

The TIDIA-Ae Portfolio Tool: a case study of its development following a component-based layered architecture

Delano M. Beder*

Department of Statistics, Applied Mathematics and Computing - São Paulo State University (Unesp- Rio Claro)
delano@rc.unesp.br

André C. Silva*

Nucleus of Informatics Applied to Education (Nied) - State University of Campinas (Unicamp)
andrecons@nied.unicamp.br

Heloísa Vieira da Rocha*

Institute of Computing (IC) & Nucleus of Informatics Applied to Education (Nied) - State University of Campinas (Unicamp)
heloisa@ic.unicamp.br

**e-Labora: e-Learning Laboratory - State University of Campinas (Unicamp)*

***LIA: Advanced Interaction Laboratory Computer Department- Federal University of São Carlos (UFSCar)*

****LAI: Learning and Interaction Laboratory - Technological Institute of Aeronautics (ITA-CTA)*

Joice Lee Otsuka*

Institute of Computing (IC) & Nucleus of Informatics Applied to Education (Nied) - State University of Campinas (Unicamp)
joice@ic.unicamp.br

Américo T. Neto**

Department of Computing - Federal University of São Carlos (UFSCar)
americo@dc.ufscar.br

Ivan L. M. Ricarte*

School of Electrical and Computer Engineering (FEEC) - State University of Campinas (Unicamp)
ricarte@fee.unicamp.br

Celmar G. Silva*

Institute of Computing (IC) & Nucleus of Informatics Applied to Education (Nied) - State University of Campinas (Unicamp)
celmar@ic.unicamp.br

Alessandro R. Oliveira***

Computer Science Department - Technological Institute of Aeronautics (ITA-CTA)
aoliveir@ita.br

Junia C. A. Silva**

Department of Computing - Federal University of São Carlos (UFSCar)
junia@dc.ufscar.br

Abstract

The objective of the present case study is the validation of the Architecture and Development Process definitions used within the TIDIA-Ae Project, as exemplified in the development of the Portfolio tool. In this way, the main results obtained in the development process of the Portfolio tool, the solutions adopted for the open issues encountered in this process, and the main lessons to be learned are presented.

1. Introduction

The TIDIA-Ae Project (Tecnologias da Informação para o Desenvolvimento da Internet Avançada – Aprendizagem Eletrônica - Information Technology for the Development of Advanced Internet – Electronic Learning)¹ was initiated by FAPESP (Fundação de

Apoio à Pesquisa do Estado de São Paulo - the State of São Paulo Research Foundation) with the main objective to develop an Electronic Learning System that can explore the potential of Advanced Internet.

Considering that the project aims to integrate various research groups, with different lines of research and different needs, for the use of a single electronic learning environment, the project is guided by the following main goals:

- To define a process of collaborative development among the different research groups;
- To promote interoperability with other electronic learning systems;
- To promote the reuse of tools and/or functionalities developed by TIDIA-Ae;
- To provide a flexible system that can be “assembled” from a set of selected tools and functionalities in order to satisfy various learning scenarios.

¹ <http://tidia-ae.incubadora.fapesp.br/portal>

In order to meet the above mentioned goals within the scope of the project, and more specifically within the scope of the Architecture Working Group of TIDIA-Ae, a component-based layered architecture [1] was proposed, which has guided the development of the tools that will integrate the TIDIA-Ae system. In this way, it is expected to obtain a system that can allow for the reuse of components and the integration of new functionalities, as well as the adaptation of existing tools and functionalities to the needs of the various research groups.

This article presents a case study of development according to the TIDIA-Ae Architecture, whose activity was the development of the TIDIA-Ae's Portfolio tool. This tool was developed by the development laboratory, e-Labora, in collaboration with the associated laboratories LIA and LAI.

Section 2 briefly presents the TIDIA-Ae architecture as well as the advantages offered by its use. Section 3 presents the main results and solutions adopted during the development of the Portfolio tool. Finally, section 4 presents the conclusions, highlighting the principal lessons learned.

2. The TIDIA-Ae Architecture

A software architecture encompasses the definition of its general structures, describing the logical components that make up the system and the connectors between them [2].

The software components specification is generally divided into two parts: one for the provided services and another for the required services. The services that the component offers are defined in its **provided interfaces** and the services that the component requires, in order to supply its own services, are defined in its **required interfaces**.

