High-Level Specification of a Software Environment to Support the Early Phases in Building Design

Ulrich Flemming, Robert Woodbury

EDRC 48-31-92

High-Level Specification of a Software Environment to Support the Early Phases in Building Design

U. Flemming, R. Woodbury
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

October 1992

Abstract. This report develops a high-level functional specification for a software environment to support the early phases in building design (SEED). Particular emphasis is placed on support for the design of recurring building types. The environment is divided into modules, each of which addresses a specific design task. All modules consist of the same generic components. They are supported by a common database and interfaces with a common style. Among the data in the database are solution prototypes or cases that facilitate work with recurring building types. The first system prototype will comprise three modules supporting the following tasks: architectural programming, schematic layout design, and schematic configuration design.

This work has been supported by the Engineering Design Research Center, a NSF Engineering Research Center, and the Carnegie Mellon/ Building Industry Computer-Aided Design Consortium (CBCC).

Contents

1. Overall Goal

This report presents a high-level functional specification of a software environment that supports the early phases in the design of buildings. The overall system is called SEED.

SEED is intended to support, in particular, the design of ***recurring building types***, that is, building types dealt with frequently in a firm or institution. Such organizations - from housing manufacturers to government agencies - accumulate considerable experience with recurring building types. But capturing this experience and its reuse are supported only marginally by present CAD systems. Systems supporting work with recurring building types must reflect what is constrained and what is free in their design, should support the well-understood design tasks as effectively as possible, must provide means to capture and reuse accumulated design experience, and must be flexible enough for adaptations to a unique design context and individual designers.

The goal of the system is to support, in principle, the preliminary design of such buildings in all aspects that can benefit from computer support. This includes using the computer not only for analysis and evaluation, but also more actively for the generation of designs, a capability that remains underutilized in present CAD systems. The intent is to develop of a collection of ***generic capabilities that can be easily adapted to different building types.***

The present report describes the overall architecture and intended functionality of the first prototype from a high-level, user-oriented perspective as we envision them at the outset. It is the basis for an object-oriented software engineering process in which system specifications are developed through successive levels of abstractions.

These specifications are intended to serve several purposes. First and foremost, they are meant to implement an ***orderly software development process*** that assures compliance with the intended functionality of the system through all development stages, and logical consistency and compatibility between individual parts and modules. Furthermore, the specifications are intended to provide a means of ***communication*** between us and our sponsors to assure mutual agreement on the direction of our efforts; as a means to insure ***coordination*** between the efforts of the individual researchers involved in the project; and as ***documentation*** of our efforts.

The first prototype will be based on the experience accumulated at CMU with the LOOS/ABLOOS (Flemming et al. 1988a,b; Coyne and Flemming 1990, Coyne 1991) and GENESIS (Heisserman 1991, Heisserman and Woodbury 1992) systems. The building types addressed in the pilot version will be selected in consultation with our sponsors.

2. Overall Architecture

2.1 Phases and Modules

Our approach for the first version of the prototype starts with the assumption that the preliminary design process can be divided into ***phases***, each of which addresses a particular subtask of the overall design problem and leads to a particular set of related decisions. For example, the preliminary structural design phase may deal with the task of finding an appropriate structural system for an overall building configuration. There is no assumption that phases have to occur in a strict sequence. A partial ordering between them may, however, exist because the information needs of one phase may depend on decisions made in another phase.

We plan to develop for each phase an ***individual support module based on a shared logic and architecture***. This will allow us, on the one hand, to make local use of various pieces of existing and possibly heterogeneous software and to distribute the development efforts among individual modules. The shared logic and architecture assure, on the other hand, that the modules appear to the user as parts of a unified whole, which includes a common style for the interfaces. They also allow us to develop generic protocols for phase transitions; they make any sequence of phases logically extensible and should facilitate the "plugging in" and "plugging out" of individual module versions.

The results generated in any phase are stored in a ***database*** for re-use, either in the context of the same project or a different project, in which case they become ***prototypes*** or ***cases***.

2.2 Components

The architecture proposed for a module assumes that the task to be accomplished in any module can be divided into five generic subtasks. Each of these is supported by a specific module ***component***. Figures 2-1 and 2-2 show respectively data flow and control flow views between these components, the user, and the database.

