Integration of Existing Programs Using Frames

Žiga Turk

University in Ljubljana, Department of Civil Engineering, Institute of Structural and Earthquake Engineering, P.O. Box 579, 61001 Ljubljana, Slovenia (ex Yugoslavia); fax: 38-61-268-572, e-mail: ziga.turk@uni-lj.ac.mail.yu

Abstract

A prototype for computer integrated design/analysis environment is being developed. Due to the nature and size of our institution, we have decided for compatibility with existing and third party products as well for future developments. Frames are used in Minsky's sense to insulate knowledge and semantics of the tools being integrated. Frames are used again in a more traditional sense insulating components physically. Standards like STEP or AIS were not applied explicitly, but principles behind those standards are reflected in our solution. In the paper an architecture of shallow integration of the tools for integrated structural design will be explained in greater detail. Some of the solutions will be suggested from the blending of the OO and AI techniques.

1. INTRODUCTION

In the last decade the activities of our institute involved introduction of computer based tools, particularly to the phase of computer integrated analysis and later design of engineering details of civil engineering structures (like the dimensioning of the load bearing structure and reinforcement design). What we have is a fairly typical picture of the islands of automation /FENS91/ - the usage of discrete and stand alone tools in a largely manual design environment. The integrator of all these tools is the civil engineer. He performs three types of tasks:

- *Routine* - translating output of one program into input to the other, translating information from and to paper. The AEC industry is still exchanging information on paper /HOWA90/.
- *Laborious* - Computer based tools often produce large volumes of results with small useful information content and an engineer is analysing them.
- *Responsible* - the engineer may be legally or professionally responsible for the decisions he is taking during the design process.

The main task of civil engineering software development is to eliminate routine work, ease the analysis and help with the decisions. The way to do it is to enable computer integrated construction expected in the next century /BJOR89/.

The primary sources of problems are the prototype nature of buildings, their complexity, large number of components and participants and the interaction of multiple disciplines /SRIR90/. While looking forward to STEP/PDES, AIS, SDAIS ... and other technologies that seem to have a final goal of computer interpretable description of nearly every object that forms our civilized environment, we have to consider the installed base of in-house written programs that need to be integrated here and now. In this paper, an approach to integration will be shown, that is not (and was not designed to be compliant) with any of those standards, but was planned having in mind the principles and ideas behind those standards.

2. COMPUTERS AND DESIGN

In reference to the integration of computers into the design process we see three main stages of development:

- *Computer assisted/aided design* - computer based tools are used for some tasks or phases of the design process. Some of the routine tasks are performed by the computer.
- *Computer integrated design* - computers are used for the integration of software, hardware, engineers and processes involved in the design process. All routine and some other tasks are performed by computer.
- *Computer automated design* - asymptotically the computers are approaching the state, where they would be able to produce the entire design with no help (instructions) from the engineer. (Figure 1).

Due to the pragmatic and economic reasons the transition from one stage to the other is expected to be evolutionary, meaning that the existing solutions will be employed

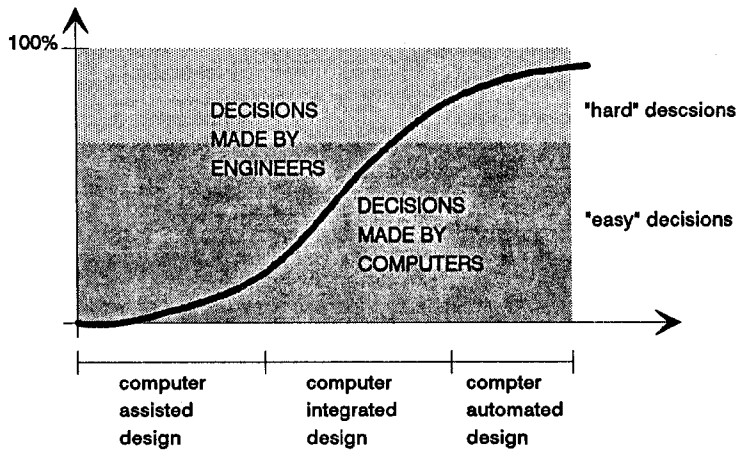

Figure 1. Stages of usage of computers during the design process. Most of practical applications are in the first level, while a lot of efforts of various researchers is centred on porting or writing applications for the second level. Elements enabling some computer automated design are expected by the end of the century.