This separation augments the flexibility and adaptability of the component, since a given component knows only its own required interfaces, but not the components that implement them. This adaptation is accomplished by the connectors that recognize the interfaces of the various components and can make the adaptation from one interface to another.

This section presents the layered component-based software architecture adopted by the TIDIA-Ae project. This software architecture is composed of the following layers:

- **Presentation layer:** provides the application user interface. The separation between the presentation logic and business rules allows software to provide different user interfaces

(web, desktop, palm, etc) that could be personalized even for users with special needs;

- **System layer:** provides an interface for the application functionality, that is, it is a *façade* for the application business rules. Generally the components in this layer implement the application use cases, utilizing the components of the e-learning layer. The main objective in defining this layer was to reduce the coupling between the presentation and business layers;
- **E-Learning layer:** provides the component interfaces that implement the application's business rules, which can be used by various applications and which use services and functionalities from the infrastructure layer to implement the business rules;
- **Infrastructure layer:** implements a set of infrastructure services such as, for example, data persistence;
- **Common services layer:** corresponds to the layer which has the public services that can be utilized and accessed by all of the other architecture layers.

The advantages of using a component-based layered software architecture, and which justify its adoption, are related to the following characteristics that are obtained [5]:

- **Abstraction:** explicit separation between the specification and the implementation of functionalities. The details of the implementation of a functionality are hidden from the clients, thus facilitating the incorporation of changes or the inclusion of new functionalities with the minimum impact on the other parts of the system;
- **Uniform composition:** in order to assemble functionalities implemented by distinct components, there must be communication between these components. This communication should always be through components' provided and required interfaces, thereby providing uniform composition. In this way, there is the possibility of component reuse, developed by other educational institutions, bringing increased productivity and quality of the software produced.

The next section presents considerations on the development of the Portfolio tool, following the architecture defined by TIDIA-Ae.

3. The Portfolio Tool

The Portfolio is a communication tool that aims to promote the collaboration among participants through the sharing of “items” (documents, programs, links, etc.). In this way, the Portfolio tool provides an area for the storage and sharing of files from each participant (user or group of users) within a context of the TIDIA-Ae (which could be a course, a project, a subject, etc.), named Participant Portfolio.

When the user stores an item in his/her portfolio, or the portfolio of a group to which he/she belongs, the user can specify the type of access to the newly stored item (for example, not shared, shared only with users who have a particular role, shared with certain participants, shared with all participants). Users who have been given access to an item in a portfolio can access the item, download attached files and add comments.

3.1. The Development Process

The development of the Portfolio tool followed the software development process called UML Components[3], according to the definitions of the Architecture Working Group of the TIDIA-Ae Project. This process has the following characteristics:

- **Component-based:** allows an application to be constructed from the composition of software components, resulting in an increase in productivity and quality of the produced software;
- **Architecture Centered:** meaning that the system’s architecture is used as the main artifact to conceptualize, construct, manage, and evolve the system in development;
- **Use case driven:** meaning that use cases are utilized as the first artifact to establish the desired behavior of the system and to verify and validate the system architecture;
- **Iterative and incremental:** in which the system is developed by successive refinements. The essence of the interactive process is that system specification evolves as the software is being developed.

In this way, the development of the Portfolio tool involved the following macro activities:

- Portfolio Tool requirements and use cases Specification²;
- Portfolio Tool Conceptual Model and Type Model Specification;
- Software components identification and mapping to the defined architecture (section 3.2)³;
- Portfolio Tool implementation (section 3.3).

3.2. Portfolio Components

This section presents the results from the phase of mapping the software components of the Portfolio tool onto the architecture defined by the TIDIA-Ae project. Figure 1 presents the components responsible for the implementation of the functionalities of this tool distributed in the layers defined by the TIDIA-Ae architecture.

The following are the principal components of the Portfolio tool:

- **Presentation:** single component for the entire LMS TIDIA-Ae. This component present in the presentation layer will integrate functionalities of all tools that have been developed in the TIDIA-Ae Project;
- **PortfolioSystem:** component present in the system layer, responsible for the implementation of all use cases specified in the Portfolio Tool;
- **PortfolioMgr:** component present in the e-learning layer, responsible for functionalities related to the participant portfolio management in a learning context (for example, enable/disable participant portfolios in a context);
- **FolderMgr:** component present in the e-learning layer, responsible for functionalities related to the folder management (for example: create folder, move folder, rename folder, etc);
- **ItemMgr: component present in the e-learning layer,** responsible for functionalities related to the item management and the history of the actions occurring on an item;