The ***input component*** is the general read interface between a module and the database. This component makes, in principle, everything specified or generated by another module or by a previous invocation of the same module available to a module. This may involve translations between data formats, or the stripping away of information. For example, if a module dealing with two-dimensional layouts reads a three-dimensional configuration of components, information about the third dimension may have to be deleted. Specific problems caused by this will have to be worked out on a case-by-case basis. But we specify this possibility in order to stress that, in principle, no predetermined order of execution for the modules is assumed.

The ***problem specification component*** allows designers to specify or modify the task to be performed in the current module. For example, if the task of the current module is to develop a schematic floor plan generated in another module into a three-dimensional configuration of building elements, this floor plan must be made available to the current module (this is the function of the input module); but the module may need additional information about the desired roof shape, wall construction etc. This is done in the problem specification component.

The ***generator*** supports the generation of solutions to the problem specified for the current module. In each module, we intend to make a broad range of phase-specific support options available, from complete automation to interactive constructions that are completely under the designer's control.

The ***evaluation component*** evaluates solutions for compliance with the specifications. Iterations at this stage may occur, where the evaluation results influence further generator action.

The ***output component*** allows a designer to write any result generated or specified in a module into the database. Examples are a solution stored as a prototype for re-use in a different project, or an intermediate solution for the current project that is to be elaborated in another module.

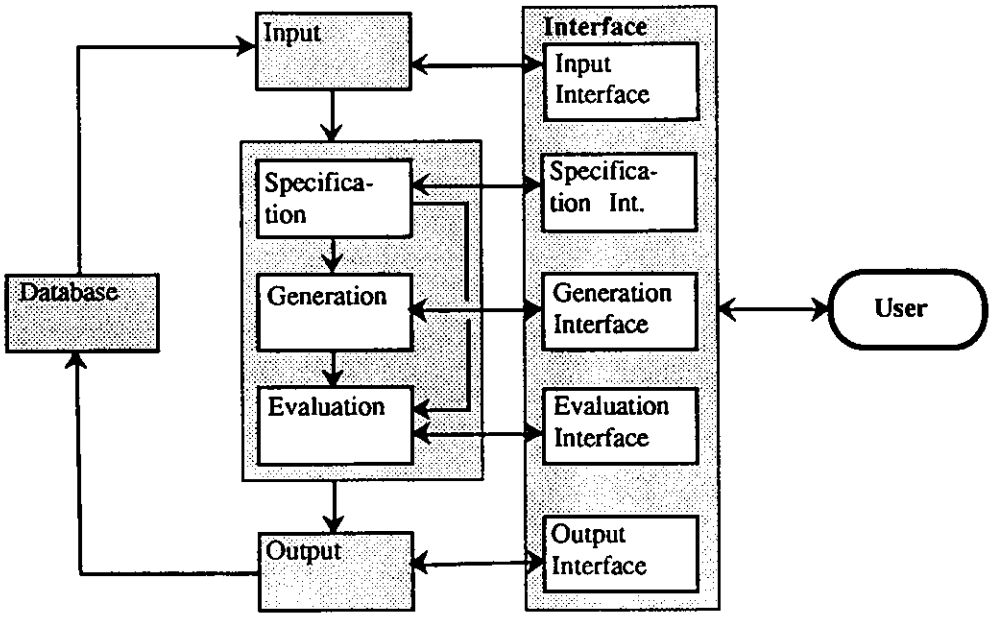

Figure 2-1: Data flow in generic SEED module

Control flow in a module follows data flow with two possible exceptions. First, the designer, acting through the interface, can exercise any form of control offered by a module at any time during the specify-generate-evaluate part of a module's execution. Second, upon completion of each module, a ***task-controller*** containing a task sequence of modules may pass control to the first module in the sequence. If control is returned to the task controller, it passes control to the next task in the sequence. This simple device would permit different overall system architectures to be configured. For example, a linear sequence of tasks occurs when each task controller has exactly one task in its sequence, with the last task being the simple output of results, whereas a tree of tasks can be described by having multiple entries in each task control sequence.

