

# Chapter 3

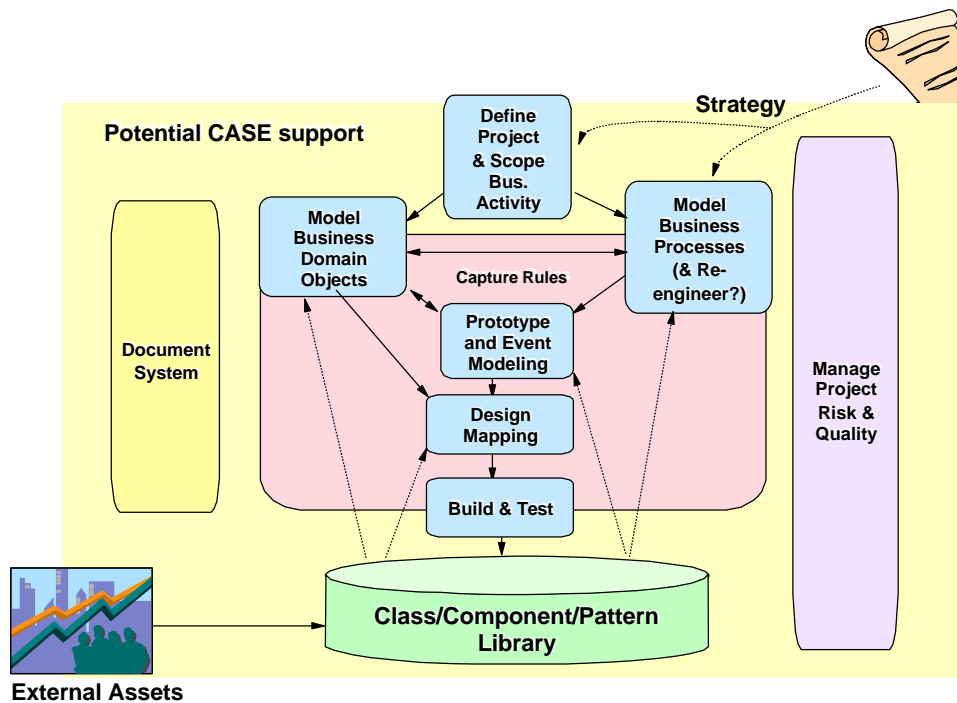
## The *Inspired* Method

### Introduction and scope

This chapter is a brief overview of the *Inspired* system delivery method. It aims to provide a high level view of the major concepts and their interrelationship. The scope of this discussion is limited to the activities which would normally form part of a system delivery project. We are assuming that a range of strategic activities have preceded this stage, including the definition of the business objectives, prioritization of projects, and initial definition of scope. In some cases, there will be a need to perform extensive enterprise modeling and business engineering (or process improvement) before the computer systems stage of the project is reached.

This overview does not aim to be comprehensive and there are many subtleties within individual steps and processes not addressed here. Treatment of these will be left for further detailed chapters. Figure 3.1 shows the major processes and steps in the method. This gives the impression that the method is linear, whereas it is, in fact, iterative. You should read the diagram as a series of iterations, which occur a limited number of times until a given deliverable has reached a certain *state* or level of completion, at which point the ensuing activity will commence. Even then, it is entirely possible to loop back to perform earlier analyses in more detail or to clarify issues which were not completely resolved in earlier operations. The idea is that by the end of each “phase” the deliverables are at a certain minimum level of completion.

Various deliverables and models will be created at a high level and then expanded in terms of detail as the lifecycle progresses. There are no major discontinuities: users should experience a smooth increase in the level of detail, accuracy and fidelity of the models as analysis progresses.



### 3.1 - Inspired Method Overview

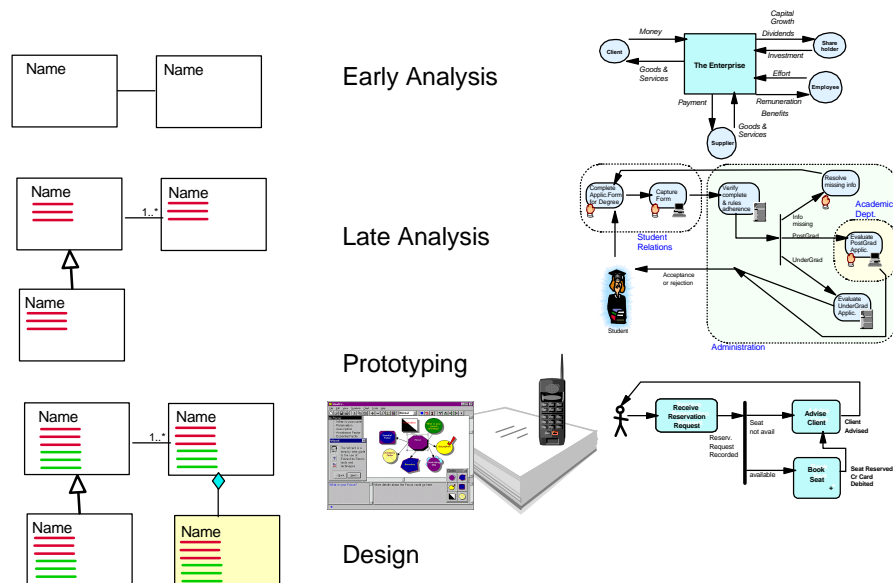
The first step involves the definition of the project and the scoping of the business activity of interest. Project Definition is performed in accordance with the guidelines in the book *Managing Information Technology Projects* (McLeod and Smith, *Course Technologies* (International Thomson Press), 1996). We discuss scoping the business activity briefly in this method overview and in more depth in a later chapter. Output from this step includes a *project definition* and a *context diagram*. The project should also be managed in accordance with other guidelines in the McLeod/Smith text related to estimating, risk assessment and control, change management and quality assurance.

