Component-Based Development Applied to Energetic Operation Planning of Hydrothermal Power Systems

R. A. L. Rabêlo, A. A. F. M. Carneiro, and R. T. V. Braga

Abstract--**The energetic operation planning of a hydrothermal power system aims to provide an economic and reliable operational policy that embraces time frames ranging from the long term energy control of hydroelectric resources to the electric and hydraulic aspects of daily generation dispatch. Due to the planning complexity and different aspects that this problem has to address, it may require a chain of computational tools to be solved. Those computational tools should permit to evaluate the current and future system performance, reliability, safety, and ability to grow with production or operating requirements. This work applies component-based development (CBD) towards the energetic operational planning of the Brazilian hydrothermal system. CBD is different from previous approaches, such as procedural programming or object-oriented programming, in its separation of component specification from implementation, and in the division of component specifications into interfaces. Further, CBD reestablishes the idea of reuse and introduces new elements.**

Index Terms—**Component, Component-based Development, Energetic Operation Planning, Hydrothermal Power Systems, Interface, Hydroelectric System.**

I. Introduction

THE introduction of competition in the electric energy industry has an important objective, which consists of obtaining economic efficiency in all segments of the industry. In order to reach this objective, electric energy systems should be better modeled. Once search for economic efficiency requires higher risks, which demanding detailed procedures for evaluating the performance of systems under more stressed conditions, to be taken by operators other than those which were taken in the past.

The planning, design and operation of electrical power systems require computational tools to evaluate the current and future system performance, reliability, safety, and ability to grow with production or operating requirements. To perform computer simulation analysis, a power system is represented by a set of data with certain structures. Analysis algorithms are implemented to process the data to produce simulation results. Different applications require different data structures. Different programs (e.g. hydrothermal scheduling, load flow, transmission planning) are built for different applications. With conventional programming languages, the coupling between the data structures and the algorithm procedures are very high [1]. Even minor changes propagate through the whole program [2]. For example, changing the data type of a variable requires changes in all the procedures that use it. As a result, software modification and evolution may require a time period proportional to the size of the software, rather than the magnitude of the changes.

Object-oriented (OO) technology, as a software development methodology, has been successfully applied to develop software for power system simulations [3]. There are OO applications to transmission planning [4], power system dynamics simulation [5], power system state estimation and optimal power flow [6], drive protective relay systems [7], graphical user interfaces [8] and database management [9].

The design process in most engineering disciplines is based on component reuse. Mechanical or electrical engineers do not specify a design where every component has to be manufactured specially. They base their design on components that have been tried and tested in other systems. These components are not just small components such as flanges and valves, but include major sub-systems such as engines, turbines or condensers [10].

On the software perspective, components [11] are units of software structured according to some specific principles. The fundamental principles they adhere to are actually the fundamental principles that underpin object technology:

• Unification of data and function: a software object consists of data values and the functions that process those data. This natural collocation of dependencies between function and data improves cohesion.

• Encapsulation: the client of a software object is insulated from how that software object's data is stored or how its functions are implemented. Therefore, the client depends on the object specification, but not its implementation. This is a very important separation of concerns and is key to managing dependencies between software and reducing coupling.

• Identity: each software object has a unique identity, regardless of state.

Components extend these object principles by elaborating the notion of an object specification with an explicit

This work was supported by the Piauí State Government (Brazil) through Piauí Revenues Secretary.

R. A. L. Rabêlo and A. A. F. M. Carneiro are with the Electrical Engineering Department, University of São Paulo, São Carlos, Brazil (e-mails: ricardor_usp@yahoo.com.br and adriano@sel.eesc.usp.br)

R. T. V. Braga is with the Department of Computing Science, University of São Paulo, São Carlos, Brazil. (e-mail: rtvb@icmc.usp.br).

representation of dependency, called an interface. This is an important level of indirection between the client of an object and the capabilities of that object. The specification of the total capabilities of an object may be split across several interfaces. Furthermore, much of the flexibility in component-based system design comes from the fact that components may offer multiple interfaces. When new requirements appear, support can be provided by adding new interfaces, and this is a better approach to dependency management than one-size-all super interfaces.

The energetic operation planning of a hydrothermal power system aims to determine an operation strategy for each plant. This strategy has to minimize the expected value of the operative cost along the planning horizon and assures the energy market will be supplied according to reliability rates [12]. Due to the planning complexity and the different aspects that this problem has to address, it may require a chain of computational tools to be solved. This paper applies component-based development (CBD) towards the energetic operational planning of the Brazilian hydrothermal system. A brief presentation of the energetic operation planning is presented in Section-2. In Section-3 it is introduced the principles of component-based development, and the development process that was applied to model the system. The requirements and specification artifacts are shown and discussed in Section-4 and Section-5, respectively. The benefits of CBD are discussed in Section-6. In Section-7, the system is tested with actual data from the Brazilian Power System to illustrate some tests carried out under different energetic operation conditions, and highlighting some important characteristics of optimal operation of reservoirs for electric generation. Finally, Section-8 presents the conclusions.

II. ENERGETIC OPERATION PLANNING OF HYDROTHERMAL SYSTEMS

The operation planning of hydrothermal power systems aims to determine a unit generation scheduling that yields an economic and reliable system operation [13]. Power generation in the Brazilian electrical power system is provided mainly by hydro plants. Hydro generation, in contrast to thermal generation, results in variable energy availability depending on hydrological conditions and the operational policies adopted, although thermal generation furnishes a constant energy availability depending only on its power capacity, the usual interruptions for maintenance and forced unavailabilities [14].

