White Paper

Designer/2000 and Object Technology Current Features and Future Directions

VERSION 1.0: MAY 1996

Contents

Introduction

For some people and organisations, the use of Object Technology (OT) or Object Oriented (OO) tools, techniques and so on, has become a goal in itself. The promise of OT– greatly improved productivity, quality and maintainability – is so alluring that it is sometimes pursued (and promoted) with scant regard to its applicability or even feasibility in a particular set of circumstances. Of course much of this is due to the hyperbolic instincts of our industry. However, in the case of OT, the claims may not be as exaggerated as usual, or at least they may not turn out to have been such exaggerations if we can successfully chart a pragmatic path from current technologies to the object-based vision of the future of software development.

Therein lies the challenge. In embedded and real-time applications OT is already established as the de facto implementation route. In specific applications in commerce, particularly in telecoms and in finance OO projects are achieving significant success. If this value is to be translated into widespread commercial acceptability there must be a proven path from here to there, and that path must be tenable not just for the mostly mythical 'green field' projects. The path must enable us to migrate all the dimensions of current systems:

- data
- designs
- skills.

It is not sufficient that OO solutions should be better, not even that they should be provably better. They must be achievable by the broad base of development organisations and achievable at a commercially viable cost. Effective migration and evolution strategies are essential to justifying the cost.

Oracle's technology, for database, tools and applications have achieved pre-eminent success in the world of relational-based client-server computing. Oracle has therefore a significant stake in its customers achieving success with OT. A significant stake in migration in all the three dimensions

This paper discusses how Designer/2000 helps organisations to preserve their investment in skills and designs while making the migration. The migration of data and applications that built on that data are covered by other product groups in Oracle corporation, for example the evolution of Oracle's server technology through the universal server towards explicit object support in Oracle 8

Designer/2000 Releases 1 and 2

Reuse: Encapsulation, Aggregation and Templates

GUI products often provide the capability to define 'classes' or 'types', usually collections of properties that are propagated to all instances of that class. These classes are not usually factory classes, that is to say you cannot create an instance of them. They might more realistically be called property templates that can be applied after the instance has been created. This is a useful feature and saves time and improves consistency - provided the developer can find them and provided the developer remembers (or chooses) to use them. Designer/2000 uses property templates, but it uses them in a more structured way.

The main code generation targets for Designer/2000 are not pure OO tools – Visual Basic, Oracle Forms and others such as HTML. Object models can be used to design programs in such tools, indeed some of them, such as Oracle Forms, use such techniques internally.

What Designer/2000 seeks to do though, is to raise the productivity of developers using these tools and to give them greater choice, later in the lifecycle, as to which target they should deploy. It can only do this by specifying the programs at a more abstract level. To achieve this it has its own model which, whilst not purely OO, **does** make extensive use of concepts familiar in OO technology. These concepts include encapsulation, aggregation, overloading and templating. Designer/2000 does this not to rack up points on the 'Object Scorecard' but to deliver consistency, productivity and maintainability to its users through *reuse*.

Each GUI program generated by Designer/2000 is made up of *items*, which are aggregated into *groups* which are aggregated into *zones* which are aggregated into *windows* which are aggregated into a *module*. Each of the target tools has its own word for the levels in the hierarchy, for example Oracle Forms has *blocks* not *zones*, HTML has *pages*, not *windows*. Not all the tools support all the levels, but in essence the hierarchy holds. Most of the base level *items* represent the use of a data item (a column of a table) and are of a given display type, for example check-box or pop-list.

Designer/2000 allows the developer to specify for each table in the database not only its data members (columns) but also its behaviour (integrity rules, derivation rules and also its display type) unified in a single definition – basic encapsulation of data, function and user interface. When a developer creates a UI item she binds it to a data item (column of a table). The item instance inherits the UI behavioural definition, and such of the business rules as have been flagged for client-side implementation. This propagation of property values is sometimes known as property value inheritance. This is **not** inheritance in the OO sense of the word, but nonetheless it provides a powerful mechanism to improve consistency and productivity.

