Integrating Legacy Subsystem Components into an Object-Oriented Model

P G Neil, K P Bright and R A Sherlock
Dairying Research Corporation
Private Bag 3123
Hamilton, New Zealand

Abstract: Object-oriented (OO) programming languages allow more manageable modelling of complex systems by facilitating the independent development of sub-components. Three methods of incorporating existing procedural code components into an OO model are described. Mixed language programming allows the building of a program in which the source code is in more than one language. Procedural languages that are able to be integrated into an OO model using this method are dependant upon the procedural linker and OO language used. Mixed language programming alone is normally unsuitable for integrating legacy components into an OO model due to the potential loss of global data storage and the restricted combinations of programming languages that can be utilised. Recompiling legacy source-code into a dynamic link library (DLL) allows integration into an OO model through loading and accessing DLL's at runtime. Functionality must be contained within routines, procedures or functions, as these form entry points into DLL's. Recompiling a legacy model as a DLL is often an achievable option but may require 'wrapping' using mixed language techniques. Default interprocess communication (IPC) mechanisms exist in host windowing operating systems which enable incorporation of models only available as executable DOS or Windows programs, or models that have been developed in proprietary modelling environments. IPC mechanisms can also be introduced into model components available as source-code to allow more direct integration with an overall OO model structure.

1. INTRODUCTION

Computer models that attempt to represent complex real-world systems to a level providing a useful description and accurate predictions commonly require the development of large, complex computer programs. Procedural programming languages are not designed to easily facilitate the development of large complex programs [Meyer, 1997]. By contrast object oriented (OO) programming languages provide design concepts to facilitate development of large programs through allowing the isolated development and prototyping of program units.

An OO program is constructed by developing isolated program units called classes (for definition of a class see Booch, [1994]) which encapsulate both variable declarations and methods. Because classes encapsulate both methods and variables they can be considered functionally similar to small procedural programs (Figure 1). Classes act as a template for the creation of many instances [Booch, 1994] which is similar to the creation of many application instances [Microsoft, 1995b] from a single executable file. The dynamic creation of instances from different classes and the subsequent co-ordination by discrete messaging is the basis of an OO program's execution. This allows the flexible building and linking of instances at runtime, providing a simulation environment which easily accommodates the modelling of scenarios involving different combinations and numbers of real system entities. The potential to describe individual system entities promotes the ability of a computer model to predict and describe events more closely aligned with those occurring in the real-world system [Plant and Stone, 1991].

When using an OO programming language to model a complex real-world system, real sub-components of the system should form the foundation for individual classes [Fishwick, 1995]. Focused development is therefore promoted through the development and refinement of model components representative of real system entities. The segregation of a whole system model into classes also better facilitates multi-contributor involvement as people with the relevant knowledge and interest in a system component can develop the corresponding class.

Following the decomposition of the design of a large system model into classes, much of the functionality required by some of the classes may be present in existing models. In many cases these will have been written in a procedural language. Reconstructing this functionality in a class in the OO programming environment is time consuming, error prone and in some cases not readily achievable due to differences in the available statements and functions. In most cases, however, it is possible to directly integrate existing component models into an OO model with little or no re-coding. Integration techniques available for incorporating legacy models are described in this paper. In choosing an integration methodology it is important that the legacy model can be incorporated in the OO model to behave as a true object in terms of messaging and data encapsulation. Two actual implementations where integration procedures allowed legacy models to behave as true objects will be explained in detail.

