example, a material strength may be used in design calculations, rather than data about the molecular structure, bond strengths between atom components and what types of quark were present.

Even with this pragmatic subset, many things are difficult to quantify. Matters of taste and aesthetic, personal prejudice and preference very much affect the final product, but are difficult to describe in any meaningful and unambiguous way. The problem partly is that such matters deal with psychology and preference, which are themselves not fully understood. Often, the closest thing to a definition is a collection of examples.

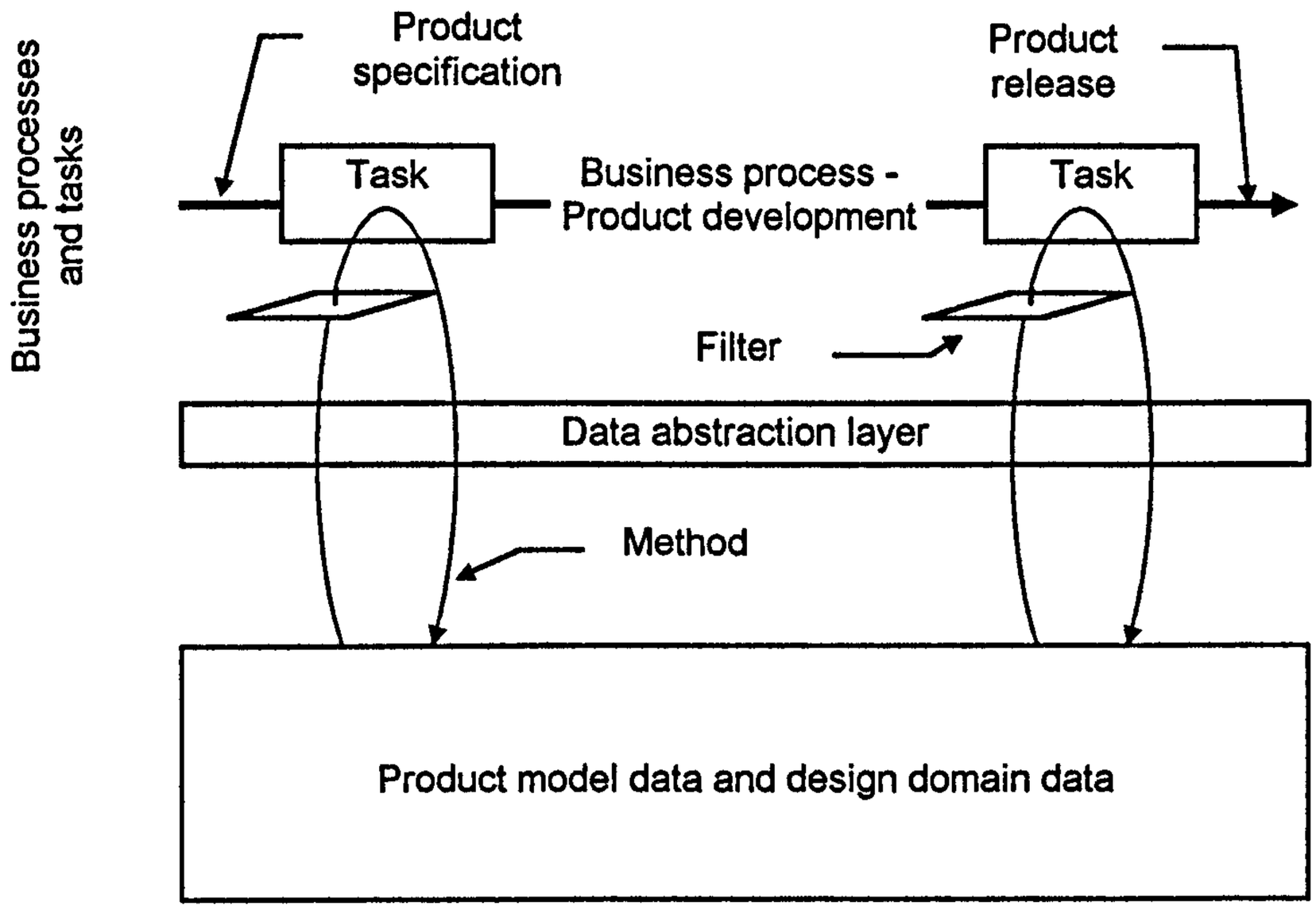


Figure 2.8 - Interaction with information during the design cycle

Methods are used to support the tasks, performing retrieval, processing and storage of data. A powerful method arsenal is able to liberate the designer. It is with this intent that Guide methods are designed, to bring to the user at their point of need the facilities they require in order to be able to perform their tasks efficiently.

There is a requirement for the necessary information to be retrieved and processed, and the result stored. This cycle is driven by methods, which, on their own or in combination with other methods execute a task. The method can invoke a filter to deliver to the user the information they require in order to make a decision. It will also perform whatever translation across the data abstraction layer is necessary.

Progression of the design relies on active agents and knowledge. The active agents will involve designers to different degrees. Some tasks require the designer to take all the decisions and perform all the calculations necessary. Others require the designer to initiate the task, which then performs its methods automatically until the outcome is reached. The last category is tasks which are programmed to happen automatically, and for which no designer involvement is necessary.

Communication of the current design status to all who participate in the activity is in itself a complex task. This problem is accentuated when design occurs at several locations of one company or across a consortium of companies. This situation is becoming ever more prevalent as development costs soar and companies forge closer links with their suppliers or collaborate on new projects with other companies with similar or complementary skills. The product model must support the communication of

data and the methods and controls applicable to those data to enable the design function to be distributed, or its results disseminated. The product model is an excellent basis for communication as it contains all the product information, links and details of the methods that created them. There is then traceablility from any part of the product model to any other, even when these are distributed among dispersed actors contributing to the design project.

2.3.5.2 Design knowledge: volatile substance

Completing a design is not the end of interaction with the product model data for the artefact in question; rather, it is the end of the beginning. Much will be done with the data before it is no longer used, even after the product is no longer manufactured. Examples of uses for these data are shown in Figure 2.9. The level of ease and effectiveness at which these processes occur is an area where large improvements can be made.

The data degrade with time. Some of them are held in a designer's head, or in their logbook. Even if the designer remains with the company, their ability to remember problems solved or interpret their own notes decays significantly over short periods of time. Some of the data are forced into inappropriate storage locations by the business operations. All these factors reflect in the decay in quality and integrity of the product model data shown in the top part of Figure 2.9. Note that this line refers to the sum of the product model data- some of these will suffer no loss even over considerable periods of time, such as electronically stored CAD data. Eventually, though, as software systems evolve, there may be a time when the data may not be read, no matter what their quality and integrity. The length of time over which this happens probably makes it inconsequential since the product is likely to have become obsolete and be out of service before it happens. The short-term re-use of design data is the more important consideration for a company.

Data and relationships, unless captured, are rapidly lost from the consciousness of the designer. The result of the design work is evident, but none of the information relating to design rationale, arguments, iterations or path [2.16]. The loss of data and relationships has several implications. Firstly, during the design project, it can result in areas of the product data model having to be re-worked, calculations performed for a second time, perhaps even mistakes re-made and corrected before processes downstream of the original design can function. Secondly, less information is available when products are being re-designed. Thirdly, if knowledge is extracted from a design and incorporated into the general body of knowledge, it is less likely to be lost and is more easily applied to other problems.

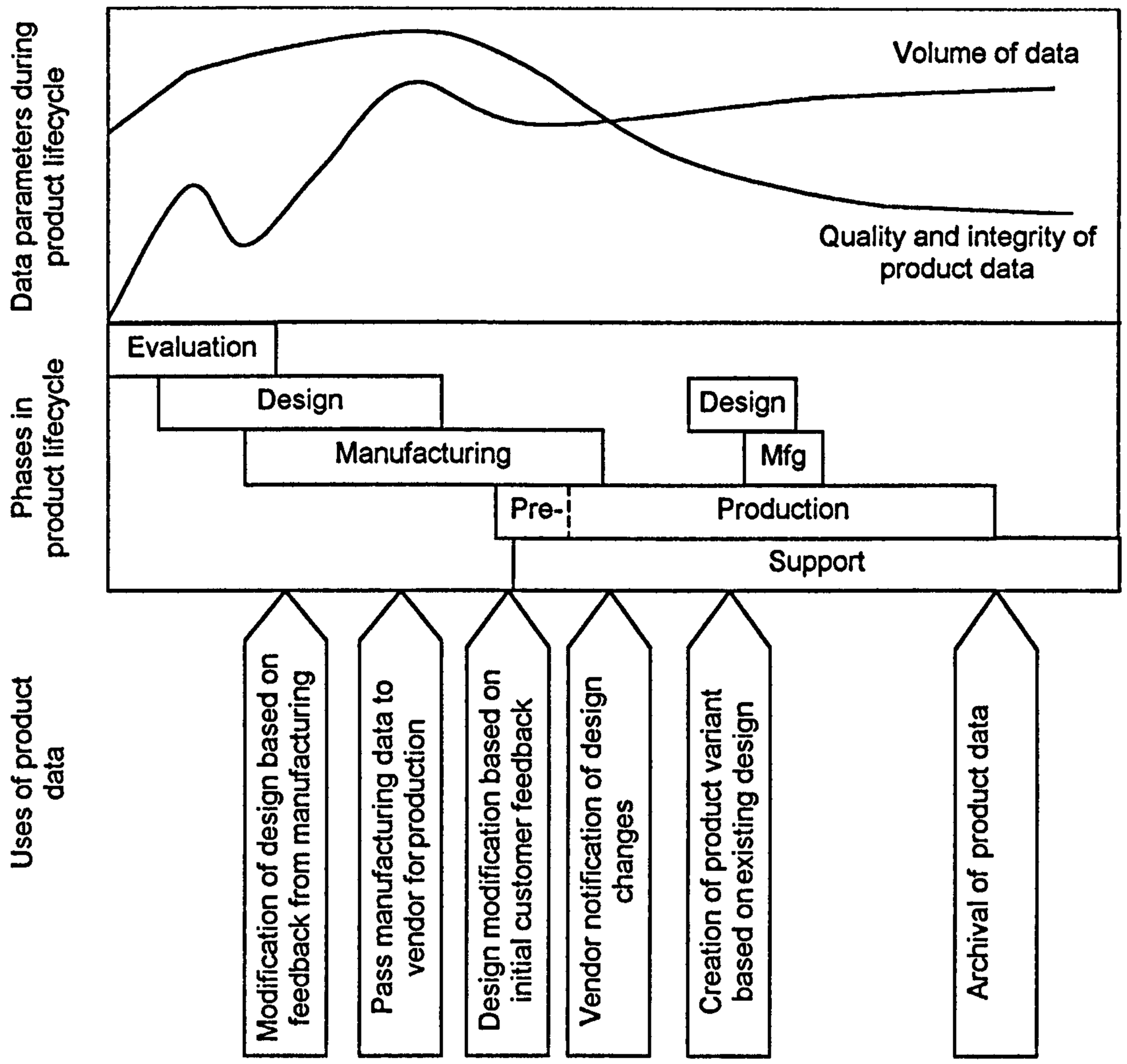


Figure 2.9 - Design information use and decay

The information repositories require periodic maintenance to remove data that are no longer required in any capacity, a process often known as "garbage collection". A proper model with explicit links assists in this process. It can be established whether the data or procedure about to be removed is still in use in other parts of the enterprise, in which case its removal would leave gaps in another product's data. The dependants of the element being removed can be traced and removed, so that no orphans are left.

2.3.5.3 Representation of the information model

Through the process of design, design data are created and populate the information model. It is the agglomeration of all the information accessed or created during the course of the product development.

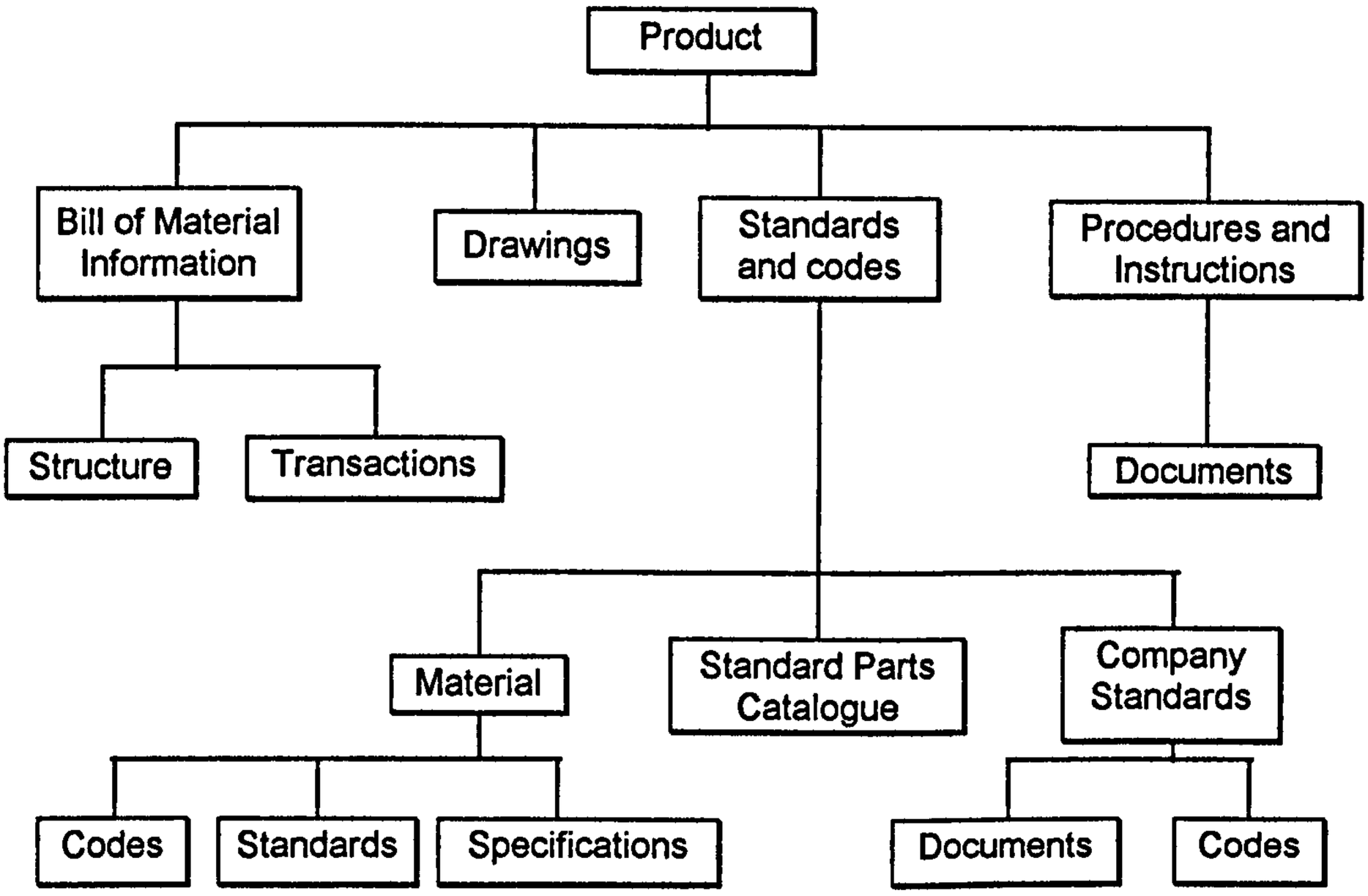


Figure 2.10 - Product information distribution and representation

Figure 2.10 shows an example of an organisational chart that could be used by a company to describe its product-related information. Several shortcomings and weaknesses are present when it is compared with the product model, which has the flexibility to store any sort of data and their inter-relationships and can support processes on those data. Furthermore, even in such a simple schema, there exist discrepancies between the model and its actual implementation. These shortcomings are described below.

- At the most fundamental level, parts of the schema are not modelled at all in such an information organisation: marketing research for example (boxes missing from Figure 2.10). The information that is modelled may not be complete (the boxes may not be full). Not all of the links (shown as line between the boxes) are represented in an explicit fashion. For example, some of these links will be contained implicitly on an assembly drawing and must be deduced from the geometry and positions shown. Where the links are represented, the description of the link may be held in an inappropriate or inaccessible location. This is the case with notes on component drawings that refer to material specifications or conformance standards. In order to collate a full list of such specifications for any product it is necessary to find and read every drawing for that product and manually collate the list - an error-prone process and one costly in time.

Another instance of this situation would be a specification document held on a stand-alone Personal Computer that is not available in read mode across the computer network. To access that specification requires knowledge as to its physical location, the ability to extract it from the knowledge repository (familiarity with the software used to store it) and effort on behalf of the engineer to go and retrieve it.

- It is difficult to introduce new data types or procedures, because of the overall rigidity of the model.
- The next layer up in the information and process hierarchy comprises the applications that manipulate the data at various stages in the product lifecycle. These store their information in various internal formats and do not have the capacity to store explicit links to related information. For example, a word processing package used to produce a specification document cannot be interrogated in order to retrieve and display the drawings for the associated product.

  A roadmap or data dictionary is required in these cases to represent the links, lying above the applications and referring to data generated by them. This approach brings with it its own problems, as there is now a layer of metadata, the accuracy and version of which the company must control. If the data being pointed to should change, there may not be any mechanism for updating the metadata, rendering them out of date. Any procedures introduced to prevent this circumvention of the data management schema adds to the rigidity of the system as a whole and to the likelihood of its early obsolescence.
- Processes and their data are optimised for one function and against the parameters important for that function. Different disciplines or functions lay claim on part of the data and will want to control its format to support their needs, without concern for the needs of other users of that information.
- In addition to the division by function and software described above, there is division of data by department or business unit. Furthermore, departmental divisions do not map directly onto functions, and subsequently applications. There is no clear business unit - function - software package correlation, so there are three problems, not just one with three facets.

One of the results of this organisation is that it is often difficult to extract new sets of information from the product model. For example, if a company is organised according to distinct product lines, the introduction of configuration management practices may cause difficulties in managing parts that span product lines. Equally, it is difficult to map new processes or to reorganise into new business units and maintain the same data model. In this case, the interfaces with the data need to be redefined.

The combined effect of the fragmentation described above means that a seemingly simple task such as retrieving a complete, accurate and up to date set of data relating to one product may take a considerable amount of time. Its success cannot be guaranteed, if there is loss of connective information in the data model. Not only does this introduce costs and extra iterations in the design cycle, it increases the probability of serious defects being left in the design of the product. This is reflected in part in research showing the large proportion of time spent by engineers in retrieving data.

2.4 Requirements of the product model

The challenges in the marketplace to which business must respond are in a state of flux, requiring the ability to respond rapidly to them or lose competitiveness.

The current fragmentation of the business model and its supporting data and information structures act as a damper to this response. The fragmentation is several fold, and includes lines drawn between departments, functions, locations, procedures, software utilities and information. After considering above the weaknesses in the current method of representing and managing information and data, the necessity for the product model and its content is affirmed.

A free flow of information needs to happen. The corporation needs to be able to act as a single engineer[2] might, who has instant access to all entities and relationships and who does not segregate activities into pre-determined types. This is particularly important in the early stages of design, when vague and imprecise definitions take form. It is important at this stage to be able to support these definitions, but also to ensure that downstream constraints can be considered. The more completely can early design be supported, the more streamlined later detailed stages of the process will be.

Control and audit functions are important monitors for a company of the health of its processes and projects. Communications are important to support functions interdepartmentally and inter-locations.

---

[2] The lone engineer is not a perfect model either, as some of the information and transactions are forgotten and poorly documented, if at all. The important attribute is the lack of pre-constraint on the way that such a unitary agent acts.

2.4.1 System requirements

For the product model to be successfully deployed, computing support tools are required. The specification for these utilities with respect to different functional areas is outlined below. Further justification and explanation of the requirements proposed are given in the following chapters, where these issues are dealt with in greater depth.

The system must:

- be adaptable to the requirements of each company,
- be able to represent all the elements used by the company,
- capture a design history record,
- support audit of the design and design processes,
- store company knowledge in secure forms,
- apply the desired level of control on processes,
- promote and support open communications between company functions.

3. Guide Facilities

3.1 Guide support of the design process

Lateral thinking, innovation and decision-making abilities are quintessentially the qualities of designers. These qualities add value to the design project, and cannot be contributed by machines. There are many other tasks necessary in the completion of a design project that do not employ the designer's talents to their best potential. Guide offers assistance with these tasks, leaving designers free to apply their creative and problem-solving talents.

Guide is designed to operate intuitively to the way designers think. The user is not required to know which elements or functions of Guide support their activity. Guide does not impose a high overhead of learning on the user pre-requisite to its application.

Rohatynski and Dabrowski encapsulate the requirements well when they describe a 'partnerous design system', the characteristics of which are that it helps designers execute the tasks they want to perform, without precondition on what those tasks are [3.1]. It should also help them to access the information they require in order to execute their chosen tasks.

Brissaud and Garro [1.22] recognise that the design project requirements are as much for management as technical solutions, as do Park et al. [3.2]. Guide accommodates this requirement by providing tools and an architecture through which management and design procedures can be devised.

3.1.1 Design operations

The end product of design is a description of the product or service to be delivered, together with information relating to any other aspect of its lifecycle such as costing, manufacture, maintenance or documentation. This information is held in Guide as product model data, and is shown as the top level in Figure 3.1.

From Figure 3.1, it can be seen that Guide represents the product model data as a continuum, with links made to relate entity instances. All of the project information is held in this fashion, so that a given entity instance is accessible from all other instances. This allows decisions to be made based on all of the business information, including that bearing on financial and technical issues [1.16].

As the design unfolds, entity instances are added to and removed from the product model data and links are created and broken. This state of flux and adjustment continues until the project management is satisfied that a solution has been constructed.

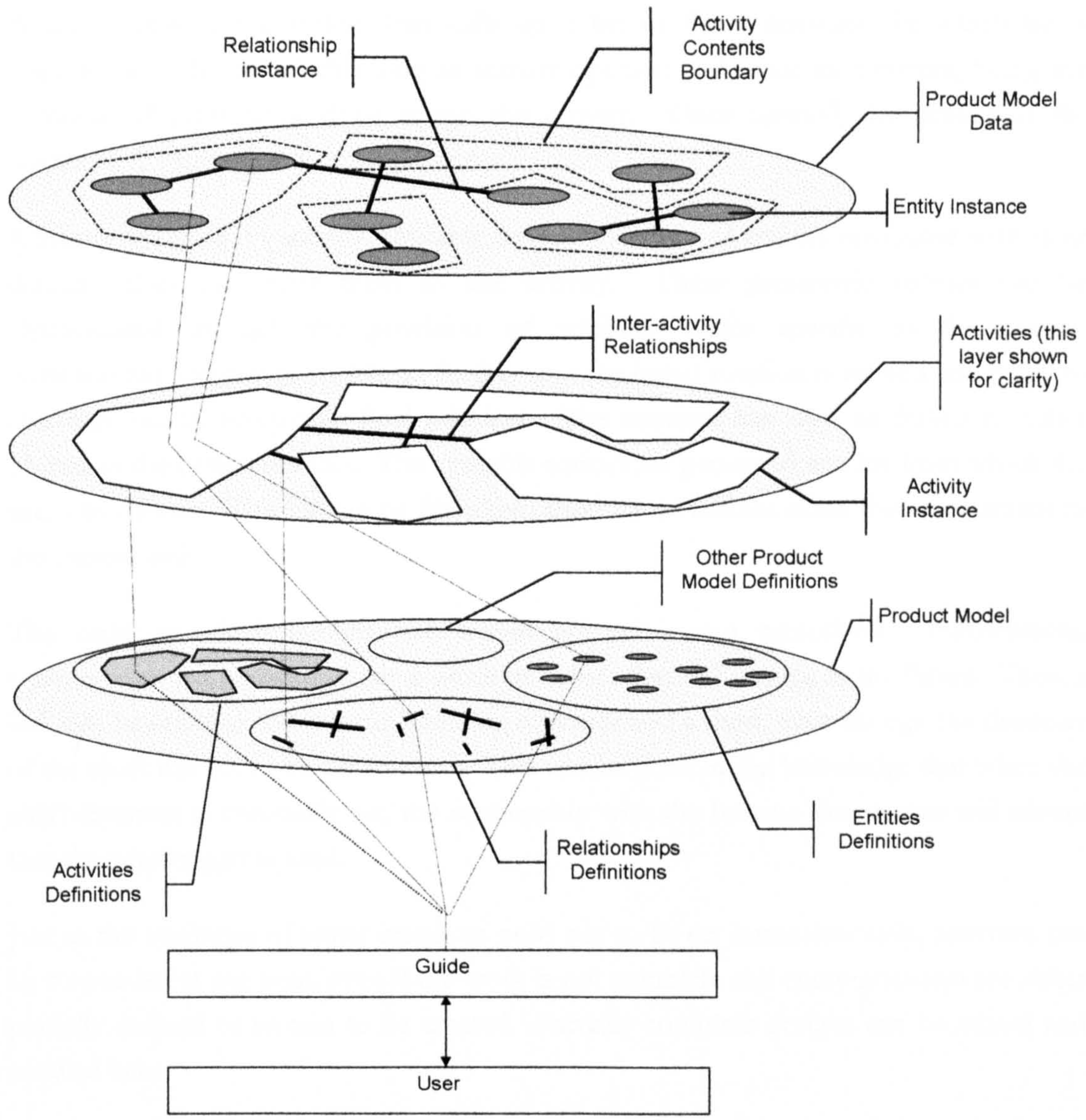


Figure 3.1 - Design workspace

The working environment in Guide is the activity. An activity provides a window onto the product model data, revealing those entity instances which are relevant to the task in hand; it is a tool to manage the design project and assist in ensuring its timely and satisfactory completion. Activities are the dashed lines of the top layer and form part of the product model data. For clarity, a projection of their outlines is shown on the next layer down.

A statement of objectives is part of the definition of an activity. This provides a specification towards which the person responsible for the activity can work. The aim of the designer is to construct a set of objects that represent the solution which best meets the specification. Guide enables the user to meet this aim through the creation of entity instances and relationships that are added to the product model data layer as the contents of the activity under which they are created.

A user accessing the system first calls up a list of those activities for which he is responsible. The act of choosing an activity opens it and loads its contents, being the outcome of prior work done within the activity. Once opened, the activity is the container for any subsequent work done.

Whilst working in an activity, the user creates instances of entities attributed with their default values and adds them to the activity. These parametric entities can be characterised through the provision of attribute values specific to the current environment and design problem. In this way, the initial solution is refined and made to converge on an acceptable final solution. The instance can also be linked to other objects in the design domain. The available entities are presented in a list from which the user can choose. This list can be filtered to show those entities alone that are relevant to the current task.

The order in which the attribute values are set is not prescribed. Furthermore, relationships can be established on attribute values that will change in the future. Thus, a link can be established between a bearing diameter and a shaft, even though the diameter of the shaft has yet to be determined. This can be done in the knowledge that when the shaft diameter is eventually set, the relationship with the bearing dimensions will ensure that the correct part is used.

Just as the attributes of entity instances need not to be set instantaneously, activities can be suspended at any time, even if the work is not complete and entity instances are either partially defined or remain to be created. Partially complete designs can be stored and recalled later.

Once all the values of the entity attributes have been set, the entity is committed. This is akin to a conventional release process for engineering data. Given that it happens on an individual entity basis, and at any level and stage of the design process, it is a more flexible way of controlling change in the design. Change to the instance after it is committed is achieved by creating a new version of the instance. This initially carries the same attribute values as its parent, which can then be changed before committal. During the committal process, integrity checks are also performed on the entity; these are described more fully later.

Links also exist between activities, such as were described in chapter one (represented by the lines linking activities in the middle layer of Figure 3.1). Principally these are links that describe the administration of the project and enable navigation of the design workspace according to the manner of its partition. These links are additional to those existing between entities in the product model data layer.

The product model data and activity descriptions are stored as data only. In order to make sense of the data, and for their values to have meaning, the product model definitions are required. These are the templates, the generic definitions, which lend the data meaning. Thus, the user accesses the product model data through the product model. The product model objects that are required include the entity definitions, activities and relationships used as the constructs of the product model data. It includes definitions of the methods and processes that operate on entity instances during the design process.

Often, information at the top level is managed by product data management systems. These are little more than extended file management systems; they have no capacity to model relationships beyond simple project ownership of data [1.9]. The latest initiatives in this area are termed Product Development Management, or PDMII. An example is Dassault Systèmes and IBM's Enovia. The core function of PDM is to manage engineering change to component geometries and the CAD data files that represent the component. PDMII adds to these the ability to maintain links to non-geometric information. It extends the metadata set beyond knowledge about CAD files and their locations. They are still very much rooted in the detail design and manufacturing phases of a design project and being external to the design process add layers of complexity to the information management overhead of a company.

3.1.1.1 Information types

Various types of information can be extracted from the product model data, by the application of a filter to the information available. Figure 3.2 shows the extraction from the design workspace of entities that describe a part and activities that encompass it.

The examples shown retrieve assembly instructions, with the purpose of creating a complete assembly instruction manual. This would be necessary if the product assembly were sub-contracted. The lower example of Figure 3.2 shows a filtering operation to retrieve all parts. This would form a Bill of Materials structure with the links between items being maintained.

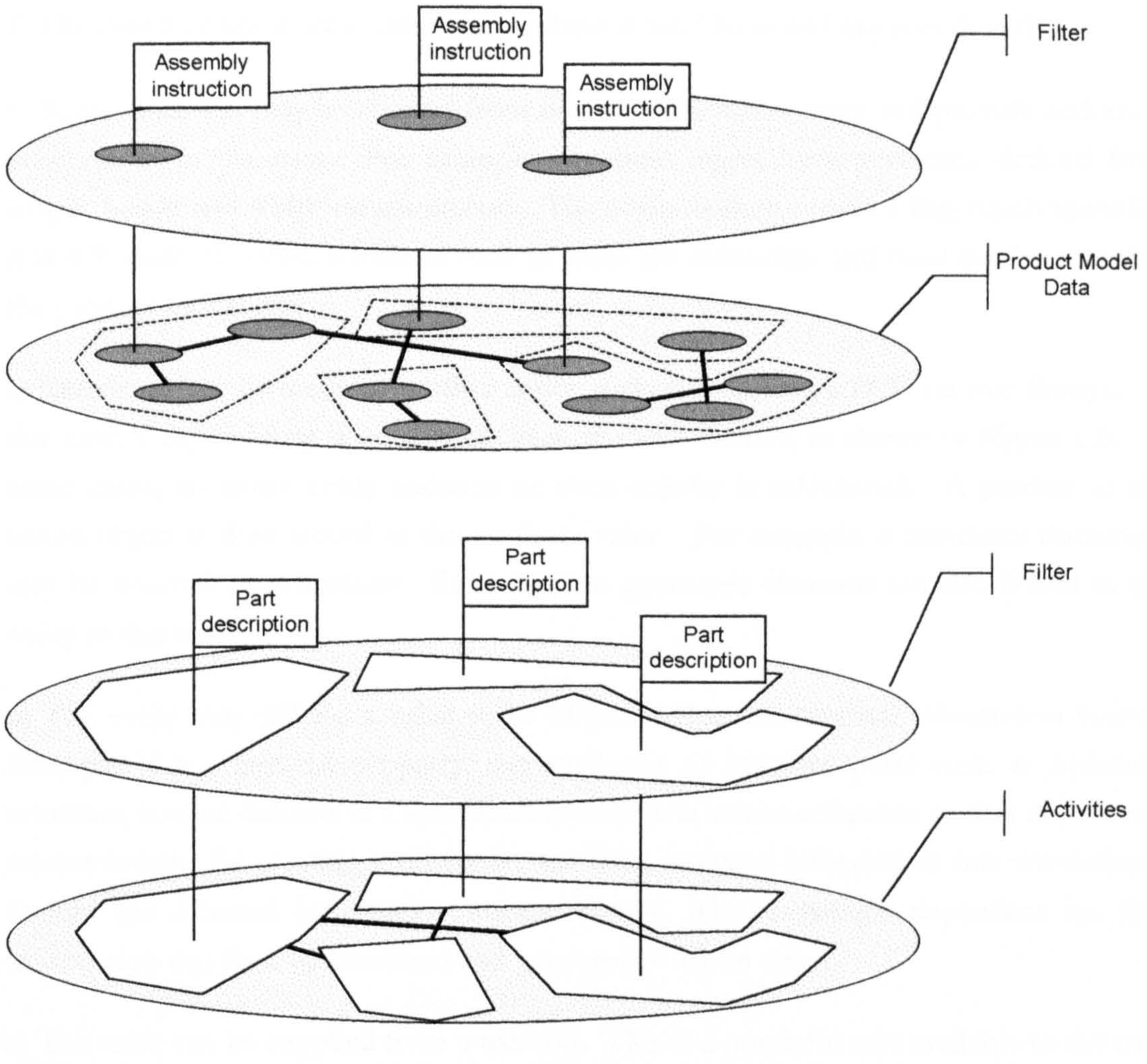


Figure 3.2 - Filter extraction of information types

3.1.1.2 Attribute value provision

The provision of values to the entity instance attributes is the process by which the desired characteristics are mapped onto the instance. The sources of values for these parameters are as shown in Figure 3.3.

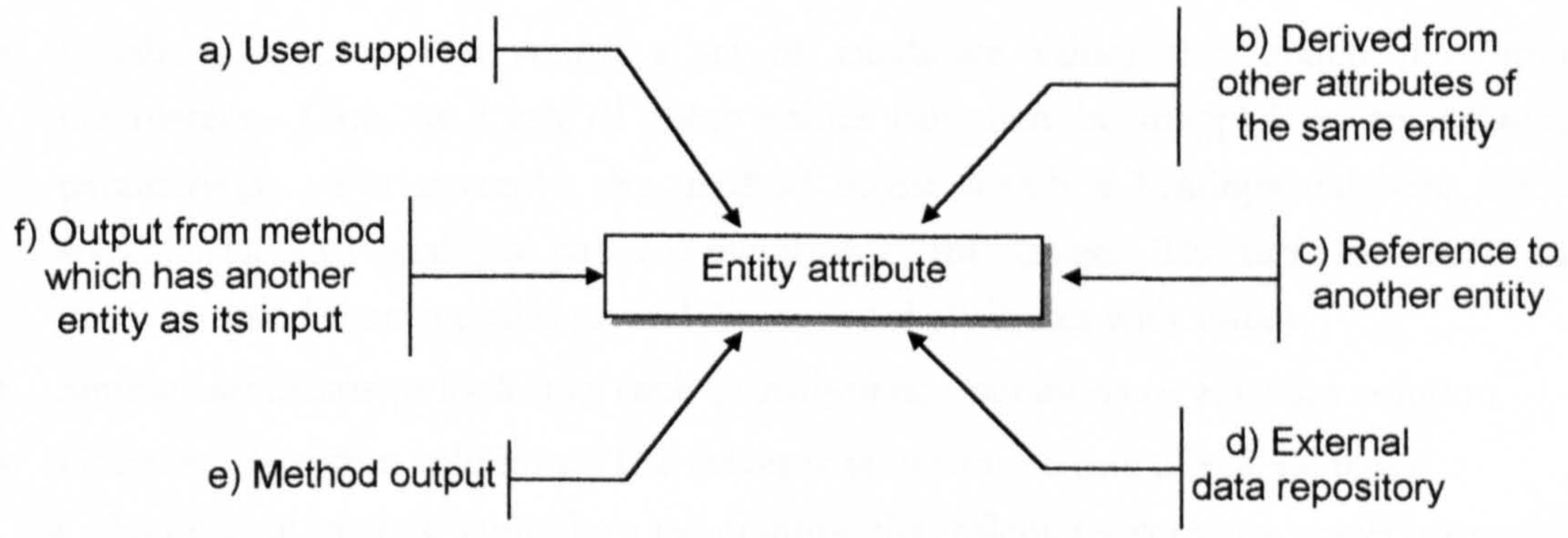


Figure 3.3 - Sources of attribute value

a) The user may know what value the attribute should have and key it in directly.

b) Some attributes may be derived from others of the same entity and provide additional information to the user. For example, a cuboid might have a volume derived from length, height and width measurements. The attribute then carries a flag which identifies it as a 'broadcast' value; attributes such as these are redundant and need not be stored in the product model data.

c) Reference may be made to another entity, such as matching a PCD on two flanges. In this case, a dependency is created between the two entities, as shown in Figure 1.6. In some cases, an entire entity instance or even activity is referenced. A pointer to the linked object is then stored as the attribute value. For example, a standards document may be referred to as a whole. References to geometric elements are also linked to the entity in this way.

d) The entity may call for a value to be obtained from an external information source. Examples are a material property, the attributes of standard parts such as hydraulic actuators, a value defined in a specification such as a colour reference or that day's stock market index. By creating a reference, the links between information sets are defined. Should the external information source change, all the entities dependent on that information can then be identified and appropriate action taken.

e) The value can be supplied from a method. This is a powerful tool available to the user that makes available the processes and information the user requires at the point of need. Guide supplies a list of methods valid to supply values in the current context. These methods can be linked either to a particular attribute, or to an entity in general. They may also be utilities that are available globally within the activity. The method may be any mechanism that supplies information back to the system or user. Examples of different types of method are listed below.

- Database searches that return a set of candidate values that match the input parameters. One, or a set, of these values can then be mapped on to instance parameter(s). For example, the method might search a bearings database for a suitable standard catalogue part for inclusion in the design. The tuple chosen from those returned then supplies several dimensional attributes with values.
- Simple algorithms or look-ups, such as arithmetic operations or equation solution.
- Complex algorithm solutions. The solver may in turn be one of several types:
  - An internal method could be a programme that calculates pressure vessel geometry based on the operational parameters provided.
  - A third party thermodynamics analysis programme could be invoked to determine the required dimensions of a heat sink.

- A human expert in the field, whose answer to the problem can be relayed to Guide.

f) Where the method has a value taken from another instance as one of its inputs, a link is created to that instance, since it affects, through the associated method, the value given to the instance attribute.

Together, these methods provide the user with access to relevant information and processes that aid their work. Their ease of access assists the maintenance of design solution integrity and conformance to applicable standards and specifications. Concurrency is an exercise of the design function such that each discipline delivers to the process the information required at the earliest moment the design process is able to assimilate and put in context that information [3.3]; methods permit this to happen. It is important that the design methods can be changed, without affecting the design methodology, i.e. some measure of abstraction is introduced [1.1]

The provision of methods directly into the design workspace also offers users access to solutions software to make them 'five minute experts' at the problem in hand. They do not need to search for the appropriate software or transfer input and output values to and from the design workspace. This relieves experts in the particular subject of performing many simple analyses, and allows more time to be devoted to the solution of complex problems referred to them through Guide.

This area is sometimes considered the domain of Expert systems. However, production systems are not suited to the complexity, ill-defined nature and lack of prescribed procedure that characterise design [1.9]. Expert systems are, though, powerful tools and may be deployed as a Guide method. They can incorporate non-geometric and process information [3.4]. They suffer from brittleness, however: it is difficult to apply the rules defined to them into other areas and so take advantage of the generative nature of knowledge [3.5]. Their set-up costs are high and they require the input of expertise from individuals capable of solving problems in the domain considered, to formulate the rules which bound the problem in a concise way, acceptable to the knowledge processing engine of the software tool [3.6]. This has an implication on the justification of the high costs of such systems, given that they cannot be used outside their narrow area of application. One piece of evidence for this is the frequency with which a single ICAD-implemented example is repeated in the literature.

The problem of establishing rules that have multiple applications is a common one. Finger et al. [1.11] discovered this in their research on concurrent engineering, where research directed at one problem was found to be inapplicable to another closely related one because of slight changes in base assumptions. Because of the association model for

Guide methods, they are available for any process or entity for which they are of utility; in short, they are designed for maximum re-usability.

3.1.1.3 Access to external software

While Guide requires the product model to be described to it, it does not demand that all the processing be carried out by it. The availability of highly effective software designed to carry out specific functions is recognised. It would be purposeless to require that all processing methods be described to Guide as methods, even though this is possible. In order to use the software, then, an interface needs to be constructed between Guide and the software application, for both the input of data and the reception of the results.

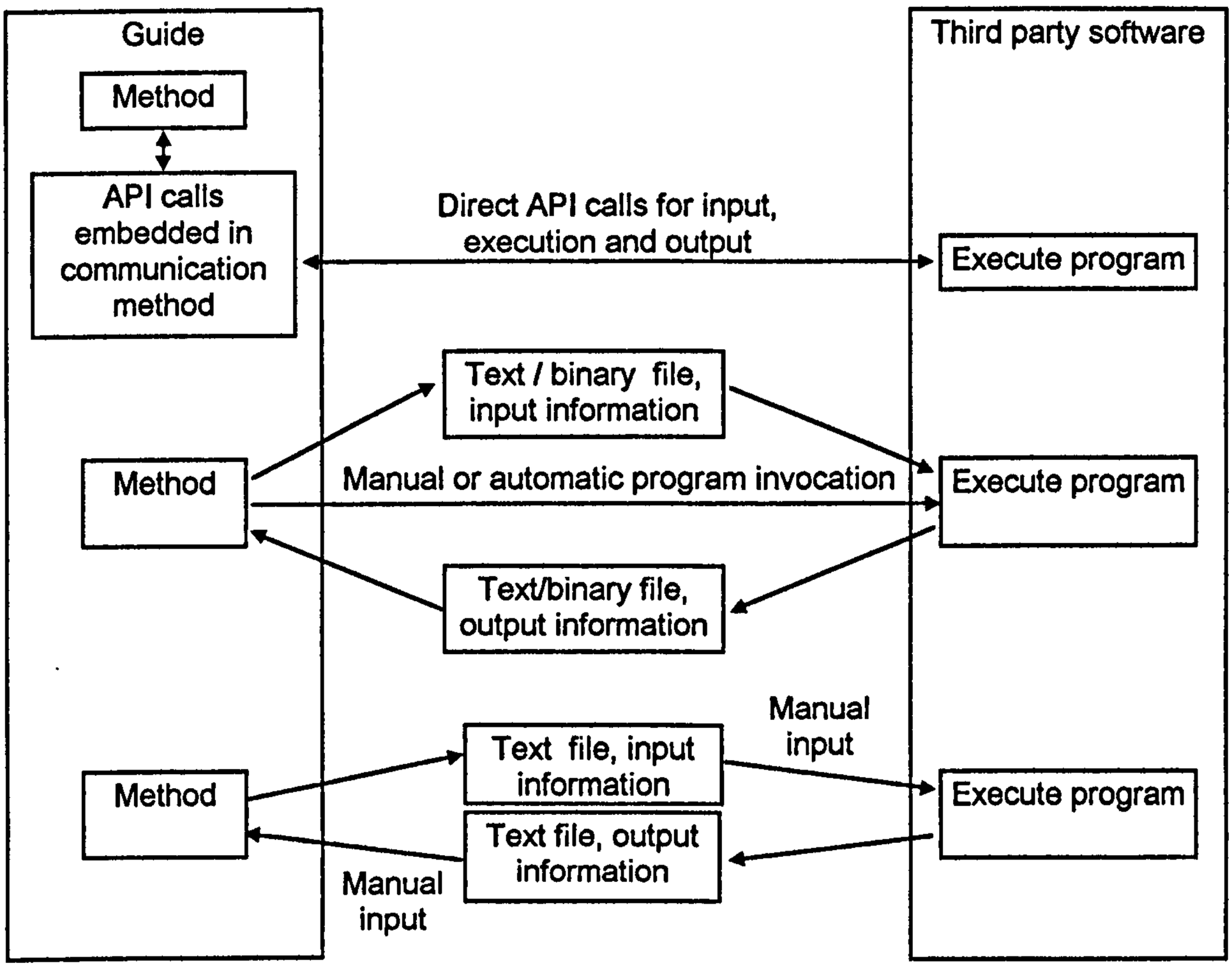


Figure 3.4 - External software invocation options

Various degrees of integration are possible, according to the degree of openness of the third party software. These are shown in Figure 3.4. The preferred route is to have available access to the ***API*** of the other software, so that direct communication can occur, without further involvement by the user. This is not always possible, so exchange through input and output data sets in the native form of the software is supported. The program itself may be triggered either directly by Guide or manually. The output again comes in the form of output data read in by Guide. In some cases, the software is completely closed. In such instances, Guide provides a text file with the values to be input manually to the software, as if it were being used in isolation. When the program

has completed its execution, the results are fed back into Guide manually. This last method may also be used when consulting a human expert. The problem parameters are given to the expert on which basis they formulate a solution, document it and return it to the design team or individual who initiated the request.

3.1.1.4 Constraints

Constraints are applied at various stages during the entity instance creation and attribute value setting processes. A constraint is a method, defined in the same way as is any other method for design support, described in the previous section. The invocation of constraints is shown in Figure 3.5. Constraints act as checks on the validity of the work done, and as assistants in the design environment. They help to ensure that appropriate standards are met.

