Our Ref: 4376.04

ADDING A CODE CONFORMANCE TOOL TO AN INTEGRATED BUILDING DESIGN ENVIRONMENT

AUCKLAND UNISERVICES LTD

a wholly owned subsidiary of

THE UNIVERSITY OF AUCKLAND

PRIVATE BAG 92019

AUCKLAND

Report prepared for:

The Building Research Association of New Zealand

Date: May 1996

Consultants:

Dr Rick Mugridge
Dr John Hosking
Mr Robert Amor
Department of Computer Science
University of Auckland
Private Bag 92019, Auckland

ADDING A CODE CONFORMANCE TOOL TO AN INTEGRATED BUILDING DESIGN ENVIRONMENT

W.B. Mugridge, J.G. Hosking, R.W. Amor
Auckland Uniservices Ltd
Private Bag 92019
Auckland

ABSTRACT
We describe the conversion of ThermalDesigner, a code conformance tool, and its connection to an integrated building model. The original version of ThermalDesigner was developed as a stand-alone application in Kea. It has been converted into Snart, the primary implementation language of our integrated building design environment. Snart has been extended with constraints over collections of objects, in order to implement code provisions as specified in ThermalDesigner and provided by Kea. VML, a declarative mapping language, has been used to integrate ThermalDesigner with the IDM.

1. Introduction

The overall aim of our research work is to explore consistency management, data modelling, and user interface issues that arise in the development of integrated building models. We have taken an evolutionary approach to this task, with the aim of building small but real systems in order to explore the range of issues that arise in the integration of multiple tools over time (Mugridge and Hosking, 1995). As it is likely that building data models will have to evolve as building technology evolves, we have assumed that change is pervasive. Unlike other approaches, we have begun with incomplete data models and developed small integrated systems with those models, tackling such issues as the appropriate model to be presented to the user of several tools. Our work has also been innovative in our use of constraint definition and propagation for consistency management within a tool, and the use of a declarative mapping language for managing data consistency between tools via an Integrated Data Model (IDM). Other approaches have taken a traditional programming approach to internal tool consistency and a batch approach to consistency between tools.

Here we report on the next step in our research programme: the integration of ThermalDesigner, a code of practice conformance tool, into a common building model. ThermalDesigner was originally developed as a stand alone application in Kea (Amor et al, 1992). In order to integrate it in our common building model architecture, it was necessary to first convert it to Snart. Conversion was eased by the fact that both languages are object-oriented and constraint-based. The code provisions of ThermalDesigner are encoded as uni-directional constraints in Kea, which map well to the multi-directional constraints of Snart. However, Snart lacked facilities to handle constraints over collections of objects; hence Snart constraints were extended in this project to include these facilities.

In addition to converting code provisions from the Kea implementation of ThermalDesigner, it was necessary to translate the forms interface and user control facilities existing in the Kea functional language to facilities in the Snart/ICAtect common building model framework, and to change the manner in which plan information is provided to ThermalDesigner. The IDM plays a central role in providing data to ThermalDesigner that was, in the Kea system, gathered by ThermalDesigner itself.

The remainder of this report is organised as follows. Section 2 introduces the original ThermalDesigner, written in Kea, and discusses some of the shortcomings of this approach. Section 3 introduces the Snart/ICAtect integrated design model and discusses the important role of VML to define mappings between the IDM and each tool. The following section discusses the conversion of ThermalDesigner to Snart, which was enabled by the similarities between many of the features of Kea and Snart. Section 5 briefly discusses the way in which change is handled automatically in Snart, illustrated by an example from ThermalDesigner. Section 6 discusses other work and the final section concludes.

2. ThermalDesigner in Kea

ThermalDesigner (Amor et al, 1992) helps a designer to check that a building design meets the requirements of the New Zealand thermal insulation standard for residential buildings, NZS4218P (SANZ, 1977). It is based on an approach developed by the Building Research Association of New Zealand (BRANZ) as a paper design guide (Bassett et al, 1990).

ThermalDesigner was developed in Kea, a system that provides an architecture for developing code conformance systems (Hosking et al, 1991). The Kea architecture includes a number of interrelated components, as shown in Fig. 1. An object-oriented building model is used to represent the structure of buildings and their components (such as floors, walls, and roofs). Within this model are embedded code provisions, represented as functions (uni-directional constraints). A PlanEntry module (written in C) allows a user to enter plan details of a building, while a forms interface is provided for the entry and display of other information. Finally, a consistency manager handles changes to user-supplied data (from plan entry or forms), ensuring that the effects of those changes are automatically propagated through the functions to compute new results (this is an extension of the dataflow model that incorporates object-orientation). Hence the programmer does not need to explicitly deal with the consequences of changes to inputs, considerably simplifying the programming task and reducing the size of the system. Kea has been used to implement several code conformance applications (Hosking et al, 1987; Mugridge and Hosking, 1988; Hosking et al, 1989).

Some of the structure, code provisions and procedural code within the class *Building* of the ThermalDesigner tool are shown in Fig. 2. For example, the value of the attribute total_seasonal_heat_losses is a function of the heat loss of each of the elements of the external envelop of the building (floors, walls, windows and roofs). It is recalculated whenever the values it depends on are changed (in a similar manner to the recalculation that occurs in a spreadsheet on a change). Thus, it will be recalculated when a new window is added to the building or when the heat loss of an existing window is changed. This in turn will lead to the recalculation of the values of other attributes that depend on total_seasonal_heat_losses, including net_heating_EU and bpi.

The Kea code of ThermalDesigner is responsible for creating the component structure of the building being modelled (a building consists of spaces, each of which have walls with windows, roofs, and floors). For example, the attribute spaces of the class Building refers to a list of Space objects; num_of_spaces of which are created. If the value of num_of_spaces changes, the number of elements in the set is automatically increased or decreased appropriately, retaining any objects that have already been created. The retention of existing objects in sets such as spaces is essential, because the user may have entered information about some of them through the forms interface.