As mentioned above, problem specifications, solution generation and evaluation are under the designer's control through an ***interactive interface*** offering various degrees of intervention in each of the processes, from the total reliance on preprogrammed options, operations, and default values to the possibility of overwriting any default value and manual execution of tasks. For example, the designer may choose in a schematic layout phase to start with prestored standard plans or to generate a custom layout from scratch; furthermore, if the second option was selected, the designer may rely on the generative capabilities of the module to generate a layout automatically or may construct the layout step-by-step interactively. It is this rich palette of opportunities for interaction and intervention that assures the desired flexibility of the system.

The information stored in a database, on the other hand, allows the designer to take advantage of prior work and experience and to expedite the design process. For example, a designer may generate a novel layout by extensively modifying a prototype retrieved from the database, or by generating the layout from scratch, may decide that it should be added to the current set of prototypes, and may consequently add it to the database.

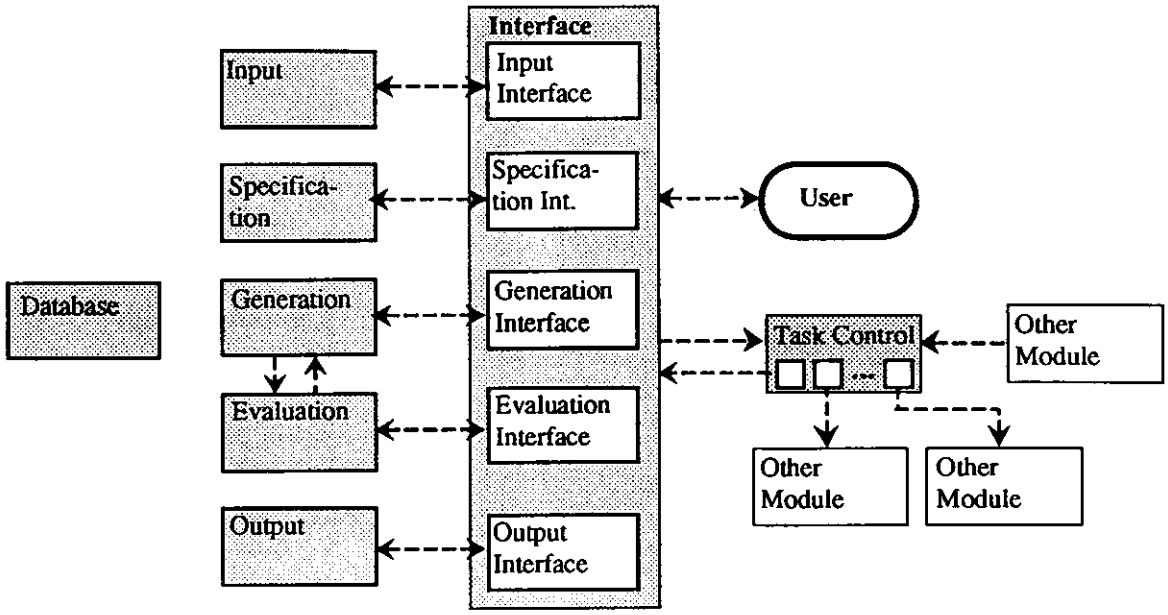

Figure 2-2: Control flow in generic SEED module

3. Modules

We now give a high-level specification of the three modules planned for the first prototype in terms of the generic components outlined above:

Module 1: Architectural programming
Module 2: Schematic layout design
Module 3: Schematic configuration design

We assume that these phases are generally executed in the given order, but iterations are possible, and each phase may be combined with others.

3.1 Module 1: Architectural Programming

The task of this module is to develop an architectural program or design brief for the project under consideration. This program can be viewed as the specification of the overall design problem posed by the project under consideration.

Input

Input from the database to this module may consist of prototypical architectural programs that have been developed in the past for similar projects, templates for program components etc. (see below).

Problem Specification

The specification component allows the designer to develop interactively the overall ***project statement***. We assume that this statement normally comprises at least the following parts:

Overall project goal:
- building type and additional indicators of overall project goals, e.g. desired rental area for an office building, no. students for a school, no. of bedrooms for a house
- budget

Project context:
- site characteristics (geographical location, size, shape, orientation, setbacks etc.)
- pointers to applicable codes and regulations

Generation

The generation component supports the interactive development of an ***architectural program*** from the project statement. The component offers procedures to generate automatically or interactively an architectural program in terms of the following data:

- required rooms and areas
- constraints on the size, orientation etc. of individual rooms and on connections between rooms
- additional criteria, code requirements, recommendations etc.

