Treating Components and Connectors Explicitly during Software Design

An Approach Based on Software Architecture

Marco Antônio Fagundes de Moraes
Universidade Federal do Pará, Departamento de Informática, Belém, Brazil, 66075-110
Tribunal Regional Eleitoral do Pará, Secretaria de Informática, Belém, Brazil, 66015-902
mfagunde@tre-pa.gov.br

and

Alexandre Marcos Lins de Vasconcelos
Centro de Informática – Universidade Federal de Pernambuco,
Cidade Universitária, Recife, Brazil, 50732-970
amlv@cin.ufpe.br

Abstract

Software Architecture(SA) is considered a critical factor in software design. The adoption of an approach that treats architecture explicitly, emphasizing the separation between "computation" and "communication", is considered an important aspect in obtaining certain benefits (e.g., reuse in high levels of abstraction). However, explicit treatment of SA has not been the focus of the most used software processes, due to some reasons: SA use specific terminology (components, connectors and configuration); the fact that SA is an emerging discipline; and little support from available tools. In this paper, we present ***ArcADe*** (software **Arc**hitecture-based **A**nalysis and **De**sign process), a process that integrates concepts and patterns largely used in SA. This process has been influenced by the RUP (Rational Unified Process) and deals with relationships between requirements and architectural abstractions, elaboration, representation and materialization of software architecture.

Keywords: Software Architecture, Software Reuse, Rational Unified Process (RUP).

1 Introduction

Software architecture is considered a critical factor in software design. The use of an approach that adequately addresses architecture (e.g., Shaw and Garlan [15]), emphasizing the separation between computation (components) and communication (connectors), can be considered a crucial aspect to obtain certain benefits. Some of these benefits include reuse in high levels of abstraction, development management and software evolution facilities, and potential application of other approaches that promote reuse (e.g., Design Patterns[6], and Middleware[4]).

Some approaches to architecture adopted in widely used software processes (e.g. Rational Unified Software Process - RUP [9]) do not clarify the separation between computation and communication in high abstraction levels. In this context, it is visible that there is no natural integration between the SA discipline and widely used processes. Some reasons explain this lack of integration: SA is a recent discipline; SA is not supported by most of the available tools; SA is described through components, connectors and configuration, and not only through class diagrams, as it is common in several object-oriented projects.

To solve the problem of the lack of integration between SA and the broadly used processes, some approaches have been proposed, for example: use UML (Unified Modeling Language) as ADL (Architecture Description Language) [7], Catalysis [3], and MDA (Model-Driven Architecture) [10]. The first approach prescribes the representation of the architecture through UML, instead of traditional ADLs, that present a certain formal rigidity and complex syntaxes. The process Catalysis encompasses all the development cycle, deals with components and connectors explicitly, and uses UML to construct its models. MDA uses UML and emphasizes the construction of models that don't depend on platform and implementation technology, with subsequent translation of these models to platform-specific software architectures.

Considering the aforementioned approaches, two points should be highlighted. First, the approaches that use UML to describe the architecture not encompass all aspects of architecture design (e.g. refinement). Second, Catalysis and MDA are not yet broadly known in software development commercial environments.

Based on the context described, we identify the need of a process that could conduct the architectural design, dealing with components and connectors separately. At the same time, this process should use methods, techniques, and standards (e.g., UML) widely adopted in industry and academy. Because of these factors, we elaborated the ***ArcADe*** Model (Software **Arc**hitecture-Based **A**nalysis and **De**sign Process), which adopts the concepts and principles of SA as its base, and uses RUP elements to organize and represent its workflow. Due us model to adopt largely used patterns (e.g. UML), we believe that their acceptance will be facilitated.

Besides this introductory section, this paper is organized as following: Section 2 introduces the basic concepts used in our proposal. Section 3 presents the structure of ***ArcADe***. Section 4 presents a case study. Finally, Section 5 presents an analysis of the model, conclusions, and possible future work.

2 Motivation

The problem of lack of integration between SA and a widely used process, specifically the RUP, was initially treated considering two main aspects: SA approach and how the RUP deals with architecture. These aspects are describes in this section.

2.1 Software Architecture

There are several definitions of SA (e.g. [2],[15]), among which Shaw & Garlan's definition [15] is the most traditionally adopted by the SA community. This definition, used as the basis of the proposed model, asserts that software architectures are generally described in terms of three basic abstractions: components, connectors and configuration (as depicted in Figure 1.a). Essentially, SA presents a software description in which the "computation" (included in components) and the "communication" (embedded in the notion of connectors) are clearly separate [14]. This separation reduces the coupling among the parts of a system, increasing the opportunities of reuse of components and connectors in other contexts.

2.2 Treatment Architecture in the Rational Unified Process (RUP)

RUP [9] is a generic software development process developed by Rational Software Corporation. RUP has four main characteristics: driven by use cases, centered in the architecture, and prescribes an iterative and incremental development. This process has five basic concepts: worker, artifact, activity, workflow, and workflow detail. Worker is a role that can be given to a person or group. Artifact is information produced, modified, or used. Activity is a unit of work that can be executed by a worker. Workflow is a sequence of activities that are grouped in stages of the development (e.g. requirement workflow). Workflow details are used to structure a flow in terms of activities, workers, input and output artifacts.

The RUP adopts a particular approach to deal with architecture (4+1 View Model [9] – see Figure 1.b). In this approach, the architecture description encompasses five views. Architects capture their design decisions in four views (logic, process, implementation, and deployment), and use the fifth view (use cases) to illustrate and validate the previous four.

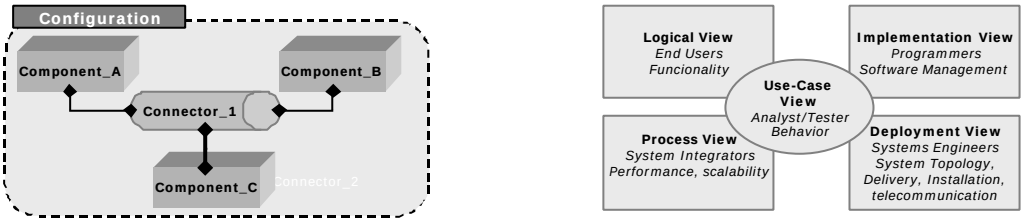


Figure 1: (a) Overview of AS (b) 4+1 View Model

To analyze these views (Figure 1.b), the architecture is described as a collection of use case, design, and deployment models. This consideration is not in conformity with the definition presented in subsection 2.1, in respect to the terminology used in SA (components, connectors and configuration) and in the little emphasis given to the reduction of coupling, consequently reducing reuse opportunities.