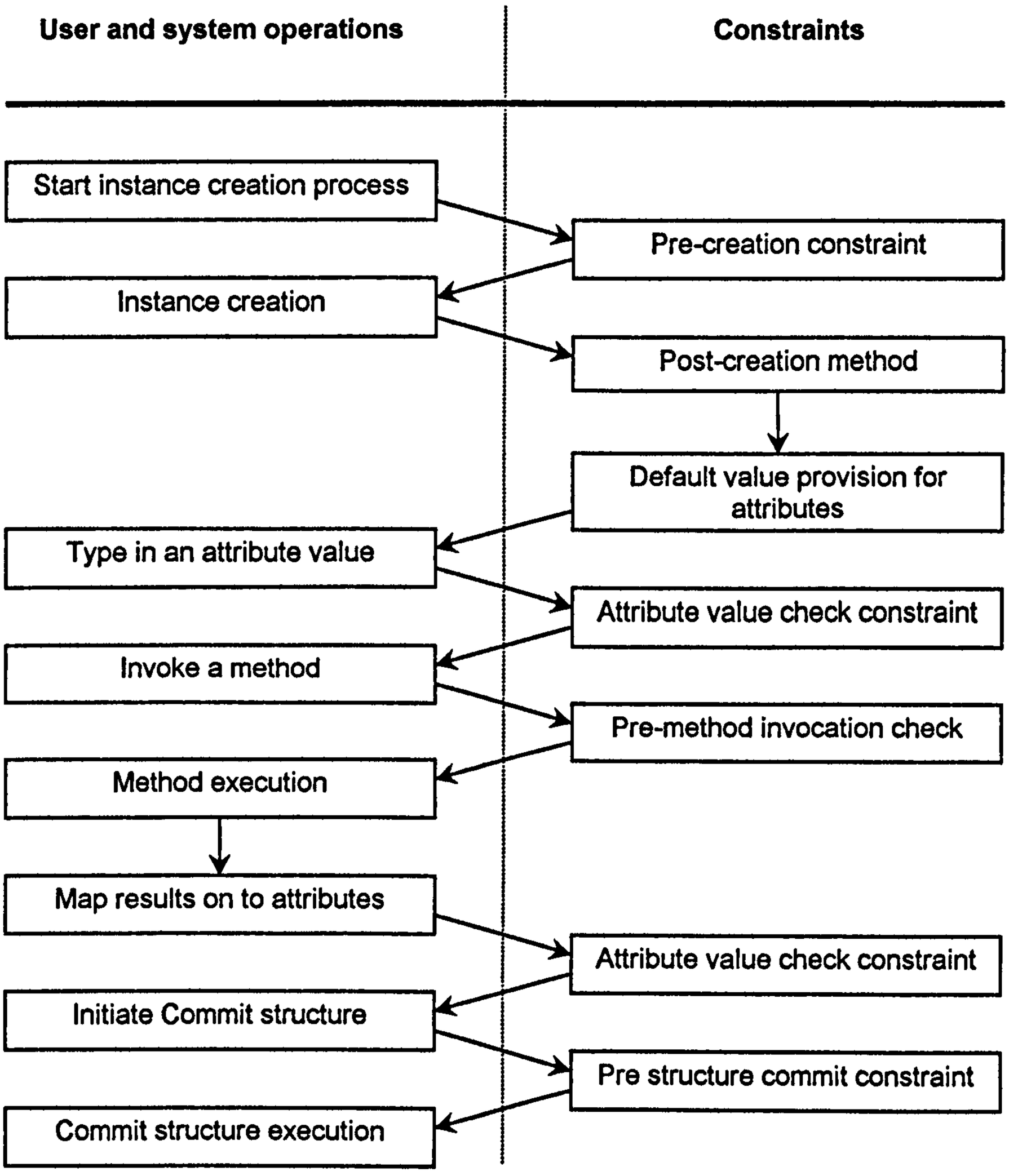


Figure 3.5 - Constraint execution schema

The first constraint fired verifies that the conditions are acceptable for the creation of the proposed entity instance. It may also perform operations that prepare for the creation of the instance. When creating a product cost entity, the system can check for a Bill of Materials (BoM) in the product model data. If this does not exist, it can cause an instance of the BoM entity to be created.

The instance is established with default attribute settings, being simple values held with the entity definition. If the default value relies on a calculation being performed, this is done through the constraint that supplies values either to a single attribute, or to a collection of attributes. For example, if an instance is created of a power cord entity, the size of the plug and the number of pins can be determined based on the destination country of the product.

Any change to an attribute value initiates a check on the validity of the new value supplied. Any constraint may be applied, from a simple logical check (a hole diameter must be greater than 0.2) to a complex operation (the size of an elevator in a building is related to the number of floors and the surface area of each floor). Constraints applied at this level help to ensure that consistency of the product model data are preserved after design actions [3.7], where controls only on operations may not.

When a method is invoked, a check is performed to ensure that the method can be fired at that stage of the process. It is not appropriate to invoke a lengthy Finite Element analysis if some of the geometric dimension information is not valid.

When the entity instance is committed, a check is performed on the integrity of the instance and its attribute values. Other operations such as sending notification of committal of a key project element to the chief designer may be carried out at this point.

3.1.1.5 Change propagation

Since Guide provides the user with the ability to link and relate entities in a variety of ways, it also provides the mechanism to propagate a change in a value to any dependent values linked in the system. Dependent values may have further dependants themselves, resulting in a change cascade across the system. The size of the cascade depends on how fundamental the changed value is. If, in a car design, the number of spokes on the steering wheel is changed from four to three, it has few consequences. If the number of cylinders is changed from four to three, however, a large number of engine components are affected.

Once the dependency network is established, the change is proposed to each value. An instance attribute carries an 'Action on Change' property, the possible values of which are shown in Table 3.1.

Action on Change property value	Effect
Ignore and do	The change is performed without feedback to the user. The user may not be aware of the change.
Warning and do	The change is performed and a warning or information message displayed to the user.
Alert and wait	Permission for the change is sought from the owner of the entity on which the change is made. A message is sent to person responsible for the activity in which the value resides. The owner decides whether to accept or reject the proposed change.
Refuse and abort	The value cannot be changed as a result of change propagation. The change request is abandoned.

Table 3.1 - Instance attribute action on change values

Where the change request proceeds, the attribute value is changed and the attribute value constraint is fired. Should this fail, the change request fails. In each case, the action is captured in the design history record, whether or not the user was aware of the change.

In the case of change failure, the change can be abandoned, or the link between the related values broken. This relaxes the constraint on that part of the product model data and allows the evolution to take place. A notice is recorded of the attempted action.

3.1.1.6 Process concurrency

Design happens in a highly non-linear fashion, across different departments with activities at company management, design management, detail design and specialist levels. Teamwork has been recognised as a major, if not the biggest influence on concurrent engineering [3.8]; Guide has robust facilities for managing projects and the teams that execute them. It allows the design team to evaluate in real time the effect that their decisions are having on the product, the project and against the design specification; this requirement is similarly specified by Fohn et al. [2.2].

Concurrency has been defined as:

> *A systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality cost, schedule, and user requirements.* [3.8]

In the preceding text, many of the features of Guide that support a concurrent approach to design activities have been explained. A summary of these follows, with particular explanations of their role in support of concurrency.

- The structure of the design project can be laid out rapidly and at every level of operation, before the details of the design solution are worked out. The recursive nature of design project management through activities allows a uniform representational format for all the information associated with the project. That is, activities can represent knowledge at any level in the design project, and can spawn and link to other activities at a higher or lower level. Exchange of information between disciplines is then routine through inter-activity links. There is no limit on the level of detail provided on the part, as Eastman [1.6] states, *a design can always be further specified.* Guide provides for this by its flexible recursive structure that can be extended to any depth.
- The product model data are not decomposed but are constructed as a network of interrelated entity instances. The use of activities segments rather than decomposes design problems and maintains links between related items explicitly. The contents of each activity are accessible from others, so relationships can continue to be built between entities, regardless of the activity in which these are held. This is an important point, since concurrency in research is often though of in the context of a monolithic design problem with no decomposition or segmentation applied [3.9]. This overly simplistic point of view ignores the dynamic of multi-contributor design processes. For management purposes, activities provide context for work to be done and a specification against which to work.
- Partially complete design entity instances and design activities are supported.
- A rich set of links can be defined. These can be among entity instances in an activity, to an activity or to entities in other activities.
- Links can be defined to design objects that are not fully or completely defined. Change propagation mechanisms then ensure the continuing integrity of the solution.
- Constraints can be established to ensure that the evolving design solution fits within the boundaries of the specification and each part of the design solution is in accord with the other. The constraints can include the user of agreed, common geometry against which parts can be designed.
- When necessary, constraints can be relaxed, with due warning to enable appropriate compensatory action to be taken.
- Communication is supported among all the agents involved in the design project. Product model data can be referenced in the communication and referred to by all parties.
- Links with sub-contractors are strengthened to improve the information flow between parent company and contractor on design issues.

McGreavy et al. [3.10] propose similar elements for the support of concurrent engineering to those used by Guide: design objects, design teams and a project manager.

They emphasised that it is not only computing or even communication tools that will enable concurrent engineering, but the framework which supports that type of activity.

Boothroyd and Dewhurst [3.11] were early proponents of Design for Assembly (DfA), in some respects a precursor to today's concurrent engineering philosophies. An example of their work is a system which interrogates the model to determine fastening the methods used, symmetry, size, angles of insertion, etc. and give estimates of the assembly time and efficiency on that basis. Their work aimed at establishing the rules for DfA, and making them available to designers as a stand-alone system.

Kunz et al. recognise the necessity of expanding the concept of concurrent engineering beyond DFA and design and manufacturing. They call this CE4, to reflect the four areas of product, process, facility and organisation addressed by the CE protocol. Yet, they did not propose a methodology for attacking the problem [1.18].

Guide provides the framework to support DfA and CE4 rules within the larger business context. This goes beyond support for DfX to XfX, the ability to examine the wider implications of ones work, irrespective of the domain the work is carried out under.

It is to be expected that the deployment of Guide in the company will stimulate new approaches to problems and will promote innovative ideas. This was found in a less dramatic and fundamental design for assembly project, where the cultural implications are less wide reaching; even so, forcing different disciplines to intercommunicate led to simple, elegant and effective solutions [3.12].

3.2 The design history record

The most up to date information on a design project, its latest state, is only a part of that necessary for effective operation. The history of the development of the project is also critical to design. This includes the rationale behind decisions, the context in which they were taken and the data produced by subsequent actions. Indeed, Guide's performance of some processes depends on access to the design history record. Some of the functions described have yet to be incorporated in the current version of the Guide software; the issue is one of implementation rather than technical difficulty, as all the pre-requisites for a successful implementation are satisfied.

3.2.1 Current design support

When confronting the need to make changes to a design or design management problem, the decision making process is often poorly supported by information. Decisions are made which attempt to rectify a perceived rather than an actual problem, or address the symptom rather than the cause. A design history record helps to address this difficulty, as it provides designers with the facility to view and analyse information relating to the project. They can then review decisions made and search along dependency branches for root causes of problems, travelling back up the evolution trail of the design until the problem is found. This will allow designers to proceed on the basis of actual, rather than perceived, data [3.13].

3.2.1.1 Retrieval of past product states

The information captured in the design history record is sufficiently complete to support a replay of the design process that originally created it. Each successive stage in the design from project initiation to completion is revisited during the recovery process. The process is shown in Figure 3.6. The Guide replay engine reads the first line of the design history record and uses it to create the first state of product model data. Thereafter, the latest state of the product model data is used together with the next instruction to create the next state. Every product model state from project start is re-created in turn.

If the project comes into unexpected difficulties, it is possible to roll back the design to a previously known safe point and resume the design process down a different path. This provides a multi-level 'undo' facility for design. Furthermore, it is possible to navigate the design information space in any arbitrary direction, at any points in time and levels of abstraction [1.8]. The evolution of an entity or relationship can be examined against variables such as time, design context or departmental involvement.

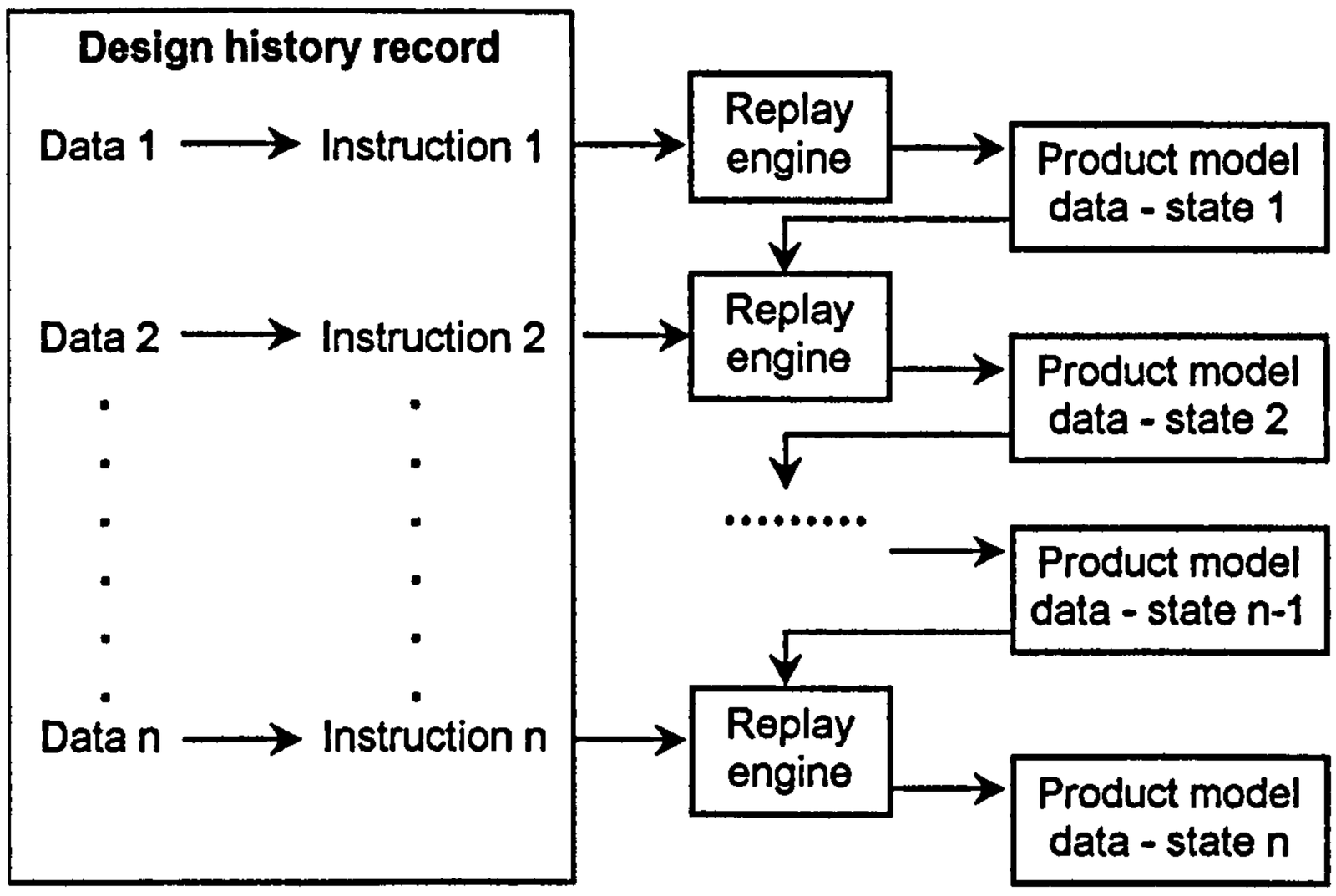


Figure 3.6 - Replay of the design history record

3.2.1.2 Evaluation of alternative solutions

At a given point in the design project, several alternative solutions to a particular problem may have to be evaluated. Some work needs to be done on each of the alternatives; the best solution is not obvious. From the base point, it is possible to pursue one design alternative, then return to the base point and develop the alternative solutions, creating an options tree. Once enough work has been done, a choice can be made of the most promising path to pursue. If later in the design this proves to be an incorrect judgement, another branch of the design, one evaluated but not initially chosen, can be re-instated and further developed. It is therefore useful to keep traces of unsolved problems, problem solution proposals and their evaluation, even those initially discarded [3.14].

At a lower level, some CAD systems provide this facility, where changes made to a solid create new branches where necessary. (Figure 3.7). All of the branches defined at any time during the design process are then available and the designer may return to a previous state.

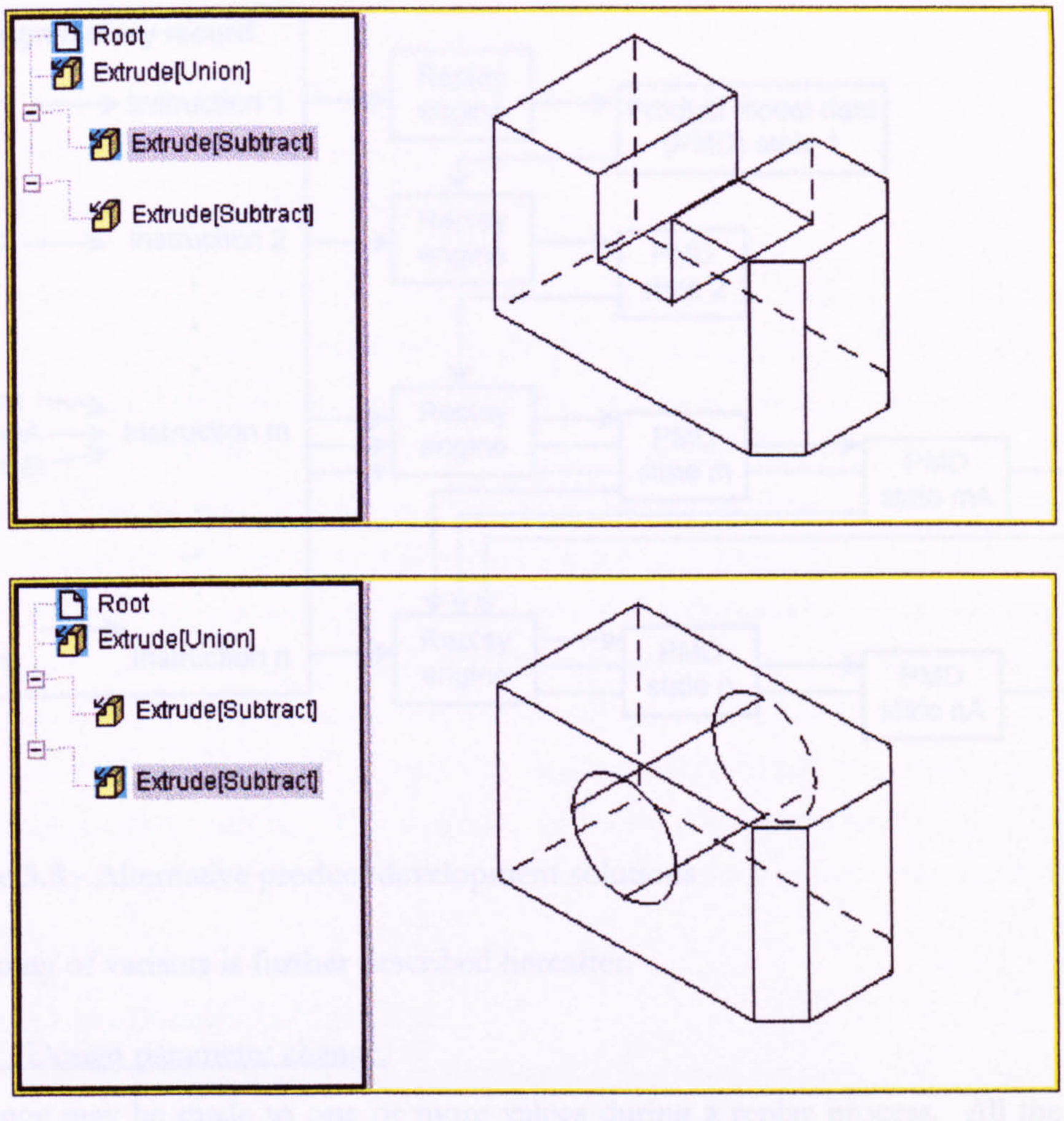


Figure 3.7 - Alternative CAD model states

3.2.2 Variants of existing design solutions

A replay of the design history record may be used to determine the effect on the design solution that changes in the design workspace have and to generate modifications to existing products. Visser [3.14] describes it thus (The observation referred to is of two designers wanting to re-use a design solution. Although they had used it in the past, they could not remember the rationale or justification for the critical dimensions):

> *This type of observation shows the usefulness – if not the necessity of keeping, in a knowledge base of reusable solution elements, not only the traces of old "final" solutions, but also their underlying justifications.*

The process is shown in Figure 3.8, where alternative data fed into the process stream generate separate solutions, different to the original. All the original constraints are fired and the same checks on the original design are re-applied, unless the user intervenes at some point in the replay to modify the constraint set.

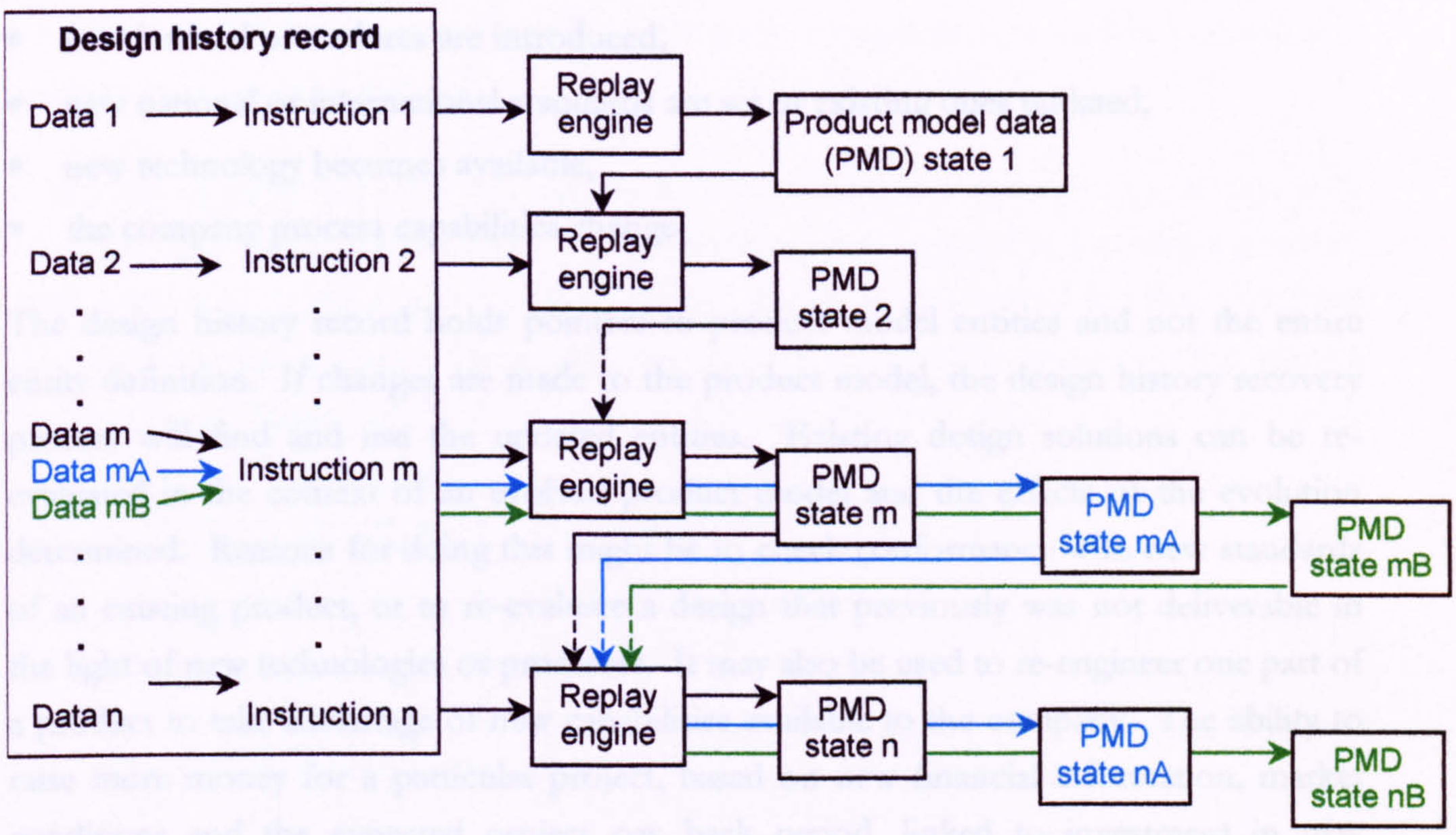


Figure 3.8 - Alternative product development solutions

The utility of variants is further described hereafter.

3.2.2.1 Design parameter change.

A change may be made to one or more values during a replay process. All the design actions from the change onwards can then be re-evaluated to yield a new product derived from the original. This process may result in some constraints failing when fired with the new values. In this case, the user can intervene either to take over the design process from that point, or to implement a local solution to resolve the problem and continue with the replay. For applications such as process plant or modifications to in-service aircraft, where design modifications are made on existing equipment, rather than on production of new units from that date onwards, the ability to determine the impact on the overall design is crucial. Guide provides the engineer with the set of assumptions on which the design was made. The repercussions of design changes may not be immediately apparent, but affect other related systems. [1.20]

Bañares-Alcántara [3.15] also talks about the utility of being able to store design states, so that they can be returned to and the necessity of being able to propagate changes downstream to all nodes from the previous state to all the branches dependent on it.

3.2.2.2 Re-evaluation of constraints.

Over the course of time, the product model employed by the company will change. The causes of change may be that:

- new internal procedures are introduced,
- new national or international standards are set or existing ones updated,
- new technology becomes available,
- the company process capabilities change.

The design history record holds pointers to product model entities and not the entire entity definition. If changes are made to the product model, the design history recovery process will find and use the updated entities. Existing design solutions can be re-evaluated in the context of an evolved product model and the effects of the evolution determined. Reasons for doing this might be to check conformance with new standards of an existing product, or to re-evaluate a design that previously was not deliverable in the light of new technologies or processes. It may also be used to re-engineer one part of a product to take advantage of new capabilities available to the company. The ability to raise more money for a particular project, based on new financial information, market conditions and the expected project pay back period, linked to investment in new technology for the company is an example. If funding had previously blocked the start-up of the project, this review might provide the necessary information to enable it to proceed.

3.2.2.3 Design solution optimisation.

Once a general design solution has been found, it may be necessary to optimise it against a set of critical criteria. One approach is to build a model that simulates the system and determines the reaction of the system to different inputs. The construction of the model is time-consuming, and any lessons learnt need then to be applied to the design problem. By using the product model data directly, the need to create a separate simulation model is removed and once an acceptable solution is found, there is no need to define it to the design engine; the information is already stored in the product model data.

By making small changes to the inputs, it is possible to determine whether the solution is stable (there is a correspondingly small change in the output) or unstable, where the output varies considerably for a small input change. This is useful in determining whether safety-critical components and systems are operating at levels close to their performance limits.

3.2.2.4 Standard parts

Standard parts will often be used during the course of a design, which may be simple mechanical components such as seals, fasteners or bearings. More complicated parts such as computer hard disk drives or graphics processor chips can also be considered as standard parts by a computer manufacture and assembly operation. Standard parts can

extend beyond the purely mechanical; in designing a bus service, a vehicle maintenance contract may be a standard part.

A product designed in Guide will have its product model data and a design history, both of which refer to a product model. When a product is referenced as a standard product, a link is created to the product model data and design history record for that part from the current design project. These then appear to the user to be integral to the current design project (Figure 3.9).

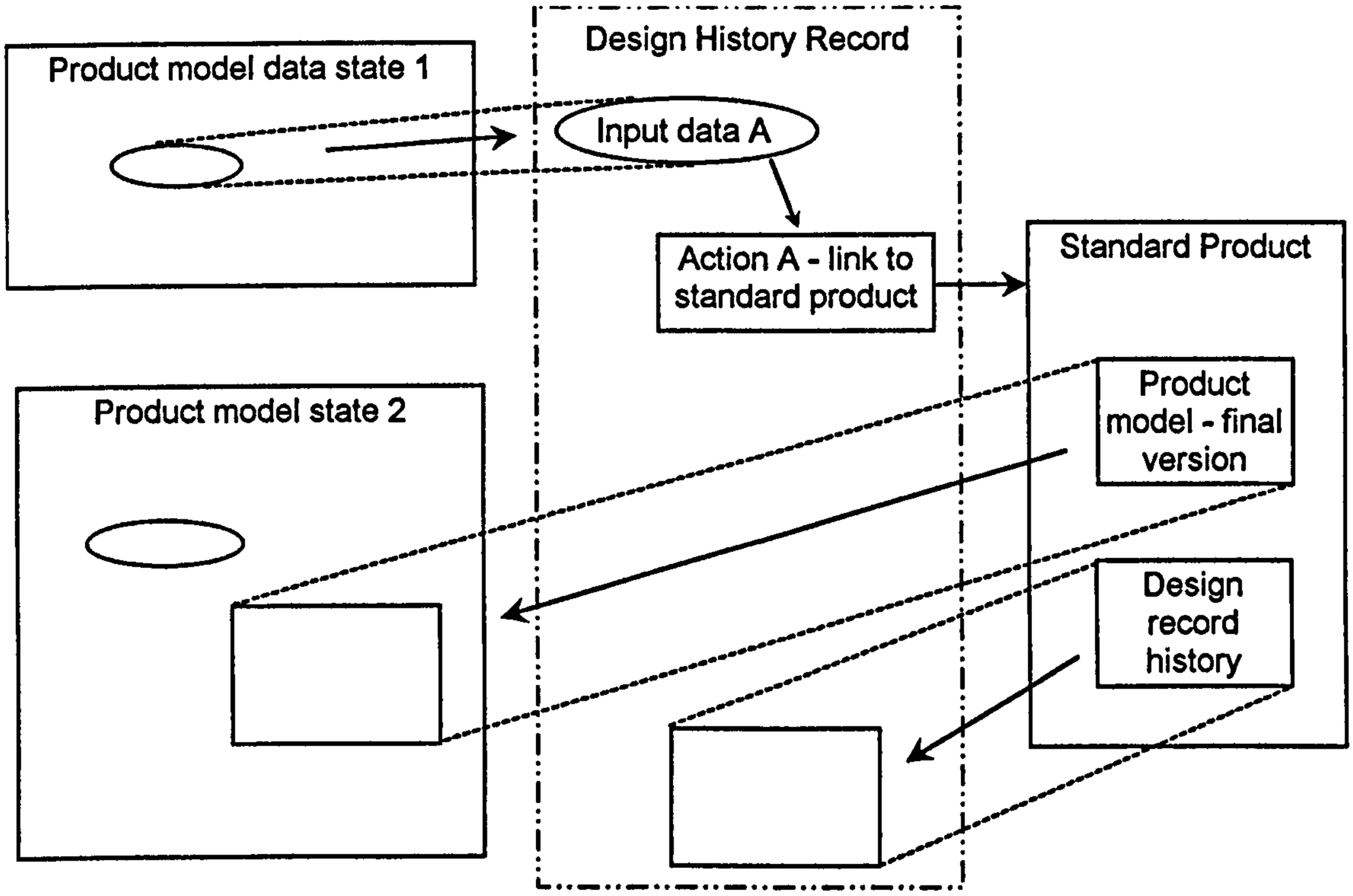


Figure 3.9 - Use of standard parts in design

Cleetus [1.7] through his UPDM recognises the value of this ability, as whole sub-assemblies can be taken from one product directly into another without loss of semantics or ability to interface to the rest of the design. In the example he gives, the car industry uses only 10-15% of new components from one model to the next. In this case, the utility of such a system is obvious. From the point of view of a contractor, it provides a good mechanism for the communication of technical information relating to their products. A subset of the product model data can be published to allow customers to examine the technical specification of the products offered. This is an elegant way of satisfying the requirements under the US Contractor Integrated technical information Services (CITIS) programme to have such information available in electronic format and available through a single entry point.

In the example of a computer graphics processor chip, important mechanical, thermal, electrical and functional properties affect the use of the component. If the product model data for the chip are available, then links to the standard part can be created as if it had been designed as part of the current project. Otherwise, the specification data for the part would have to be used and input to the system manually to ensure the design is compatible and will accommodate the part.

The links between the standard part and those assemblies that use it are bi-directional. Should the part change, therefore, all assemblies of which it is a part can be identified, and the impact of the change assessed. The same situation applied if a part becomes obsolete and is supplanted by a newer one.

If the design history record is also available for the part then it can, if desired, be edited to produce a new part for the company. While it is improbable that a vendor would reveal their design techniques in this way, parts currently produced by the company also count as standard parts, and can be used in more than one product. If it is necessary to perform a modification to an internal standard part, the new part can be retro-trialled against its existing applications. This provides the company the opportunity of evolving the part, rather than creating a variant and consequently having two parts to maintain and produce.

3.2.2.5 Configuration and part management

A number of problems face the company that strives for economy in the management of different configurations of one product, which are described below.

- Ensuring that the correct collection of parts comes together for the intended destination and recipient. For example, in configuring a PC for a particular country, the correct voltage of power supply, mains plug type, modem telephone cable and connector, warranty forms and manuals must be packed. To complicate the process, the items come from different areas of the company: production, insurance and documentation, say. Some of the parts such as power leads may be bought in from outside on a 'Just in Time' basis, and so are linked to purchasing procedures in which there is little flexibility.
- The explicit storage of relationships within the product model data ensures that the correct items can be found from any place in the product model data. The emphasis is again on the product model data storing the relationships, rather than configuration management software holding them independently.
- Changes are made to one part that is also used in other assemblies. The links from and to a part are held explicitly, therefore every affected assembly can be identified

and the engineers responsible for them notified. This removes the possibility of problems emerging later. Several options are available to manage the change:

- Create a version of the part and link it to the current assembly. No other product is affected, but an extra version of the part needs to be managed.
- Propagate the change to all assemblies.
- The part is not used in any other active product and can therefore be changed without causing any data integrity problems.

- Avoidance of re-invention of the same part. This problem affects large multinational companies employing distributed manufacturing bases. Powerful searches can be performed through the product model on any entity parameter. Furthermore, the product model data can be searched for occurrences of a particular entity instance or pattern of attributes. This contrasts with traditional methods that allow for searches on a descriptive field only.

3.2.2.6 Data exchange

Communication of product model data does not always occur within a homogenous computing resource. The native formats for the different agents involved in the design process and invoked by Guide, or used by external agents, will differ from that defined within Guide.

The completeness of the information supported by Guide - the product model, product model data and design history record - expand the amount of information that can be communicated when compared with function-specific software. At an elementary level, any piece of information, relationship or set of related information can be extracted. Beyond that, any part of the information contained by Guide including methods and constraints can be communicated. For the communication to be interpretable by the receiving party, Guide can pass the product model through a translation filter to a neutral format externally. For instance, the product model could be translated to an Idefx model, product model data to STEP/PDES data and accounting information to spreadsheet files. Communication between two Guide nodes will, as with any computing system, be the easiest to achieve and no conversion will be necessary.

Brissaud and Garro [1.22] found that communication between different functions (or modules as they called them) happens throughout the development cycle; there is no time during which they do not occur. Admittedly, the project they considered was a small one, and so problems of communication were incidental. What they lacked, and realised they lacked, was a common basis for all the functions, they described the problem as:

"The difference of language employed by each module, where data and treatments (tools used) are specifically job-oriented"

Either the whole or part of the system can be communicated. Typically, data transmission was of CAD model data or orders and invoices. With Guide methodologies, the sphere of communicable information is expanded to any business-related information. Sub-parts of the information can be extracted and filtered for distribution. For example, an agreement with a sub-contractor may involve the transmission of input data, procedures to be followed for the design or assembly work and measures by which progress is to be monitored.

Use of a product model means that the full power of PDES is available to be used. Indeed, if the user chose to do so, the Product model entities could be modelled on the basis of PDES structures, which have been shown to map well onto object-oriented constructs [3.16]. They are not restricted to PDES structures, since any standard is slow to evolve and is unlikely to contain definitions for all the entities used by a company.

Information about a project will be transmitted between parties involved in the design of a particular product at different levels, from organisation structure information to details of a particular component. This helps to address the current problem of reconciling the information of different types that is transmitted between two organisations at different levels. It happens that information about a specific issue is passed between a company and its sub-contractor through management or sales channels. At the same time, technical information about the same project, which is affected by the other information can pass between technical departments. This leads to the potential for confusion and ill-informed decisions because of fragmented information. The information does not always reach the intended recipient in a timely and accurate fashion. Because Guide permits the communication of related information at all levels to the sub-contractor, the recipients can make informed decisions based on the overall requirements of the customer, including those that have an indirect effect on their work.

3.2.3 Design audit

3.2.3.1 Measures against status of current design

Feedback is essential to any project administration situation. Project planning software may be used to map out the various stages and responsibilities of the project, but the progress against the timetable needs to be fed back into the system for the loop to be complete and the planning activity to have meaning. It is only with feedback that informed decisions about amendments to the plan or resource allocation can be made. The quality and accuracy of the information will affect the impact that planning has on the smooth and effective roll-out of the programme. Guide provides the facility for the

management of the project rather than managing through a model of the project. As well as providing the mechanisms to include project planning as an integral part of the project, the Guide architecture supports several metrics describing the design project's progress. The measurable indicators to supplement any reporting mechanisms are as follow.

- The rate of activity initiation. The outline of a design project and the framework for pursuing the design will be established during the early phases of the project. As part of this process, many activities will be initiated. If the rate of activity generation is slow, the design is likely to be in its later stages, during which the design problems are resolved into solutions.
- The proportion of activities completed. This is an indication of the state of the project. Since incomplete children activities prevent parents from achieving completion, this is a good guide of the project state.
- Expansion or contraction of the number of activities. This is a corollary of the previous two indicators; when the design is in its completion stages, activity completions will outnumber activity initiations.
- Identification of activity trees with a high proportion of unfinished activities, which may hold the project up.
- Measurement of time taken to complete activities can highlight bottlenecks and slow departments or long processes.
- Determination of the number of links between activities either on the same or different branches of the activity tree. This provides an indication of the level of complexity of the project and the amount of coupling between the different areas of work.

These quantities can all be measured directly from the product model data. Additionally, the design history record may be consulted. It can be used to track resources, control costs and monitor staff time allocation. Indeed, any quantity that is known and expected to vary throughout the duration of the project may be monitored; forecasts may also exist for these quantities against which progress can be assessed.

3.2.3.2 Verification of adherence to standards

Should a problem arise with a product, the position of the company confronting litigation is considerably strengthened if it can demonstrate that reasonable care and attention was paid during the design process, including recognition of and adherence to mandatory standards applicable to the product. In this respect, the design history record complements the design process audit functions described in section 2.3.3.3.

3.2.3.3 Evaluation of performance of design processes.

One application is the verification of constraints [3.17]. The execution and results of constraints may be examined over a period of time. With design records available, they may also be examined over several projects, not just the current one. Several scenarios may then be identified from the examination:

- Constraint never fails. The constraint may be redundant, in which case it can be removed, decreasing the processing overhead. It may also be that the success criteria are too lax, in which case they may be tightened.
- Constraint fails by a small amount. If appropriate, the constraint band can be widened
- Constraint always fails. The underlying reasons for this failure can be examined. If a constraint always fails, further work must be done to circumvent the problem. By identifying the constraint, effort can be directed at avoiding future failures.
- Number of times a constraint is executed. If a constraint is consistently not applied, no record is made of the interaction.

The theory of constraint absorption is used by Meehan and Brown [3.17]. Their application was to reduce the brittleness of expert systems and evaluate not only whether rules were satisfied or not, but establish a measure of degree. For examples, it would be possible to identify a constraint that is consistently failing by a small amount. Certain constraints are not necessary, given that in the environment they are exercised, they will never fail. They propose the use of a design history to assess a constraint's application and outcome over a period of several designs.

3.2.3.4 Record of contractor actions.

If the contractor employs a similar design history recording system to the company, similar benefits to those achieved for the parent company are possible. In addition, there exists a traceable record of activity for the product as a whole, even if some of it is invisible to the company, being the proprietary information of the contractors. Nevertheless, the links exist and can be followed where changes need to be propagated, or knowledge about the design gathered.

3.2.4 Extension of design capacity

Companies face a problem in how to keep, maintain and capitalise on the design experience and expertise it owns. A company leases much of its knowledge from its employees whom together and between them own a significant part of the corpus of company expertise. To survive in the long term, the company relies on recruiting people who have knowledge and experience to offer, or who can learn from current employees

and take over from them in due course. As well as the problem of employees taking valuable knowledge with them when they leave, only they have access to it before hand.

If knowledge is to be extracted from the design process, then some sort of record is necessary as the basis from which to derive the generic rule. The record is commonly the memory of the designer, learning from previous designs and approaches they have tried as alternatives to the established methods. This is the designer doing what human beings are very good at, abstracting rules from their experience and data. There are limits to the sorts of problems to which this approach can be applied, however, apart from the human limitations such as loss of memory, lack of time or variable lateral thinking capabilities of the different people. If the rules affect larger pieces of work, the one designer will never know or see the full picture and therefore not be able to abstract the rule(s). If several people are going to collectively address the issue, then they need a common frame of reference on which to base their work.

Enhancing the ability to extract new rules from the design history has several benefits. Research can take a long time to complete and is expensive. Furthermore, it can take a long time for new findings to be translated into usable design rules. Any learning and transfer of formalised knowledge from higher to lower levels that can be enabled will therefore give benefits in terms of:

- low costs (the work is being done anyway),
- immediacy of applicability, because the knowledge is being distilled from an existing process.

This is not to suggest that this sort of knowledge gathering will ever replace structured research, since it is an experiential learning tool, rather than an investigative one.

One way in which expertise is captured is to build knowledge bases and exercise them thereafter in the design process. In order to do this, specialists in the field concerned need to be engaged in the task of defining the rules to populate these knowledge bases. They do this based on their experience, in effect a distillation of previous designs and analyses they have performed. Knowledge bases such as these require considerable amounts of resource to set up. They also inherit the problems endemic to knowledge bases, such as the need for completeness in the rule set and the narrow domain of problems to which they can be applied. They also form yet another source of fragmentation within the product information model, as they do not contain links to related information.

Any system that helps the company retain the expertise of its workforce and make it available internally is to be welcomed. Expert systems seek to take on this task and are

constructed by extracting the knowledge and rules applied by individuals and formalising it. Su, Tseng and Mayer [1.10] state that:

> *The key to successful CE applications is the capture of life cycle knowledge and timely effective delivery of that knowledge to the appropriate decision-making event.*

This theme is echoed in work by Zhu [3.18], who expands on the necessity for design support tools (with a focus on CAD systems) to learn and refine their support of the designer as he or she carries out design functions. The requirements identified are: to enhance accuracy of design attributes, to guide the user past the first rough estimate of entity attribute values and support the generation of concepts from specifics and capture variations in design environments, such as expand the capabilities of the computing tools for new entity representations.

3.2.4.1 Best design practice extraction

The presence of the design history record offers the opportunity of reviewing previous designs to identify and extract either good engineering practice or frequently used design solutions. The record may also be analysed to support the analysis of some novel implementation or solution to a problem and better understand invention. These can then be formalised and incorporated as methods and constraints into the company's Guide installation. New entities may also be defined to better support the function.

Another option is to use the solution as a standard parametric part. In this approach, the standard part is imported into the current design project. The controlling parameters can then be changed from their default values to those required in the project, possibly through referring to other entity instance attributes in the project.

The audit mechanisms provide an evaluation of the way in which design rules are interpreted and deployed within the design process – well and innovatively or poorly. These findings are useful in shaping new design procedures.

3.2.4.2 Design template construction

Where a complete design task has been established, a design support utility can be implemented through Guide. Specifically, a set of activities can be constructed such that creating an instance of the first spawns, through constraints, the creation of the entire activity set. Each activity can have post-instantiation constraints to create instances of entities within itself. The result is a complete set of activities and entity instances to describe the process in question. This could be a part or a service. For example, in the creation of a new design it could be used to ensure that a design checklist was followed, eliciting requisite information from the customer and ensuring that a complete and

accurate specification for the problem was developed. Methods can further be used to guide the user in the provision of values for the instance attributes.

This is of particular utility in established industries, where most of the technology is known, and most design problems involve variations that can be expressed parametrically.

Sugumaran and Bose [3.19] construct process models. These are collections of objects and methods destined for one purpose. They have to be built from the ground up, however, and are then tailored to one specific task. The objective of their work is to remove the burden of tedious tasks from designers. They describe their processes as domain-independent. What this means is that they can construct their process management systems equally to request travel authorisation as design a gear. It does not mean that a process can be taken and applied in a different domain.

Reed and Struges [3.20] describe a mapping of certain engineering functions onto a collection of artefacts. The function chosen for the job will be predicated by the conditions under which the function is being exercised. This is an extension of group technology, where the mapping is not onto a previously created part, but onto a parametric template for the part. This resembles the Guide design template inasmuch as it relates to a part definition, rather than an existing part. Where it differs is that it does not setup a framework for design and creativity; rather, it provides pre-existing solutions in an efficient way.