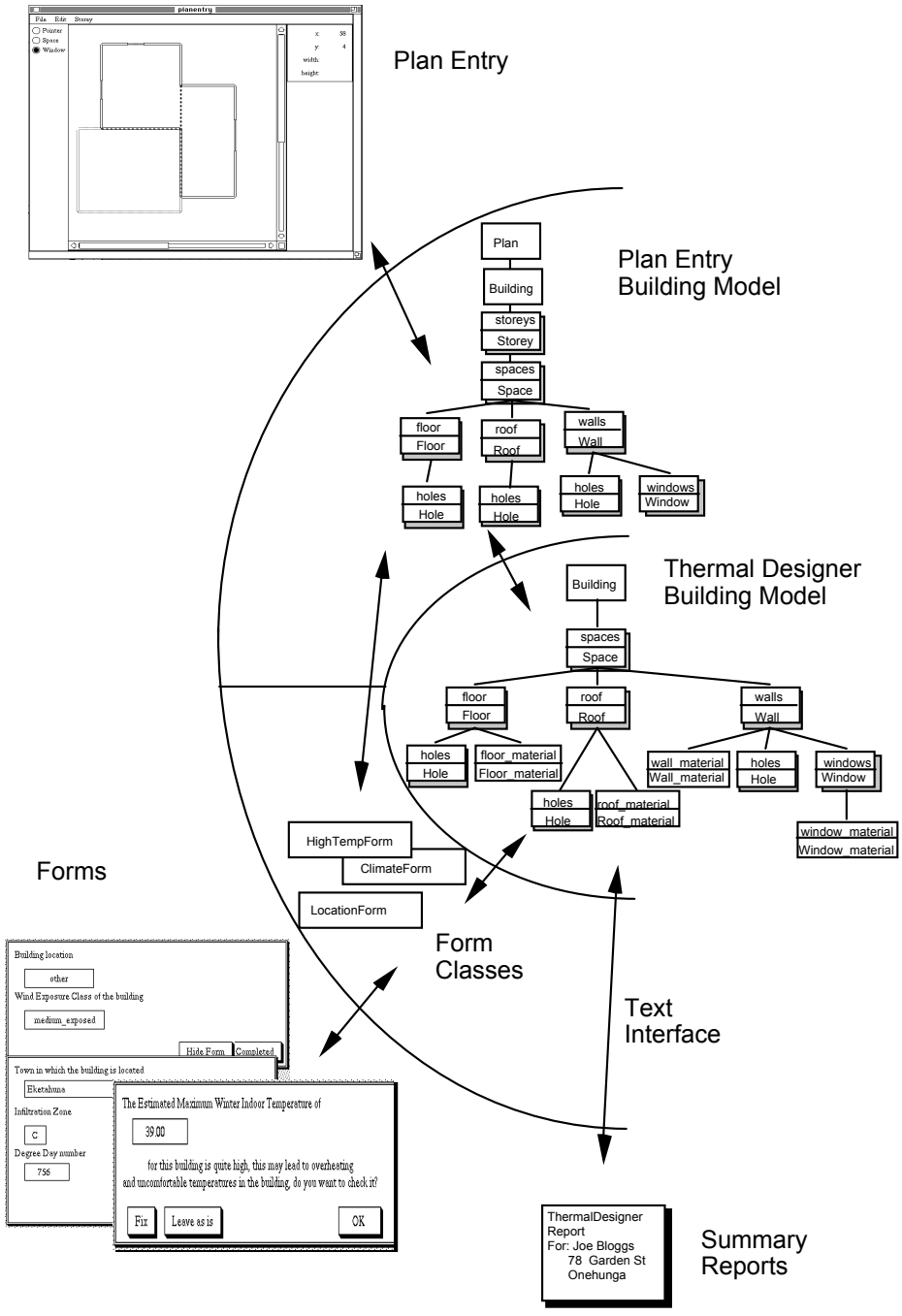

Figure 1: Architecture of Kea-Based ThermalDesigner

The procedure climate_button is responsible for showing a form for the entry of climate information associated with the building. This procedure is called when the user clicks on a button in the main form of ThermalDesigner, which is displayed when the application is used. Other procedural code, which is not shown here, is used for displaying the results of checking the code conformance of a building in the form of a report.

```
class Building.
spaces : list Space
       := collect(r in 1..num_of_spaces, new Space(space_num := r))
            if num_of_spaces > 0
        | [].
total_seasonal_heat_losses : float := air_heat_loss
       + sum(collect(f in spaces, head(collect(r in f^floor, r^heat_loss))))
       + sum(collect(w in spaces, w^walls_heat_loss))
       + sum(collect(r in spaces, head(collect(f in r^roof, f^heat_loss))))
       + sum(collect(i in spaces, i^windows_heat_loss)).

total_seasonal_gain : float
       := internal_heat_gain + sum(collect(i in spaces, i^windows_heat_gain)).
glr : float := total_seasonal_gain / total_seasonal_heat_losses.
useful_fraction : float
       :=   1.07870 - 0.42465 * glr + 0.0518650 * glr ** 2 + 0.0043706 * glr **
3
            if (effective_thermal_mass =< 0.15)
        | 1.10220 - 0.38544 * glr + 0.0075756 * glr **2 + 0.0147310 * glr ** 3
            if  (effective_thermal_mass  >  0.15  and  effective_thermal_mass  =<
0.45)
       ...
useful_heat_gain : float := total_seasonal_gain * useful_fraction.
net_heating_EU : float := total_seasonal_heat_losses - useful_heat_gain.
bpi : float := net_heating_EU / (climate^degree_days * floor_area_total).
...
procedure climate_button.
       call show_form of climate
       call determine_climate of climate
       call hide_form of climate
end climate_button.

end Building.
```

Figure 2: Example Kea class in ThermalDesigner

As we see later, the code provisions are incorporated into the new system with minor syntactical changes but the forms-based interface and related procedural code are largely replaced by the connection to the IDM[1].

A number of limitations of the Kea architecture have been identified:

• Such applications are stand alone; there is a strong need to integrate them with other applications to avoid redundant data entry and to ensure consistency between the data supplied to multiple tools. For example, a change to the design of a building may necessitate making changes to the input of a several different tools if they are stand alone. This means that the model of some tools may not be changed correctly and others may not be changed at all. An integrated approach solves this problem by sharing data between tools (Mugridge et al, 1995). If data sharing occurs dynamically, where a change in one tool may be immediately transferred to other affected tools, the exploration of the effects of a change through multiple tools at once provides even more timely and effective support to a designer.

[1] See Fig. 6 for the corresponding Snart class td_building.

• It is inconvenient for the user to have to enter some information on the plan and other, related information through forms. This requires that a naming scheme for elements of the plan is available. It would be better to provide for data entry and result information to be more directly related to the plan, either by overlaying such information on the plan or by providing hyper-links from the plan to associated data. However, different tools require different information to be associated with the plan, so the PlanEntry module needs to be easy to extend in order to provide this flexibility. The original PlanEntry module is, however**,** a large, complex program, written in C, that is difficult to extend. The use of a higher-level language and a better architecture would solve these problems.

• The Kea consistency model is based on functions (uni-directional constraints) and this reduces flexibility in the design of inputs in the forms user interface. It means, for example, that if either the height or the slope of a gable-end roof is sufficient to determine the other when roof plan information is known, a decision has to be made as to which the user will supply. If the height is required as input, but the user requires the roof to have a particular slope, they are forced to calculate the corresponding height by hand. A multi-directional constraint approach would avoid this type of problem by allowing either the height or the slope to be changed; the other value is calculated automatically.

3. Integrated Design Model

The problems of the previous section have been solved with the Snart/ICAtect model (Amor, 1990; Amor et al, 1990). This provides an integrated design model (IDM) which allows for the integration of a number of tools, including generic ones for plan entry and materials selection. It is based on Snart, an object-oriented extension of Prolog that incorporates multi-directional constraints (Mugridge et al, 1995) and VML, a mapping language that allows for the declarative specification of mappings between the data models of tools and the central IDM (Amor, 1994; Amor et al, 1995). The architecture of this model is shown in Fig. 3, with VISION3D, a tool for visualising and manipulating 3-D models, and the new ThermalDesigner system used for illustration.

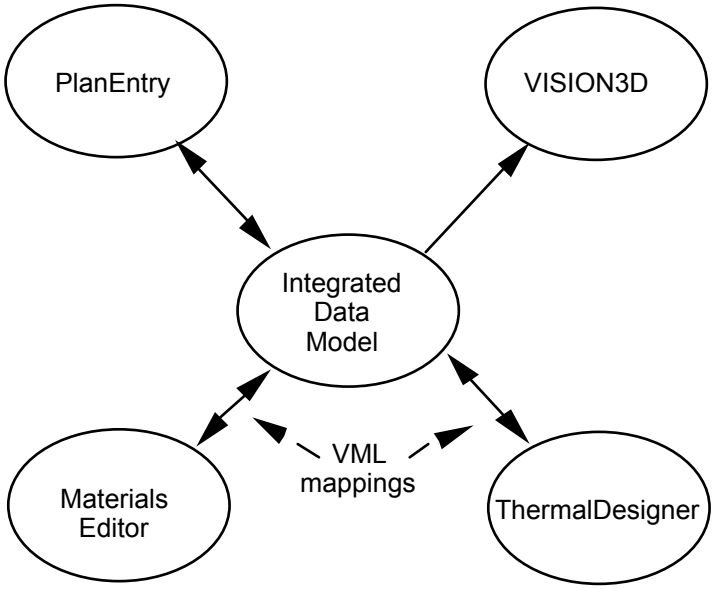

Figure 3: Architecture of the IDM-based Approach

The PlanEntry tool here is different from the one used with the Kea system. Snart was used in a recent project to develop a new multi-view PlanEntry tool that makes considerable use of constraints, thus drastically reducing its size and complexity (Hosking et al, 1994). This new tool enables hyperlinks from elements of the plan (such as a wall segment) to other tools to be easily added. This is used, for example, to provide links from the components of a plan to the MaterialsEditor, which allow for the selection of materials for the faces (walls, floors, etc) of a building. Such material selection replaces the restricted approach to selection that was used in the previous, Kea-based system, and provides materials information to the new ThermalDesigner system in the form of R-values.

The Integrated Data Model (IDM) is central to the architecture. This data model stores an integrated, object-oriented data model of a building. VML defines mappings between this model and the model of each of the attached tools. Each tool is only interested in a subset of the IDM and may organise that data in a different manner from the IDM (and indeed the data of VISION-3D is relational, rather than object-oriented). For example, ThermalDesigner needs no information about the internal geometry of the building; only the external envelope (including windows and materials) is of interest.

The mapping definitions are used to propagate changes from one model to another, through the IDM. A change to the building model made in the PlanEntry tool, for example, will be mapped to a corresponding change in the IDM, which in turn leads to changes in other tools that are affected by that change (according to the mappings defined in VML). For example, the addition of a new window in a wall indirectly leads to a change in the data model of ThermalDesigner, due to the two stages of mappings that are carried out. In this project, VISION-3D has been used purely as a visualisation system, so the mappings are one-way from the IDM, as opposed to the two-way mappings between the IDM and the other tools.

```
inter_class([idm_space_face, idm_material_face, idm_building], [td_window],
     invariants(
          idm_space_face.type_of_face = 'opening',
          idm_space_face.location = 'ext',
          idm_material_face.type_of_face = 'opening',
          plane_equivalence(idm_space_face.plane, idm_material_face.plane),
          point_equivalence(idm_space_face.min, idm_material_face.min),
          point_equivalence(idm_space_face.max, idm_material_face.max)
     ),
     equivalences(
          idm_space_face.max=>x - idm_space_face.min=>x = width,
          idm_space_face.max=>y - idm_space_face.min=>y = height,
          idm_material_face.material=>shading_coefficient = shading_coef,
          idm_material_face.material=>r_value = r_value
     ),
     initialisers(
          td_window.shading_coef = 1.0,
          td_window.r_value = 0.23,
          td_window@create(idm_space_face.orientation,
                           idm_building.environment=>degree_days)
     )
).
```

Figure 4: Example Mapping between the IDM and ThermalDesigner

Fig. 4 shows an example of the use of VML to define mappings between the IDM and ThermalDesigner. Mappings are defined between two or more classes; in this case between `idm_space_face, idm_material_face,` and `idm_building` from the IDM and `td_window` of ThermalDesigner . The *invariants* specify conditions under which this mapping is applicable. In this case, for example, the `idm_space_face` must be an external opening, the `idm_material_face` must be an opening type, and the two idm objects must have conicident location. If the invariants apply, the equivalences specify mappings between attributes. Thus, for example, the `td_window` width is calculated from the `idm_space_face max=>x` and `min=>x` attributes. Initialisers specify default attribute settings and method calls to make when one of the objects involved in the mapping does not exist and requires creation. In this case, initialisers are provided for `td-window` objects

Some of the mappings between the tools are complex. For example, PlanEntry models the building in terms of cuboids, while the MaterialsEditor deals with surfaces on a plane. A mapping is required between the cuboids and planes, taking account of abutment between them.