Designer/2000 also allows the developer to specify the programming styleguide for the user interface. It does this in terms of a hierarchy of rules – for modules, windows, zones/blocks, groups and items and in terms of template code and layouts. When a module uses a data item just as it inherits the display type of the underlying data item, it also inherits the appearance and GUI behaviour of that display type from the styleguide.

For example we may have a *Customer* table, and in the *customer* table a column *credit_status.* Let us assume that the status must be '**C**' meaning in credit, '**D**' meaning in debt or '**S**' meaning in suspense. The database definition specifies that the *credit_status* is stored as a single alpha character and is validated on the server for the values 'C', 'D' or 'S'. The data definition also specifies that *credit_status* is normally to be displayed as a radio group with titles **'In Credit'**, **'In Debt'** or **'In Suspense'**. Whenever an item is bound to this column it inherits all this. Additionally it inherits from the styleguide in force how radio groups are to be displayed (vertical or horizontal, decoration, fonts, colours and so on).

Hence to define a new program a Designer/2000 developer only has to identify the required data. All the code for persistency, business rules and user interface are generated from the common definitions which are reused

automatically. The developer can, of course override the inherited definitions, but the simplest path is the path of maximum reuse, hence maximum productivity and maximum consistency.
The terminology used in Designer/2000 is not always the standard OO terminology. Long time users of Designer/2000 and its predecessors, and some have many years' experience, would not welcome wholesale change to the vocabulary with which they are familiar. However the principles that OO has formalised and popularised have influenced the development of Designer/2000 and many significant features of OO thinking are already visible and exploited in Release 1.

Integrating Object Development and a Relational Store

The Designer/2000 C++ Object Layer Generator

Release 1.3 of Designer/2000 includes a new generator, the *C++ Object Layer Generator*[1].
Many C++ developers wish to use relational datastores for their persistent objects because they want industrial strength database capability for the objects they create and they want access to data shared with relational applications. Many of these developers agree that two of the most time-consuming tasks of such a development are devising and maintaining the mapping from C++ classes to relational tables, and developing and maintaining the code to provide persistency of the classes within those tables. The *C++ Object Layer Generator* in Designer/2000 targets precisely these two tasks.

Designer/2000 has long had the capability to model hierarchical specialisation structures such as are necessary for class hierarchies. This is entity relationship (ER) modelling with multi-level sub-typing and inheritance of attributes. These structures only support data members and associations (attributes and relationships) so they are incomplete as object definitions, but they **do** deliver object structure modelling. Designer/2000 supports multiple implementations for mapping these structures onto flat relational schemas. For example: one table per sub-type, one table for all sub-types with typing column and others. Designer/2000 also generates the DDL to create these schemas. It can also reverse engineer relational schemas and allow developers to map their classes onto them.

The *C++ Object Layer Generator* generates the C++ definitions of classes to correspond one-for-one with the hierarchical entity subtypes and generates the code to provide transparent persistency for these classes. It does this regardless of the complexity of the mapping to underlying tables.

Most importantly, underlying schema changes are completely transparent to the C++ programmer provided the class-to-schema mapping is kept up-to-date. And, even after custom behavioural methods have been added to the generated classes, the class definitions can be regenerated if, for example data members or associations (attributes and relationships) change.

[1] *"The Designer/2000 C++ Object Layer Generator, Product Overview", Oracle Corporation 1996*

Component-Based Generation

Reuse has long been a goal of software developers. The first assembly language macro initiated the reuse of code. The first multi-user database initiated reuse of data. Reuse means less coding, hence less testing, fewer bugs, fewer inconsistencies; overall it means better quality. Object technology offers a very serious opportunity to raise achievable levels of reuse. By unifying data and functionality into objects, it delivers a single place for defining the components for reuse. Inheritance and overloading mean that when an object is reused it can be modified to tune it for the particular circumstances. Designer/2000 Release 1 makes extensive use of property-level inheritance to deliver high levels of reuse.

In Release 2, the design-time toolset of Designer/2000 changes significantly. Improved tool integration and interoperability makes users even more productive, especially in RAD environments. But another, potentially more far-reaching, change in Designer/2000 Release 2 is the introduction of reusable **module components**. The property-level inheritance in Release 1 has proven highly successful but what it enables is reuse of very fine-grain components – columns of tables. In large-scale developments there are larger units which are used over and over again. For example an edit facility for the *customer* table with information about the *region* the customer is in, and specific groupings of *delivery address, invoice address, account details* etc.