Smith and Ball [3.21] set out the requirements for a design framework of the type described here as the ability to represent:

- the design phases from market investigation to product sales,
- the preferred route through each phase in terms of activity sequence but also allow the facility for over-ride and iteration to accommodate project peculiarities,
- guidance between the phases, again with the facility to over-ride and iterate.

These are all met in the activity model.

3.3 Review

Guide operates through activities to manage projects. The activities contain instances of entities and their relationships that together describe the artefact in development.

Once an instance is created, it must be characterised by the provision of attribute values congruent with the aims of the design and desired function of the artefact. Methods are provided which assist the engineer in this function. Methods are also used as constraints to safeguard the integrity of the emerging design. Constraints provide a large amount of control and are able to be applied at exactly the desired level.

A design history record is captured and maintained. This supports functions of:

- options evaluation,
- review of previous solutions,
- creation of product variants,
- efficient use of standard parts,
- configuration management,
- extension of the company's knowledge through analysis of the record,
- audit of the design and the design process.

The approach taken by Guide is to provide a rich and powerful framework for the representation and utilisation of the product model. Commercial software addresses the market with two strategies:

1. Large, powerful software suites such as CATIA, Pro-Engineer or CADDS5. These have developed a range of modules to address specific modelling or process requirements beyond geometry representation, such as robotics, finite element analysis, piping and duct layout, or NC machining. The software suites achieve a high degree of integration among the applications, in isolation from the rest of the company's product information. The storage medium is the model file, which is informationally barren and the internal format of which is generally jealously guarded to prevent access to the knowledge contained therein.

2. Product Data Modelling (PDM) and Product Development Management (PDMII) tools such as Enovia and Metaphase which allow the management of files by maintaining a set of metadata. They still rely on the adoption of rigid product description models and methodologies imposed by the application software. Communication is not enabled through any novel mechanism and still relies on bespoke translators.

4. Guide architecture

4.1 Architectural overview

Guide is built on industry-standard tools for representation and interaction, but is not dependent upon them; particularly, the product model structure is invariant. This is demonstrated by the two versions of Guide, implemented through different technologies but able to operate on the same product model. The first uses the services of the Dassault Systèmes CATIA CAD and engineering software package for geometry representation and for user interface interactions. The second version, developed to support the Rolls Royce Ceramics Design System is an X-windows and Motif implementation. In this version, the geometry visualisation is through ComputerVision Cadds5.

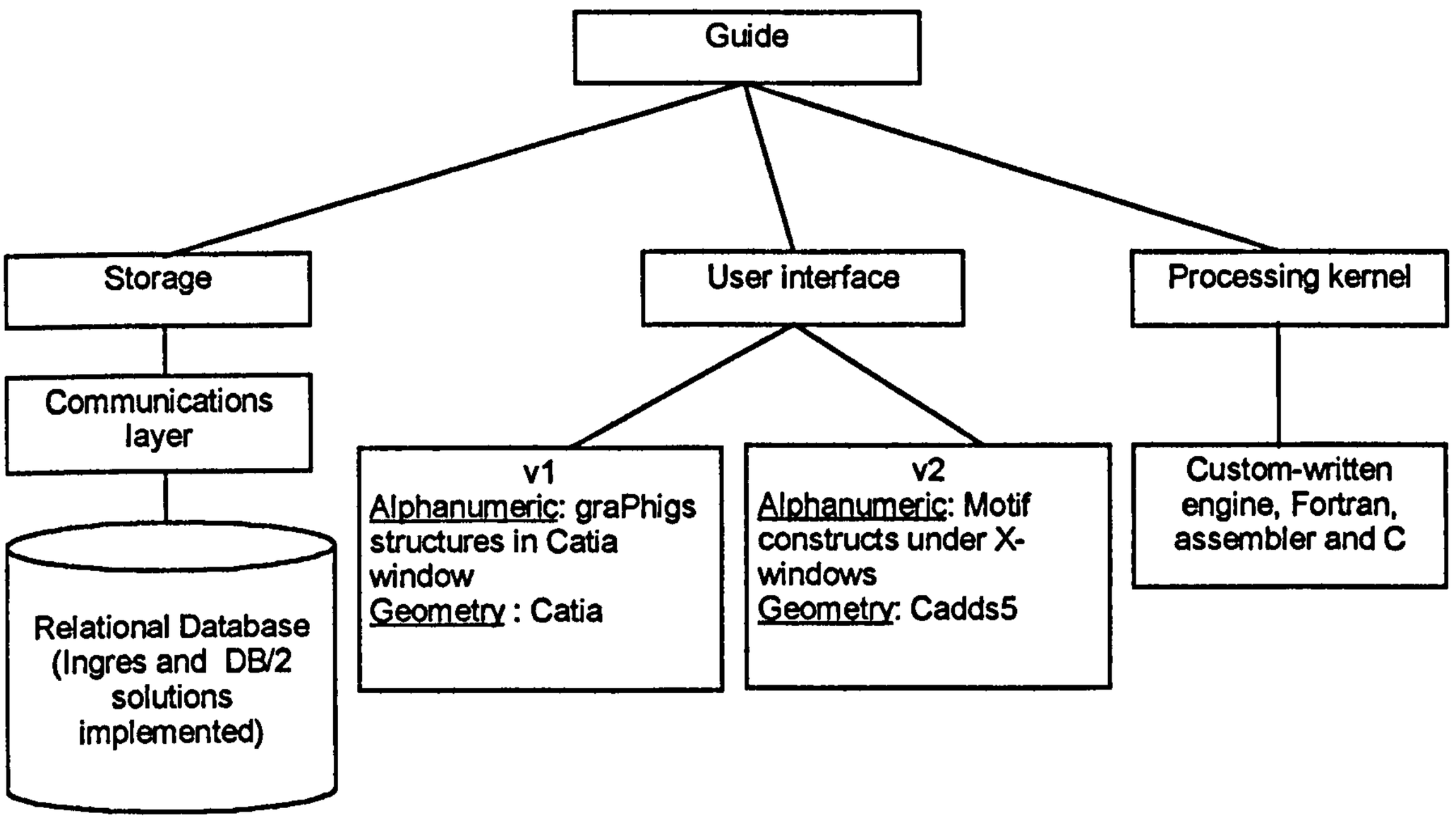


Figure 4.1 - Guide contributory software utilities

Relational database technology is well established and widely used and respected in business. The technology is powerful enough to support the representation constructs without the prior imposition of a specific object-orientation paradigm. Guide entity descriptions resemble objects, but the concepts of method ownership are different to those of conventional object orientation. An SQL interface is used, allowing the use of any relational database, with changes to the interface limited to the data buffer management peculiarities of different database products.

This structure is more flexible than conventional object-orientation, in which the method must be written during the definition of the object and where the potential for code re-use is low [3.10].

Law et al. [4.1] also use the relational model as the storage element of an object-oriented application to avoid the early binding characteristic of object-oriented storage schemes. And work by Ehrlenspiel and Schaal [4.2] also adopt RDBMS technology to store frame representations, although in their case they were limited to product geometric features and associated manufacturing operations.

While the kernel activities and user interface interactions are carried out on the local machine, the product model definitions and product model data may be held on several distributed machines accessible by the whole enterprise. The communications layer allows for access to the storage location. Several Guide servers may be set up in the company, to ensure system stability through redundancy or to manage different, unconnected business units. Part of the Guide logging on procedure is to select a server (Plate 4) and design history record repository (Plate 5) from those available. The chosen server then handles all Guide interactions for that session. The interactions include reading methods and product model entity definitions from the repository and the storage of product model data.

4.2 Entity representation

One of the major challenges in the construction of Guide is described by Richter [4.3] in the following terms:

> *Technical information, in general weakly formalized, must be changed into information structures of computer science which has to be strongly formalized.*

The basic Guide entity is the ***structure***, a frame-based construct (Figure 4.2). A header block describes each structure and provides a basic description of the structure contents and the entity it represents. This concept is also adopted by Bello et al. [4.4]. The structure characteristics come from its associated ***atoms*** and ***child structures*** that are linked to the structure in any number and combination. An atom holds a value or a pointer; child structures are ordinary structures used to impart complex attributes to the parent structure while maintaining clarity of representation. A structure does not own its atoms and child structures; they are associated with the structure through a relationship held by the system. This leads to an economy of representation, as it is possible to link atoms to several different structures. For example, a material description atom can be linked to several structures such as part, fastener and bearing; several department structures can use the same staff resource list.

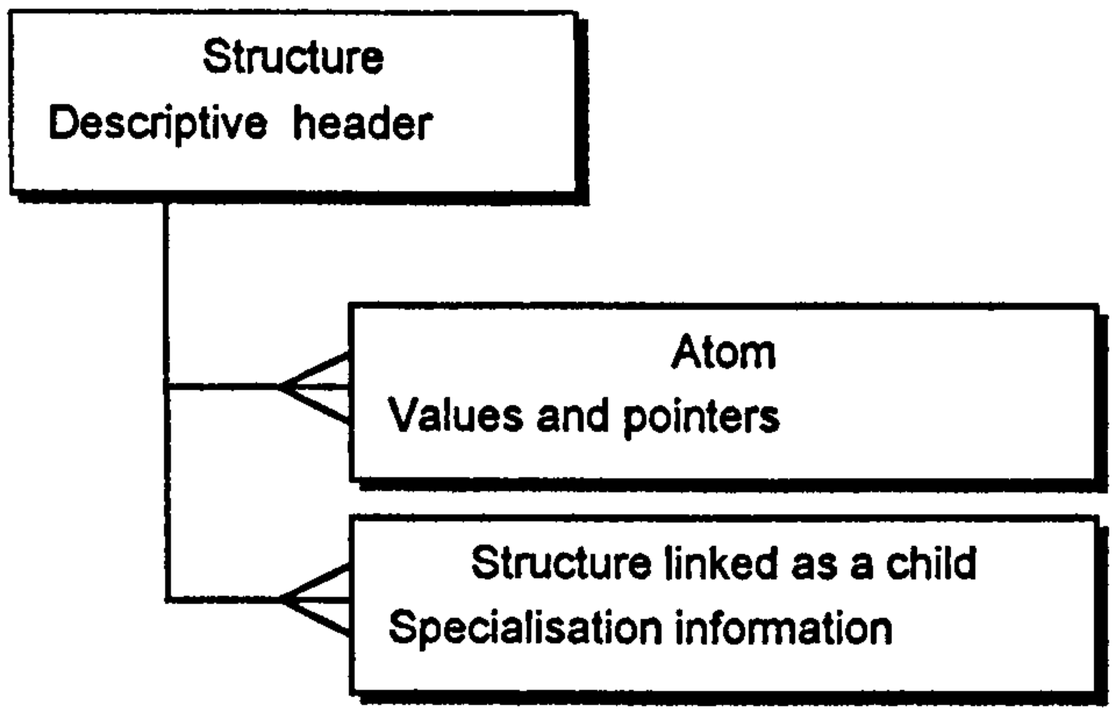


Figure 4.2 – Structure components

4.2.1.1 Child structures

The use of child structures offers several benefits over an atoms-only schema.

- Complex and detailed information can be included in the structure description without obscuring the essential structure content.
- Several structures can refer to the same set of information by storing a single link. It would be possible for pointer atoms to refer to the same information set, but in this case, a link for each value would be required.
- It allows for a degree of abstraction in the logical schema; the child structure can change internally in certain ways without affecting the parent structure.

Child structures can be created during the structure edit process (Plate 6), or a link to an existing structure can be established. This is the reason that the child structure is not prepared at the time the parent is prepared and so receives a status of "Not prepared". Otherwise, to link to an existing structure would involve replacing the prepared, blank structure that would then be redundant in the system. The sequence of operations for creating or linking a child structure is shown in Figure 4.3.

4.2.2 Derived entities

Structures and atoms may evolve over time, through the generation of new versions or variants derived from the originals. The descendant entities can replace the original, or they can be derived and complement the original structure or atom. The creation of a version involves first making changes to the structure or atom then making the old version inactive and activating the new version.

The outgoing version of the entity is not deleted but is retained in the system. This is necessary because the product model data relies on the product model to give it meaning. Earlier versions of structures and entities must be kept to ensure that all previous projects can be correctly interpreted. Storage of the structure and atom versions in the

instance data ensures that the correct product model definition is used in subsequent operations.

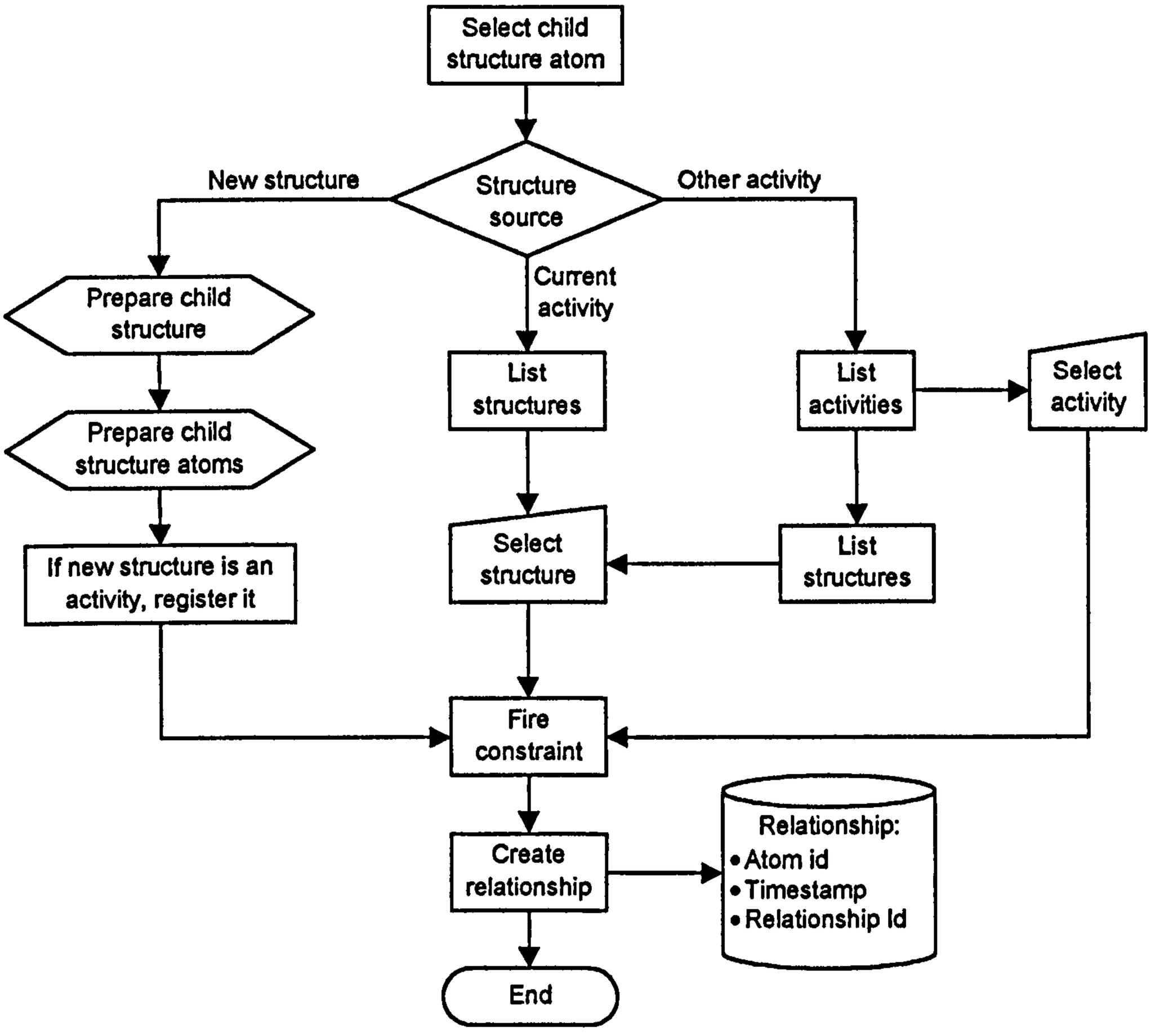


Figure 4.3 - Child structure association

It is also possible to reactivate an older version of a structure or atom. Care must be taken when doing this, as some of the ***element***s originally linked to it may themselves have been updated. The choice then needs to be made upon whether to reactivate those linked elements also, such as old versions of constraints, or whether to re-define them. In practice, a safer solution is to create a new variant of the latest version of the structure to match the original description. In this way, the linked elements are all known to be up to date.

A derived structure or atom is a subtype of its parent and carries with it all of the properties of its parent, plus the extra characteristics which differentiate it; Figure 4.4 shows an example. The top half of the figure shows a parent structure that represents direction. Underneath is a derivative of direction, line. The derived structure inherits the attributes (atoms, child structures and constraints) from its parent. It adds an extra atom,

length, together with a constraint on its value. The atom Vector has also been further constrained to Unit vector.

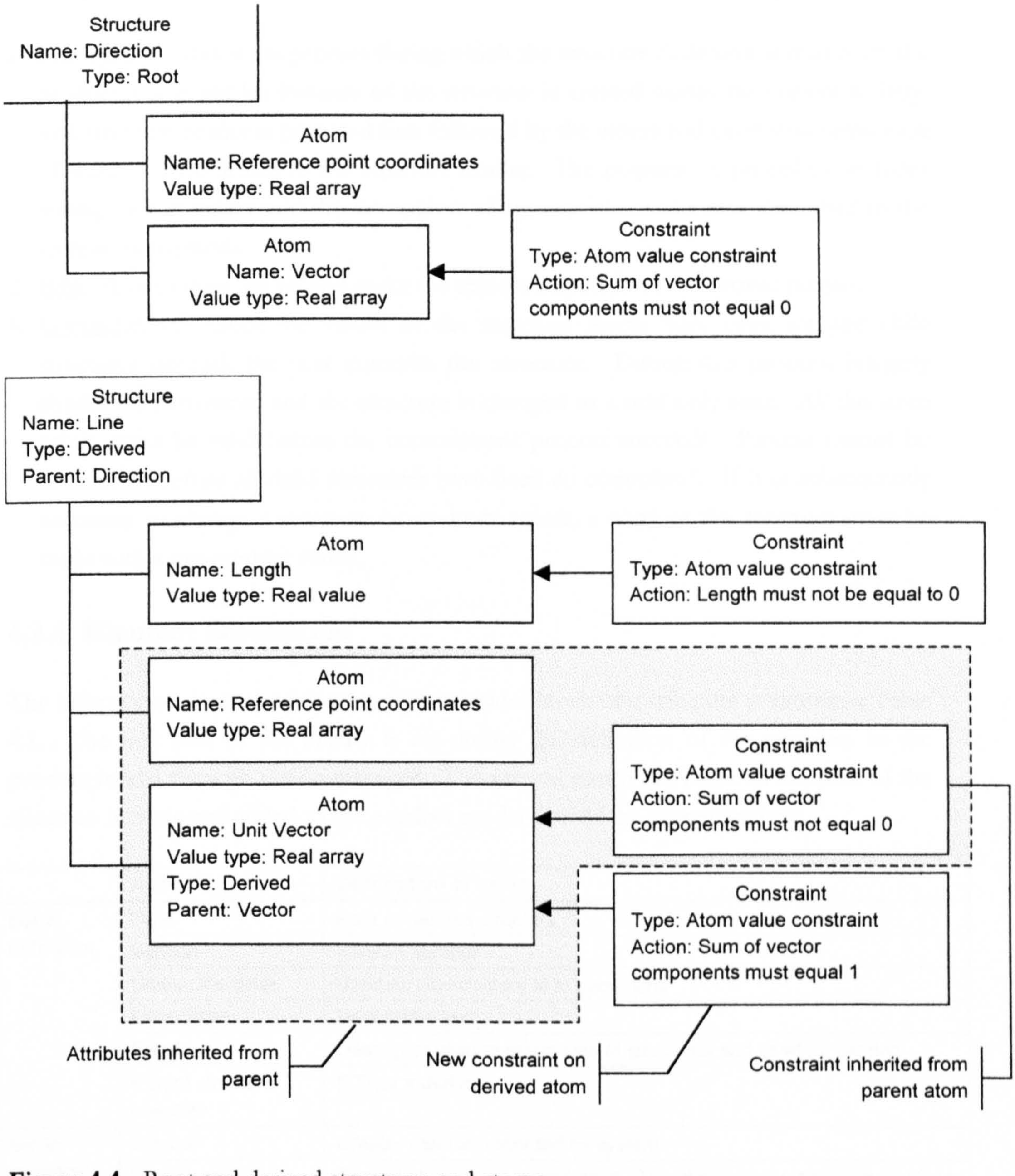


Figure 4.4 - Root and derived structures and atoms

To define a derived entity, only the extra attributes and a new header need be stored. These are the items outside the shaded box in the derived structure and atom example of Figure 4.4. The derived entity retrieves its other attributes from its parent. This approach offers a concise representation, especially in storing derivatives of complex structures. It is possible to nest the derivation of structures and atoms to any depth.

4.2.3 Structure use cycle

The sequence of operations for the use of a structure involves the following steps:

1. Preparation. This is the process during which the structure definition is read from the product model and an instance of the structure is created within the current activity. The structure header is prepared first followed by the atoms and child structures, each of which is then linked to the structure header. The preparation procedure includes setting default values and the invocation of appropriate constraints, described in the section on methods.
2. Edit. Atom values are set and make the structure specific to the current project.
3. Commitment. Once the values of the structure atoms have been set and child structures defined, the user commits the structure. During this process, integrity checks are performed and the structure is changed to a read-only state. All the atom values must be valid before the commitment process succeeds. Parents cannot be committed before all child structures have been so committed. If it is subsequently necessary to change a structure or its atom values, a copy of the structure must be made with a pre-commit status.

4.2.4 Element description

The information content of the descriptive header block of a structure is shown in Table 4.1. The first part of the header is set during the definition of the structure to the product model through Guide manager. The second part is set when an instance of the structure is made and added to the product model data of the current activity.

	Attribute	Description or value
Set in definition:	Type	Root or derived structure
	Version	Version number
	Unique identifier	Used as the structure id in operations
	Description	Descriptive text
	Family	Descriptor used to group sets of structures and used for filtering
	Parent structure identifier	If Type = derived
Set in instance:	Creator	Creator used id, recorded by system
	Timestamp	Record of structure instantiation. Together with the structure id, forms a unique identifier for the structure instance
	Status	Incomplete: atoms not stored Complete: Atoms stored, structure validated.

Table 4.1 - Structure description header contents

An atom is described in a similar fashion to a structure, with definition and instance parts to its descriptive header, as shown in Table 4.2.

	Attribute	Description or value
Set in definition:	Type	Root or derived atom
	Version	Version number
	Value type	Data type. Options are:- Values: numeric or textual, single values, matrixes or arrays Spatial elements: vector, plane Pointers: structure, geometry, relationship
	Value sub-type	Used to refine the value type field. For example, a type of geometric element can be specified through the sub-type.
	Value length	Space allocated for value
	Value source	System, user, method output, calculated
	Number of values	Number of values in array; can be unlimited
	Value units	Reference to units table. Allow for dynamic conversions and equivalencies
	Unique identifier	
	Description	Descriptive text
	Parent atom identifier	if atom type = derived
	Default value	Provides a simple value to be given to an atom at instantiation. Defaults may also be provided through default value provision methods at the time of instantiation.
	Variable name	Used in user-defined methods.
	Dispose status	Store or broadcast. Broadcast values are used for information only.
	Mapping	Storage location for atom values
	Mapping method	Storage method for the atom value
Set in definition:	Change indicator	Determines what action to take on a request for change because of a change cascade. Options are: • change, • change and notify, • warn and wait for confirmation, • refuse and warn.

Table continued on next page

Table continued from previous page

	Attribute	Description or value
Set in instance	Timestamp	Same as structure instance timestamp
	Status	Changes to reflect the stage of the instance creation process. Value can be: • Default taken • Valid value with dependants • Valid isolated value, no dependants • Not valid (constraint on value failed) • Child not valid (constraint failed on a dependent) • Derived from a parent

Table 4.2 - Atom description header contents

4.2.5 Activities

The activity is a structure, distinguished by a family attribute set to "Activity" and like a structure it can have any number of atoms and child structures. As a minimum, an Activity must contain the set of atoms shown in Figure 4.5 and (Plate 2), viz.:

- Requester. This is set to the Guide user's unique identifier by default and cannot be altered.
- Responsible. This is blank by default. The status is set to "Invalid" to ensure that the activity cannot be committed unless a responsible party has been allocated to it.
- A reason is required for the activity. This could be a simple description of the task required in the activity, or a document. It is also possible to associate a method with this atom to elicit specific information from the activity requester.

Guide uses the requester and responsible atoms to control the flow and assignment of activities. When an activity starts, authority over design data passes from the requester to the responsible agent. The latter could be the same person, someone within the same company or an external contractor. A user will have a list of activities for which he is responsible (Plate 7). For example, an engineer starting an activity for the manufacture of a part, with a sub-contractor as the responsible party. The sub-contractor opens the activity and examines it, then initiates two new activities, one to the engineer requesting clarification on a detail, the other to the accounts department of the company requesting payment of half of the activity cost.

Since activities are special types of structures, they can be further characterised by the addition of extra atoms and child structures (Figure 4.6). This provides two mechanisms for linking activities: by pointer atoms that reference an activity, or child structures that

are activities. A further way of creating a link between activities is by starting a sub-activity within another activity.

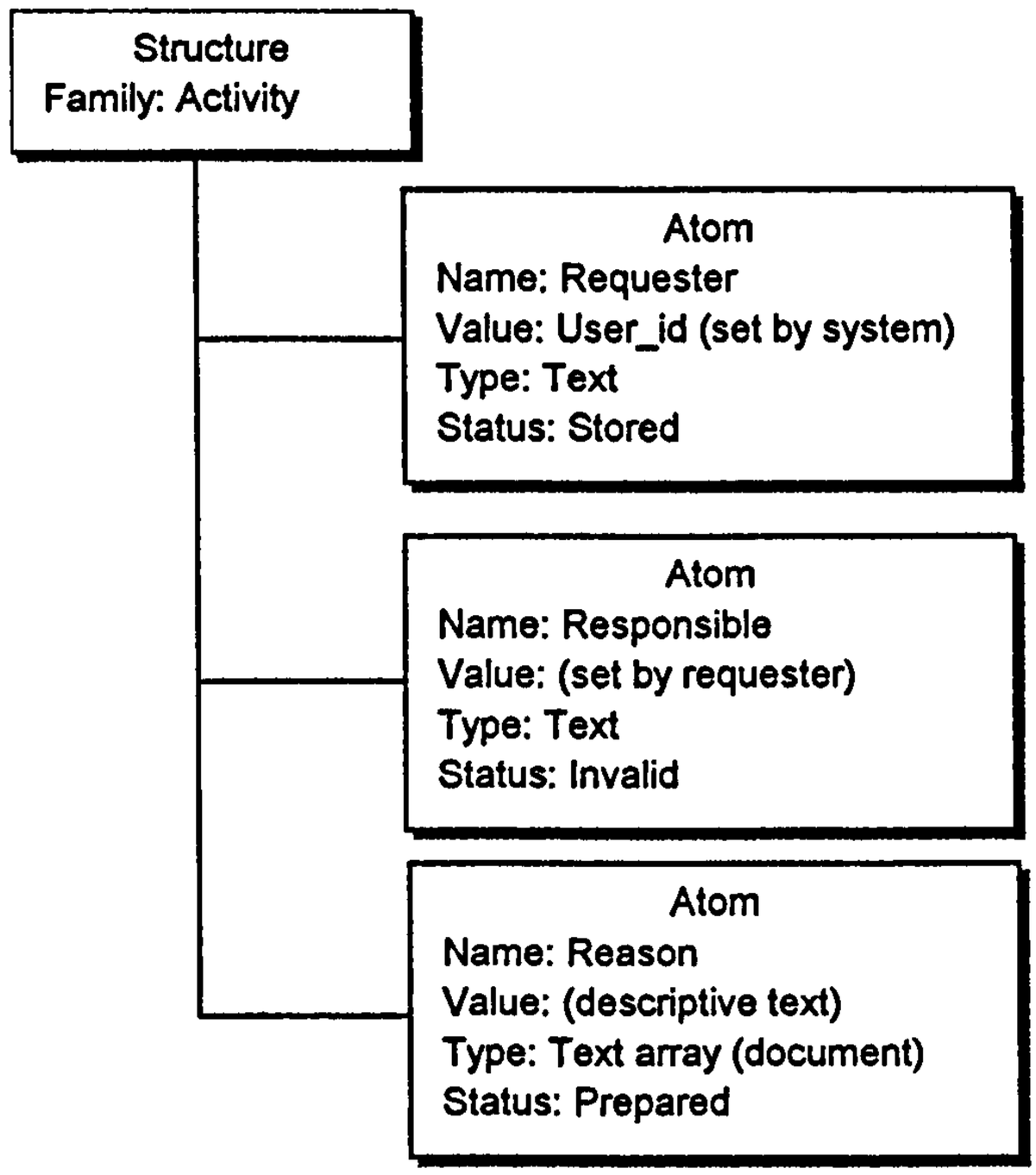


Figure 4.5 - Minimum atom complement of an Activity

These linking mechanisms provide a flexible structure through which to manage projects:

- Activity - child structures provide the highest degree of control and specify exactly which dependent activity is required. In Figure 4.6, the use of child activities ensures that no part design activity is considered to be finished without completion of a validation analysis check on the part and production of the service manual. Child structures therefore provide the opportunity to define a set of activities that together satisfy the requirements of a particular business task in a cohesive manner. Furthermore, the user is guided through the project and is presented with logical, discrete activities to complete which, in sum, satisfy the project demands.
- Pointer atoms enable any number of non-specified activities to be linked, within the bounds of the atom value constraint. Generally, the linked activity is evaluated independently of the parent and is equally applicable. Examples in other contexts are reference materials such as standards and company-wide information, such as staff lists, warehouse space or distance from the factory to the distribution centre.
- Sub-activities meet the requirement for the solution of specific parts of the problem at any time during the activity. Work on the sub-activity solution is independent of the parent without compromising the integrity of the product model data, traceability of information or maintenance of links in an explicit fashion. The use of sub-

activities maintains clarity in the parent workspace where all the information displayed can be at a similar level of granularity.

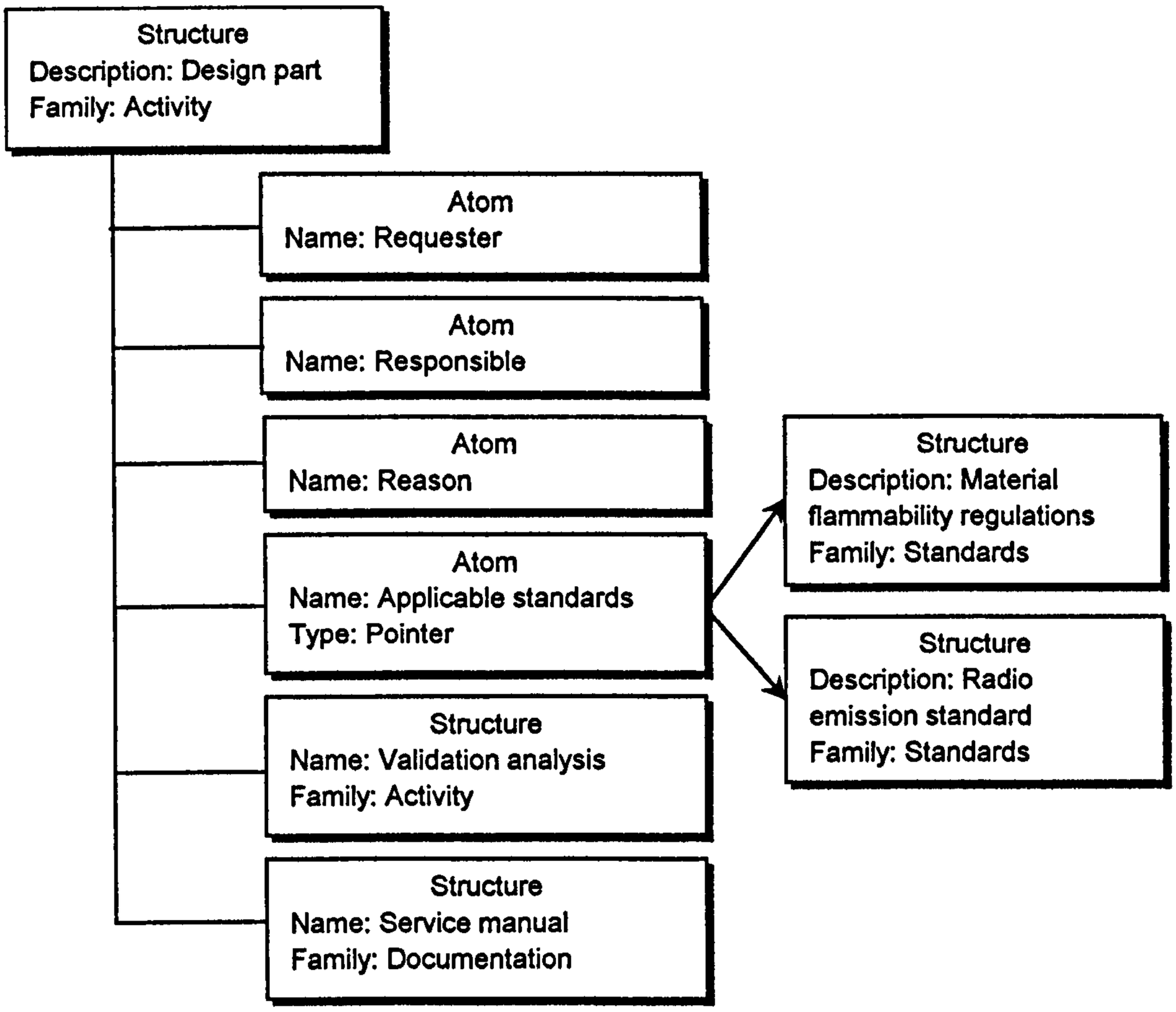


Figure 4.6 - Augmented activity definition

An additional means of control over the content of an activity is to cause the activity post-preparation method to prepare a set of structures in the activity workspace. These are the entities that the user must manipulate and whose atom values they must edit to satisfy the project aims.

Levels of dependency in activities may be more than one deep, for example a child activity structure has itself a child activity as part of its definition and spawns three sub-activities during its solution.

An activity register is used to maintain the status of an activity during operations on it. An entry is added to the register upon initiation of an activity which records the time, activity unique identifier, values of the minimum atom complement and the activity file storage location. The activity file holds references to the structure instances created under it. It does not hold the structure instance data, as this may be distributed among several storage locations and formats as appropriate. The state of the activity is also recorded and updated every time this changes. Possible states are: requested, open, suspended, closed and committed.

A separate record is held of interactions with the activity that holds details of activity opening, suspension, closure and committal. Each interaction is timestamped and the agent recorded. This is especially important when activities are issued to a group rather than an individual, to determine the actor in the activity at any time.

The sequence for activity preparation is similar to that for structures, with additional information stored in the activity register (Figure 4.7). An activity repository is an active Guide server, accessible by the client workstation across the network. The users may have a choice of repositories available to them.

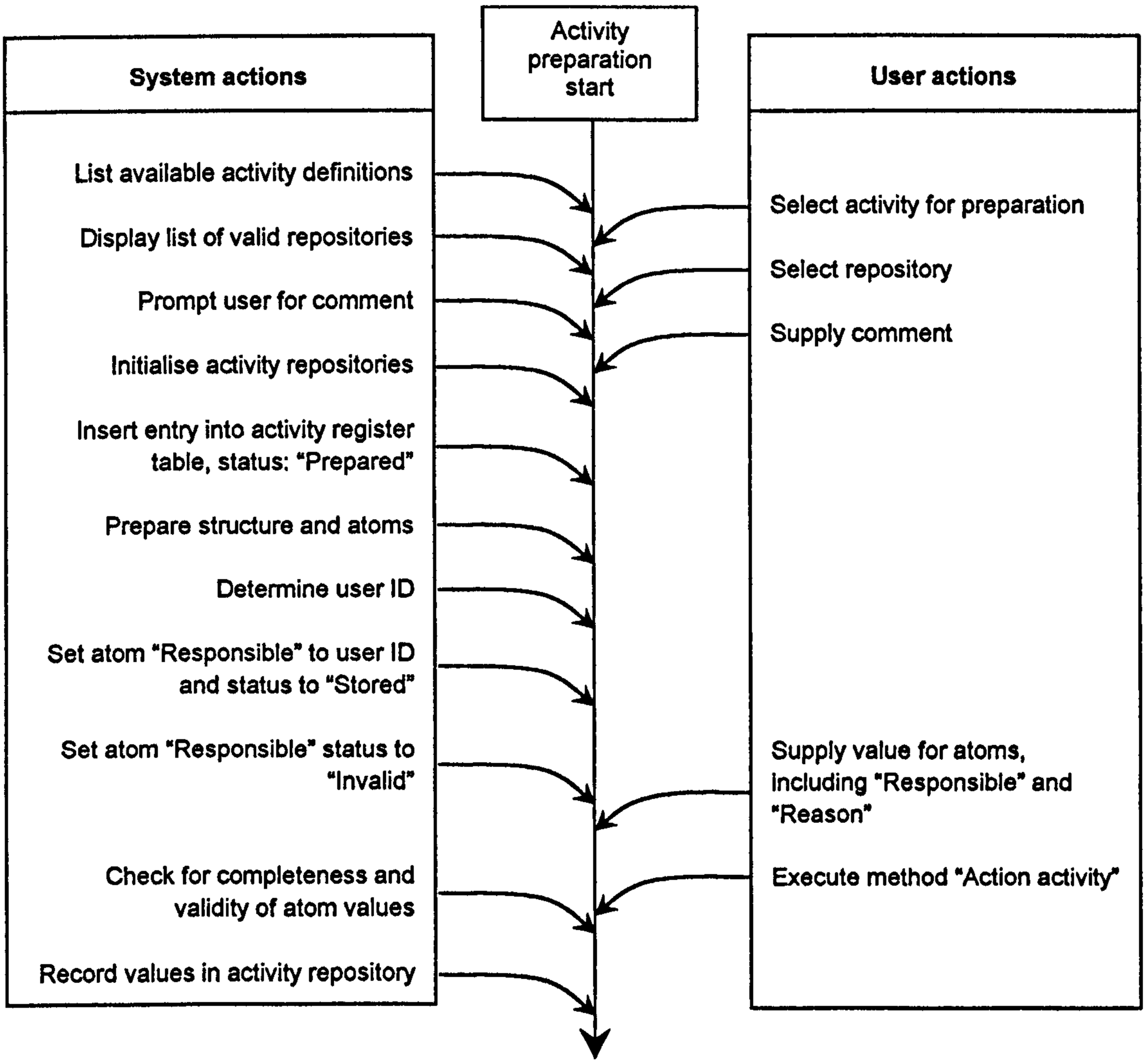


Figure 4.7 - Activity preparation sequence

4.2.6 Relationships

Relationships defined explicitly underpin concurrency in the enterprise and enable several related functions to operate in an orchestrated fashion. They occupy an important place in the operation of Guide and the facilities it supports. Guide aims to store and manage explicit relationships in the design workspace; it does not seek to represent inferential relationships.

Inferential relationships are not readily represented by the methods of knowledge engineering. The ease with which the human mind is able to understand them belies the complexity in achieving a suitable model for them. Some of the difficulty arises in the volume of background information required to formulate these relationships: it is difficult to make this available to a system in an understandable way. For example, in one machine learning project, researchers found that after several months of entering information, the system deduced that all people are famous.

The application of relationships has already been described in chapter 2; the following sections describe the mechanics of relationship definition, creation, storage and management.

4.2.6.1 Pointer atoms

A pointer atom allows a link to be made between the atom (and therefore its parent structure) and multiple other entities. These entities may be of any sort: structures, activities, relationships and atoms. For simple links, it is possible to specify the linked entity type; for more complex and multiple associations, any restrictions are applied through atom value constraints. Three types of pointer atoms exist, differentiated from each other and other atom types by their type identifier. They are pointers to geometry (atom type 9), structures (type 10) and relations (type 11).

In the creation of a reference from a pointer atom to the linked structure, the sequence of events shown in Figure 4.8 occurs.

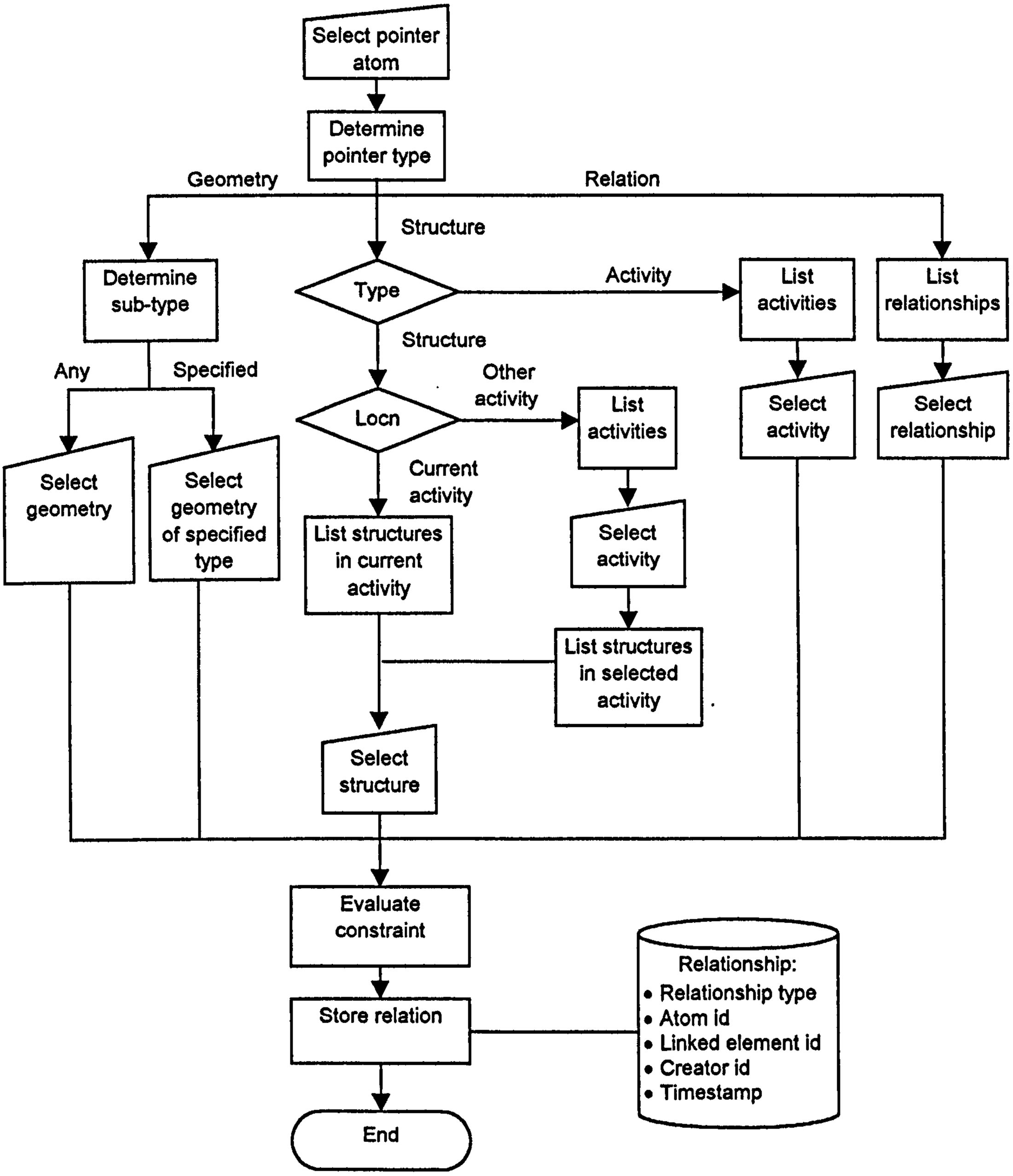


Figure 4.8 – Attribution of values to pointer atoms

Examples of the application of pointer atoms are given in Figure 4.9. Two types of pointer atoms are shown: pointer to geometry (a clamp in the example), and pointers to structures. A filter is applied to the geometry subtype that restricts the selection to a definite type of element such as points, splines or solids. No such restrictions apply to pointer to structure atoms; any structure may be so linked. Constraints may be applied to the atom value and constitute a filter to valid selections. In Figure 4.9, the type of

coating is limited to one compatible with the material, which can be retrieved from the value of the material atom.