VML also permits procedural mappings to be defined. This means that both data and events may be mapped between tools through the IDM. Procedure mappings are used to implement the hyperlinks from PlanEntry to the other tools. For example, when the user selects a plane of a building in PlanEntry a procedure call mapping causes that the MaterialsEditor to display all the walls on that plane, ready for defining their materials (by overlaying the materials information over the walls).

VML takes a transaction-based approach to managing consistency between models. More than one concurrent user can use tools to manipulate a building model, however, such changes are only made in the local data model of each tool. When a user is ready to commit the changes to the IDM (which may be done in coarse or fine-grained manners), the ICAtect system ensures that there are no inconsistencies between the local model and the current version of the IDM (see Amor, 1994 for further details).

The major way in which the new architecture differs from the Kea-based one is in the almost complete separation of input facilities from the applications in the former. In the old ThermalDesigner, the application itself is responsible for gathering information from the PlanEntry system and for controlling the display of forms for data entry and the display of results. In the new ThermalDesigner system, the application itself is rather passive; the initial procedural code is responsible for displaying a single form for a few ThermalDesigner-specific inputs and for the display of ThermalDesigner results. Almost all of the data values encoded within the application are changed from the outside by the mapping system (due to changes to the plan and materials information, managed by generic modules). This accentuates the distinction between the interface and the application code that computes according to input data. Through the use of multi-directional constraints, as supplied by Snart, the application code is almost completely declarative; data value changes are automatically propagated through the constraints of the application code, leading to changes in the displayed ThermalDesigner results.

4. Conversion of ThermalDesigner to Snart

The conversion of ThermalDesigner from a Kea-based implementation to one based on Snart was reasonably straightforward because of the close match between the two language systems. However, the conversion of a tool based on a traditional programming language (such as C++) or traditional CAD environment (such as AutoCAD) would be problematic, due to the large amount of procedural code that would have to be modified to take account of the change from a stand alone system to one that depends on an IDM for input and which must react appropriately to changes in that IDM. This issue is discussed further in the last section of this report.

The conversion of ThermalDesigner involved a number of changes, which resulted in an overall reduction in code size and complexity of the application. These changes were as follows:

- Minor conversion of code provisions within classes due to syntactical differences between Kea and Snart. For example Figure 5 shows some elements of the Snart class *td_building* that correspond to the Kea class of Figure 2. The Kea code provisions are encoded in Snart as uni-directional constraints. In some cases the form of code provisions was changed considerably due to the use of Prolog predicates to handle tables, rather than the if-then-else form of Kea. For example, compare the provisions for the attribute useful_fraction in the Kea and Snart implementation fragments shown in Figures 2 and 5 respectively.

```
class(td_building,
   features(
        building_name: text,         % Defined from IDM
        num_of_occupants: int,       % Defined from IDM
        effective_thermal_mass: float,    % Defined from IDM
        spaces: list(td_space),      % Defined from IDM
        climate: td_climate,         % Defined from IDM
        create),
   constraints(
     total_seasonal_heat_losses: float
        := air_heat_loss
        + sum(collect(s in spaces, s@floor@heat_loss + s@roof@heat_loss
                        + s@walls_heat_loss + s@windows_heat_loss)),
     total_seasonal_gain: float
          := internal_heat_gain
               + sum(collect(s in spaces, s@windows_heat_gain)),
     glr: float := total_seasonal_gain / total_seasonal_heat_losses,
     useful_fraction: float := useful_fract(effective_thermal_mass,glr),
     useful_heat_gain: float := total_seasonal_gain * useful_fraction,
     net_heating_EU:     float     :=     total_seasonal_heat_losses     -
useful_heat_gain,
     bpi:    float    :=    net_heating_EU    /    (climate@degree_days    *
floor_area_total),
     ...
     ),
  demons(
    cw_set_item('Thermal Designer',bpiText,number_atom(bpi))
)).

useful_fract(ETM,GLR,V) :-
     ETM =< 0.15, uf(1.07870, 0.42465, 0.0518650, 0.0043706, GLR, V).
useful_fract(ETM,GLR,V) :-
     ETM =< 0.45, uf(1.10220, 0.38544, 0.0075756, 0.0147310, GLR, V).
...
uf(A,B,C,D,GLR, V) :- V is A - B * GLR + C * GLR * GLR + D * GLR * GLR *
GLR.

td_building::create :-
  set_prop(td,building,self),
  td_dialog.  % Set up the top-level TD dialog
```

Figure 5: Class td_building in the Snart-based ThermalDesigner

• As Snart did not previously handle constraints over collections of objects (*collect* in Kea), the constraint system of Snart was extended to include them in a syntactic form similar to that of Kea.

• Elimination of code to construct components and subcomponents of the building. This is now handled completely by mappings between the PlanEntry module and ThermalDesigner (via the IDM): *idm_space* objects are defined by the IDM and mapped to *td_space* objects in ThermalDesigner.

• Elimination of many Kea forms definitions that were made redundant by the MaterialsEditor, the new materials selection module that is hyper-linked from the building plan. This also avoids the use of names to refer to elements of the building, as needed in the Kea-based system. However, tool-specific information was encoded in the general-purpose MaterialsEditor due to the feedback requirements of the user interface. It is necessary to allow for either the display of an R-value, based on the material that has been selected, or the input of an R-value when the designer has chosen materials unknown by the system (or has not decided on which material should provide the required R-value). It may

be better to avoid the incorporation of such application-specific code into the MaterialsEditor, by incorporating complete data for material selections and the resulting R-value in the IDM, with the MaterialsEditor remaining general-purpose and the ThermalDesigner taking responsibility for calculating the R-value from the selected materials (where possible).

• The definition of VML mappings between the IDM and the ThermalDesigner schema; an example is shown in Fig. 4 .

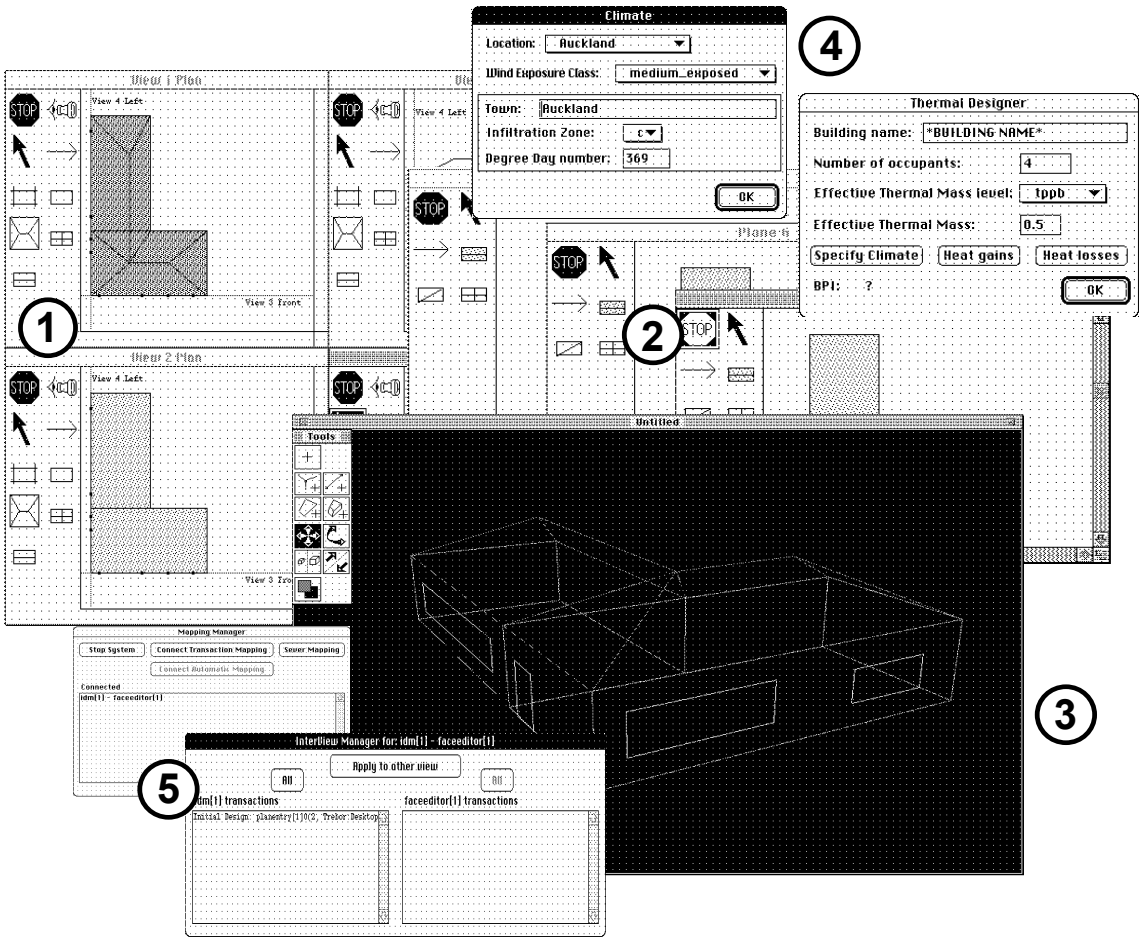

Fig. 6: Composite screendump of the Snart implementation of ThermalDesigner in use. 1. PlanEntry views of building. 2. FaceEditor views of building planes. 3. Vision3-D visualisation of building. 4. ThermalDesigner-specific forms. 5. Mapping manager forms