Release 2 of Designer/2000 allows the developer to define this object as a module component which can be generated as a free standing module (for example in Oracle Forms or Visual Basic) – or it can be reused as part of other module definitions. This is reuse of larger granularity objects, objects more recognisable and useful to the business.

A larger granularity of reuse implies greater productivity, but of course there is a trade-off. The more complex an object is, the less likelihood there is that it will be reusable without modification. This is why the most successful reuse, currently, is of very small objects, for example the UI widgets one can buy on a CD-ROM for $99. This is also why there is so much activity in such groups as the Object Definition Alliance (ODA) and the vertical sector task forces of the Object Management Group (OMG)[2] to define generic reusable but large scale business objects based on the needs of specific industries. Examples of these might include *Customer*, *Account*, *Product*, etc. In time standard business object definitions will no doubt become available. In the meantime developers and designers must seeks other means such as Designer/2000's module components.

The module component in Designer/2000 is a type of module. This means that existing modules can be re-classified as reusable components after they have been built and tested. The benefit of this is that it is not necessary to identify which components will be reusable before development commences in order to gain any benefits. When an opportunity for reuse is recognised, it can be exploited with little additional effort. Object oriented development projects are normally faced with a choice: spend time on a very detailed technical architecture to identify opportunities for reuse, or reuse as you go. The first approach takes time and as the level of detail increases the return on the time invested diminishes. The second approach results in very low levels of reuse, and what reuse there is, is often just 'copy and hack' Designer/2000's module component seeks to reconcile this conflict by providing a mechanism to support effective ad-hoc reuse, as well as planned reuse.

This is reuse of client-side components (GUI behaviour and business rules) to complement the reuse of server-side components (data and business rules) which Designer/2000 Release 1 already delivers.

---

[2] Soley, R.M. (ed.). *Object Management Architecture Guide (revision 2.0)*, Object Management Group, Framingham, USA, 1992.

Beyond Designer/2000 Release 2

User-Accessible Business Modelling

Designer/2000 Release 1 is already delivering remarkable levels of productivity to developers[3], Release 2 goes further both releases exploit many of the fundamental concepts of object technology. However in the releases up to and including Release 2, the business modelling component of Designer/2000 remain very much based on the techniques of Information Engineering[4] and latterly on the process modelling techniques popularised by Business Process Reengineering (BPR[5]).

The emphasis on process modelling has been very successful in maximising user participation in requirements definition. People generally find it more natural to talk about what they **do** rather than what information they use, so they feel more at home in a dialogue about business processes rather than information entities. Because they feel more comfortable they quickly become more involved and the result is a definition of the user requirement that is understood by the users themselves and is so more easily checked for completeness and correctness **-** by the users themselves.

No matter how productive the implementation tools are, without an accurate specification of the requirements the best we can hope for is to build the wrong system quickly. Of course, many successful systems are built precisely this way: through the evolution of prototypes. But such success is inevitab**ly** limited to tactical systems, responding to a clear localised requirement. Scaling to enterprise level requires a way of mapping the tactical onto the strategic. Organisations that believe that information systems need to match overall corporate goals as well as responding to tactical needs recognise the value of business modelling. Within the context of a business model, prototyping projects can meet the tactical need as well as matching the strategic goals.

Process-led business modelling concentrates on the structure of the business: the processes, the organisations, the workflow, **not** on the information. This means that there is a second stage to requirements gathering, the information analysis, then there is the need to tie these together. Clearly a more unified approach is desirable.

Object Oriented Modelling

Object oriented models provide a more unified and more powerful model for process, information and behaviour. However object oriented modelling tools and techniques are primarily rooted in technical systems development; they may be more powerful, but they are no more user-accessible than the structured analysis techniques of the eighties. Also most approaches do not naturally deal with business objects. The desirable scale of object for modelling businesses and for reusing in business systems would be, for example that of *Customer, Account, Product,* etc. Each of these objects may well be an aggregate of simpler objects, but we may not wish to burden users with this complexity. Traditional object modelling does not support the concept of aggregated business objects – which map naturally onto user terminology – as separate from the underlying domain objects – which are more efficient for design.

