PLATFORM INDEPENDENT SIMULATIONS: THERMAL SIMULATION AS AN OBJECT

Ali M. Malkawi, Ph.D.
John Wambaugh
The University of Michigan
College of Architecture and Urban Planning
2000 Bonisteel Blvd.
Ann Arbor MI, 48109-0155.

ABSTRACT

The paper discusses the development of a computational thermal simulation engine as an object. The simulation engine utilizes Transfer Function (TF) Algorithms where all TF procedures have been implemented. The simulation is integrated with the Microsoft Windows environment and has been made to take advantage of many of this operating system's features. In particular, the separate components of the simulation take advantage of the Windows feature to communicate - Object Linking and Embedding (OLE). Individual modules and databases exist as distinct "objects" in the memory, providing their data and functionality to each other via an object-to-object interface seen only by other programs. The OLE implementation has meant that the object developed can readily communicate with and take advantage of other applications.

INTRODUCTION

Communication standards for application integration have been lead primarily by two developments, Common Object Request Broker Architecture (CORBA) and the Component Object Model (COM) which is the basis for the Object Linking and Embedding (OLE) technologies. Each has their merits and domain of applicability. CORBA has adapted a network centric approach and extended it toward the desktop. COM and OLE have developed leading desktop technology and are being extended to the network.

Over the past few years our development of building thermal simulation has been primarily concentrated on desktop computers. As a result, the Object Linking and Embedding technology was selected for our investigation. Essentially, OLE applies concepts from Object-Oriented Programming (OOP) to applications running through the Windows environment (Johnson 1997). OLE allows applications developed separately to communicate and share resources (Lalonde 1996). As part of the OOP paradigm, OLE applications are seamlessly included in the programming phase of software development.

Central to Object-Oriented Programming is the concept of an object. In computer programming terms, an object can be a collection of variables capable of holding data. By bundling numerous variables together in a logical fashion to create an object, those variables can be more easily manipulated. For the developers of such databases, it is much easier to copy, move and delete objects, than it is to keep track of all the variables that make up the object.

In addition to containing variables, OOP allows objects to contain methods. Methods are functions that operate on variables in the object. Some methods apply algorithms to variables in the object. Since the variables of an object may need to be formatted in a particular fashion, other methods are often used to provide input and output functions that make sure that information to be stored in an object is appropriate and formatted correctly. Often, methods are the only way an object allows variables to be manipulated – the variables themselves are hidden away to functions external to the object. The collection of methods for an object is called an interface. The interface is used in order to get at the features of the object.

An important feature of OOP is that it allows compartmentalization – the developer of an object writes all the code necessary for the object's methods to work, and then that object can be used by anyone else developing software without the later programmers having to know how the object functions (Horn 1996). In this way an object is self-contained. Several different developers could create different objects, and other developers could write software using those objects with only the knowledge of the objects' interfaces.

Object Linking and Embedding treats every application as an object. To create an OLE-aware application, an interface of methods that act on the

application's own internal variables needs to be provided. In this way, OLE allows software applications to communicate by accessing each other's methods. Furthermore, the functionality of previously developed applications can be incorporated and reused by treating those applications as objects in a newly developed code.

This paper discusses a thermal simulation environment that was developed in such a way that the entire simulation engine exists as one object. As a result, the developed simulation exists "behind the scenes" with the object-to-object interface communicating with the system, but no user interface is visible. Data can be passed to the simulation by giving it the name of a database object arranged in a particular manner. The simulation will return the results as another database object. This allows software developers to quickly create their own user interfaces and simulations by taking advantage of the features of the simulation object.

THE THERMAL SIMULATION AS AN OBJECT

Over the past few years our research team has developed a detailed thermal analysis module utilizing the Transfer Function Method (TFM), (Malkawi 1995). All the TFM procedures were fully implemented for the simulation to take place (ASHRAE 1992). The simulation provides the designer with a thermal evaluation of the proposed design. The simulation engine is used to calculate the cooling and heating loads the building generates and to provide a passive and active interior temperature analysis based on heat extraction and floating temperature procedures (ASHRAE 1993). The TFM in the system uses: 1. sol-air temperature to represent the dynamic nature of outdoor conditions and 2. assumes constant indoor air temperatures. Despite the second factor, the TFM can be set to recognize the variation of indoor temperature after being stabilized utilizing heat extraction rate and space air functions. This allows the simulation to be conducted for determining different sets of schedules and floating temperatures.

