ThermalDesigner: an application of an object-oriented code conformance architecture

Robert Amor, John Hosking, Rick Mugridge, John Hamer, Mike Williams[1]

Abstract

In an earlier paper an architecture was described for supporting code conformance applications based on Kea, an object-oriented functional language. Here we describe a prototype application, developed using Kea, for checking conformance with a thermal insulation standard. Called ThermalDesigner, this application incorporates an object-oriented building model, a graphical plan entry system for editing plans, and form-based interaction for obtaining non-plan information and supplying results to the user.

Introduction

In Hosking et al (1991), an architecture was proposed for developing code of practice conformance applications. This architecture includes the following components:

- An object-oriented building model
- A functional representation of code provisions which are embedded within the object-oriented building model
- A CAD-like drafting front end to allow users to enter plan details
- A forms interface for entry and display of non-geometric information
- A consistency manager to handle changes to user-supplied data

The first two elements of the architecture are embodied in the Kea object-oriented-functional programming language. Kea has been developed in parallel with work on code conformance applications, and has been used to implement several such systems (Hosking et al, 1987; Mugridge and Hosking, 1988; Hosking et al, 1989). The plan entry and forms interface to Kea were developed to solve deficiencies in data entry for these applications. Consistency management has been an important component of our model since its inception. To this end, Kea provides an automatic consistency manager, freeing the programmer from explicitly dealing with the consequences of changes to user inputs.

In this paper ThermalDesigner is described. This is an application constructed using the code conformance architecture outlined above. The second section describes ThermalDesigner's problem domain. The following two sections illustrate how each component of the architecture is used in ThermalDesigner's implementation. We complete the paper by briefly describing current work aimed at developing a common building model based on a generalisation of the code conformance architecture.

The thermal insulation problem

ThermalDesigner assists the process of checking that a building design meets the requirements of the New Zealand thermal insulation standard for residential buildings,

[1]Department of Computer Science, University of Auckland.

NZS4218P (SANZ, 1977). It embodies an empirical approach to this process developed by the Building Research Association of New Zealand (BRANZ) and published as a (paper) design guide (Bassett et al, 1990).

For designs to comply with NZS4218P, they must have a building performance index (bpi) less than 0.13 kWh m^{-2} (^{o}C day) $^{-1}$ (units as used in NZS4218P) where

bpi = Purchased Heat (kWh) x (Seasonal degree days (^{o}C day) x floor area (m^2))$^{-1}$

The method of Bassett et al (1990) is to calculate empirical estimates of:

- the thermal resistance (R-value) of each element of the building, calculated from properties of materials and building techniques used in construction;
- heat losses related to air infiltration, based on the volume, joint length, together with regional and microclimate effects;
- heat losses of each element of the building based on the relationship between R-values, orientation, and the standard climate of the region;
- internal heat gains, based on the number of occupants and on the window areas and orientations, shading, and location;
- internal and solar heat gains, based respectively on the number of occupants; a factor for standard household equipment is included;
- the net useful heat gain, based on the above heat gain and loss estimates, together with an estimate of the thermal mass level of the building;
- the degree of winter overheating through excess solar gains;
- the amount of energy needed to keep the building at 20 ^{o}C and the bpi.

The ThermalDesigner Building Model

Figure 1 shows part of ThermalDesigner's object-oriented building model, including the principal classes and the major whole-part relationships between them. Thus a building is composed of a number of spaces, each of which includes a number of floors, roofs and walls, etc. Most of the classes shown have associated sub or super classes representing specialisations and generalisations of the concept they embody. The object-oriented model was derived by performing an object-oriented analysis (after the style of Coad and Yourdon, 1991) on Bassett et al (1990) and NZS4218P.

Construction of the object-oriented model also involved identification of attributes of the classes and methods for calculating those attributes. These attributes arise from provisional requirements of NZS4218P and additional data items required as part of the Bassett et al (1990) methodology. The attributes and their methods of calculation are represented functionally in Kea, following the arguments of Fenves at al (1987), but embedded within the class they relate to.