The following step involves the definition of the initial *business domain object model* detailing the entities in the business or organizational domain. This model will be progressively expanded later to include full attributes of a complete object model for the entire application. At this stage, it will include

only the business objects, their relationships and their attributes. Behaviours are deferred until later stages. Examples of business objects would be Customers, Products, Orders etc.

Enterprise and business process modeling can commence in parallel to the business object modeling. This involves examining and modeling the key business processes within the scope of the project. Analysis will include functions which are currently computerised, as well as those which are not. It seeks to build a complete picture of how key business events in the environment are handled by the organization. In the process, we often examine the “value chain” or "value network" Involved - i.e. how the various activities add value in transforming the inputs from the environment into desirable outputs from the organization. Examples of business processes include the processing of a sale in a retail environment, or the processing of a policy application in an insurance company. The outputs from this step involve a *stakeholder model* and a collection of *business process models*.

Should it be felt that the current business processes are ineffectual or that they do not take sufficient advantage of opportunities provided by technology, then clients may opt to undertake a redesign of the business processes. The techniques employed allow easy assessment of a variety of competing scenarios. Other enhancements also encourage the building of business processes which are self-optimizing once implemented.



### 3.2 - Increasing Fidelity with Progress Through the Lifecycle

During the preceding two steps (Business object modeling and business process modeling), a start is made on the building of a dictionary or

repository containing details of objects, their attributes and the characteristics of these attributes. For example, we may know about a Customer object that has a telephone number attribute which is an integer type. As we discover this type of information it will be captured and maintained in the *dictionary/repository*. The repository will be updated in all subsequent steps, gradually becoming more comprehensive, detailed and accurate.

Once we have a business object model and the business process models, we can commence prototyping and event modeling. Prototyping involves building representative interfaces to the system, including screens and reports, as well as considering the flow of the system under various *scenarios*. It allows users to see what the system will look like and how it will behave without the full investment in development of the business logic and production system controls. The prototype should clarify requirements, validate the system “look and feel” and capture more detail in the object attributes for inclusion in the repository. We will often only prototype ‘kinds of things’, not every instance - for example, one domain object maintenance function, one report, one interactive query etc. We can also make use of existing or newly defined patterns to rapidly create consistent functional interfaces which are standards conformant.

A further use of prototyping is to determine the feasibility, complexity or best approach to follow when using new technology. This can reduce risk and save effort in later stages of the project.

*Event (or in UML terms, activity) models* are built which are similar in concept to the business process models, but now concentrating on the computerised aspects of the proposed system and linking these tightly to the business object model. Events are recorded which show the changes in state of underlying objects. For example, a withdrawal operation may affect a customer account object, causing its state to become overdrawn. These models help to express the exact behaviour required of the new system, and to verify the ability of the business object model to support these requirements. During the process, the object model will normally be extended and enriched. Outputs from this stage include the *prototypes* and *event models*. During this stage, we may also capture various business policies and algorithms in the form of *business rules*, which can be attached to the other models.

Once requirements are well understood, we can move to the design mapping task. This involves the apportioning of responsibilities for the various behaviours required (as identified in the dynamic event models) to the classes in the business object model. Also, we will translate the overall logic of the event model and associated rules to reside in a layer of the software

application, generally termed the controller or business logic layer. This separation allows business logic to be expressed at a higher level, making for easier maintenance in future, as well as facilitating distribution of function (e.g. in a client server environment).

User interface requirements (as expressed in the prototype) are mapped to a third layer - the user interface layer (sometimes also called the view). Keeping these in a separate layer allows flexibility in having multiple interface styles for a single application e.g. for in-house use via a GUI such as Windows, or for use via an Internet browser. The design step can also involve creating interfaces to existing computer systems or data files. Outputs from this stage include a *three layer application model* which can be expressed in an expanded object model or using UML package diagrams. This will include the business objects (now with behaviours as well as attributes), objects in the business logic layer which manage transactions and business events, commensurate with the business rules, and objects which create the user interface.

This design can be directly implemented in object oriented tools such as Smalltalk, Java or C++ coupled with an object oriented database. Where the data will reside in a non-object database (e.g. Oracle or Sybase relational systems), business object structures must be mapped to relational tables. This involves additional work and can impact performance of the resulting system negatively. Designers can perform this activity themselves, but since it is nontrivial and fairly generic, they could also make use of a growing number of *frameworks* available from vendors to accomplish the mapping effectively. Examples include TopLink™ and IBM's Object Extender™.

The design model is implemented by writing code as well as customizing and assembling components sourced from outside the project. These could include various interface widgets, such as Microsoft ActiveX Controls™ or Java Beans™; or business level objects such as Enterprise Java Beans™. This is followed with comprehensive testing.

Documentation of the system (in hard copy or electronic form) is built incrementally during development and is finalised during testing.

## **Project Definition and Scoping**

The objective of this step is to properly define the project including the following aspects:

- Title
- Sponsor (who is paying for it and will derive main benefits)