The main issue in hydroelectric energy operational planning is the difficulty of adequately handling the stochastic nature of inflows and the individualized representation of hydro power plants [15]. A very common approach has been to aggregate the hydro system into composite reservoirs in order to allow its optimization by stochastic dynamic programming [16, 17]. The main limitation of the composite reservoir approach is that it does not adequately take into account some important aspects of the hydroelectric operation, such as water head effects, hydrological diversity and localized spilling [18]. An alternative approach consists of the use of a deterministic optimization tool where an optimal solution with individual hydro representation is obtained for forecasted, historical or synthetic stream flows. On the other hand, a simulation tool can also be useful for energetic operational planning studies, since simulation models allow to evaluate the system operational performance many times under different energetic operation conditions of inflows and load.

The composite reservoir approach produces an inaccurate representation of hydro production and underestimates the spills in the systems, especially in large scale hydro systems where the hydrologic diversity is accentuated. To overcome this inaccuracy it is necessary to represent the hydro system in detail, considering individually each hydro plant with its operational constraints and production characteristics. So, the hydrothermal planning can be formulated as a nonlinear capacitated network flow problem [19]. Further, the network has a very special structure as was pointed out in [20]. It is a temporally expanded arborescence where each node has only two outgoing arcs.

III. COMPONENT-BASED DEVELOPMENT

CBD consists of building software systems from existing components. A component is structurally defined as a reusable set of OO classes [21]. The system architecture is designed in such a way that its components offer operations to each other, through well-defined interfaces. One component can invoke an operation offered by another component just by knowing its interface (and consequently the operations implemented by that interface). An interface implements several operations and a component can offer one or more interfaces. Fig. 1 illustrates UML notation for components. In the figure, the component “Existing Component” offers an interface called “IxMgt”. An “Existing Client” and a “New Client” can call methods of this interface. These clients can also call methods of the “IxMgt+” interface. Therefore, a component can be replaced by a new one if the new component provides the old interfaces.

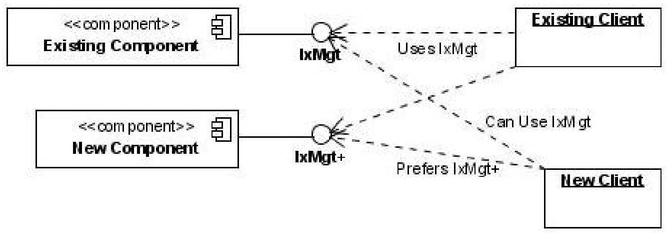


Fig. 1. Interface dependency supports component replacement [11].

CBD is different from previous approaches, such as procedural programming or OO programming, in its separation of component specification from implementation, and in the division of component specifications into interfaces. Splitting component specifications into one or more interfaces means that the intercomponent dependencies can be restricted to individual interfaces, rather than encompass the whole component specification. This reduces the impact of change, because one consumed component may replace another even if it has a different specification, as long as its specification includes the same interfaces the consuming components require. These characteristics allow a component to be upgraded or replaced with minimal impact on the clients of that component [11].

One of the primary objectives of a component is that it must be easily replaceable, either by a completely different implementation of the same functions or by an upgraded version of the current implementation. This places the emphasis on the architecture of the system, on being able to manage the total system, as its various components evolve and its requirements change.

Further, CBD reestablishes the idea of reuse and introduces new elements [22]. In CBD, software systems are built by assembling components already developed and prepared for integration. CBD has many advantages. These include more effective management of complexity, reduced time to market, increased productivity, improved quality, a greater degree of consistency, and a wider range of usability [23].

A. The Development Process

Many authors have presented methods to component-based software development, such as Catalysis [24], KobrA [25, 26], and UML Components [11]. All of them use UML (Unified Modeling Language) as the graphical notation. Among these methods, in this work we have adopted UML Components [11], because we considered it simpler to use, and at the same time it supplies all the resources we need in our specific case study. Next we present a brief overview of UML Components.

Fig. 2 shows its overall development process. The boxes represent Workflows. Each workflow represents a stage of the development process, where inputs are processed and outputs are produced, leading to the final component-based software. Our primary concern regarding the UML Components process is the specification workflow. Section-4 covers just those aspects of the requirements workflow that we need to generate inputs for the specification workflow (Section 5).

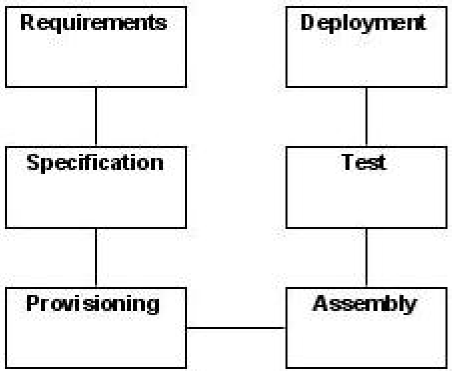


Fig. 2. The workflows in the overall development process.