whenever feasible. The possibility of turning over some portions of control and decisions to the computer must be taken into account during the development of the computer integrated environments as well. The possible integration of software must be considered during the development of the new stand alone tools and during the adaptation of older ones.

2.1 Integrated design

We can distinguish two main approaches to implementing a computer integrated design environment:

- *Pipeline architecture* - various tools are connected through some interfaces with each other. Although not very popular for academic research, the approach is giving efficient short term results, particularly when interfacing a smaller number of programs using similar or same abstract models.
- *Centralized architecture* - we define a common concept that serves as an integration instrument. On the physical level, this can be a central database, an expert system shell, an operating system tool ... or a combination. On the conceptual level, it can be a common data model type, compatible abstract models or building product models.

In fact, there is no real conflict between the two. The first can be effectively employed when connecting programs working on the same abstract model, pre- and post-processors and the like. An all encompassing integration is inconceivable without some sort of central concept, although it is possible, with considerable programming efforts, to provide the user with an illusion of an "integrated" environment by supplying a nice front end for the mess behind it.

2.2 Model of computer integrated design environment

In Figure 2 there is a model of a computer integrated design environment that has been divided into five sub-models.

The relation between the models is clear from their function. During the *design*

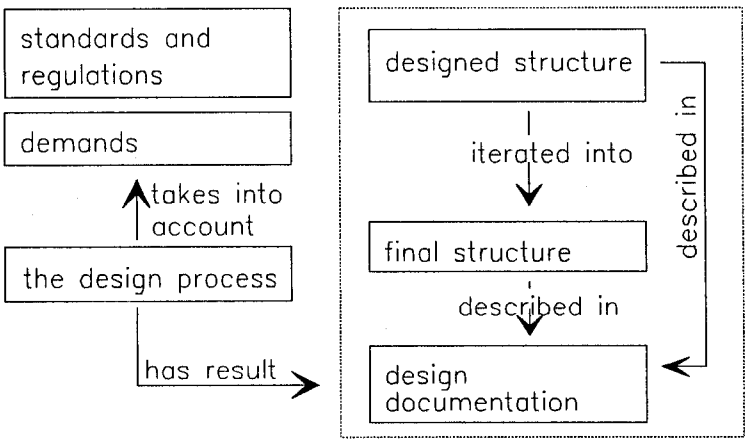

Figure 2. Semantic network of models participating in computer integrated design.

process, various *alternative designs* and calculations are made in order to be able to finally produce and refine all the necessary *plans and documentation* for the *designed structure.*

- *Standards, regulations and demands* define the boundary conditions that the design should fulfil. Some are enforced by the state authorities, some by the investor of the building. Their semantic nature and syntax are often similar. The investor might want a building at least 12 storeys high, the state regulations may require all buildings lower than 10 storeys because of the unstable ground.
- *The design process* models the actions and participants of the design process. The actions like analysis, synthesis and evaluation ... are preformed by the participants (designers, programs, computers ...).
- *The designed structure* models the design of the building. It is an extension of the final structure and also contains information of various versions of the design, design decisions and other information regarding the design that was used in the design process. It is sometimes referred to as a "process model" /CORN91/.
- *The final structure* models the building as it will be built. It describes the final, chosen design or a "static model". Most building product models being developed describe this model.
- *The design documentation* models the interface to the human interpretation of the design. For some time to come, final designs will have to be communicated in a form that will be used without computers.

The five models are closely related. They are all based on the static (final design) model, being its extension or translation (process model, documentation models) or being implicitly referred to in standards, regulations and in the design process /TURK91a/.