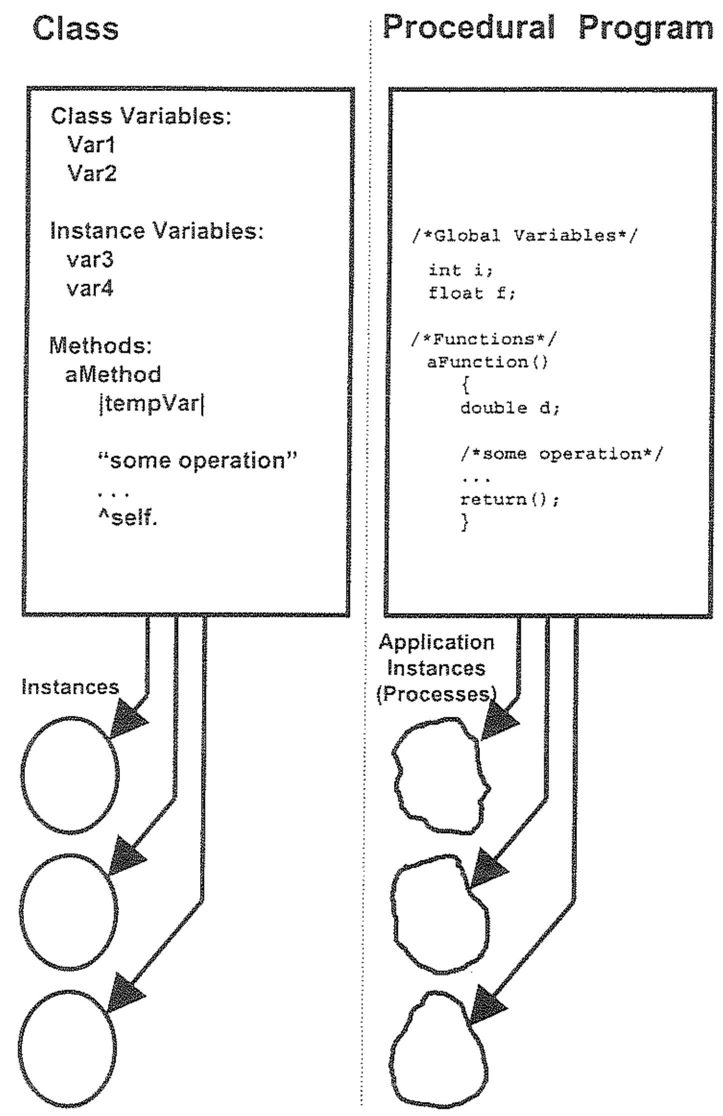

Figure 1: Classes are similar to executable procedural programs in the following ways: global variables (i, f) in a procedural program are the functional equivalent to instance variables (var3, var4) defined in a class. Variables declared inside functions (d) are functionally equivalent to temporary variables (tempVar) declared inside a method in a class. Executable programs can act as a template for many application instances, similarly a class acts as a template for the creation of many true class instances.

2. INTEGRATION STRATEGIES

2.1 Mixed Language Programming

Mixed language programming is the building of an executable program in which the source code has been written in more than one programming language. This includes the building of both procedural/procedural and OO/procedural mixed language programs. The potential for building OO/procedural programs provides a mechanism by which the source code of legacy component models can be directly incorporated into an OO program. For the purposes of the rest of this paper, functions, procedures and routines will generically be called routines. Also, because of a lack of accepted generic OO terminology, we will use Smalltalk 80 naming conventions [Goldberg and Robson, 1989] for illustrative purposes.

The inclusion of procedural source code into an OO program using this approach involves developing a class which calls routines from the legacy code unit to implement its methods. The functionality of the legacy code unit must therefore be directly accessible through callable routines. During the building of a mixed language program some global variable storage, such as COMMON block variable declarations outside functions in FORTRAN, are lost. If global variables are lost during the build process these must be re-introduced either as a data structure in the legacy code or through re-direction into a data structure in the OO language.

When creating a mixed language class any global variables declared in the legacy code exhibit class variable behaviour in instances of that class (Figure 2). Instances can therefore only access one common set of variables from the legacy component of the class rather than a unique set for every instance. Under this scenario the legacy model does not exhibit true instance behaviour in terms of data encapsulation because more than one instance can modify the global variables in the legacy model. This could lead to unpredictable and erroneous results if more than one instance of this class is used. Legacy global variable storage can be re-directed into instance variable data structures in an OO class, thus enabling true instance creation. Each instance then has its own encapsulated set of the legacy global variables.

