4.1. Problems Analysis

Domain Analysis Concepts Present in Framework Design

The analysis phase of software development deals with the creation of a model for the problem space, whereas the design phase is targeted at creating a corresponding model in the solution space. The earlier versions of a framework are usually the result of the straightforward mapping of domain concepts identified during analysis stage into a very similar design model. Difficulties faced for reusing a framework result in continuous framework redesign [ROB97]. As a framework evolves, most classes, originally representing domain objects, simply disappear or became attributes of other classes, and completely different ones are created, representing generic design solutions for applications in the similar domain [ROB97].

Design patterns provide solutions to recurrent problems that limit flexibility and reusability of framework designs. In particular, they are an invaluable aid to recognize limitations due to straightforward domain concepts representation in a framework, and to make the transition from analysis to reusable design frameworks.

With the aid of design patterns, we have realized a recurrent problem in all frameworks designed for the Capital Budgeting case study: most framework classes had a direct counterpart in terms of domain concepts identified during analysis. They were thus a model of the domain, and not a generic reusable design. A very simple example is the Tree framework, originally designed as a specialization of DAG. Classes of Tree framework included almost no additional functionality with regard to their superclasses, and they served to distinguish the structural properties of two different types of graphs. The evolution of DAG and Tree frameworks for representation of reusable designs, instead of domain concepts, is discussed in Section 4.2.

Emphasis on the White-Box Approach for Creating, Adapting and Connecting Frameworks

As pointed out in Section 2, black-box frameworks are easier to use than white-box ones. Consequently, several framework experts recommend the use of techniques that result in black-box frameworks, such as the catalog of design patterns of [GAM94]. Design patterns reflect the experience of framework developers on finding solutions to common problems that jeopardize reusability, and most of these solutions lead to more mature, black-box frameworks. Design patterns are important tools for framework evolution.

Using design patterns, we have observed that all frameworks originally designed for the Capital Budgeting case study belong to the white-box category. More importantly, the assemblage process for connecting frameworks of different parts was also based on this strategy, since it relies on the use of subclassing and dynamic binding for functionally extension. Very often, earlier framework design capturing analysis concepts are also white-box frameworks, since to create black-box frameworks, one must have a deeper comprehension and more experience on future uses and limitations of current design [ROB97]. At earlier stages of framework life-cycle, one often does not have this knowledge. The example presented in Section 4.2 also illustrates how the design patterns known as Strategy and Composite helped to transform a white-box framework design into a black-box one.
Lack of Standard Interfaces to Connect Frameworks of Different Parts

A white-box approach was also suggested for the assemblage process of distinct types of frameworks [BEC96], as briefly illustrated in Section 3. Indeed, the architecture is based on the assumption that to connect objects of different levels, the interface of upper level classes (e.g., presentation) has to be extended, in particular by including almost all interface of the related corresponding lower level classes (e.g., decision), in order to be capable of delegating the appropriate actions. This process leads to a proliferation of subclasses, and the developer has to understand all details of each framework in order to perform such an extension. In addition, if later one wishes to extend the functionality of resolution or decision levels objects, the subclasses generated during the assemblage process have to modified accordingly, or new subclasses have to be created. Therefore, maintenance of specific DSS is also awkward.

The use of design patterns made us realize that to turn frameworks assemblage an easy and transparent process, we have to identify and implement standard interfaces for the communication of objects belonging to different levels. It is through these interfaces that messages will be propagated along the parts, linking the elements of higher levels with their correspondents in the lower ones. The architecture therefore must be redesigned to include such a standard interface for easily connecting presentation and decision objects, as well as decision and resolution objects. Design patterns help to create stable standard interfaces based on black-box techniques, making easier to connect and substitute components that implement these interfaces.

To create standard interfaces, one must have a deep knowledge about the future uses of a given framework. For instance, to create a standard interface to connect decision situation and resolution frameworks, we have to carefully study and identify the commonalties between the most used problem solving techniques. Thus, the standard interface have to contain the common protocol of communication hiding the differences between the resolution methods. We are currently studying the possibility of applying patterns as Bridge and Strategy [GAM94] and 1:1 Connection [PRE95] for defining standard interfaces for framework assemblage.

Lack of Appropriate Documentation

Lack of appropriate documentation is a common problem that jeopardize framework reuse [JOH97][JOH92], particularly when white-box frameworks are involved. During the last 5 years, various people in our university were involved in the implementation and use of frameworks for DSS prototypes in the domain of capital budgeting, and we have observed some difficulty in understating the specific set of frameworks designed in [BEC93], as well as the architecture and the assemblage process.

Pattern languages have been used in the last years for guiding software development in general and framework documentation in particular. We are presently investigating how to document the redesigned architecture in a pattern language format, in order to make it easier to understand and use it as a guide for DSS development.

4.2. A Small Example

Due to space limitations, we illustrate the use of design patterns for framework redesign using a very small example, which is nevertheless representative of the invaluable aid provided by design patterns. The example involves two related frameworks, named DAG and Tree. The original