Although the original simulation was developed with a custom interface that allowed a user to enter a building description and conduct thermal simulations, the new development allows the creation of this simulation to exist as an OLE object, with no user interface. The objective of this OLE development of the simulation is to allow other simulations to make use of the features this simulation can provide and make it platform independent. Essentially a simulation developer can create his or her own user interface with whatever features he or she desires and then use the simulation object developed "behind the scenes" to actually do the thermal simulation calculations.

The new system has been written using Object Linking and Embedding (OLE) through the creation of Dynamic Link Library (DLL). OLE is an application integration technology that can be used to share information among applications within the windows environment. DLL is a library of routines that is loaded and linked into an application. In short, DLL allows a set of library routines to be used in run time to create an "object" in the computer's memory. This object can be accessed by any software running on the computer through OLE function calls, thus it allows for communication between widely varied software applications.

The DLL is a single file that provides the code for the simulation engine to any OLE aware programming language. Essentially, the DLL contains a "class". A class is a definition for an object – it describes the properties belonging to the object and the functions internal to the object that make use of its properties. An object is simply a copy of a class placed into active memory. In this way, a class can be used to create identical objects each time it is used.

It is easy for the features of the DLL to be used by other programs. First, the DLL is registered with the external software. This DLL contains the class. When the external software executes, it can use the class definition to create an object. The external software then interacts with the object, setting its properties and eventually calling the function to return the thermal simulation of a building, figure (1).

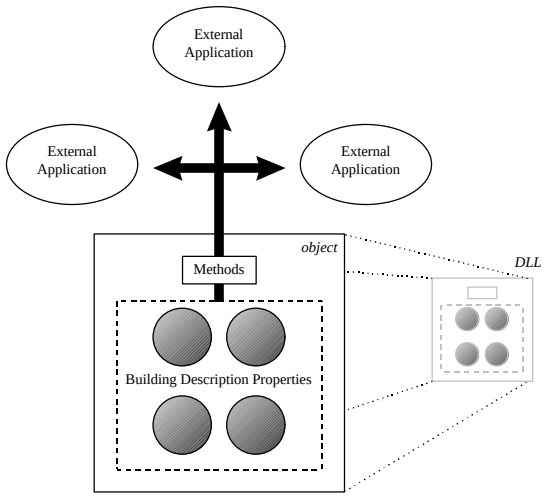

Figure 1 ***Object Interaction with External Applications***

The simulation object developed has two public "methods" or functions that can be called to give to or to receive from the simulation information and many properties. The first method is “initialize”. Initialize takes one argument, the path of the database input object containing a building description. This input object is a collection of simulation input values that represent properties of the class. Properties within this object can be set separately. For instance, the “wall area” property can be set to a specific value from an external application. When Initialize is called, it analyzes the data object given, calls in other DLL if needed, and prepares the simulation object for the second method. The second method is “calculate” which actually performs the thermal simulation. It produces a second data object containing the thermal simulation output for the building specified by the first data object’s description, figure (2).