Evaluation

The evaluation component supports an evaluation of the program in its context (financial feasibility, compliance with codes, life-cycle costs, etc.)

Output

The output component allows the designer to write results produced in this module into the database for various purposes. For example, the project statement and the architectural program can be saved to provide a problem statement for the schematic layout phase, or a project statement and architectural program are in an intermediate state of resolution and saved to allow additional work in a subsequent session. The same data may also serve as a prototype to be re-used when a program for a similar project is to be developed; in this case, the project description may serve as index for the retrieval of this prototype

Database

In order to support the module effectively, the database must contain at least the following information:

- a collection of generic room types or classes, including constraints associated with room types, so that the generation component can assemble a list of required rooms from instances of these classes
- building codes
- cost statistics (e.g. typical construction cost/sq.ft for recurring building types)

As the system is used repeatedly for a recurring building type, program prototypes will become available that may drastically reduce the effort required to develop a program for this type of building. The project statements will be stored with the programs and serve as index in the retrieval of such prototypes.

If energy-efficiency is a desired aspect of preliminary design, data supporting the derivation of design guidelines from locational information must also be available in the database.

3.2 Module 2: Schematic Layout Design

The task of this module is to generate a schematic layout of the program units in "center line form"; that is, spatial units are delineated by lines, which indicate possible locations for interior and exterior walls and partitions.

Input

The input component supports this module by reading appropriate information from the database. For example, it can be used to produce an initial problem specification, that is, a context specification and architectural program generated in module **1**. The input component may also retrieve a prototypical solution, or a solution saved in a previous session, which the designer wants to develop further.

Problem Specification

The specification component allows designers to modify or edit the current problem statement, for example, to add rooms, or modify the constraints associated with a room.

Generation

The generation component supports the generation of schematic layouts for the rooms in the program in at least four modes:

- ***retrieval*** of saved or prototypical layouts from the database
- ***designer-controlled construction*** of a layout; that is, the designer controls the addition, deletion, and reinsertion of rooms in the current layout
- ***automatic generation*** of one or all feasible layouts or partial layouts; that is, the system generates layouts that satisfy constraints, guidelines, etc. under its own control
- ***interactive editing*** of a layout created in any one of the above modes.

Designers should be able to switch freely between modes. For example, a designer may ask the system during designer-controlled construction to generate automatically all feasible ways to insert the next unit, select one possibility, and return to designer control.

The basic units that are directly manipulated in each of these modes are the units of the program, not walls or partitions that separate these units from each other. We selected this approach in order to provide a very flexible layout tool that allows designers, for example, to start with "loose" arrangements of major areas and to add gradually other units of the program, including circulation spaces, and to rearrange the layout until it is sufficiently "dense" and developed so that walls can be drawn as center lines.

Evaluation

The evaluation component allows for an evaluation of a layout at any state of development according to the constraints and criteria specified in the current problem statement. Several options should be available:

- evaluation according to all aspects
- evaluation according to selected aspects.

Designers should furthermore be able to execute each option for selected areas or the entire layout. They should also be able to set a switch that triggers an evaluation option automatically after each design decision or only upon an explicit designer request.

Output

The output component allows designers to write any result produced in this module into the database: a schematic layout or layouts in center line representation to be elaborated in module 3; a problem statement or a problem statement/layout pair to provide a basis for work in a subsequent session; or a problem statement/layout pair stored for re-use as a prototype in a similar project. In each case, evaluations can be stored with the respective layouts.

Database

In order to support the module effectively, the database must contain a collection of prototypes that can be adapted to various contexts and allow the storing intermediate solutions. For the editing of the problem statement, the data available for module 1 must remain available for this module.

3.3 Module 3: Schematic Configuration Design

The goal of this module is to generate a schematic three-dimensional configuration of building spaces and physical components as well as evaluations of this configuration according to given criteria.