2.3 Object orientation

The above models are very complex. In a description of a building several thousands of various data item types are used. In order to organize these vast amounts of data types (and even larger amounts of data items) it has been recognised that object oriented approach has many advantages /FENG90/, /POWE88/.

It has a built in generalisation/specialisation and characterisation mechanisms which are two of the three fundamental abstractions used in the product model definition space. The third abstraction is aggregation/decomposition. When used with instances (not with types) we tend to understand it as a more general topological abstraction giving data items a reference to spatial relations with other data items. Some relations can be described as *is_part_of*, other are better represented as *is_connected_to*, *is_enclosed_in*. Also, in systems where geometry is not of paramount importance, some other relation may take its role.

A possible inconvenience with object oriented programming is that an object is a collection of data *and* methods that manipulate that data. In an object, data and methods (procedures) are one. This is in contrast with the general idea of most knowledge based systems that are characterized by the separation of the data (knowledge base) from the

procedures (inference engine), and more, by the separation of the knowledge based part from the algorithmic part.

The main motives for this separation were:

- Existing engineering programs were very complex and adding additional knowledge seemed not feasible. Separated knowledge based modules lowered complexity. Object orientation provides other means to fight complexity.
- It would be easier to maintain, add and customize the knowledge, if it was separated from the rest of the software. It turned out that manipulating the knowledge base requires trained knowledge engineers and usually can not be done by end users.
- It would be possible to make plug compatible products that would enable to plug different knowledge modules into the same engine, making it possible, for instance, to apply different national codes to the same analysis program. The feasibility of this does not depend only on the architecture of knowledge based system, but mostly on compatibility of codes.

We feel that the two technologies, AI and OO should be merged and have developed a prototype showing OO hierarchy as a skeleton for class/object specific knowledge expressed both proceduraly and declaratively /TURK91b/.

3. FRAME ARCHITECTURE

The Smalltalk project introduced the Modell-View-Controller paradigm /GOLD83/. We like to see our integrated design environment in a similar way. It is shown in Figure 3.

Since we are building the environment from the ready made parts (existing, old programs) we are fond of a top (controllers) to bottom (model) development.

The *controllers* are the existing (and new) programs. Each has an implicit, but physically very well defined data model (at least the formats of input and output files). That model may bare poor resemblance to the designed structure, but it is a model of some of its features.

Some programs may share the same conceptual model or at least similar models (i.e. all geometric modellers could share a very general *view* of the buildings geometry). This is what we call a view. There are many views, in fact each application or group has its own and they are ordered using the specialisation criteria into a directed acyclic graph. It is easy to integrate controllers which are on the same DAG branch (series of connected arches) since the transformation of the views only requires generalisation. We use the pipeline architecture for integration on DAG arches.

General integration of controllers, however, requires more than that, because the views are still strictly separated into branches. The solution is, of course the *model*. It can be designed using conceptual modelling techniques or as a union of all the views. In the latter case, the model is not very general and open. Although not as extensive, the architecture of such a system is similar to the one that would use Application Interface Specification (AIS). The views are playing the role of the AIS as presented in /MAGL90/.