Figure 2 shows how example attributes of the building class are represented as Kea functions. The expressions (function bodies) associated with each attribute define how to calculate a value for the attribute. The functions may refer to other functions, possibly associated with other objects, in order to perform their evaluation. For example, the *total seasonal gain a*ttribute of class Building is calculated as the sum of the Building's *internal heat gain* together with the sum of the *windows heat gain* of each of the Spaces in the *spaces* list associated with the Building. The *internal heat gain* of the building is calculated using one of two empirical formulae, the choice of which depends on whether the number of occupants of the building is known or not.

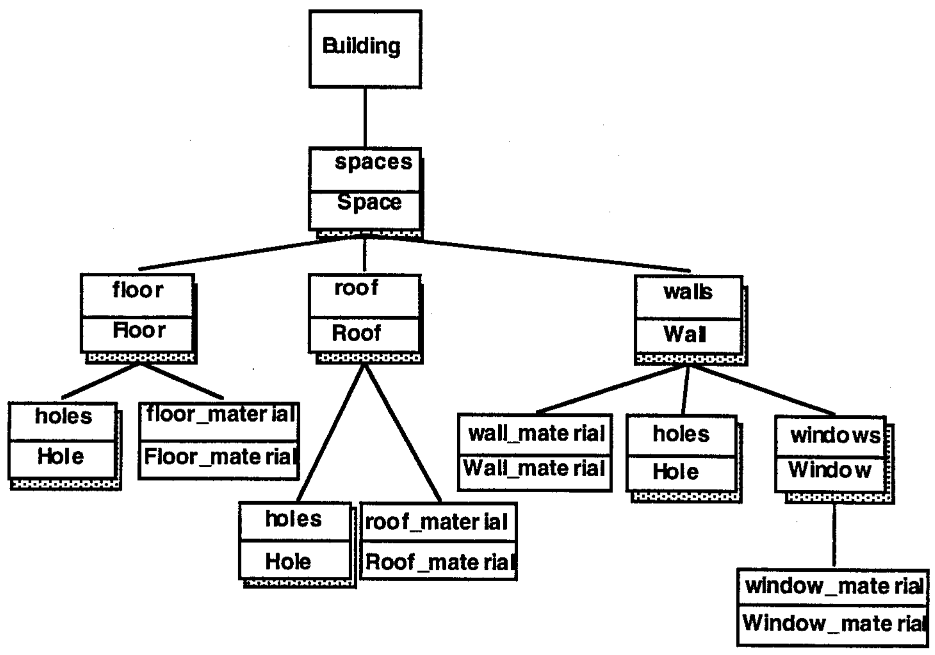


Figure 1: Class structure diagram of ThermalDesigner building model

```
class Building
    spaces: list Space := .... .
    . . .
    gain_loss_ratio: float :=
        total_seasonal_gain / total_seasonal_heat_losses.
    total_seasonal_gain: kWh :=
        internal_heat_gain +
        sum(collect (s in spaces, s^windows_heat_gain)).
    internal_heat_gain: kWh :=
        % Empirical formula: ALF worksheet 30
        900.0 + 150.0 * num_of_occupants if
known_num_occ
        | 1500.                          % the | means
"else"
    total_seasonal_heat_losses: kWh :=
        air_heat_loss +
        sum(collect(s in spaces .....)).
    air_heat_loss: kWh := .... .
    num_of_occupants: integer := .... .
    known_num_occ: boolean := .... .
    . . .
end Building.
```

Figure 2: functional representation of building attributes

Other features of Kea as a programming language can be found in Hosking et al (1991) and Hosking et al (1990). These include:

- strong typing
- multiple inheritance
- dynamic classification of objects at execution time
- object creation functions
- a limited form of procedural control, used to enforce sequence
- a form of daemon called a "when conditional" allowing opportunistic execution. This is useful for error handling.

Interfacing to the building model

The Kea-based code conformance architecture includes two components for interacting with the object-oriented building model:

- PlanEntry: A CAD-like drafting front end to allow users to enter plan details. The plan is translated into corresponding Kea objects and object attributes.
- A forms interface for entry and display of non-geometric information. The use of form classes allow forms to be easily defined and incorporated into an application. Upon creation of a Kea object inheriting from a form class a corresponding form is created and displayed on the screen.