² Documents on Requirements, Use Cases, and the Conceptual Model of the Portfolio are available in: http://tidia-ae.incubadora.fapesp.br/home/documents/internal_documents/worki ng_drafts/gt_IHC/Trabalhos_13-05-2005/Nucleo_Campinas/portfolio/

³ Specification of the Portfolio is available in http://tidia-ae.incubadora.fapesp.br/porta l/documents/internal_documents/worki ng_drafts/SG_arquitetura/Tools%20Specification/PortfolioTool/

- **FileAttachmentMgr**: component present in the e-learning layer, responsible for file attachment management;
- **AssessmentMgr**: component present in the e-learning layer, responsible for functionalities related to the Portfolio participation assessment management;
- **PortfolioDAO, FolderDAO, ItemDAO, AssessmentDAO**: components present in the infrastructure layer, responsible for the persistence of entities managed by the e-learning layer.

It is important to mention that although these components were initially specified and implemented to attend the Portfolio Tool requirements, they can be reused in the development of other tools. In this way, components such as **FolderMgr, ItemMgr** and **FileAttachmentMgr** can be shared for reuse in the development of tools that have requirements related to folder management, item management, and file attachment management respectively.

The adoption of a component-based layered architecture has facilitated actions such as the following:

- Alternative implementation and possible substitution of an entire layer without affecting the components in the other layers, for example:

- Substitution of the presentation layer, which is currently based upon web technology, for a different solution based on desktop or mobile technologies;
- Substitution of the infrastructure layer, which is currently based on relational database technology, for a different solution, for example, based on an object-oriented database technology.

- Substitution of one component for another, for example:

- Substitution of the solution implementation utilized for the persistence of system entities (for example, migrate from JDBC to Hibernate) without affecting the other components and/or layers;
- Substitution of the **FileAttachmentMgr** component, which currently uses a solution based on relational database technology, for a component that implements a solution based on file system technology;
- Substitution of the **FolderMgr** component, which currently implements a hierarchical folder structure, for a component that implements an alternative folder structure (non hierarchical, cyclic, etc).

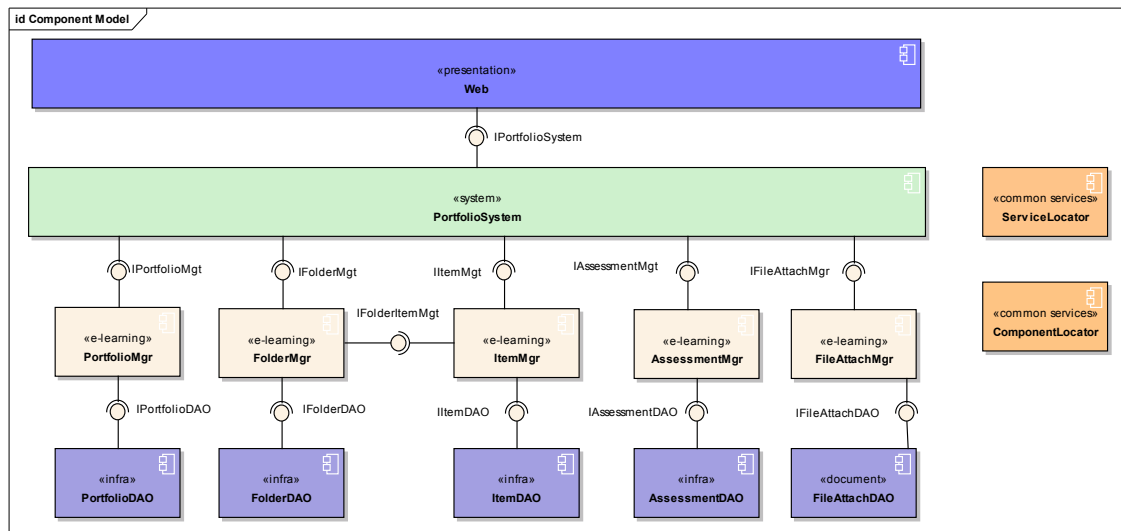


Figure 1 - Portfolio Components mapped to the TIDIA-Ae architecture

3.3. Implementation

In this section we present the technological solutions utilized in the implementation of the Portfolio tool. Following the project definitions, we used J2EE[6] as the development platform. In relation to this technology, we have the following decisions:

- JBoss [7] was adopted by the TIDIA-Ae Project as the main J2EE application server;
- We used Stateless Session Beans in the system and e-learning layers according to the decisions of the Architecture Group;
- The TIDIA-Ae project also decided to use Hibernate [8] as the persistence framework. Hibernate provides object-relational mapping, that is, it maps the entities managed by the software onto a relational database;
- The Campinas Nucleus opted to use Struts [9] as the framework for the development of Web applications, based upon the Model-View-Controller (MVC) standard. The use of JSF [10] for the implementation of MVC is being analyzed by the Architecture Group.
- The Campinas Nucleus also opted to use XDoclet [11] for the automatic generation of the interfaces for the J2EE components and for the JBoss and Hibernate configuration files.

Each component was mapped to the implementation level in a Java package. A Java package is defined as a general purpose mechanism to organize semantically related elements in groups. The architectural elements (component and connectors) were implemented by a set of classes and interfaces defined in the packages. Two sub-packages were defined – one sub-package for the component specification interface (**spec**), and the other for the component implementation (**impl**). The component implementation sub-package (**impl**) is made by the implementation interfaces (in the case of J2EE, the *home* and *remote* interfaces of *Session Bean*) and by the classes that implement the component and its connectors. In this way, component specification was separated from its implementation, and the encapsulation of the component implementation was guaranteed. Only the specification interface of a component (spec sub-package) is visible to its clients.

We utilized the following J2EE [12] design patterns:

- **Session Façade.** The *Session Façade* manages the business objects and provides an access layer which is uniform to the services and of coarse granularity for the clients [12]. In the Portfolio tool implementation, the component

of the system layer **PortfolioSystem** serves as a *façade* for the components from the e-learning layer, reducing the coupling between the presentation and e-learning layers.

- **Business Delegate.** The *Business Delegate* is a design pattern used to hide the implementation details related to the J2EE [12] technology, reducing the coupling between the system components. For example, a *session bean A* that needs to use the functionalities of another *session bean B* (in the same or a different layer) does not need to instantiate the *B home* and *remote* interfaces to access its functionalities. The *session bean A* just needs to instantiate a *Business Delegate*, that is responsible for the instantiation of the *B home* and *remote* interfaces.

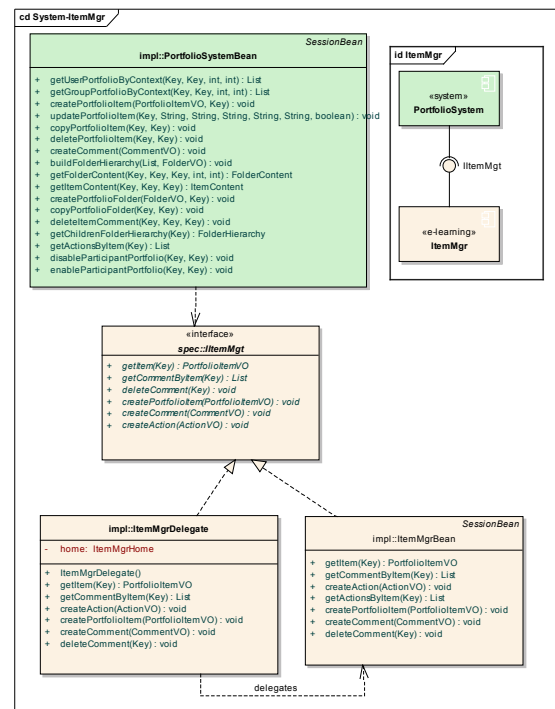


Figure 2. PortfolioSystem and ItemMgr components.

At Portfolio Tool implementation, the *Business Delegates* are the component connectors in the system and e-learning layers. Figure 2 illustrated the implementation of the **PortfolioSystem** and **ItemMgr** components. As presented previously, logical components and connectors were implemented by a set of classes and interfaces organized in packages.

PortfolioSystemBean is the *Session Bean* responsible for the implementation of functionalities provided by the **PortfolioSystem** component.

The interface **IItemMgt** is found in the **itemMgr.spec** package and is provided by the component **ItemMgr**. Two classes of implementation are found in the **itemMgr.impl** package: 1) **ItemMgrBean**, *Session Bean* responsible for the implementation of the functionalities of the **IItemMgt** interface; 2) **ItemMgrDelegate**, *Business Delegate* that serves as a connector to the functionalities provided by **ItemMgrBean** and that implements the **IItemMgt** interface.