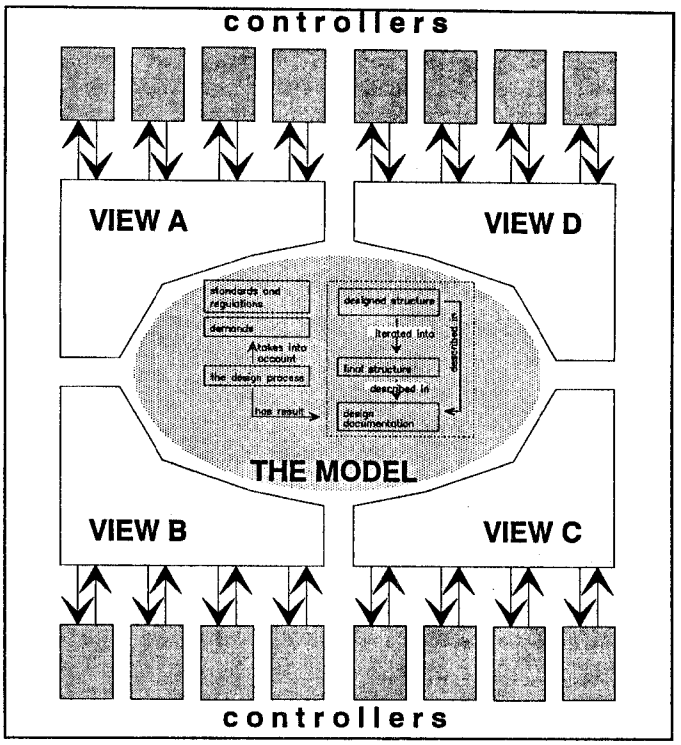

Figure 3. MVC paradigm applied to integrated design environment.

Also, analogy with the three layer ANSI/SPARC concept is apparent. All three solutions have a common characteristic, a middle layer, an interface between physical storage, (or syntax or database), and the users of the data stored there. In our approach, a view acts as that middle layer. The meaning of the view is given through the set of tools that are manipulating the view

3.1 Conceptual frames

The concepts described above are represented as objects classified as shown in Figure 4. Beside views and controllers there is also an abstraction for parts of design process (the process class).

A *process* describes one "run" of a program, its inputs and outputs. Its main components are lists of view and controller objects. The generic process class is shown in Figure 5 as an object or a frame. It has two "knowledge island" (KI) slots. One to manipulate input, the other to analyse output. The run method first starts the prologue method, which among other, extract data from the view, consults KI_validate input, then the program is run and, finally, the epilogue method is called to analyse the output (through KI_validate output) and the results are translated into the view compatible form. The translation is performed using methods in the class that describes the output file in question.

This is also the reason to have two types of views - direct ones on the top branch and the so called file views (Figure 4). All existing programs can only operate through file views which can later be translated into more abstract direct views and finally into the model.

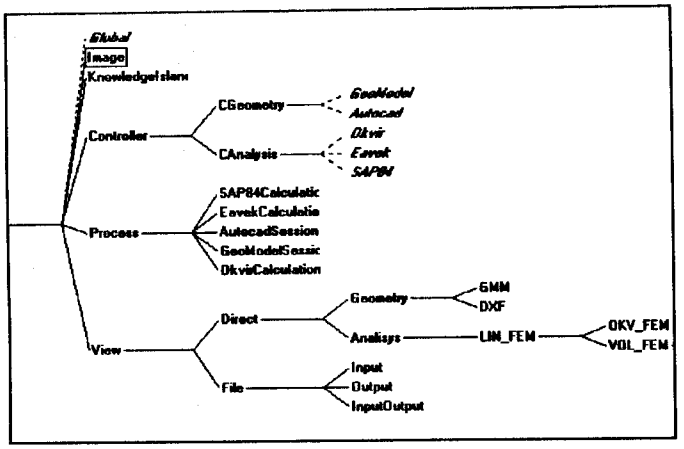

Figure 4. Simplified tree of objects that model existing programs in the computer integrated design environment. Views are structured hierarchically.

3.2 Physical frames

The term "frame", as used in this paper, has two meanings. Both describe programs, tools and their relation to some integrated environment. The Minsky's sense has already been demonstrated. Each process is described in a frame like structure with slots as in Figure 5

The other meaning of a frame is in the sense of physically insulating the program from the environment and the user form the details of that environment. It is borrowed from the EIS project /PAYN90/ and the CFI backplane architecture /WADE90/. It is a very practical approach to insulate users and tools from repository differences. Since there was quite some "insulation" already performed through the multi layered views on conceptual level, what is left is only to add consistent user and operating system interface.

We could say, that in the Minsky's sense, the frame is on the conceptual, semantical level, while in the traditional sense it is on the physical level. The physical schema is on figure 6.