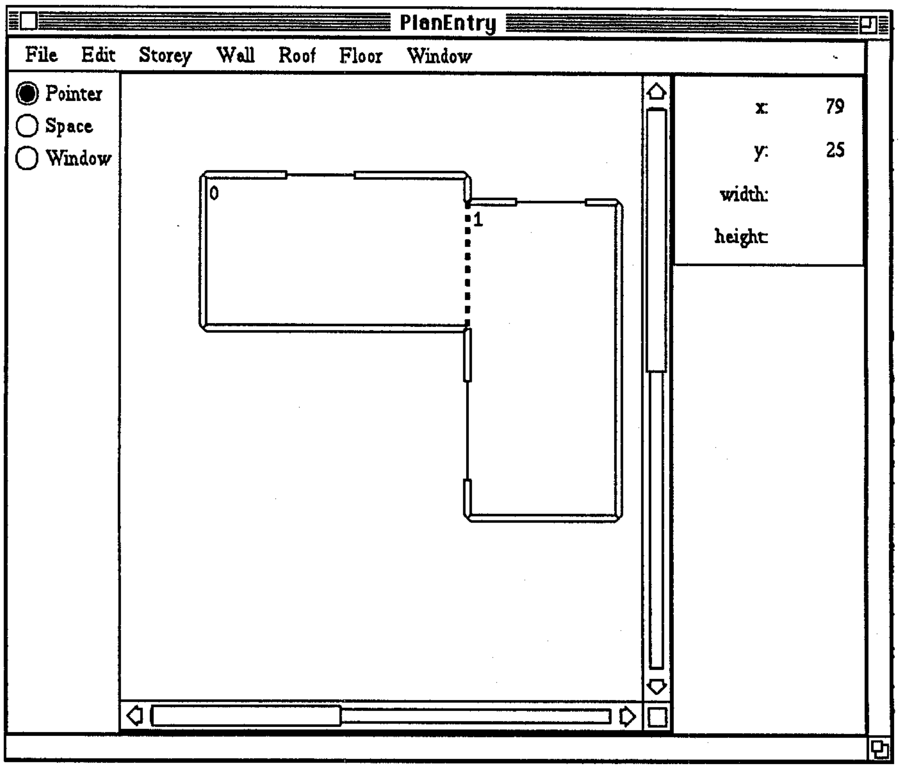


Figure 3: PlanEntry system in use

PlanEntry is a generic package for entry of plan information. Figure 3 shows the PlanEntry system in use as part of ThermalDesigner. PlanEntry appears to the user as a fairly standard, if rudimentary, X-window based CAD-like drafting system, which includes the following features:

- tools for mouse-based drawing of spaces and external openings
- automatic removal of external walls in overlapping spaces
- ability to handle multiple storeys
- standard mouse-based resizing and cut and paste
- ability to associate attributes (eg material types) with plan objects (eg walls, roofs) via menus

Corresponding to each type of plan object is a Kea class. Construction of a plan by a user causes creation of corresponding Kea objects, which have access to the plan's object's attributes. Tailoring PlanEntry for use with an application involves a combination of specialising the Kea plan classes and transforming the generic aggregation structure to one suited for the application. An example of the latter is the elimination of storeys from the aggregation structure needed for ThermalDesigner.

In addition, the attribute information that can be associated with plan objects is completely tailorable. For example, menu entries corresponding to material types are provided in ThermalDesigner. The information in these menus, material names and the Kea objects associated with them, is constructed within ThermalDesigner and passed to the PlanEntry system.