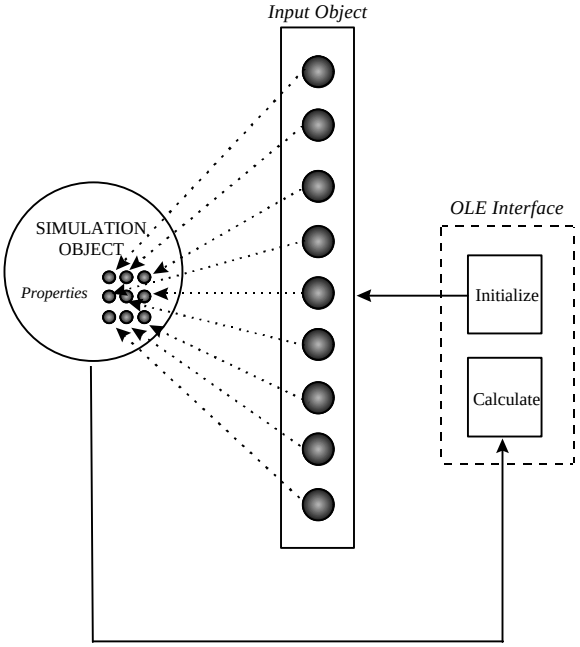

Figure 2 ***Object Interface Structure***

It is possible to implement this simulation object with only one method that takes in a building description and immediately calculates the thermal performance as a function of the building’s thermal properties. However, the input and simulation/output phases should be separated to allow easier determination of the source of possible errors. In other words, if the “initialize” method fails it will be directed toward the data object, whereas failure of the “calculate” method would be the fault of the simulation object.

Databases for the simulation object developed were designed to allow future expansion for other simulations to take place. As a result, the simulation object was designed to interact with separate databases developed as DLL. For example, the Wall and Roof Transfer Functions databases can be called by the input object when the “initialize” function is performed. This facilitates the development of external databases that contain their own search mechanism to aid in performing the simulation.

AN INTER-SIMULATION DATABASE

While creating a simulation object allows application developers to easily include the functionality of this simulation with a new interface, several simulations can be developed as objects and share data through different interfaces. This entails developing a building description database that would be “rich” enough for more than just a thermal simulation. With this database, a newly developed interface would only have to generate one building description database object and then access different simulations using OLE, all of which could perform their calculations based upon the data in this database object.

To investigate the applicability of this approach with the newly created object simulation, the original database of the simulation was restructured to test accommodating the building description information and providing links to other object databases developed. The specific form of the database object reflects its inter-simulation format – instead of being put together in a method that is only convenient for a particular simulation, the data is stored simply in a manner logical to one describing a building. In addition to information describing the building’s name and location, the object contains lists of the different components of the building. For instance, each wall is given its own entry that contains an identifying number, the wall’s ASHRAE wall type, and number of Cartesian points describing the wall’s position and geometry in space. The window entries can then refer to that wall’s identifying number so that, if necessary, a simulation can make the connection between the window and the wall the window is in. In that same vein, multiple walls are later grouped into zones describing different regions of the building as enclosed by several walls. The database allows the building component, a wall or a door, etc., to be described independently as objects with their own characteristics, while at the same time the component’s relation to other components of the building is also specified.

The database receives information from other

databases that were developed using the same methodology. Several building component databases were designed as separate objects and compiled as Dynamic Link Libraries (DLLs) with function calls that control its behavior, figure (3).

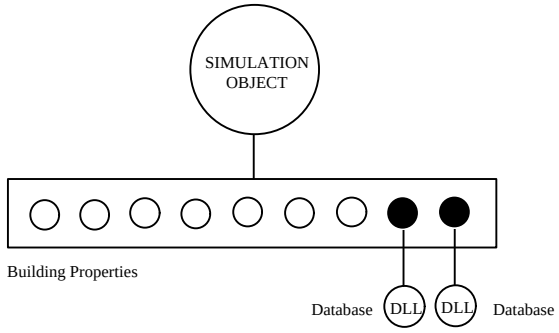

Figure 3 ***Interfacing with the Databases***

The functions are designed to be called from an interface with their output directed toward the database object created. This allows the flexibility of creating separate databases for separate building components and linking them to the building database using OLE. For example, since only a component's ASHRAE type number is stored, the Transfer Function Wall database created for the simulation developed will be used to look-up non-geometric information (Transfer Functions) about the particular wall. The database is intended to allow users to quickly and easily retrieve the transfer functions for almost any combination of wall or roof materials, thus simplifying the use of the Transfer Function Method of simulation early in the design process (Malkawi 1997).

The thermal simulation object developed centers on the concept of zones, and reads in only the zones composing the building description. Not interested in the components independent of each other, the simulation object uses the connections indicated by the new data format to piece together entire zones. Building components considered irrelevant to the thermal simulation, such as interior walls, are ignored and they are not included in any zones.