---

[3] Empirische Untersuchung des Einflusses eines CASE - Werkzeuges auf die Produktivität von Wartungseingriffen in kommerziell genutzte Informationssysteme': Jenns Hoffmann, Institute of Informatics, Technical University of Munich 1995

[4] Martin, J. Information Strategy, Planning and Development, Prentice Hall, 1990

[5] Rummler, G. A. and Brache, A. P (1990). *Improving Performance.* Jossey-Bass, San Francisco.

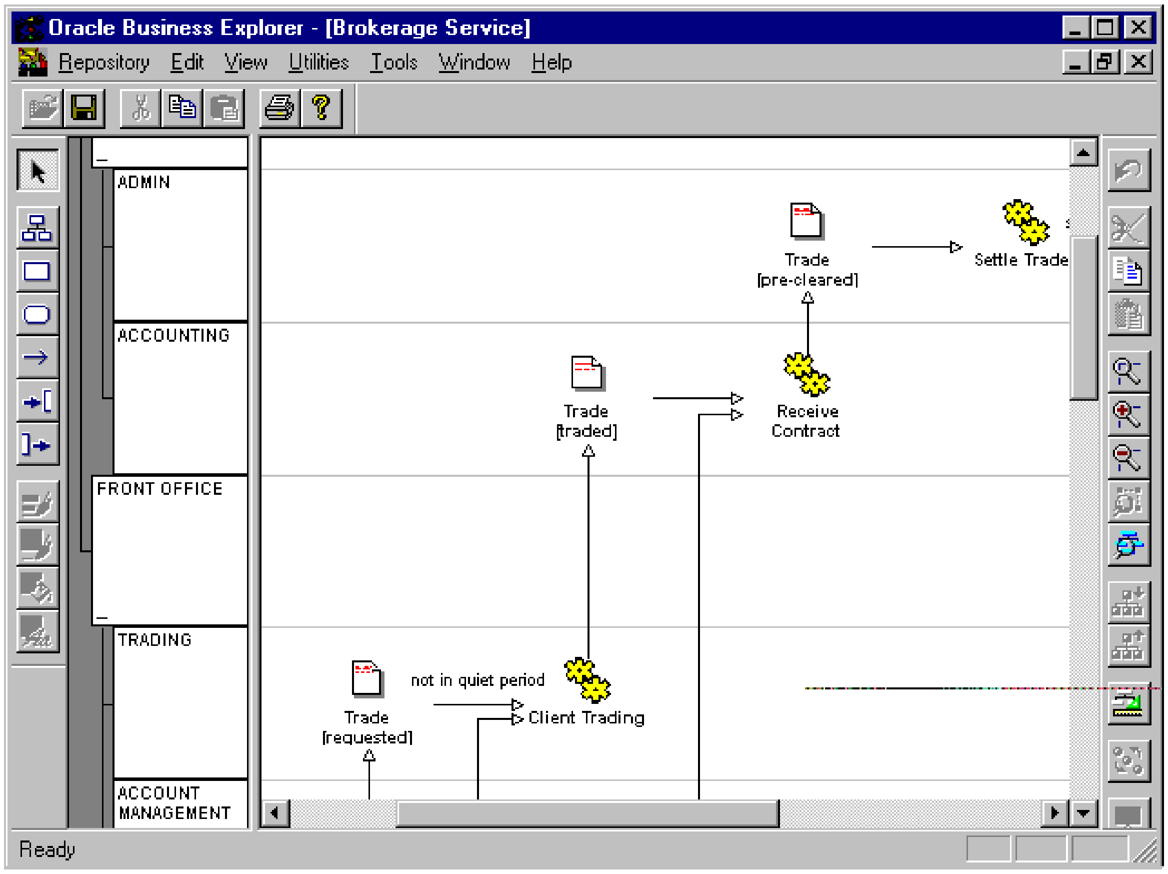
The challenge is to unite the power of object modelling with the accessibility of process modelling– and apply it to a business context.