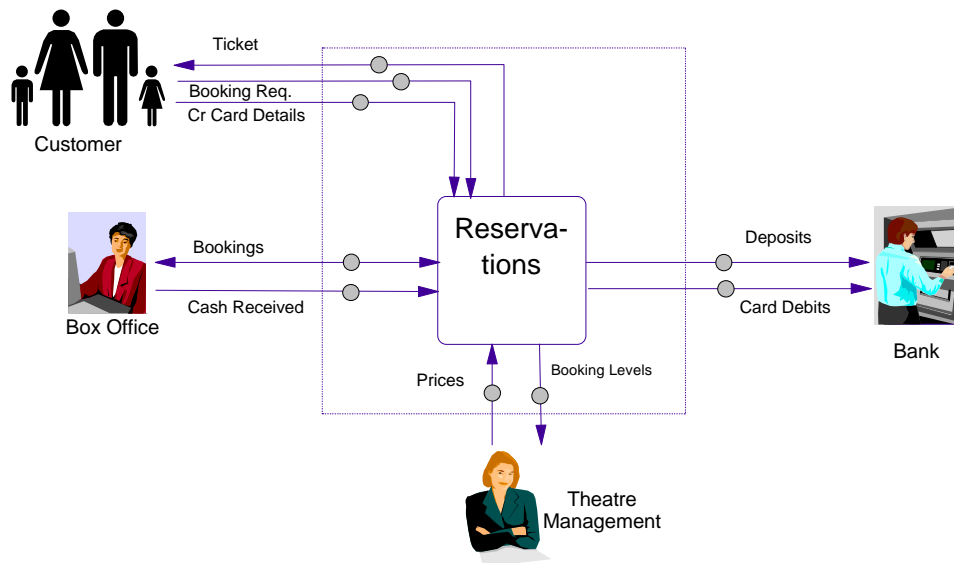
- Overall goal
- Objectives
- Relative priorities of quality, schedule and cost
- Assumptions
- Terms of reference
- Business deadline
- Proposed budget
- Related Projects
- Any legal, moral or ethical issues

The Mcleod/Smith text mentioned previously provides a Project Definition Form to assist with the above, as well as providing guidance in the respective items and their completion.

A *context diagram* is usually prepared. This gives us a concise way to express the relationship of the proposed business activity to its environment. It includes the proposed business activity as a single central box, surrounded by other symbols representing parties and things with which it interacts. These may be various stakeholders, people, organizational units, or other systems. They can be internal to or external to the organization. Connecting the proposed business activity and parties are arrows indicating the movement of data/documents and other artifacts into and out of the business activity. These are labeled to indicate what information or item is moving. A boundary is often drawn to show which things are considered to be under the control of the business activity designers, and which are not. A node on a flow line outside the boundary would indicate a format which the team would be obliged to adhere to, while one inside the boundary would be one that the team has the discretion to design. Figure 3.2 shows a simplified context diagram for theatre reservations.

We can use symbols familiar to the user community, as is done in presentations or the rich pictures technique of soft systems modeling. If we are using CASE tools, we may restrict ourselves to symbols available, such as UML Actors from Use Case Diagramming.

The context diagram is a valuable starting point for analysis, and is also useful to perform quantitative estimating of the likely effort required for the project. This can be done quantitatively at later stages of the project, using Function Points, for example.



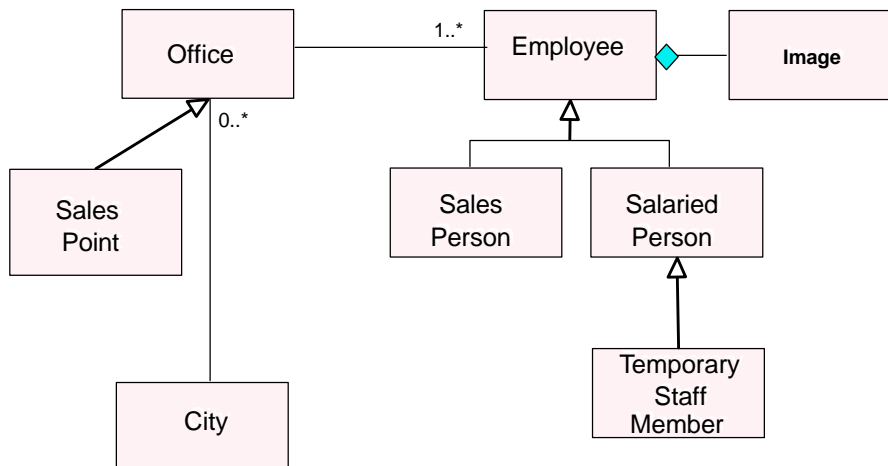
3.3 - Sample Context Diagram

Where the project will result in changes to, or requires the installation of, new technology infrastructure, we will also prepare a *technical environment model*, which depicts graphically what the various components are in the technical environment, including hardware platforms (e.g. PC or mainframe), software (e.g. Operating system, network control, database) and how the various components are interconnected (communications network and protocols). The diagram will normally also indicate numbers of relevant equipment types and the physical geographic distribution of the components. These models are also described in the McLeod/Smith text. Sophisticated organizations may also be using an architecture based approach to the management of their business, applications, information and technology infrastructure. *Inspired* supports such planning, management and evolution by providing comprehensive architecture frameworks and tools which address these issues. These will be the subject of a forthcoming text on Business Advantage through Architectures. In the interim, consult the web site at: [www.inspired.org](http://www.inspired.org) for further details.

## Initial Business Domain Modeling

This is done to gain an understanding of the objects inherent in the business domain, without regard for functionality required. Business objects provide a stable base for the creation of flexible systems. We use a technique derived from the Unified Modeling Language (UML) devised by the “three amigos” -

Rumbaugh, Booch and Jacobson now all at Rational Corporation. UML has been adopted by the Object Management Group (OMG) as a standard. We include some extensions based on advanced work by James Odell. The technique is easy for commercial developers with an information engineering background to grasp and we have found that it scales well in practice. It is also supported by a large number of CASE tools.



### 3.4 - Sample Business Object Domain Model

The object model comprises rectangles representing objects or classes of things. These are interconnected by relationships. Three types of relationship are shown:

- *Inheritance*, where one type is derived from another e.g. Salesperson is derived from Employee
- *Containment* or embedding e.g. where the Employee object contains an Image (which could be a digital photograph of the person)
- *Associations*, where one object simply knows about another one. e.g. Employee knows which Office he/she works in

Relationships are normally named to make the model more meaningful, and to distinguish between relationships occurring between the same two objects. The last two types of relationship will also have multiplicities or ratios. These are indicated by the notation at the end of the line. For example, a City has 0 or more Offices. It is possible for object types to be related to themselves. This normally occurs where we are building a hierarchy: as in a reporting

structure within an organization, or the bill of materials which makes up a product.