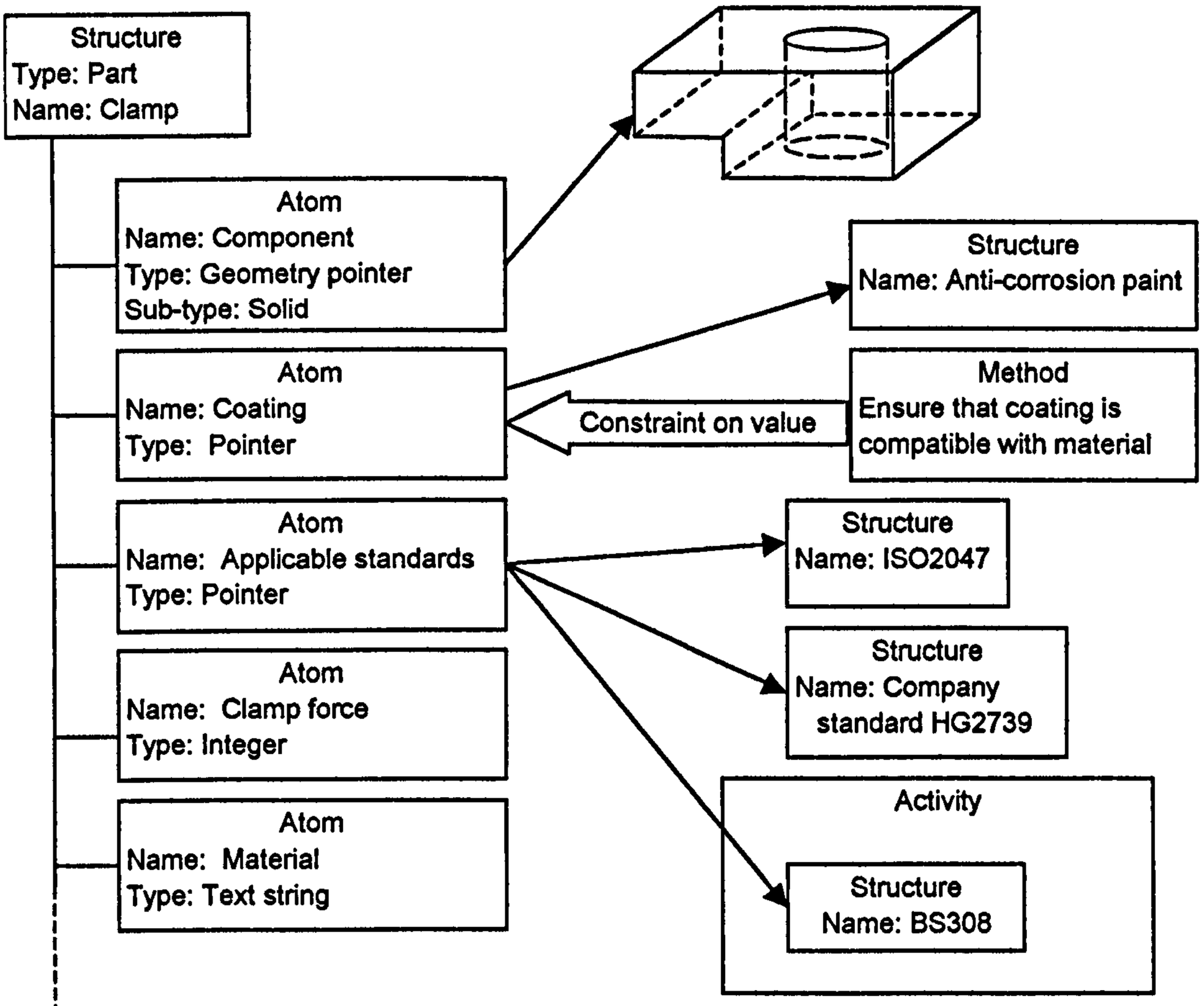


Figure 4.9 – Applications of pointer atoms

4.2.6.2 Child structures

Child structures provide the ability to refine the definition of a structure or activity while separating the detail from the principal structure attributes. They also offer the opportunity for sharing of properties among several structures without creating an explicit relationship (Figure 4.10). In this example, the motherboard and video card of a personal computer are connected implicitly through mutual links to a connector standard structure. This contains details of connector configuration and accepted use and function as well as constraints on connector geometry.

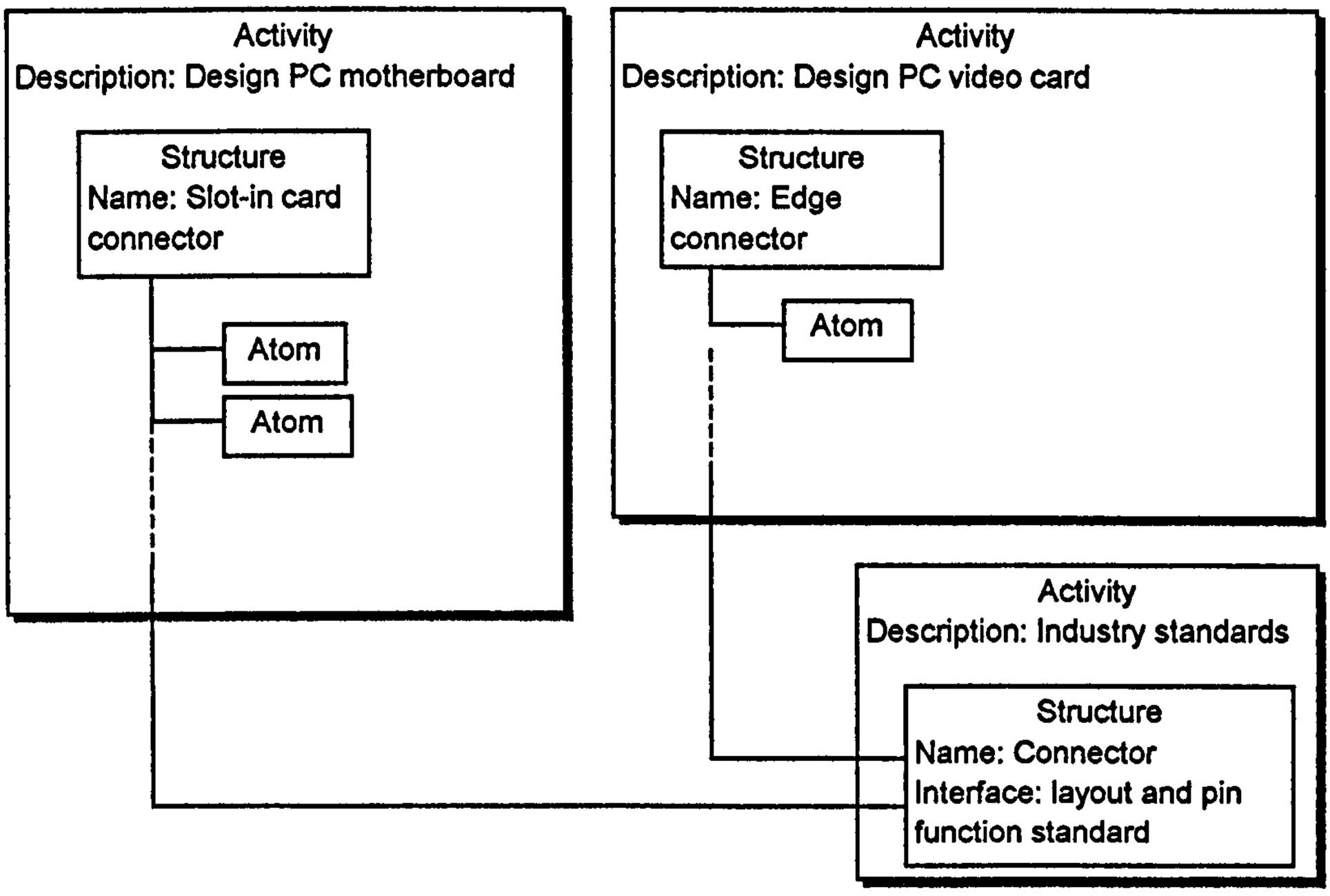


Figure 4.10 - Relationships formed through links to a common part

4.2.6.3 Mappings

The most elementary mapping occurs in the use of a value taken from one atom by another atom. The atoms can belong to different structures which are of dissimilar types and which reside in different activities. The mapping is stored in the relations table.

Should the linked value be changed at any instant, the operator is warned of any dependencies on the value. The user may choose to take one of the following actions:

- Change the value and break the link. In this case, the link creator is notified, and may, appropriately, re-form the link.
- Change the value and propagate the change. Subsequent actions depend upon the Action on change indicator of the linking atom. This was described in Chapter 2.
- Cancel the change.

If the linking atom value changes, then the link to the previously linked atom is broken. No notification is given to the owner of the linked atom, as they did not establish the dependency.

The mapping sequence action is shown in Figure 4.11. Further mappings are possible through the application of methods, described in section 4.2.7.

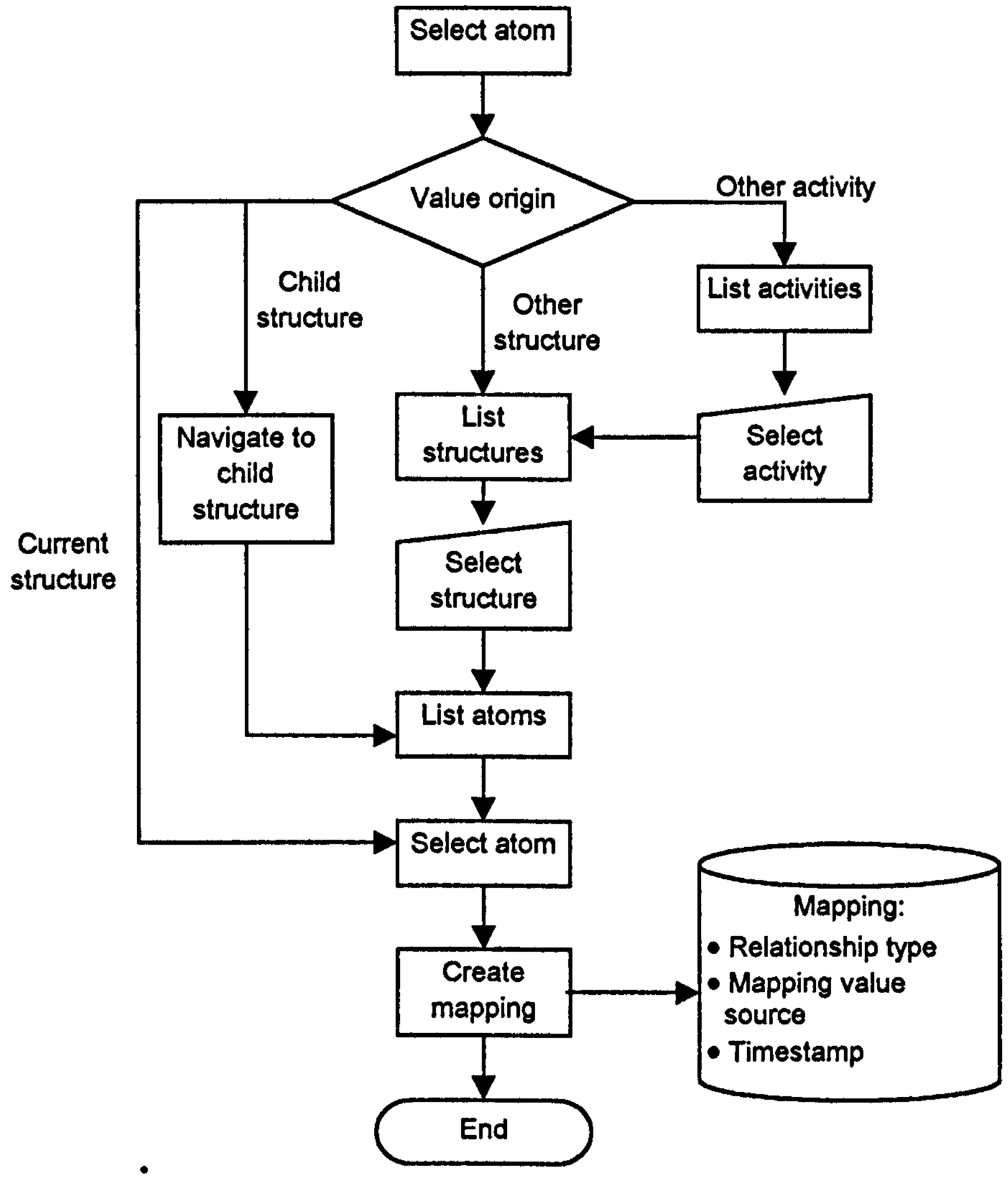


Figure 4.11 – Operations creating maps

4.2.6.4 Group relationships

Not all relationships are well structured, and many computing systems employ the concept of a "bag" of loosely related elements. Guide addresses this need by forming a group relationship, from which entities may be added or removed. For example, individuals from different departments may form a working group, which is represented by a group relationship.

A common application for the group relationship is in the mapping of the output values of related methods. If several atoms derive their values from the output values of the execution of the same method, they are linked in a group relationship. This is desirable, since the values of all the atoms in the group came from the execution of a method using one particular set of inputs. This is particularly true when several values are set from a tuple from a database query.

4.2.7 Methods

Methods support many of the essential functions of Guide: utilities to support the designer, retrieval of data, constraint execution, access to external programmes and data. Despite the diversity of their application and functions, they are controlled by the same fundamental rules. The definition of a method covers its executable and the inputs and outputs associated with it.

4.2.7.1 Method types

Methods are categorised in three types and can be deployed in the roles described below. Some methods will be peculiarly relevant to specific applications. Particularly, straightforward database queries are less likely to be used in applying constraints and more likely to be used to set atom values.

- System methods (Figure 4.12). These are programs defined to carry out system tasks. The method controls all of the user interactions and display routines; the sequence of execution is:
  - the user is presented with a list of input values required by the method;
  - input values are specified by the user;
  - the system method is executed;
  - outputs are displayed and mapped.

  The inputs and outputs are optional and these steps are only executed if either are detected. It is possible to have methods executing without user interaction, for example system constraints.

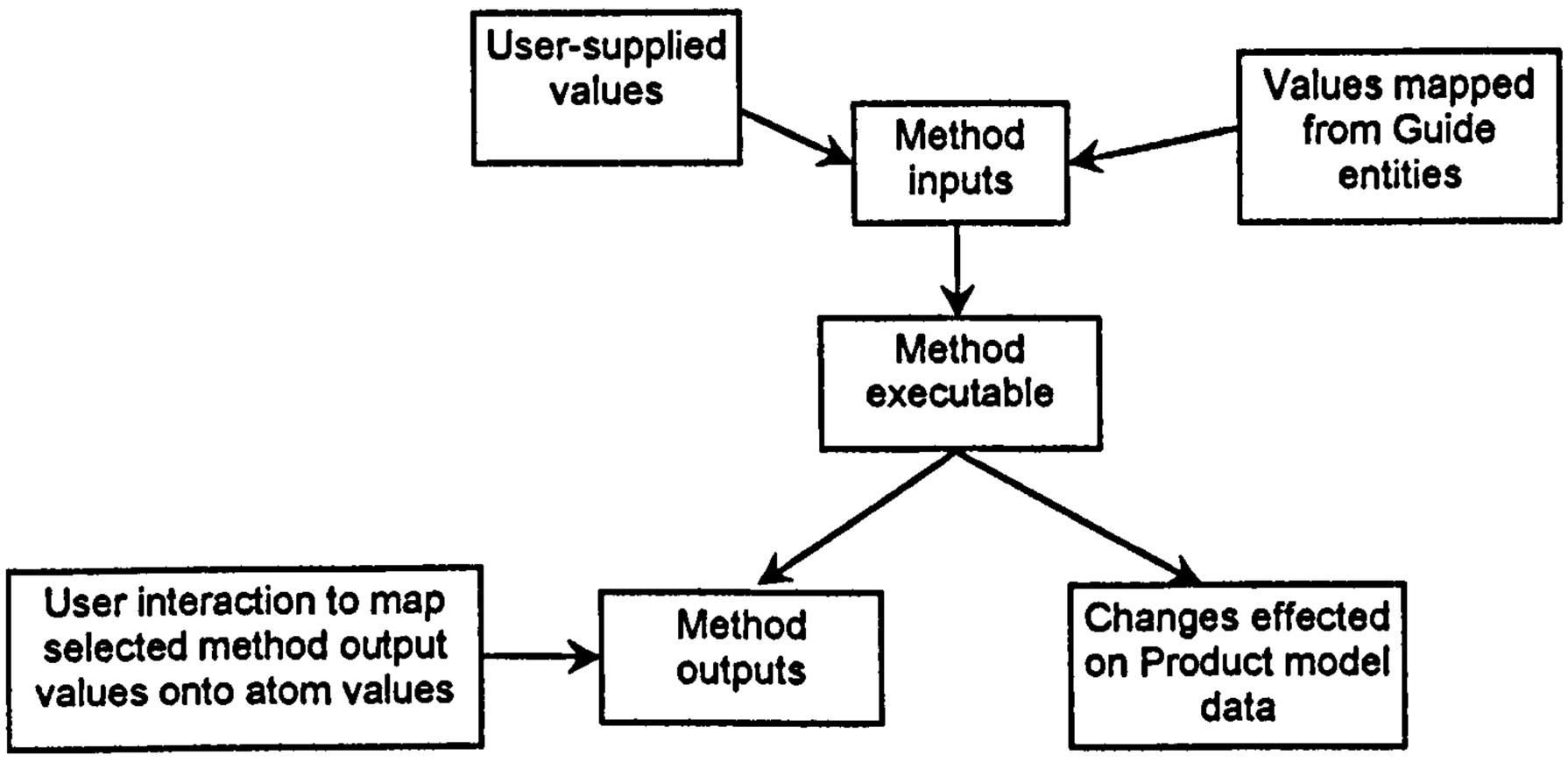


Figure 4.12 - Flow of system method execution

- Interactive methods (Figure 4.13). This type of method allows for interaction and user intervention during its execution. This allows the user to influence the course the method takes and make choices based on the results supplied partway through

the method. The user controls the flow of the method using the tools provided for him or her by Guide. To signal the end of the method execution, control over the program flow is returned to the Guide system. Whereas system methods are essentially static, interactive methods are dynamically created and compiled or interpreted. Their content and function are not pre-determined. Third party software of choice may be invoked from an interactive method, even when this does not have an open architecture and callable API. This also allows the integration of new pieces of software into the system as and when these are developed and released [3.2].

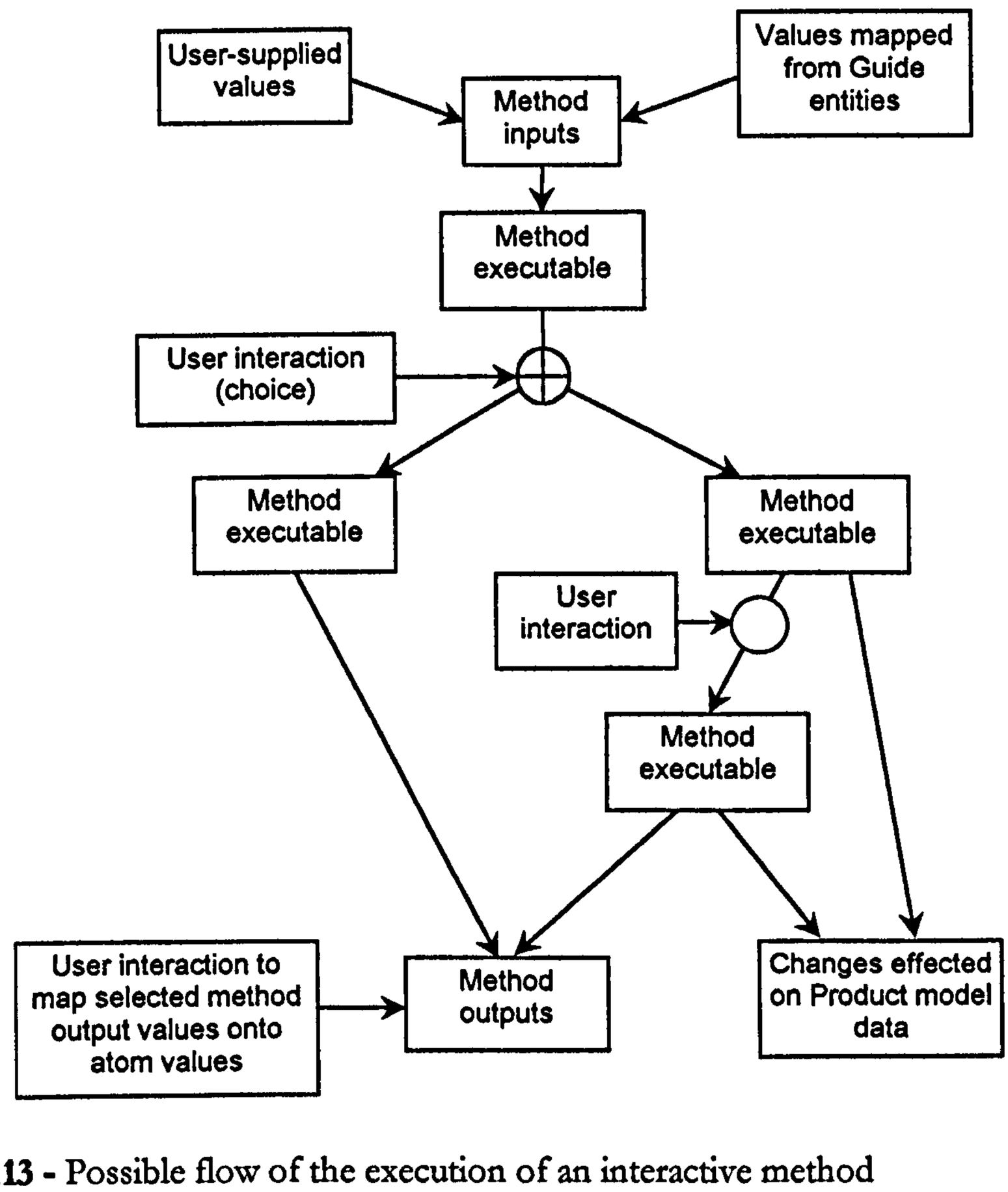


Figure 4.13 - Possible flow of the execution of an interactive method

- Database access method (Figure 4.14). An SQL interface is provided to allow access to data stored in relational databases. The database table(s) from which the data is to be retrieved are identified in the method's definition. The user can build a search condition to retrieve information from the database tables that is applicable to their particular requirements for the problem they are currently addressing. An interactive query builder assists their task (Plate 8).

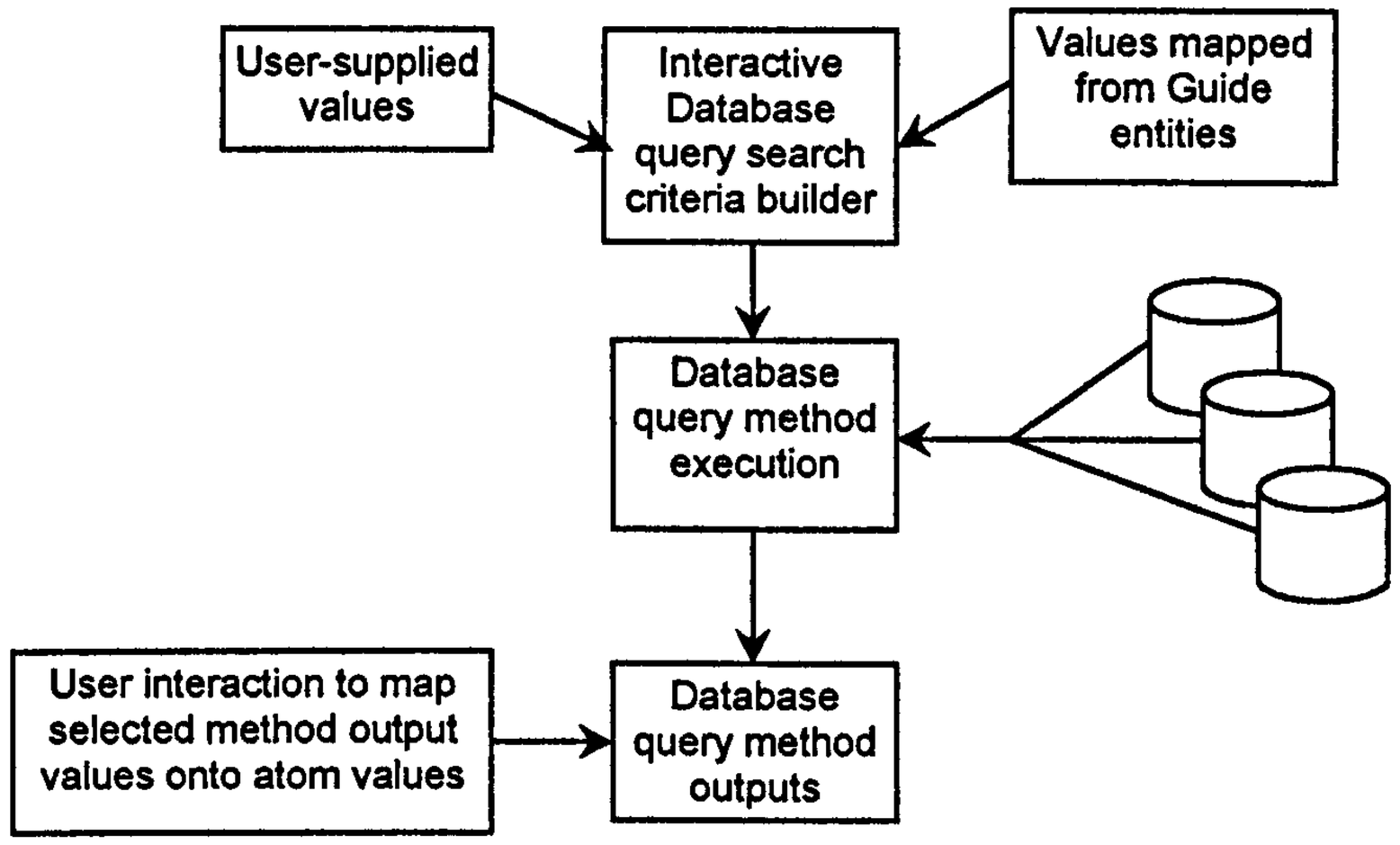


Figure 4.14 – Process flow of a database query method

4.2.7.2 Constraints

Constraints are generated by methods, executed by the system, that act either before or after a change is made to the product model data. They provide the opportunity to engage the following functions.

- Integrity checking.
- Assessment of the appropriateness of the value.
- Communications. At any stage of the preparation of a structure, edit and commitment, notification may need to be sent to different actors involved in the project. For example, when the design and manufacturing stages of a moulding die are completed, messages may be sent to the company accounts department to issue an invoice for the tool and to the logistics department to arrange shipment of the tool to the customer.

The types of constraints and their application are listed in Table 4.3.

Guide provides two strategies to set the default value of atoms. The first applies methods that are linked to atoms individually. The second can set values in one or more atoms once the atoms have been prepared. In this case, the method providing the value is linked to the structure. Atoms used by several structures carry their default setting method with them. The group defaults method provides the opportunity to override that default when required by a particular structure. Eastman [1.6] gives a name to a structure, its children, constraints and methods: an accumulation.

Constraint	Execution	Function
Pre-prepare	Before structure or activity is prepared	Ensures that any pre-requisites for the structure about to be prepared are present
Atom default	Executed for each atom, once they are all prepared.	Set the default value for each atom, where this is not a simple value that can be stored in the atom definition default value attribute
Group defaults	After structure and atom preparation	Sets defaults for one or more atoms. Carries out any actions concomitant to the structure preparation
Atom value constraint	After edit of atom value	Apply any constraints onto the values proposed for an atom. Integrity and validity check
Structure completeness constraint	Before structure commit	Ensures that all atoms are set to proper values and that the structure information content is complete and accurate
Concomitant action	After structure commit	Engages any processes to be executed after the completion of the structure
Pre-method constraint	Before method invocation	Ensures that the prerequisites for the method firing are satisfied. Important where a method causes irreversible changes or actions to occur

Table 4.3 - Constraint types and their execution conditions

4.2.7.3 Methods output mapping

Methods are used to assist the designer to provide meaningful and appropriate values for the structure atoms and to perform support functions during the course of the engineer's work. Three types of methods exist, when characterised by their associations with other product model entities (Figure 4.15). These are:

- Atom methods – a method can be attached to an atom by declaring that one of its outputs is a valid source for the atom's value (Plate 9).
- Structure methods, where the method executable links to the structure header; no further constraints are made on output mappings.
- Unattached methods, which have no pre-defined output or association mappings, and which may be used universally within the activity. An example might be a calculator method.

When the output of a method that is tied to an atom is single valued, the mapping can be automatic. When the method returns several output values, one must be chosen before the mapping to the atom can take place. This is also the case with database queries, which return selected tuples under the database table columns defined by the query. Each of the results columns can be tied to an atom; any value returned under that column is a potential value for the atom. The user must choose the desired tuple from the results before the method can assign the correct value to the atom. When the

execution of a structure or unattached method is completed, the user must select both a value from the method's outputs and identify the atom onto which it must be mapped, since the mapping is not defined.

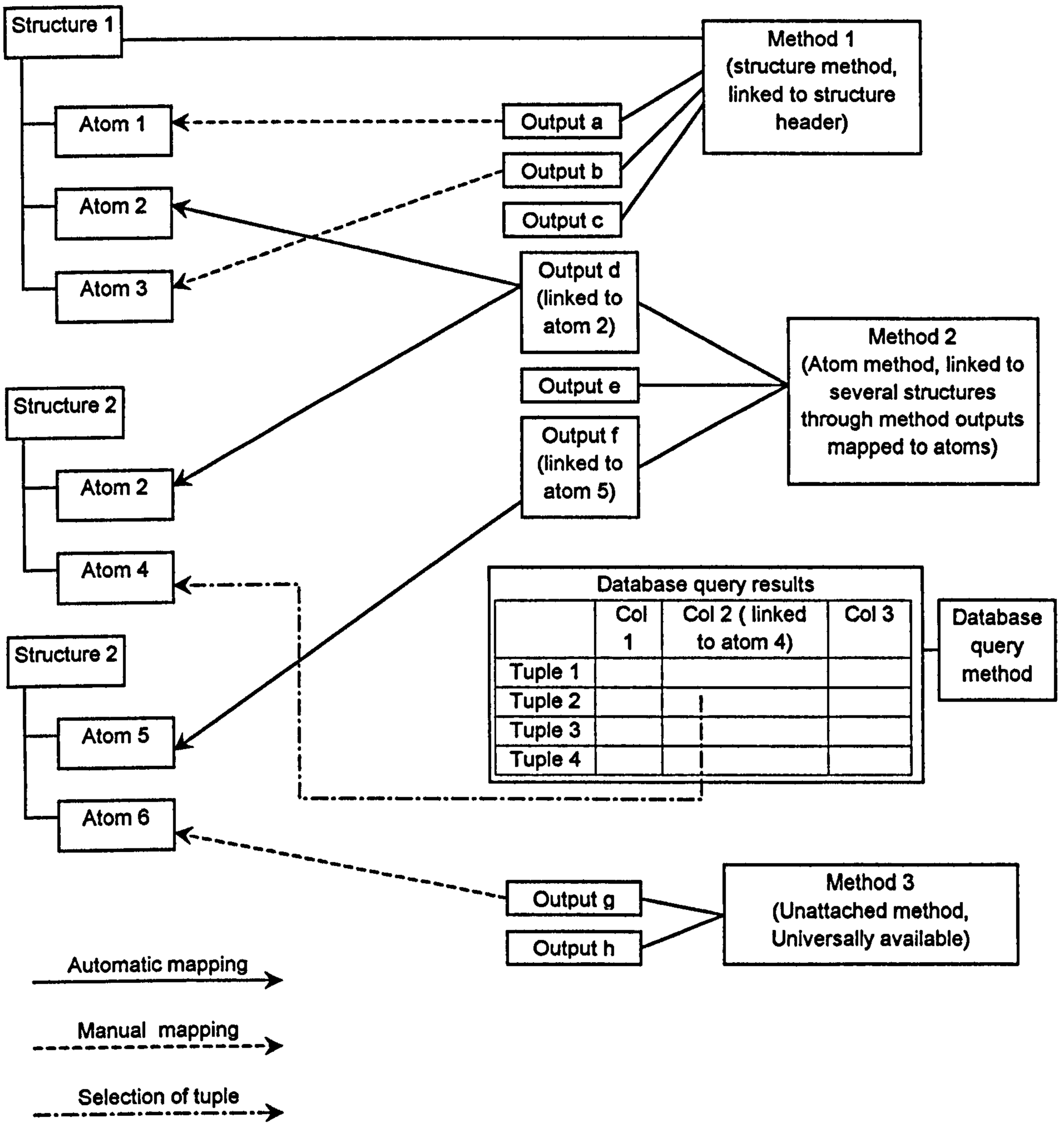


Figure 4.15 – Association of method types

A single method can generate values for atoms in different structures. Where it is necessary to map all the values at once because they are valid only when used simultaneously, problems may arise if one of the recipient structures is unprepared. In such cases the method can incorporate the instructions to necessary prepare the structure and so enable the simultaneous mapping of its values to the appropriate atoms.

A single method may be attached to several atoms and structures. As atoms are used in different structures, they carry not only their information content, but also their rules to test integrity and value provision aids. This approach is in contrast to the object-oriented paradigm, where methods belong to the object and offers the ability to easily re-use code. A method can be used in different circumstances; each of its method outputs may be associated with different atoms and so be invoked to different purposes dependent on the context in which it is required. A method may also, by virtue of the links made to it and its method outputs, act alternately as a structure or an atom method.

Methods can obtain their input values from a variety of sources (Figure 4.16). Their provenance can be:

- a value keyed in by the user,
- an atom value, from the current or other structure and in any activity,
- the output of another method.

Methods can therefore be nested and used in conjunction, one with another. The mechanism for linking methods is similar to that for linking a method to an atom. A method output is linked to the input of another method as a suitable source of the required input values. As with structures, several outputs can be linked to one input and one output can be a valid source for several atoms.

This is another instance of the economy of the Guide representational schema. By allowing methods to nest in an infinite range of combinations, supple and complex execution paths are presented to the user, with a minimum of code necessary. Code re-use is obviously easy, requiring only that links be created between method data.

The dotted line between atoms 2 and 4 in Figure 4.16 denotes an implicit relationship created between them because of the method execution. Though this relationship is not stored explicitly within Guide, it is always available through the design history tree.

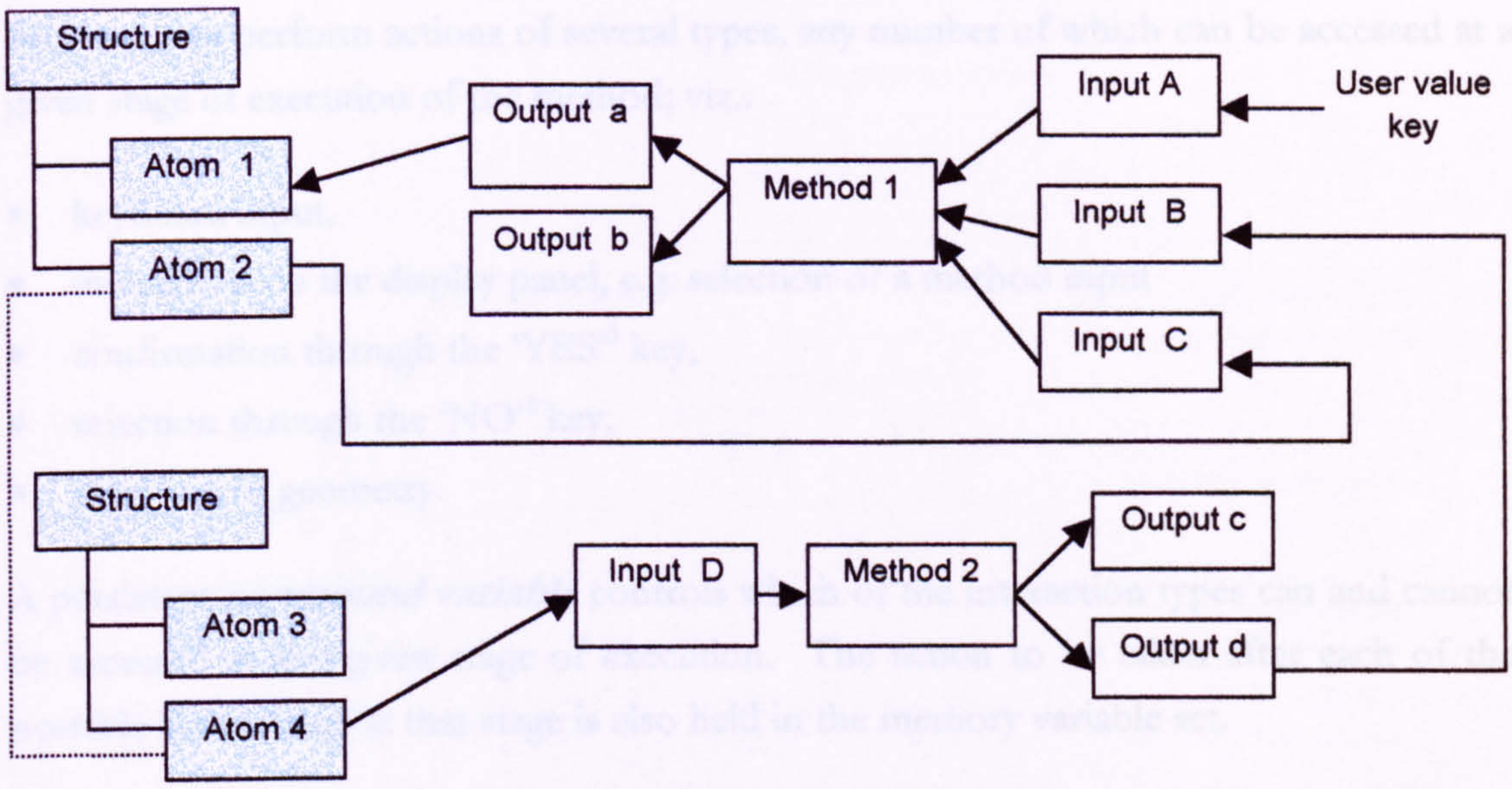


Figure 4.16 - Method input sources

Method inputs and outputs are collectively termed method data. They share a representational format, which has an 'IOType' flag to differentiate between them. This simplifies the association of method data with methods, since only one relationship type is necessary: method to method data. The representation block for method data is shown in Table 4.4.

Field	Description
OID	Unique ID of method data
Prompt	Displayed against prompt as descriptor
Iotype	1 = Input 0 = Output
Vtype	Data type, values as for atoms
Varray	Array size, for array data types
Vlength	Value length, applicable to text strings
Name	Method data variable name
Mapping	Mapping object (optional)
MapMeth	Method used to effect the changes to the target atoms (optional)
Version	Method data version number
Creator	Creator unique ID

Table 4.4 – Information content of method data representation

4.2.7.4 Construction of interactive methods

Guide supplies a resource toolkit to assist developers of interactive methods and allow them to concentrate on the functionality of the programme rather than on user interface design and flow control considerations. The user interface employs panels based on ***graPhigs*** constructs.

The user can perform actions of several types, any number of which can be accessed at a given stage of execution of the method; viz.:

- keyboard input,
- interaction on the display panel, e.g. selection of a method input
- confirmation through the 'YES'[1] key,
- rejection through the 'NO'[1] key,
- selection of geometry.

A persistent ***compound variable*** controls which of the interaction types can and cannot be accessed at any given stage of execution. The action to be taken after each of the possible interactions at that stage is also held in the memory variable set.

Instead of using the Guide user interface options, the method writer may take control of all interactions with the method and the user interface. In this case, the re-entry point must be defined carefully and control returned to Guide from the user exit routine. The options for method execution are shown in Figure 4.17.

The structure of the variable set that holds the information necessary for Guide to control the flow of interactive methods is shown in Table 4.5.

In the example in Figure 4.17, the settings are: Ikey = 0, Ipan = 1, Iyes = -1, Ino = 1, Isel = 0. This means that valid actions are a panel, Yes or No interaction. Entering values from the keyboard and selecting objects in the workspace are not valid options and are not available for this interaction. A panel interaction will take the user to the next step in the interaction, the No key would return the user to the previous state and the Yes key would accept the state of the method and exit.

---

[1] The Yes and No keys are for acceptance of the current interaction or rejection of the results of the previous one. The concept is used in several commercial software applications, including Catia

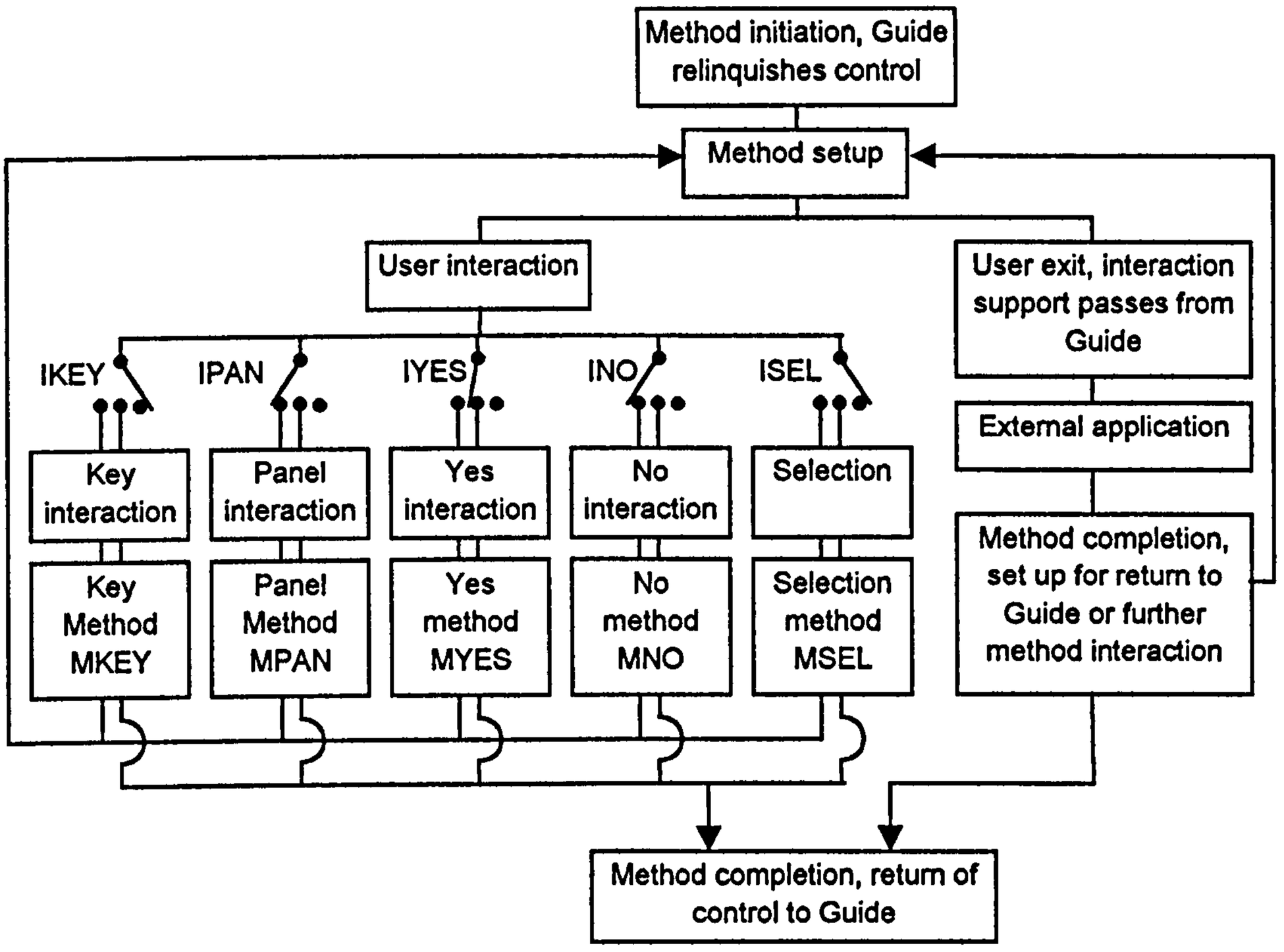


Figure 4.17 – Interactive method:- control of execution

Variable name	Description
IKEY	Switch for Key interactions. Possible values are: >=1: Interaction valid 0: Interaction not valid. The interaction is not offered as an option to the user <=1: Interaction valid and exit after method execution
MKEY	Name of method to execute in the case of a keyboard entry onto the command line. The expected data type, if specified, is held in ITYPE
IPAN	Switch for panel interactions. Values as for IKEY.
MPAN	Name of method to execute in the case of an interaction with the method input and display panel.
IYES	Switch for 'YES' key interactions. Values as for IKEY.
MYES	Name of method to execute in the case of a YES interaction. This is a keyboard entry onto the command line.
MNO	Name of method to execute in the case of a YES interaction. This is a keyboard entry onto the command line. (The NO interaction is always valid, hence the lack of INO value)
ISEL	Switch for selection interactions. Values as for IKEY.
MSEL	Name of method to execute in the case of a selection in the workspace of geometric elements. This reads the YES interaction. This is a keyboard entry onto the command line.
ITYPE	Data type for keyed and selected values (same as for atoms).
ISTYPE	Data subtype, e.g. to specify selection entity type (same as for atoms).

Table 4.5 – Variable set for the control of method flow

4.2.8 Geometry representation

Geometry has always occupied a central position in the representation and communication of engineering designs and deserves particular attention. In some newer branches of engineering design, such as electronics, function takes precedence over form, but in mechanical and civil engineering, two-dimensional drawings continue to occupy a very important place in the lifecycle of the design project.