• Conversion of four Kea-based forms that are central to the ThermalDesigner application into Prolog dialog boxes. Any change to an input field on a dialog box leads to an assignment to the corresponding attribute of the underlying ThermalDesigner object. For example, when the 'Number of occupants:' field is changed in the 'Thermal Designer' dialog, the num_of_occupants attribute of the underlying building object is changed. Any change to an output field of a dialog is handled through the use of demons. A demon executes whenever the value of any of the attributes it references changes. For example, the demon at the end of the definition of class *td_building*, as shown in Fig. 5, ensures that

whenever the attribute bpi changes, the bpiText field in the 'Thermal Designer' dialog is changed (through a call to the Prolog predicate *cw_set_item*).

The *create* method of class building, as shown in Fig. 5, is used to procedurally create the main ThermalDesigner dialog when an object of class building is first created.

Fig. 6 shows a screen dump of ThermalDesigner in use together with the other components of the demonstration integrated environment.

5. Handling Change in ThermalDesigner

Changes to values may be due to either changes to fields in dialog boxes managed by ThermalDesigner or due to changes mapped through from the IDM, following changes made in the PlanEntry or MaterialsEditor tools. Due to the use of constraints within the ThermalDesigner system, such changes are automatically propagated and dependent results recalculated. For example, consider when the value of the shading coefficient of a window (represented by object o34) within a wall (object o33) of a space (object o30) in a building (object o28) is changed to 0[2]. This results in a number of other attributes of that window and the other objects being automatically updated through constraint propagation, illustrated as follows:

```
o34@(shading_coef:=0)
 o34@heat_gain := 0 (:=o34@area*o34@shading_coef*agfR(o34@... in td_window)
  o33@solar_gain_total := 0 (:=sum(collect(w in o33@windows... in td_wall)
   o28@window_N_gain := 358.46 (:=sum(collect(s in o28@spaces, su... in td_building)
   o30@windows_heat_gain := 408.45 (:=sum(collect(w in o30@walls,... in td_space)
    o28@total_seasonal_gain := 2675.37 (:=o28@internal_heat_gain+su... in td_building)
     o28@useful_heat_gain := 2591.34 (:=o28@total_seasonal_gain*o28@... in td_building)
      o28@net_heating_EU := 4754.77 (:=o28@total_seasonal_heat_losses... in td_building)
       o28@bpi := 0.08 (:=o28@net_heating_EU/(o28@climate@degree_da... in
td_building)
     o28@gain_loss_ratio := 0.364 (:=o28@total_seasonal_gain/o28@t... in td_building)
      o28@glr := 0.364 (:=o28@gain_loss_ratio in td_building)
       o28@useful_fraction := 0.986 (:=useful_fract(o28@effective_th... in td_building)
        o28@useful_heat_gain := 2638 (:=o28@total_seasonal_gain*o28@... in td_building)
         o28@net_heating_EU := 4707 (:=o28@total_seasonal_heat_losses... in td_building)
          o28@bpi := 0.08 (:=o28@net_heating_EU/(o28@climate@degree_... in td_building)
```

The final change to the bpi attribute in turn leads to the firing of the demon in class *building* and thus the bpi field is updated in the dialog box.

Hence defining code provisions with Snart constraints means that the programmer of the ThermalDesigner system does not need to write procedural code to handle changes to data values. Instead, the changes are propagated automatically. The use of such constraints can

[2] The code for the classes involved is given in the Appendices.

drastically reduce the size and complexity of an application, making it easier to build and maintain.

There are two levels of consistency management in the integrated system. At a local level, the design tools implemented in Snart (PlanEntry, MaterialsEditor, and ThermalDesigner) use Snart's local propagation constraint system to maintain internal consistency. Between tools (and the IDM) the ICAtect VML mapping manager is used to manage consistency in a transaction-oriented fashion. When used in a fine-grained mode, where each transaction corresponds to an individual attribute change, object creation, or procedure call, the VML mappings have a very similar effect to the local constraint propagation mechanism. When used in a coarse-grained mode, the VML mapping mechanism "bulks up" a sequence of changes before commiting them to the IDM (and then on to other tools). This necessitates additional machinery to deal with conflicts resulting from merging the changes to and from the IDM. This is discussed further in Amor (1996).

6. Related Work

While much research has been carried out on the schemas for integrated design models (Björk and Pentilla, 1989; Carroll et al, 1991; Augenbroe, 1992; STEP, 1992), little attention has been given to the design and construction of software architectures that are suitable for managing an IDM. Our work is unusual in that we have started with the assumption that data mappings between tools will be most useful when they are dynamic, rather than necessarily assuming a batch mode of operation. Our work is also unusual in our use of constraint-based languages for the development of tools that are integrated through an IDM, thus simplifying the task of building new tools that are to be integrated. While our IDM architecture supports such a dynamic model, it also allows for traditional, procedure-based tools, which must explicitly manage changes to data values.

The work that is closest to the research reported here is that of Eastman (1995) and Eastman et al (1995) in developing the EDM-2 system. EDM-2 is organised around a database model. It also makes use of mappings and constraints, but there are several major differences from our work:

- Constraints are only used in the declaration of the IDM. They are defined for two purposes: as database-style constraints to determine whether a changed data model is consistent with respect to the constraints (to ensure across-tool consistency) and as functions which compute new values when data values change. There are two forms of consistency constraints: variants need not be satisfied (at the users discretion) while invariant constraints must be satisfied, otherwise a commit is aborted.

- EDM-2 constraints and mappings are written in C and thus do not enjoy the declarative benefits of Snart constraints.

- EDM-2 includes "accumulations" to allow for external code to compute values. When any of the preconditions of the external code are changed, the results (post-conditions) of the external code in the IDM are marked as inconsistent. Hence other applications know that data is invalid at the moment and any affected external code can be automatically scheduled for execution (assuming that only one tool is available for

computing the given result). This approach is functional, as compared to the more general mappings provided by VML.

7. Conclusions

We have described the conversion of ThermalDesigner, a code conformance tool, and its incorporation into an integrated building design system. This exercise has shown that the conversion from a Kea-based system to a Snart-based one has been reasonably straightforward. This has been due, in large part, to the similarities between Kea and Snart; both are object-oriented and constraint-based. With the inclusion of constraints over collections of objects in Snart, as carried out as a part of this project, it now has a more general form of constraints than Kea.

The almost complete elimination of input/output handling (and associated procedural code) was the most interesting change made to ThermalDesigner, due to the automatic mappings between the IDM and ThermalDesigner, as defined by VML mappings. Resulting from the use of constraints to define code provisions, little procedural code remains in ThermalDesigner, accentuating the benefits of representing the code provisions in a declarative, constraint-based manner.

As noted in Section 4, the conversion of tools that have been developed using a traditional approach would be less straightforward. In such tools, the mappings between the IDM and the tool concerned would have to make use of method mappings in order to procedurally inform the tool that some change has been made. The tool would have to be explicitly programmed to take account of all such changes and to recompute affected outputs.

The integration of ThermalDesigner into the ICAtect/Snart framework has increased our confidence in our approach. However, a number of issues remain. A major hindrance to the full-scale use of this approach is due to the implementation vehicle: Snart based on Prolog. It has been an excellent approach for testing out research ideas, but it is now approaching limits. Not only is Snart not seriously used beyond a few research projects, it is also too slow to be a viable platform for future larger-scale work. Hence work is underway to convert the results of this work onto a more suitable platform: C++. We plan to extend C++ with the constraint facilities developed in Snart and then to extend those to incorporate VML. This development will raise new issues, such as the best means of providing persistence for the IDM and other tools, the best object model to use (such as SOM), and the best way of handling transactions in a distributed approach to integration. Longer term, the suitability of an extended C++ for developing new tools to be incorporated with the IDM will also need to be evaluated.

Another potential issue arises from the need to more closely integrate the user interfaces of a set of tools. It is likely to be difficult to integrate them cleanly without considerable effort in changing the tools themselves. For example, the handling of R-values between the MaterialsEditor and ThermalDesigner was less than satisfactory. A new approach is needed to manage such interactions in a more general way, so that the specific interactions of all pairs of tools do not have to be considered (just as the IDM avoids this for data mapping). Further work is needed in understanding the issues of user interface integration before solutions can be found.