Relationships are inherited along with attributes and behaviour. Subclasses can participate in relationships in addition to those inherited from the parent. Generally, circular relationships should be avoided. The modeling technique will allow you to express them, but most software environments (particularly databases) will have problems with them, and they are very rare in the real world, so you should check that it is actually an accurate model if you have one.

## **Business Process Modeling and Reengineering**

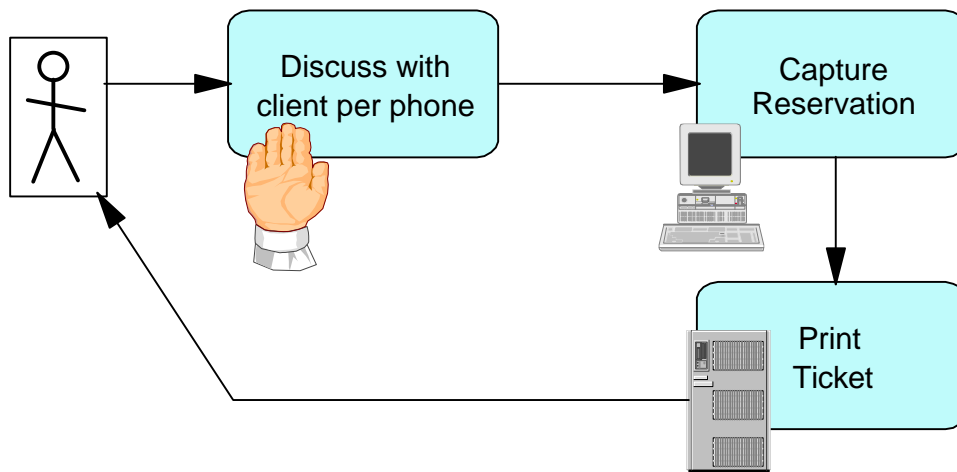
We normally start this activity by drawing a single stakeholder model. This depicts the enterprise or business unit under examination from the perspective of all the agents in the environment that have an interest in the continued functioning and well-being of the unit. These normally include customers, staff, shareholders, and suppliers. Next we identify the value chains or networks which link the resources provided to the enterprise with the value derived as output by the various parties. We examine the way in which inputs are transformed to outputs and, in particular, how the value generated by the enterprise is added. This stage uses techniques popularised by Michael Porter.

Each discrete business event can become the focus of one or more business process models. These detail the processes which occur along the value chain (both value adding and non-value adding) before the output is generated. We distinguish in the model between three types of processes:

- *Manual processes* which are not accomplished through any automation. The receipt of a telephone call by a clerk could be an example
- *Computer supported processes*, where there is computerised support, but human intervention is required. An example could be the logging of a support call into a tracking system by an operator
- *Fully automated processes*, where a computer system operates without intervention. An example would be the batch printing of statements at the end of the month, using information already present on the client accounts

Processes are linked to show their dependencies. They may occur in parallel if this is possible in the business. Boundaries can be used to show where activity is physically taking place e.g. at head office or a branch. Groupings could also be logical, showing departments within an enterprise. Resources, volumes

and timings may be added to the process models to gain an appreciation of the processing demands and performance requirements of technology.



### 3.5 - Business Process Model

Where existing processes are deemed inadequate for whatever reason (inefficiency, ineffectiveness, poor quality, cost etc.) we may embark upon reengineering. This examines the outputs required and the processes to derive new, innovative, creative and often technology-supported ways to achieve these outputs more effectively, more efficiently, or with higher quality. In making radical process design changes, we can draw on the work done by Michael Hammer and later adherents.

We should be cautious, however, since reengineering involves considerable risk for the organization and stress for those affected. Great care must be taken to involve all participants, to thoroughly examine existing processes to understand the real problems, and to examine alternatives to the extent that we can prove that they are superior and obtain the commitment of all parties involved to change to the new way of doing things. Change cannot be done too quickly or too often without serious consequences for morale, productivity and core organizational competencies. Reengineering should therefore be an infrequent occurrence. Where it is done, we encourage the use of simulation and prototypes/pilots to prove viability before widespread deployment.

We can gain much from a less radical approach whereby we instrument processes so that we can measure their operation and pinpoint problem areas. In this way we can make small incremental improvements over time, which can cumulatively yield large results. This is much less traumatic for the

organization. Our approach encourages building this capability into all processes which are automated in the project and into any processes which are reengineered. The implementation involves the use of techniques of statistical process control and the Japanese *kaizen* (continuous improvement) philosophy.

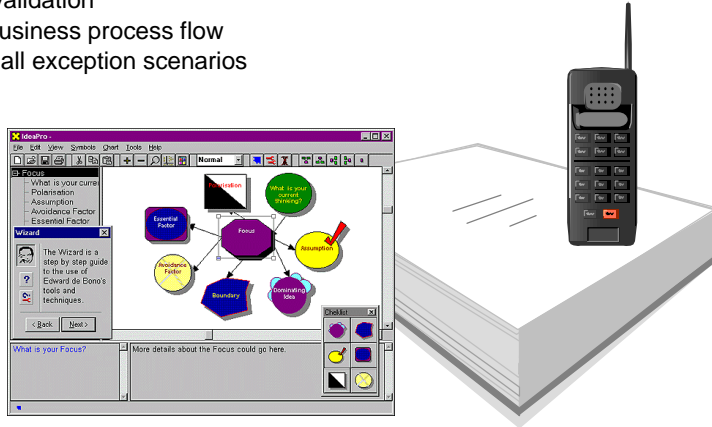
## Prototyping and Event Modeling

Prototyping is a technique where we “mock up” the system to see what it will look like before we undertake the real construction. Like an architect’s paper model of a building, it is something which allows you to validate requirements and design choices. It assists us in refining functional requirements, in generating a good user interface, and in discovering more attributes of objects for inclusion in the object model. It can also lead to the discovery of business rules which can be recorded for later implementation.