It consists of software tools (controllers) connected to a common user interface on one end and to the view on the other. Hierarchic nature of views is also demonstrated. Underlying it all there should be the strongest integrating factor like product data

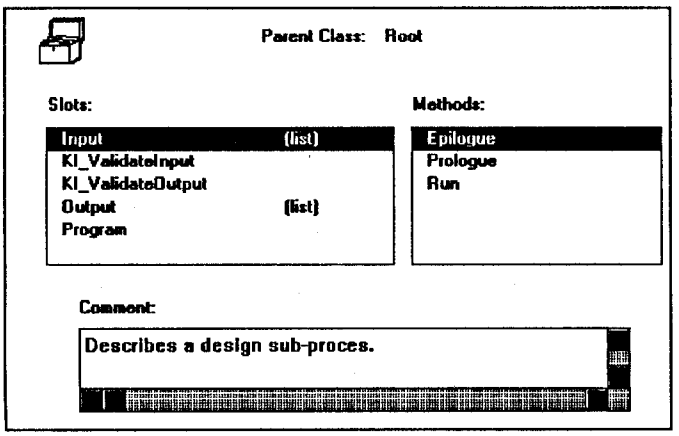

Figure 5. The structure of a generic process object.

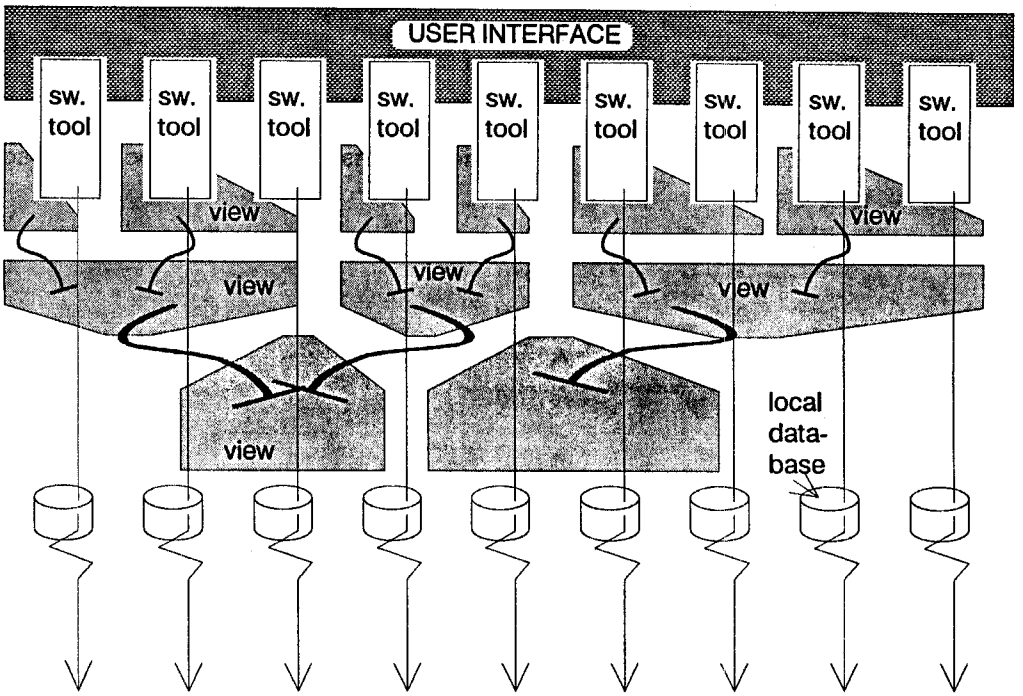

Figure 6. Physical frames. Each application (or software tool) is framed by common user interface on top, and by view on bottom and on the side. The views are loosely connected into hierarchic order.

database, but it is not implemented as such at present.

