An Object-Oriented Architecture for Decision Support Systems

Karin Becker
Instituto de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre - RS - Brazil
kbecker@music.pucrs.br

François Bodart
Institut d'Informatique
Facultés Universitaires Notre-Dame de la Paix
Namur - Belgium
fbodart@info.fundp.ac.be

Abstract : Reusability is considered to be the key for achieving productivity and quality in software, and much has been claimed about the particular contributions of the object-oriented paradigm towards the achievement of these goals. Frameworks are coarse-grained reuse units, composed of a set of classes specifically designed to be refined and used as a group. In this paper, we discuss the nature of frameworks necessary to build a particular type of systems, namely Decision Support Systems (DSS). DSS are systems intended to improve the effectiveness of decision making, but information technologies can only have a major impact on decision making if techniques allowing the easy and rapid development of DSS are available. Much benefit is expected in terms of easiness and rapidity of development by constructing DSS from domain-oriented reusable components, as well as in terms of quality of DSS in this way developed.

1. Introduction

One of the persistent challenges faced by managers is to comprehend and respond in a timely manner to the opportunities and threats of the market place with decisions that will make the best use of corporate resources. Decision Support Systems (DSS) are computer-based systems intended to improve the effectiveness of decision making, by helping decision-makers use models and data to solve semi-structured to unstructured decision problems [24]. Major problems identified in early DSS designs were: 1) the systems were so time consuming and costly to develop and maintain that they were difficult to adapt to rapid changes in organisation's decision support requirements, and 2) they were too user and problem specific. From these realizations, and aided by rapid advances in information technology, much of DSS research has been oriented towards the investigation of more flexible approaches for the easy and rapid development of DSS.

Despite their differences, most propositions found in the literature aim at the discover of some degree of *genericity* as the foundation for the flexible and rapid development of DSS. Two major trends can then be identified since late 1970's: DSS generators and Generalized Model Management Systems (MMS). DSS generators, such as spreadsheets, financial modeling languages, MS/OR packages, etc., are implementation tools targeted at reducing the time, cost and effort involved in implementing and maintaining a DSS. The focus on Generalized MMS [5, 9, 17, 19] is rather to extend the scope of DSS, so as to allow the support of multiple problems using a variety of models and data sources, as well as the centralized management of models as an organizational resource. In both approaches, the underlying genericity for DSS development is *function-oriented*. Whereas DSS generators offer high-level implementation functions, the genericity sought in GMMS is at the level of generic facilities for development, storage, manipulation, control and effective utilization of models as a corporate resource of an organization, in an attempt of evening it up with Database Management Systems (DBMS), which offer the same facilities for data.

The easy and rapid development of computer-based systems is not an exclusive concern of the DSS field. For example, it does not differ from the basic concerns of Software Engineering with regard to the development of applications in general. In that field, *reuse* is regarded as the key for

improving quality and productivity of software development, with a primary focus on the potential contributions of the object-oriented (OO) approach [16]. A framework [8, 25, 21] 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 [15], operational systems [7]), which can be *customized* to meet the particularities of the problem at hand. Being more coarse-grained reuse units than individual classes, frameworks are expected to promote *application component-oriented reuse*, by addressing the development of applications as the activity of *refining* and *binding* pre-defined, plug-compatible components that are truly application oriented [21]. The development of specific applications based on the reuse of frameworks induces a new application development life-cycle, in which two distinct activities can be distinguished [13]: 1) *Application Engineering*, related to the development of frameworks, and 2) *Application Development*, where frameworks are customized to develop specific applications.

The OO paradigm has lately attracted the attention of the DSS research community, but more emphasis has been given to the benefits of the unifying characteristics of this paradigm in the design of DSS [19, 20, 10, 18]. We have been investigating the potential contributions of the *domain-oriented genericity* underlying OO for the DSS field. The issue of decision problem formulation and modeling base on domain-oriented components has been addressed in [1, 2], where frameworks were used to represent classes of decision problems (e.g. industrial activity planning, capital budgeting), expressed in terms of the prototypical decision elements. This approach is intended to *decision-makers* as a *modeling tool*, since a framework, regarded as a generic decision model for problems of a certain class, can be customized through an *instantiation process* to represent the particular decision problem at hand. Specific models can be constructed by selecting, adapting and combining *problem-domain concepts*. The methodological role played by a framework based modeling environment is discussed in [3].

In this paper, we go one step further, considering the development of the DSS that provide such modeling capabilities. The approach is this time intended to professionals (i.e. Software Engineers, programmers with knowledge on the OO mechanisms) who are to construct DSS through the reuse of frameworks. A similar approach is considered in [11] and [4], who developed frameworks to build DSS in the limited domain of financial instruments analysis. Their experiences confirm that application engineering is a hard task for which presently no consistent formal support exists [22], and therefore subject to much empiricism and creativity, and doomed to a trial-and-error process [25].