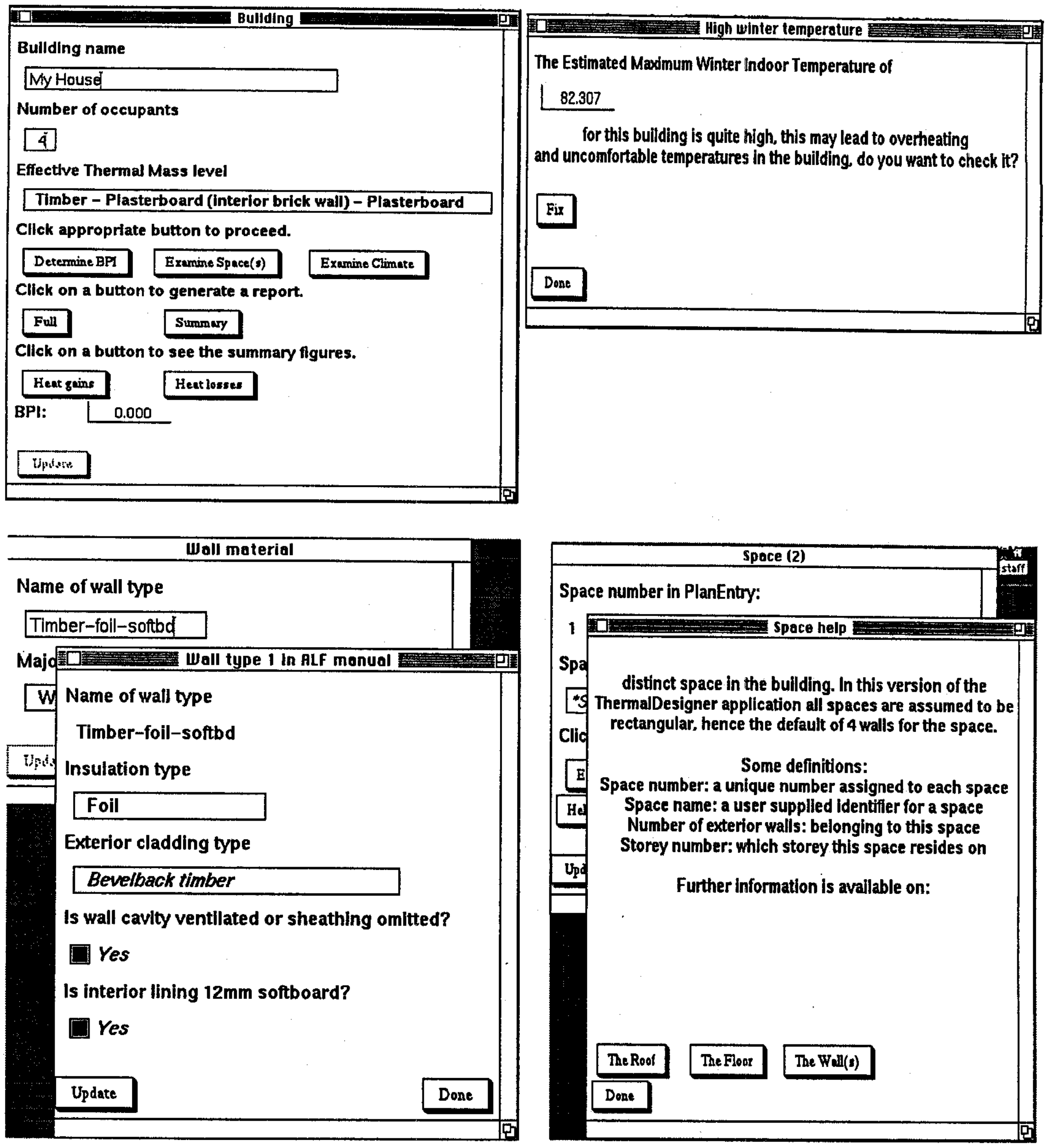
Kea form classes are used to define forms. Forms can include:

- Input (textual, toggle button, and menu) and output fields: Input fields of form objects are like ordinary object attributes but with externally calculated values. Output fields are object attributes which have their values displayed on the form upon calculation.
- Buttons: each having an associated Kea procedure, which is executed on a button click.