Like the architect’s model, however, the prototype does not have the strength or plumbing of a real system. There is little of the validation, recovery, archival, help facilities etc. which a real system will require. Users must be educated to understand that the prototype is an opportunity to explore requirements and design options at relatively low cost before major resources are committed to full scale construction. They should also understand that the prototype is not a real system and that there will be a delay between seeing this and having the real system available. They should understand that changes after the system is built for real will be much more difficult and costly than during the prototyping stage.

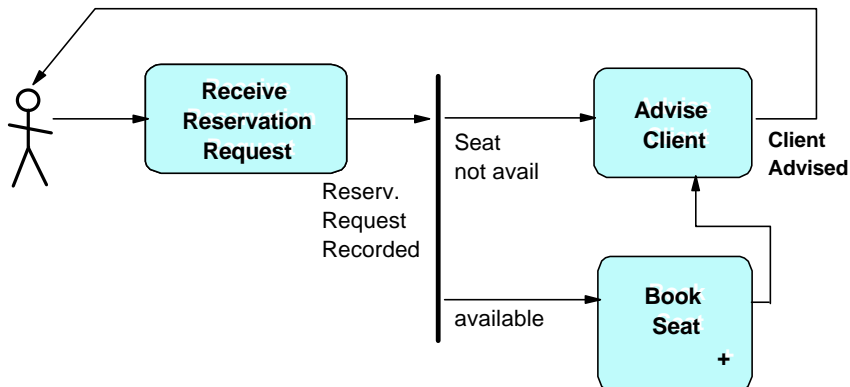
Concurrent with prototyping, we build event models for key processes in the system. Processes of a generic nature (e.g. the maintenance of tables (add, edit, delete, query, print)) can be designed just once and reused for all similar requirements. *Pattern libraries* can assist us in finding many generic solutions. The event models show the triggering external operations (which can be transactions or time related), the sequence of operations which occur within the system to respond to these stimuli, and the outputs generated. In addition, events are recorded where the operations change the state of underlying objects. This explicitly links the event model to the previously generated object model. In this way the object model is verified and, where necessary, extended.

- Refine functionality
- Define User Interface
- Refine attributes
- Clarify validation
- Verify business process flow
- Identify all exception scenarios



### 3.6 - Prototyping

The look and feel of the event models is very similar to the higher level business models, but with more rigour. It is well suited to implementation via event driven software, the three tier design architecture and/or work flow packages. Processes can be designed so that operations can execute in parallel and that asynchronous operations can occur (the initiator does not have to wait for the requested activity to finish before continuing). Event models can be enhanced with rules which embody business policy or important algorithms. Where necessary a narrative specification or reference can be recorded for an operation box.



### 3.7 - Sample Event Model

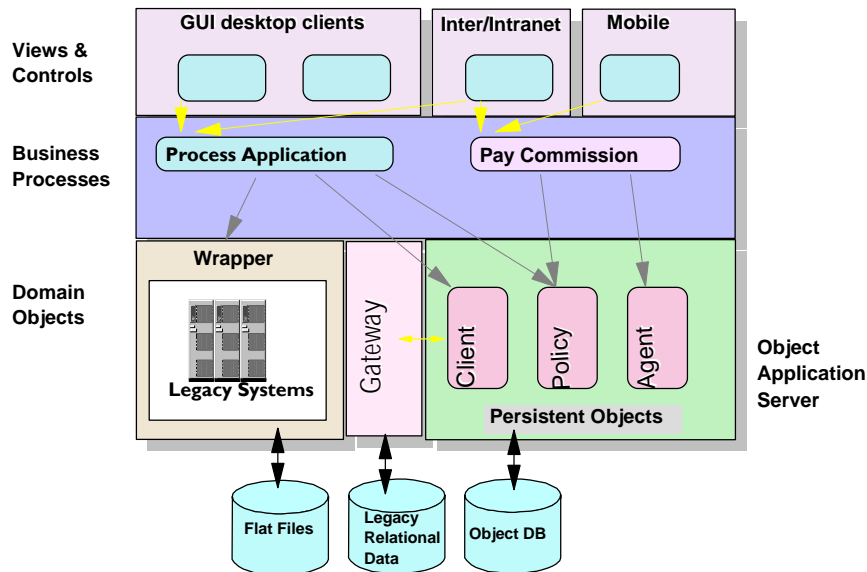
## Design Mapping

### Layered Architecture

In mapping to the technology available, we will normally seek to implement a model with at least three layers:

- The *business object layer (model)* which reflects as accurate as possible a model of the objects found in the problem domain. These objects should, as far as possible, have little awareness of each other. This increases the reliability of the implementation and the flexibility in the face of change. Typical objects in this layer will include Customers, Products, Employees and the like. This layer is normally made persistent via storage in a database
- The *business process or logic layer (controller)* which is responsible for coordinating activity between the user interface(s) and the underlying model. This will normally have two sub-levels. The first deals with the manipulation of the interface itself for actions which have no lasting impact (e.g. shrinking or expanding a window on a GUI screen). The second deals with the logic to handle a business event (e.g. handling a client payment once the data is validated and the user clicks the OK button). Logic in the latter layer may be invoked from multiple points in the technical layer. This allows different user interface elements to drive the same business logic. For example, we could have a menu selection, a tool bar icon and a text command trigger the same processing.
- The *view layer* consists of objects implementing the user interface. It normally comprises predefined components such as windows, sliders, buttons, menus and the like. These are normally obtained as components which are then dropped onto a canvas to paint the desired interface. They may be customized in terms of size, font, colour etc. An alternate view may provide a text/command interface or a batch interface to the same underlying business logic

We may need additional layers depending upon the technical environment. For example, the business logic may need to be split between a local and a remote machine or we may need a layer which wraps a legacy database or system and makes this accessible to the model layer.