Object Business Modelling

Although the origins of object modelling are primarily in embedded software, other real-time applications and GUIs, it is already recognised as having value in business modelling[6] – specifically for process modelling and BPR[7]. While there is little doubt that object oriented modelling can and should be applied to business modelling, it is early days yet for effective tools to support this approach.

The next generation of business modelling tools in Designer/2000 will replace the model that has underpinned structured analysis for many years; entities, relationships, functions, dataflows and so on. The new model will be based on the most well established object models and will use public standards where possible. The techniques and tools will seek to achieve two goals that set it apart from current technology:

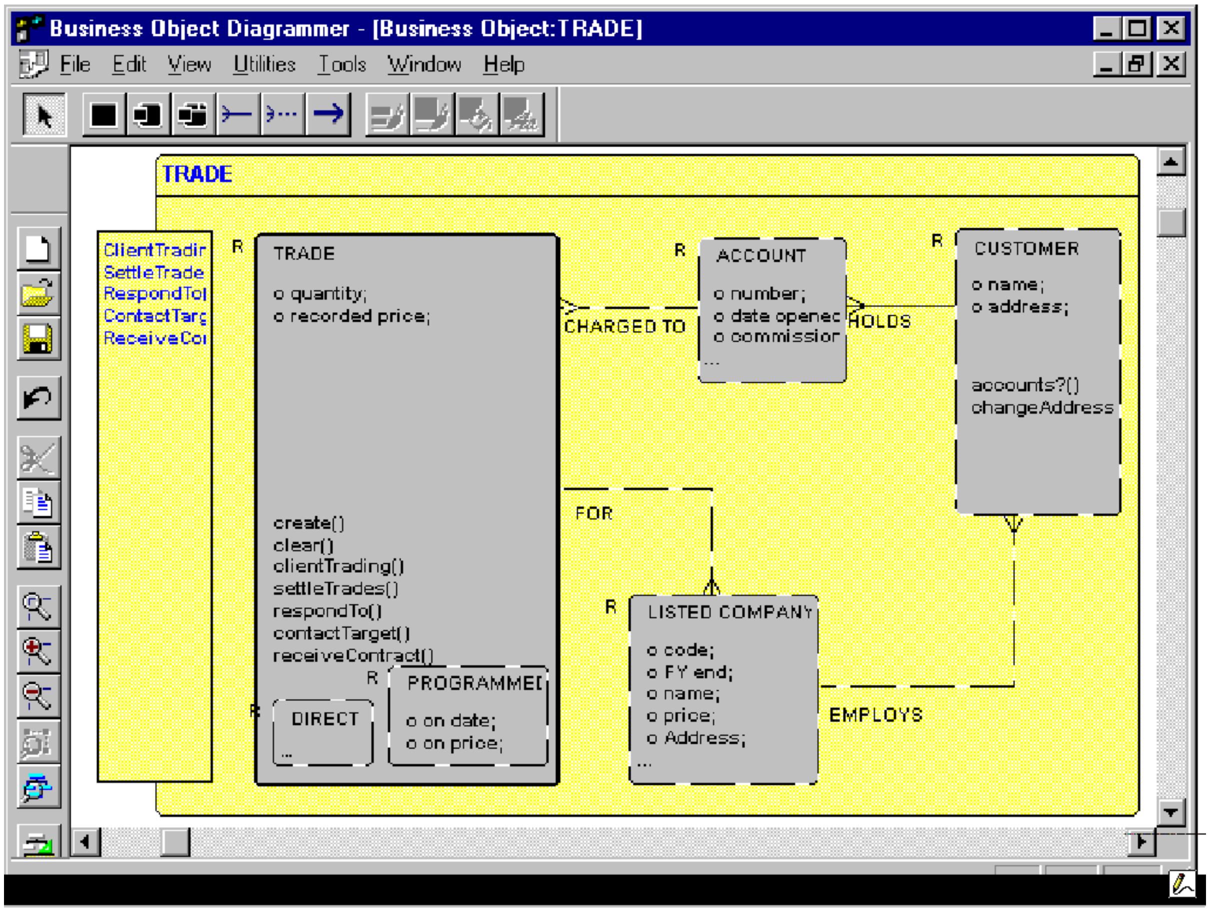
> **User Accessible**: the techniques will be understandable by business domain experts, and other end-users. There may be levels of complexity which the user interface can conceal in order to focus on the critical concepts. *Domain experts who are not object-literate will be able to read, understand and validate all the models relating to requirements*