Some example ThermalDesigner forms are shown in Figure 4. In ThermalDesigner forms are used for:

- Obtaining location, climate, and owner information (eg the building form)
- Obtaining material description information, for incorporation into the PlanEntry menus (eg the wall material definition forms)
- Providing results (eg the BPI field of the building form)
- Providing help in a hypertext-like manner (eg the space help form and its associated buttons which allow navigation to other forms)
- Providing feedback on design errors (eg the high temperature form)
- Providing control of the running application (eg the building form buttons)

Figure 5 shows the structure of the complete ThermalDesigner system in an iconic form, indicating how the two interface components interact with the building model. An additional textual interface permits summary reports to be generated.


Building

Building name

My House

Number of occupants

4

Effective Thermal Mass level

Timber - Plasterboard (interior brick wall) - Plasterboard

Click appropriate button to proceed.

Determine BPI | Examine Space(s) | Examine Climate

Click on a button to generate a report.

Full | Summary

Click on a button to see the summary figures.

Heat gains | Heat losses

BPI: 0.000

Update

High winter temperature

The Estimated Maximum Winter Indoor Temperature of

82.307

for this building is quite high, this may lead to overheating and uncomfortable temperatures in the building, do you want to check it?

Fix

Done

Wall material

Name of wall type

Timber-foil-softbd

Wall type 1 in ALF manual

Name of wall type

Timber-foil-softbd

Insulation type

Foil

Exterior cladding type

Bevelback timber

Is wall cavity ventilated or sheathing omitted?

■ *Yes*

Is interior lining 12mm softboard?

■ *Yes*

Update | Done

Space (2)

Space number in PlanEntry:

1

Space help

distinct space in the building. In this version of the ThermalDesigner application all spaces are assumed to be rectangular, hence the default of 4 walls for the space.

Some definitions:
Space number: a unique number assigned to each space
Space name: a user supplied identifier for a space
Number of exterior walls: belonging to this space
Storey number: which storey this space resides on

Further information is available on:

The Roof | The Floor | The Wall(s)