Input

The input component allows designers to read data from the database to support work in the current module: for example, a context specification and building program generated in module 1, and a schematic layout generated in module 2. Taken together, these data specify initially the problem to be solved.

Problem Specification

This module allows the designer to elaborate and edit the problem specification for the current module, for example, indicate the configuration (zoning and massing) strategies and physical building systems to be applied in this module. These specifications may be prestored in the database, input by the designer, or result from a combination of these two sources.

Generation

The generation component acts by applying the selected configuration strategies and the building systems to the problem specification to generate one or more alternative three-dimensional configurations. Several modes of action are supported:

- retrieval of standard or prototypical configurations or sub-configurations from the database
- definition of a configuration by direct manipulation; analogously to the previous section, this mode will be called ***designer-controlled construction***
- automatic generation of one, some, or (in special cases) all feasible configurations, that is, configurations that meet the goals given in the input module
- automatic generation within a limited spatial and technical search scope. By describing what strategies and technical systems may be applied where in a design and to what resolution, designers can customize their level of interaction with the system
- interactive editing of configurations created in any of the above modes using the operators provided for designer-controlled construction

Designers should be able to switch freely between modes and configurations. For example, a designer may wish to generate a single configuration automatically and then modify that configuration interactively. At a later time, the designer might return to the automatically generated configuration and use it as the basis for continued development.

The units that are directly manipulated in each of these modes are spatial and physical components, which are defined in the spatial strategies and make up the physical building systems described in the specification stage. For example, if sloped roofs framed with trusses are specified, then roofs would exist as conceptual parts of the configuration and could be created and modified in references to roof centerlines, slopes, bay modules, and bay articulations. We selected this approach in order to give maximum flexibility in the development of specific modules, as the conceptual parts that are manipulated in the configuration module are likely to be specific to the particular building type or context being addressed.

Evaluation

The evaluation component allows for evaluations at those stages in the development of a design that provide sufficient information for evaluations to occur. At any time, any eligible evaluation should be able to be invoked on any part of the design to which it applies. Evaluations are self-priming, specification-driven and designer-controlled: they are self-priming because they determine when and where they can apply by examining the state of the configuration and input information; they are specification-driven because each is associated with one or more items provided in the specification, and their execution determines how well that item is satisfied; they are designer-controlled because the designer has ultimate control over when and to what part of a design they should be applied.

Output

The output comprises the input to the module, the specifications used or created, configurations of three-dimensional spaces and physical elements, and the results of evaluations according to the criteria stated in the problem statement. These may be stored in any degree of completion subject to data-dependencies; for example, a configuration may be stored with the input and specifications from which it is derived but without any evaluations. Similarly, the input and specifications might be stored alone.

Database

The database must support storage and access for at least the following information:

- data required for the layout component
- lexicons of configuration strategies and physical building systems
- configurations
- evaluations and evaluation methods

As the system is used repeatedly for a recurrent building type or method of construction, prototypical solutions will become available that may greatly reduce the effort required to develop future solutions. These will be stored in the database, indexed by the input to which they respond and the specifications by which they are generated.

References

R.F. Coyne and U. Flemming (1990), "Planning in Design Synthesis - Abstraction-based LOOS," *Artificial Intelligence in Engineering. Vol. 1 - Design (Proc. Fifth International Conference, Boston, MA)* , J. Gero, ed., New York: Springer, pp. 91-111

R.F. Coyne (1991), *ABLOOS - An Evolving Hierarchical Design Framework*, Ph.D. Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA

U. Flemming, R. Coyne, T. Glavin and M. Rychener (1988a), "A Generative Expert System for the Design of Building Layouts - Version 2," *Artificial Intelligence in Engineering; Design (Proc. Third International Conference, Palo Alto, CA)* , J. Gero, ed., New York: Elsevier, pp. 445-464

U. Flemming; R. F. Coyne; T. Glavin; Hung Hsi; M. D. Rychener (1988b), *A Generative Expert System for the Design of Building Layouts (Final Report)*, Report 48-15-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA

J. Heisserman (1991), *Generative Geometric Design and Boundary Solid Grammars*, Ph.D. Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA

J. Heisserman and R. Woodbury (1993), "Generating Languages of Solids Models", submitted to *Solids Modeling*