3.8 - Layered Application Architecture

## Object Technology

The cleanest mapping is to pure object technology such as represented by a language such as Smalltalk, Eiffel or Java in combination with an object database such as Gemstone™, Versant™ or Object Store™. In this case the mapping proceeds as follows:

- Prototypes are evolved into the view layer, making use of available components and frameworks from vendors.
- Rules and the high-level logic from the event models are implemented in the business logic (controller) layer. This consists mainly of obtaining activity from the view and sending messages to appropriate objects in the model then passing results back to the view, managing transaction integrity and security as appropriate
- The business object portion of the object model is implemented in the business object (model) layer. This normally contains the persistent objects which will remain accessible over time and which will be sharable between users of the operational system and between systems. This persistence is best achieved through an object database management system.

## **Relational Technology**

Where the mapping is to relational technology, it is common to have the business object layer stored in the relational environment, while still making use of objects in the client software (i.e. in the business logic and view layer). Depending upon the capabilities of the tools available, we may have only a view which is object oriented, with a procedural language behind it.

The best strategy for these environments is to convert the object model to the closest relational equivalent before doing detailed design work. This will allow the use of components which understand the relational model (e.g. PowerBuilder™ data windows or Delphi™ Master/detail controls). Users should be aware that there are considerable compromises in implementing the object model on relational technology. The accuracy of the simulation declines and performance can be problematic for complex data structures. Where the data is simple and well understood, the overhead is tolerable. Middleware products such as TopLink™ and UniSQL™ can provide a fairly object view of relational data. Frameworks such as the Object Extender™, part of the IBM Visualage™ family, provide a fairly seamless and competent mapping between object processing and relational storage.

Where it is planned to migrate to object database, the implementation should aim to provide a consistent and stable messaging interface to the data, which may currently reside in relational or conventional technology, but which will in future move into an object database. This will allow legacy components to be selectively and easily “unplugged” and replaced over time without disrupting the applications which use them. A good degree of encapsulation can be achieved in relational environments by the use of stored procedures, but the languages provided by the various vendors are fairly proprietary, limiting portability. An emerging solution to this problem is the growing support for stored procedures written in Java, making them portable.

## **Business Rules**

Starting early in the lifecycle, and continuing until well into design, we may uncover and need to document a variety of business rules. We have a flexible approach to these, considering a rule to be anything which we need over and above the graphic models to fully define the system requirements. Examples of rules include:

- How certain decisions are reached based upon business policy e.g. When to order stock, who should receive discount - these are normally expressed as IF [condition] THEN [action] style rules

- How values are calculated or derived e.g. How to raise interest on an outstanding loan. These may be expressed as a formula or algorithm
- References to recognised procedures, laws or standards e.g. The prescribed method for calculating a VAT submission, or determining a medical aid deduction
- Integrity rules which guard the accuracy and integrity of our domain knowledge e.g. Ensuring that an item can only be ordered when there is a valid product code and a recognised supplier, or that dependents information may only be captured where there is a valid employee

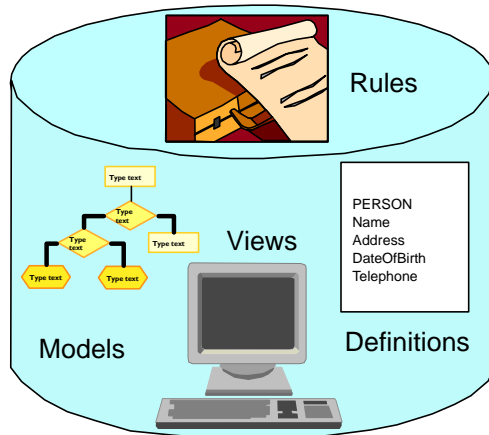
Rules can be expressed in a near English pseudocode, making use of domain class names and attribute names to enhance clarity. Alternately, we can use a formal language such as Object Constraint Language (OCL) sanctioned as an addendum to UML. In either case, the rules are cross referenced from the models and should be stored in the repository or implementation rule base.

## **Repository and CASE**

Where the applications become complex (most places!) and there are multiple project teams, it can become problematic to keep models up to date. We should, as a minimum, keep the data dictionary or repository in an electronic form which allows easy updating and flexible query and Reporting. Various computer aided systems engineering (CASE) tools are available to support the modeling activity. These range from fairly simplistic tools which are essentially diagram editors and do not understand the models which they hold and will not integrate changes across models or views, to very sophisticated tools which may integrate multiple perspectives, support multiple users and projects, manage changes and generate production quality database schemas and code.

Really high-end tools will even support partitioning across a variety of platforms and the deployment of the production application. Key requirements for a suitable tool would include the following:

- It should support the modeling approach chosen
- It should have a comprehensive and integrated repository
- It should provide consistency across multiple model types and views
- It should be open - the repository should be in a form which is intelligible to other products, or provide export facilities



- Rules
  - Business Process/Statutory
  - Integrity
  - Derivation

- Models
  - Static
  - Dynamic
  - Object
- Views
  - Inputs/Outputs
  - Interfaces
  - Security
- Definitions
  - Classes
  - Attributes
  - Methods

### 3.9 - Repository Contents

- It should ideally integrate with the development tools providing
  - Database schema generation
  - Code generation from business rules
  - Prototyping (where the downstream tools do not provide this)
  - Round trip engineering - the ability to alter the generated code and feed these changes back into the CASE models

With CASE, you tend to get what you pay for. Some tools are cheap but are frustrating in their lack of facilities, while others are comprehensive and capable, but prohibitively expensive. We also caution that CASE in itself does not lead to high productivity and quality, unless it is accompanied by suitable investment, commitment, training, skill and discipline in use.