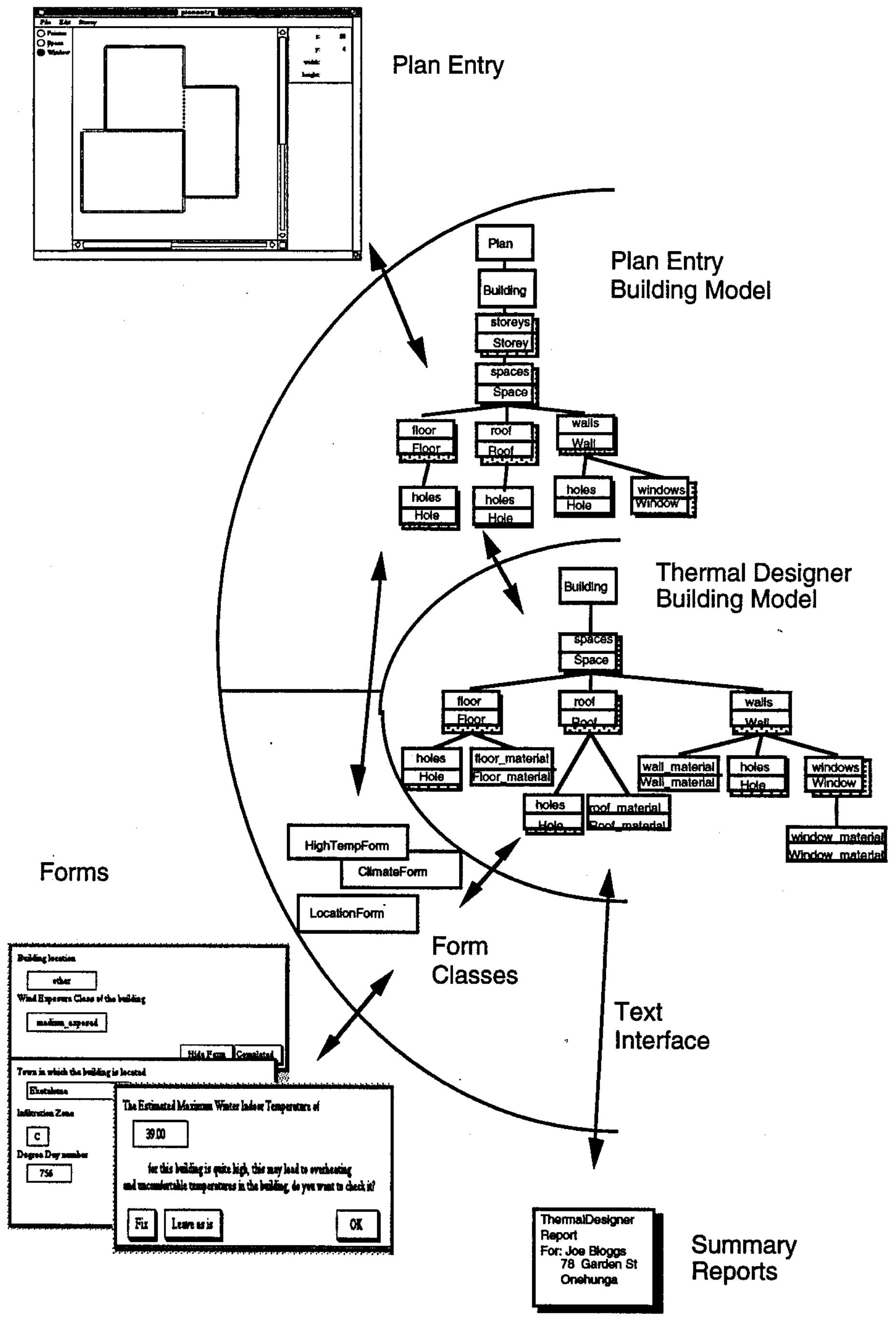
Done

Figure 4: Example forms: building form (top left), high temperature form (top right), wall material definition forms (bottom left), and space help form (bottom right)

Consistency management

An important feature of the Kea approach to modelling is that the programmer programs as if the building is described once with no subsequent modifications to that description. Thus there is no explicit code in the Kea application to handle modifications to the building design. This approach simplifies the application code considerably.

In practice, however, it is essential to permit the user to modify a building design to experiment with different design alternatives, and to correct design flaws. Kea permits the user to modify *any* input, either from a form or from PlanEntry, and automatically propagates the effect of such a change. The state following the design change is as if the application had the modified value as its initial entry. Thus, for example, a user can modify the size of a space drawn in PlanEntry and all results dependent on that, such as the bpi value, are updated to reflect the new space size.



Building location

other

Wind Exposure Class of the building

medium_exposed

Town in which the building is located

Ekatahuna

Infiltration Zone

C

Degree Day number

756

The Estimated Maximum Winter Indoor Temperature of

39.00

for this building is quite high, this may lead to overheating and uncomfortable temperatures in the building, do you want to check it?

Figure 5: Architecture of ThermalDesigner

This is similar to the automatic recalculation facilities of spreadsheets, but is significantly more complicated as changes to a user input can cause:

- recalculation of values of output fields of forms
- invalidation of inputs to a form, as the value modified changes a prompt field on another form

- elimination of forms (and any inputs supplied to them), because the changed value eliminates the reason for the creation of those forms
- creation of additional forms, as the changed value causes additional information to be requested or displayed

An example of the use of this facility is illustrated by the high temperature warning form (Fig. 4). This form comes into existence when the estimated maximum winter indoor temperature (EMWIT) is greater than an empirical threshold. The Fix button on the form assists the user in deciding how best to modify the design to lower the estimated temperature and, in effect, provides a way of navigating to the objects most suitable to modify. When the user decides upon a modification, the effects of the change are propagated. If the EMWIT is now lower than the threshold, the reason for displaying the high temperature warning form is no longer valid and the form is eliminated, providing feedback to the user that the fault is repaired. Otherwise the form is updated with the newly calculated EMWIT value.

Current work

Current work aims to generalise the code conformance architecture to form the basis of a common building model. We are pursuing two projects to obtain information on the best approach to making this generalisation.