Such aspect hinders an adaptation/extension of RUP to treat SA, because the difference is in the base concept: "architecture." Thus, a probable adaptation of RUP would imply in the redefinition/reorganization of all the activities involved in the treatment of the architecture. However, the benefits of RUP, such as, diffusion of use and tool support should not be discarded. Based on this context, we have formulated a process model that integrates SA with RUP elements. This model, named ***ArcADe*** [12] (Software **Arc**hitecture-Based **A**nalysis and **De**sign Process), adopts the concepts and principles of SA as its base, and uses RUP elements to organize and represent its workflow.

2.3 Scope and Use of Model

The *ArcADe* can be applied to any software development organization wanting to reach very abstract levels of reuse and to improve its software development, evolution, operation, and support capabilities. The model doesn't assume a particular organizational structure or management philosophy. On the other hand, it assumes an iterative and incremental development model, organizing the analysis and design activities, so as to assist the professionals involved in the understanding and use of the SA discipline during software development. Concerning the use of *ArcADe*, we highlight the aspects listed in Figure 2.

Who?	Why?	How?	When?
Analysts, Designers, and Software Architects.	To obtain the benefits of SA (e.g., clear structuring of software, reuse in a high level of abstraction, easiness of complexity management through decomposition of the problem.).	Considering a scenario of use (see Section 4), following the workflow defined in the model and executing its activities according to its guidelines.	During the mapping of the requirements to the architecture, together with the realization of the architectural abstractions, in order to obtain a detailed design to be submitted to the implementation.

Figure 2: Use of *ArcADe*

ArcADe can be used in big and complex software projects (e.g. systems with different platforms, multiple programming languages), as well as projects of reduced size and complexity. Additionally, the model can be integrated to an existing process in the organization, as long as the referred process recognizes the input and output artifacts (information produced, modified, or used) of the model. In the next section, we present the abstraction levels and the organization of the proposed model.

3 The *ArcADe* Model

ArcADe (software **Arc**hitecture-based **A**nalysis and **De**sign process) is an analysis and design process model that integrates SA with RUP elements (concepts, methods, and techniques). The model adopts SA as the basis for definition of the stages of development and uses RUP elements to organize and to represent the process in a workflow. We emphasize that our model is not an adaptation/extension of RUP, but an integration of SA and the elements of that process.

It is important to emphasize that the integration must encompass the functional requirements (FRs) as well as the non-functional (NFRs) ones. However, in our proposal, we considered only the functional requirements, because RUP is driven by use cases, which are only employed in the description of the functional requirements.

A future adaptation of the proposed model to treat non-functional requirements can be made using techniques proposed in [14], for example. In the following subsections, we present the abstraction levels, organization and the structure of the proposed model.

3.1 Abstraction Levels

ArcADe follows the iterative and incremental development model adopted by the RUP. In this model, the basic idea is to compose an abstract architecture from requirement analysis, supplying a description of components and their interactions in a high level of abstraction. This architecture will have a subset of the architectural abstractions (see architecture design dimension) mapped to a concrete architecture, supplying an architectural representation more directed to the implementation (see detailed design dimension). In the level of detailed design, the architectural abstractions (components and connectors) will be materialized through the use of project strategies (e.g. Design Patterns) or through the incorporation of existing products (e.g. Commercial Off-The-Shelf – COTS [1], Middleware CORBA [16]). This process is repeated (iteratively) until the global architecture (incremental) has been materialized in terms of design elements. This scenery is presented in the Figure 3.

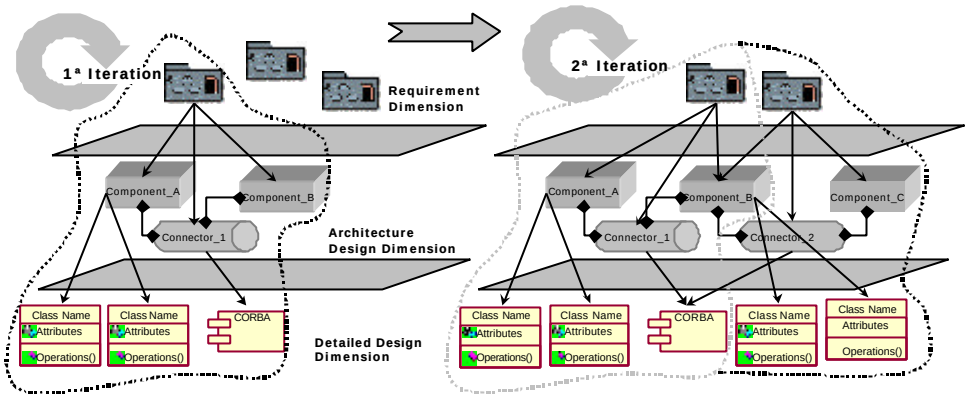


Figure 3: Iterative and Incremental Development in *ArcADe*

3.2 Organization of the Model

The RUP specifies activities, artifacts, and guidelines to treat architecture (according to its approach – see Subsection 2.2). These elements are in the Analysis and Design workflow. Considering this workflow and the abstraction levels presented in subsection 3.1, we have structured *ArcADe* in eight workflow details (see Figure 4.b).