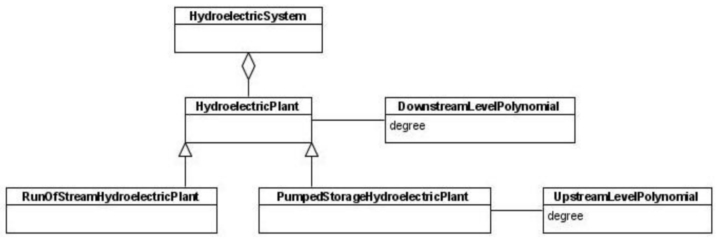
The specification and requirements workflows are responsible for producing a number of modal artifacts. The main requirements artifacts we will be dealing with are the Conceptual Model and the Use Case Model. During the specification workflow we will produce the Interface Specifications, Component Specifications, and Component Architecture.

IV. Requirements Artifacts

Requirements are a description of needs or desires for a product. The primary goal of requirements is to identify and document what is really needed, in a form that clearly communicates to the client and to development team members [27]. The requirements workflow must deliver to the specification workflow a conceptual model and a set of use cases.

A. Conceptual Model

A conceptual model is a representation of concepts in a problem domain [28, 29]. In UML, a conceptual model is illustrated with a set of class diagrams in which no operations are defined. The conceptual model has the advantage of strongly focusing on domain concepts, not software entities. For example, Fig. 3 shows a partial conceptual model in the hydroelectric system domain. It illustrates that the concept of hydroelectric plants, run-of-stream hydroelectric plant, and downstream level polynomial are significant in this domain, that a pumped-storage hydroelectric plant is related to an upstream level polynomial, and that an upstream or a downstream level polynomial has a degree. In addition to decomposing the problem space into comprehensible units (concepts), creating a conceptual model aids in clarifying the terminology or vocabulary of the domain. It can be viewed as a model that communicates (to interested parties such as developers) what the important terms are, and how they are related.


HydroelectricSystem
HydroelectricPlant
DownstreamLevelPolynomial
degree
RunOfStreamHydroelectricPlant
PumpedStorageHydroelectricPlant
UpstreamLevelPolynomial
degree
Fig. 3. Conceptual model in the domain of the hydroelectric system.

B. Use Case Model

A use case describes interactions between a user (or other external actor) and the system, and therefore helps to define the system boundary. The use case model is the set of use cases the developer considers to be representative of the total functional requirements. A sample use case diagram for the energetic operation planning system is shown in Fig 4. The purpose of the diagram is to present a kind of context by which one can quickly understand the external actors of a system and the key ways in which they use it [27].

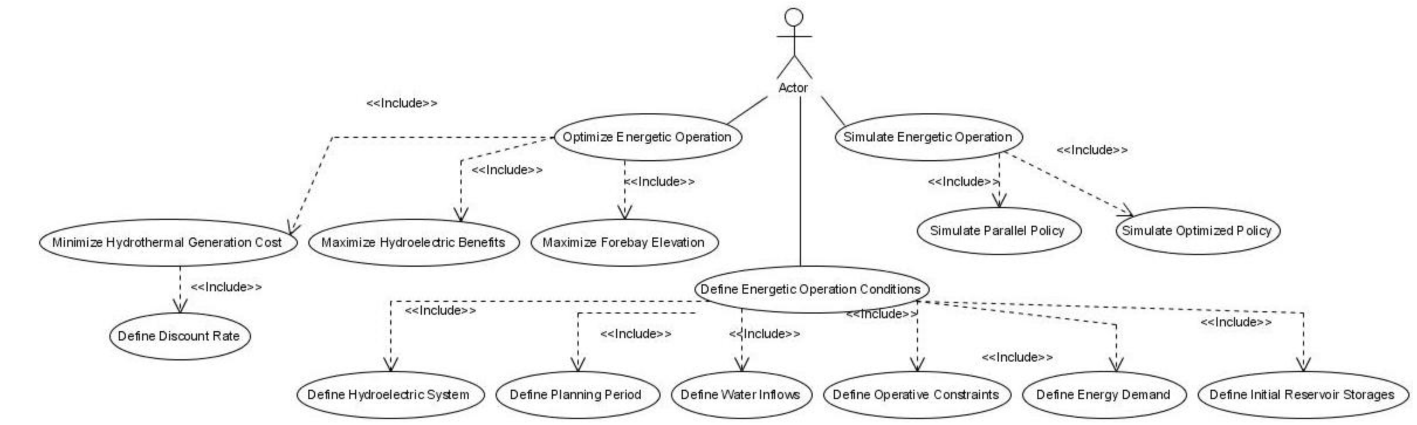

Actor
Optimize Energetic Operation
Simulate Energetic Operation
<<Include>>
Minimize Hydrothermal Generation Cost
Maximize Hydroelectric Benefits
Maximize Forebay Elevation
Simulate Parallel Policy
Simulate Optimized Policy
Define Discount Rate
Define Energetic Operation Conditions
Define Hydroelectric System
Define Planning Period
Define Water Inflows
Define Operative Constraints
Define Energy Demand
Define Initial Reservoir Storages

Fig. 4. The use case diagram.

V. Specification Artifacts

The specification workflow takes as its input (from the requirements workflow) a use case model and a conceptual model. It generates a set of component specifications and a component architecture. These outputs are used several times during the development process: in the provisioning workflow to determine which components to build or to buy; in the assembly workflow to guide the correct integration of components; and in the test workflow as an input to test scripts.