As with other aspects of product representation, the communication and accessibility of the stored information is as important as the capacity to store it. Access is required to enable links to be built between the geometry and other objects in the design workspace. A distinction is made between access to the geometry as a whole and access to specific aspects of the geometry. For example, finite element analysis can be performed over the whole geometry and the mass determined from the volume of the whole body, but a surface finish would be applied to a defined area of the part.

Geometry also played a significant role in the development of Guide. Its roots lie in a software utility, written as an extension to a CAD/CAM package, which selected from a library tooling suitable for the manufacture of the part under consideration, and applied appropriate machining parameters. This system was driven by the user making choices from options presented in a series of panels. Part of the information required for the system to work was the nature of the geometry and the material. It was realised that the material constituted an attribute of the part, and the geometry attributes, which would determine the choice of tooling, should be available from the CAD model, an idea echoed by Chen in [4.5]. As a result, the first feature-based modeller (FBM) was written, the precursor to Guide.

According to Chen [4.5], a feature is an entity which captures a characteristic of a local area of a part, however many other definitions exist such as those given by Wang et al. [4.6]:

> *'a specific geometric configuration formed on the surface, edge, or corner of a workpiece intended to modify outward appearance or to aid in achieving a given function; regions of a part with some manufacturing significance; a computer representable data relating to functional requirements, manufacturing processor physical properties of design; a higher level clustering of dimensional, material-related and shape information which implicitly contains a specific functionality.'*

The feature-based approach has supported a wide range of applications, although they remain tightly focused around geometry for design and manufacture, rather than considering it as a subset of a more general representational model. Chen and Wei use it to support the application of manufacturing rules and advisors [4.7].

The FBM had a set of standard form features, with the capability for users to define additional sets. An integrated process planning utility was provided, which interrogated the model to find the types, occurrence and parameters for each of the features contained in the model. A draft process plan, containing options available for the manufacture of the part was presented to the user. Users could then refine the process plan, select their preferred manufacturing options and establish an operational sequence. A similar approach with a reduced set of features was later taken by Chen et al. [4.8] and in 2D by Wang et al. [4.6].

It was quickly decided that the standard feature set was restrictive, and that users might be better left to specify features of utility to them. This depended upon a framework in which any entity could be represented. Shah and Rogers [2.7] declared that a feature definition mechanism must be immensely flexible to allow designers to define features in any form, at any level, and in any combination, as appropriate to their needs. In this statement, they were already alluding to features beyond geometry, in line with the concept of a Guide structure. Features became one part of a system, rather than driving the design and manufacturing processes. Guide now employs a parametric feature-based modeller to manage geometry visualisation. The geometry template is an attribute of a structure and can be of any complexity or size from a hole to a complete casing. The parameters are driven through the Guide interface as structure atoms.

The above, seemingly simple paragraph has wide-ranging implications. It gives Guide the capability to represent geometry in three dimensions and to associate other information with the whole or parts of the geometry further information. Parts with a geometric representation can be linked together in any desired way, e.g. in a Bill of Materials. Dimensional and positional relationships can be defined between parts and fits evaluated.

The geometry is available to external programs. It can either be interrogated directly to provide information on any aspect of the geometry (how many holes are there in this plate?) or be used to export geometry in a neutral format, using the full range of expressions offered by STEP/PDES appropriately.

Geometry is linked to a structure through a geometry pointer atom. Other atoms in the structure can be used to drive the geometry parameters. When the value of variables is changed, the geometry is not immediately updated. The reason for this is that sometimes a single geometry update may result in invalid geometry, whereas a combination of two has the necessary integrity. The current power of computing platforms is also catered for, as updates on complex shapes are time-consuming. As CAD systems and hardware platforms develop, this can change to give real-time shape representation. This illustrates

an aspect of the flexibility of Guide: the method need only be changed once for all geometric entities to use the new geometry update regime.

Geometry has been used as a driver for other functions. Geiger and Dilts [4.9] formulate a design to cost systems to help in the area of cost awareness in the design system. Based on a set of rules and feature–based, it calculates at every step the likely cost implications of the design created. It also uses feature similarities and group technology to scan the company's database for components with similar properties. Even if no component is found which can replace the proposed one at less cost, it is possible that exposure to the other products will catalyse ideas for cost reduction, perhaps by incorporating some their features. It does not operate at an early enough stage to have an impact on cost commitment. It is an example of tools that help optimise the product in later stages of its development process, in this instance against cost.

Das et al. adopt a similar system, but extend its capabilities to the automatic redesign suggestion stage to eliminate costly machining operations by replacing them with easier to machine ones [4.10]. Such tools could be enhanced and applied through methods.

4.2.9 Navigation

The explicit relationship definitions held in the Guide product model data permit the user to navigate the design workspace (Plates 10-13). A top-down approach may be taken, listing the activities available to the user to view and opening one of them in read-only mode.

The user may also start from their current position and follow links to other entities in the product model data. A 'Navigate' button on the Guide interface loads the object attached to the current link (Plates 11, 13). Where more than one object is linked (for example, a pointer atom which hold several references), a list of the linked items is displayed for user selection.

4.3 Guide Manager

The Guide manger provides a graphical environment to assist the Guide system administrator. Structures and activity headers are defined through a series of panels (plates 14-16). Method data and atoms are similarly defined, and method executables defined to the system. Once these elements are defined, links can be created to build structures, methods, activities and associations between them.

The product model has a structure and relationships at the entity definition level. The Guide Manager allows the administrator to navigate the product model in a similar

fashion to the facility offered to system users navigating product model data. The links are the formalisation of the natural relationships which exist and which are exercised during the design process [4.11]. The system administrator can use Guide manager functions to inquire of entity definitions and follow links defined in the product model, e.g. from structure to atom to method data to method definition. The administrator must know to which structures an atom is attached to before it is updated, or a new version created.

Navigation is especially important when the administrator establishes a set of activities as might be required by a design automation system. In the creation of such collections of activities, structures and methods, it is essential that the completeness of the system constructed can be confirmed by following the links that were previously defined.

4.4 Control system

Guide permits users to work in a fashion that is natural to them. There are no *a priori* restrictions on the sequence in which operations are performed or the nature of these operations [1.20]. Furthermore, it permits recursion to deep levels in areas such as method execution and structure creation. For example, a constraint on an atom value could cause a new activity to be generated whose type is determined from a database query. In order to support the flexibility of the system, the control mechanism must permit actions at any time, while retaining its current context and ensuring that any work started is completed. In the above example, the activity needs to be properly prepared, the constraint on the atom value needs to be completed, and the value accepted or rejected based on the constraint results.

The environment under which Guide version one was developed was of a procedural nature and did not support event-based programming similar to a Microsoft Windows environment, where the user can perform a variety of tasks in any order, rather than following sequences of operations which are pre-determined. In any case, this programming approach may prove to have deficiencies in dealing with the level of complexity and nesting of Guide interactions. The system controls carefully those interactions that are valid at any time, allows them to happen and stores the context in which they were invoked so that, when they are complete, the original task can continue. The basis for this system is a set of complex variables and Last In First Out ***variable stacks*** in which to hold them. Figure 4.18 shows the symbols used in the flow control examples.

Filter for command branch. Flow passes though the filter if the current stack values match the filter for Type (T) and Subtype (ST) .

Set control stack value. The bottom (current) value in the control stack is overwritten with new T / ST values. n is the number of values on the stack

Push value onto control stack. A new T / ST value is added to the bottom of the control stack and becomes the current value. n is the new number of values on the stack.

Pull value from control stack. The current T / ST value is pulled from the bottom of the stack. The next value in the stack becomes the new current value and is indicated. n is the new number of values on the stack.

Figure 4.18 - Control flow diagram legend

As an example, Figure 4.19 shows a fragment of the control flow, during the preparation of a structure. The role of action filters and persistent compound variable stacks is illustrated.

The flow has been simplified slightly for the example. Thus, the flow is depicted as being linear down the page. In reality, all interaction types are available at any time and are activated by the values on the control stack. The use of some of the variable stacks has been illustrated. When a particular interaction Type/Subtype commences, it retrieves its input data from the appropriate variable stack. The various interactions are described in Table 4.6.

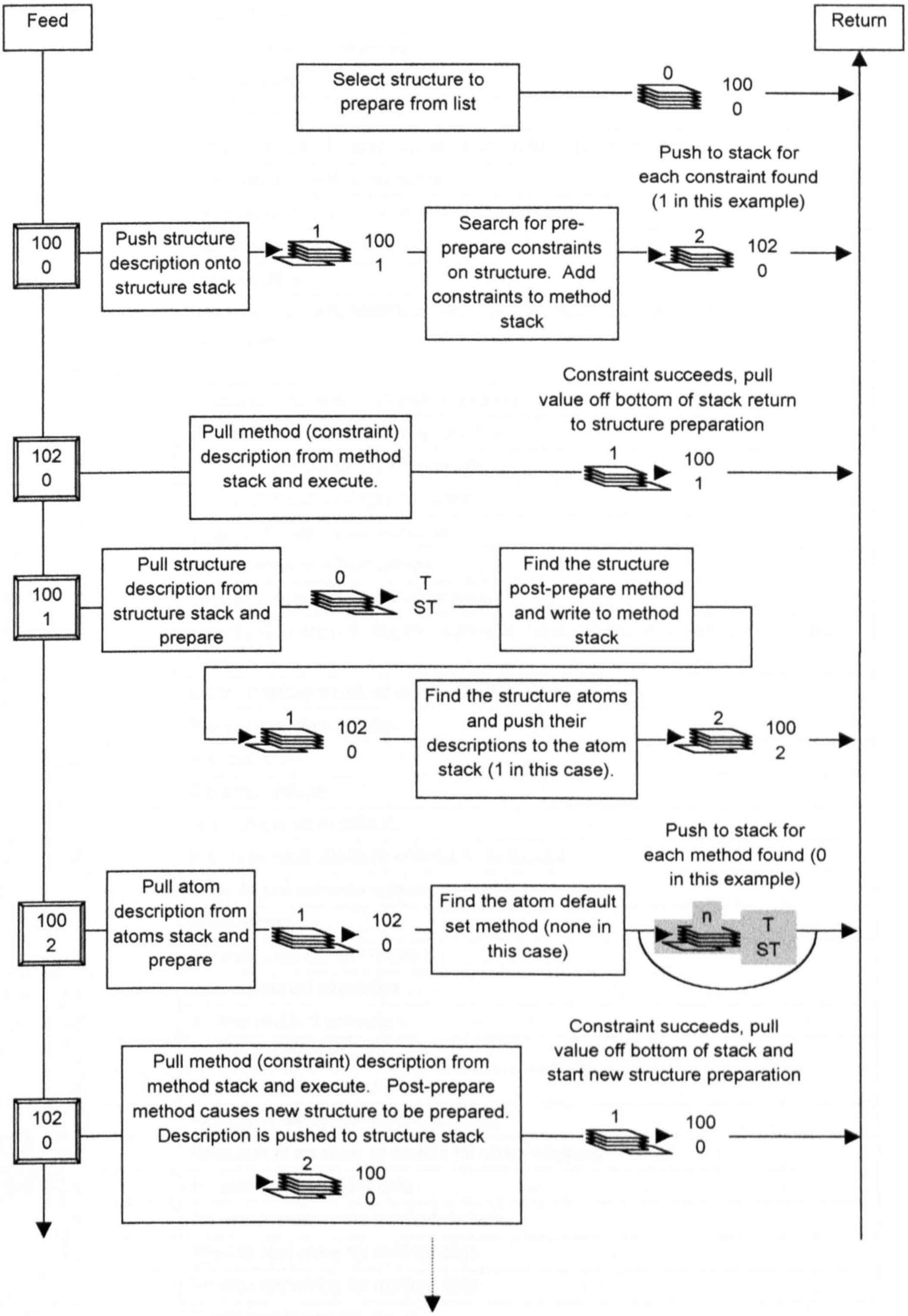


Figure 4.19 – Flow control during the preparation of a structure

Type	Subtype	Description
1		Key interaction - integer value
2		Key interaction - real number
3		Key interaction -text string
4	Size	Array definition - Integer number array with <size> elements
5	Size	Array definition - Real numbers
6	Size	Array definition - Text string array (document)
7		Vector definition
8		Plane definition
9	Geometric element subtype.	Geometric element selection. Subtype matches the subtype values for atom interactions.
10		Pointer to structure relationship creation
1		Pointer to relation relationship creation
12		Pointer to activity relationship creation
13		Matrix definition - Integer numbers
14		Matrix definition - Real numbers
15		Matrix definition - Text strings
20		Child structure association interaction
99		Select child / atom during structure edit. Selected element becomes the current atom.
100	0	Initiate prepare structure interaction sequence
	1	Prepare structure header
	2	Prepare atom
	3	Set atom default
	4	Set multiple atom default
101	0	Initiate commit structure interaction sequence
	1	Execute pre-commit method
	2	Store atoms
	3	Execute post-commit method
102	0	Initiate method execution
	1	Normal method execution
	-1	Interactive method execution
	2	Database query execution
	-2	Interactive database query execution
103	0	Selection of an atom as source for value mapping
104	0	Process method input data
	1	Provide integer value for method data
	2	Provide real value for method data
	3	Provide text string for method data
105	0	Display method error panel
106	0	Display method error panel for interactive methods

Table continued on next page

Table continued from previous page		
Type	**Subtype**	**Description**
107	0	Process database query input data
	1	Add table column name to SQL search conditions
	2	Add search operator for numeric data types to search conditions (=,>,<,>=,<=,<>)
	3	Add search operator for text data types to search conditions (=, like)
	4	Add a logical search operator to search conditions (not)
	5	Specify integer as search constraint
	6	Specify real number as search constraint
	7	Specify text string as search constraint
	8	Select join operator (And / Or)
108	0	Set atom value and fire constraints
	1	Constraint firing
	2	Display atom dependants warning panel, when the atom has dependants which will be affected by a change to the current atom
109	0	Set geometric type atom value and fire constraints
	1	Constraint firing
110	0	Create activity
111	0	Process method
	1	Process method constraints
	2	Execute method
112		Select line from method outputs. This can then be used to map a value from the line or tuple onto specified atoms, including the current atom
113		Actions on method output selected single line
114		Action on value selected from 113
116		Process of mapping atom values from multiple method output
117		Process of mapping atom values from another structure
118		Process pointers to atoms
119		Add a link to a pointer
120	0	Replace a link to a pointer
	1	Process constraint on pointer value
121		Delete a link from a pointer atom
125		Operate on child structures
130		Process link to geometry

Table 4.6 - Interaction type codes

Guide uses several persistent compound variables in the control of the system; the one for control of interactive methods was described earlier. Figure 4.20 shows how the interaction filter works with the variable block in permitting certain classes of interactions.

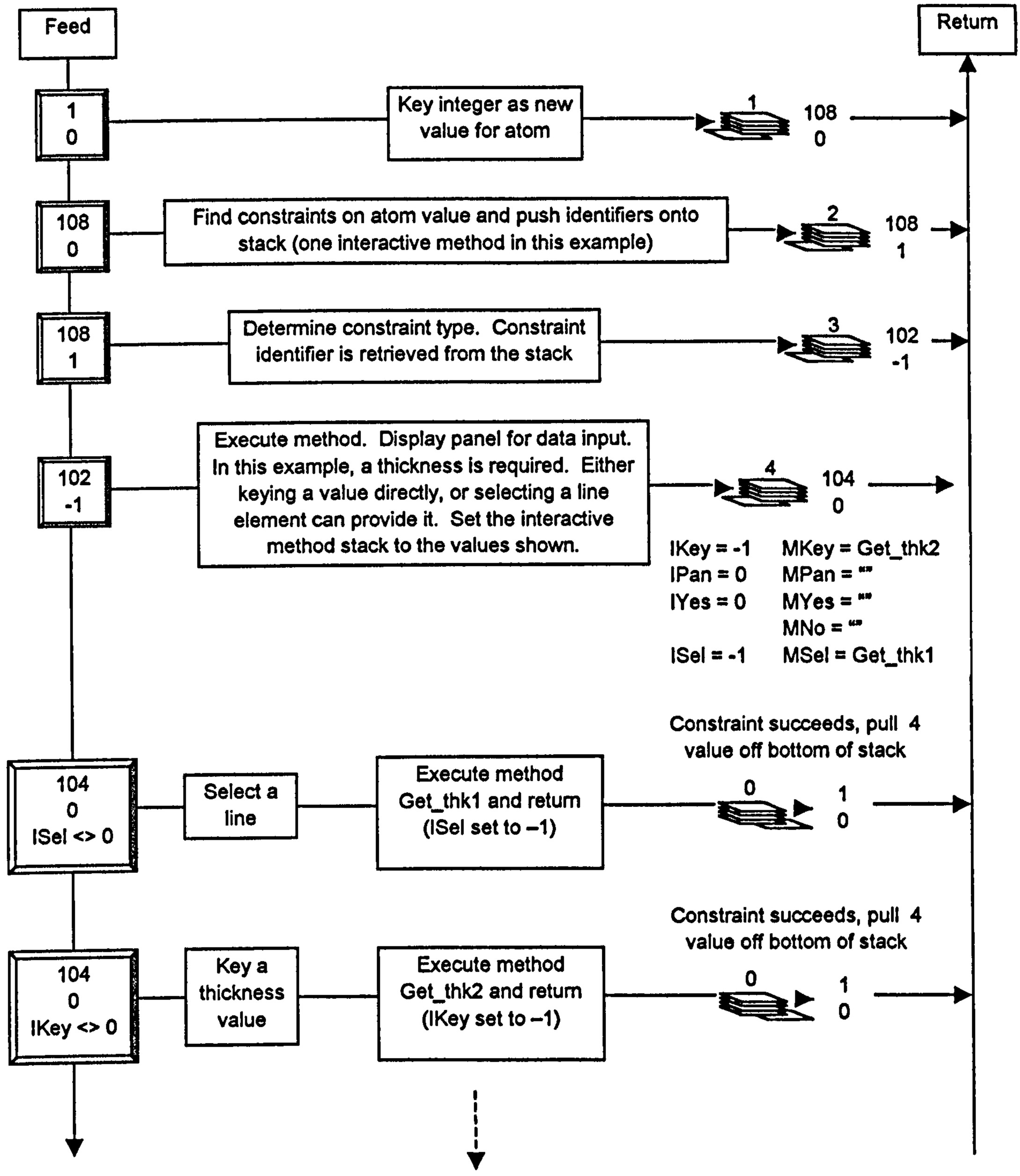


Figure 4.20 - Flow of control and user of variable blocks for interactive methods

The other variable blocks control several aspects of the system and play a crucial role in ensuring that tasks once started are completed. However, many sidetracks were taken from the original task. Details of the variable blocks are given in Table 4.7.

Variable name	Description and *details of compound contents*
Geomlink	Address of geometric element being linked
Curmdat	Method data output currently being mapped
	Method data information
	Proposed value
Atomval	Value to be given to an atom
	Atom address
	Proposed atom value
	Value source type: *0 – user-defined* *1 – mapped atom value* *2 – mapped atom value from another activity*
	Address of mapping atom
Arrayval	Value
	Data type
	Number of values in the array
	Length of the values
	Array of values
Matomval	Atom value stack
Marayval	Stack for atom array values
Ptradd	Structure or activity to be added to pointer atom as a linked item
	Type: *1 – structure* *2 – activity*
	Address of item to be linked
	Structure description block
Ptrdel	Structure or activity to be removed from pointer atom
	Type as for ptradd
	Structure description block
Prepstr	Structure currently in preparation
Pmtatom	Parent atom pointer
	Parent atom address
Strstack	Stack for partially processed structures
	Number of structures on stack
	Structure identifier
	Structure timestamp
	Structure address
Fsdstack	Flow control stack for interactive methods
	Number of control blocks
	Control block contents
Mthstack	Methods currently in progress
	Number of methods (depth of nesting)
	Method information nblock

Table continued on next page

Table continued from previous page	
Variable name	**Description and *details of compound contents***
Atmstack	Atoms to be prepared
	Number of atoms
	Atom addresses
Tststack	Timestamp stack for history record blocks
	Number on stack
	Block start timestamp
Blkstack	Design history record stack
	Primary code
	Secondary code
	Block status
	Level
	Block begin timestamp
Infsdctl	System flow control (described in section 4.4)
Catomaea	Current atom address
Cstruaea	Current structure address
Curacdef	Current activity definition
	Location of activity record file
	Timestamp
Actsrvck	Current Guide product model server
	Communication socket number
	Error number
	Server address in hex format
	Server IP address in dot-separated format
Dbdefsck	Current database server
	Communication socket number
	Error number
	Server address in hex format
	Server IP address in dot-separated format
Clinkadr	Address of element that atom is to be linked to
Fltrsfam	Filter on structure families
	Number of conditions
	Filter condition rules
Fltrdsc	Filter for structure description
	Description filter
	Filter operator (equals / like)
Fltrscrt	Filter on structure creator
	Filter value
	Filter operator (equals ./ like)

Table 4.7 - Guide system variables

4.5 Design history record

The design history record has previously been described as a record of all the transactions that take place to alter the state of a design, together with the reasons and justification for the particular choices made or options selected. It is also important to record under what context the transactions took place, that is, what information was available, what ***function*** the transaction was supporting.

Instances of one atom may be associated with several structures, and each structure may be prepared several times during any activity. To record a change in the value of an atom of that type therefore carries little information. The context, i.e. the structure the atom belongs to and the current activity, must also be recorded before the change in value takes on meaning. The ability of Guide to operate in a recursive fashion makes the specification of the context even more important.

When determining or extracting the knowledge from the design history record, there are questions of what constitutes a transaction, how finely the design should be analysed, what steps should be recorded [2.10]. In the Guide approach, everything is recorded. The analyst can then subsequently determine what segment of the design history record is of use for the application they have in mind. Because the record is not pre-segmented, this leaves a great deal of flexibility to extract several different but overlapping utilities from the one record.

The attributes of the design history record system are that it must be

- Unobtrusive: it is important that the capture of a design history imposes no visible overhead on the user and does not distract them from their main occupation. If they do, the systems are likely to fall into disrepair [1.11]
- Structured,
- Based on a design language [1.9].

Guide records the actions taken by the system chronologically and in context. The format of the record conforms to the model below:

```
Start block 1
        Action 1
        Action 2
        Start block 2
                Action 3
                Action 4
                Action 5
        End block 2
        Action 6
        Start block 3
                Action 7
        End block 3
End block 1
```

The contents of the block and action descriptions are shown in Table 4.8.

Block type	Primary code	Secondary code	Description	Specific data
Prepare structure	50000	0		Structure oid.timestamp
	50001			Atom oid.timestamp
		1	Normal atom	
		2	Child structure atom	
Edit structure	50002			Structure timestamp
Edit atom	50003			Atom timestamp
Set atom value	50004			Atom timestamp
Execute method	50005			Method oid
		1	Structure pre-prepare constraint	
		2	Structure post prepare method	
		3	Structure pre commit constraint	
		4	Structure post commit method	
		5	Method on structure during edit	
		6	Atom default supply method	
		7	Atom constrain on value setting	
		8	Method on atom during edit	
		9	Constraint on method execution	
		10	Method on method input	
Edit method input	50006			Method data oid
Filter method outputs	50007			

Table continued on next page

Table continued from previous page				
Block type	**Primary code**	**Secondary code**	**Description**	**Specific data**
Map method output	50008			Method data oid, value, atom oid.timestamp
Key interaction	50009			
Select interaction	50010			
Map from atom	50011			Atom oid.timestamp, map atom oid.timestamp, activity oid.timestamp
		1	Current activity	
		2	Other activity	
Mapping group creation	50012			
Run method	60000			Method oid
		1-10	As above	
List methods	60001			Parent oid.timestamp (e.g. current atom)
		1-10	As above	
Change atom value	60002			Atom oid.timestamp, value
Change atom status	60003			Atom oid.timestamp, status
Change structure status	60004			Structure oid.timestamp, status
Set method input value	60005		Method data oid, value	
Establish dependency	60006			Atom oid.timestamp, parent atom oid.timestamp, activity oid.timestamp
Set mapping group	60008		Number of items, oid.timestamp of each item	
Break mapping group	60009			Number of items, oid.timestamp of each item
Add item to mapping group	60010			Item oid.timestamp
Link child to atom	60011			Atom oid.timestamp, structure oid.timestamp

Table 4.8 - Design history record entry contents

4.5.1 Recording mechanisms

Other researchers have tried a variety of methods to capture the design history, all of which are more suited to the study of how humans perform design tasks than any viable

method of real-time design history record capture. The systems commonly used, including those described by Stomph-Blessing [4.12], are:

- Real-time data recording. This has been shown to be the most effective way of capturing data. Otherwise, the information decays and so, when a capture is attempted, only a reduced set is available [2.16].

- Post-design interview and retrospection. This suffers two problems: it is unlikely that a project team will consider the work involved to document a design history once the project is complete to be justifiable. The method attempts to turn the interviewee into a researcher, having to express complex ideas in a simple manner. They are most likely to want to continue on to the next project and bring that to fruition and a revenue-generating state as quickly as possible. The other problem is that of loss of memory even after short periods of time of actions and decisions taken and the rationale behind them.

- A forced recording overhead on the designer [4.13]. The justification and rationale for decisions needs to be explicitly recorded as part of the designer's work. The extra burden on the designer and lack of formalised design rationale description makes is improbable that consistent and complete information will be recorded by this method. The significant overhead on the designer's time means that all ways of bypassing the recording system are likely to be sought.

- Observed design and think-aloud protocols. The process is to encourage the designer(s) to articulate their thoughts during their periods of design activity. This is recorded on videotape in some instances [4.14], [2.10]. Several studies have been conducted where contrived or real design sessions have been observed. [3.14]. Most authors view this mechanism not as a method for gathering design history information in industry, but as a technique to assess the scope and requirements of systems which would automatically perform this task.

  The spoken record tends to consist of often arbitrary sentence fragments, which are variable in their information content and quality. They are certainly not exhaustive and complete.

  Fundamental design assumptions have been shown to not be recorded, since they were not raised or discussed at design meetings [2.10], even though they are of great importance, especially when they relate to compliance to standards. Guide does not discriminate in the generation of its design history record: everything is stored, even common or well known and understood knowledge.

Ullman reconstructed histories from videotaped sessions [4.14]. He had it in mind to enable the navigation of the design workspace through his recorded data.

Esterline et al. [3.5] use the technique to determine what states a design goes through and examine links created. This information needs to be recovered from the design session transcript.

- Time-sampling: In this process, the complete state of the design is recorded by an observer at various moments in time. The data collection requirements of this process are extremely onerous and the work of distributed multidisciplinary teams would be impossible to document.

- Knowledge Acquisition Documentation and Structuring This is model-based rationale recording. Knowledge is used to attempt to understand the actions of a designer. Projects on this type of record have generally focused on a well-defined and understood area of engineering and amount to large-scale parametric design.

  Favela et al. [4.15] propose a design rationale capture which in fact turns out to be a restriction on the types of relationships which may be defined. By restricting the user to this set, tailored to a specific design domain, design decisions taken are put in a framework of technical justification. They are, however, finally the expression of the component's functional requirements, rather than an intent beyond the primary intent to meet the specification and produce product pleasing to the customer. The approach is consistent with their view of design, which is one of imposed design methodologies and the discipline they provide as a co-ordinating influence on collaborative design. This idea has some merit, but an environment that supports collaboration will also encourage it.

  Another system to force the description of the rationale during the edit is the AIDEMS system proposed by Thomson and Lu [4.16], where the user needs to provide justification for the link creation as part of the creation of links between entities.

  Garcia et al. also used the videotape method to analyse design rationale and provide support for some aspects of the design process. They used it to, in effect, define the parameters and parameter constraints on a particular closed design domain, that of HVAC systems [2.10].

  Ramesh and Dhar [4.17] developed a system (REpresentation of MAintenance or Process knowledge, REMAP) which allowed design teams to display their deliberations in an issue-based format (graphical IBIS). This was providing a tool immediately useful to a design team, even though it still relied on feeding users with

the information as the design progressed. They developed a set of entities to support this process, based on a preliminary observation of what was used in practice. Guide provides the ability to use precisely those entities used in design and management, without restructuring them to a pre-defined set. Another advantage is that Guide does not pose an overhead on the designer. The Guide system allows the user to proceed against a goal and target, and to refine it until the constraints imposed are fulfilled.

The history understood by Goodwin and Chung [1.20] consists of recording the alternatives for the solution of a problem. They later map their work onto an IBIS methodology, so their history is a series of snapshots of the design at different decision points. In a sense, this captures the rationale of decision making, but not the path leading to that decision stage. The development process is not maintained by IBIS constructs and provides no support for integrating and knitting in of process information.

4.6 Application Programming Interface

Guide provides a full set of application programming interface (API) routines to perform any of the tasks available under the standard product. Some of these routines are necessary in order to allow methods and interactive methods to be written. The routines go further, allowing the preparation of structures, activities, setting of atom values and interrogation of the product model data.

The kernel is available from outside sources. This is important for the extension of the product. Functions can also be imbedded into applications in a monitoring role. For example, an accounts package may invoke the Guide kernel to make a record of the details of invoices sent by the company. This would provide an audit trail and would allow incurred costs to be linked explicitly to design projects. The application might not need full integration with Guide, but the data is still recorded and may then be used elsewhere in the company.

One implication is that though an external piece of software may not allow Guide to invoke it and manipulate data through it, the application's macro language might be able to call Guide functions.

4.7 Architectural précis

Guide has the ability to represent any entity, through its use of structures and their characterising atoms. Derivation and version control are applied to structures, atoms and methods, as well as to the product model data. A rich set of relationships can be established.

Activities support the planning, control and execution of design projects. They allow for the granularity of a project to be decreased by successive recursive invocation of the activity framework, with each level providing a specification to its children.

Methods support the user in the environment in which they work, providing computation and information retrieval facilities for them at the point of need and in a relevant format. They can also be system controlled and applied as constraints. Many system functions are implemented through methods, implemented in the normal fashion. Some of these system methods appear to the user as constraints.

Interactive methods allow the user to influence the direction of execution of the method. Bridges can be built to third party software applications, with varying degrees of integration.

Geometry is managed via a feature-based modeller. The geometry is an attribute of a structure, which may have other, non-geometric attributes.

The Guide manager enables the system administrator to define to Guide the entity and method set used by the company and to maintain it.

A control system allows the users the flexibility they require in performing operations at different levels and in no pre-determined order. This is not a recipe for anarchy, since checks and constraints are applied by the system. Also, as they build links to other systems and dependencies are in turn applied on the data they generate, so their potential sphere of solution narrows.

An API affords access from methods to Guide functions, for example the instantiation of a structure. Guide functions can also be embedded in other applications. This is another method for deploying Guide across the company, if the system cannot be interfaced with Guide, it is possible to make it swallow Guide.

Navigation tools allow the user to browse the product model definitions and the relationships that exist there (plates 17-19). The product model data can also be browsed. The product model data can be browsed through time by navigating the design history record. This is gathered without further overhead on the user.

Guide meets the requirements of Peckham and Maryanski [2.17] for what they term semantic models, namely:

1. Unstructured objects
2. Relationships
   - Abstractions
   - Classifications
   - Generalisation
   - Aggregation
3. Association
4. Network or Hierarchies
5. Derivation / Inheritance
6. Insertion / Deletion / Modification constraints
7. Degree of constraint application
8. Dynamic modelling

These elements and more are included in the Guide specification, and its implementation.

5. Guide Applications for Industry

5.1 Solutions in context

The evolution of Guide was paralleled by its exercise operationally in solving industrial problems. Thus was the validity of the Guide paradigm proved and understood. These were severe tests, for rigorous solutions that would be robust when applied in complex commercial environments were needed.

These implementations forced attention to detail and the expression of the system concepts in concrete examples. They tested the ability of the governing principles to manage real engineering data, processes and constraints in a simpler and more powerful manner than the incumbent mechanisms might. Particularly, they required that the cumbersome and problematic legacy of artificial design methods and constraints should be circumvented. User interfaces, utilities and ergonomics were tested strenuously, often in prejudicial circumstances. The navigation facilities offered by the Guide user interface were greatly strengthened by their exercise in this way.

This chapter describes two disparate applications, one concerned with the complexity of a quantitative design task, the other seeking to solve an equally complex challenge of design management. Its purpose is to identify aspects of those design and business functions that were enhanced through the application of Guide, and not to provide a detailed explanation of the implementation specifics. These are available in Tsiotsias [1.4] and Hopper [5.1].

5.2 Management of engineering change in a multinational company

This application was developed in concert with IBM within its Shared University Research (SUR) programme. The aim was to devise and implement an engineering change (EC) process, compatible with the current product information model, which would overcome problems being experienced under the regime then extant.

The case study illustrates the convoluted progress of an EC engendered by a vendor who was actively involved in the design and manufacture of a new range of monitors. Injection moulding work for the plastic casings had been sub-contracted to the vendor who discovered that a clip provided on the front bezel did not match the equivalent recess on the rear cover. The EC cycle covers the events and related exchanges of information from the time that the problem was recognised and the customer was notified until a resolution was found.

The requirement for this application arose from one of the main objectives of the SUR work. This was to establish consistent and complete product-related information communication protocols between IBM and its vendors. A related objective was to cause the user to pull information from the originator, rather than to wait on a push from the originator. The investigation was of the application of the PDES standard for product information exchange and the value of the implicit modelling formalisms the standard employs. Figure 5-1 shows the project plan for this task.

It was quickly evident that the problems being experienced with vendor communications were only the symptom of complicated and inflexible product information storage and manipulation systems used internally by the company. The effect was to leave the product model incomplete at the stage of its transfer via the PDES protocol. The investigation widened its scope to address the root causes of the initially perceived problem.

The dependencies discovered are represented in Figure 5-2, which shows the layers that underpin the top level data exchange transactions. Specific tasks manipulate the product model data in order to execute functions designed to meet the company's strategic aims. The use of Guide allows the focus to remain at the strategic level - the primary goal is the rapid creation of a highly competitive product, not merely to transmit geometric information to a sub-contractor. Thus, the starting point for the analysis of a particular data transaction should be set at the highest level. In this way, the fundamental and root causes of the problem will be discovered, and effort will not be dissipated in solving repeated, though apparently different, manifestations of a single underlying problem.

In this example of an EC transaction, the primary business directive is to develop a new monitor and maintain a competitive edge. One requirement is to ensure the conformance of the design and manufacturing information in the presence of engineering changes, however they are engendered. Options for part sourcing, including the case of sub-contract manufacture to an agreed and promulgated in-house specification and design must be accommodated. This involves an element of communication with the chosen vendor. The problem is not one of communication alone, but of employing the sub-contracted manufacturing capacity effectively and, thereby, meet the business objective. The communication function is subservient to this higher level objective.

	Phase 1 Requirements	Phase 2 Examination of constraints and generic solutions	Phase 3 Resolutions or proposed techniques	Phase 4 Concept generation	Phase 5 Software development	Phase 6 Prototype	Phase 7 Roll out
Task	Communication of product data →	PDES /STEP →	Draft standard →	Define communication procedures →	Provide software solution →	Deliver prototype →	Test and release

Figure 5-1 - Initial product data exchange programme schedule

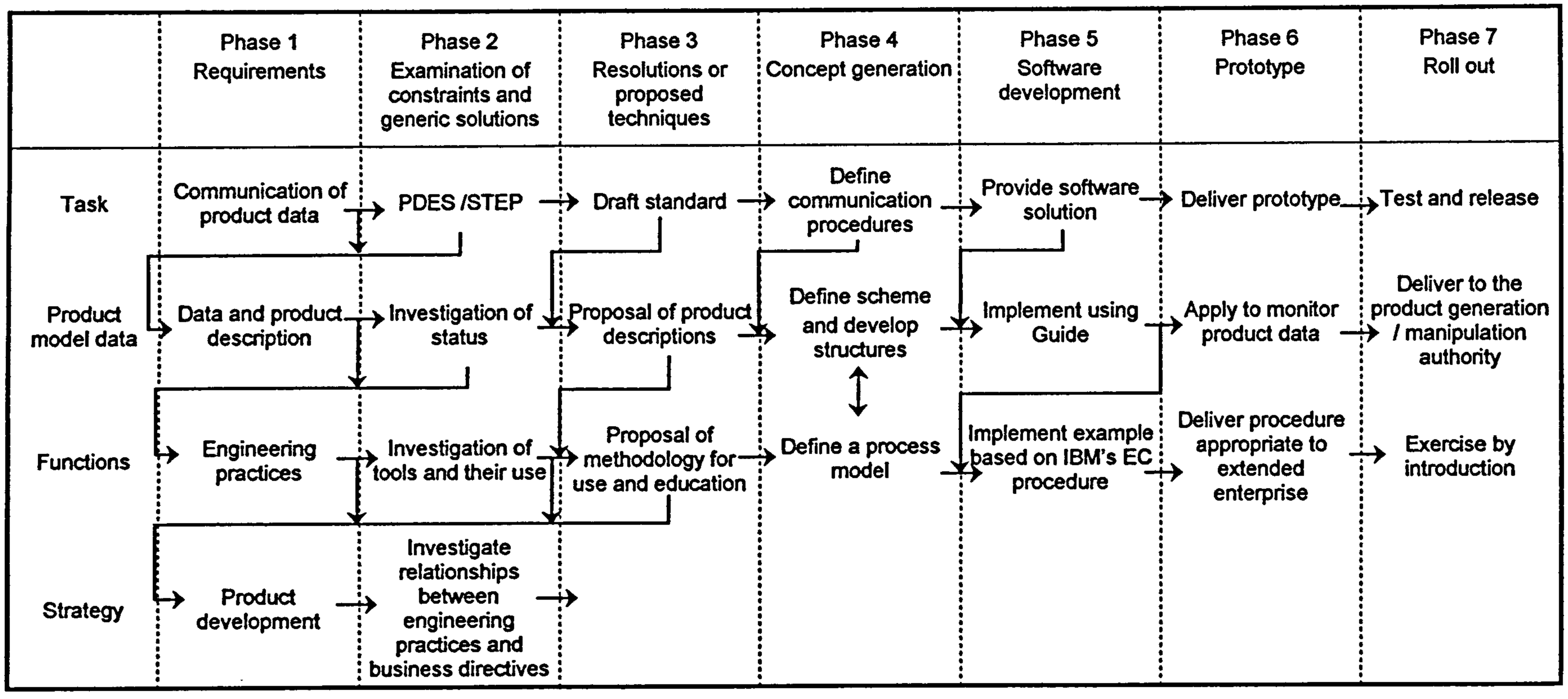


Figure 5-2 - Expanded product data exchange project schedule

5.2.1 Guide schema for Engineering change transactions

The solution to the data transfer requirements reflects the application of a top-down approach. Rather than provide a custom designed piece of software that provided exactly the requirements at the task level, a complete product model based solution was offered. The sequence of interactions with the system is depicted in Figure 5-3. The benefits derived from the Guide implementation are summarised below.

The fundamental Guide approach is taken despite the assertion by Weissflog [5.2] that the amount invested in CAD and other systems would preclude subsequent radical changes to systems and that advances would be of an evolutionary nature. Rather, they need to be revolutionary.

5.2.1.1 Design authority transfer and communication formalism

The transfer of design responsibility occurs through the initiation of activities. Communication is made unambiguous by imposition of the condition that every activity has associated with it an authorised requestor and a responsible actor. This ensures that actions are traceable and commissions are completed.

The need to transfer product model data communication is a concomitant of every decision that transfers the responsibility for design between actors. Guide offers the facility to define activities in the way described and to signal their successful completion. Then, the responsibility for notification of an engineering change, should one be required by the customer or, otherwise, be noticed or required by the vendor, is allocated unequivocally. The initiator of the activity is forced to make its definition complete and rigorous. However, this rigour does not invoke spurious control activity: no communication occurs unless a change is identified.

5.2.1.2 Compatibility with legacy systems and product model evolution

In this case study, the application of Guide functions is to fundamental issues affecting major aspects of IBM engineering practice. It is not a holistic solution, however, and care must be taken that other functions of the business can continue unimpeded. One such example affects the Development and Product Record System (***DPRS***), an EC and BoM information repository. DPRS is not necessary for design control, yet manufacturing relies on its contents as a statement of the requirements falling upon it. Maintenance of DPRS is assumed by constraints on the EC activity, which update it with the modified parts in the EC package.

This support of a phased implementation is a vital attribute of Guide. It also demonstrates that engineering practices are not subservient to the tools used; the product model can evolve independently of the tools employed in its construction.

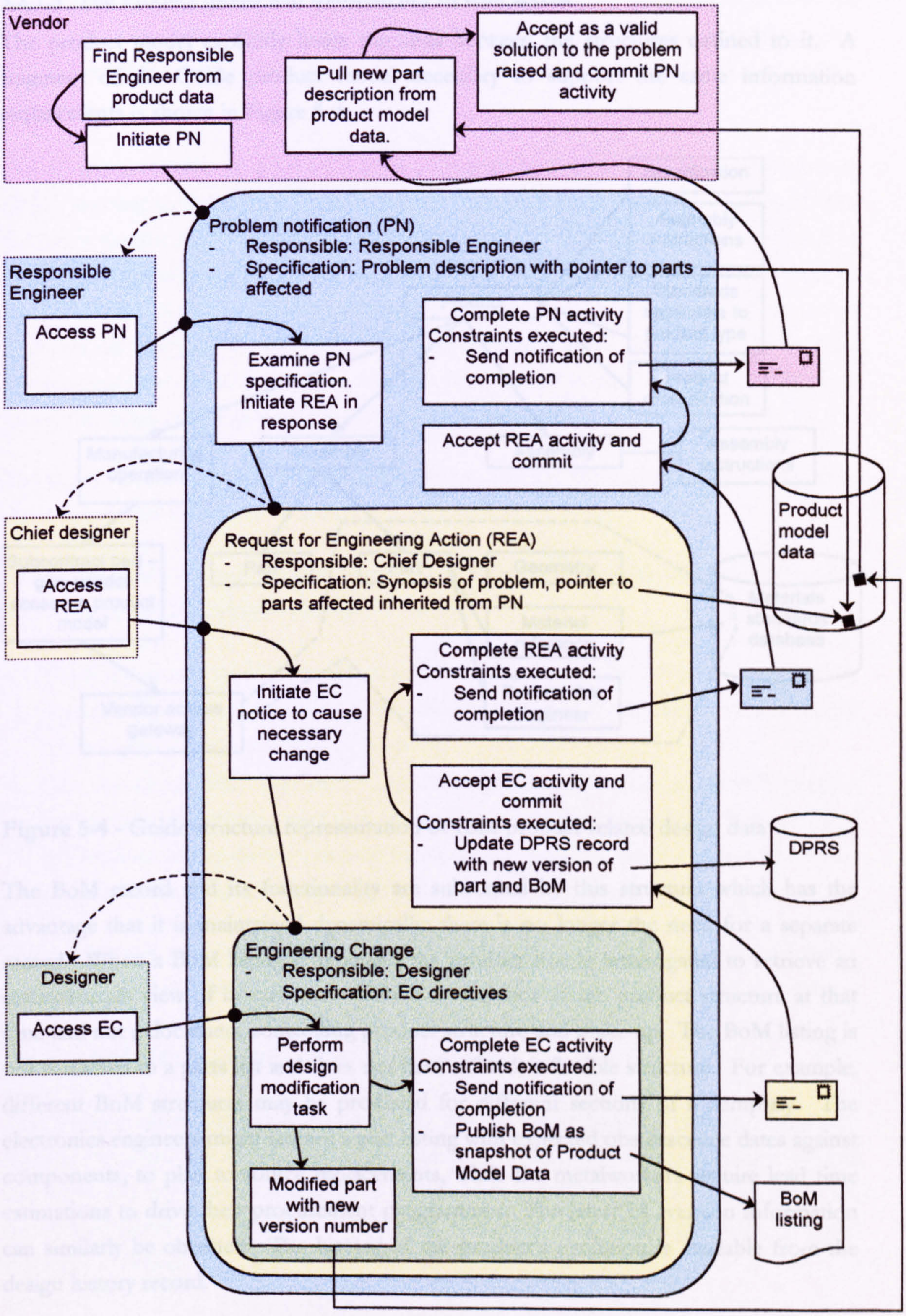


Figure 5-3 - EC procedure implemented using Guide supported product model and services

5.2.1.3 The Product model and transparency of data access

The product model explicitly holds the links between the structures defined to it. A fragment of the Guide product model necessary to support the same information requirements is shown in Figure 5-4.