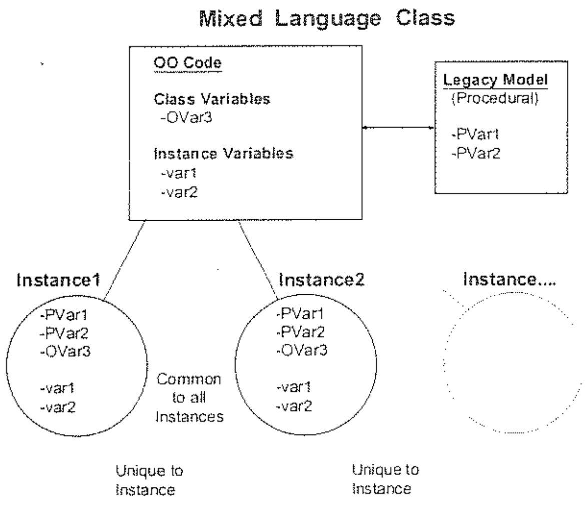

Figure 2. Variables existing in the legacy model (PVar1, PVar2) component of a mixed-language class exhibit class variable behaviour in the subsequent instances.

The building of a mixed language program is dependent upon both the procedural linker and the OO language used. For example the Microsoft FORTRAN linker [Microsoft, 1995a] allows the building of Basic/FORTRAN, MASM/FORTRAN, C/FORTRAN procedural mixed language programs. The only OO/procedural mixed language program possible with the Microsoft FORTRAN linker is C++/FORTRAN. The VisualWorks Smalltalk mixed language linker [ParcPlace-Digitalk, 1995] by contrast only allows the building of Smalltalk/C mixed language programs. The ability to use mixed language programming to incorporate legacy models into an OO framework is therefore restricted to a discrete subset of OO/procedural language options.

2.2 Dynamic Link Libraries

Dynamic linking provides a way for an executable program to call a routine that is not part of its code. The executable code for this routine is located in a dynamic-link library (DLL) containing one or more routines that are compiled, linked and stored separately from the programs using them [Petzold, 1992]. These DLL's can be loaded dynamically by a program, which is then able to directly access routines contained in the DLL. Source code from most procedural programming languages i.e. C, PASCAL, COBOL, FORTRAN and BASIC can be recompiled as a DLL with little or no source code changes. Legacy models available as source code can therefore be recompiled as a DLL and be integrated into a OO model by loading and accessing at runtime. Because access to a DLL is via routines, essential operations of a model must be implicitly contained in callable routines. Global variable declarations in the procedural source code are maintained during recompilation but they are not directly accessible. The recompiling of legacy models into a DLL, and the subsequent loading of a DLL by an OO program at runtime, provides a mechanism by which an OO model can utilise a legacy component model.

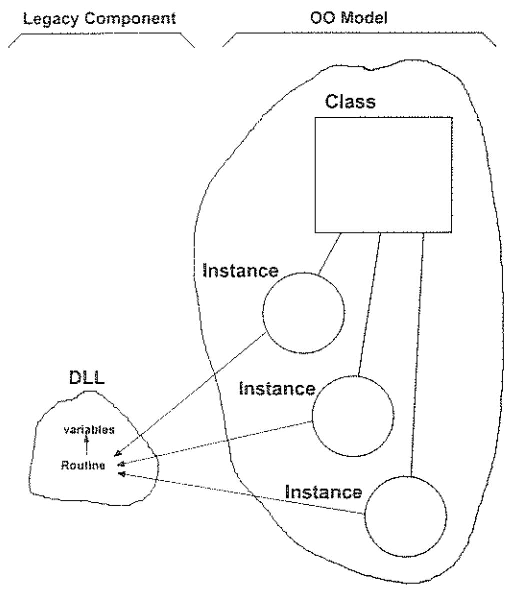

Figure 3: A single DLL exhibits non-encapsulated behaviour when more than one instance references the DLL. Each instance in the OO model has the potential to independently modify global variables in the DLL.

In the OO model a class is developed, instances of which interface to a DLL. Only one application instance of a DLL can be loaded into memory at any one time. If more than one instance interfaces to such a DLL unpredictable results may occur since variables in the DLL, changed by one instance, can affect subsequent calculations invoked by routine calls from other instances (Figure 3). Hence if more than one instance is to interface to a legacy model DLL which 'maintains state' in the form of global variables, a unique DLL must be created for each instance which requires it. Only one instance is then able to modify a global variable in a DLL, thus encapsulating the behaviour of the DLL within a single instance in the OO model (Figure 4). The creation of multiple DLL files can either involve the manual creation of a DLL repository, or be implemented through code in the OO program.