Finally, further work is needed on handling global consistency. While each tool should ensure consistency in the data that it manipulates from the IDM, there will arise consistency issues between the data that is manipulated by different tools. As the IDM definition is implemented in Snart, the ability to define constraints within the IDM is already available. A related issue is in determining that the data results supplied by a tool to the IDM may be out of date, because the input data on which that tool depends have been changed since it was last applied to the IDM. Rather than trying to automate the application of tools, as discussed by Eastman (1995), we suspect that it is better to provide appropriate information to users about potentially "old" data; users can then apply tools as they are ready, in order to bring the IDM data into global consistency. Unlike the approach of Eastman, the Ictatect approach makes use of version control information on the data itself to determine whether the results of a tools may be out of date and to schedule tools that are potentially runnable.

References

AMOR, R.W., "ICAtect: integrating design tools for preliminary architectural design", MSc thesis, Department of Computer Science, Victoria University of Wellington, NZ, 1990.

AMOR, R.W., PhD thesis, Department of Computer Science, University of Auckland, 1996 (in preparation).

AMOR, R.W., GROVES, L., DONN, M.R., "Integrating design tools: an object-oriented approach", procs. *Building Systems Automation-Integration '90*, University of Wisconson, Madison, June 1990.

AMOR, R.W., HOSKING, J.G., MUGRIDGE, W.B., HAMER, J., WILLIAMS, M. “ThermalDesigner: an application of an object-oriented code conformance architecture”, *Proc. CIB Joint International Workshop on Computer Integrated Construction and Computing and Building Standards*, Montreal, Canada, May 1992.

AMOR, R.W. "A mapping language for views", Internal report, Department of Computer Science, University of Auckland, 1994.

AMOR, R.W., HOSKING, J.G., MUGRIDGE, W.B. "A declarative approach to inter-schema mappings", Procs *CIB W78/TG10 Conference on Modeling of Buildings through their Life-cycle*, Stanford, August 1995.

AUGENBROE, G. "COMBINE; a joint European project towards integrated building design systems", procs. *Building Systems Automation-Integration '92*, pp10-12, Dallas Texas, 1992.

BJÖRK, B-C. and PENTILLA, H. A scenario for the development and implementation of a building product model standard, *Current Research and Development in Integrated Design Construction and Facility Management*, CIFE, Stanford Ca, 28-29 March, 18pp (1989).

CARROLL, W.L., SELKOWITZ, S.E., WINKLEMANN, F.C. "The energy design advisor: a new desktop design tool based on DOE-2", a white paper for discussion and comment, Lawrence Berkeley Laboratory, USA, 1991.

EASTMAN, C.M., CHO M.S., JENG T.S., ASSALL H.H. "A data model and database supporting integrity management", in *International Computing Congress in Civil Engineering*, ACSE, Atlanta Georgia, June 1995.

EASTMAN C.M., 1995. "Managing integrity in design information flows", to appear in *Computer Aided Design*, IPC Press.

HAMER, J., MUGRIDGE, W.B., and HOSKING, J.G. 'Object-oriented representation of codes of practice'. *Procs. Australasian Conference on Expert Systems in Engineering, Architecture, and Construction*, Sydney Australia, pp209-221, December 1989.

HOSKING, J.G., MUGRIDGE, W.B., and HAMER, J. '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, 1991.

HOSKING, J.G., MUGRIDGE, W.B., and BLACKMORE, S., Objects and constraints: a constraint based approach to plan drawing, in Mingins, C. and Meyer, B. *Technology of object-oriented languages and systems TOOLS 15,* Prentice Hall, Sydney, pp 9-19, 1994.

MUGRIDGE, W B, HOSKING J G 'Towards a lazy, evolutionary common building model', *Building and Environment* 30 (1) pp99-114, 1995.

MUGRIDGE, W B, GRUNDY, J., AMOR, R, HOSKING, J., "Snart94 User and reference manual", Report, Department of Computer Science, University of Auckland, 1995.

STEP, *Standard for the exchange of product model data*, proposed as ISO 10303, TC184/SC4 (1992).