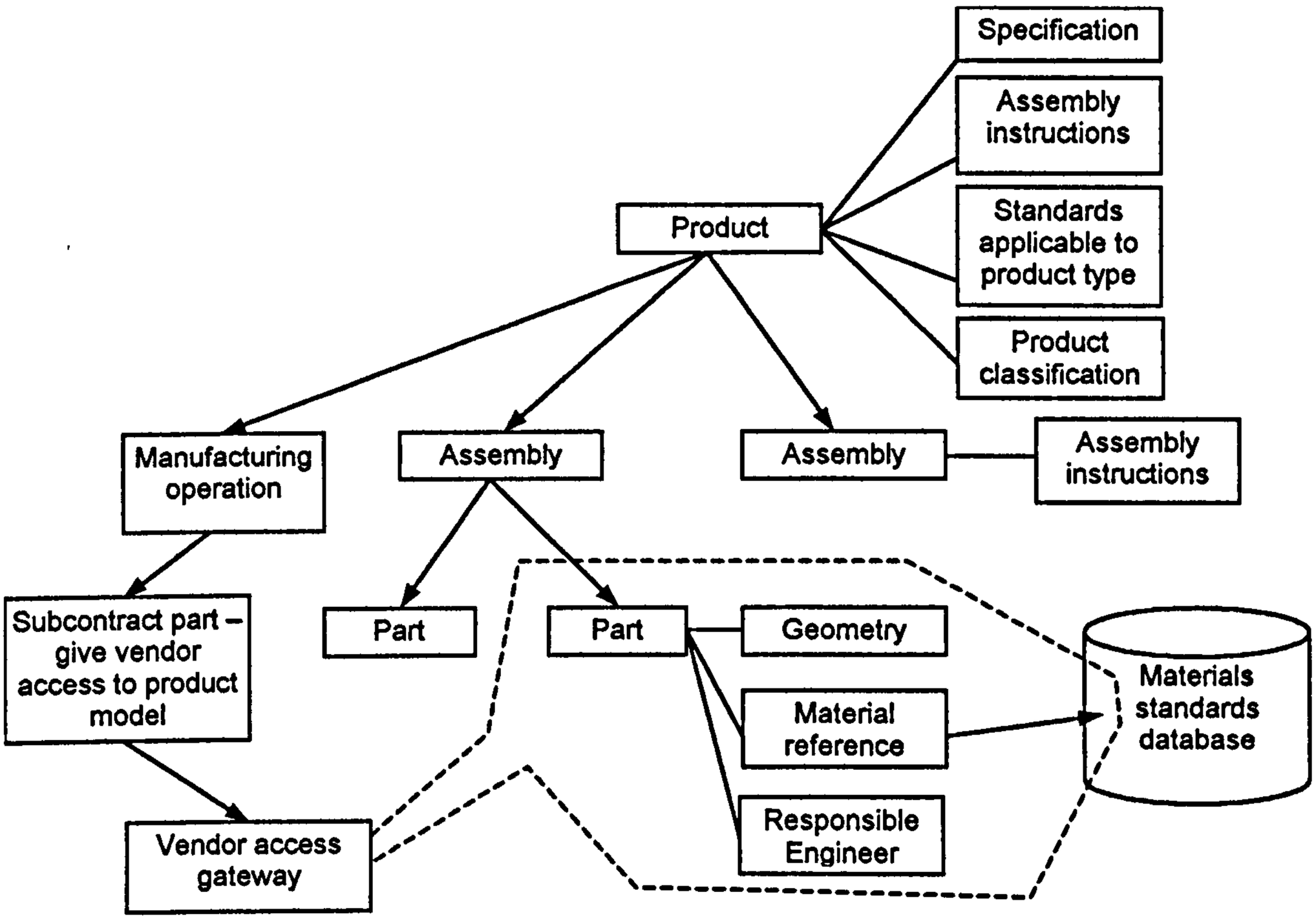


Figure 5-4 - Guide structure representation of IBM product-related design data

The BoM record and its functionality are subsumed by this structure which has the advantage that it is maintained dynamically; there is no longer the need for a separate record. When a BoM listing is required, the product tree is interrogated to retrieve an instantaneous view of its contents. This is a snapshot of the product structure at that time and not a document controlling product structure and make-up. The BoM listing is not restricted to a parts list and does not have a fixed, inflexible structure. For example, different BoM structures may be produced for different sections of a company. The electronics engineers might require a part listing with expected obsolescence dates against components, to plan to source replacements, while the metalworkers require lead time estimations to drive their procurement programmes. The latest EC version information can similarly be obtained. The history of the product's evolution is available from the design history record.

Each assembly, sub-assembly and part is represented by an activity, the contents of which describe its geometrical characteristics through the structure instances held therein.

The activity header contains version information on its content; the revision level of the part is therefore always available. Applicable standards are associated with the part through explicit relationships; implicit links using the part number as an index are not necessary.

The creation of links to the part - links to materials, Product Information Exchange (***PIE***), number or standards information - is managed through methods associated with the part's specification structure, which assists the user to find the correct reference. For example, it is easier for engineers to create a new PIE code and description than to search for the closest description match for the part they are designing. This action degrades the utility of the PIE database. A method can assist the engineer to find the correct code from those available and restrict the creation of new codes by making them justify the creation of a new code and take responsibility for it.

The activity header contains version information; if an activity contains a part definition, the revision level of the part is always available.

The initial goal- viz. information transfer - is supported through the product model, which brings the following advantages to the task:

- There is a record of a communication sent to the vendor in the form of an activity. A notification is sent to the company once the activity is complete and the vendor has achieved the task assigned to them.
- The activity includes a pointer to the relevant section of the company's product model data. The vendor may then access this information directly from the body of the product model data. There is therefore no need to create a copy of the part information, with all the dangers inherent to the maintenance of an isolated copy.

The vendor can also pass a reference to the part(s) affected in the problem notification. There is, therefore, no need to pass copies with all the associated risks of asynchronism.

5.2.2 IBM Product information structure

The solution offered above is in contrast to the IBM information structure, that which led to the initial problem. IBM distributes product information amongst several locations and in different repositories (Figure 5-5). Links between related items are maintained principally through a product part number and EC level- they are not stored explicitly anywhere.

Geometric information is held in a Catia model file repository, accessed through a Catia Data Manager (CDM) installation, which links to ERE, a central repository in Germany.

The drawings are annotated with some assembly instructions, manufacturing and material specifications and references to appropriate standards. These are held as textual notes on the drawing.

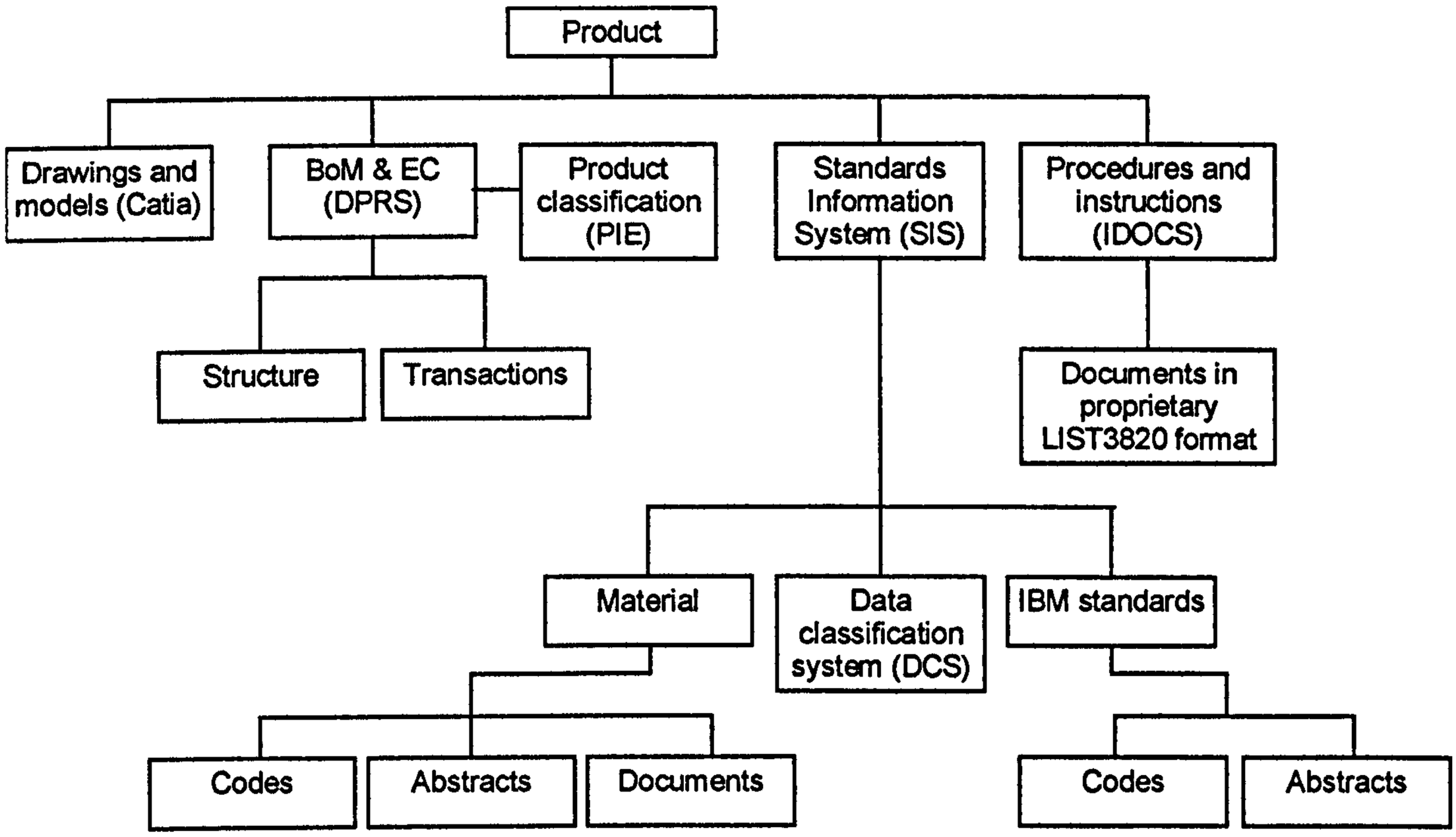


Figure 5-5 - IBM product-related data storage schema

The Development/Production Record System (DPRS) holds information on the Bill of Materials structure. This contains a record of the transactions effected as the BOM structure for a product evolves, and includes details of Engineering changes. It tracks those parts that are changed, added and removed from the product. The information content of the transactions is heavily coded and it requires considerable experience and comprehensive documentation to understand the system. A Product Data Analyst (PDA) is responsible for coding the transactions and insuring the integrity of the system.

Each part in the DPRS record is assigned a PIE number. This is a group technology classification code, used to reduce re-design effort by enabling a search of the PIE database for products that approximate to the specification of the intended new part.

The SIS holds a variety of documents, coded according to an internal classification system. A standards librarian assists in the location and recovery of required information. Other standards and procedures are held in IDOCS, a document management system. These include product assembly instructions complementing those on the Catia drawings.

Communication with the vendor requires for a package of information to be assembled from the relevant data repositories and sent to them. Manual collation is necessary, often using a common part number to retrieve related information from different sources. Once the information package has been assembled and sent, it is no longer live; data changes at the customer location which affect the sub-contracted products will not automatically be relayed. If the information package is not complete, the vendor has no way of assessing what information is lacking, or a means for its recovery.

The problems this product model data structure poses are revealed in the case study

5.2.3 Case study – comparison of approaches

The IBM sequence of events for the example scenario, from problem identification to full resolution, is shown in Figure 5-6. While the end result of the process is the same as that in the Guide system, the steps taken and implementation approach are different. A comparison with the Guide solution on key points is held in Table 5.1.

Aspect	IBM procedure	Guide support
Vendor to IBM	No fixed format, often informal Data package contents unpredictable Responsible engineer established by PDA Reference to problem part through transmission of a copy	Mechanism provided, with controls on format, content and completeness Identity of the responsible engineer is retrieved from the product model data. Activity includes links to parts affected, not just references or descriptions.
Responsible engineer to Designer	Formal route seldom taken	Formalised data structure from vendor can be forwarded, with comments.
Engineering change transaction	Changes made in isolation from rest of product information	Changes made to parts within the context of product model data. Normal constraints apply, and notification will be given of further changes arising from action.
Responsible engineer to vendor	Entire EC package and drawings sent	Notification sent. Vendor can then pull required information from system (Figure)
DPRS transaction stream	Executed manually by PDA, after EC has been approved. There is a risk that the correct information is not relayed to the PDA at the appropriate time	Method on closure of EC – automatically executed. There is therefore no delay between the approval of the EC and the DPRS transaction. The PDA does not need to be involved at this step.
Data structure	Product model distributed among several repositories, linked implicitly through common part numbers of via metadata	Consolidated product model with explicit relationships. Information may be distributed among different repositories, but there is no requirement for the user to be aware of this.

Table continued on next page

Table continued from previous page		
Aspect	**IBM procedure**	**Guide support**
Design authority	Responsible person not clearly defined in the system, relies on external management.	Direct mapping of responsibility over the design to activities.
Audit	Simple EC history in DPRS	Complete audit trail and record of transactions, including communications through the design history record.
PDA	Requires high level of skill to manage DPRS information – catching up with the design	Free to maintain and develop product model – enabling the design.

Table 5.1 - Comparison of existing and Guide-enabled change process steps

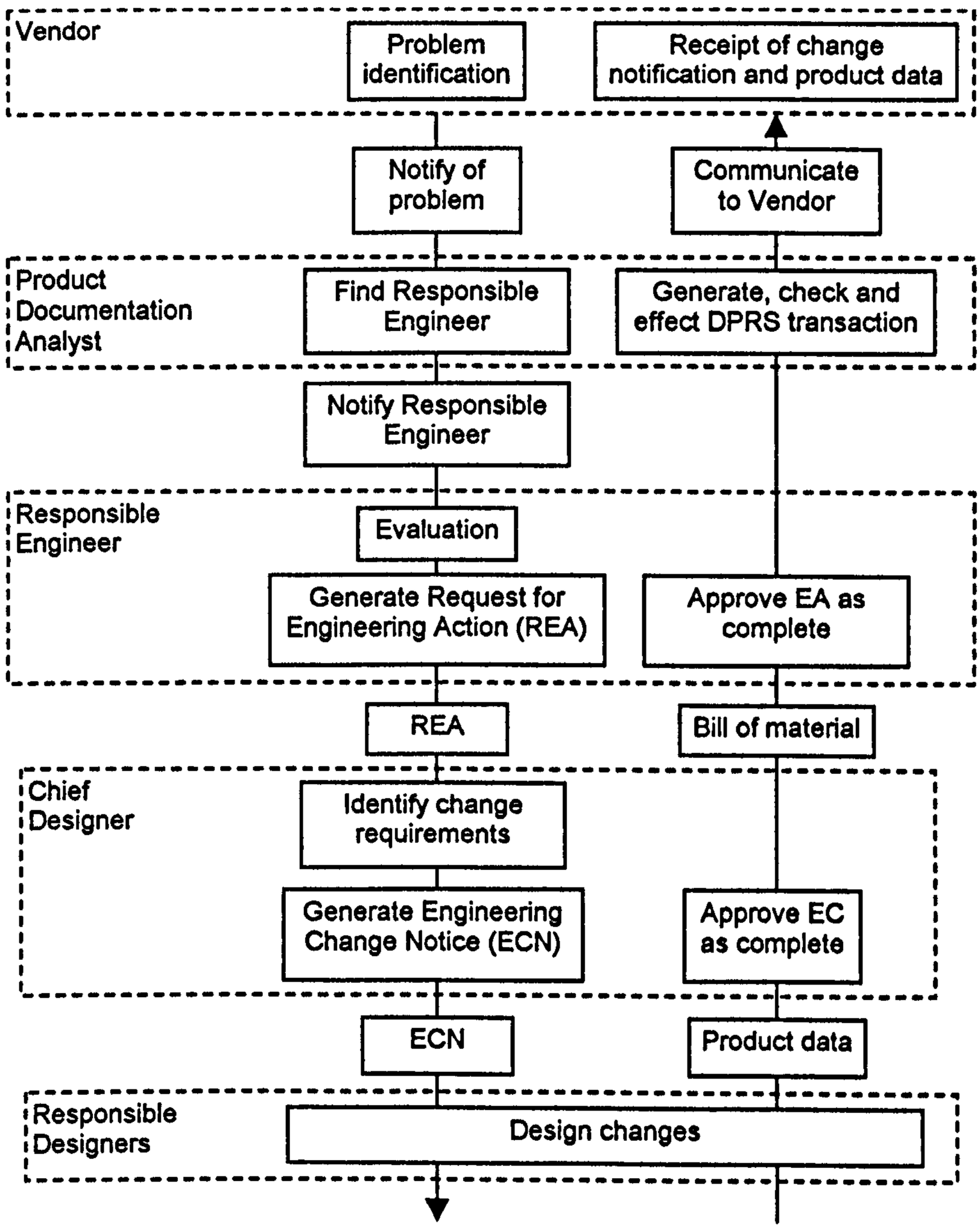


Figure 5-6 - IBM sequence for vendor-initiated product change

Guide offers benefits over the current system at every level. The company's business aims are better put into practice and supported with appropriate tools at the task level.

5.2.4 Wider implications of the work done under the IBM project

The investigation of a communication task led to the identification of layers of contributory factors. A focus on the excellent performance of a function, rather than a task, would avoid many of them.

5.2.4.1 Origin of product model

This task-centric focus is a child of the IBM structure for information service provision. Various departments are responsible for the efficient running of one aspect of the IBM software environment. Their goal is then to implement a solution which can be used by IBM worldwide and which is easy to administer. It is not to promote the business through improved functionality. Any attempt at this is frustrated by the number of such departments, each with their own imperative. Coordination to achieve a better whole is difficult.

Often, in an attempt to find a data representation solution that is universally applicable, the lowest common denominator is adopted. A desire to keep a clean data structure means that data from vendors is viewed with suspicion, which attitude is hardly conducive to close relationships with the vendors.

The opposite is true when software is to be selected. In order to meet the majority of requirements, the highest level of software is designated as the standard. A single tool is preferred, for example Catia is designated the 3D CAD tool which for many applications is severely under-utilised. This makes CAD information easy to handle and exchange. A common set of software tools provides for ease of administration, data exchange and intercommunication. It ignores however local technical and cultural needs, precludes the use of the best and most appropriate software solution for a range of widely differing engineering problems. It will rarely produce the most cost-effective solution. The use of Guide enables a company to deploy different tools operating on a standard set of information repositories: the product model is fixed, the application software tools are not.

Another problem with global software implementations is that they create inertia. Because of its complexity and the large volumes of data held, any changes to the system constitute a major project, requiring extensive consultation and a long planning phase. They tend therefore to lag several years at least behind best practice. Changes requested of the system from different locations may also be in direct conflict, one with the other

and the solution decided on centrally and imposed across all sites may be universally unpopular.

5.2.4.2 Information structure problems

The complex web of information repositories and consequent mass of metadata needed to manage them are justifiable when considered singly and in isolation. For example, the PIE system is a sensible effort aimed at reducing duplicated design effort. That they are individually justifiable also causes problems. The development of a point solution to address one particular problem independently of its impact on the rest of the system is then rendered permissible.

The rigidity of such centrally imposed systems was evident. As mentioned earlier, manufacturing uses DPRS for its parts information. This information is only available once the part is released and not earlier in the design cycle, which would allow them to make outline plans and preparations based on preliminary design information. To meet this need and promote shorter lead times, a system called Fasttrack was implemented to by-pass DPRS change control during design. The product information is patched into DPRS once the design is finished.

Instead of procedures integrated in the software utilities, they are held externally and have to be interpreted and applied to the problem in hand. One of the results of the programme was the identification of the high workload and possibility of errors arising due to the highly complex nature of the product data management systems and the low assistance offered by the software systems intended to perform exactly that function.

5.3 Rolls Royce Ceramics Design System

The Ceramics Design System (CDS) is a methodology developed for the design of ceramic reinforced ceramic components for aircraft engines, applicable to any fibre-reinforced material.

Because of the recent development of these materials, no established design methods or guidelines exist to assist engineers. Engineers require a system to help them manage the different aspects and factors concerned in the design of ceramic composites. It needs to lend enough structure to the process, but be flexible in the way in which it is used. Furthermore, the design science for ceramic composites will mature and, as it does, the design system must be able to accommodate the new developments, while retaining access to previous data.

Guide was used as the enabling technology for the CDS and sits at the core of the system. It exerts its influence at several levels: detailed component structure, design operations and design project management. Each of these is examined below.

5.3.1 Detailed component structure

The challenge is to find a method of representing the different attributes that make up a part. The material in non-homogeneous and anisotropic and has matrix and fibre components in varying quantities. The fibres are oriented in three dimensions and with different fibre densities. The physical properties of the part will be affected by all of these interacting factors.

The model identifies different functions and attributes of the component and maps them to the geometry model of the object. A base geometry is defined, the collection of sub-areas of the part. The constitutive elements of the base geometry are:

- Architectural regions. These are areas that have a common characteristic. Primarily they are designed to represent areas in which the plies of the fibre preform are held by the matrix. A mapping exists between an architectural region and a 3D solid primitive in the chosen CAD modeller or geometry representation engine.
- Base material (matrix).
- Architectural tags, which indicate fibre densities and directions in the local area. These are represented by a series of axis systems, aligned to the principal fibre orientation. This provides a method for ready visualisation of the fibre lay-up
- Materials descriptions, to give the properties of the matrix and fibre materials.

The relationship of the elements and their representation is shown in Figure 5-7.

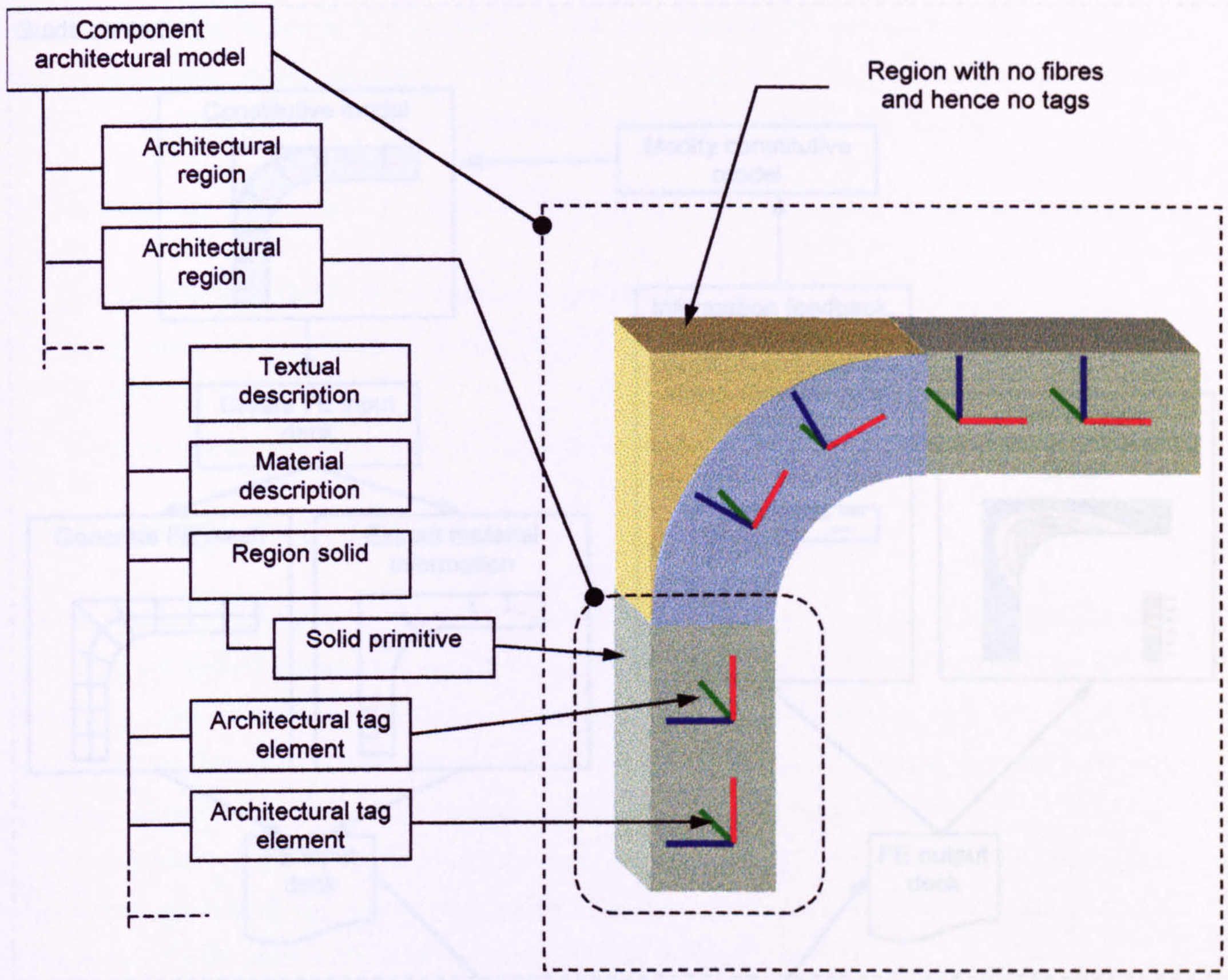


Figure 5-7 - Architectural schema and physical representation of ceramic composite parts.

This aggregation of properties for a part cannot be modelled using conventional tools. While several CAD systems provide the ability to associate properties including material properties with a part, they do not support variations in the property throughout the mass of the body. So, a material property for a homogenous isotropic component can be defined, but this falls far short of the detail required for this application. Furthermore, visualisation techniques do not exist for representing fibre layup properties.

5.3.2 Design operations

The design system employs predominantly an architectural model and FE analysis to refine an initial design. The main interface is the CAD modeller, which holds representations of the constitutive model. Guide methods are used to prepare and export the information held in the model for analysis by the FE programme, and to analyse the results. The sequence of events for a cycle of the analysis loop is shown in Figure 5-8

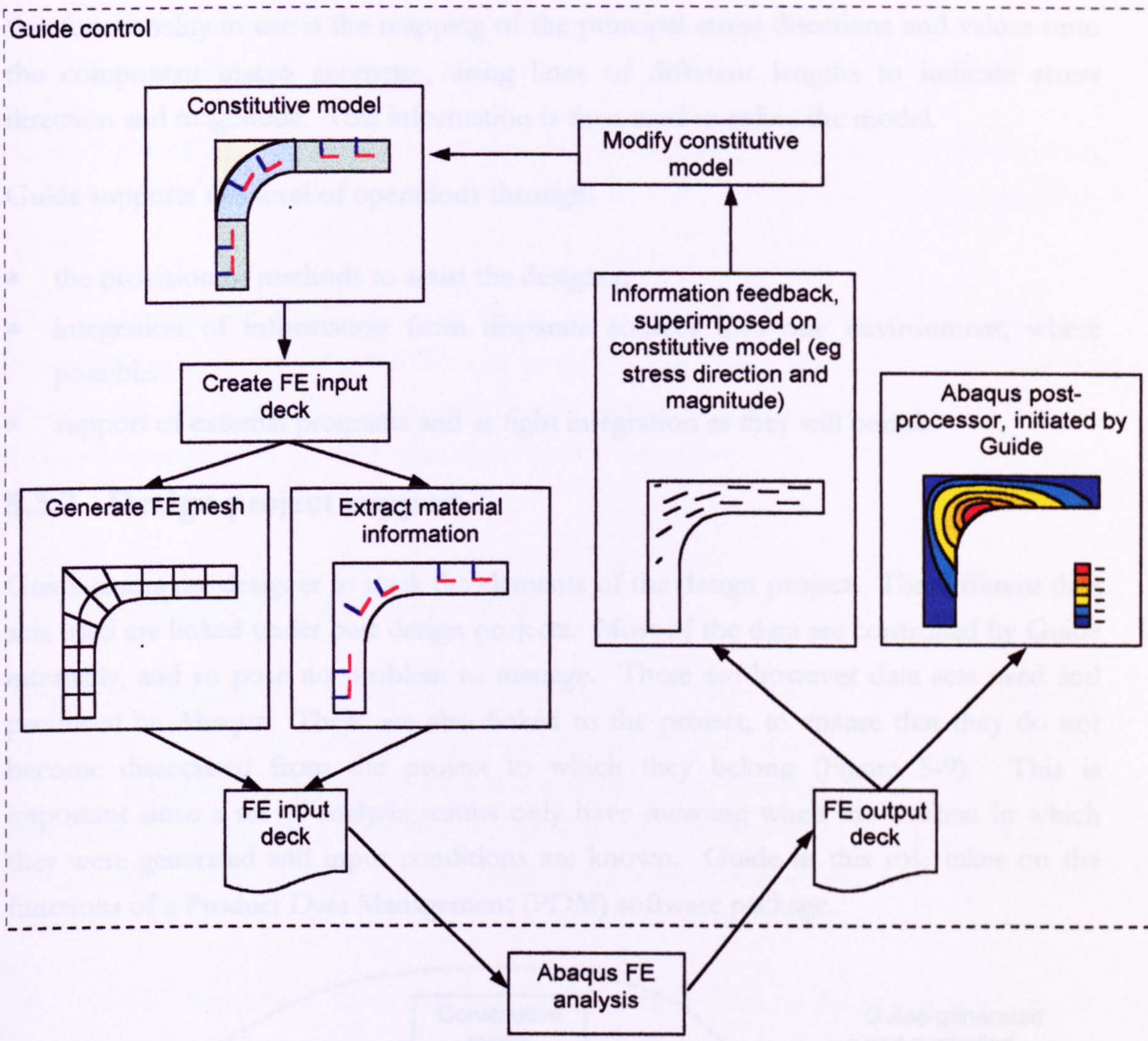


Figure 5-8 - Guide methods used to obtain and analyse FE data

The first step of the iteration is the creation of the Abaqus Finite Element solver input file. This comprises of two parts: the FE mesh, generated by a Guide method and the local material description. The relationship between the two elements is extracted from the constitutive model and does not require regeneration.

There follows the use of the input deck for analysis, a process not controlled by Guide. If Abaqus had a callable API, Guide could invoke the operation of this software. As it is, the input deck is prepared directly from the constitutive model to make the external program invocation as close to pushing a single button as possible. An output file is produced, the result of the FE analysis work.

Guide can invoke the Abaqus post-processor to display the results of the analysis. It can also read the FE output deck and create in the Guide geometric workspace feedback information that is superimposed on the part geometric representation. An example of

this functionality in use is the mapping of the principal stress directions and values onto the component macro geometry, using lines of different lengths to indicate stress direction and magnitude. This information is then used to refine the model.

Guide supports this level of operations through:

- the provision of methods to assist the designer,
- integration of information from disparate sources into one environment, where possible.
- support of external programs and as tight integration as they will permit.

5.3.3 Design project support

Guide assists the designer to track the elements of the design project. The different data sets used are linked under part design projects. Most of the data are controlled by Guide internally, and so pose no problem to manage. There are however data sets used and produced by Abaqus. These are also linked to the project, to ensure that they do not become dissociated from the project to which they belong (Figure 5-9). This is important since a set of analysis results only have meaning when the context in which they were generated and input conditions are known. Guide in this role takes on the functions of a Product Data Management (PDM) software package.

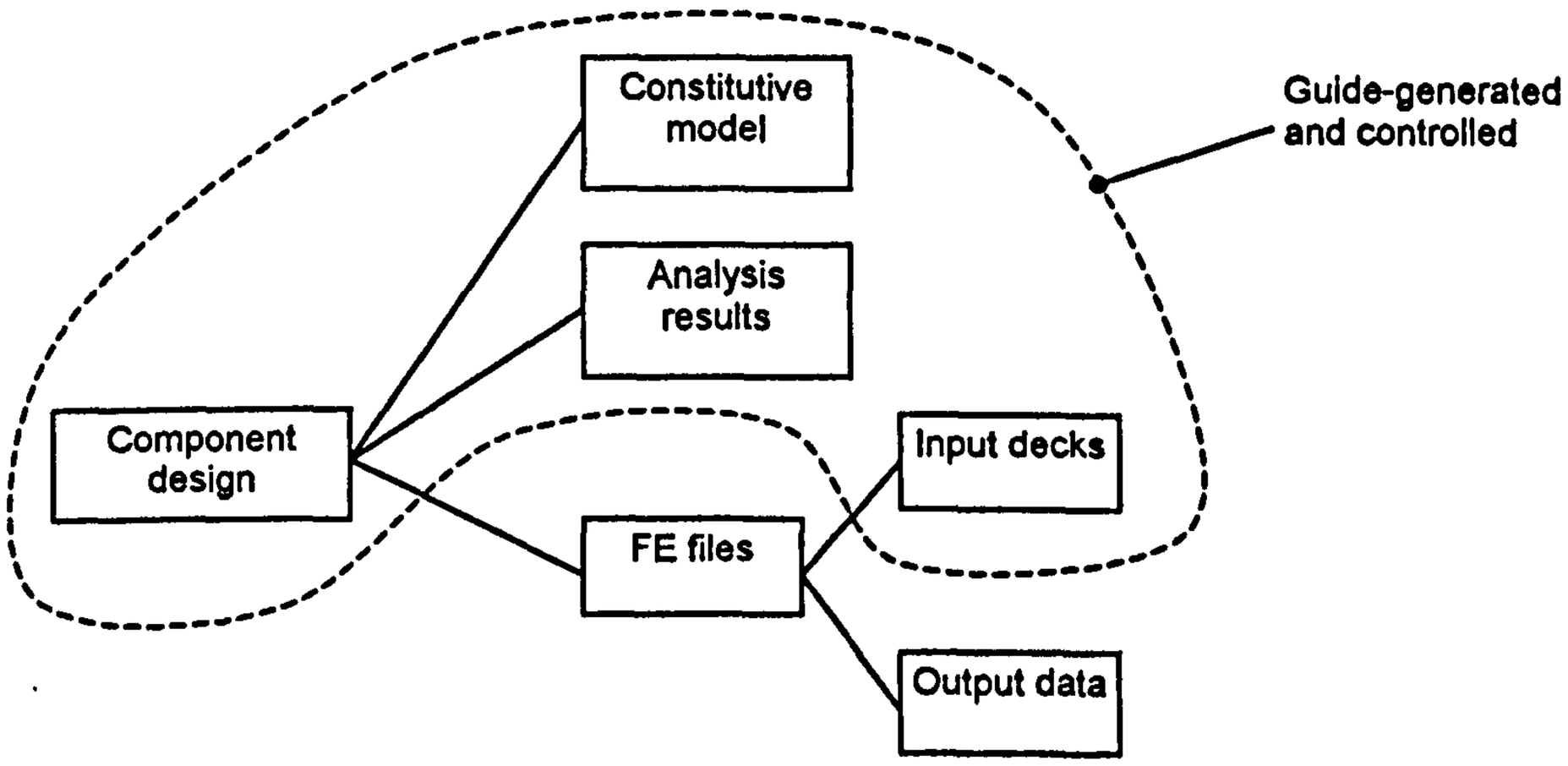


Figure 5-9 - Design data links maintained by Guide

A methodology for the design of reinforced ceramic components exists as a result of this work. The design process is illustrated in Figure 5-10.

The system does not predicate a path through the design space, but identifies possible design states through which the designer can pass, if necessary. Refinement loops can be

created to progress the design towards a competent solution. It is also possible to pursue activities in parallel and collaboratively.

As the body of knowledge on how to design with ceramic composite materials expands, so new entities and methods can be defined to the CDS and are then instantly available for use.

The applicability of Guide was also shown across platforms. The CDS in its latest incarnation interfaces with CADDS5 and runs on a Sun workstation. The definitions for the entities are generated using the Guide manager, which uses Catia services to construct its user interface and runs on an IBM workstation.

5.4 Summary of applications

Guide has proven its utility in real commercial situations of major engineering companies. In the first case study, it is used to reform and support core engineering and business management processes, adopting existing data repositories and keeping sound practice while replacing and eliminating many procedural overheads and impediments to good design. This resulted in the ability to re-deploy staff away from indirect cost activities.

The second case study is an explanation of a design methodology created for an emerging material technology, for which no such formalisms or experience existed. It represents complex and abstract entities, while providing strong links to tangible geometry. Its ability to integrate with third party software while retaining control of the design process and data is also shown. The design information was able to be constructed in a novel and unique fashion, through the flexible links and relationships types available within Guide and the powerful entity modelling abilities. A mature and exhaustive design method resulted from the work, in a short period of time.

The system developed is not, for all that, a standalone application. The entities created as part of that product model are available across the extent of the company and can establish links in either direction on the tree with other product model data.

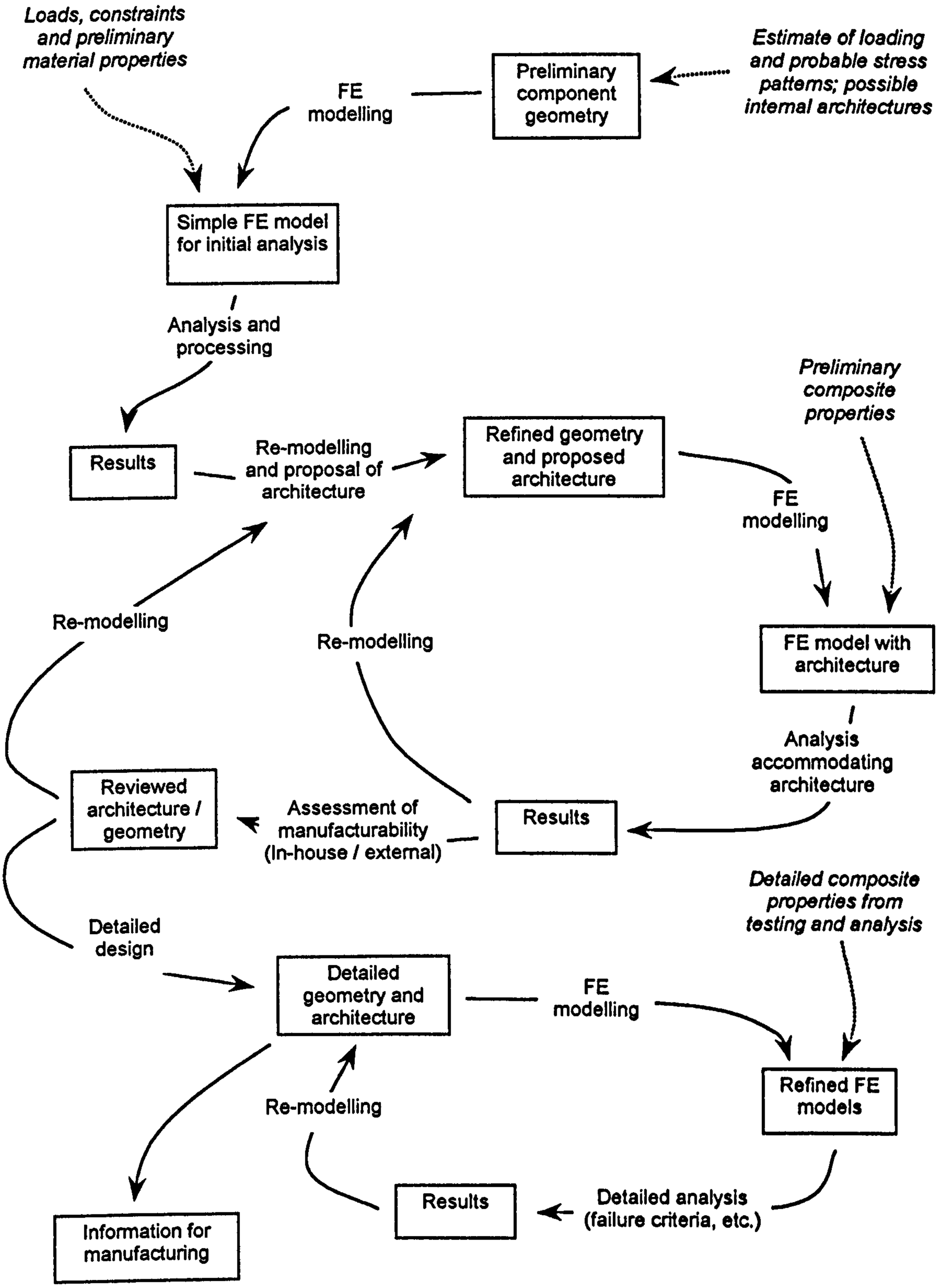


Figure 5-10 - Design process for composite component design.

6. Further work and developments

6.1 System development

The exercise of Guide across a wide range of businesses and function is now essential to discover in depth the benefits it can bring in different applications. Like lasers, the investigation of its application is now as important a research topic as the technology itself. It will allow for the development of implementation strategies appropriate to different types of application.

A standard set of methods, entities and relationships will be useful for any company implementing Guide. This starter set can then be evolved to sustain a product model tailored to the company's needs. The source for such a starter set is twofold: firstly, the PDES standard entities can be offered as Guide structures; secondly, the application of Guide in varied circumstances will result in the development of product model constructs of general utility.

A range of support tools for the user are still to be developed:

- A utility for the assisted extraction of knowledge from the design history record. One of the envisaged functions of the utility is an ability to trawl the body of design history knowledge as it evolves, searching for common approaches taken to different problems. As research on design rationale develops, Guide is in a strong position to capitalise on benefits to arising from this technique and augment the information available from the design history record.
- Utilities for the easy and rapid creation of methods. A modular, graphical user interface would allow new methods to be built from building blocks, using a drag and drop type user interface.
- The various aspects of the audit function need to be supported with methods the better to perform commonly used audit tasks.

6.2 Exploitation of new environments

6.2.1 Hardware

The advantages gained from faster, more affordable computing power and larger capacity, cheaper storage devices are obvious for the use of Guide, particularly in large enterprises.

Portable computing is becoming more effective, in the form of notebook computers and Personal Digital Assistants (PDAs). A PDA able to access a Guide server would have many applications beyond those able to be done from a desk. Guide makes the company's information and processes available to the user at a single point - mobile computing would liberate that access point from the desktop. For example, a maintenance crew on a ship or oilrig, no two of which are identical, would benefit from the product model data and modification history to their vessel. Ordering spare parts would also be assisted: the required section of product model data which contains the full set of manufacturing instructions, relevant standards and billing information could be sent to a manufacturer.

6.2.2 Software

In the period during which Guide was constructed, the software programming environment has evolved considerably. A port of the kernel would offer advantages in code size reduction and portability. Internet-oriented languages and communication protocols would assist in adoption of Guide across distributed sites and locations. In addition, languages such as Java that enable code to be used on any computational platform will mitigate the general problems of portability that affect co-operation across organisations.

6.3 Design language

A commonly agreed language needs to be defined for design and its parameters. Different authors use widely differing nomenclatures for the entities and concepts defined in this publication. These may reflect the authors' aspirations and research direction rather than progress within an agreed domain and framework. This confuses what has actually been attempted and achieved by different groups of researchers. An agreed set of definitions is required to describe the elements of business processes and engineering design, particularly those which happen at a strategic or conceptual level, where they are less well defined and understood. Additionally, the scope of any term used is important. The concurrent processes that occur in a designer's mind do not impute the company with enterprise-wide concurrent engineering.

7. Closure

7.1 Conclusions

Guide has been demonstrated supporting business processes at all levels and across departments through a single set of knowledge constructs known as the product model. This comprises definitions of entities, relationships and methods. Its exercise performing business processes results in:

- Product model data, the instances of product model constructs, which describe the object of the creative business process (for example a description of the artefact resultant from a design process describe)
- A design history record, which captures and stores all interactions with the system for re-use by subsequent processes. The record is a chronicle of the product development including methods engaged and values attributed to entity attributes.

The knowledge structures which support the product model are complex, but their use simplifies considerably many aspects of business. This simplification is the result of investing in a complete and competent product model, obviating the need for extraneous information and the extra control overhead required. The simplicity refers only to an ease of use; highly complex information structures and processes are supported, but the level of representational complexity required is not imposed on the user community.

The product model is not static; version control mechanisms and management tools allow it to evolve in a controlled manner. Management of the product model is not an extra burden placed on the business. Staff are already employed in the attempt to maintain links between different knowledge and data repositories. This effort can be re-directed to sustain the product model and so invest in the company's expertise.

Use of the product model as the foundation for business processes contrasts with the current situation of departmental functions optimised without regard to their effect on the wider business. Different functions within the company are supported by a filtered view of the product model and product model data. This supports the function without dissociation of the company's data and knowledge. Best practice can be planned and developed from a process requirement specification and not predicated by the capabilities of the tools used and the structure of the company; departmental boundaries do not exist in the corpus of the product model.

Design operations are supported by a rich set of enabling methods that provide computation, communication and information retrieval services to the user at the point of need. Constraints are method applied in real time to provide a boundary to the design

and ensure its eventual validity. Methods are defined independently of the entities on which they act, and can be used wherever appropriate and without restriction.

Relationships in both the product model and product model data are expressed explicitly. Both models are therefore semantically complete and do not require metadata further to define the topology and influences among design entities or instances. This removes the administrative overhead of maintaining a metadata layer and its synchronisation with the underlying data. The relationships can be of any type, from simple parent-child to complex algorithms involving several entities.

The effect of a change to the product model data determined by its cascade down the dependency relationships to affected entities dependent on the changed value. This can be done with partially defined structures to carry out what-if analyses and seek a set of values that simultaneously satisfy all of the constraints applied to the related entities.