The first project involves collaboration with the Victoria University of Wellington group who have been developing the ICAtect common building model (Amor et al, 1991). The development of ICAtect has primarily focussed on integrating several simulation packages (mainly thermal) via a common building model. The collaborative project is investigating the integration of the heuristic-based ThermalDesigner system with the predominantly simulation-tool based ICAtect model. Full details of this project may be found in Amor et al (1992).

In the second project, we are integrating ThermalDesigner with another Kea application, WallBrace. The latter is a loadings code conformance checker and hence concerned with a structural rather than thermal model of the building. Our approach is to generalise PlanEntry and use it, together with appropriate forms, as a building model editor. Common building model schema will be represented as Kea classes. Code specific to ThermalDesigner and WallBrace will make use of the common building model to perform their services, and may supplement the common model information with forms of their own. The proposed architecture is shown in Fig. 6. It is still an open issue as to the best way to modularise the application specific code within the object-oriented framework. We expect that obtaining a suitable solution to this problem will provide an insight into modularity and information hiding requirements within object-oriented systems.

In an ancillary project, we are designing and implementing a low-level graphics interface to Kea. When implemented this will permit the bulk of PlanEntry to be written in Kea, rather than as a monolithic C-based package as it is currently. A Kea implementation will permit PlanEntry to be far more readily tailored than it is currently and will assist in the generalisation of PlanEntry to a building model editor.

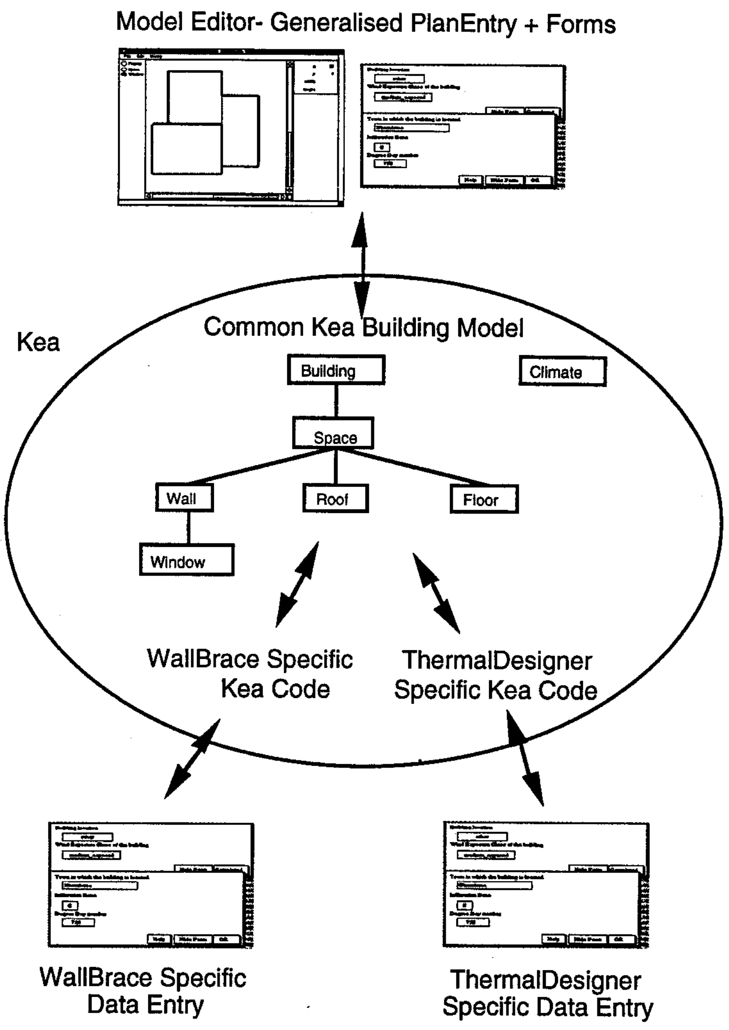

Model Editor- Generalised PlanEntry + Forms
Kea
Common Kea Building Model
Building
Climate
Space
Wall
Roof
Floor
Window
WallBrace Specific Kea Code
ThermalDesigner Specific Kea Code
WallBrace Specific Data Entry
ThermalDesigner Specific Data Entry