The contribution of this paper is the analysis and description of the different natures of frameworks that can be combined for developing DSS, and their organization as a generic OO architecture. In restricting our concerns to the DSS context, we identified a number of *common criteria* for DSS framework engineering, in particular the *nature* of generic solutions for classes of decision problems, the *type of knowledge* they embody and how to organize this knowledge, aiming at a less hazardous approach for performing the application engineering and application development activities in the DSS domain.

The rest of this paper is organized as follows. In Section 2, the striking characteristics of target DSS, i.e. the type of DSS that can adequately be developed through the reuse of the suggested reusable components, are highlighted. The distinct types of reusable components for target DSS are then discussed in Section 3. In Section 4, the application development of DSS is

briefly discussed, and Section 5 draws conclusions.

2. Striking Features of Target DSS

No generic solution can be constructed without a clear understanding of the nature and characteristics of the specific solutions that one wishes to be derived from it. DSS are difficult to define, and many are the sources of disagreements for their definition [23]. A generally agreed upon functional architecture for DSS is the frame of reference proposed by [24], which identifies 3 main functional components, namely the *model component* (MMS), the *data component* (DBMS) and the *dialogue component*. DSS following this functional architecture are referred to as *model-oriented DSS*.

Acceptance of (model-oriented) DSS by managers as a decision-making supporting tool is still limited. Model formulation is typically performed by experts, and managers often feel reluctance on using models they do not fully understand, and in which development they had little participation. One of the major goals in the DSS research is the provision of modeling environments for users who are not modeling specialists [12]. The reusable components discussed in this paper are targeted at the development of *DSS supporting modeling capabilities* for non-experts.

The basic idea is that, by using application component-oriented components to construct the modeling capabilities of DSS, the domain concepts represented by those components become also apparent to *DSS users* (i.e. decision-makers), such that model formulation, from their point of view, becomes the simple process of choosing and applying a set of special-purpose, domain-oriented concepts for describing the decision situation at hand, in a process fully compatible with their profiles, as described in [2]. This and other features of target DSS are further considered below. For illustration purposes, a very simple example of target DSS that could be constructed according to the proposed approach is considered. Figure 1 presents some elements of the modeling facilities of a simplified DSS for capital budgeting decision problems.

a) modeling paradigm: decision-makers should be able to express their specific problems in a conceptual level, directly in terms of decision elements of the problem and independently of resolution and implementation concerns. This is possible if domain-oriented building blocks used to construct the DSS are offered to decision-makers as *modeling concepts*. Model modeling becomes then the activity of customizing a generic decision model for classes of decision problems, by the instantiation of concepts and binding of those instances [2]. For example, the Capital Budgeting [6] class of problems involves decisions about investments whose returns are expected to extend beyond a year. The lifetime of a project can thus span over several years, and it implies forecasting all incomes and expenditures incurred in this period. The profitability of an investment can be evaluated by a number of criteria, such as the Net Present Value (NPV), Internal Rate of Return, Payback Period, etc. Models for this class of decision problem can be structured and formulated in terms of concepts such as *lifetime* of the investment project, *time-schedules* of *incomes* and *expenses*, *cash flow*, *profitability criteria*, *cost of capital*, etc. Figures 1.(a,b) show some of those concepts, and Figure 1.(c) depicts the instantiation of an object *time* by a user.

b) visual representation: domain concepts are given a graphical representation, using familiar presentation structures, such as table, graph, report, chart, etc. It is through this visual representation that users can create and manipulate instances of model concepts. For example, a

tabular form is the most frequent graphical representation for representing Capital Budgeting problems (Figure 1.(a)), with columns representing the concept of time, whereas the rows would represent various concepts such as cash flow, profitability criteria, etc. Instances of those concepts would be created as a result of the creation of columns and rows (Figure 1.(f)). The inherent hierarchical structure among budgetary items, used to describe the incomes and expenditures of an investment project, can be represented using a tree structure. The generic item tree presented in Figure 1.(b) shows to the user the nature of the items that can be specified, and the valid relationships between them. A customized item tree is shown in Figure 1.(d). For instance, the "working cash flow" is the balance between the "results" of the project , and the "taxes" that must be paid.