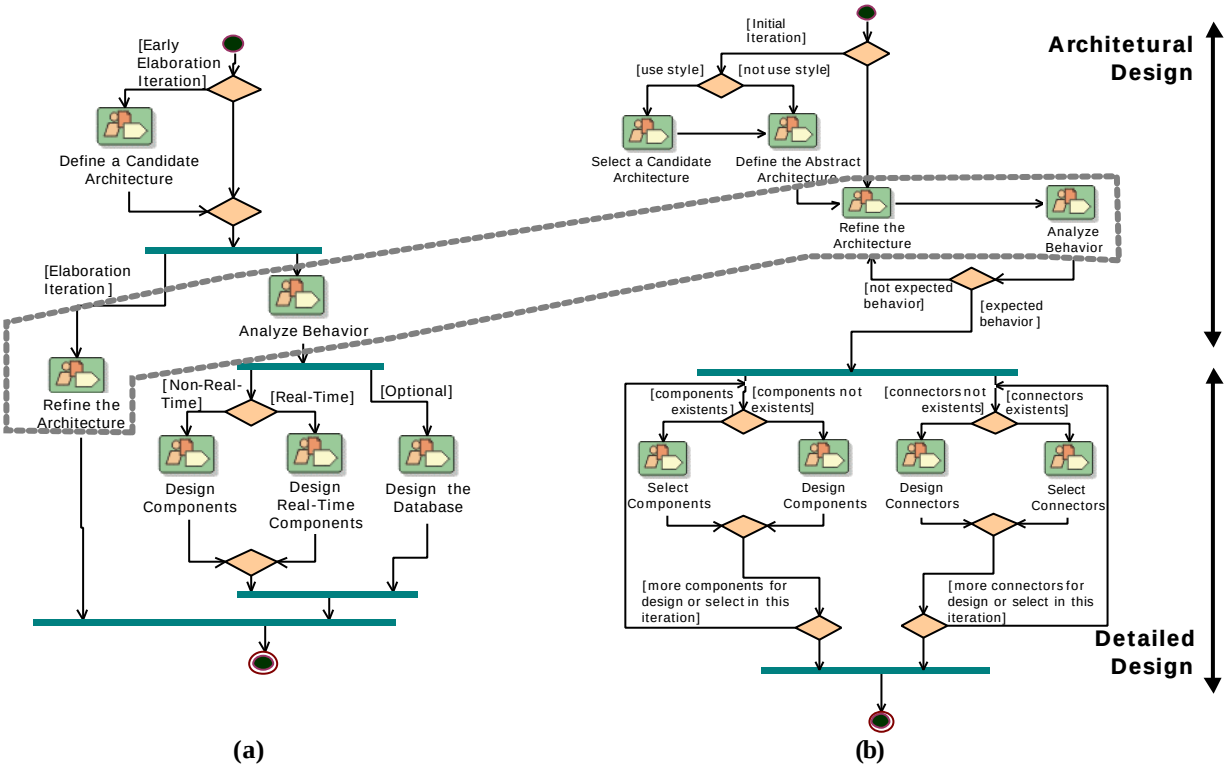


Figure 4: (a) RUP Analysis and Design workflow (b) *ArcADe* Model

The activities presented in the diagrams of Figure 4 represent "workflow details" (see RUP – Figure 4.a – and *ArcADe* – Figure 4.b) that, in certain cases, have the same name. However, in the proposed model, the activities and guidelines involved are based on SA definition (see Subsection 0). Considering the differences in the organization of the two diagrams, it is important to emphasize the execution of the workflow details Refine the Architecture and Analyze Behavior (see highlighted area in Figure 4). In the RUP, this execution is in parallel, but in *ArcADe* it is sequential. We have chosen this organization because the refinement may cause structural changes in the architecture (components and connectors, can be decomposed, aggregated, or removed). Because of this structural change, it is necessary to make an analysis of behavior to verify whether the structural properties are preserved after the refinement. For example, if two components don't communicate in the abstract architecture, they should not be linked in the refined architecture.

3.3 The *ArcADe* Workflow Details

The *ArcADe* supplies a compilation of a set of approaches based on SA (e.g. [4], [5] and [7]) organized in eight workflow details. Each workflow is described as a set of activities, which are detailed step by step. The model also encompasses strategies for software reuse (e.g. Design Patterns and COTS). All these elements are gathered with the objective of guiding developers through the requirement-architecture-implementation abstraction levels.

In the next subsections, we present a summarized description of each workflow detail of the proposed model. This description is grouped according to the levels of abstraction presented in subsection 3.1. We emphasize that some activities/steps of the workflow details were elaborated from the approaches based on SA (as mentioned in the beginning of this section), while others were imported from the RUP.

To illustrate this situation, the Figure 5 and Figure 6 show the workflow details, activities and steps of *ArcADe*, using a convention in the activity/step to indicate its origin: (*) adapted; (+) elaborated; (RUP) imported from the RUP; and (no indication) imported in its original form from the used approach.

3.3.1 Architecture Design Dimension

This level assembles activities designed to relate requirements with an abstract architecture, which will be refined through aggregation, decomposition, or removal of components and connectors. This refinement causes structural changes. Thus, it is important to guarantee the preservation of the architecture's properties by analyzing its behavior. In this level, four workflow details are emphasized: Selection of candidate architecture; Definition of the abstract architecture; Refinement of the architecture; and Behavior analysis.

Select candidate architecture. The CBSP (Component-Bus-System-Property) approach [5] is used to relate domain elements with architectural abstractions (components and connectors) in order to elaborate an intermediary model of the architecture. This model must be related with architectural styles [15] for selection of a candidate architecture.

Define the abstract architecture. Components and connectors of the intermediary model (produced in the previous workflow detail) are mapped to the vocabulary of the candidate architecture and organized according to the configuration rules of an architectural style. As a result, a preliminary model of the architecture is produced. This model shows the services provided by the components and must be converted to the definitive representation of the software architecture using UML. In an adaptation of the proposed model with the objective of dealing with non-functional requirements (as mentioned in the beginning of section 3), it would be possible to use methods to analyze the architecture, such as Software Architecture Analysis Method (SAAM) [2] (like *ArcADe*, SAAM is based in scenarios). However, this topic will be left for future investigation.

Refine the architecture. The abstract architecture (produced in the previous workflow detail) is submitted to refinement rules [13] in order to obtain an architecture in a less abstract level, which will be the basis of the materialization phase of the architectural abstractions.

Analyze behavior. Due to the fact that UML does not support rigorous behavior verification, as the ones allowed by Wright [11], we have performed a limited analysis through the construction of state diagrams for the components (abstract and concrete) involved in the refinement. After analyzing the aforementioned resulting diagrams, it could be verified if the behavior of the global architecture been preserved after the refinement.