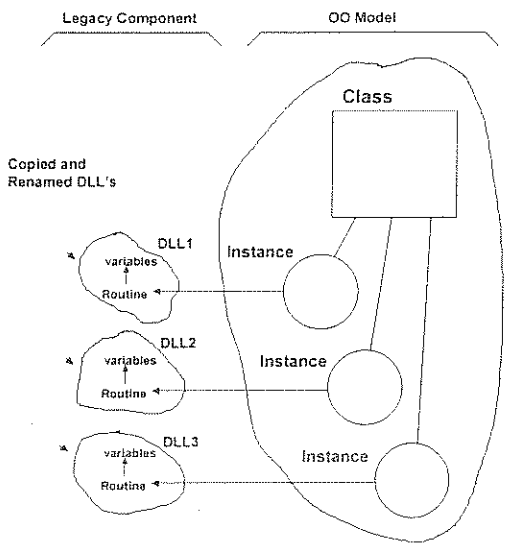

Figure 4: When each instance has a unique copy of the DLL, the DLL's exhibit encapsulated behaviour as variables in any DLL can only be modified via calls from a single instance.

2.3 Interprocess Communication

Interprocess communication (IPC) is the exchange of data between processes simultaneously loaded into computer memory. A process is an executable application loaded into computer memory. It consists of a unique address space, a data segment and reference to a shared code segment [Microsoft, 1995b]. IPC is supported by multi-tasking, window based operating systems and covers several mechanisms: intermediary data protocols such as the clipboard, dynamic data exchange (DDE) and file mapping, through to direct data transfer mechanisms such as Sockets, Pipes and object linking and embedding [Microsoft, 1997].

To use IPC as an integration methodology a class is developed in the OO model such that its instances implement an IPC conversation with a legacy component model. Following instance creation the legacy model is loaded into computer memory using an operating system call to initiate a process. A conversation is then established between the process and the instance in the OO model. In operating systems that support the creation of multiple application instances from a single executable file it is possible to load a separate copy of a legacy application for every instance that is to communicate with it. Application instances have unique variable storage which means that when a unique instance interfaces with a legacy model in this form it can be utilised as a true object (Figure 5).

The clipboard provides one mechanism for the exchange of data between all applications running under a window based operating system. It is primarily designed to facilitate user controlled exchanges of data between application interfaces. Programmatic window manipulation, messaging and virtual key stroke generation, however, allows the clipboard to be used dynamically by one computer program to control and obtain data which is available from the interface of another program. Legacy models that can be executed under window based operating systems can therefore be incorporated into an OO model. These include models that are only available as compiled and linked executable programs, models developed as part of a proprietary modelling environment, models which are not suitable for recompiling as a DLL, or models not suitable for mixed language programming.

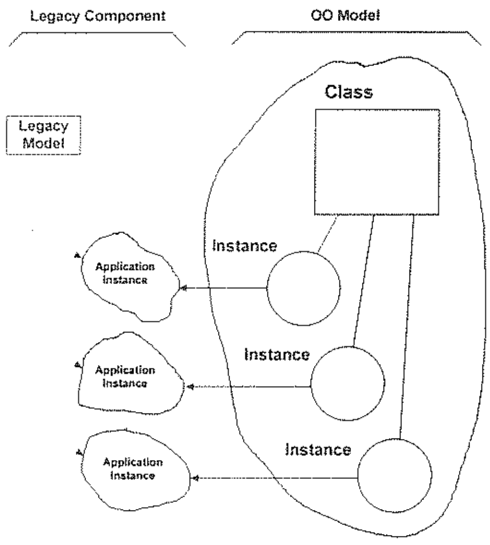

Figure 5: Each instance in the OO model has an exclusive link to a corresponding application instance of a legacy model.