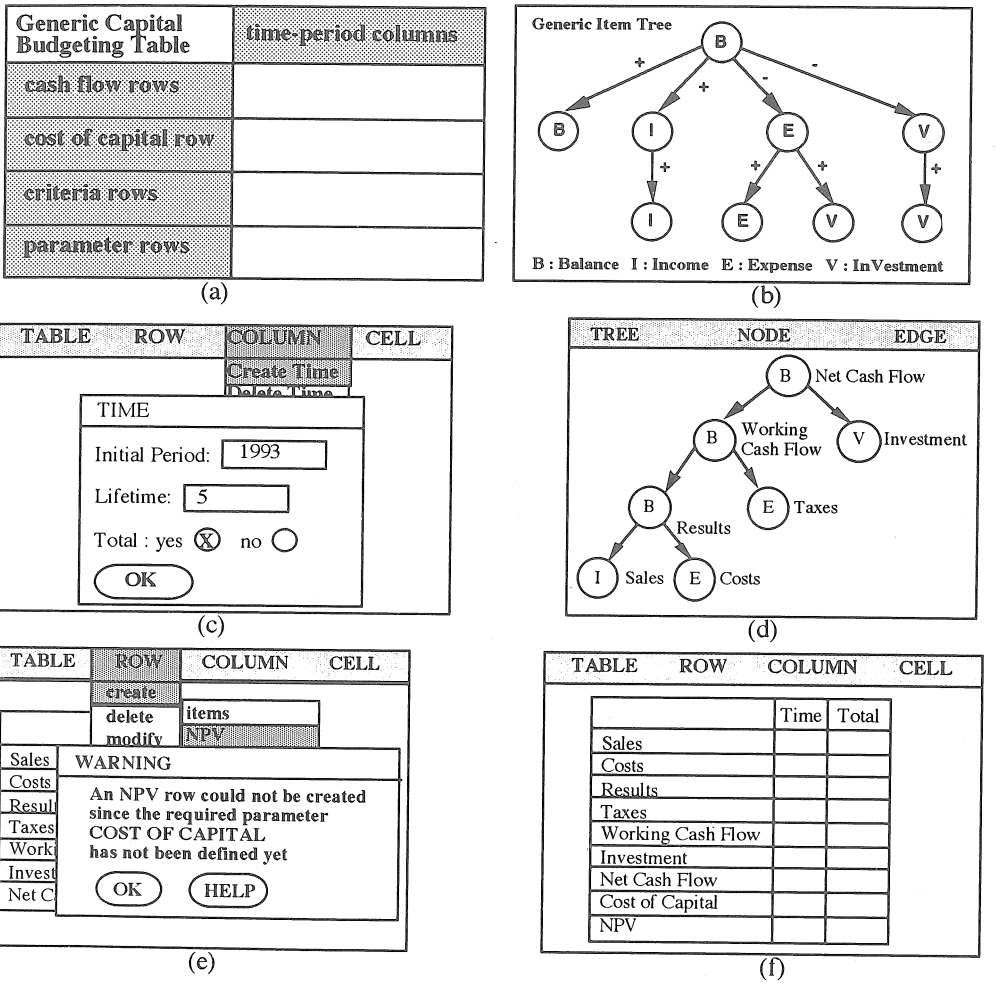

Figure 1 - Modeling Capabilities of a Simplified Capital Budgeting DSS

c) guidelines: guidelines for model formulation are given by the dialogue component. They should help on the selection of concepts for model formulation, and on the instantiation and combination of instances. Guidelines are also necessary for conducting the execution of models. Two simple examples of guidelines related to modeling facilities are depicted in Figure 1. Figure

1.(c) shows the dialogue that guides users in the definition of the life-time of the project (i.e. instantiation of object *time*). In Figure 1.(e), in an attempt to create a row representing the NPV criterion, the user is warned that the parameter cost of capital must be defined first to keep the consistency of the model.

3. Reusable Components for DSS

3.1. OO Architecture for DSS

Considering the target DSS discussed in the previous section, we propose an OO architecture for DSS divided into 3 parts, namely *semantics*, *presentation* and *guidelines* (Figure 2). This architecture corresponds to the one nowadays adopted for interactive applications, which promotes independence between application semantics and interface [14]. The distinction among the parts in the DSS architecture enabled us to identify different responsibilities to be included in the set of classes representing each part, as follows :

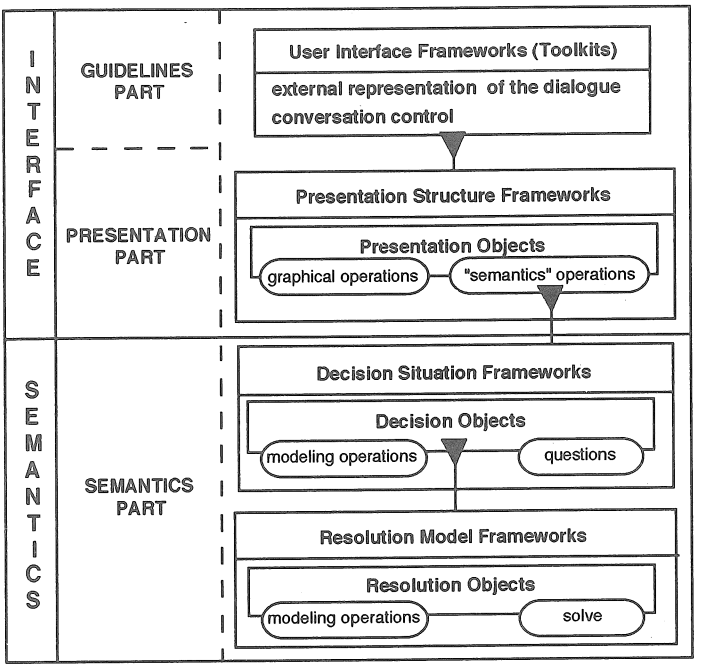

Figure 2 - OO DSS Architecture

- *Semantics Part*: its classes are used to describe a generic solution targeted at a class of decision problems. Methods are offered to model the decision problem at hand, and to solve it, through questions and solving procedures.
- *Presentation Part*: its classes are used to describe how general-purpose representation structures, such as table, graph, etc., can be adapted to become the visual presentation for the concepts captured in the semantics part. Methods are offered to deal with the conceptual meaning of the presentation structure, and with its graphical behavior.
- *Guidelines Part*: its classes capture all conversational aspects of the interface, defining the interaction scenarios for modeling and execution activities.