A. Interface Specifications

An interface is a set of operations. Each operation defines some service or function that the component object will perform for the client. The interface defines everything the client needs to know, but no more than that. For example, an interface does not specify how its implementations must interact with other components to fulfill their responsibilities. The interface specification may also imply interactions, but it does not specify how they happen. So, an interface specification defines the relationship between a component object's interface and a client, characterizing a usage contract. Further, an interface of a component can be defined as a specification of its access point [30]. The client accesses the services provided by the component using these points. If a component has multiple access points, each of which represents a different service offered by the component, then the component is expected to have multiple interfaces.

The reason for grouping operations into contractual units (i.e. interfaces) is because their pragmatic usage needs the existence of other operations and because they are rarely used independently of those other operations. Further, interfaces group operations that belong together. This means that if a client uses one operation of an interface, it is quite likely that it will use some of the other operations, too. Sets of such operations naturally belong together and have related effects on the state of the component object [11]. The interfaces can be divided into system and business interfaces. The system interfaces provide access to the services of the system. So the system interfaces and their operations emerge from a consideration of the use case model. As a first-cut approach we define one system interface per base use case (those that include others) as shown in Fig 5.

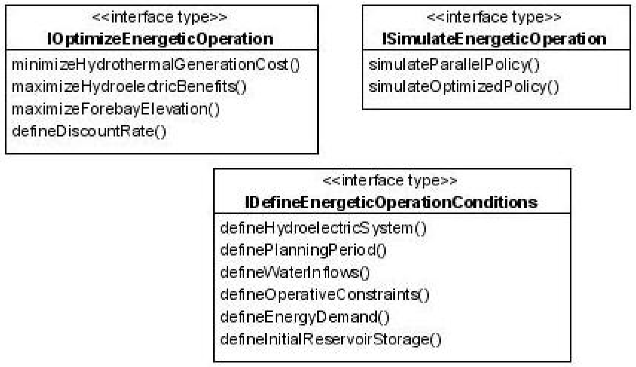

<<interface type>>
IOptimizeEnergeticOperation
minimizeHydrothermalGenerationCost()
maximizeHydroelectricBenefits()
maximizeForebayElevation()
defineDiscountRate()
<<interface type>>
ISimulateEnergeticOperation
simulateParallelPolicy()
simulateOptimizedPolicy()
<<interface type>>
IDefineEnergeticOperationConditions
defineHydroelectricSystem()
definePlanningPeriod()
defineWaterInflows()
defineOperativeConstraints()
defineEnergyDemand()
defineInitialReservoirStorage()

Fig. 5. Initial system interfaces – initial operations

The business interfaces group operations of core business information. They are often reused across several systems. So, the business interfaces are abstractions of the information that must be managed by the system. The business interface for energetic operation of a hydroelectric system by using the individual representation of the hydroelectric plants is shown in Fig. 6.

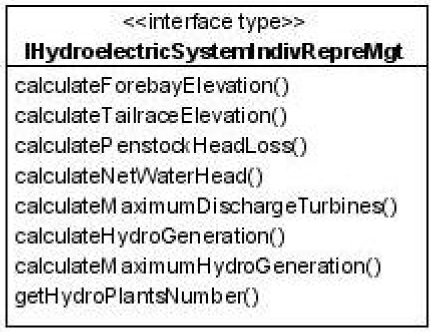

<<interface type>>
IHydroelectricSystemIndivRepreMgt
calculateForebayElevation()
calculateTailraceElevation()
calculatePenstockHeadLoss()
calculateNetWaterHead()
calculateMaximumDischargeTurbines()
calculateHydroGeneration()
calculateMaximumHydroGeneration()
getHydroPlantsNumber()

Fig. 6. Business interface for hydroelectric system.

B. Component Specifications

The interface specifications showed so far deal with the usage contract. Now it is necessary to consider the additional specification information that the component implementer and assembler need to be aware of, specially the dependencies of a component on other interfaces. This information forms the component specification [11].

For every component specification it is important to establish which interfaces its realizations must support. For example, the component specification diagram for the *HydraulicTurbineCatalogDescription* component shown in Fig. 7 tells us that this component must offer the *IHydraulicTurbineCatalogDescriptionMgt* and is not constrained to use any other interfaces. On the other hand, the component specification diagram for the *EnergeticOperationPlanning* system component shown in Fig. 8 tells us that this component must offer the three system interfaces and must use the three other interfaces.

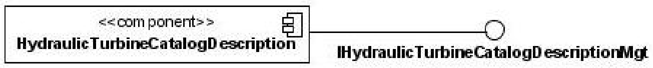

<<component>>
HydraulicTurbineCatalogDescription
IHydraulicTurbineCatalogDescriptionMgt

Fig. 7. Component specification for HydraulicTurbineCatalogDescription.

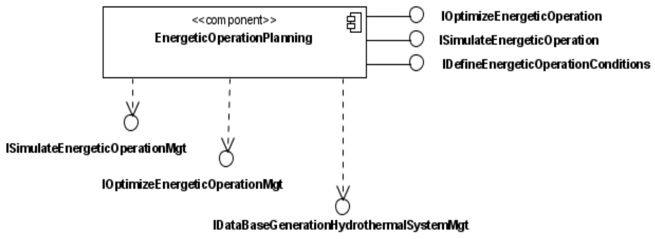

<<component>>
EnergeticOperationPlanning
IOptimizeEnergeticOperation
ISimulateEnergeticOperation
IDefineEnergeticOperationConditions
ISimulateEnergeticOperationMgt
IOptimizeEnergeticOperationMgt
IDataBaseGenerationHydrothermalSystemMgt

Fig. 8. Component specification for Energetic Operation Planning.