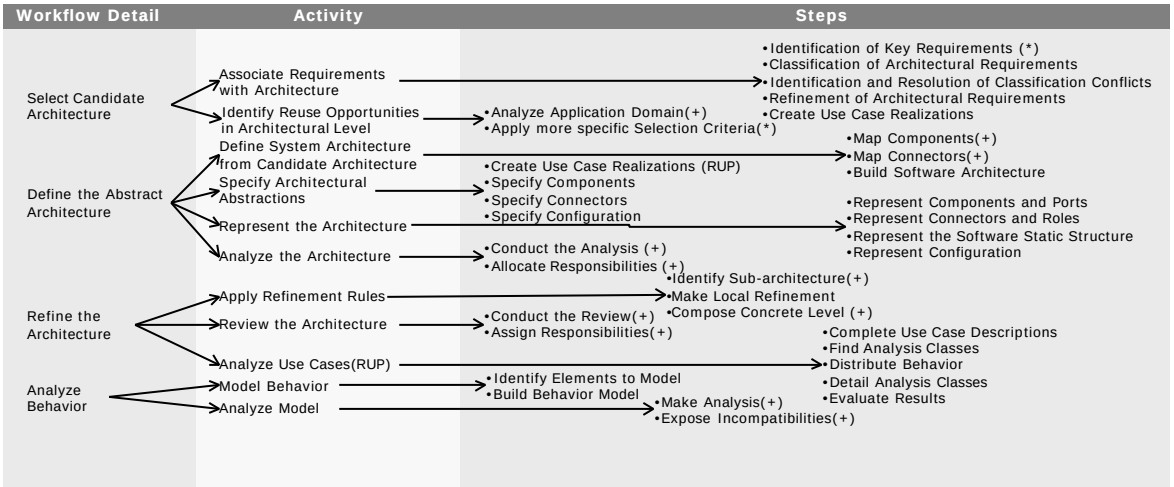


Figure 5: Activity and Step of the Architecture Design Dimension

3.3.2 Detailed Design Dimension

This level assembles activities designed to provide a concrete architecture, which supplies a more implementation-oriented representation, including design strategies (e.g. Design Patterns) or selection of available components and connectors (e.g. Commercial Off-The-Shelf [1]). In this level, four workflow details are emphasized: design or selection of components and design or selection of connectors.

Design components. In a more concrete design level, components can be built through the application of Design Patterns. According to this approach [6], the patterns that supply the solution to the problem are selected. Next, it is necessary to detail class properties (e.g. attributes) and use cases involved, in order to define the structure and functionality of the component. Finally, the OMG IDL (Interface Description Language) [16] is used to specify the interface of the component.

Select components. When confronted with other processes (e.g. RUP), an advantage of ***ArcADe*** is that it guides the construction (design) and also supports the incorporation of existing products (e.g. COTS). For the incorporation, we use the CRE method [1], which identifies the candidate products that fulfill a selection criterion (e.g. Component Interface). This method indicates, through a rank, which product should be integrated in the final system. In case no product is selected, we suggest the execution of the workflow detail Design Components.

Design connectors. Sometimes, the basic type of connections (e.g. Remote Procedure Call) are not enough to fulfill the communication needs between the components. In this case, we have suggested an extension of basic types, using the approach [17] in order to produce a new connector.

Select connectors. Similarly to component reuse, connectors can be mapped to middleware technologies (e.g. ORB). In order to guide the selection process of appropriate middleware, we again suggest the CRE method [1]. From the selection criterion (e.g. support to multiple platforms) the candidate technologies are identified. The CRE method indicates which of these technologies will be integrated in the final system. In case no technology is selected, we suggest the execution of the workflow detail Design Connectors.

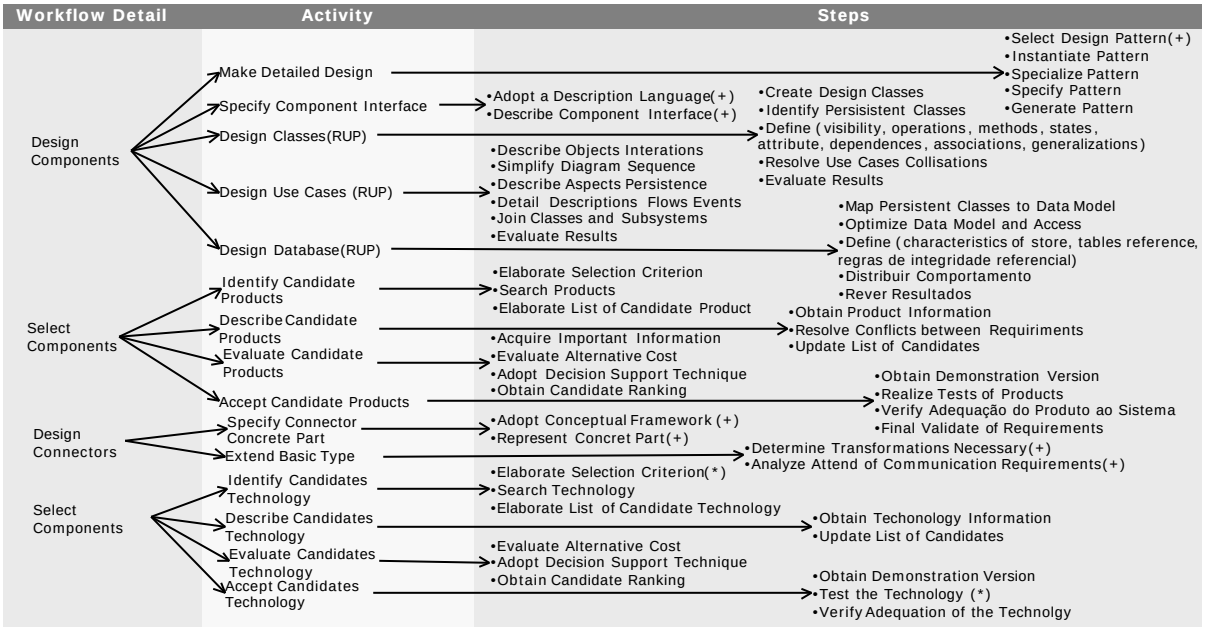


Figure 6: Activity and Step of the Detailed Design Dimension

4 A Use Scenario for *ArcADe*

This section illustrates the use of ***ArcADe*** in a real case study, in which the activities contained in the workflow details are executed based on their guidelines. In this study, we used an application, SIG@UFPE, developed in collaboration with the Information Technology Center of Federal University of Pernambuco (Núcleo de Tecnologia da Informação da Universidade Federal de Pernambuco – NTI/UFPE).