This distinction led to the identification of *distinct natures* of frameworks potentially reusable in the development of DSS, and the role they play in a DSS. In other words, our architecture

could be thought as a *higher-level framework* for target DSS construction, composed of *lower-level framework types* and *guidelines* to combine them. A framework acts as a "specialized methodology", streamlining the application development activity by fixing in advance as many choices as possible [21], and this is done with regard to DSS by detailing the nature and characteristics of the different building blocks that can be used in its construction, and their relationship for systematic combination. The identified framework types are depicted in Figure 2, and they further discussed in the remaining of this section.

3.2. Reusable Components for the Semantics Part

3.2.1. Decision Models and Model Modeling

Decision model is a semantically overloaded term, where the different interpretations reflects views at different levels ranging from user-oriented to execution-oriented [19]. For our purposes, a decision model is the representation, according to some notation, of the elements characterizing a decision problem, in order to seek, analyze and evaluate possible solutions. As any model, a decision model is always an incomplete and approximate representation of the reality, and as such, it constitutes a *particular approach* for addressing the original decision problem, and can be expressed according to successive *levels of abstraction*. By analogy with the Software Engineering field, we regard the expression of a decision model according to three main abstraction levels, namely *specification*, *design*, and *implementation*, as depicted in Figure 3. The former is concerned with the *problem space*, whereas the latter two deal with the *solution space*.

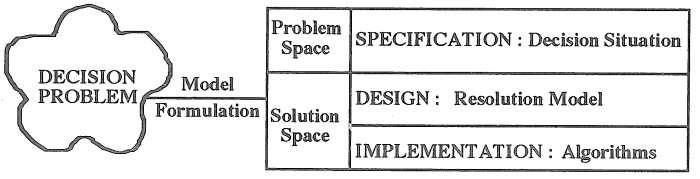

Figure 3 - Decision Model: Abstraction Levels

The solution space in the DSS field concerns the choice of a method or technique for *solving* the problem. Among the dominant resolution methods and technologies, we can mention mathematical programming, spreadsheet capabilities, simulation, etc. Each resolution technique presupposes a particular *modeling paradigm*, consisting of a set of specific concepts and relationships according to which a decision problem has to be structured. For instance, if a decision model is formulated as a linear program, then model modeling consists in identifying the decision variables and coefficients, and structuring them as a linear objective function and a set of linear constraints. At design level, one can abstract from a number of details, such as the choice of algorithms, and the idiosyncrasies of particular languages available in tools that can be used for implementation. Instead, one focuses on the structuring of a decision problem according to the modeling paradigm underlying the resolution method chosen.

The problem space in DSS consists in characterizing the decision situation, independently of the technique to solve it. One of the many ways to approach a decision problem at specification level, is to make use of an existing theory addressing that *class of decision problems*, such as those theories derived from Economics or Financial Management fields, or set of common practices, in this paper referred to as *domain theory*. If a domain theory is used, its underlying concepts and relationships can guide the identification of the elements in the decision problem that

can be used to analyze and evaluate it, and the structuring of these elements in a decision model. For instance, the application of the capital budgeting domain theory to specify an investment problem involves the characterization of an investment project by its lifetime and cash flow, the selection of relevant profitability criteria, etc.

Two different natures of frameworks for developing the semantics part of a DSS were identified, namely the *decision situation framework* and *resolution model framework*, depicted in Figure 2 and further described below.

3.2.3. Decision Situation Framework

A decision situation framework provides a *generic model formulation paradigm* targeted at a particular *class of decision problems*, according to which specific problems of that class can be specified, independently of how they can be solved. A decision situation framework is thus a set of interacting classes that represent domain-oriented, specific-purpose modeling concepts derived from a *domain theory*. Specific decision problems can be specified by creating instances of those concepts.

To act as a model formulation paradigm, a decision situation framework has to capture not only the concepts and relationships underlying a domain theory, but more importantly, the *generic aspects* of its *application* to represent a particular problem. However, the application of a domain theory to specify a decision model is by no means a deterministic process. It may be affected in various degrees by ill-structure of decision problems in general, the particular characteristics of the decision problem at hand, as well as by the particular view a decision-maker might have of the problem. Being the number of possible transpositions of a domain theory for modeling a particular problem very large, possibly and most probably even infinite, it is virtually impossible to capture them all. Therefore, the genericity of a decision situation framework with regard to a class of decision problems relies on the *invariants* that can be identified in modeling decision problems of that class with the concepts and relationships underlying a domain theory.

In representing these modeling invariants, the classes composing a decision situation framework are abstractions that either:

- capture concepts and relationships as they appear in the domain theory;
- factor out common characteristics of a set of concepts for representing higher-level abstractions;
- are specifically added for structuring purposes, or to allow more flexibility in the model modeling process.