Fig. 9 shows us that the *HydroelectricSystem* component must offer two interfaces (the first to operations related to the energetic operation considering the individualized representation of hydro power plants, and the second to operations related to the storage in the database) and must use the offered interface by HydraulicTurbineCatalogDescription Component.

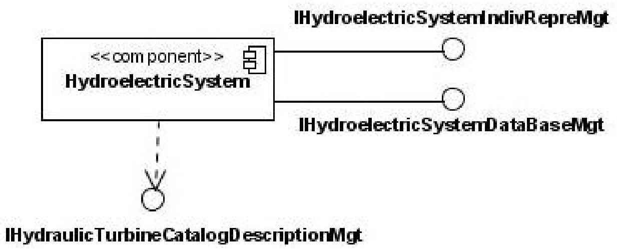

IHydroelectricSystemIndivRepreMgt
<<component>>
HydroelectricSystem
IHydroelectricSystemDataBaseMgt
IHydraulicTurbineCatalogDescriptionMgt

Fig. 9. Component specification for HydroelectricSystem.

C. Component Architecture

Now there is an initial set of component specifications, including their supported interfaces and their interface dependencies. Since there are no interfaces being offered by more than one component specification, we can bind the interface dependencies of the component specifications directly onto their corresponding component specifications interfaces, giving us the component specification architecture shown in Fig. 10. The component architecture describes how the component specifications fit together in a given configuration. The architecture shows how the building blocks are combined to form a system that meets the requirements.

VI. Discussion

Components adopt object principles: unification of data and function, encapsulation, and identity. Components extend these principles by strengthening the role of the interface and by adding a separate notion of component specification. It is important to note that an interface offers no implementation of any of its operations. Instead, it merely names a collection of operations and provides only the descriptions and the protocols of these operations. This separation makes it possible to replace the implementation part without changing the interface, and in this way improve system performance without rebuilding the system; also, new interfaces (and implementations) can be added without changing the existing implementation, thus improving the component adaptability. For example, if it is necessary to develop some computational tools for considering the composite representation of hydroelectric systems, the designers will only care about developing a new interface on *HydroelectricSystem* component without bothering with other details. Fig. 11 shows a sketch of the new specification component. So the *HydroelectricSystem* component' clients may also be flexible in their requirements. They may require certain interfaces as a minimum, but will leverage additional or newer interfaces if they are available.

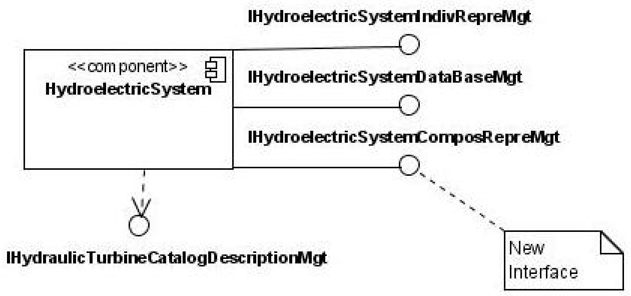
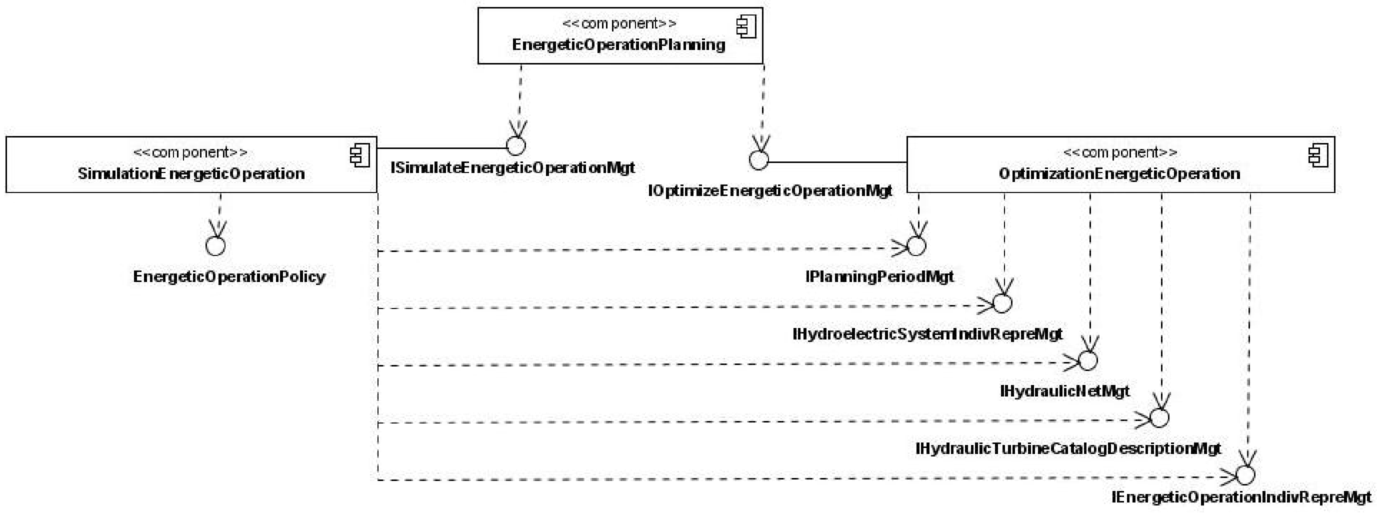