The environment is being developed under MS-DOS operating system (as are all our existing applications). The user interface works under MS Windows. The user sees the project as three hypertext stacks:

- a stack of software tools (available programs),
- a stack of processes,
- a stack of documents, that will be included in the design documentation.

The communication between the components is through files, pipes and filters (for older programs) and through the dynamic data exchange when using Windows compatible products.

4. CONCLUSIONS

The views are implemented as an object oriented expert shell interface over a relational database. The shell is also used as a data modelling tool and a simulation OO database. There is a one to one mapping between objects in an expert system shell and some of records stored in data base files. The data base is storing values of objects, expert system shell is storing most of the knowledge. Some procedural knowledge is coded in

specialized programs. The solution is temporary and will be replaced with an object data base.

Among other tools, most of the integrated environment shell is being built using the Kappa hybrid expert system shell and Toolbook hypertext authoring system. We found the shell very good for the modelling and the analysis of data, suitable for prototyping, but rather insufficient for implementing the real system. It lacks the data storage capabilities and the efficiency of object oriented databases (ODB) and provides very limited tools for the combination and cooperation or merging of several models and objects involved. We would find a simple inference engine a useful add-on to ODBs. Toolbook is easy to use, powerful, but slow in some operations. Both tools implement exotic programming language syntax.

In the Figure 4 there is no model branch. Without a model and a formalized data base our integration may not be considered very deep. It clearly integrates the views and the controllers, but the model does not (jet) exist physically. This is also demonstrated in Figure 6. Clearly, the level of integration is decreasing with depth. The implementation gives relatively simple and efficient temporary solution to the users needs and, as shown on Figure 6, the user has an illusion of integration. Further work is devoted to make a better foundation that will also enable to increase the functionality of the whole system.

5. REFERENCES

BJOR89 Bjork, B.C., Penttila,H., "A scenario for the development and implementation of a building product model standard", Adv. Eng. Software, Vol.11, No.4, 1989.

CORN91 Cornick, S.M., Leishman, D.A., Thomas, R.J., "Integrating building codes into design systems" , VTT workshop Computers and Building Standards, Espoo, Finland, 27.-29.5. 1991.

FENG90 Fenves, G.L., "Object oriented programming for engineering software development", Engineering with Computers 6, 1-15 (1990).

FENS91 Fenves, S.J., "Status, needs and opportunities in computer assisted design and analysis", Structural Engineering International, 2/91.

GOLD83 Goldberg, A., Robson, D., "Smalltalk-80: the language and its implementation", Addison Wesley 1983.

HOWA90 Howard, H.C., "Prototype integrated environment for AEC data", proceedings of the ASCE 6th Conference: Computing In Civil Engineering.

MAGL91 Magleby, S., Ranyak, P., Sanford, D., Jackson, D., Bean, R., "Towards a standardized application interface for product modellers: issues, developments and results", proceednings of the Design Productivity International Conf., Honolulu, HA, Feb. 3-9, 1991.

PAYN90 Payne, J.H., "EIS Framework Perspective", published in /PDES90/.

PDES90 CAM-I/PDES, "Proceedings of the Joint CAM-I / PDES, Inc. Workshop - Standardized interfaces in product data", St. Louis, Missouri, USA, October 4-5, 1990. .

POWE89 Powell, G.H., Bhateja, R., "Data base design for computer-integrated structural engineering", Engineering with Computers 4, 1988.

SRIR90 Sriram, D., Groleau, N., "Object oriented databases for cooperative design", proceedings of the ASCE 6th Conference: Computing In Civil Engineering (1990).

TURK91a Turk, Ž., "Building Model Standard as a Foundation for Computer Integrated Design", VTT workshop Computers and Building Standards, Espoo, Finland, 27.-29.5.1991.

TURK91b Turk, Ž., Duhovnik, J., "Using Knowledge Islands in an Object Oriented Framework for Integrated Structural Design", conference Artificial Intelligence CIVIL-COMP 91, Oxford, England, 3-5. September, 1991.

WADE90 Wade, A.E., "CFI", published in /PDES90/.