The accompanying methods enable to create, update and destroy instances, as well as to retrieve information associated to instances. Methods also exist to define typical questions one would like to pose for problems of that class. Typical questions for the capital budgeting class of problems would be, for instance, "calculate NPV", "maximize NPV", etc. Most methods defined for questions will be abstract [8], since decision situation frameworks are independent of resolution model frameworks. Indeed, different resolution techniques can be applied to the same problem specification, and decision situation and resolution model frameworks are only assembled in the development of specific DSS, as discussed in Section 4.

For illustration purposes, a very limited aspect of the capital budgeting problem is considered: the profitability evaluation of a single investment project by the criterion Net Present Value (NPV). The following elements characterizing an investment project serve as parameters for this

evaluation:
- the lifetime of the project: N;
- the time-schedule of incomes: $I = (I_0, I_1, \cdots , I_N)$;
- the time-schedule of expenses: $E = (E_0, E_1, \cdots , E_N)$;
- the time-schedule of investments: $V = (V_0, V_1, \cdots , V_N)$;
- the net cash flow time-schedule: $CF = (I_0 - E_0 - V_0, I_1 - E_1 - V_1, \cdots , I_N - E_N - V_N)$
- the cost of capital: c

The profitability criterion NPV is defined by Formula 1. The investment is considered profitable if NPV > 0.

$$NPV = \sum_{t=0}^{N} \frac{CF_t}{(1+c)^t} \quad (1)$$

An investment project is not necessarily formulated as concisely as suggested above. The main issue is to characterize the cash flow of the investment project, i.e. the in-out flow of money. This implies identifying all sources of expenditures for implementing the project, as well as the source of all incomes and expenses implied by the project over its lifetime, and distributing them over the entire lifetime of the project. The cash flow of an investment project is actually described by a number of time-schedules representing the elementary incomes, expenses and investments. Additionally, for problem structuring purposes, one can specify as well a number of intermediate time-schedules, representing either intermediate balances between incomes and expenses, or aggregated forms (i.e. totals) of incomes, investments and expenses. In other words, the cash flow of an investment project can be characterized by the successive aggregation of time-schedules. This is just one among the many invariants that can be identified in the specification of investment decision problems.

Figure 4 presents a simplified decision situation framework addressing the capital budgeting class of problems. Due to space limitations, the picture depicts only a few structural relationships between the main classes of the framework.

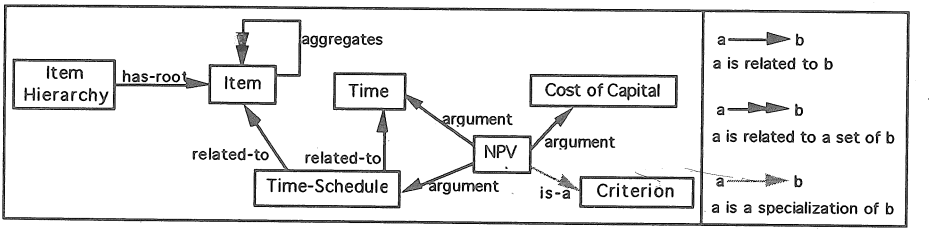

Figure 4 - Simplified Investment Project Analysis Framework

Item is a class factoring out the commonalties between the concepts of income, expense, investment and balance, which are regarded as the attribute *nature* of Item. Items are arranged in a multilevel *Item Hierarchy*, i.e. an item can be an aggregation of items, themselves aggregated or not. The root of the hierarchy is the item representing the net cash flow of the project. A number of constraints are specified for this hierarchical structure (Figure 1.(b)).

A *Time-Schedule* relates an item with the concept of *Time*. It is characterized by an expression defining the forecasting of values for the lifetime of the project. If the related item is an aggregation of other items, it is possible to deduce that the time-schedule is characterized by an algebraic expression based on sum/subtraction of other time-schedules, i.e. those related to the items aggregated. For example, for t=0..N, Production $Costs_t$ = Fixed $Costs_t$ + Variable $Costs_t$.

NPV is one among many profitability criteria, and it is related to its arguments, from which an algebraic expression can also be automatically deduced, according to Formula 1.

3.2.4. Resolution Model Framework

A resolution model framework captures on the one hand the *generic modeling paradigm* underlying a particular resolution technique, and on the other, the *algorithmic details* of that technique. In capturing the modeling paradigm, the set of classes composing a resolution model framework captures the essential concepts, relationships and assumptions necessary to the *formulation* of a decision model in the solution space. Besides the modeling paradigm, it also contain methods that either implement the algorithms for execution, sensitivity analysis and optimization of models [5], or interface with implementation functions of DSS generators/tools. The second alternative is based on the pragmatic observation that target implementation tools for widely used resolution techniques are already available in the market (spreadsheet, MS/OR and statistical packages, etc). According to the solution adopted, the framework may contain additional classes for representing data structures necessary to apply the resolution algorithms, as well as transformation procedures [9]. For example, to solve a linear program, it must be transformed from a set of algebraic expressions (model formulation paradigm) to a matrix that contains only the coefficients of the decision variables.