IHydroelectricSystemIndivRepreMgt
<<component>>
HydroelectricSystem
IHydroelectricSystemDataBaseMgt
IHydroelectricSystemComposRepreMgt
IHydraulicTurbineCatalogDescriptionMgt
New Interface

Fig. 11. Providing new interface (functionality) for HydroelectricSystem.

Further, it is possible to design components once and reuse them over and over again in different contexts, thereby realizing large productivity gains, taking advantage of best-in-class solutions, the consequent improved quality and so forth.

Furthermore, a software component can be designed and implemented so that it can be reused in many different programs [31]. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today's user interfaces are built with reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structures and processing detail required to build the interface are contained within a library of reusable components for interface construction.

Coupling is a qualitative measure of the degree to which classes are connected to one another. As classes (and components) become more interdependent, coupling increases.



<<component>>
EnergeticOperationPlanning
<<component>>
SimulationEnergeticOperation
ISimulateEnergeticOperationMgt
IOptimizeEnergeticOperationMgt
<<component>>
OptimizationEnergeticOperation
EnergeticOperationPolicy
IPlanningPeriodMgt
IHydroelectricSystemIndivRepreMgt
IHydraulicNetMgt
IHydraulicTurbineCatalogDescriptionMgt
IEnergeticOperationIndivRepreMgt

Fig. 10. Initial component specification architecture.

An important objective in software design is to keep coupling as low as possible. Splitting component specifications into one or more interfaces means that the intercomponent dependencies can be restricted to individual interfaces, rather than encompass the whole component specification.

Cohesion can be described as the "single-mindness". Within the context of component-level design for object-oriented systems, cohesion implies that a component or class encapsulates only attributes and operations that are closely related to another and to the class or component itself. CBD improves the cohesion of the system because the specification of the total capabilities of an object may be split across several tight related interfaces.

VII. Numerical Results

Components presented in this paper were used for evaluating their performance through some important characteristics of optimal operation of reservoirs. From this way we intend to validate the proposed tool so that it can be used in practice on the energetic operation planning. The system that has been chosen for the case study is composed of 7 of the most important hydro plants in Brazilian Southeast Hydroeletric Power System. They are coupled along the Paraná River Cascade as illustrated in Fig. 12, and correspond to 13220MW of installed capacity.

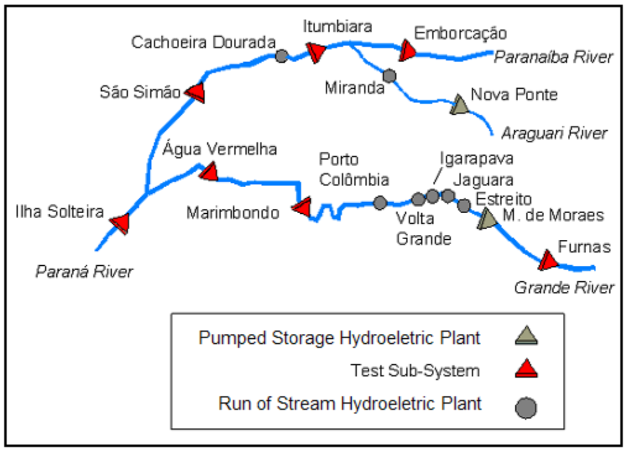


Fig. 12. The test system.

The first test reports the influence of the hydrological condition on the behavior of the reservoirs. Fig. 13 illustrates the storage trajectories for different water inflows of Emborcação (at the left upper side of the Fig. 12). This test shows that as higher inflow level as higher oscillations. Therefore, this result indicates that the lower the inflow oscillation range, the smaller the necessity of regulation in the system.

The second test illustrates the influence of the reservoir location in the cascade over its behavior. Fig. 14 shows the results obtained. The necessary regulation of the cyclic inflows is mainly assumed by the upstream reservoirs (Emborcação and Itumbiara in the figure) whereas the downstream reservoirs operate nearly always as run-of-river plants (Ilha Solteira). This plants behavior maximizes the total generation efficiency of the hydroelectric system. This simple test is sufficient to illustrate that reservoirs could play a completely different role in the system, depending on their relative position in the cascade. For this reason, it was developed an optimized policy to be used in the simulation tool, because the parallel policy leads to a non optimal operation.

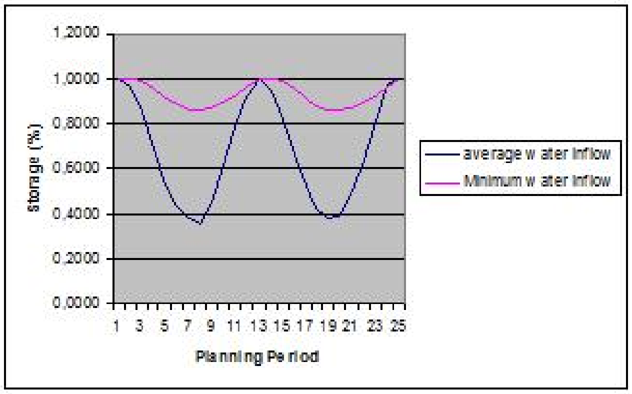


Fig. 13. Emborcacao storage trajectories for different water inflows.