Appendix A: ThermalDesigner Code Listings

A.1 ThermalDesigner classes

```
/*
   File:   ThermalDesigner.sn
   Purpose:  Prototype of the ALF thermal design system.
   First Developed in Kea by:
        Robert Amor
        Department of Computer Science,University of Auckland
   Date:   April 1991

   Then redeveloped in Snart by:
        Rick Mugridge
        Department of Computer Science, University of Auckland
   Date:   July 1995
   Copyright  (C) 1991,1995 Building Research Association of New Zealand

*/

%____________ Climate _____________

class(td_climate,
  features(  % All Defined from IDM
   exposure_class: [sheltered, medium_sheltered, medium_exposed,
exposed],
   infiltration_zone: [a, b, c, d],
   degree_days: int,
 create % Set up the local dialog for Climate
)).

td_climate::create :-
  set_prop(td,climate,self),
  climate_dialog.

%____________ Space _____________

class(td_space,
  features(
   floor: list(td_floor),  % Defined from IDM
   roof: list(td_roof),    % Defined from IDM
   walls: list(td_wall)),  % Defined from IDM
  constraints(
   space_volume: float := sum(collect(f in floor, f@area)) *
max(collect(w in walls, w@height)),
   walls_heat_loss: float   := sum(collect(w in walls, w@heat_loss)),
   floor_heat_loss: float   := sum(collect(f in floor, f@heat_loss)),
   roof_heat_loss: float   := sum(collect(r in roof, r@heat_loss)),
   windows_heat_loss: float := sum(collect(w in walls,
w@solar_loss_total)),
   windows_heat_gain: float := sum(collect(w in walls,
w@solar_gain_total)),
   solar_glazing_area: float
      := sum(collect(w in walls, w@window_area_total where (w@orientation
in [n, ne, nw]))),
   joint_length: float
      := sum(collect(w in walls, w@joint_length)) + sum(collect(f in
floor, f@joint_length))
   + sum(collect(r in roof, r@joint_length))
)).

%____________ Floor _____________

class(td_floor(degree_days: int),
   features( % All defined from IDM, except r_value
      construction_type: classifier(td_suspended_floor,
td_concrete_floor),
      length: float,
      width: float,
```

```
      floor_covering_r_value: float,
      r_value: float), % Calculated in subclasses
   constraints(
      area: float := length * width,
      perimeter_length: float := 2.0 * (length + width),
      area_perimeter_ratio: float := area / perimeter_length,
      alf: float := 3.5343 + 0.045608 * degree_days - 1.1001e-5 *
degree_days * degree_days,
      heat_loss: float := area * alf / r_value,
      joint_length: float := length + width
)).

%____________ Suspended Floor _____________

class(td_suspended_floor, inherits(td_floor),
   features(  % All from IDM
   subfloor_protection: [a,b,c],
   foundation_height: float,
   floor_insulation_r_value: float),
   constraints(
   perimeter_wall_area: float := area * foundation_height,
   area_perimeter_wall_area_ratio: float := area / perimeter_wall_area,
   r_value :=
subfloorR(subfloor_protection,area_perimeter_wall_area_ratio) +
floor_insulation_r_value + floor_covering_r_value
)).

subfloorR(a,_,0.01).
subfloorR(b,Ratio,V) :- V is 0.05714 * Ratio.
subfloorR(_,Ratio,V) :- V is 0.1 * Ratio.

%____________ Concrete Floor _____________

class(td_concrete_floor, inherits(td_floor),
   features( % Defined from IDM
   insulation_depth: float),
   constraints(
   slab_r_value: float := 0.4 + 0.2 * insulation_depth +
area_perimeter_ratio *
          (0.464 + 0.052 * insulation_depth),
        % Formula derived from the graphs given on ALF p16.
        % Extrapolation beyond insulation depth = 1m is unwise
   r_value := slab_r_value + floor_covering_r_value
)).

%____________ Roof _____________

class(td_roof(degree_days: int),
   features( % Defined from IDM
      length: float,
      width: float,
      r_value: float,
      colour: [light, dark]),
   constraints(
      area: float := length * width,
      alf: float := roof_alfR(colour,degree_days),
      heat_loss : float := area * alf / r_value,
      joint_length : float := length + width
)).

roof_alfR(dark,Days,V) :-
  V is -9.7212 + 0.065364 * Days - 1.9697e-5 * Days * Days.
roof_alfR(light,Days,V) :-
  V is -2.8380 + 0.063549 * Days - 1.9522e-5 * Days * Days.

%____________ Wall _____________

class(td_wall(degree_days: int),
   features( % All from IDM
      width: float,
      height: float,
```

```
      orientation : [n, ne, nw, w, e, sw, se, s],
      colour: [dark, light],
      windows: list(td_window),
      r_value: float),
   constraints(
      alf: float := alfR(colour, orientation, degree_days),
      heat_loss: float := area * alf / r_value,
      window_area_total : float := sum(collect(j in windows, j@area)),
      area: float := width * height - window_area_total,
      solar_gain_total: float := sum(collect(w in windows, w@heat_gain)),
      solar_loss_total: float := sum(collect(w in windows, w@heat_loss)),
      % a hack for joint length, presume that it is half the joint length
of a wall
      % as each wall joint touches the wall joint of another wall or
ceiling or floor.
      joint_length: float := width + height
)).

alfR(light, Or, Days, V) :-
   member(Or,[n, ne, nw]),
   V is -5.9841 + 0.060934 * Days - 1.8473e-5 * Days * Days.
alfR(light, Or, Days, V) :-
   member(Or,[e, w]),
   V is -3.9841 + 0.060934 * Days - 1.8473e-5 * Days * Days.
alfR(light, Or, Days, V) :-
   member(Or,[s, se, sw]),
   V is -2.6186 + 0.060696 * Days - 1.8240e-5 * Days * Days.
alfR(dark, n, Days, V) :-
   V is -16.875 + 0.065122 * Days - 1.9639e-5 * Days * Days.
alfR(dark, Or, Days, V) :-
   member(Or,[ne, nw]),
   V is -13.005 + 0.063322 * Days - 1.8765e-5 * Days * Days.
alfR(dark, Or, Days, V) :-
   member(Or,[e, w]),
   V is -11.529 + 0.067010 * Days - 2.0688e-5 * Days * Days.
alfR(dark, Or, Days, V) :-
   member(Or,[s, se, sw]),
   V is -7.0047 + 0.063322 * Days - 1.8765e-5 * Days * Days.

%___________ Window ___________

class(td_window(facing: orientation_type, degree_days: int),
   features(
      r_value: float,
      shading_coef: float,
      height: float,
      width: float),
   constraints(
      area: float := height * width,
      alf: float := 12.0 + 0.02143 * degree_days,
      heat_loss: float := area * alf / r_value,
      heat_gain: float  := area * shading_coef * agfR(facing,degree_days)
)).

agfR(n,Days,V) :- V is 429.41 - 0.241220 * Days + 4.1833e-5 * Days *
Days.
agfR(Or,Days,V) :- member(Or,[ne,nw]), V is 354.59 - 0.234590 * Days +
5.8192e-5 * Days * Days.
agfR(Or,Days,V) :- member(Or,[e,w]), V is 221.18 - 0.158000 * Days +
4.1833e-5 * Days * Days.
agfR(Or,Days,V) :- member(Or,[se,sw,s]), V is 99.468 - 0.071612 * Days +
2.4488e-5 * Days * Days.


%___________ Building ___________

class(td_building,
   features(
        building_name: text,    % Defined from IDM
        num_of_occupants: int,    % Defined from IDM
        effective_thermal_mass: float, % Defined from IDM
```

```
        spaces: list(td_space),   % Defined from IDM
        climate: td_climate,   % Defined from IDM
        create),     % Used to set up local dialogs
   constraints(
   floor_area_total: float := sum(collect(s in spaces, sum(collect(f in
s@floor, f@area)))),
   building_volume_total: float := sum(collect(r in spaces,
r@space_volume)),
   joint_length: float := sum(collect(s in spaces, s@joint_length)),
   joint_volume_ratio: float := joint_length / building_volume_total,
   air_leak_temp: float := air_leak(climate@infiltration_zone) *
joint_volume_ratio,
   air_leakage_rate: float := air_leak_rate(climate@exposure_class) *
air_leak_temp,
   air_leakage_alf: float := -3.3678 + 0.022838 * climate@degree_days
          - 6.7557e-6 * climate@degree_days * climate@degree_days,
   air_heat_loss: float := building_volume_total * air_leakage_alf *
air_leakage_rate,
   total_seasonal_heat_losses: float
      := air_heat_loss
      + sum(collect(s in spaces, s@floor_heat_loss + s@roof_heat_loss
             + s@walls_heat_loss + s@windows_heat_loss)),
   internal_heat_gain: float := 900.0 + 150.0 * num_of_occupants,
   total_seasonal_gain: float
      := internal_heat_gain + sum(collect(s in spaces,
s@windows_heat_gain)),
   gain_loss_ratio: float := total_seasonal_gain /
total_seasonal_heat_losses,
   glr: float := gain_loss_ratio,
       % Equations for the graphs in ALF: use the closest graph for
finding
      % a point instead of a trickier interpolation to the correct value
-
      % things are so vague anyway that this slight inaccuracy won't
matter
      % a damn
   useful_fraction: float := useful_fract(effective_thermal_mass,glr),
   useful_heat_gain: float := total_seasonal_gain * useful_fraction,
   net_heating_EU: float := total_seasonal_heat_losses -
useful_heat_gain,
   solar_glazing_area: float := sum(collect(s in spaces,
s@solar_glazing_area)),
   solar_glazing_floor_area_ratio: float := solar_glazing_area /
floor_area_total,
   est_max_winter_indoor_temp: float
      := 20.0 + 142.857 * (1.0 - 0.38333 * effective_thermal_mass)
         * solar_glazing_floor_area_ratio,

   bpi: float := net_heating_EU / (climate@degree_days *
floor_area_total),
   window_N_gain: float
      := sum(collect(s in spaces, sum(collect(w in s@walls,
w@solar_gain_total where
         (w@orientation = n)))),
   window_NE_NW_gain: float
      := sum(collect(s in spaces, sum(collect(w in s@walls,
w@solar_gain_total where
         (w@orientation in [ne, nw])))),
   window_E_W_gain: float
      := sum(collect(s in spaces, sum(collect(w in s@walls,
w@solar_gain_total where
         (w@orientation in [e, w])))),
   window_SE_SW_S_gain: float
      := sum(collect(s in spaces, sum(collect(w in s@walls,
w@solar_gain_total where
         (w@orientation in [se, sw, s])))),
   slab_heat_loss: float
      := sum(collect(s in spaces, sum(collect(f in s@floor, f@heat_loss
where
           (f@construction_type = td_suspended_floor)))),
   suspended_heat_loss: float
```

```
      := sum(collect(s in spaces, sum(collect(f in s@floor, f@heat_loss
where
          (f@construction_type = td_concrete_floor)))),
   wall_N_heat_loss: float
      := sum(collect(s in spaces, sum(collect(w in s@walls, w@heat_loss
where
        (w@orientation = n)))),
   wall_NE_NW_heat_loss: float
      := sum(collect(s in spaces, sum(collect(w in s@walls, w@heat_loss
where
        (w@orientation in [ne, nw])))),
   wall_E_W_heat_loss: float
      := sum(collect(s in spaces, sum(collect(w in s@walls, w@heat_loss
where
        (w@orientation in [e, w])))),
   wall_SE_SW_S_heat_loss: float
      := sum(collect(s in spaces, sum(collect(w in s@walls, w@heat_loss
where
        (w@orientation in [se, sw, s])))),
   roof_heat_loss: float
      := sum(collect(s in spaces, s@roof_heat_loss)),
   window_heat_loss : float
      := sum(collect(s in spaces, sum(collect(w in s@walls,
w@solar_loss_total)))),
  demons(
    cw_set_item('Thermal Designer',bpiText,number_atom(bpi))
)).

air_leak(a,15.00000).  air_leak(b,13.33333).
air_leak(c,11.81818).  air_leak(d,10.43478).

air_leak_rate(exposed,0.09091).
air_leak_rate(medium_exposed,0.07857).
air_leak_rate(medium_sheltered,0.06666).
air_leak_rate(sheltered,0.05714).

useful_fract(ETM,GLR,V) :- ETM =< 0.15, uf(1.07870, 0.42465, 0.0518650,
0.0043706, GLR, V).
useful_fract(ETM,GLR,V) :- ETM =< 0.45, uf(1.10220, 0.38544, 0.0075756,
0.0147310, GLR, V).
useful_fract(ETM,GLR,V) :- ETM =< 0.80, uf(1.11800, 0.36126,-0.0059524,
0.0113678, GLR, V).
useful_fract(ETM,GLR,V) :- ETM =< 1.25, uf(1.06130, 0.12136,-0.2161800,
0.0673400, GLR, V).
useful_fract(ETM,GLR,V) :- ETM =< 1.75, uf(0.97333, 0.23106,-0.5170500,
0.1420500, GLR, V).
useful_fract(ETM,GLR,V) :- ETM  > 1.75, uf(0.95071,-0.33541,-0.5955100,
0.1578300, GLR, V).

uf(A,B,C,D,GLR, V) :- V is A - B * GLR + C * GLR * GLR + D * GLR * GLR *
GLR.

td_building::create :-
  set_prop(td,building,self),
  td_dialog.  % Set up the top-level TD dialog
```