4.1 The SIG@UFPE Application

One of the goals of SIG@UFPE (Information and Academic Administration System of UFPE) is the automation of the academic control process in all teaching levels: under-graduation, graduation, and extension. It provides services like registration, accompaniment, and term finalization. All these operations are accomplished in a decentralized and secure way through the Web.

Because of the existence of several possible execution scenarios of *ArcADe*, and because of the size of SIG@UFPE, we illustrate the use of the proposed model considering only two aspects. First, we selected a representative execution scenario, involving three elements: use of architectural styles; use of nonexistent components; and employment of middleware to materialize connectors. Second, we limited the scope of SIG@UFPE to some of its functionalities (see Subsection 4.2) and we executed each activity of the workflow details of the proposed model according to their guidelines.

This scenario implied in the execution of six workflow details: Select Candidate Architecture, Define the Abstract Architecture, Refine the Architecture, Design Components, and Select Connectors. This execution is detailed in the following subsections

4.2 Base Artifacts for the Execution of the Workflow Details

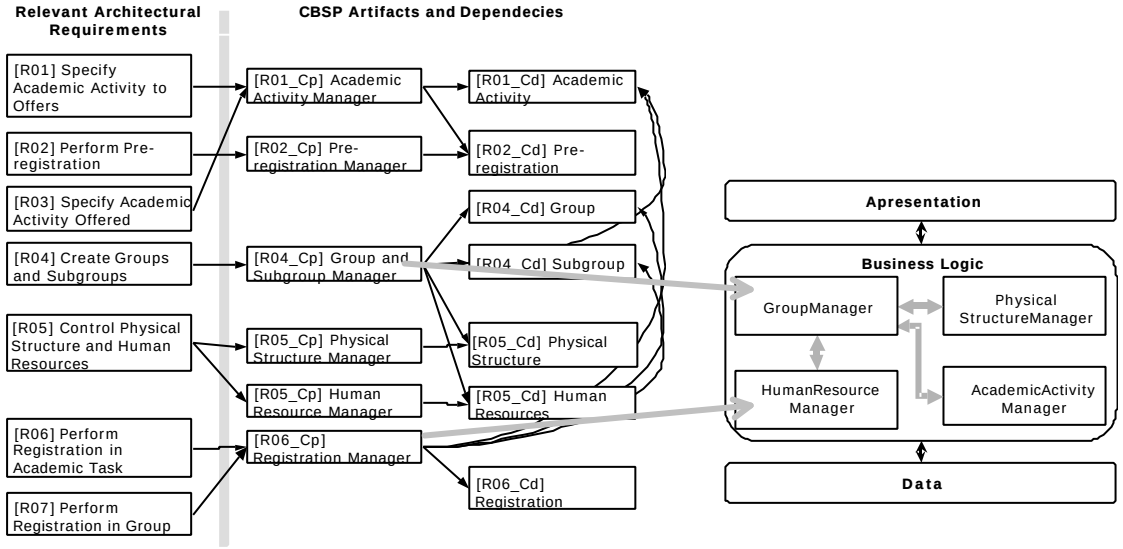
ArcADe considers the requirement document as one of its main inputs. In the SIG@UFPE system, the module used in the case study deals with registration planning. The main functional requirements of this module are showed [R01]...[R07] (see Figure 7.a). Among these, we have limited the scope of the study to the requirement [R04], because we have considered it one of the most representative requirements, due to its complexity degree, number of elements involved during its realization, and need of communication with other components to accomplish its responsibilities.

4.3 Execution of *ArcADe* Workflow Details

4.3.1 Workflow Detail: Select Candidate Architecture

Initially, we associated the requirements with the software architecture via CBSP (Component-Bus-System-Property) approach [5], in which the requirements ([R01]...[R07]) were considered of architectural relevance and classified as processing component (Cp). The dependencies among these components were obtained from the communication needs among them. For example, in Figure 7.a, the component Group and Subgroup Manager depends on information from Physical Structure (data component – Cd).

Components and dependencies help the architect to find an appropriate architectural style. Considering the application domain (Web) the client-server style [15] organized in three levels was selected (Figure 7.b). From the CBSP model (Figure 7.a) and this style, the abstract architecture of the system was defined.



(a) **(b)**

Figure 7: (a) CBSP Model (b) Service relationships

4.3.2 Workflow Detail: Define the Abstract Architecture.

As mentioned in the beginning of Subsection 4.2, we limited the scope of the case study to requirement [R04]. From this requirement, and considering the organization of the architecture in three levels: the GUI component is dedicated to the presentation level; the Manager components are allocated to the level of business logic; and the data components are dedicated to the data level. This preliminary model, which shows the list of services, is depicted in Figure 7.b.

Next, the architectural abstractions are documented in a form based in [11]: Component Interface (Offered Services – input and output ports, Requested Services), Type, and Semantics. The Figure 8 presents a summary of the specification of the GroupManager component using this form.

Interface

Offered Services

1. Input Ports
   specifyAcademicActivity: Offers the service responsible for the assignment of an academic activity to a group.
   ...
2. Output Ports
   listGroup: Supplies data related to a specific group.
   ...

Requested Services

listClassrooms: Requests the service responsible for obtaining the available rooms...

Type

GroupManager: This type of component encapsulates the functionalities to create groups...

Semantics

In order to create a group, the GroupManager must request the academic activities ...

Figure 8: GroupManager component specification (partial)

With the specification of the architectural elements (see The Figure 8) and with the service list model (see Fig. 7.a), we represented the architecture in UML using the approach described in [7] (see Fig. 7.b), where we defined the stereotypes *<<Arch-Component>>* and *<<Arch-Connector>>* to model components and connectors, respectively. The interfaces of the architectural elements were modeled through interface classes. The ports and roles were represented by the stereotypes *<<in>>* and *<<out>>*, indicating the direction of communication. The model of Figure 9.b will be submitted to refinement rules.

