A third approach has emerged from the software engineering field, based on the reuse of domain-components encapsulated in object-oriented (OO) frameworks. Frameworks are a reuse technology [JOH97], and as such, they are expected to improve the quality and productivity of software development by reifying proven software designs and implementations. Frameworks [FAY97][JOH88][JOH97] can be regarded as a high-level application or subsystem architecture, consisting of a set of classes that are specifically designed to be refined and used as a group. A framework represents a generic solution for the development of applications in a specific domain (e.g. graphical editors [JOH92], graphical user interfaces [LEW95]), which can be customized to meet the particularities of the problem at hand. With the use of frameworks, the activity of application development becomes the process of refining and binding pre-defined components.

Works as [EGG92][BIR93][BAU96][BAU97][BEC93][BEC96] address the issue of DSS development rapidity and flexibility through the customization and reuse of frameworks. Frameworks for the domain of financial instruments analysis are reported in [EGG92], [BIR93], [BAU96]. Their experiences confirm that framework design is a hard task for which presently no consistent formal support exists, and therefore subject to much empiricism and creativity, and doomed to a trial-and-error process. As an attempt to manage the complexity of framework development and reuse in large scale industrial banking projects, [BAU97] proposed the Gebos System, an architecture in which distinct types of frameworks are organized in 5 distinct layers: business domain, business section, application, technical kernel and desktop. The Gebos System makes it possible to configure and adapt new application systems in a comparatively short time, since it identifies and organizes frameworks with loose coupling and fewer dependencies.

In [BEC93], we analyzed and described the different natures of frameworks that can be combined for DSS development, and organized them in a generic OO DSS multi-layer architecture [BEC96], aiming at a less hazardous approach for the development and reuse of frameworks in the DSS domain. The architecture serves as a guide to the selection and combination of frameworks for the development of a specific DSS. Target specific DSS focus on the provision of modeling facilities for decision-makers, making model formulation a simple process of customizing a generic decision model addressing a class of decision problems [BEC95].

To validate the DSS framework-based development approach, two case studies were developed in [BEC93]. Though the results were interesting, in practice the development process was not as easy and flexible as originally expected. As pointed out in [JOH88], reusability does not come for free. It implies the use of good OO design practices [RIE96], and design for change, foreseeing future opportunities of reuse in different contexts, and identifying stable and variable aspects of a generic design. Design patterns [GAM94][PRE95][COP95][VLI96][MAR98] have become a popular way to reuse design experience that cannot be expressed in terms of components, and have been extensively used as a guide for the development of more reusable frameworks. Most design patterns document successful framework designs that overcome problems and situations that make reuse difficult [GAM94].

This paper describes our experience on the use of design patterns to evaluate the limitations of a set of frameworks created for DSS development in the domain of Capital Budgeting, as well as to redesign them. We discuss how patterns were useful not only in the identification of the main design problems with these specific frameworks, but in the generalization of these problems with regard to the whole DSS architecture. We also illustrate the use of design patterns in the redesign of a framework, discussing the advantages of the new design in terms of flexibility for reuse.
The rest of this paper is structured as follows. Section 2 discusses OO frameworks and design patterns. The striking features of the architecture for framework-based DSS development are presented in Section 3, and illustrated by the Capital Budgeting case study. The use of design patterns for recognizing the problems with a set of frameworks developed for a case study in particular, and with the whole architecture in general, are addressed in Section 4, together with illustration of improvements achieved by the application of two design patterns. Conclusions and future work are discussed in Section 5.

2. Frameworks and Design Patterns

Frameworks have emerged in the OO field to improve reusability in software development. They intend to avoid the development of applications from scratch, allowing developers to reuse successful solutions to software problems, encapsulated in frameworks. Frameworks promote not only code reuse, but more importantly, design reuse. They record the decisions made during software design, which is a more creative and difficult to automate task, compared to implementation.

There are several definitions for the concept of framework [JOH88][JOH97][FAY97][GAM94]. Basically, a framework is a generic architecture of communicating classes addressing a family of problems, that can be adapted and extended to meet the particularities of a specific problem. They are an abstraction of software subsystems, just like abstract classes in regard to concrete (i.e. completely implemented) classes. They capture all common properties and behavior of a set of classes that interact and collaborate to solve a generic problem.

A framework usually implements all its fixed or stable elements, and provides an open interface to its variable elements, in which some or all fixed aspects are based. The developer customizes the framework through this interface, creating subclasses and/or connecting components. The stable and volatile aspects of frameworks are sometimes called, respectively, frozen and hot-spots of frameworks [PRE95].

The first and most successful frameworks were in the general domain of graphical user interfaces (GUI). Well-known frameworks in this domain are MVC, MacApp, ET++, Interviews, Borland OWL, to mention just a few [LEW95]. Frameworks have been developed as well for a variety of other domains, such as graphical editors, document editors, operational systems, network and distributed applications, manufacturing, banking and financial engineering, etc. [LEW95][EGG92][JOH92].

According to the way a framework is adapted and extended, it can be categorized as white-box or black-box [JOH88][FAY97]. White-box frameworks are extended by creating subclasses and overriding hook methods (hot-spots). Black-box frameworks are extended by plugging component objects that conform pre-defined standard interfaces. Frameworks hot-spots are in this way implemented by delegating operations to the plugged components.

Very often, white-box frameworks are more difficult to use than black-box, since they usually force the developer to have a deep knowledge about framework implementation details. Also, the use of inheritance to combine and extend functionality may lead to subclasses proliferation, increasing software complexity. Adapting and extending black-box frameworks is easier than white-box, since one just have to know the standard interface defined for connecting the pluggable components that change the standard behavior.