A.2 ThermalDesigner dialogs

```
/*
   These are the dialogs associated with TD and controlled by it.  Some
of the data that they supply into td_building and td_climate objects
could be mapped back into the IDM, but we don't have any immediate need
to do that.

The TD and materials dialogs all work by treating the default values as
initial values.  A change to any dialog input is effective immediately.
Hence the OK buttons are mostly superfluous  (on some windows, the OK
button hides the window; on others, it does nothing).
*/

%------- TD DIALOG ---------
td_dialog:-
   Win = 'Thermal Designer',
   td_note(Win),
   cw_create( Win, 50, 20, 200, 330, [ dialog, visible] ),
   cw_add_item( Win, _, text( 10, 10, 20, 120, 'Building name:' )),
   cw_add_item( Win, nameEdit, edit( 10, 120, 20, 200, '*BUILDING NAME*'
)),
   cw_add_item( Win, _, text( 40, 10, 20, 180, 'Number of occupants:' )),
   cw_add_item( Win, numEdit, edit( 40, 215, 20, 50, '4' )),
   cw_add_item( Win, etmlPopup, popup( 70,10, 20, 290, 'Effective Thermal
Mass level:', [spp, sppb, smpp, smp, smc, s23pp, s23ppb, s12pp, s13pp,
s13ppb, tpp, tppb, cpp, cppb, cmpp, cmp, cmc, other], tppb )),  % Later
disable following unless = 'other'
   cw_add_item( Win, _, text( 100, 10, 20, 200, 'Effective Thermal Mass:'
)),
   cw_add_item( Win, etmEdit, edit( 100, 215, 20, 40, '0.5' )),

   cw_add_item( Win, climateButton, button( 130, 10, 20, 110, 'Specify
Climate' ) ),
   cw_add_item( Win, heatGainsButton, button( 130, 135, 20, 80, 'Heat
gains' ) ),
   cw_add_item( Win, heatLossesButton, button( 130, 230, 20, 90, 'Heat
losses' ) ),
   cw_add_item( Win, _, text( 160, 10, 20, 40, 'BPI:' )),
   cw_add_item( Win, bpiText, text( 160, 60, 20, 40, '?' )),

   cw_add_item( Win, okButton, boldbutton( 160, 255, 28, 68, 'OK' ) ),
   cw_set_program( Win, click, tdClicker ),
   (tdClicker(Win,okButton); true).

tdClicker(Win,nameEdit) :-
  cw_get_item( Win, nameEdit, Name ),
  get_prop(td,building,Building),
  Building@building_name := Name.
tdClicker(Win,numEdit) :-
  cw_get_item( Win, numEdit, OccupantsAtom ),
  get_prop(td,building,Building),
  number_atom(Occupants,OccupantsAtom),
  Building@num_of_occupants := Occupants.
tdClicker(Win,etmlPopup) :-
  cw_get_item( Win, etmlPopup, EffectiveThermalMassLevel ),
  eff_thermal_mass(EffectiveThermalMassLevel,EffectiveThermalMass),
  number_atom(EffectiveThermalMass,EffectiveThermalMassAtom),
  cw_set_item( Win, etmEdit, EffectiveThermalMassAtom ),
  get_prop(td,building,Building),
  Building@effective_thermal_mass := EffectiveThermalMass.
tdClicker(Win,etmEdit) :-
  cw_get_item( Win, etmEdit, EffectiveThermalMassAtom ),
  number_atom(EffectiveThermalMass,EffectiveThermalMassAtom),
  get_prop(td,building,Building),
  Building@effective_thermal_mass := EffectiveThermalMass.
tdClicker(_,climateButton) :-
  climate_dialog.
tdClicker(_,heatGainsButton) :-
  heat_gains_dialog.
tdClicker(_,heatLossesButton) :-
```

```
  heat_losses_dialog.
tdClicker(Win,okButton) :-
  tdClicker(Win,nameEdit),
  tdClicker(Win,numEdit),
  tdClicker(Win,etmEdit).

eff_thermal_mass(spp,1.0).   eff_thermal_mass(s23ppb,1.0).
eff_thermal_mass(cmpp,1.0).  eff_thermal_mass(sppb,1.2).
eff_thermal_mass(smpp,1.5).  eff_thermal_mass(cmp,1.5).
eff_thermal_mass(smp,1.9).   eff_thermal_mass(cmc,2.0).
eff_thermal_mass(smc,2.2).   eff_thermal_mass(s23pp,0.8).
eff_thermal_mass(s12pp,0.7). eff_thermal_mass(s13ppb,0.7).
eff_thermal_mass(s13pp,0.5). eff_thermal_mass(tppb,0.5).
eff_thermal_mass(tpp,0.2).   eff_thermal_mass(cpp,0.0).
eff_thermal_mass(cppb,0.4).

%_________ CLIMATE DIALOG __________

climate_dialog:-
   Win = 'Climate',
   is_win(Win, _),
   wfront(Win).
climate_dialog:-
   Win = 'Climate',
   td_note(Win),
   cw_create( Win, 160, 120, 195, 330, [ dialog, visible] ),
   cw_add_item( Win, wherePopup, popup( 10,10, 20, 220, 'Location:',
['Auckland',
         'Hamilton', 'Napier', 'New_Plymouth', 'Wellington',
'Christchurch',
         'Dunedin', 'Invercargill', other], 'Auckland' )),
   cw_add_item( Win, windPopup, popup( 40,10, 20, 310, 'Wind Exposure
Class:',
         [sheltered, medium_sheltered, medium_exposed,exposed],
medium_exposed )),
   cw_add_item( Win, _, text( 75, 10, 20, 50, 'Town:' )),
   cw_add_item( Win, townEdit, edit( 75, 70, 20, 250, 'Auckland' )),
   cw_add_item( Win, zonePopup, popup( 100,10, 20, 190, 'Infiltration
Zone:      ',
                [a,b,c,d], c )),
   cw_add_item( Win, _, text( 125, 10, 20, 140, 'Degree Day number:' )),
   cw_add_item( Win, degreeDayEdit, edit( 125, 158, 20, 50, '369' )),
   cw_add_item( Win, _, box( 70, 5, 78, 320 ) ),
   cw_add_item( Win, okButton, boldbutton( 160, 255, 28, 68, 'OK' ) ),
   cw_set_program( Win, click, climateClicker ),
   (climateClicker(Win,okButton);true).

climateClicker( Win, wherePopup ):-
  cw_get_item( Win, wherePopup, other ),
  cw_set_item( Win, townEdit, 'Other' ).
climateClicker( Win, wherePopup ):-
  cw_get_item( Win, wherePopup, Where ),
  td_climate_table(Where,DegreeDays,Zone),
  number_atom(DegreeDays,DegreeDaysAtom),
  cw_set_item( Win, townEdit, Where ),
  cw_set_item( Win, zonePopup, Zone ),
  cw_set_item( Win, degreeDayEdit, DegreeDaysAtom ),
  get_prop(td,climate,Climate),
  Climate@infiltration_zone := Zone,
  Climate@degree_days := DegreeDays.
climateClicker(Win,windPopup) :-
  cw_get_item( Win, windPopup, Wind ),
  get_prop(td,climate,Climate),
  Climate@exposure_class := Wind.
climateClicker(Win,zonePopup) :-
  cw_get_item( Win, zonePopup, Zone ),
  get_prop(td,climate,Climate),
  Climate@infiltration_zone := Zone.
climateClicker(Win,degreeDayEdit) :-
  cw_get_item( Win, degreeDayEdit, DegreeDaysAtom ),
  number_atom(DegreeDays,DegreeDaysAtom),
```

```
  get_prop(td,climate,Climate),
  Climate@degree_days := DegreeDays.
climateClicker(Win,okButton) :-
  climateClicker(Win,windPopup),
  climateClicker(Win,zonePopup),
  climateClicker(Win,degreeDayEdit).

% Map town to degreeDayNo and infiltration zone:
td_climate_table('Auckland',369,c).  td_climate_table('Hamilton',685,d).
td_climate_table('Napier',612,d). td_climate_table('New_Plymouth',611,c).
td_climate_table('Wellington',705,a).
   td_climate_table('Christchurch',965,c).
td_climate_table('Dunedin',944,c).
   td_climate_table('Invercargill',1113,c).

%_________  HEAT LOSS ____________

heat_losses_dialog:-
   Win = 'Heat Losses',
   is_win(Win, _),
   wfront(Win).
heat_losses_dialog:-
   Win = 'Heat Losses',
   td_note(Win),
   cw_create( Win, 60, 370, 250, 200, [ dialog, visible] ),
   cw_add_item( Win, _, text( 10, 10, 20, 120, 'Slab Floor:' )),
   cw_add_item( Win, slabText, text( 10, 130, 20, 40, '?' )),
   cw_add_item( Win, _, text( 30, 10, 20, 120, 'Suspended Floor:' )),
   cw_add_item( Win, suspendedText, text( 30, 130, 20, 40, '?' )),

   cw_add_item( Win, _, text( 70, 10, 20, 120, 'Wall N:' )),
   cw_add_item( Win, nText, text( 70, 130, 20, 40, '?' )),
   cw_add_item( Win, _, text( 90, 10, 20, 120, 'Wall NE + NW:' )),
   cw_add_item( Win, nenwText, text( 90, 130, 20, 40, '?' )),
   cw_add_item( Win, _, text( 110, 10, 20, 120, 'Wall E + W:' )),
   cw_add_item( Win, ewText, text( 110, 130, 20, 40, '?' )),
   cw_add_item( Win, _, text( 130, 10, 20, 120, 'Wall SE + SW + S:' )),
   cw_add_item( Win, seswsText, text( 130, 130, 20, 40, '?' )),
   cw_add_item( Win, _, text( 160, 10, 20, 120, 'Roof:' )),
   cw_add_item( Win, roofText, text( 160, 130, 20, 40, '?' )),
   cw_add_item( Win, _, text( 190, 10, 20, 120, 'Window:' )),
   cw_add_item( Win, windowText, text( 190, 130, 20, 40, '?' )),

   cw_add_item( Win, okButton, boldbutton( 220, 130, 28, 68, 'OK' ) ),
   cw_set_program( Win, click, lossesClicker ),
   get_prop(td,building,Building),
   td_heat_loss@create(_,Building).

lossesClicker(Win,okButton) :-
  whide( Win ).

class(td_heat_loss(building:td_building),
  demons(
    cw_set_item( 'Heat Losses', slabText,
number_atom(building@slab_heat_loss)),
    cw_set_item( 'Heat Losses', suspendedText,
number_atom(building@suspended_heat_loss)),
    cw_set_item( 'Heat Losses', nText,
number_atom(building@wall_N_heat_loss)),
    cw_set_item( 'Heat Losses', nenwText,
number_atom(building@wall_NE_NW_heat_loss)),
    cw_set_item( 'Heat Losses', ewText,
number_atom(building@wall_E_W_heat_loss)),
    cw_set_item( 'Heat Losses', seswsText,
number_atom(building@wall_SE_SW_S_heat_loss)),
    cw_set_item( 'Heat Losses', roofText,
number_atom(building@roof_heat_loss)),
    cw_set_item( 'Heat Losses', windowText,
number_atom(building@window_heat_loss))
)).
```

```
%__________  HEAT GAINS _____________

heat_gains_dialog:-
   Win = 'Window Heat Gains',
   is_win(Win, _),
   wfront(Win).
heat_gains_dialog:-
   Win = 'Window Heat Gains',
   td_note(Win),
   TitleSize = 80,  ValueOffset is TitleSize + 20,
   cw_create( Win, 270, 80, 130, 150, [ dialog, visible] ),
   cw_add_item( Win, _, text( 10, 10, 20, TitleSize, 'N:' )),
   cw_add_item( Win, nText, text( 10, ValueOffset, 20, 40, '?' )),
   cw_add_item( Win, _, text( 30, 10, 20, TitleSize, 'NE + NW:' )),
   cw_add_item( Win, nenwText, text( 30, ValueOffset, 20, 40, '?' )),
   cw_add_item( Win, _, text( 50, 10, 20, TitleSize, 'E + W:' )),
   cw_add_item( Win, ewText, text( 50, ValueOffset, 20, 40, '?' )),
   cw_add_item( Win, _, text( 70, 10, 20, TitleSize, 'SE + SW + S:' )),
   cw_add_item( Win, seswsText, text( 70, ValueOffset, 20, 40, '?' )),

   cw_add_item( Win, okButton, boldbutton( 100, 40, 28, 68, 'OK' ) ),
   cw_set_program( Win, click, lossesClicker ),
   get_prop(td,building,Building),
   td_heat_gain@create(_,Building).

class(td_heat_gain(building:td_building),
  demons(
    cw_set_item( 'Window Heat Gains', nText,
number_atom(building@window_N_gain)),
    cw_set_item( 'Window Heat Gains', nenwText,
number_atom(building@window_NE_NW_gain)),
    cw_set_item( 'Window Heat Gains', ewText,
number_atom(building@window_E_W_gain)),
    cw_set_item( 'Window Heat Gains', seswsText,
number_atom(building@window_SE_SW_S_gain))
)).

td_note(Win) :-
  (recall(td_windows,Windows) ; Windows = []),
  remember(td_windows,[Win|Windows]).

td_kill_dialogs :-
  recall(td_windows,Windows),
  forall(member(W,Windows),wkill(W)),
  forget(td_windows).
```