A simplified example of a resolution model framework is depicted in Figure 5. The resolution model framework is called DAM (*D*escriptive *A*lgebraic *M*odel), and it is essentially based on the specification of directed causal relationships among variables. It assumes a resolution technique where the values assigned to variables are either explicitly given, or deduced by the application of the relationships, thus similar to the resolution technique underlying spreadsheets. Again, Figure 5 shows only main structural relationships between these classes. The accompanying methods enable to create, update, retrieve and destroy instances, as well as solve methods for executing the model.

As a first approximation, a DAM can be thought of as a set of *variables*, where each variable is characterized by an *evaluation expression* (algebraic expression or input expression), defining how values are assigned to the variable. Two other main concepts complete the framework, namely *variable type* and *dimension*. Every variable is derived from a variable type, and it is further classified as *simple variable* or *dimensioned variable*, according to whether it is associated to dimension(s). Whereas a simple variable is an abstraction for a single value, a dimensioned variable is an abstraction for a set or sequence of values. For example, given variable type "sales", and dimension "product", it is possible to define the dimensioned variable "$sales_{product}$" (the sales per product) and the simple variable "sales" (the total sales of products).

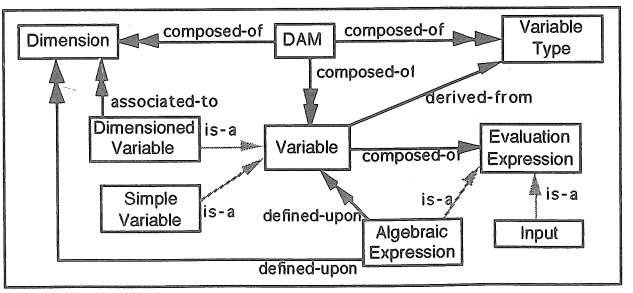

Figure 5 - DAM Resolution Model Framework

3.3. Reusable Components for the Presentation Part

The main goal of the presentation part is to extract from the semantics part those concepts that are relevant for the specification and execution of decision problems of a given class, providing DSS users with a coherent and homogenous view of models, compatible with their cognitive world. This work is based on the premise that there is a number of known general-purpose presentation structures with which decision-makers feel familiar, and that can be conveniently be adapted and used as the visual presentation of a variety of decision problems. A table, for instance, could conveniently represent a budgetary problem, a capital budgeting problem, a forecasting problem, etc. Among the presentation structures familiar to decision-makers, we can mention tables, forms, hierarchical structures, graphs, (textual) algebraic formulae, etc. Frameworks describing general-purpose representation structures are referred to as *presentation structure frameworks*.

General-purpose presentation structures can be characterized by a number of concepts, relationships, and behavior, independently of the particular use that can be made of them. These determine, on the one hand, the underlying basic visual properties of the structure and, on the other hand, how it can be manipulated. For example, a table is composed of a set of interacting rows and columns, and it is manipulated by the insertion/removal/update of rows and columns. The static and dynamic properties defined for the classes of presentation structure frameworks manipulate either the semantics of the presentation (i.e. its underlying concepts), or its visual properties (e.g. size, display). Figure 6 depicts the structural relationships between the classes of the Table framework. Methods include, besides the creation, update and deletion, graphical behavior, such as move (row, column), display, erase, etc.

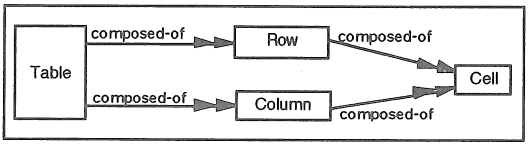

Figure 6 - Table Framework

3.4. Reusable Components for the Guidelines Part

The guidelines part captures all conversational aspects of the interface, defining the interaction scenarios for model formulation and execution. The domain of human-machine interface is one of the most fruitful ones in terms of generic reusable components available for its development. This genericity ranges from fine-grained (or low-level) reusable units, such as the ones offered by toolkits and user-interface classes (e.g. windows, dialogue boxes, buttons) available in the class library of OOPLs, to more coarse-grained user-interface development systems. Any of them is suitable for guidelines part development, given that it requires reusable components for dialogue development.

4. Assemblage of Frameworks for DSS Development

So far, we followed a mental process of *decomposition* to isolate frameworks at different levels of abstraction, and with different roles, based on a clear definition of target DSS. Application development is the complementary composition process, where those reusable components will be refined and put together to develop specific DSS. Typically, application development involves [25]: 1) *defining* any new classes that are needed, most often by the *specialization* of the classes of the framework, and 2) *configuring* applications by creating objects

of those classes and connecting them together, so as to define the global behavior of the application. We use the term *assemblage* just to highlight the fact that in the DSS context, those classes come from distinct framework types.