6 Martin, J. and Odell, J.J. *Object-Oriented Methods, a Foundation*. Prentice Hall, 1995

7 Jacobson, I. et al. *The Object Advantage*. Addison-Wesley, 1994.

Business Analyst Accessible: the techniques will be recognisable by business systems analysts. There will be components of many of the techniques that will be new to a business analysts skilled in current practices, and some of the techniques may be entirely new. *Business analysts who are not initially object-literate will be able to understand, learn and exploit the modelling techniques as an evolution and growth of their existing skills.*



These goals are critical if object oriented business modelling is to achieve wide acceptance in enterprise information systems (IS) development.

However, acceptability in requirements definition is not enough. The models created must be directly transformable into systems. Current process modelling techniques are understandable and acceptable to users, but they do not capture sufficient information to lead directly to implementation (see User-Accessible Business Modelling above). Object oriented business modelling must significantly increase the semantic content of the models; this means it must deliver a full object model and an object model that is accessible and understandable to implementors.
But what of those implementors and the options they have for delivering production systems from business models?

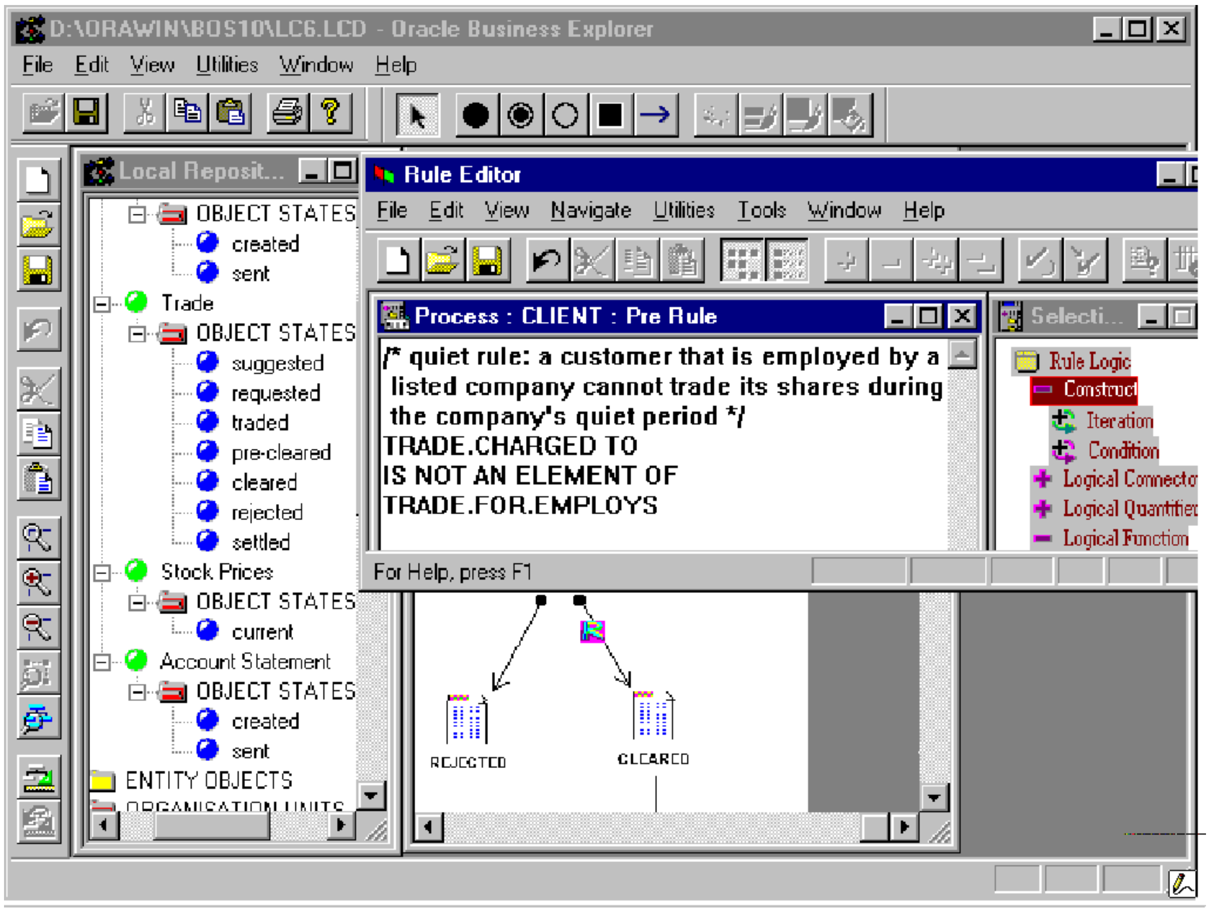
The implementation options available include:

code generation to a variety of target tools, for example C++, Java, Oracle Developer/2000 or Visual Basic. This would represent transformation of object business models into the implementation technologies of established OO languages or established GUI builder tools.

direct execution of the models by a new generation of integrated object technology. Because object models can progress from requirements to implementation by refinement and extension, rather than by transformation it is possible to consider direct execution of the requirements models.[8] This technology provides instant feedback to users and developers alike improving both productivity and user acceptability. Oracle corporation is currently prototyping tools to deliver this capability as part of our Object technology program.

hand crafting applications code using the models simply as specifications. Clearly this is the least desirable of the options since it cannot guarantee conformance of the delivered system to the requirements models but it is the option which is least dependent on technology.
Whatever the implementation choice, this need defines a third goal for object oriented business modelling:

> **Implementor Accessible:** the techniques will include full coverage of the object model, not just information and process, but also behaviour and interaction. *Implementors will be able to use the models to drive implementation through code generation, through hand crafting and if appropriate, through direct execution of the models.*



```
/* quiet rule: a customer that is employed by a
 listed company cannot trade its shares during
 the company's quiet period */
TRADE.CHARGED TO
IS NOT AN ELEMENT OF
TRADE.FOR.EMPLOYS
```

[8] Ramackers, G. Integrated Object Modelling, an executable specification framework for business analysis and system design. Thesis Publishers, Amsterdam, 1994.

The Designer/2000 Object Business Modelling Toolkit

The Designer/2000 object business modelling (OBM) toolkit will support a wide range of techniques to meet these goals. However the techniques will **not** be delivered as a disparate collection of independent tools, with an OMT-style Type Modeller over here, a Jacobson style Object Interaction Diagram over there and an Oracle-defined Process/Object Model somewhere else. OBM will deliver an integrated toolset operating on an integrated object model in a unified shared repository. A definition created in one tool is available in any of the other tools. A domain object defined in the type modeller is immediately available to be aggregated into a business object and used as the input to a process step which transforms it into another state which is immediately visible on the state transition model of that object.

The critical success factors for this approach are:

- an object model broad enough to embrace the whole business
- a model that talks the language of business users
- a set of tools that builds on proven business-oriented approaches
- a technology that integrates multiple tools efficiently on the model

The development program which will deliver OBM includes consultation in established OO forums[9] and with methodologists, participation with industry standards organisations such as the OMG and through field trialing by customers. This is a demanding program, but one which will build a vital bridge to enable the crossover of object technology in to mainstream IS development.

[9] Ramackers, G. and Clegg, D. Object Business Modelling, requirements and approach. In J. Sutherland, D. Patel, C. Casanave, G. Hollowell, and J. Miller. (eds.), Proceedings of the OOPSLA95 workshop on Business Object Design and Implementation, Springer Verlag, (in press).

Conclusion

Designer/2000 has proved itself a powerful tool in IS development in terms both of productivity and maintainability and the exploitation of object concepts already makes a significant contribution
Future releases build on this contribution which will lead to a transformation of Designer/2000:

- from a highly effective toolset rooted in relational technology
- to an even more effective toolset that fully exploits object technology
- and, because this transformation is evolutionary,
- while preserving the valuable investment in knowledge and skills.

This is not object technology because it is a fashionable buzz in the novelty-obsessed information technology industry, but object technology because it represents the next step forward in delivering what users really want: better systems, more well aligned to true user needs, delivered more quickly and maintained more easily.