Appendix B: VML Mapping for IDM <-> ThermalDesigner

```
inter_view(idm, integrated, thermaldesigner, read_write, complete).

inter_class([idm_building], [td_building],
   equivalences(
      name = building_name,
      num_of_occupants = num_of_occupants,
      effective_thermal_mass = effective_thermal_mass,
      spaces = spaces,
      environment = climate
   ),
   initialisers(
      td_building.building_name = '*BUILDING NAME*',
      td_building.num_of_occupants = 4,
      td_building.effective_thermal_mass = 0.5
   )
).

inter_class([idm_environment], [td_climate],
   equivalences(
      exposure_class = exposure_class,
      infiltration_zone = infiltration_zone,
      degree_days = degree_days
   ),
   initialisers(
      td_climate.exposure_class = 'medium_exposed',
      td_climate.infiltration_zone = 'c',
      td_climate.degree_days = 369
   )
).

inter_class([group(idm_space_face), group(idm_roof)], [td_space],
   invariants(
      idm_space_face.location = 'ext'
   ),
   equivalences(
      bijection(idm_space_face[].type_of_face = 'wall', walls[]),
      bijection(idm_space_face[].type_of_face = 'floor', floor[]),
      idm_roof = roof
   )
).

inter_class([idm_space_face, idm_material_face, idm_building], [td_wall],
   invariants(
      idm_space_face.type_of_face = 'wall',
      idm_space_face.location = 'ext',
      idm_material_face.type_of_face = 'wall',
      plane_equivalence(idm_space_face.plane, idm_material_face.plane),
      point_equivalence(idm_space_face.min, idm_material_face.min),
      point_equivalence(idm_space_face.max, idm_material_face.max)
   ),
   equivalences(
      idm_space_face.max=>x - idm_space_face.min=>x = width,
      idm_space_face.max=>y - idm_space_face.min=>y = height,
      idm_space_face.orientation = orientation,
      bijection(idm_space_face.openings[]@class(idm_space_face),
windows[]),
      idm_material_face.material=>colour = colour,
      idm_material_face.material=>r_value = r_value
   ),
   initialisers(
      td_wall.colour = 'light',
      td_wall.r_value = 1.7,
      td_wall@create(idm_building.environment=>degree_days)
   )
).
```

```
inter_class([idm_space_face, idm_material_face, idm_building],
[td_window],
   invariants(
      idm_space_face.type_of_face = 'opening',
      idm_space_face.location = 'ext',
      idm_material_face.type_of_face = 'opening',
      plane_equivalence(idm_space_face.plane, idm_material_face.plane),
      point_equivalence(idm_space_face.min, idm_material_face.min),
      point_equivalence(idm_space_face.max, idm_material_face.max)
   ),
   equivalences(
      idm_space_face.max=>x - idm_space_face.min=>x = width,
      idm_space_face.max=>y - idm_space_face.min=>y = height,
      idm_material_face.material=>shading_coefficient = shading_coef,
      idm_material_face.material=>r_value = r_value
   ),
   initialisers(
      td_window.shading_coef = 1.0,
      td_window.r_value = 0.23,
      td_window@create(idm_space_face.orientation,
idm_building.environment=>degree_days)
   )
).

inter_class([idm_roof, idm_building], [td_roof],
   equivalences(
      idm_roof.max=>x - idm_roof.min=>x = width,
      idm_roof.max=>y - idm_roof.min=>y = length,
      idm_roof.material=>colour = colour,
      idm_roof.material=>r_value = r_value
   ),
   initialisers(
      td_roof.colour = 'light',
      td_roof.r_value = 3.6,
      td_roof@create(idm_building.environment=>degree_days)
   )
).

inter_class([idm_space_face, idm_material_face, idm_building],
[td_suspended_floor],
   invariants(
      idm_space_face.type_of_face = 'floor',
      idm_space_face.location = 'ext',
      idm_material_face.type_of_face = 'floor',
      idm_material_face.material@class('idm_suspended_floor'),
      plane_equivalence(idm_space_face.plane, idm_material_face.plane),
      point_equivalence(idm_space_face.min, idm_material_face.min),
      point_equivalence(idm_space_face.max, idm_material_face.max)
   ),
   equivalences(
      idm_space_face.max=>x - idm_space_face.min=>x = width,
      idm_space_face.max=>y - idm_space_face.min=>y = length,
      idm_material_face.material=>floor_covering_r_value =
floor_covering_r_value,
      idm_material_face.material=>subfloor_protection =
subfloor_protection,
      idm_material_face.material=>foundation_height = foundation_height,
      idm_material_face.material=>floor_insulation_r_value =
floor_insulation_r_value
   ),
   initialisers(
      td_suspended_floor.floor_covering_r_value = 0.4,
      td_suspended_floor.subfloor_protection = 'c',
      td_suspended_floor.foundation_height = 1,
      td_suspended_floor.floor_insulation_r_value = 0.3,
      td_suspended_floor@create(idm_building.environment=>degree_days)
   )
).

inter_class([idm_space_face, idm_material_face, idm_building],
[td_concrete_floor],
```

```
   invariants(
      idm_space_face.type_of_face = 'floor',
      idm_space_face.location = 'ext',
      idm_material_face.type_of_face = 'floor',
      idm_material_face.material@class('idm_slab_floor'),
      plane_equivalence(idm_space_face.plane, idm_material_face.plane),
      point_equivalence(idm_space_face.min, idm_material_face.min),
      point_equivalence(idm_space_face.max, idm_material_face.max)
   ),
   equivalences(
      idm_space_face.max=>x - idm_space_face.min=>x = width,
      idm_space_face.max=>y - idm_space_face.min=>y = length,
      idm_material_face.material=>floor_covering_r_value =
floor_covering_r_value,
      idm_material_face.material=>insulation_depth = insulation_depth
   ),
   initialisers(
      td_concrete_floor.floor_covering_r_value = 0.4,
      td_concrete_floor@create(idm_building.environment=>degree_days)
   )
).
```