## Hypermedia and Multimedia Specifications

Just as new technology makes possible new business models and innovative ways of working for our client organizations, it also affects what we can do in crafting systems. A recent innovation is the use of hypermedia and multimedia specifications. Within our method, we encourage the rich interlinking of models via hyperlinks. This can, with standard Internet technologies, now be

accomplished across tools and distances. It is quite feasible to have a prototype screen in CASE Tool A in Johannesburg, which has a hyperlink from a field on the screen to an object model residing in a repository in New York. There are some serious integrity management challenges, of course!

Often, in the past, subtle requirements expressed by users in interviews with analysts were lost in the translation to models passed on to designers and developers, only to re-emerge at the user testing or implementation stage, with attendant high cost. A technique which can overcome this is to videotape interviews with key users during requirements definition (with their permission, of course) and store these videos digitally, hyperlinked to the relevant models. This provides a unique opportunity for a programmer, for example, to replay the specification process and clarify concepts. It has been used to great success in the development of new air traffic control systems in Europe.

## **Facilitation**

At many points in the lifecycle, we will make use of facilitation (otherwise known as Joint Application Development (JAD)). This is a technique of holding intensive facilitated sessions with all relevant parties present to achieve the following aims:

- Reach consensus on important decisions, models and approaches
- Increase ownership of the results
- Improve accuracy of requirements and efficacy of design
- Compress the lifecycle in terms of calendar time

Facilitation is suitable for project scoping, object modeling, business process modeling, prototyping and event modeling activities. It can also be used for resolving design questions and planning testing and implementation. JAD will be covered in a later chapter.

## **Existing Systems**

We seldom have a “green field” situation with no previous systems or data to worry about. More often than not, we will need to interface to older systems capabilities and access data created by such systems. The preferred technique to deal with these is to create a “wrapper” which provides an object oriented messaging interface on the outside, while communicating with the legacy

system or accessing the legacy data in its own preferred way on the inside. The actual interface to the product can include calls, passing parameters, executing programs or jobs, simulating screen transactions etc.

The creation of such wrappers is not trivial, but it can make functionality available which would be difficult or expensive to re-implement. Again, if the interface is maintained consistently, the problematic components can be gradually phased out and replaced without impact on the newer applications.

## **Workflow**

Workflow can be both a concept and a software product. In the case of the former it is a concept whereby work flowing through an organization can be understood, scheduled, monitored and made more efficient. As a software utility, workflow packages provide facilities to control the routing of transactions, movement of files and data between processes, and the invocation of the necessary services to perform the next required step in the overall process. Sophisticated products also provide extensive capabilities for gathering statistics, analysing these and balancing workload across resources.

Using the design model which we described earlier, it is fairly easy to use a workflow engine to implement the high level logic of a business process or event model. Some packages will also allow us to leverage legacy technology, treating it as a step within the overall process.

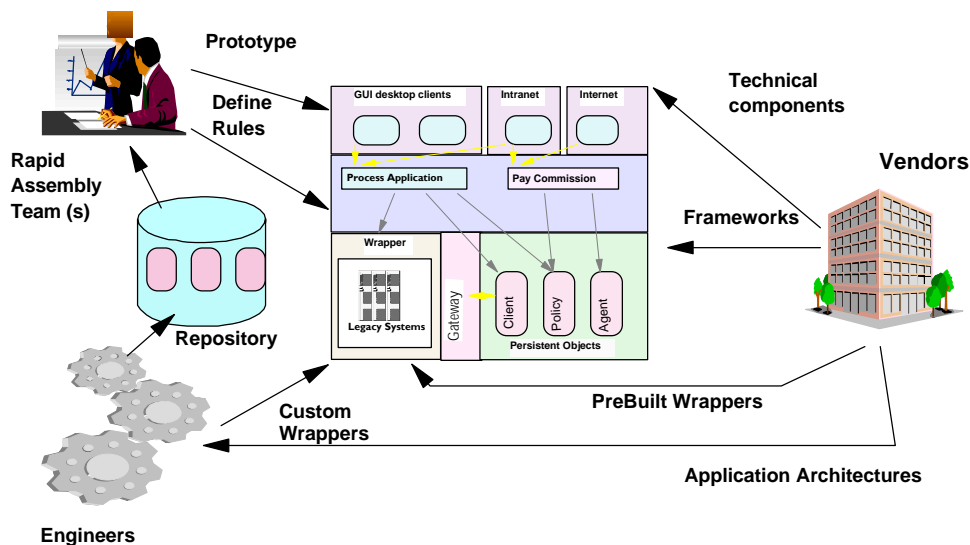
## **Internet Deployment**

The Internet (and use of Internet technology on private networks, an *intranet*) offers great opportunities but also challenges. The design mapping discussed above can be easily used to target these delivery platforms. The three layer architecture maps well to client-server, which the Internet represents. Essentially the browser and forms become the view layer of the application. If the browser supports the loading of applets (e.g. via Java Bean™ or ActiveX™ controls), then some business logic can also execute on the client machine. In theory, data could also reside remotely on the client platform. With the current state of these technologies such data would be very insecure and this should probably not be contemplated except for personal data maintained by the workstation operator.