- **Service Locator.** The *Service Locator* reduces the complexity of locating the J2EE components by hiding the details of the implementation of lookup and access to these components [12]. Various clients can utilize the same Service Locator as a centralized point for locating J2EE components. Therefore, in the implementation of the Portfolio tool, the Service Locator (Figure 1) was implemented as a service in the common services layer of the architecture.
- **Data Access Object.** *Data Access Object* (DAO) encapsulates all the accesses to the data source (for example, a relational database). The DAO manages the connection with the data source to obtain and store data. The components of the infrastructure layer (**PortfolioDAO**, **FolderDAO**, **ItemDAO**, **FileAttachDAO** and **AssessmentDAO**) are DAOs and were implemented using the Hibernate technology [8].

3.3.1 Solutions adopted for open issues

At the beginning of the implementation of the Portfolio tool, certain issues were still unresolved and, therefore, some decisions needed to be made in order to solve/respond to these open issues. This section presents and discusses the solutions adopted by the Campinas Nucleus, under the coordination of the development laboratory e-Labora, in order to try to address the open issues:

- **How will the software components be instantiated?**

In the Portfolio tool implementation, there is a component/service (*ComponentLocator*) in the common services layer that is responsible for the instantiation of the other system components. This component uses the *nanocontainer*⁴ (*Dependency Injection Container*) for the instantiation of the system components. The main reason to use the *ComponentLocator* is to hide the details of the implementation of a component, since each component is only instantiated by its provided interfaces. That is, so long as it is desirable to have an explicit separation between the specification and the implementation of a functionality, a client solicits the instantiation of a given provided interface, irrespective of the component that implements the functionality.

The *nanocontainer* needs a configuration file in XML where the relationships between the provided interfaces and the respective classes that implement these interfaces are described (Figure 3). It is important to note that this XML file is the only location that describes these relationships.

However, it is not possible for the *nanocontainer* to instantiate *Session Beans*. In order to overcome this obstacle, the complementary solution adopted consists of making the container instantiate the *Business Delegates* that are responsible for the access to the functionalities implemented by the *Session Beans*. In this way, when the instantiation of a component is requested (through its provided interface) the *nanocontainer* instantiates the *Business Delegate* which implements the requested provided interface, and this is responsible for requesting the execution of the functionality by the component. For example, according to the configuration file presented in Figure 3, when an instantiation of the **IItemMgt** interface is requested, the *container* will instantiate the class **ItemMgrDelegate**, which is the *Business Delegate* for the functionalities of the *Session Bean* **ItemMgr**.

In this way, the *nanocontainer* functionality that was used in the implementation of the *ComponentLocator* was the instantiation of the components according to the XML configuration. Future evolutions of the *nanocontainer* could implement the instantiation of the *Session Beans*, eliminating the need to use the *Business Delegate* as part of the solution for the component instantiation. Another alternative could be the implementation of a Dependency Injection mechanism for the instantiation of EJBs (through the use of *annotations*, for example), as anticipated in the EJB 3.0 specification [13].

⁴ <http://www.nanocontainer.org/>

```

<!-- system layer components -->
<component-implementation
interface='portfolio.spec.IPortfolioSystem'
class='portfolio.impl.PortfolioSystemDelegate'
/>
<!-- e-learning layer components -->
<component-implementation
interface='portfolio.itemmgr.spec.IItemMgt'
class='portfolio.itemmgr.impl.ItemMgrDelegate'
/>

```

Figure 3 – Nanocontainer configuration

We believe that with this project solution the developer can use a component without being concerned with the details of its implementation. In addition, the update and/or exchange of components will not impact the other components in the system and is hidden to them. This is the case because the instantiation of components is executed through the *ComponentLocator* and *nanocontainer*, which, together with the *Business Delegates*, hide the details of the component implementation (for example, whether it utilizes the J2EE or .NET technology). Finally, the exchange of one component for another component that implements at least the same functionalities (provided interface) can be executed at any moment. One only needs to specify in the configuration file of the *nanocontainer* that the provided interface is now implemented by a different class. This class will be instantiated by the container the next time the component instantiation is requested through its provided interface. This solution provides a high degree of system adaptability, as it is unnecessary to recompile the code responsible for the instantiation of components, as is the case with solutions based on the *Abstract Factory* [14] design pattern. On the other hand, the application of *Abstract Factory* has the advantage of being an implementation that uses only standardized resources from the Java language, such as Properties files and Reflection. In this way, we believe that an even more in depth study is needed to identify the best solution for the issue raised.