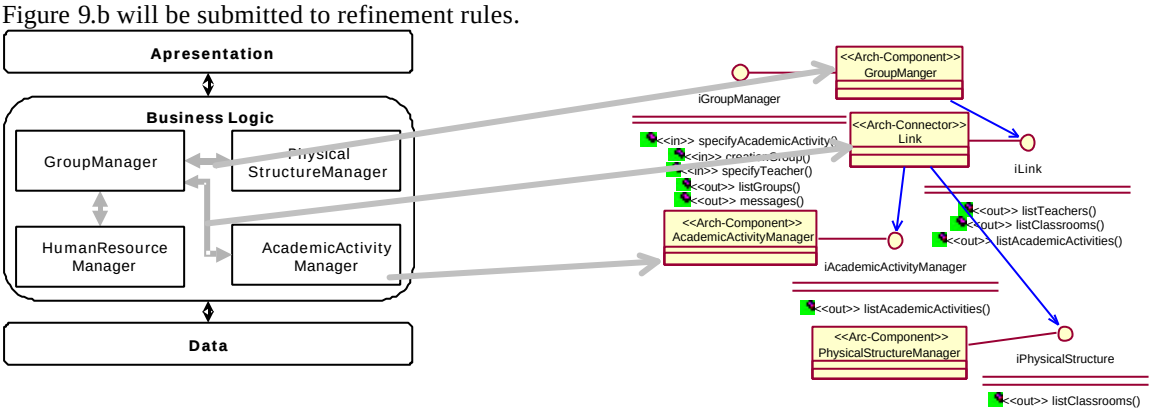


Figure 9: (a) Service relationships (b) UML Representation of the architecture

4.3.3 Workflow Detail: Refine the Architecture

With the abstract architecture defined, the next step is to translate it to a more concrete level. To illustrate this, we refined the component GroupManager by decomposing it in two (see Figure 10). GroupCreationManager offers the service of creating a group (assignment of the academic activity) and managing group details (e.g. classrooms, schedule, etc). TeacherAllocationManager offers the services for allocation of one or more teachers to a group previously created.

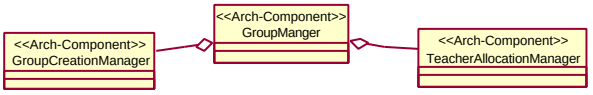


Figure 10: Component GroupManager Refinement

4.3.4 Workflow Detail: Analyze Behavior

Because the refinement may cause structural changes, we must verify if the refined architecture preserves the behavior of the abstract architecture. To accomplish this, we built an UML state diagram for each component involved in the refinement (see Figure 10). As mentioned in Subsection 3.3.1, UML supports a limited behavior analysis, in which we analyze the state diagrams and observe if the behavior of the components resulting from the refinement (see Fig. Figure 11.b and Figure 11.c) is contained in the behavior of the abstract component (see Figure 11.a). In the example, we verified that the behavior of the abstract architecture was preserved.