## Components and Frameworks

The method promotes the use of components and frameworks obtained from external vendors and internal sources as follows:

- A suitable framework which implements the three tier design model should be found and acquired or built. This will provide application programmers with a standard way of handling the user interface, communications, layering, handling events, and packaging of the application for delivery



### 3.10 - The Big Picture

- Vendor graphical user interface, communications and database libraries should be used. Specialised components for other aspects of the technical infrastructure, such as communications, security or interfacing to special devices, can be added.
- Application class libraries can be incorporated into the business object model at the requirements stage, providing a rapid leg up to comprehensive and viable models
- Legacy application wrappers for popular products can be obtained from vendors or third parties
- Application developers should, as far as possible, concentrate on the delivery of excellent user interfaces and rapid implementation or

adaptation of business functionality. Most of their work will be with the view and controller scripts. A second group can engineer components for reuse at a corporate level, or obtain suitable components from outside the organization

## **Quality Assurance**

At each step of the project, there are defined deliverables which should be refined or finalised. The team should adhere to guidelines for the content, format and presentation of these. Good examples should be collected and catalogued so that new staff will be able to see what is expected and what good work looks like. It is beyond the scope of this chapter to detail the quality philosophy here, so we will make this the focus of a later chapter. Additional information can be found in the McLeod/Smith text mentioned earlier.

## **References and Further Reading**

Avison DE, Fitzgerald BA, 1995. Information Systems Development: Methodologies, Techniques and Tools. Second Edition. McGraw-Hill, 1995.

Avison DE, Fitzgerald G, 1988. Information systems development: current themes and future directions, Information and Software Technology, October 1988.

Avison DE, Fitzgerald G, 1986. Information Systems Development Methodologies for Microcomputer Applications, Management Forum, Sept 1986.

Avison DE, Wood-harper AT, 1990. Multiview: An Exploration in Information Systems Development, Blackwell Scientific Publishers, Oxford 1990.

Avison DE, Wood-harper AT, 1991. Information Systems Development Research: An Exploration of Ideas in Practice, The Computer Journal Vol 34 No 2, 1991, pp 98-112.

Avison DE, Fitzgerald G, 1988. Information Systems Development Methodologies, Techniques and Tools, Blackwell Scientific, 1990. Booch, Grady, 1994, Object Oriented Analysis and Design with Applications (Second Edition), Benjamin Cummings

Bourne, John, 1992, Object Oriented Engineering: Building Engineering Systems Using Smalltalk-80. Richard D Irwin, Homewood, Ill.

Brinkkemper S, Lyytinen K & Welk J (Editors), 1996, Method Engineering: Principles of method construction and tool support, Chapman & Hall, 1996.

Butler-Cox 1988, Computer Aided Software Engineering, Butler-Cox Research Report 67, Dec 1988

Card D N, Clark T L, Berg R A 1987, Improving software quality and productivity, Systems Vol 29 No 5 June 1987 pp 235-241

Coad P, Yourdon E 1990, Object-Oriented Analysis, Yourdon Press 1990

Colter M A 1984, A Comparative Examination of Systems Analysis Techniques, MIS Quarterly Vol 8 No 1 March 1984 pp 51-66

Colter MA 1982, Evolution of the Structured Methodologies in Advanced Systems Development Techniques, John Wiley 1982 pp 73-96

Comcon, 1987, The GOLD Project Management Methodology, Comcon (Pty) Ltd, 1987

Comcon, 1987, A Project Manager's Guide to System Development Projects, Comcon (Pty) Ltd, 1987

Downs E, Clare P & Coe I, 1988, Structured Systems Analysis and Design Method - Application and Context, Prentice Hall International, UK, 1981

Graham, Ian, 1993, Object Oriented Methods, Addison Wesley

Graham I, 1994. Migrating to Object Technology, Addison-Wesley, Wokingham, U.K. 1994.

Hackathorn R D, Karimi J 1988, A Framework for Comparing Information Engineering Methods, MIS Quarterly June 1988 pp 203-220

Humphrey, Watts S 1995, A Discipline for Software Engineering, Addison Wesley, 1995

IEEE, 1987, Draft Standard for Software Project Management Plans - IEEE p1058, Institute of Electrical & Electronic Engineers Inc., 1987

Infomet, 1988, The Infomet Methodology, Infomet (Pty) Ltd, 1988

Jayaratra N, 1994. Understanding and Evaluating Methodologies. NIMSAD a Systemic Framework, McGraw-Hill, 1994.

Jones L H, Kydd C T 1988, An Information Processing Framework for Understanding Success & Failure of MIS Methodologies, Information & Management Vol 15 1988 pp 263-271

Mahmood M A 1987, System development methods - a comparative investigation, MIS Quarterly Vol 11 No 3 Sept 1987 pp 293-311

Martin, James and Odell, James 1993, Principles of Object Oriented Analysis and Design, Prentice Hall, Englewood Cliffs, NJ

Martin, James & Finkelstein, Clive, 1981, Information Engineering, Volume 1, Savant Research Studies, Carnforth, Lancashire, U.K.

Martin, James, 1987, Recommended Diagramming Standards for Analysts and Programmers, Prentice Hall, Englewood Cliffs, NJ

McLeod G, 1992. A Model for Representation, Integration and Management of Methods. South African Computer Journal.

McLeod G, 1993. A generic method model for the Representation, Comparison, Integration and Management of Methods, Proceedings: 1st International Conference on Information Systems, Henley, U.K., 1993.

McLeod G, 1995. MetaTool: A tool for the engineering and management of methods, Proceedings: 1st International conference on Meta-CASE. Sunderland, U.K., Jan 1995.

Olle T W, Sol H G, Verrijn-Stuart A A, (EDS) 1982, Information Systems Design Methodologies: A Comparative Review, North Holland, Amsterdam 1982

This page intentionally left blank.