Object-Oriented Software for Dynamic Modelling

Peter Scott

Otago Polytechnic
Dunedin, New Zealand

peters@tekotago.ac.nz

Proceedings of the 15th Annual NACCQ, Hamilton New Zealand July, 2002 www.naccq.ac.nz

ABSTRACT

This paper discusses an alternative approach to dynamic modelling of systems involving a significant quantity of individual components or entities. Traditionally such problems have been translated into a system of equations representing the desirable and undesirable indices, and the parameters for which an optimal solution is desired. The solution then relies upon traditional numerical analysis. Such an approach offers high computational efficiency and usually ensures a global optimisation, but can be difficult to implement in the first instance and to maintain beyond the predetermined scope of problems catered for. In essence there are limitations on the reusability of code and flexibility of software constructs that are not based upon an object-oriented approach, and for certain problems where computational efficiency no longer holds a high priority an alternative approach utilising object-oriented design principles appears to offer numerous benefits.

1. INTRODUCTION

This paper suggests the development of a specification for the creation, behaviour and use of objects to transfer information, where presently data input and output streams place limits on the usefulness of the information. Data transfer objects fulfilling this specification could be used to implement an alternative approach to modelling of systems involving a significant quantity of individual components or entities, utilising object-orientation to facilitate flexibility and scalability.

The use of computer modelling to simulate real-world systems existed prior to the age of digital computing, when analog electrical circuits were tuned to produce behaviour analogous to that of mechanical mass-spring-damper systems. Those circuits were highly specialised, but nonetheless contained subsets of components that represented individual elements of the mechanical system, and could be reconfigured to reflect changes made to the mechanical system. The structure and operation of analog computers has more in common with object-oriented software development than the intervening era of strictly procedural algorithms - the output of the circuit is generated by the transfer of signals between sub-circuits that, in themselves, are implementations of real-world objects. Software engineering practices adhering to similar principles have been adopted wholeheartedly and continue to develop, however computer modelling and optimisation of systems involving significant numbers of entities remains largely in the domain of procedural programming.

One approach, the finite volumes method used to model a continuum, involves division of the problem space into discrete control volumes, and subsequent solution through the balancing of equations that describe transfer of heat, mass, momentum and any other conservative quanta across the surface of each control volume. The solution to the resulting system of equations, usually expressed as an array, is derived using linear algebra and numerical analysis. An object-oriented approach could arguably achieve the same solution, with each control volume defined by an object, and each object messaging its immediately adjacent objects to negotiate the transfer of conservative quanta and typically satisfy that the net transfer into the control volume is zero. Any object-oriented implementation of the finite volumes method would be dependent on extensive messaging between the control volumes and would be likely take orders of magnitude longer than the procedural equivalent to arrive at a comparable solution, but would offer extensibility and responsiveness to minor changes without recompilation of the entire problem.

This capacity to modify and extend the problem domain without rebuilding the existing solution is referred to in this paper as dynamic modelling. An advantage of this approach is the return of a modified solution within a timeframe in rough proportion to the magnitude of change - incremental changes should take less time to solve as the increments become smaller. A second important advantage is that the execution of the solution code is well suited to distributed processing, vastly improving the ability to utilise ordinary workstations during idle periods. There are, however, improvements that could be made to the information transfer mechanisms between objects and between applications that could make this distributed method of solution applicable to a much wider problem set involving data of all types held within applications and databases as well as numerically specified as an initial condition. The intent of this paper is to encourage the development of a specification for data transfer using objects rather than data formatted for file or an existing protocol.

2. EVIDENCE OF MIXED SUCCESS IN SOFTWARE DEVELOPMENT

In some respects the adoption of object-oriented programming has instigated tremendous change in the ability for an untrained computer user to quickly access information and functionality provided within the applications they have purchased. Primary school children can publish to the web, most home users can create and edit multi media presentations, produce correctly formatted accounting information and exchange information via e-mail without training. The reasons for this include:

1. Provision of a consistent interface for all applications
2. Drag-and-drop, point-and-click wizardry that is predictable and intuitive
3. Careful concealment of internal data handling procedures.