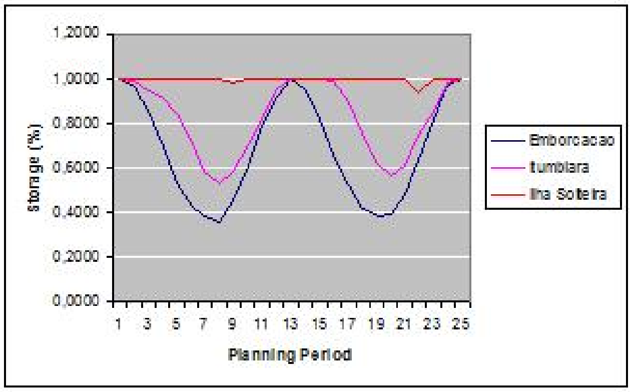


Fig. 14. Emborcacao, Itumbiara and Ilha Solteira storage trajectories.

VIII. Conclusions

The energetic operation planning attempts to minimize system operational costs while attaining specified levels of reliability. This problem is quite complex due to various modeling aspects such as randomness of inflow, interconnection of hydro plants in cascade, nonlinear operational constraints of hydro and thermal units, and transmission constraints of the electric network. Hence, the problem is usually divided into stages so that each stage represents certain details while simplifying or even neglecting others. Thus, the problem requires a chain of computational tools to evaluate the system operational performance many times under different energetic operation conditions.

This paper has been concerned with the component-based development (UML Components) for specifying computational tools applied to the Brazilian generation hydrothermal system. The tools have been specified and developed considering the individual representation of the hydroelectric plants. So, the tools can take full advantage of the hydraulic resources, because the individual characteristics of each plant and the existent hydraulic coupling among them are explicitly considered. The tools also take into account some important aspects of the hydroelectric operation, such as hydrologic diversity, water head effects and localized spilling.

The solution we propose to this complex problem is based on the well known and widely used *Divide Conquer principle*, which states that complex problems can be solved more easily if they are divided into smaller parts. However, solutions that divide the problem simply by using object-oriented concepts are not always good enough, because the interconnections among objects are not clearly documented and understood. Thus, the use of components is proposed to better specify these interconnections.

By taking a component-based approach, and paying particular attention to specification and architecture, the software engineer will be better positioned to create robust but adaptable applications in a cost-effective way.

Furthermore, component architecture may apply to a single application or to a wider context, such as a set of applications serving a particular business process area, for instance, the energetic operation planning. Creating this logical view of the components allows us to understand how tightly or loosely coupled our system is, and to reason about the effects of modifying or replacing a component.

As future work we intend to add new modules to the system, in order to evaluate the effort needed to evolve the component-based version of the system, compared with the previous evolution performed when the system was simply OO. The experience we have already gained so far allows us to foresee that CBD will improve both the level of reuse and the system evolution capacity.

Furthermore, the numerical results obtained show that the proposed tools can be useful for energetic operation planning purposes.

IX. References

[1] E. Z. Zhou, “Object-Oriented Programming, C++ and Power System Simulation”, *IEEE Trans. on Power Systems*, vol. 11, No. 1, February. 1996.

[2] A. F. Neyer, F. F. Wu, and W. Imhof, ”Object-Oriented Programming For Flexible Software: Example of a Load Flow”, *IEEE Trans. On Power Systems*, vol. 5, No. 3, Aug. 1990.

[3] J. Zhu, and P. Jossman, “Application of Design Patterns for Object-Oriented Modeling of Power Systems”, *IEEE Trans. On Power Systems*, vol. 14, No. 2, Mat. 1999.

[4] E. Handschin, M. Heine, D. Konig, T. Nikodem, T. Seibt, and R. Palma, “Object-Oriented Software Engineering for Transmission Planning in Open Access Schemes”, *IEEE Trans. On Power Systems*, vol. 13, No. 1, February. 1998.

[5] A. Manzoni, A. S. e Silva, and I. C. Decker, “Power Systems Dynamics Simulation Using Object-Oriented Programming”, *IEEE Trans. On Power Systems*, vol. 14, No. 1, February. 1999.

[6] S. Pandit, S. A. Soman, and S. A. Khaparde, “Object-Oriented Design for Power System Applications”, *IEEE Computer Applications in Power*, vol. 13, No. 4, October. 2000.

[7] Q. Liu, and S. Cheng, “Object-Oriented Methods Drive Protective Relay System”, IEEE Computer Applications in Power, vol. 13, No. 1, January. 2000.

[8] M. Foley, A. Bose, W. Mitchel, and A. Faustini, “An Object Based Graphical User Interface for Power Systems”, IEEE Trans. On Power Systems, vol. 8, No. 1, February. 1993.

[9] D. G. Flinn, and R. C. Dugan, “A Database for Diverse Power System Simulation Applications”, IEEE Trans. On Power Systems, vol. 7, No. 2, May. 1992.

[10] I. Sommerville, *Software Engineering*, Addison-Wesley. 2006.