In short, a concise building description format was created as a test for conducting multi-simulations. This database allows information to be accessed by different simulation objects through third-party user interfaces.

THE OBJECT'S DATABASE IMPLEMENTATION

Testing the applicability of existing database applications that utilize OLE objects to aid in developing the system was an additional objective of this study. These applications must provide sturdy, easy and nearly universal abilities to store and retrieve information so that several applications can access it. By their very nature as Windows Office components, data stored in them can readily be put to use by developers of different applications with little contact. Applications developed in isolation, by using standard formats, can provide information to a multitude of other applications.

In particular, two simple products – Access and Excel were used. Access is a database program that not only readily stores information in an easily accessible format, but also provides numerous ways of searching for that information. At run-time, when the simulation is actually running, the Excel application is utilized to conveniently store the data developed in an OLE format – the Excel objects. The concept of an array, essentially a subscripted variable, is common to all programming languages a simulation might be developed in. The precise implementation of this concept, however, varies from language to language, and even from compiler to compiler. Excel objects were utilized as, essentially, two-dimensional arrays or matrices. Instead of the developed data being internal to the developed application, it is available for any OLE-aware application to access. Additionally, although Excel objects expected to be created by a simulation interface, there is the additional benefit that a spreadsheet-savvy user could create a worksheet object for the developed simulation, essentially a building description, from scratch.

CONCLUSIONS

This paper illustrates the conversion of an existing thermal simulation engine into a DLL object that makes use of OLE applications. The engine was designed to allow external databases to interact with it using OLE. This implementation of the simulation engine has meant that software developed by other parties, presumably other simulation software with its own interface, can easily access the functionality of the simulation modules developed. This allows a simulation developer to pick and choose the features of the developed simulation s/he wants to use. In addition, this implementation illustrates the potential of using this approach in designing multi-simulations each independent from the other with calls from external interfaces. A database that allows sufficient building description will be necessary to enable these simulations to perform. This paper discussed a limited version of such a database. A larger implementation of such a

database is currently being developed that will allow several simulations to run.

In building the OLE simulation, it was critical to decide which functions were necessary to maintain and which were not. While one could have provided numerous input/output functions allowing individual building properties to be set, the implementation was to create an object that reads another input object containing the entire building description based on the assumption that someone using a simulation object would already have a design they wanted to test. As with input, numerous output methods could have been created to allow users to retrieve data from the simulation object. Since the calculations can be done once, however, it was decided that all the data would be generated at once.

Interfaces designed with Java, or similar languages, could make use of the simulation without concentrating on the coding of the simulation itself. Placed in a client-server environment, the simulation would be essentially platform independent and could be utilized in a web based environment.

REFERECNCES

ASHRAE, “1993 ASHRAE Handbook Fundamentals”, Atlanta: American Society of Heating, Refrigerating and Air Conditioning Engineers, Inc., 1993.

Horn, C., A. O’Toole, “Distributed Object Oriented Approaches”, Proceedings of the IFIP/IEEE International Conference on Distributed Platforms: Client/Server and Beyond: DCE, CORBA, ODP and Advanced Distributed Applications, 1996. 7:17.

Johnson, R.E., “Components, Frameworks, Patterns”, Software-Engineering-Notes. vol.22, no.3, May 1997. 10:17.

Lalonde, W., J. Pugh, “Using OLE clients”, Journal of Object Oriented Programming. vol.9, no.1, March 1996. 76:82.

Malkawi, A., “A New System For Accessing Transfer Function Coefficients For an Architectural Computer-Aided Thermal Optimization Tool”, Proceedings of the 5th International Building Simulation Conference, September, 1997, Prague, Czech Republic.157:163.

Malkawi, A., “Simulation and Reasoning: Intelligent Building Thermal Problem Detection”, Proceedings of the 4th International Building Simulation Conference, August, 1995, Madison, Wisconsin.176:182.

McQuiston, F.C. and J.D. Spitler, “Cooling and Heating Load Calculation Manual”," 2nd ed. Atlanta: American Society of Heating, Refrigerating and Air Conditioning Engineers, Inc., 1992.