In the semantics part, the assemblage can comprise three aspects: assemblage of distinct decision situation frameworks, assemblage of distinct resolution frameworks, and assemblage of decision situation framework(s) with resolution model framework(s). The goal of the first kind of assemblage is to enlarge the boundaries of the selected domain, by integrating related sub-problems. For instance, a capital budgeting DSS could address the following set of interrelated problems: analysis of individual investment projects, selection of alternative investment projects under capital constraints, cost of capital estimation, and selection of sources of financing, each of them represented as a framework, and potentially reusable. The purpose of the assemblage of resolution model frameworks is the same: to integrate more resolution paradigms to cope with the needs of decision situation frameworks in terms of questions resolution. The third kind of assemblage is due to the fact that a decision situation framework must be always assembled with at least a resolution model framework, so as to guarantee that formulated models can be analyzed (i.e. executed). The objective is to capture the correspondence between decision concepts and their "equivalent" representation in terms of resolution concepts, as well as to automate the transition from one to another. Figure 7 illustrates some correspondences between the Investment Project Analysis framework (Figure 4) and the DAM framework (Figure 5). The basic idea is that when an instance of a decision concept is created, it triggers the creation of the corresponding instance at the resolution level. This triggering relationship is represented by the black triangles depicted in Figure 2. Likewise, the questions formulated at the highest level need to be related to the methods that actually trigger execution procedures at resolution model level. For example, "calculate NPV" can be solved by triggering the method "calculate" defined for Variable class in the DAM framework.

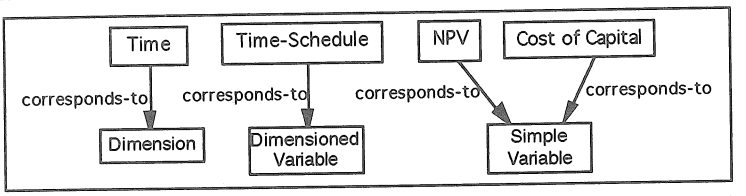

Figure 7 - Correspondence Between Decision and Resolution Concepts

As already mentioned, decision models are given a visual presentation, and it is through this visual representation that users can manipulate (formulate and execute) models. According to the characteristics of the class of problems addressed and cognitive studies, one or more presentation structures are selected to constitute its visual presentation. The presentation structure framework(s) will be *assembled* with the frameworks composing the semantics part of the DSS to associate a meaning to the components of a general-purpose presentation structure. For example, in a table there are rows and columns. In a Capital Budgeting Table there are cash flow rows, criterion rows, time columns, etc. The basic idea is that the users create instances of modeling concepts through the visual, concrete presentation given to them. For instance, the creation of a *Time-Column* instance triggers the creation of an instance of the *Time* class.

Finally, the dialogue for the formulation and execution scenarios are developed, using the user-interface reusable components. It should be clear that any new classes that are needed to

implement additional features (to the DSS semantics or interface) not foreseen by the frameworks can be added at will.

5. Conclusions

In this paper, we discussed an OO approach for the easy and rapid development of DSS, based on the reuse of frameworks of distinct types. The definition of striking characteristics for target DSS, the clear interpretation given for the semantically overloaded term decision model, and the definition of an OO architecture, enabled the discovery of distinct natures of reusable components : decision situation frameworks, resolution model frameworks, presentation structure frameworks and user interface development frameworks. Each type of framework was characterized and exemplified, and their assemblage to develop specific DSS was discussed. This approach can be regarded as a higher-level framework, composed of lower-level frameworks (i.e. the distinct framework types identified) and guidelines to use those frameworks for developing specific DSS.

A prototype DSS for the Capital Budgeting problem is currently under implementation at the Pontifícia Universidade Católica do Rio Grande do Sul, Brazil. This prototype, implemented using the language C++, is being developed by the assemblage of frameworks, among them the frameworks briefly discussed in this paper.

We claim that benefits exists not only for DSS developers, but also for users of resulting DSS. Indeed, it was pointed out that the main goal of target DSS is to support model formulation performed by non-experts, which is a crucial problem that must be overcome if model-oriented DSS are to be more widely accepted. The recommended strategy is to make apparent to DSS users, in the form of available modeling concepts, the same domain-oriented building blocks used for developing the DSS. In that way, also from a user point of view, models are formulated by the customization of generic solutions, in terms of prototypical decision elements of a given class of problems [2].

The suggested approach is limited to the formulation of models for those classes of problems for which there exists at least one well-accepted and well-formed domain theory. The discovery of modeling invariants is the main creative task in the development of frameworks, and certainly the most difficult one. Indeed, due to the ill-defined structure of decision problems, the generic solutions should cope with a variety of contexts and different points of view. The generality of the invariants captured are the determinant factor for the success of a decision situation framework. We believe that the approach is more suitable for the formulation of analysis models, used to asses and refine problems already identified in early stages of the decision-making process. It should be noticed that the other framework types should present no major problems for their engineering, given that they are based on stable and widely accepted knowledge.

Future research topics include the investigation of techniques, models and tools to better support the generic model formulation activity, particularly techniques easing the extraction of knowledge from selected domains. A repository for frameworks storage, location and retrieval is also necessary. Other research topics are the extension of the proposed approach as an aid in the formulation of non-quantitative models, and the investigation of additional DSS features that could be supported with the proposed approach.