Guide activities enable project management functions and the co-ordination of child activities to produce a solution that satisfies the stated specification. Authority over and responsibility for parts of the product model data are devolved unambiguously to the actors who contribute to the design. The project management tools operate on the actual project data rather than on a model thereof; this removes another set of data which need to be maintained from the business. Concurrency is supported through the capacity for recursion of activities to fine levels of granularity and the relationships established explicitly between entities in different activities.

Design methodologies can be established to support the designer in performing well understood design tasks. This does not remove the designer's ability to invoke functions at will and without prescription as to their nature or order; Guide provides a framework for the designer rather than a path that must be followed.

The storage of definitions independently of design data allows for definitions as well as data to be exchanged between actors without prescription of their form such as is imposed, for example, by PDES. Thus, PDES data could be transferred to a client, or a whole set of entities definitions and the derived product model data could be sent to a project collaborator. The benefits over use of the restricted PDES entity set is that the required definition may not exist or may not fit the requirements of both parties in the exchange.

The design history record can be used to:

- roll back the design to a previous state,
- explore several options before choosing to pursue one of them,

- provide standard parts complete with their development history and all their properties,
- capture the knowledge of designers and make it available to other designers in a permanent and usable format,
- audit design processes and constraints for effectiveness and appropriateness.

Third party utilities and legacy data are accessible from Guide. The use of third party software such as geometry representation engines ensures that the capacities of the system may always be state of the art.

Communications occur in native format, without translation. This occurs at many different levels which include those communications necessary to undertake concurrent processes. The design history record is stored using the same semantic definitions as the rest of the design environment and can itself therefore be easily transmitted, without loss of information.

This is a key difference between Guide and commercial software and the approach taken by many researchers. Commercial software use translators and metadata to manage what is not represented by large integrated applications. They focus on the detail and implementation phases of design and provide tools to manage design data rather than design projects.

An approach common to several researchers is to establish concurrent engineering or some other goal within a narrow, defined domain. This is similar to the development of commercial integrated applications and does not solve the problems of interface between their application and the rest of the company and business communities.

The Guide system demonstrates its abilities to perform in the business environment through applications and case studies developed with the industrial partners in the research programmes engaged. Design and business functions are supported to any level or degree of granularity – two examples show this: a detailed design methodology evolved and applied in a specific field and a general engineering change control process employed in an extended enterprise.

References

The references are organised according to the chapter and sequence in which they are first referred to.

Chapter 1 The Product Model and its application using Guide

1.1 Dixon J.R. "Knowledge-based systems for design" *Journal of Mechanical Design: Transactions of the ASME* 1995 Vol. 117, pp 11-16

1.2 Urban S.D., Shah J.J., Rogers M., Jeon D.K., Ravi P. and Bliznakov P. "A heterogeneous, active database architecture for engineering data management *International Journal of Computer Integrated Manufacturing* 1994, Vol. 7 No 5 pp 276-293

1.3 Shaw N.K., Bloor M.S. and de Pennington A. "Product data models" *Research in Engineering Design* 1989 Vol. 1, pp 43-50

1.4 Tsiotsias, A.S. "Design method and management utility enabling the concurrent exercise of distributed expertise" *PhD Thesis*, 1994, University of Glasgow

1.5 Edwards K. and Murdoch T. "Modelling engineering design principles" *International Conference on Engineering Design* 1993 The Hague, pp. 1676-1683.

1.6 Eastman C.M. "The contribution of data modelling to the future development of CAD/CAM databases" *Engineering Databases: An Engineering Resource* ASME 1991, pp 49-54

1.7 Cleetus K.J. "Modeling evolving product data for concurrent engineering" *Engineering with Computers* 1995, Vol. 11, pp. 167-172

1.8 Majumder D. and Fulton R.E. "Designer in the ICAD environment: The information perspective" *ASME International Conference on Computers in Engineering* 1990 Boston pp. 85-92

1.9 Shah J.J., Jeon D.K., Urban S.D., Bliznakov P. and Rogers M. "Database infrastructure for supportting engineering design histories" *Computer-AidedDesign.* 1996 Vol 28, No.5, pp 347-360

1.10 Su C.-J., Tseng, M.M. and Mayer R.J. "EKAMD – A knowledge-based concurrent engineering support system" *Concurrent Engineering: Research and Applications* 1997 Vol. 5, No 1, pp 59-76

1.11 Finger S., Konda S. and Subrahmanian E. "Concurrent design happens at the interfaces" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 1995 Vol. 9, pp. 89-99

1.12 Hammer M. and Champy J. "Reengineering the corporation: a manifesto for business revolution" 1993 Nicholas Breadley Publishing, London

1.13 Minsky M. "A framework for representing knowledge" in *The Psychology of Computer Vision* edited by P.H. Winston, McGraw-Hill, pp. 221-277

1.14 Xue D. and Dong Z. "Developing a quantitative intelligent system for implementing concurrent engineering design. *Journal of Intelligent Manufacturing* 1994, Vol. 5, pp.251-267

1.15 Finger S., Gardner E. and Subrahmanian E. "Design support systems for concurrent engineering: a case study in large power transformer design" *International Conference on Engineering Design* 1993, The Hague, pp 1433-1440.

1.16 Willcox M.D. and Sheldon D.F. "How the design team in management terms will handle the dfx tools" *International conference on Engineering Design* 1991, Zurich, pp.875-881

1.17 Wood III W.H. and Agogino A.M. "Case-based conceptual design information server for concurrent engineering" *Computer-Aided Design* 1996, Vol. 28, No.5, pp. 361-369

1.18 Kunz J.C., Luiten G.T., Fisher M.A., Jin Y. and Levitt R.E. "CE4: Concurrent engineering of product, process, facility and organization" *Concurrent Engineering: Research and Applications.* 1996, Vol. 4, No. 2, pp. 187-198

1.19 D'Ambrosio J., Darr T. and Birmingham W. "hierarchical concurrent engineering in a multiagent framework" *Concurrent Engineering: Research and Applications* 1996 Vol. 4, No. 1, pp.47-57

1.20 Goodwin R. and Chung P.W.H. " An integrated framework for representing design history" *Applied Intelligence* 1997 Vol. 7 pp 167-181

1.21 Hardwick M. and Downie B.R. "On object-oriented databases, materialized views, and concurrent engineering" *Engineering Databases, an Engineering Resource* 1991, Proceedings of the ASME International Conference on Computers in Engineering, Santa Clara, pp 93-97

1.22 Brissaud D. and Garro O. "An approach to concurrent engineering using distributed design methodology" *Concurrent Engineering: Research and Applications* 1996, Vol. 4, No 3, pp. 303-311

1.23 Erens F., McKay A. and Bloor S. "Shortcomings of today's design frameworks" *International Conference on Engineering Design"* 1993 The Hague, August 17-19, pp 1279-1286

1.24 Ranta M., Mäntylä M., Umeda Y. and Tomiyama T. "Integration of functional and feature-based product modelling – the IMS / GNOSIS experience" *Computer-Aided Design"* 1996 V28 No5 PP 371-381

1.25 Sinclair M.A., Siemieniuch C.E. and John P.A. "A user centred approach to define high-level requirements for next-generation CAD systems for mechanical engineering" *IEEE Transactions on Engineering Management* 1989, Vol. 34, No. 2, pp. 211-228

1.26 Aasland K. "Design history in connection with the domain theory" *International conference on engineering design* 1993 The Hague pp. 1385-1392.

1.27 Garcia A.C.B. and Howard H.C. "Acquiring design knowledge through design decision justification" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 1992 Vol. 6, No. 1, pp. 59-71

1.28 Salomons O.W., van Slooten F., van Houten F.J.A.M. and Kals H.J.J. "A computer support tool for re-design" *International conference on Engineering Design* 1993 The Hague pp. 1559-1569

1.29 Mattinsley B.W. "Engineering process redesign – the management agenda for the 90s" *IFAC* 1992 pp1-5

1.30 Mäntylä M. "Advances in product modelling" *Proceedings of the Technodata symposium,* 1990, Berlin, pp11-27

1.31 Beitz W. and Feldhusen J. "Management systems and program concepts for an integrated CAD process", *Research in Engineering Design* 1991, Vol 3, pp 61-74

1.32 The Bible, New International Version. Hodder and Stoughton, 1987 Book of Genesis, Ch. 11, vv. 4b-8

Chapter 2 The Product Model in a Business Context

2.1 Nijhuis W. and Oudt H.A. "The relation of general management and design management, a status report" *International conference on Engineering Design* 1991, Zurich, pp. 841-846

2.2 Fohn S.M., Greef A.R., Young R.E. and O'Grady P.J. "A constraint-system shell to support concurrent engineering approaches to design" *Artificial Intelligence in Design* 1994 Vol 9 pp 1-17

2.3 Hanson P. and Voss C. "Made in Britain - The true state of Britain's manufacturing industry" 1993, IBM / London Business School

2.4 McDonald R.E. "The critical importance of database management in industrial automation" *ASME Computer in Engineering* 1990 Boston, pp131-144

2.5 Colton J.S. and Dascanio II J.L. "An integrated, intelligent design environment" *ASME International Conference on Computers in Engineering* 1990 Boston, pp. 9-15

2.6 Stark J. "Engineering information management system – Beyond CAD/CAM to concurrent engineering support" 1992 Van Nostrand Reinhold

2.7 Shah J.J. and Rogers M.T. "Functional requirements and conceptual design of the feature-based modelling system" *Computer-Aided Engineering Journal* 1998 February Vol. 5, No. 1, pp.9-15

2.8 Urban S.D., Shah J.J., Liu H. and Rogers M. "The Shared Design Manager: interoperability in engineering design" *Integrated Computer-Aided Engineering* 1996, Vol. 3, No. 3, pp. 158-177

2.9 Arkwright A.J. "Simultaneous engineering - viable proposition or pipe dream?" 1992, IFAC, pp 1-7

2.10 Garcia A.C.B., Howard H.C. and Stefik M.J. "Improving design and documentation by using partially automated synthesis" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 1994 Vol. 8, pp. 335-354

2.11 Wood K. "The organisation and control of design for a large multidiscipline construction project" *International Conference on Engineering Design* 1989, Harrogate, pp. 111-122

2.12 Gierlinger C., Tjoa A.M. and Wagner R.R. "A methodology for computer aided modelling of information systems based on the extended entity relationship model BIER" *Lecture Notes in Computer Science* 1992, Vol 585, pp. 566-584

2.13 Eder W.E. "Information systems for designers" *International Conference on Engineering Design* 1989, Harrogate, pp1307-1319

2.14 Hoffman B. "Albert Einstein, creator and rebel" 1973, New American Library Publishers, New York.

2.15 Court A.W., Culley S.J. and McMahon C.A. "The information requirements of engineering designers" *International Conference on Engineering Design* 1993 The Hague pp. 1708-1716

2.16 Bradburn B. "A comparison of knowledge elicitation methods" *International Conference on Engineering Design* 1991 Zurich pp. 298-305

2.17 Peckham J. and Maryanski F. "Semantic data models" *ACM Computer Surveys* 1988, Vol 20, No 3, pp153-189

Chapter 3 Guide Facilities

3.1 Rohatynski R. and Dabrowski Z. "Partnerous CAD system – a new concept of aiding of engineering design" *International conference on Engineering design* 1991 Zurich pp. 904-908

3.2 Park H., Cutkosky M.R.,Conru A.B. and Lee S.-H. "An agent-based approach to concurrent cable harness design" *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 1994, Vol. 8, pp. 45-61

3.3 Van Harpen N.T. and Luiten M.E.G. "Information technology: the key to an integrated development process" *International conference on engineering design* 1993 The Hague, pp 1263-1270

3.4 Saxena M. and Irani R.K. "A knowledge-based engineering environment for automated analysis of nozzles" *Concurrent Engineering: Resaearch and Applciations* 1994, Vol. 2, pp. 45-57

3.5 Esterline A., Rosen D., Otto K., Nelson L., Hessburg T., Riley D.R. and Erdman A.G. " A methodology for capturing mechanical design expertise" *Computers in Engineering* 1988 San Francisco ASME Vol. 1, pp 47-55

3.6 Kelly M.J. "Expert systems: a misnomer" *ASME International Conference on Computers in Engineering* 1988, San Francisco, pp. 205-207

3.7 Katz R.H. "Computer-Aided Design Databases" *IEEE Design and Test of Computers*, 1985, Vol. 2, No. 1, pp. 70-74

3.8 Lawson M. and Karandikar H.M. "A survey of concurrent engineering" *Concurrent Engineering: Research and Applications* 1994 Vol. 2, pp. 1-6

3.9 O'Grady P.J., Kim Y. and Young R.E. "A hierarchical approach to concurrent engineering systems" *International Journal of Computer Integrated manufacturing* 1994, Vol. 7, No. 3, pp. 152-162

3.10 McGreavy C., Wang X.Z., Lu M.L., Zhang M. and Yang S.H. "Objects, agents and work flow modelling for concurrent engineering process design" *Computers in Chemical Engineering* 1996 Vol. 20 Supplement pp. s1167-s1172

3.11 Boothroyd G. and Dewhurst P. "Design for Assembly: a designer's handbook" 1983 Boothroyd Dewhurst inc., Amherst

3.12 Suri R., Desiraju R. and Simizu M. "Design for analysis leads to a new concept in plating lines" *Industrial Engineering* 1993, Vol. 25, No. 8, pp. 54-60

3.13 Jayaputera G.T. and Cheng K.E. "SoftEAM: a design history and justification maintenance tool" *The Australian Computer Journal* 1994 Vol. 26, No. 4, pp.124-133

3.14 Visser W. "Collective design: a cognitive analysis of co-operation in practice" *International conference on Engineering Design* 1993, The Hague, pp 385-392.

3.15 Bañares-Alcántara R. "Representing the engineering design process: two hypotheses" *Computer-Aided Design* 1991 Vol.23, No. 9, pp. 595-603

3.16 Gu P. and Chan K. "Product modelling using STEP", *Computer-Aided Design* 1995, Vol. 27, No 3, pp 163-179

3.17 Meehan E.J. and Brown D.C. "Constraint absorption and relaxation using a design history" *ASME International Conference on Design Theory and Methodology*, 1990, Chicago, pp. 193-202

3.18 Zhu Q. "A modular learning structure for knowledge-based CAD systems" *ASME International Conference on Computers in Engineering* 1988, San Francisco, pp. 293-299

3.19 Sugumaran V. and Bose R. "Expert system technology in organisational process domain modelling" *Expert Systems* 1996 Vol. 13, No. 1, pp.15-28

3.20 Reed G.R and Sturges R.H. "A model for performance-intelligent design advisors" *Concurrent Engineering: Research and Applications* 1994 Vol. 2, pp 59-65

3.21 Smith D.G. and Ball A.A. "Support systems for design and manufacture – the real needs" *International Conference on Engineering Design* 1993, The Hague pp. 1271-1278

Chapter 4 Guide architecture

4.1 Law K.H., Wiederhold G., Barsalou T., Siambela N., Sujansky W. and Zingmond D. "Managing design objects in a sharable relational framework" *Enginering Data management: The technology for integration ASME Computers in Engineering* 1990 Boston pp. 61-66

4.2 Ehrlenspiel K. and Schaal S. "Ways to smarter CAD-systems – exemplified through design-concurrent calculation" *International Conference on Engineering Design* 1991 Zurich pp. 609-617

4.3 Richter D. "Management for product data" *Proceedings of the Technodata symposium*, 1990, Berlin, pp51-60

4.4 Bello R., Galvez D., Benavides G., Garcia M.M. and Sanchez G. "Some tools oriented to knowledge based cad systems implementation" *International Conference on Engineering Design* 1991 Zurich pp. 1065—1068

4.5 Chen C.-S. "Developing a feature based knowledge system for CAD/CAM integration" *Computers and Industrial Engineering* 1988 Vol 15., pp. 34-40

4.6 Wang M.-T., Waldron M.B. and Miller R.A. "A prototype integrated feature-based design and exert process planning system for rotational parts" *International computers in Engineering* 1990 Boston pp.33-40

4.7 Chen Y.-M., Wei C.-L. "Computer-aided feature-based design for net shape manufacture" *Computer Integrated Manufacturing Systems* 1997, Vol. 10, No 2, pp 147-164

4.8 Chen C., Swift F., Lee S., Ege R. and Shen Q. "Development of a feature-based and object-oriented concurrent engineering system" *Journal of Intelligent Manufacturing* 1994 Vol. 5, pp. 23-31

4.9 Geiger T.S. and Dilts D.M. "Automated design-to-cost: integrating coting into the design decision" *Computer-Aided Design* 1996 Vol. 28, No. 6/7, pp. 423-438

4.10 Das D., Gupta S.K. and Nau D.S. "Generating redesign suggestions to reduce setup cost: a setp towards automated redesign." *Computer-Aided Design* 1996, Vol. 28, No. 10, pp. 763-782

4.11 Rane P. and Isaac J.R. "Functionality: the key concept towards integration of the product cycle" *ASME International Conference on Computers in Engineering* 1990 Boston, pp. 317-326

4.12 Stomph-Blessing L.T.M "Analysing an engineering design process in industry" *I.Mech.E International Conference on Engineering Design,* 1989, pp. 57-64

4.13 McMahon E.H. "Group design system" *International conference on Engineering Design* 1991 Zurich, pp 101-104

4.14 Ullman D.G. and Dietterich T.G. "Mechanical Design process Research Group: Research overview" 1989 Oregon State University

4.15 Favela J., Wong A. and Chakravarty A. "Supporting collaborative engineering design" *Engineering with computers* 1993 Vol 9 pp 125-132

4.16 Thomson J.B. and Lu S.C.-Y. "Design evolution managgement: a methodology for representing and utilizing design rationale." *Proceedings of the ASME Design Technical Conference* 1990, Chicago, pp. 185-191

4.17 Ramesh B. and Dhar V. "Supporting systems development by capturing deliberations during requirements engineering" *IEEE Transactions on Software Engineering* 1992 Vol. 18, No. 6, pp. 498-510

Chapter 5 Guide Applications for Industry

5.1 Hopper I., McCafferty J. and Gibson R. "Ceramic composites for aerospace propulsion: Final report" 1993 Glasgow University

5.2 Weissflog U. "CIM dictionaries: an evolutionary approach to CIM data integration" *Proceedings, ASME Computers in Engineering,* 1991, Santa Clara, pp. 71-77

Plates

The plates are captured from the computer screen running Guide. The version shown uses Catia services to provide a user interface. The colour map has been changed from that of the original images that had a dark background. This provides less attractive, but more legible images on paper.

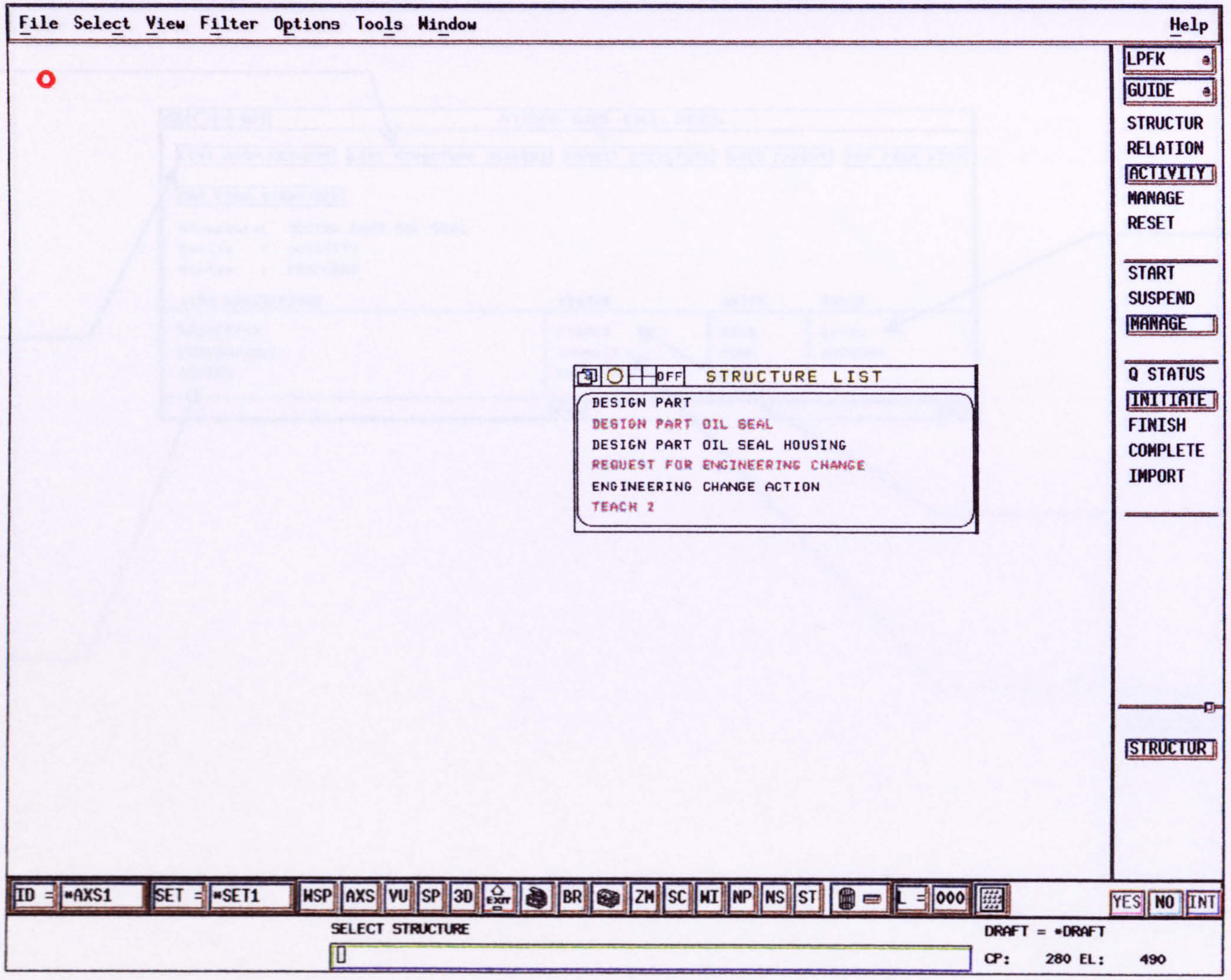


Plate 1 - List of activities available for instantiation

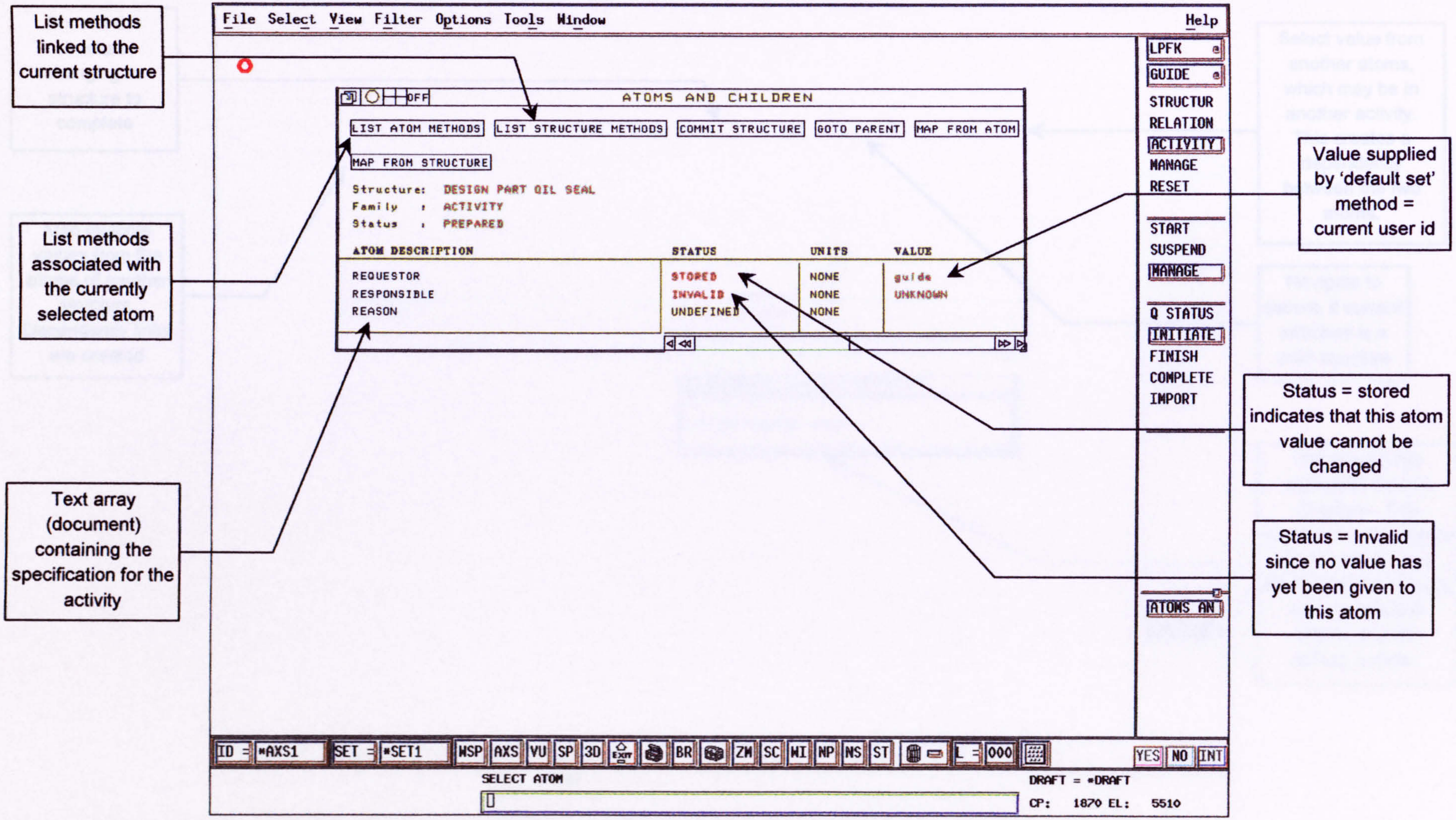


Plate 2 - Prepared instance of chosen activity

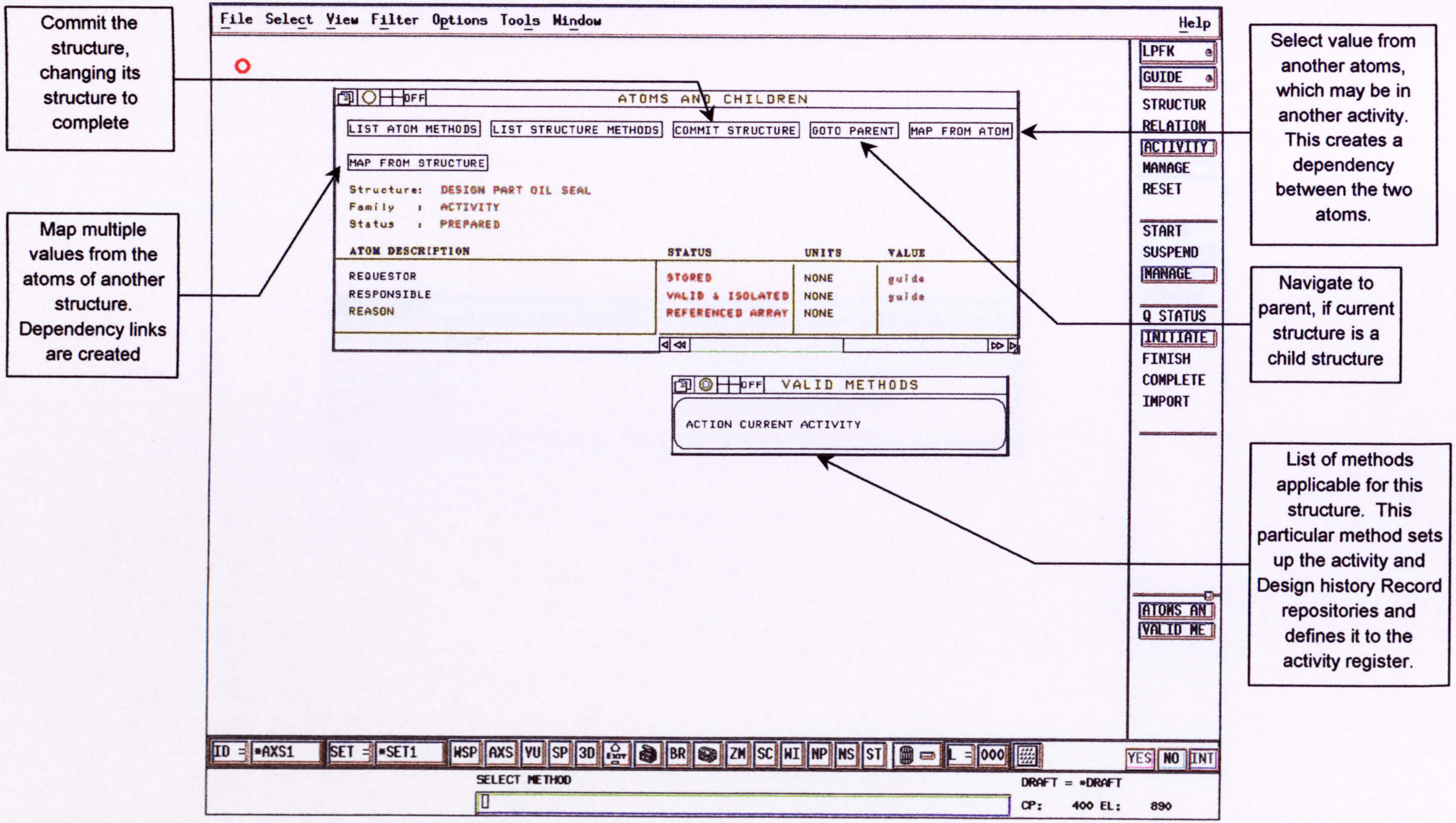


Plate 3 - Prepared activity, about to be actioned

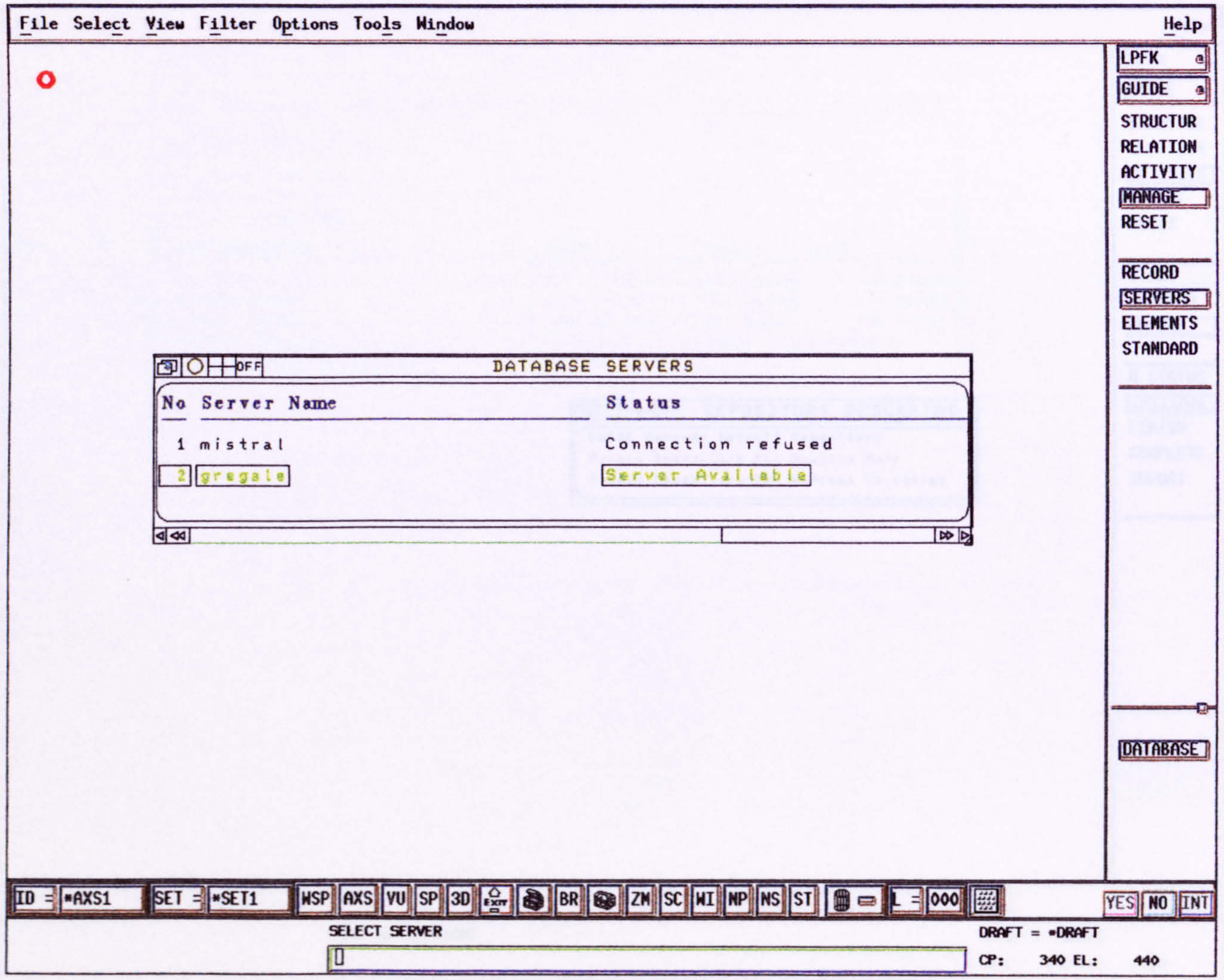


Plate 4 - Selection of Guide server from those available

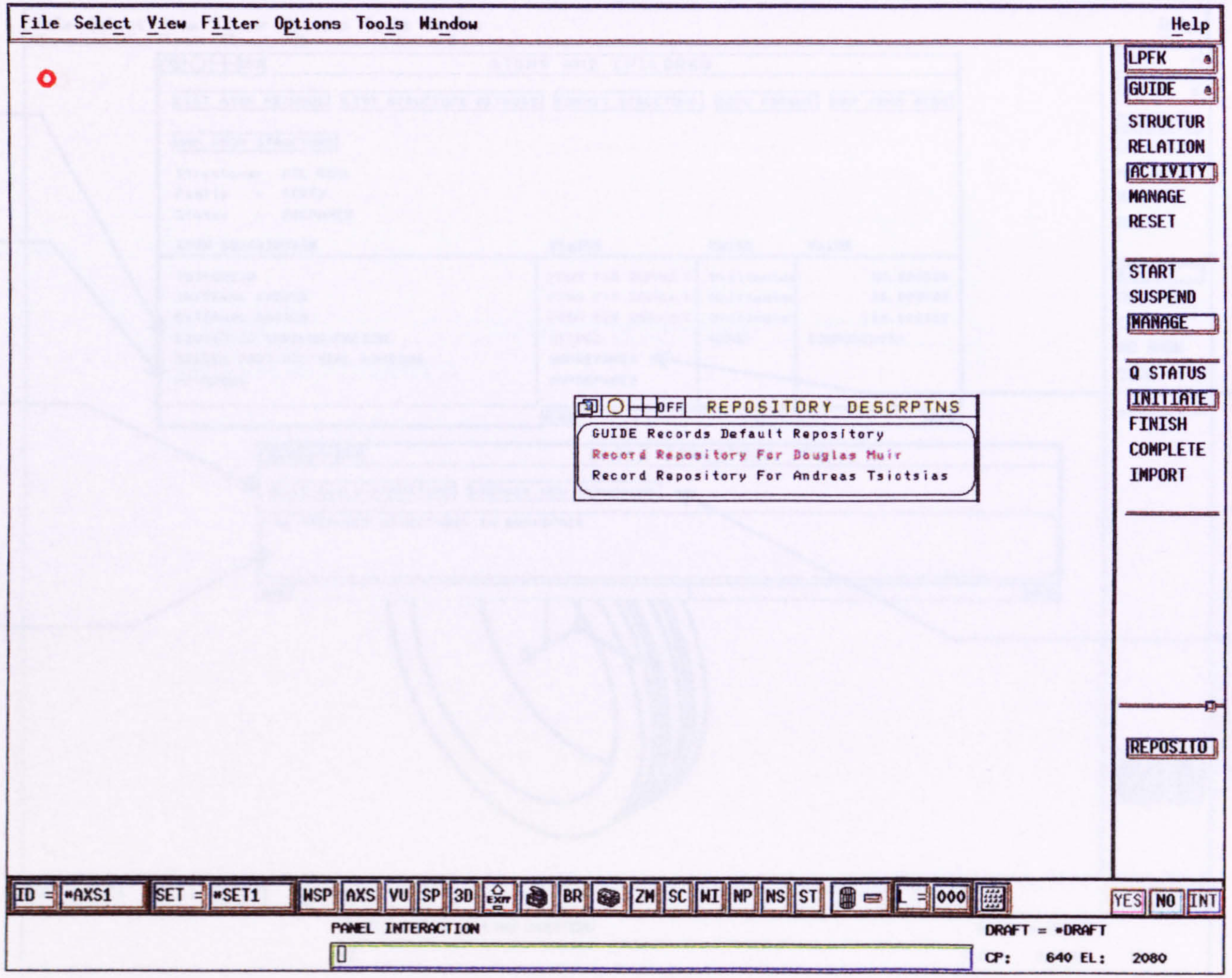


Plate 5 - List of repositories available for the storage of Guide product model data