Figure 6: Integrating ThermalDesigner and WallBrace

Summary

We have described an application of an object-oriented code conformance architecture, demonstrating the use of each component of the architecture. Particular benefits of using this architecture include:

- the object-oriented approach to modelling the building. This provides a natural framework to embed the computational components of code conformance checking. This framework is reasonably robust against modifications to code of practice provisions (Hosking et al, 1991).
- a functional representation of provisions, which is a preferred approach in most influential work on code conformance (eg Fenves et al, 1987).
- the combination of PlanEntry and the forms interface, which provides a flexible set of tools for developing building model editors. The object-oriented nature of these tools encourages their reuse via appropriate specialisation.
- the ability for users to readily navigate between forms, entering information as desired. This provides the user with considerable control over execution of the system, rather than the user being required to act as a passive information supplier

- the consistency management system. This allows the application to be programmed without the need to include code to handle modifications to the building design. This simplifies the programming considerably, without losing the ability for users to arbitrarily modify their designs to experiment with alternative design decisions and to correct design flaws.

We have also described current work aimed at generalising the architecture to form the basis of a common building model. This consists of two projects to generalise the architecture incrementally in two directions. These involve, on the one hand, integration of heuristic and simulation-based tools in the same application domain (thermal modelling) and, on the other hand, integration of two heuristic-based tools in different application domains (thermal and structural).

Acknowledgements

The authors gratefully acknowledge the support of the Building Research Association of New Zealand and the University of Auckland Research Committee in pursuit of this research.

References

[Amor 91] Amor, R., Groves, L, Donn, M, 1991: Integrating design tools: an object-oriented approach . *Building Systems Automation-Integration, First International Symposium,* University of Wisconsin, Madison, Wisconsin, June.

[Amor 92] Amor, R., Hosking, J, Groves, L, Donn, M, 1992: Design tool integration: model flexibility for the building profession. Accepted for presentation to Building Systems Automation-Integration 1992 Symposium: Computer Integration of the Building Industry, Dallas, Texas, June.

[Bassett 90] Bassett, M.R., Bishop, R.C. and van der Werff, I.S., 1990: *ALF MANUAL, Annual Loss Factor Design Manual,* An aid to thermal design of buildings, Building Research Association of New Zealand, Judgeford, New Zealand.

[Coad 91] Coad P. and Yourdon, E., 1991: *Object-Oriented Analysis Second Edition*, Prentice Hall, New Jersey.

[Fenves 87] Fenves S J, Wright R N, Stahl F I, Reed K A, 1987: *Introduction to SASE: Standards Analysis, Synthesis, and Expression*, Report NBSIR 87-3513 U.S. Department of Commerce, National Bureau of Standards.

[Hosking 90] Hosking, J.G., Hamer, J. and W.B. Mugridge, 1990: Integrating functional and object-oriented programming, *Proc TOOLS Pacific '90*, Sydney, pp. 345-355.

[Hosking 89] Hosking, J.G., Lomas, S., Mugridge, W.B., and A.J. Cranston, 1989: The development of an expert system for seismic loading. *Civil Engineering Systems*, **6** (1-2), pp 27-35.

[Hosking 87] Hosking, J.G., Mugridge, W.B. and M. Buis, 1987: FireCode: a case study in the application of expert systems techniques to a design code. *Environment Planning and Design B*, **14**, pp267-280.

[Hosking 91] Hosking, J.G., Mugridge, W.B., Hamer, J., 1991: An architecture for code of practice conformance systems, in Kahkonen and Bjork (eds) *Computers and Building Regulations*, VTT Symposium 125, VTT Espoo, Finland, pp171-180.

[Mugridge 88] Mugridge, W.B. and J.G. Hosking, 1988: The development of an expert system for wall bracing design. *Proc. NZES'88 The third New Zealand Expert Systems Conference*, Wellington, New Zealand , pp10-27.

[SANZ 77] SANZ, 1977: *NZS4218P 1977: Minimum Thermal Insulation Requirements for Residential Buildings, a New Zealand Standard*, Standards Association of New Zealand, Wellington, New Zealand.