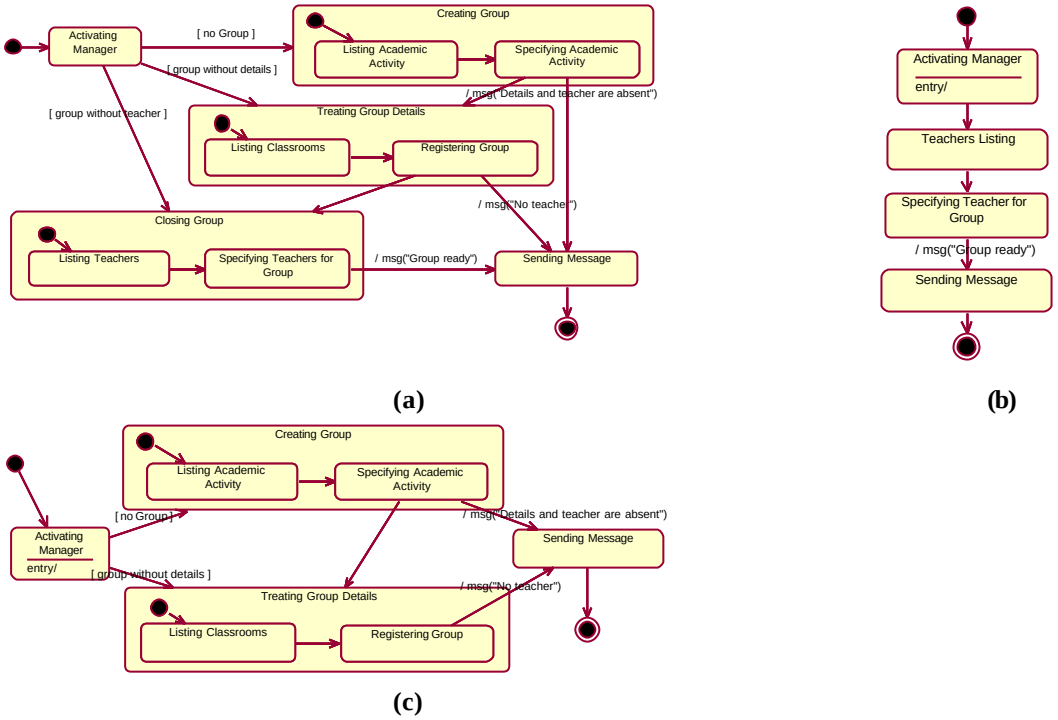


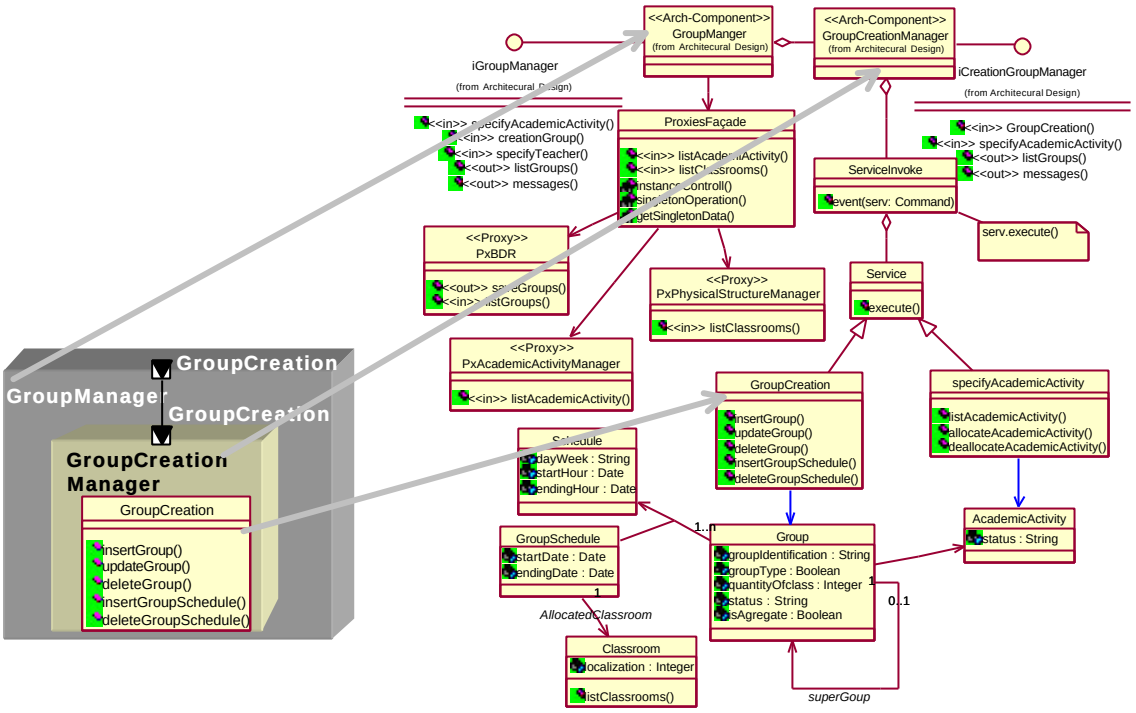
Figure 11: States Diagrams – (a) GroupManager (b)TeacherAlocationManager (c)GroupCreationManager

4.3.5 Workflow Detail: Design Components

Each component of the refined architecture must be implemented via design strategies. Observing the specification of the architectural component (see Figure 8) and the SA representation (see Figure 9.b), we selected three design patterns according to their purposes: Façade, Proxy, and Command. These patterns were applied in the design of the components GroupManager and GroupCreationManager. In this application, the patterns were detailed and instanced in the context of the components (see Figure 12).

In Figure 12.b, we can observe that the component was designed according to the SA discipline (Subsection 0), providing services through ports, which will be implemented by a set of operations. For example, in Figure 10.a the port “GroupCreation” of the external component “GroupManager” is linked to the port “GroupCreation” of the internal component that will be implemented by the class “GroupCreation”. In other words, in the port “GroupCreation”, the operations that will be provided are insertGroup(), updateGroup(), and so on.

After the design of the internal structure and functionality of the component, its interface must be specified with the OMG IDL (Interface Description Language).



(a) **(b)**

Figure 12: GroupManager Component - (a) Informal schema (b)UML detailed design

4.3.6 Workflow Detail: Select Connectors.

Initially, the options of middleware that are potential candidates are located according to the selection criterion (see Figure 13.a). After that, we selected the middleware that fulfilled the criterion (see Fig. Figure 13.b).

Selection Criterion
▪ Intra and Inter-process communication support
▪ Connector characteristics
▪ Platform support
▪ Communication Method

(a)

Technology	Developed by	Source
ILU	Xerox PARC	ftp://ftp.parc.xerox.com/pub/ilu/ilu.html
ORB(Visroker)	Inprise	http://www.inprise.com
RMI	Sun Microsystems	http://java.sun.com/products/jdk/rmi
RMI/IIOP	Sun Microsystems	http://java.sun.com/j2ee/

(b)

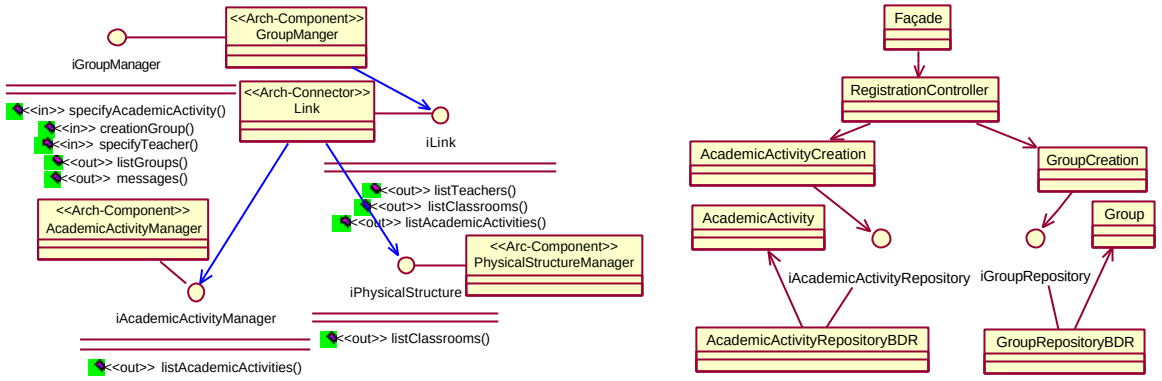
Figure 13: (a) Middleware selection criterion (b) List of candidate *middleware*

After that, we collected detailed information about the technologies, in order to expose the positive and negative characteristics of each one. Analyzing this information, we eliminated two technologies: RMI, for not promoting interoperability; and ILU, for not being so widely known, having little available documentation and support. Finally, we verified which technology better fulfilled the communication requirements. Then, we performed some tests to check legal and communication aspects. Considering the use of the decision-taking technique and the conformity with the tests, the middleware ORB (Visibroker) was selected.

Components and connectors must be integrated to compose the global architecture of the system. This integration should occur in later stages of the development process, particularly, in the system implementation stage.

4.4 Benefits Attained

The benefits attained with the use of ***ArcADe*** are the same ones supplied by SA (e.g. reuse in abstract levels and treatment of connectors as first-class entities). In contrast with the process used in NTI/UFPE (see Figure 14), we can highlight the aspect describe in this section.