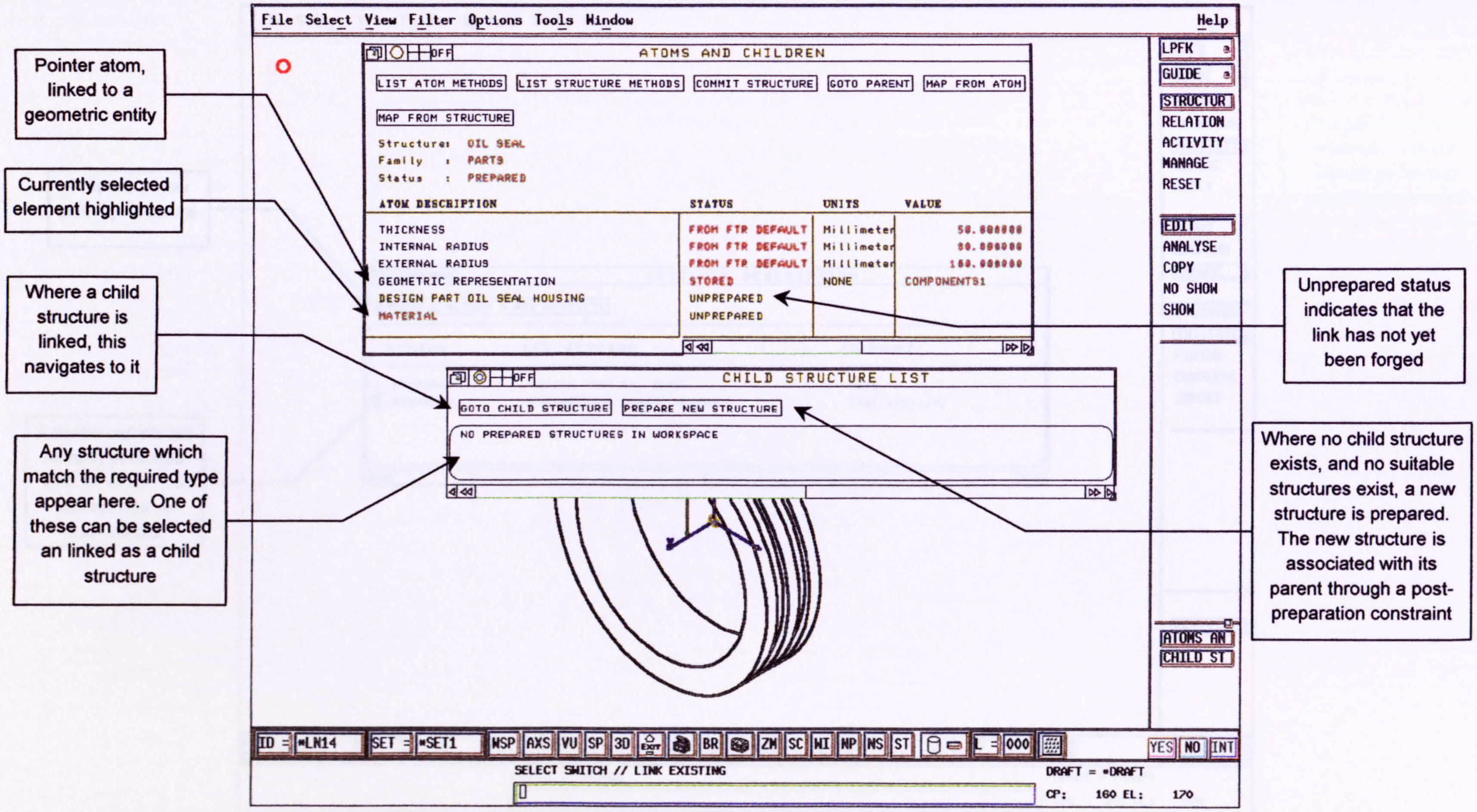


Plate 6 – Preparation of a new material structure as a child structure

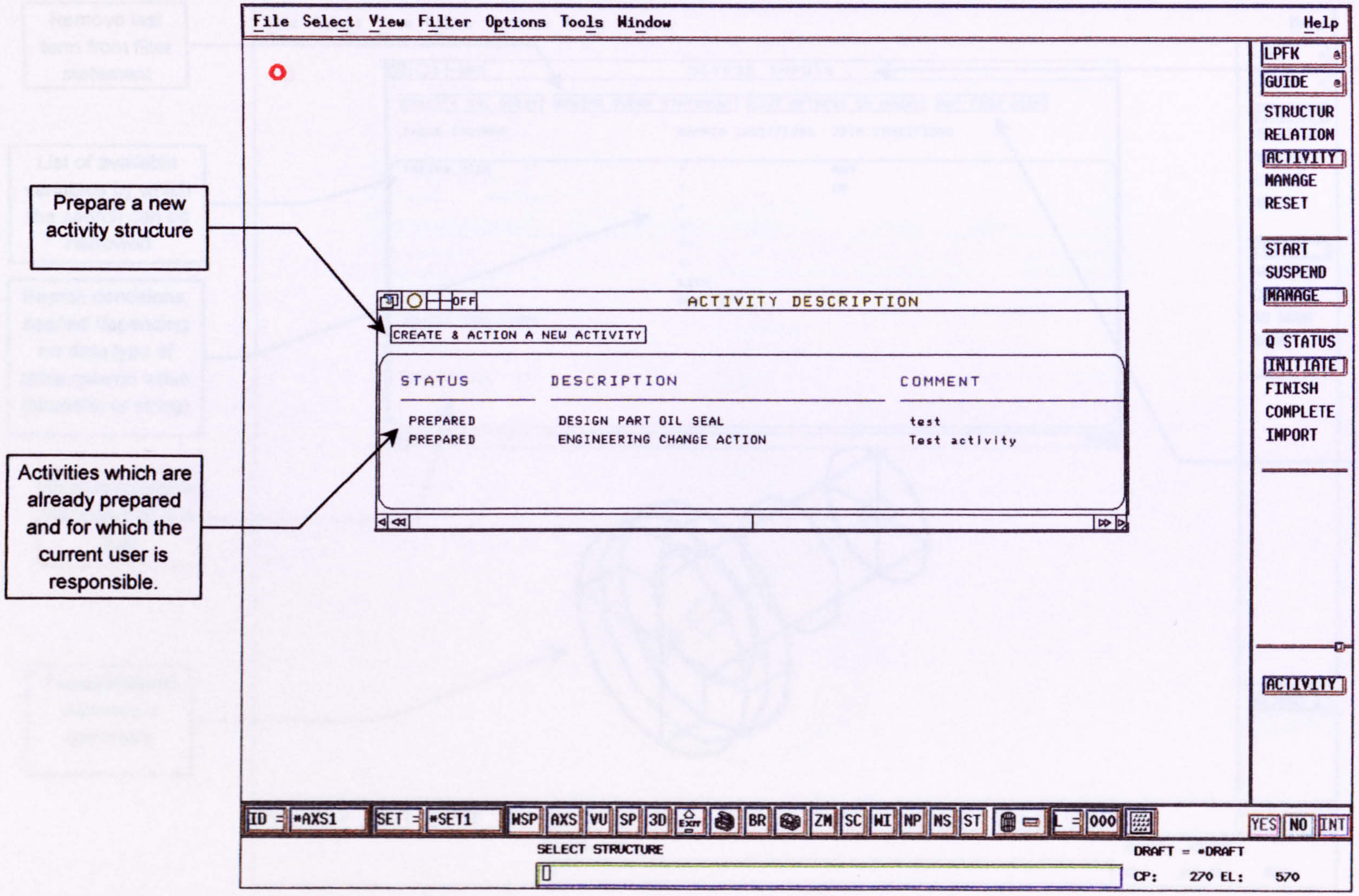


Plate 7 – List of activities for which the current user is responsible

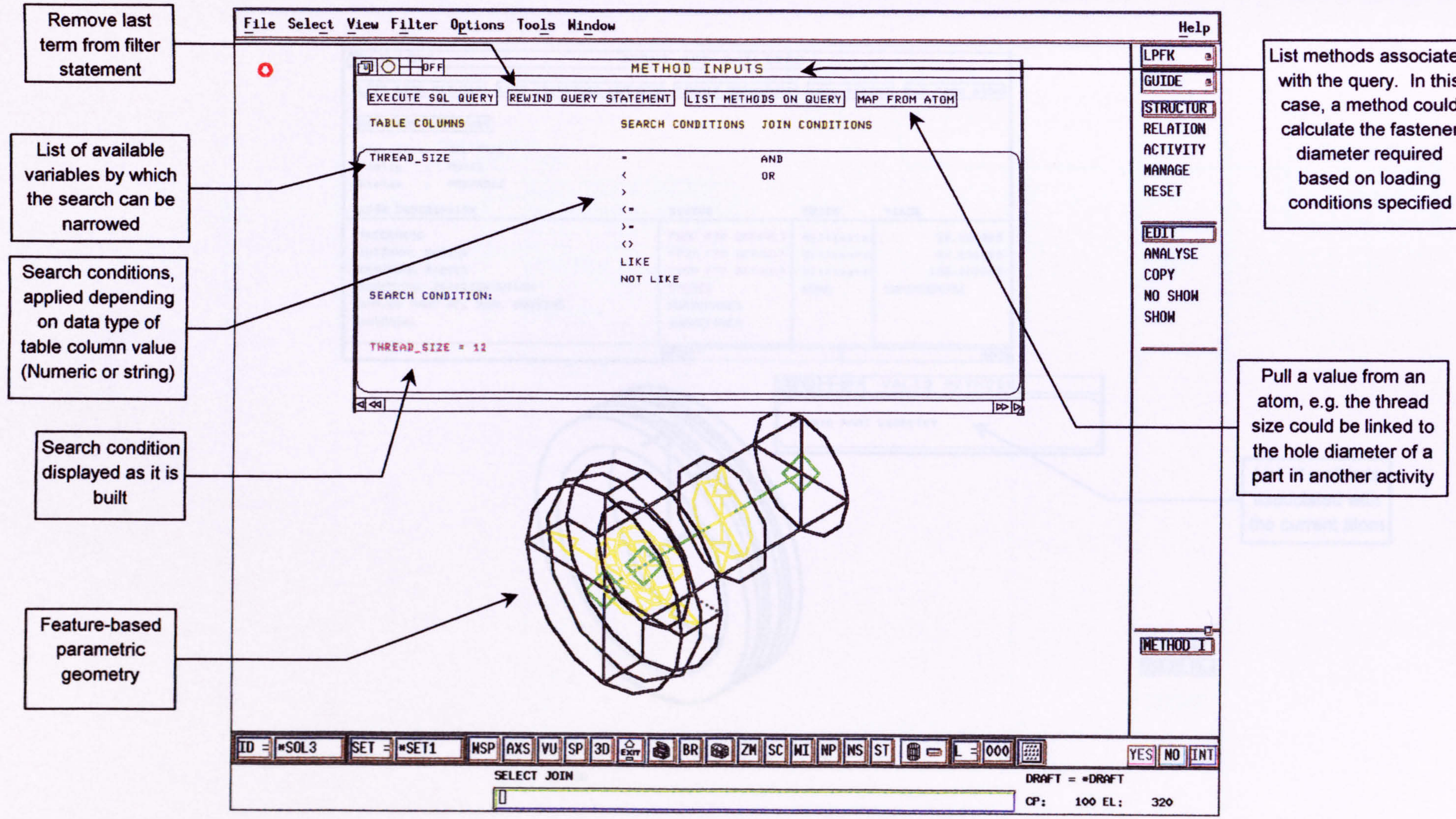


Plate 8 – Database query search condition panel

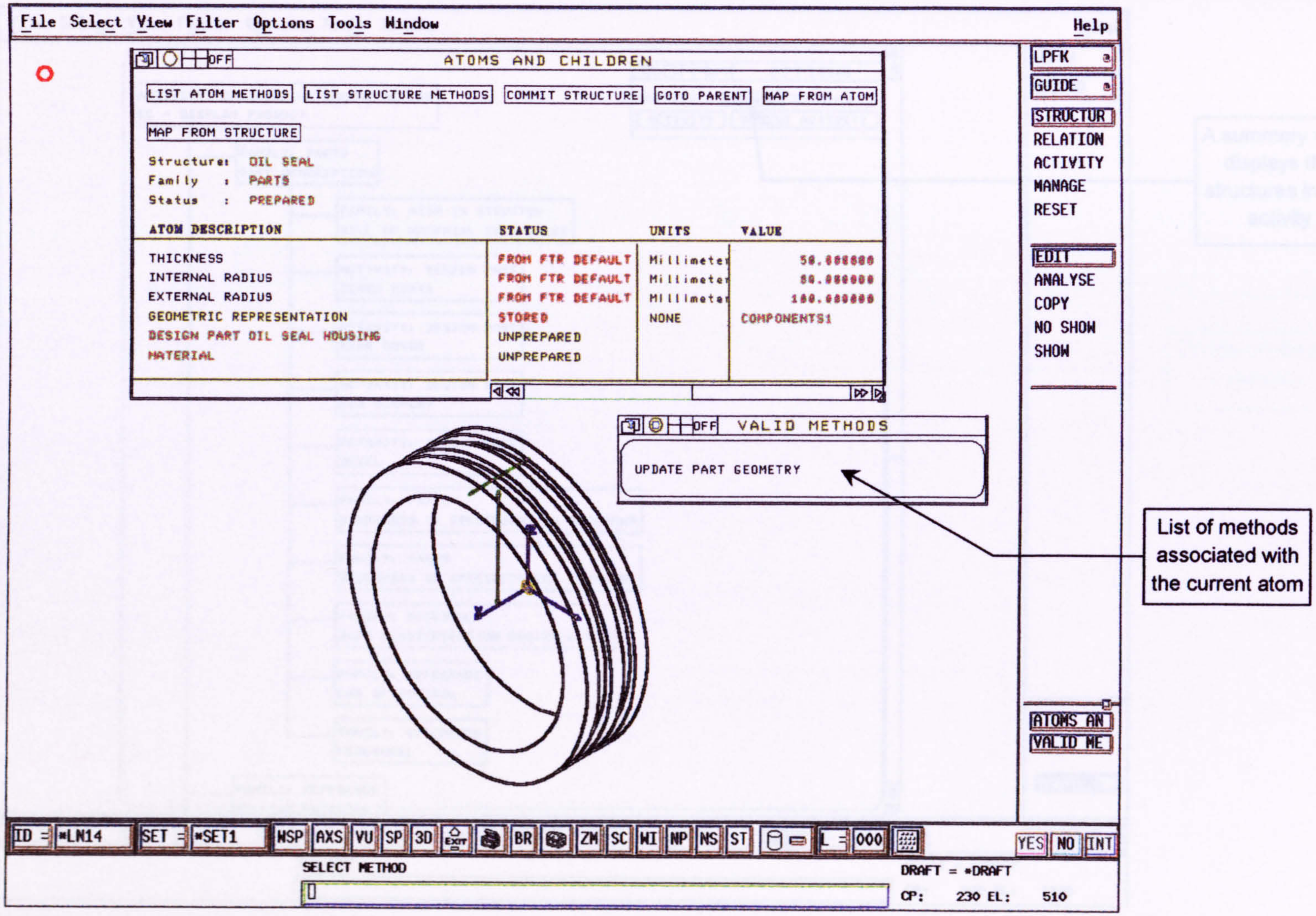


Plate 9 – Method list for an atom

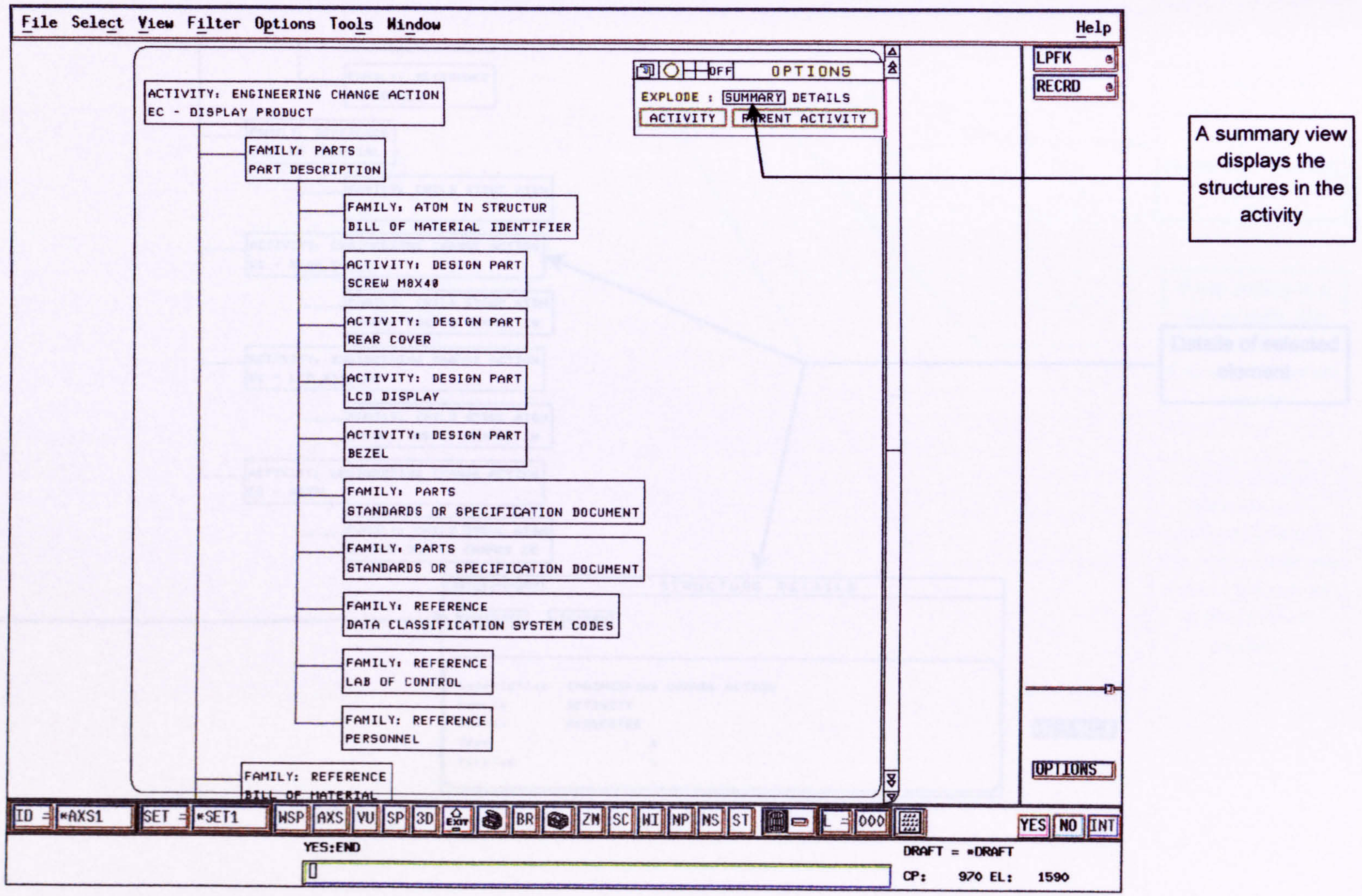


Plate 10 – Product model data navigation; contents of EC control activity

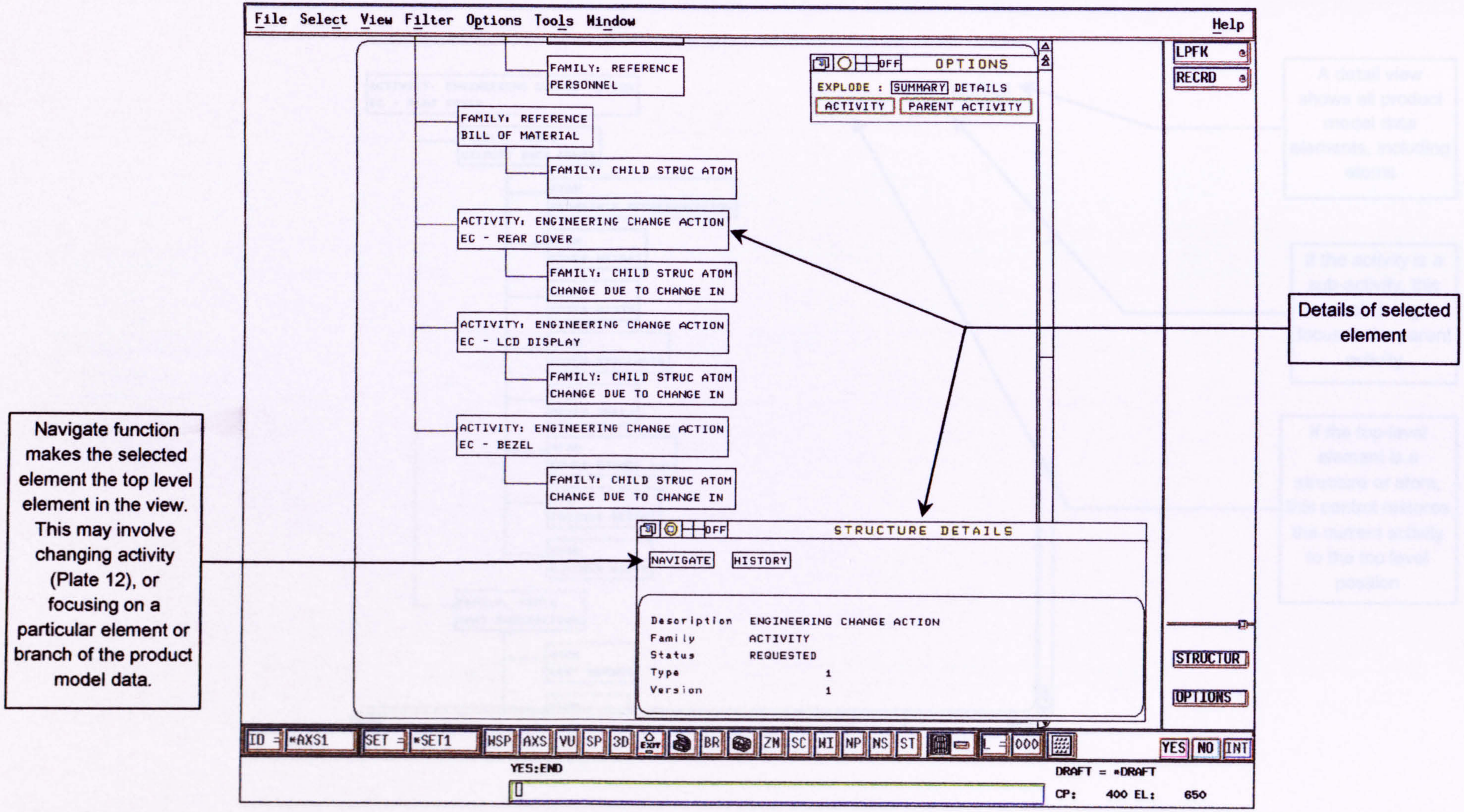


Plate 11 – Display of activity details

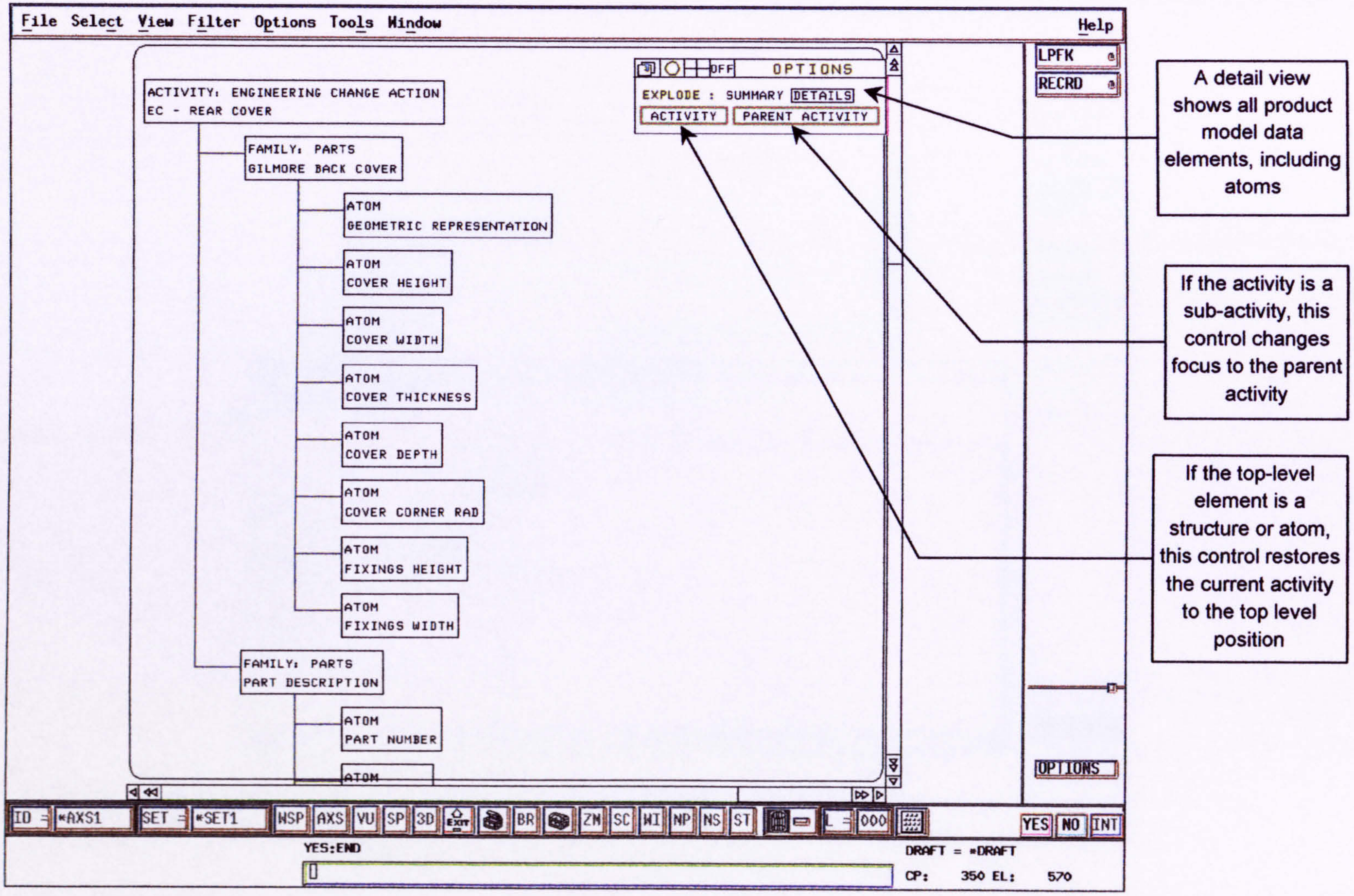


Plate 12 – Sub-activity displayed in detail mode

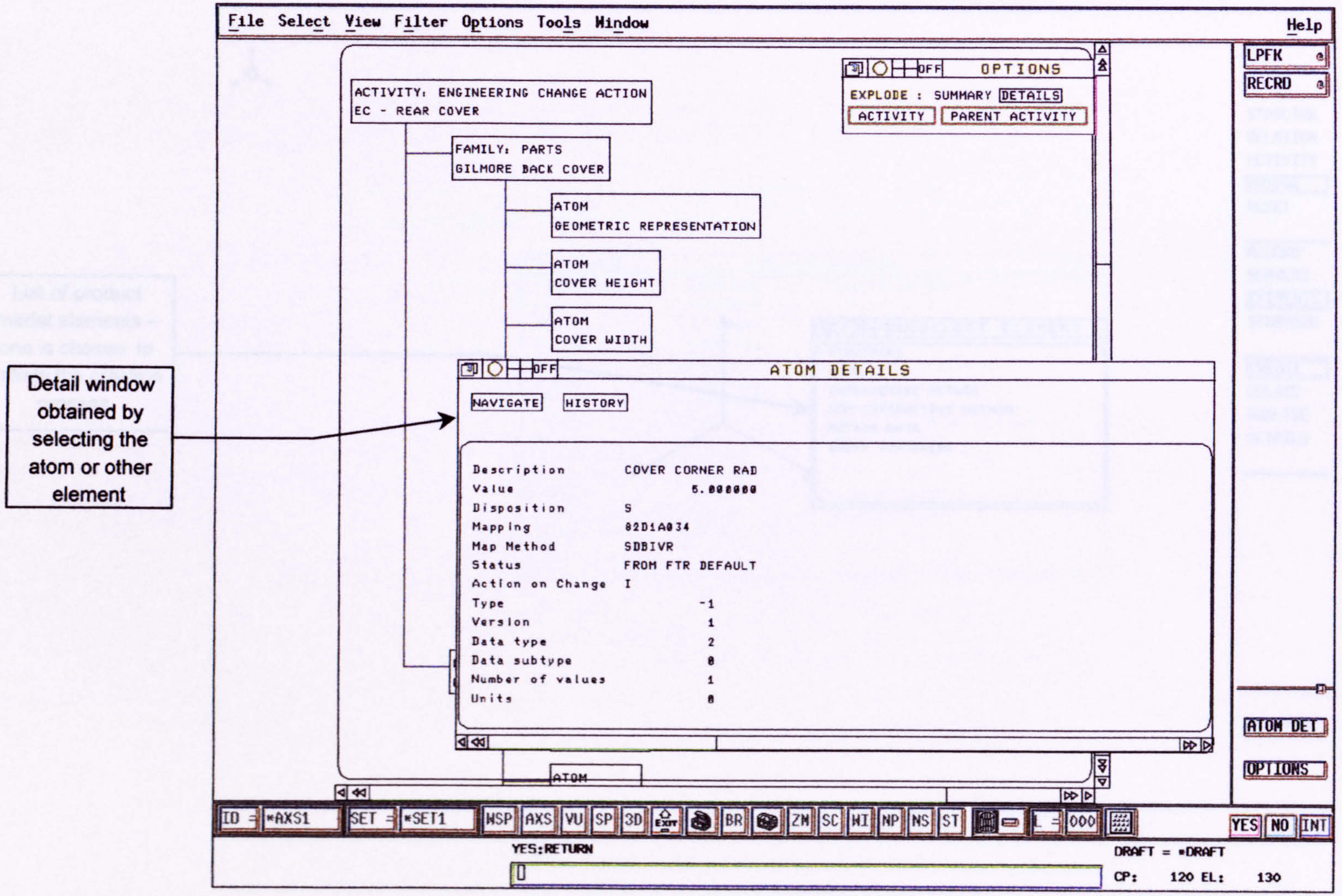


Plate 13 – Details of product model data element

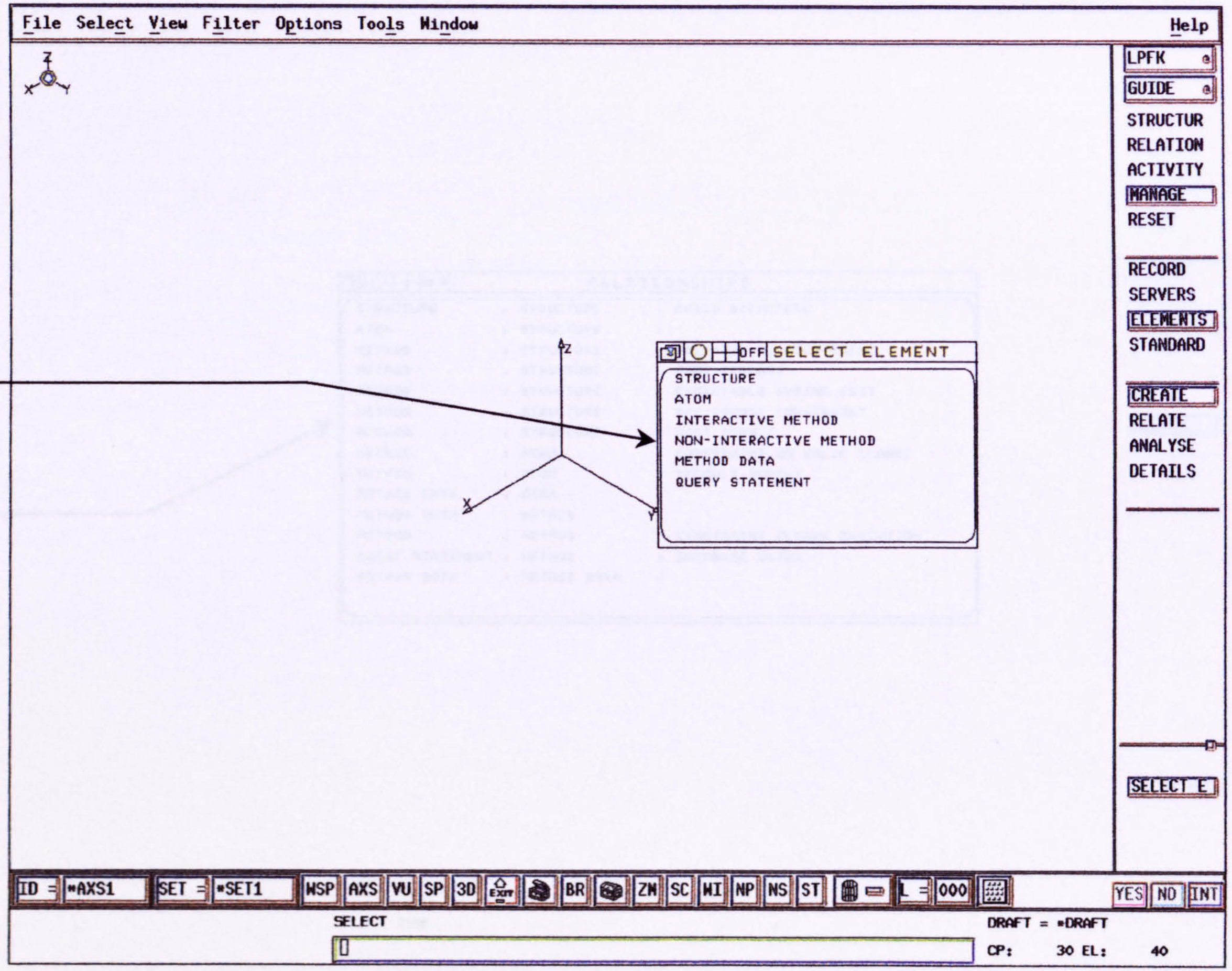


Plate 14 – Creation of a new product model element

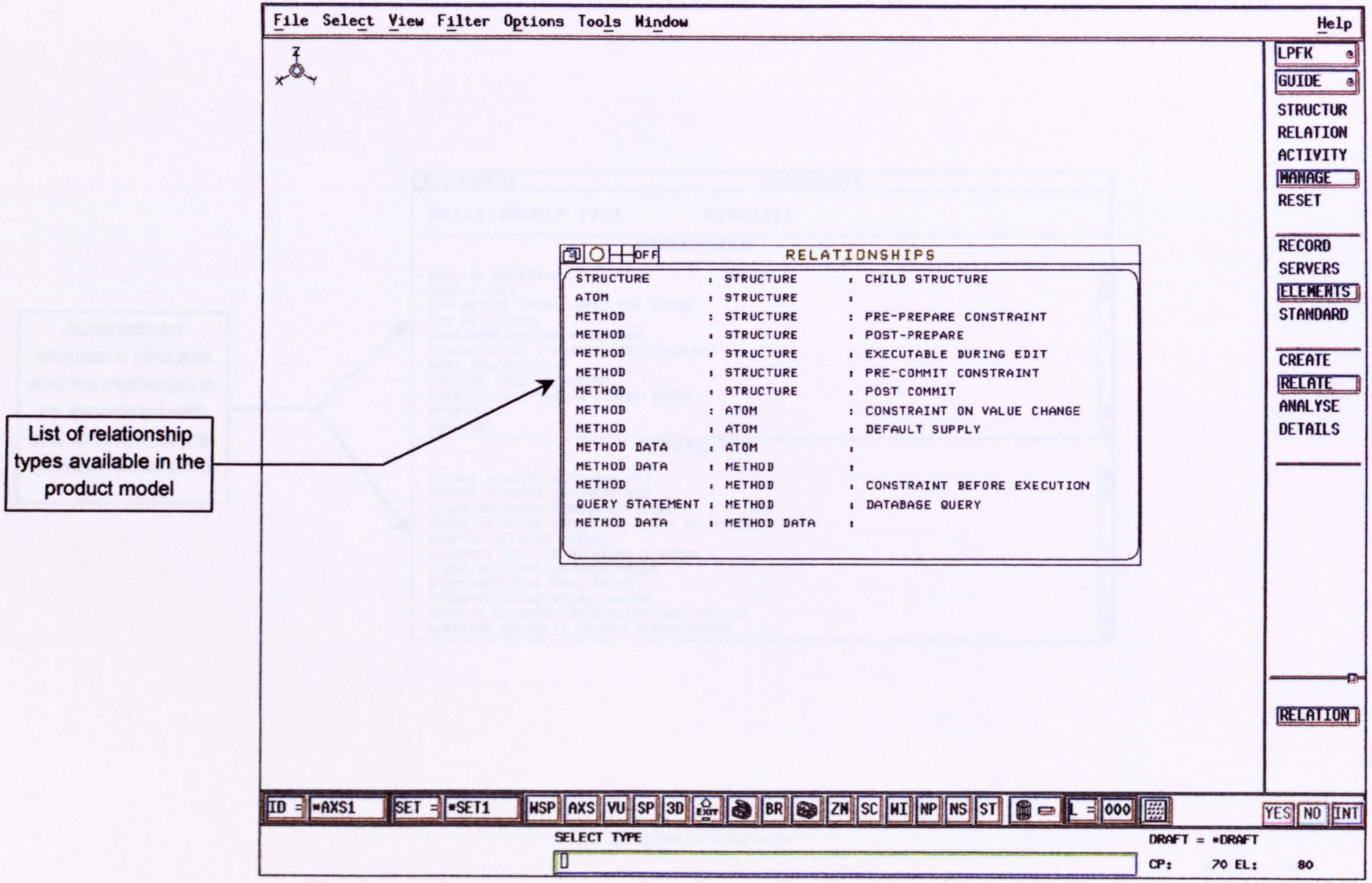


Plate 15 – Creation of a relationship in the product model: choice of relationship type to create

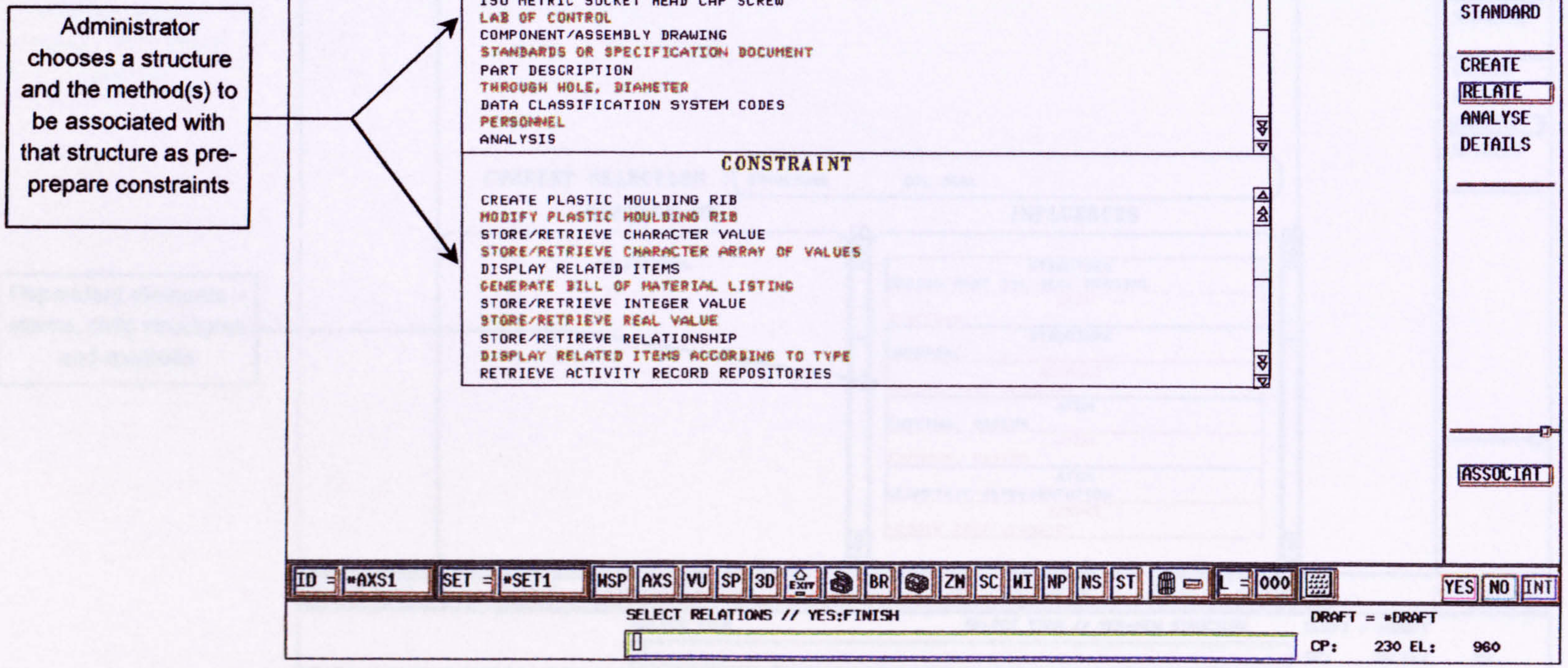


Plate 16 – Dialogue panel for the creation of a pre-prepare constraint relationship on a structure

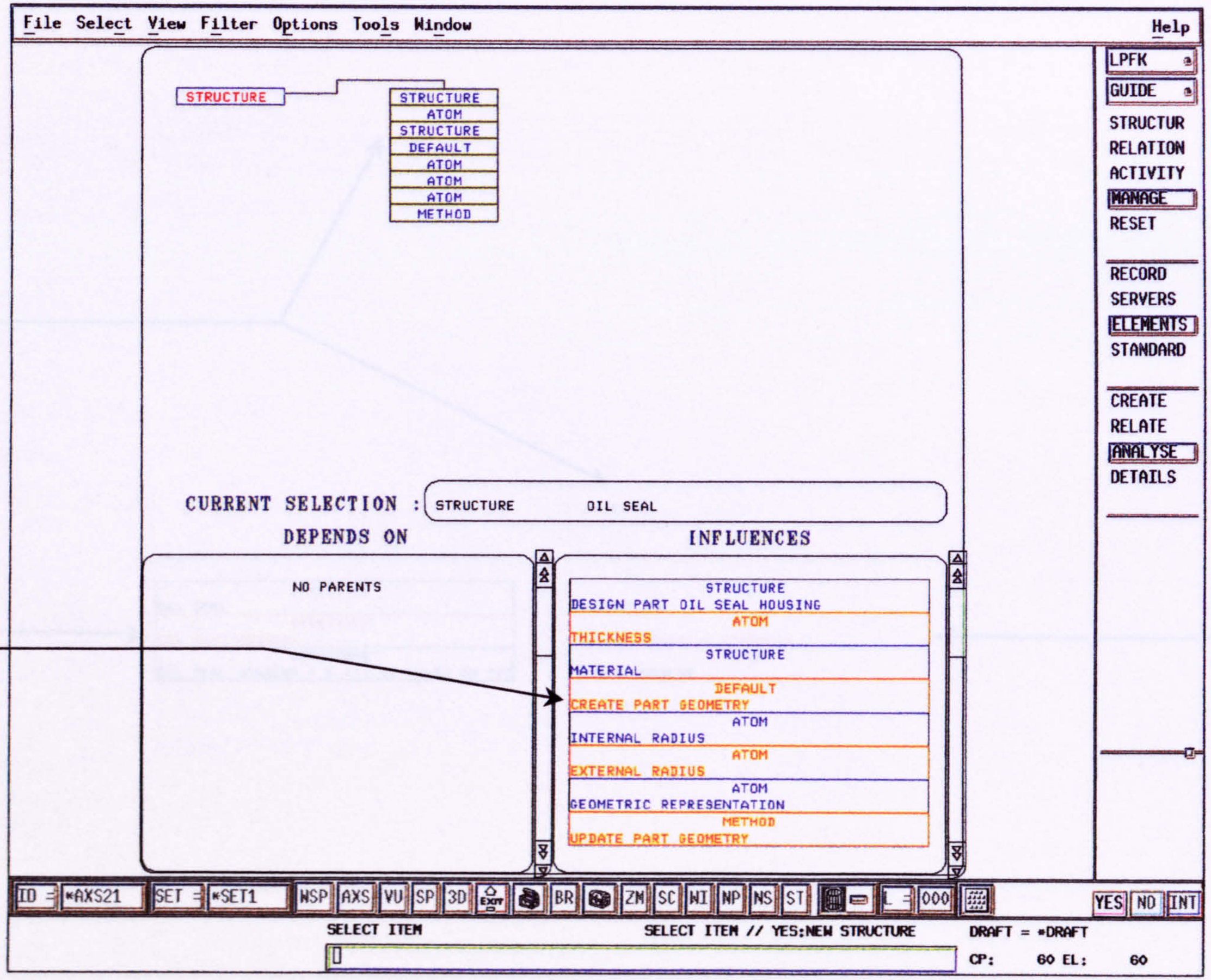


Plate 17 – Analysis of an oil seal structure definition

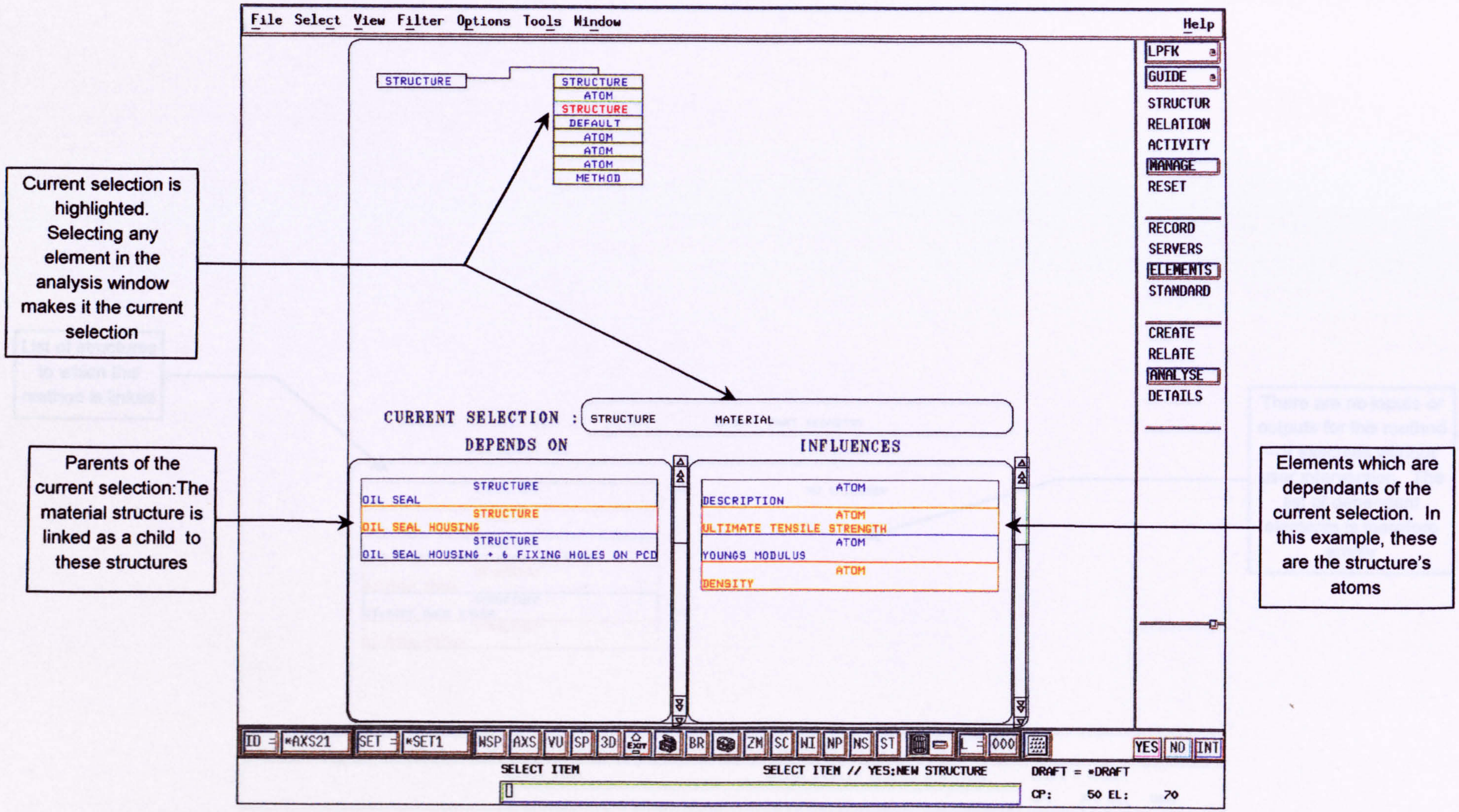
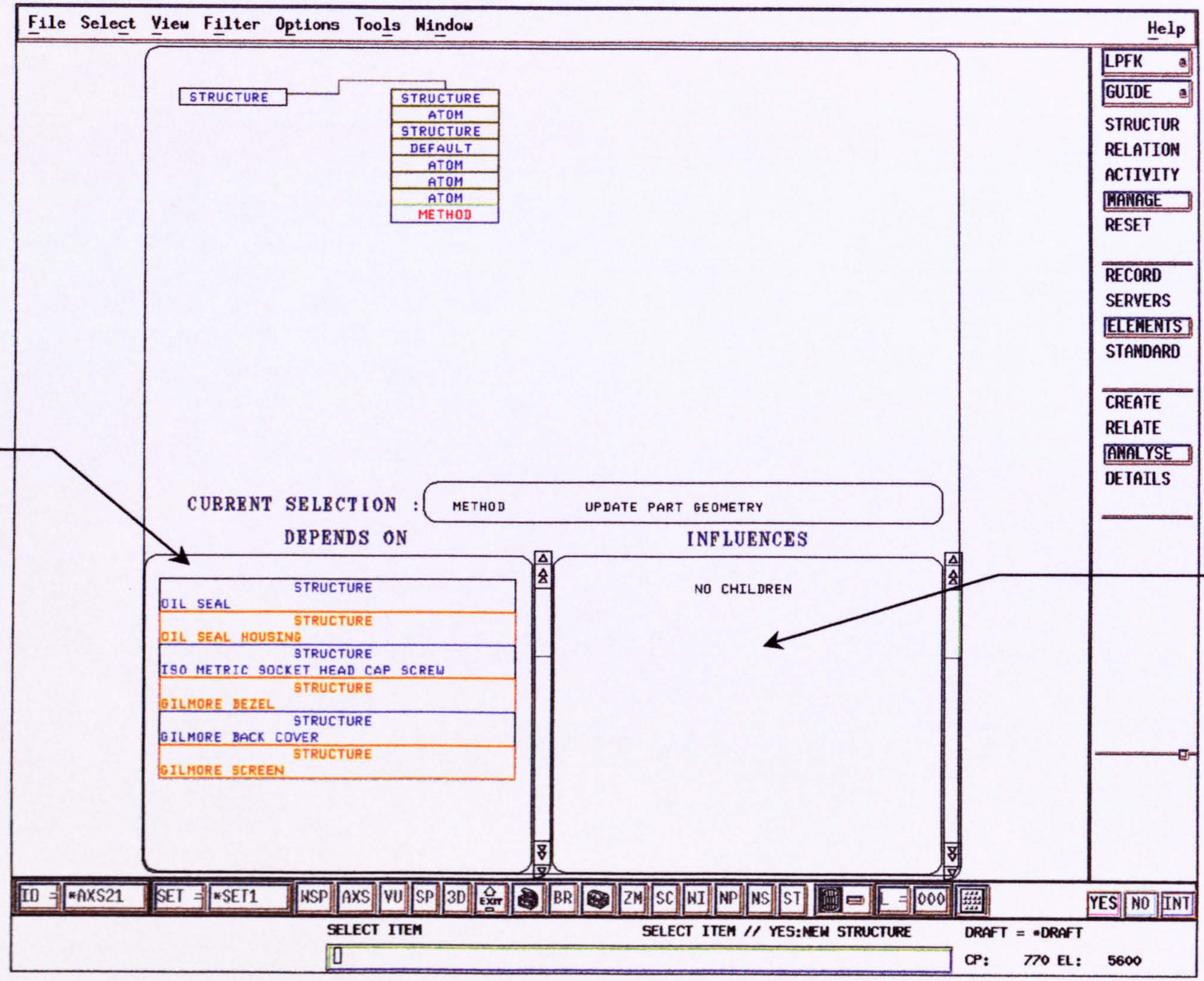


Plate 18 – Analysis of a Material structure definition

List of structures to which this method is linked



There are no inputs or outputs for this method – it executes without user intervention. The list of dependant elements is therefore empty

Plate 19 – Details of 'Update part geometry' method definition