Some proprietary modelling environments have support for more direct modes of IPC data transfer. ACSL for example [Mitchell and Gauthier Associates, 1995] can act as a DDE server which responds to input commands and requests for data from a separate client application. More direct IPC mechanisms are preferable to using the clipboard in combination with virtual key stroke synthesis and window manipulation because they are easier to implement and are not restricted to data and inputs available via the interface. Models that are available as source code can have a more direct IPC introduced directly into the source code. However, the IPC mechanism chosen depends upon the detail of how the model is to be implemented. To improve model execution times, computationally intensive legacy model components may be better executed on a different networked computer. IPC methods such as DDE, Pipes and Sockets allow for seamless distribution as the communication protocols are the same between applications running on a single computer or on networked computers. The implementation of a particular IPC mechanism is programming language dependent, a fact which may restrict the data exchange protocols that are able to be used. Mixed language programming can, however, expand the IPC mechanisms that are able to be incorporated into legacy source code.

3. ACTUAL IMPLEMENTATION OF INTEGRATION PROCEDURES

In our development of an OO dairy farm system computer model [Sherlock et al., 1997], two procedural legacy computer models were identified as having functionality required by objects in the OO model. A FORTRAN model [McCall, 1984] contained the required pasture growth functionality while an ACSL cow model, MOLLY [Baldwin, 1995], provided the required bovine digestion and metabolism functionality. The integration strategies used were chosen to allow individual cows and paddocks to be described in the OO dairy farm model.

3.1 Integrating a FORTRAN Source Code Model into a Smalltalk OO Farm Model

The version of Smalltalk [VisualWorks (ParcPlace-Digitalk), 1995] used to develop the OO model only supported the calling of C modules either as part of a C/Smalltalk mixed language program or through loading and accessing routines in a DLL. This environment had no capacity to directly interface to FORTRAN which meant direct incorporation of the legacy FORTRAN model was not possible. Because the building of a mixed language program was possible using C and FORTRAN source code [Microsoft, 1995a, 1995b] it was possible to 'wrapper' the FORTRAN code to behave as a C module. In a C/FORTRAN mixed language program, routines in the FORTRAN code can be declared as functions in the C code by prefixing the function declaration using a '_stdcall' keyword, which forces the routine to be called pushing associated parameters left to right. This is how parameters must be passed to FORTRAN routines. The additional declaration of extern "C" tells the compiler that the routine is defined outside the current text file:

C code:

```
extern "C" void _stdcall DRIVER(double CLIMAT[][7],
int *inputDAY, double *PRAD, double *METLAT, double
*PTEMP, double *METALT, double *ALT, double
*TEMPPROP);            /*FORTRAN subroutine
                       declaration in C */
```

FORTRAN code:

```
SUBROUTINE DRIVER (CLIMAT,inputDAY,PRAD,METLAT,PTEMP,
METALT, ALT, TEMPPROP) !The actual FORTRAN subroutine
...
```

The FORTRAN routines are then able to be called from a C program as if they were C functions. i.e. the wrapping procedure produces C functions which mirror the FORTRAN routines. The calling of these proxy C type functions from VisualWorks Smalltalk using an external interface subclass [ParcPlace-Digitalk, 1995] therefore provided access to the FORTRAN model. The FORTRAN/C mixed language program was compiled into a DLL [Petzold, 1992]. The proxy function calls existing in the C code were specified to be exported during the DLL build, which enabled them to be externally visible and therefore callable from Smalltalk.

The OO farm system model consisted of multiple paddock objects so to represent the unique growth parameters and states of individual paddocks it was necessary to create an instance of a pasture growth model for each paddock object in the simulation model. This was achieved through the creation of a file repository of uniquely named DLL's from which a unique DLL was loaded into memory for each of the paddock objects instantiated at runtime (as described in Section 2.2).

3.2 Integrating an ACSL Model into a Smalltalk OO Farm Model

ACSL is a proprietary modelling environment which compiles and links code which is subsequently executed in a proprietary runtime window. The runtime window provides DDE server capabilities, responding to input instructions and data requests from other applications loaded into memory [Mitchell and Gauthier Associates, 1995]. VisualWorks Smalltalk [ParcPlace-Digitalk, 1995] contains base classes that can utilise the Microsoft Windows dynamic data exchange management libraries (DDEML) that allow the creation of DDE server or DDE client objects. DDE therefore provided a mechanism for the inclusion of the ACSL cow model into the OO farm system model. In the OO model a DDE client class was developed whose instances would interface directly to a unique MOLLY project. Because a real dairy farm is composed of many different cows, it was desirable to create many separate cow objects in the OO model. ACSL supports the simultaneous running of different projects so this was possible by the creating and utilising of a repository of MOLLY projects, each accessed via an exclusive DDE client instance.