This approach has been very successful in promoting computational information management, but many unfortunate realities have been brought to bear upon more experienced computer users seeking increases in flexibility efficient use of time, through the adoption of new information management software:

1. Highly developed interfaces for access to information are not necessarily very flexible or efficient from a user perspective. Information retrieval (especially when using client-server systems) is often no faster electronically than if the user looked through a printed record on hand at the time the information is needed.
2. Refinement of the scope of a search for information is often a time-consuming, repetitive process, for which 'book marking' features (providing the ability to repeat a similar, if not identical search without starting from scratch) are not generally consistent.
3. Point-and-click and drag-and-drop mechanisms, although intuitive, can lead to OOS after extensive use, especially if the input controls require careful placement of the mouse. (In some cases the only keyboard-driven alternatives involve selection though repetitive alt, tab and arrow keystrokes - far from ideal, but beyond the scope of this paper.)
4. The information, once retrieved, can usually be viewed, printed and 'exported', but not necessarily reformatted or manipulated whichever way the user desires from within the application that created it. In addition, the exported information is often inappropriately formatted for immediate use by other applications, and mechanisms to automate this process are not usually included with the applications.

Ironically, the information returned to the user would benefit from the object-oriented philosophy that has accompanied software development in recent times and brought such significant enhancements in first-time usability. The potential to apply this philosophy is illustrated in the following two examples: The user will, in many cases, require a static data set (a snapshot of the data values without any ongoing link to them), but in other situations what is really wanted is a handle to the relevant information which is able to be relocated and otherwise modified from within the output information object. Secondly, note that requests from experienced users could be (but usually are not) interpreted into a user profile, automating the book-marking process described above. Both of these behavioural aspects would be natural features of software that is engineered to treat the user input and information output as objects of user input and output data classes. Such classes would have methods and attributes, which, although necessarily undefined at first, are likely to conform to one of a number of patterns over time. The objects of the classes should provide structured access to the encapsulated data in a well-specified (but not necessarily uniform) way, to allow useful manipulation of both the information request and the data returned from an external application acting as a host for the object. Finally, the output data should be encapsulated in such a way that any changes in the source information can propagate through to the data object, and any changes made to the data object may be reinjected into the source information when permitted.

An interesting indicator of the limited flexibility and user acceptance of new information-handling software is the persistent use of proven paradigms for the transfer, storage and analysis of information. Spreadsheets are in common use as databases despite minimal features for security and integrity. E-mail is the preferred vehicle for the exchange (both distribution and requisition) of information in small quantities. Hard copy of information to look at it for reference or to suggest modifications remains standard practice even if a hard copy of the information is not the end goal, and conversely information is generally requested via printed form (generated electronically) even when there is access to computers which could allow electronic capture of the information. The handwritten information is typically entered in a batch process that introduces both a delay and scope for errors in transcription.

These realities of the implementation of information-handling software detract so greatly from the potential gains in accuracy, efficiency and accessibility that concerns exist over whether there is any benefit in the purchase and deployment of new software when its potential users are already familiar with existing, albeit flawed, information systems. The likely outcome is that they will substantially continue on as before, exporting information to spreadsheets for manipulation, printing copies for reference and e-mailing or, even worse, printing requests for more information thus creating the need for manual data transfer or entry. User-friendly interfaces are inadequate to overcome deficiencies in the flexibility of input and output mechanisms, and may not make location and modification of information any faster, let alone facilitate location and modification of information by a third party.

3. IDENTIFYING A POTENTIAL INNOVATION

Some of the inadequacies mentioned above would be resolved through efforts to provide increased input and output stream flexibility. The addition of configurable tools for importing queries and exporting data, support for existing data exchange mechanisms such as SQL, and even expert user interfaces available as an option to simplified novice interfaces such as available within WINZIP, would all be of benefit to users that required more efficiency and flexibility from their applications. These tools are not uncommon at present, however streamlining the flow of information between software packages can still be stumped by compatibility issues, communication delays and bandwidth restrictions. An alternative approach utilising object-oriented design principles addresses the compatibility issues and propagation delays through the following features:

1. The software providing the source data (or 'parent data') is referred to as a 'data parent', and is capable of creating 'data children', which essentially contain and manage copies of subsets of the parent data (referred to as the 'child data'). A data child may be hosted within the data parent, or relocated to and hosted by a third party. It not only encapsulates the output data (and/or requests for data), but also provides mechanisms for messaging and data transfer, to ensure data integrity between the parent data and the child data.
   The data child should be able to offer its host metadata regarding its contents and data about the permissions and integrity of the child data with respect to the parent data. A data child may also convey requests for information or services from the data parent to the host, equivalent in purpose to (for example) an electronic form, or requests for appointments, authentication or metadata about the host. A data child is essentially an object that conforms to a class specification detailing the methods that fulfil the above requirements, however the class specification should not place any restriction on the internal data or its structure.