[11] J. Cheesman, and J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*. Boston: Addison-Wesley. 2001.

[12] P. T. Leite, A. A. F. M. Carneiro, and A. C. P. L. F. Carvalho, “Energetic Operation Planning Using Genetic Algorithms”, IEEE Trans. On Power Systems, vol. 17, No. 1, February. 2002.

[13] T. Marques, M. Cicogna, and S. Soares, “Assessment of Energy Availability of Hydro System Under Different Operation Policies”, Proceedings of IEEE – Power Tech, Saint Peterburg – Russia, June. 2005.

[14] S. Soares, and A. A. F. M. Carneiro, “Reservoir Operation Rules for Hydroelectric Power System Optimization”, *Proceedings of the 1993 Athenas Power Tech – IEEE/NTVA*, vol. II.

[15] A. A. F. M. Carneiro, S. Soares, and P. S. Bond, “A Large Scale Application of an Optimal Deterministic Hydrothermal Scheduling Algorithm”, *IEEE Trans. On Power Systems*, vol. 5, No. 1. February. 1990.

[16] N. V. Arvanitidis, and J. Rosing, “Composite Representation of a Multireservoir Hydroelectric Power System”, *IEEE Trans. On Power Apparatus and Systems*, vol. PAS-89, No. 2, February. 1970.

[17] N. V. Arvanitidis, and J. Rosing, “Optimal Operation of Multireservoir Systems Using a Composite Representation”, *IEEE Trans. On Power Apparatus and Systems*, vol. PAS-89, No. 2, February. 1970.

[18] S. Soares, and A. A. F. M. Carneiro, “Optimal Operation of Reservoirs for Electric Generation”, *IEEE Trans. On Power Delivery*, vol. 6, No. 3, July. 1991.

[19] M. F. Carvalho, and S. Soares, “An Efficient Hydrothermal Scheduling Algorithm”, IEEE Trans. On Power Systems, vol. PWRS-2, No. 3, August. 1987.

[20] R. E. Rosenthal, “A Nonlinear Network Flow Algorithm for Maximization of Benefits in a Hydroelectric Power System”, Operations Research, vol. 29, no. 4, July-August. 1981.

[21] W. Hironori, F. Yoshiaki, “A technique for automatic component extraction from object-oriented programs by refactoring”. Sci. Comput. Program. 56(1-2): 99-116. 2005.

[22] I. Crnkovic, and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood: Artech House. 2002.

[23] A. W. Brown, *Large-Scale Component-Based Development*, Upper Saddle River, NJ: Prentice Hall. 2000.

[24] D. F. D'Souza, and A. C. Wills, *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Reading, MA: Addison-Wesley. 1998.

[25] C. Atkinson, J. Bayer, C. Bunse, O. Laitenberger, R. Laqua, E. Kamsties, D. Muthig, B. Paech, J. Wust, and J. Zettel, *Component-Based Product Line Engineering with UML*. Component Series, Addison-Wesley. 2001.

[26] C. Atkinson, J. Bayer, and D. Muthig, *Component-Based product Line Development: The KobrA Approach*. The First Software Product Line Conference, August, p.28-31. 2000.

[27] C. Larman, *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development*. Pearson Education. 2005.

[28] J. Martin, and J. Odell, *Object-Oriented Methods: A Foundation*. Englewood Cliffs, NJ: Prentice-Hall. 1995.

[29] M. Fowler, *Analysis Patterns: Reusable Object Models*. Reading, MA: Addison-Wesley. 1996.

[30] C. Szyperski, *Component Software-Beyond Object-Oriented Programming*. Reading, MA: Addison-Wesley. 1998.

[31] Pressman, R. S., *Software Engineering – A Practitioner’s Approach*. McGraw-Hill Companies. 2005.

X. Biographies

Ricardo de Andrade Lira Rabêlo was born in Piaui in Brazil. He received the B.Sc degree in Computer Science from Federal University of Piaui, Teresina. He is pursuing the Ph.D. degree in electrical engineering from the Electrical Engineering Department, São Carlos Engineering School, University of São Paulo, São Paulo, Brazil. His areas of research interest are electric energy systems operation planning, energetic operation policies for reservoirs, fuzzy systems, artificial neural networks, software engineering, and nonlinear programming.

Adriano Alber de França Mendes Carneiro was born in Brazil 1944. He received the B.Sc. degree in electrical engineering from the Catholic University of São Paulo, São Paulo, Brazil, the M.Sc. degree in power systems from the Federal School of Engineering of Itajubá, Itajubá, Brazil, and the Ph.D. degree in operation planning of power systems from the State University of Campinas, Brazil. He is currently an Assistant Professor with the Electrical Engineering Department, São Carlos Engineering School, University of São Paulo, São Carlos, Brazil. His area of research is planning and operations of electric energy systems.

Rosana Teresinha Vaccare Braga was born in Sao Paulo (Brazil), 1965. She is a Professor at ICMC -- University of Sao Paulo - Brazil, where she had also had her PhD in 2003, her Master's degree in 1998, and her B.Sc degree in Computer Science in 1986. She has worked with business systems development from 1986 to 1996. Her research activity is in Software Engineering (Product Lines, Frameworks, Pattern Languages, Aspect-oriented Programming, Reengineering, and Reverse Engineering).