(a) **(b)**

Figure 14: Architecture produced: (a) by ***ArcADe*** (b) by the process used in NTI/UFPE

First, in ***ArcADe*** the components are differentiated from each other through a stereotype indicative of its type. Second, the proposed model treats architectural elements in very abstract levels. For example, in the interface of the components, the services represent a set of operations to be provided. Particularly, the service "create groups" supplies the inclusion, exclusion and alteration operations for a group. Finally, in ***ArcADe***, connectors have first-class status, that is, the design of connectors becomes explicit (see class Link in Figure 14.a). As a result, there is a reduction in the coupling among system components, increasing the opportunities of reuse of components and connectors in other contexts.

NTI-RUP is based in RUP. As a result, it has the same differences in the treatment given to architecture, according to the definition used in our model (see Subsection 0 and 2.2). ***ArcADe***, on the other hand, is based on the SA discipline, resulting in its inherent benefits.

In NTI-RUP, in code level, each basic class (e.g., Group) generates a package (directory) that has the basic class, the repository interface, the collection class, and the RDB(Relational Data Base) repository class. In order to make it possible to reference a class in another package, it is necessary to import the package. As a result, the communication mechanisms are embedded in the components, making connector design not explicit.

In this scenario, we observed an increase in the coupling among components. Such an aspect reduces the potential reuse of components and connectors. On the other hand, ***ArcADe*** emphasizes the separation between computation elements and communication mechanisms, reducing coupling and facilitating component and connector reuse in other contexts. This is possible because our model treats component design and connector design explicitly and separately, resulting in language independence, treatment of legacy systems, and distribution facilities.

In the case study, we illustrated the implementation of a component (GroupManager) and the selection of middleware to implement a connector. It is worthwhile to point out that ***ArcADe*** considers the iterative and incremental model. Consequently, the other components and connectors must be implemented in future iterations, until the concrete level of the global architecture has been reached.

This case study was important for the incorporation of improvements in our model, because it made it possible to verify the coherence of the activities and steps involved in each workflow detail.

5 Conclusions and Future Research

This paper presented the ***ArcADe*** (Software **Arc**hitecture-Based **A**nalysis and **De**sign Process) model, which defines how the architecture must be explicitly treated in an iterative and incremental development process. The model treats computation elements (components) and communication mechanisms (connectors) explicitly and separately. This approach results in some benefits (e.g., reduction of coupling and consequent increase in reuse opportunities). This model has eight workflow details, which are grouped according to the abstraction levels of the model. The Architecture Design level possesses four workflow details: Select candidate architecture, Define the abstract architecture, Refine the architecture, and Analyze behavior. The Detailed Design level also possesses four workflow details: Design components, Design connectors, Select components, and Select connectors.

The workflow details of the proposed model were tested, experimentally, in the SIG@UFPE project (developed by NTI/UFPE) to illustrate their use, detect inconsistencies, and verify the coherence of the activities and steps defined in each workflow detail.

ArcADe didn't encompass the whole software development process, because it focuses on the analysis and design stage. However, other stages of the process can also be affected and, consequently, need to be adapted. For example, the implementation stage must be redefined, because if a preexisting architectural element is selected, there should be adaptation (and not an implementation) effort.

Only one scenario was used to illustrate the execution of the proposed model, limiting the verification of ***ArcADe*** to just six workflow details. Consequently, the creation of new scenarios to exercise and verify the other workflow details, as well as the definition of the other stages of the software development process, are research work to be analyzed in future, so that we can have a complete development process based on software architecture.

References

[1] Alves, C.: Seleção de Produtos de Software Utilizando uma Abordagem Baseada em Engenharia de Requisitos. Dissertação de Mestrado. Universidade Federal de Pernambuco – UFPE. Mar, 2001.

[2] Bass, L., Clements, P., Kazman, R.: *Software Architecture in Practice*. Addison-Wesley, 1998.

[3] Catalysis. Website: http://www.catalysis.org, All contents copyrighted © 1998. Last access in Fev/2004.

[4] Dashofy, E., Medvidovic, N., Taylor, R.: Using Off-The-Shelf Middleware to Implement Connectors in Distributed Software Architectures. 1998.

[5] Egyed, A., Grunbacher, P., Medvidovic, N.: Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach. In *Proc. of the From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, May 14, 2001.

[6] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos.* Porto Alegre: Bookman, 2000.

[7] Garlan, D., Kompanek, A., Cheng, S.: Reconciling the Needs of Architectural Description with Object-Modeling Notations. http://www-2.cs.cmu.edu/~able/publications/uml01/. Last access in Fev/2004.

[8] Jacobson, I., Griss, M., Jonsson, P.: *Software Reuse – Architecture, Process and Organization for Business Success*. Addison-Wesley, 1997.

[9] Krutchen, P.: *The Unified Software Development Process, An Introduction*. Addison Wesley, 1999.

[10] MDA Website: http://www.omg.org/mda, Last Updated Tuesday, March 12, 2002. Last access in Fev/2004.

[11] Medvidovic, N., Taylor, R.: A Classification and Comparison Framework for Architecture Description Languages. *IEEE Transactions on Software Engineering*, Vol 26. Nº 1, Jan, 2000.

[12] Moraes, M.: Um Framework de Análise e Projeto Baseado em Arquitetura de Software. Dissertação de Mestrado. Universidade Federal de Pernambuco – UFPE. Mar, Jul, 2002.

[13] Moriconi, M., Qian, X., Riemenschneider, R.: Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, April, 1995, Vol.21, Nº 4, pp. 356-372.

[14] Rosa, N., Justo, G., Cunha, P.: A Framework for Building Non-Functional Software Architectures. In *16th ACM Symposium on Applied Computing*, Las Vegas, USA, 2001. 16th ACM SAC. , 2001. p.141 – 147

[15] Shaw, M., Garlan, D.: *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, New Jersey, 1996.

[16] Siegel, J.: *CORBA Fundamental end Programming*. John Wiley & Sons, Inc.1996.

[17] Spitznagel, B., Garlan, D.: A Compositional Approach for Constructing Connectors. In *Proc. of the Working IEEE/IFIP Conference on Software Architecture* (WICSA'01), 2001.