Acknowledgments

This work is financially supported by the Conselho Nacional de Desenvolvimento Científico e

Tecnológico (CNPq).

References

[1] BECKER, K. & BODART, F. *Reusable object-oriented specifications for decision support systems.* In: IFIP WG 8.1 Working Conference on the Object Oriented Approach in Information Systems - Quebec City, Oct. 28-31, 1991. Proceedings. North-Holland, 1991. p. 137-155.

[2] BECKER, K. Reusable frameworks for decision support development. Ph.D. Thesis. Namur, Institut d'Informatique - Facultés Universitaires Notre-Dame de la Paix, Sept. 1993.

[3] BECKER, K. & BODART, F. *Methodological and software environment for model formulation based on reusable object frameworks* In: Third International Conference on Decision Support Systems - Hong Kong, June 22-24, 1995. Proceedings. Hong Kong University of science and Technology. p. 611-625.

[4] BIRRER, A. & EGGENSCHWILER, T. *Frameworks in the financial engineering domain : an experience report.* In: ECCOP'93 - Kaiserslautern, July 1993. Proceedings. Springer-Verlag, 1993. p. 19-35.

[5] BLANNING, R.W. *A relational theory of model management.* In: HOLSAPPLE, C.W. & WHINSTON, A.B. (eds.). Data base management: theory and applications. Springer-Verlag, 1987. p. 15-53.

[6] BRIGHAM, E.F. & GAPENSKI, L.C. Financial management: theory and practice of managerial finance. (Sixth Edition). Dryden Press, 1991.

[7] CAMPBELL, R.H.; ISLAM, N.; RAILA, D.; MADEANY, P. Designing and implementing CHOICES: an object oriented systems in C++. Communication of the ACM, 36(9): 117-126, Sept. 1993.

[8] DEUTSCH, P. *Design reuse and frameworks in Smalltalk-80 system.* In: BIGGERSTAFF, T. & PERLIS, A.J. (eds). Software reusability : applications and experience. ACM Press, 1989. p. 57-71.

[9] DOLK, D.R. & KONSYNSKI, B. *Knowledge representation for model management systems.* IEEE TSE, 10(6): 619-628, Nov. 1984.

[10] DOLK, D. & ACKROYD, M. *Enterprise modeling and object technology* In: Third International Conference on Decision Support Systems - Hong Kong, June 22-24, 1995. Proceedings. Hong Kong University of science and Technology. p. 235-245.

[11] EGGENSCHWILER, T. & GAMMA, E. *The ET++ SwapManager : Using object technology in the financial engineering domain.* In: OOPSLA'92, Vancouver , Oct. 1992. ACM Press, 1992. p. 166-178.

[12] ELAM, J.J. & KONSYNSKI, B. *Using artificial intelligence techniques to enhance the capabilities of model management systems.* Decision Sciences, 18: 487- 502, 1987.

[13] FUGINI, M.G., NIERSTRASZ, O.; PERNICI, B. *Application development through reuse: the ITHACA tools environment.* SIGOIS Bulletin, 13(2): 38-47, Aug. 1992.

[14] HARTSON, H. & HIX, D. *Human-computer interface development : concepts and systems for its management.* ACM Computing Surveys, 21(1):5-93, March 1989.

[15] JOHNSON, R.E. *Documenting frameworks using patterns.* In: OOPSLA'92 - Vancouver, Oct. 18-22, 1992. ACM SIGPLAN Notices, 27(10): 63-76. Oct. 1992.

[16] JOHNSON, P. *Object-oriented technology: the competitive advantage.* GEC, 9(1): 28-41, 1993.

[17] KONSYNSKY, B. *Model management in decision support systems.* In: HOLSAPPLE, C.W. & WHINSTON, A.B. (eds). Database management : theory and applications. Reidel Publishing Company, 1993. p. 131-154.

[18] MA, J. *An object-oriented framework for model management.* Decision Support Systems, 13(2): 133-149, Feb. 1995.

[19] MUHANA, W.A. *An object-oriented framework for model management and DSS development.* Decision Support Systems, 9(2): 217-229, Feb. 1993.

[20] MUHANA, W.A. *SYMMS: a model management systems that supports model reuse, sharing and integration.* European Journal of Operations Research, 72(2): 214-242, Jan. 1994.

[21] NIERSTRASZ, O.; GIBBS, S.; TSICHRITZIS, D. *Component-oriented software development.* Communications of ACM, 35(9): 160-165, Sept. 1992.

[22] PRIETO-DIAZ, R. & ARANGO, G. Domain analysis and software systems modeling. IEEE Computer Press, 1991.

[23] SILVER, M.S. Systems that support decision-makers : description and analysis. John Wiley & Sons, 1991.

[24] SPRAGUE, R.H. *A framework for research on DSS.* In: FICK, G. & SPRAGUE, R. (eds.). Decision support systems: issues and challenges. Pergamon Press, 1980. p. 5-22.

[25] WIRFS-BROCK, R.J. & JOHNSON, R.E. *Surveying current research in object-oriented design.* Communications of ACM, 32(9): 104-124, Sept. 1990.