4. CONCLUSION

It is possible to integrate existing models in a variety of languages and environments into an OO model so that they behave as true objects in the OO model. However, this is not readily achieved using mixed language programming alone due to the restricted subset of OO/procedural mixed language programs that can be built, and the class variable behaviour exhibited by global variables in the legacy model. Mixed language programming can, however, play an important part in the successful adoption of DLL and IPC techniques.

Recompiling legacy model source code into a DLL provides an integration method that allows legacy models to behave as true objects in the OO model, provided that key functionality is contained within callable routines and that for each interfacing instance there is a unique DLL. Depending on the type of interfacing supported by a specific OO programming environment, it may be necessary to 'wrapper' the legacy code using mixed language programming techniques to allow the legacy model's functionality to be accessed.

Interprocess communication (IPC) provides versatile integration options and all existing models can in principle be incorporated into an OO model using these techniques. This includes models available only as an executable application, as the clipboard at least can be used as a default data exchange mechanism. The introduction of more direct IPC mechanisms into models available as source code generally allows easier and more flexible integration procedures. Mixed language programming techniques may, however, have to be used to introduce the desired IPC mechanism. An OO instance implementing an IPC conversation can interface to a unique application instance, which means the legacy model is able to exhibit true object data encapsulation.

5. ACKNOWLEDGEMENTS

The authors would like to thank Mike Maguire and Leigh Roberts for their valuable advice on detailed aspects of VisualWorks Smalltalk. This work was supported by the New Zealand Foundation for Research, Science and Technology.

6. REFERENCES

Advanced Continuous Simulation Language (ACSL) *User manual/reference. Version 11*. Mitchell and Gauthier Associates Concorde, 362pp., MA, 1995.

Baldwin, R. L., *Modeling Ruminant Digestion and Metabolism*, Chapman and Hall, 578pp., London, 1995.

Booch, G., *Object-Oriented Analysis and Design*. The Benjamin/Cummings, 589pp., Redwood City, CA., 1994.

Fishwick A., *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall, 448pp., Englewood Cliffs, NJ, 1995.

Goldberg, A. and Robson, D., *Smalltalk-80 – The Language*, Addison-Wesley, 585pp., Reading, MA, 1989.

Petzold, C., *Programming Windows 3.1.* , Microsoft Press, 983pp., Redmond, WA, 1992.

Plant, R. E. and Stone N. D., *Knowledge-Based Systems in Agriculture*, McGaw-Hill, 259-287. New York, NY, 1991.

McCall D. G., *A Systems Approach to Research Planning for North Island Hill Country*, PhD Thesis, Massey University, NZ, 1984

Meyer, B., *Object-Oriented Software Construction.*, 2nd Ed Prentice Hall, 1250pp., Englewoodl Cliffs, NJ, 1997.

Microsoft FORTRAN PowerStation. *Books Online. Version 4.0*, Microsoft Corporation, Redmond WA, 1995a

Microsoft Software Development Kit Documentation http://premium.microsoft.com/msdn/library/, Redmond, WA, 1997.

Microsoft Visual C++. *Books Online. Version 4.0*, Microsoft Corporation, Redmond, WA, 1995b

Sherlock, R. A., Bright, K. P and Neil, P. G., An Object-oriented simulation model of a complete pastoral dairy farm, in McDonald, A.D., Smith, A.D.M and McAleer, M (Eds) MODSIM97; Proceedings of the International Conference on Modelling and Simulation, Modelling and Simulation Society of Australia, Hobart, Dec 1997.

VisualWorks DLL and C-Connect *User's Guide*, Rev 1.2, ParcPlace-Digitalk, 266pp., Sunnydale, CA, 1995.