2. The host should be capable of providing message transport between any data children it hosts and the corresponding data parents. A host may be given permission by the data parent to request (through the data child) a different subset of the parent data. The data child may also have permission from the data parent to send modifications made to the child data to be applied to the parent data (the mechanism for data input).
1. Data output intended for one-off usage (essentially snapshots) and similarly, casual user interfaces for instantaneous one-off requests are still supported, but intended for the provision of short-life information only. Long-life data requests and outputs are distinguished as data transfer functions which may be intended for use in third-party software, may exist long enough to become inconsistent with the parent data, or may be re-actioned at a later time. Long-life data transfers (both input and output) are actioned through the use of a data child.

4. BALANCING INGENUITY WITH ACCEPTABLE PRACTICE

Innovations such as the SMTP protocol, ZIP data compression and the SQL standards have arguably become accepted because of a necessity to ensure information can be accessed and exchanged between applications and across platforms. In the time since their appearance, the growth in communication capacity and general proliferation of software-based information tools throughout industry has led to new challenges in software design, made more difficult by a widening of target platforms to include PDAs and cellular telephones. Distribution of information between applications and across platforms can be implemented by the schema described here, but will not achieve the potential gains in compatibility and workplace efficiency unless supported by a well developed and widely accepted standard for its structure.

5. CONCLUSION

There is unrealised potential for information management software to provide more than snapshots of data for further analysis within other applications, and to accept more than preformatted input files and manual data entry. Creation of data objects that conform appropriately to a functional class specification that is present across a range of applications and yet internally can maintain data integrity when referenced by a host application and return information to the parent application could provide vastly improved compatibility and reduce manual data transferral. Further possibilities lie within the area of distributed, server-less data storage, and distributed optimisation in which each object may negotiate with others by hosting a data child from each one, improving the speed with which the desired individual equalities can be met. The author hopes that this paper might encourage cooperation towards the goal of making information as distributable and flexible as possible.

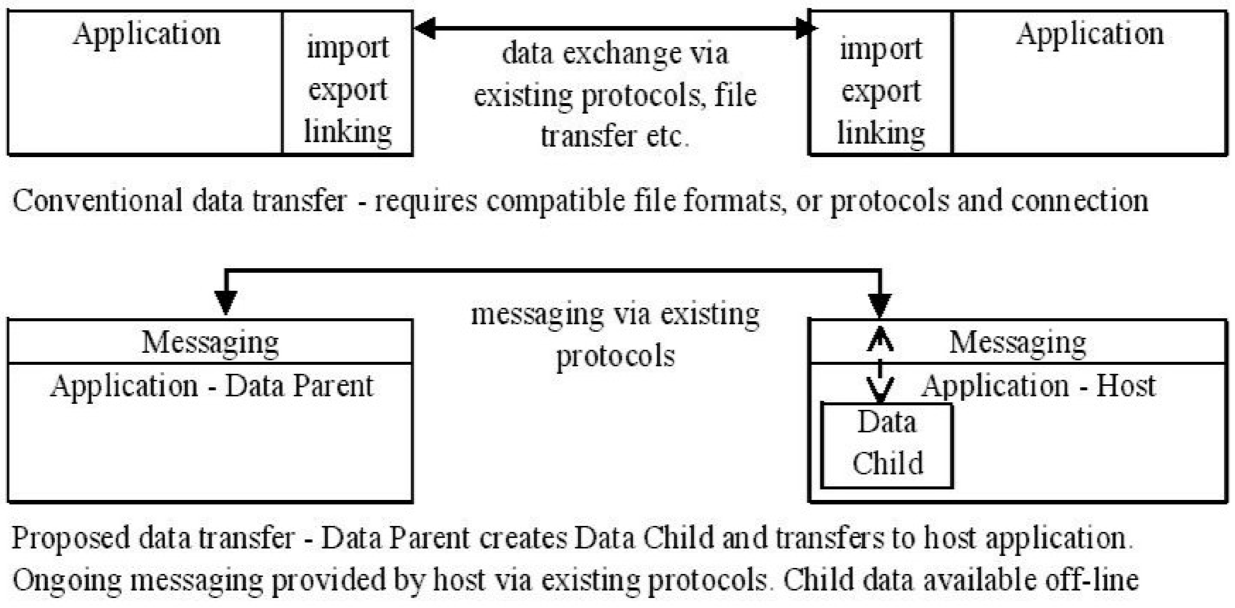
Application
import
export
linking
data exchange via existing protocols, file transfer etc.
import
export
linking
Application
Conventional data transfer - requires compatible file formats, or protocols and connection
messaging via existing protocols
Messaging
Application - Data Parent
Messaging
Application - Host
Data Child
Proposed data transfer - Data Parent creates Data Child and transfers to host application.
Ongoing messaging provided by host via existing protocols. Child data available off-line

Figure 1: Conventional vs proposed data transfer mechanisms