- **How can the uncoupling of components be guaranteed? How will the connectors between components be implemented?**

In our implementation, the uncoupling is guaranteed by the utilization of the *Business Delegates* in the implementation of the connectors between the components in the system and by using the *ComponentLocator* in the instantiation of our components. In this way, the details of the implementation of a given component are not visible to the other components.

The solutions proposed in this article to open issues, and implemented in the Portfolio tool, are generic and can be used in other tools. We believe that these and other open issues impact the integration of the tools that will compose the TIDIA-Ae environment and, therefore, need to be discussed by the project participants in order to obtain a consensus that facilitates the integration of the tools developed in the project.

3.3.2 Integration of the Portfolio with other TIDIA-Ae tools

In general, the solutions adopted in the development of the Portfolio tools presented above facilitated the process of integration with the TIDIA-Ae management tools and also the iterative and incremental development of the Portfolio tool. This conclusion was obtained by verifying that the solution adopted for the instantiation of the software components allowed for the creation of new functionalities and the maintenance of those already implemented, while the integration process was in progress.

In order to allow for this integration and evolution of the Portfolio tool to happen in parallel, some of the classes were developed to simulate the management tools in the environment, returning expected information. This allowed for the maintenance of the tool while it was being integrated. It is important to highlight that the management tools in the environment were also developed iteratively and incrementally; in other words, the first versions developed did not attend all of the demands of the Portfolio tool. Nevertheless, the solutions adopted allowed for the ongoing development of the Portfolio tool. A quick modification of the *nanocontainer's* configuration file permitted the tool to use, at one moment, the version of the management tools (to verify the integration) and, at another moment, the classes that were elaborated for their simulation (to verify the modifications or new tool functionalities).

There were a few obstacles related to the integration due to changes in the provided interfaces by the management tools and that were required by the Portfolio tool. Therefore, it is important that the provided and required interfaces of the components in the project be rigorously defined, in order to avoid spending resources on adjusting code or on the implementation of *adapters* [14] between tools. This does not imply that the interfaces should be immutable, but that each modification made should be strongly based in the artifacts elaborated during the

development process (such as documents of use cases and of requirements).

4. Conclusions

The adoption of a component-based software paradigm in a project like the TIDIA-Ae adds complexity to the development of the system since the specification of the components, and the relations between them, need to be specified, taking into consideration the architecture defined by the project. Nevertheless, the importance of the paradigm is reflected in the flexibility and in the reutilization provided. As an example, it is intended to utilize the components developed for the Portfolio tool to derive other tools that involve text and file storage functionalities, as a tool to store support materials from a learning context (like the TelEduc's Support Material Tool [15]) and a tool to organize the Learning Activities within a context.

The development of the Portfolio tool was an important experience for the participating laboratories since it made possible the development of human resources in the adopted technologies, in addition to the validation, not only of the architecture proposed by the TIDIA-Ae Architecture Group, but of the technological solutions adopted. We can highlight a few of the lessons learned in this process:

- **Need for the development of human resources:** the solutions adopted in this project involve cutting edge technologies and, therefore, the investment in the development of human resources is fundamental;
- **Need to attend the decisions of the project:** in a collaborative development project it is essential that there are well defined policies and that these are followed by all of the collaborators. The principal definitions of the project which guided the Nucleus's work was the Architecture defined by TIDIA-Ae and the development process based on UML- Components;
- **Need to attend the premises of development based on components:** during the development of the Portfolio tool, special care was taken to search for development solutions that would make possible the implementation of the characteristics inherent to development based on components. Noteworthy among these characteristics are the uniform separation and composition (discussed in section 2) through the definition of the sub-packages **impl** and **spec** for each component, and

of the use of *Business Delegates* and of the *nanocointainer*;

- **Need for well defined interfaces:** in a project with the size of TIDIA-Ae, it becomes essential that the component interfaces to be developed are rigorously specified before implementation of the components and, primarily, before they are published;
- **Need for tools to analyze the developed code:** The verification of the adequacy of the developed code to the project decisions is an arduous task that, in certain cases, can be facilitated by the development of small code analysis tools. For example, verification that the defined architecture is being respected is a task that can be automated by a script in Perl⁵;
- **Need for collaboration among the research groups:** the development of all of the specification artifacts for the Portfolio tool followed a dynamic of weekly revision by pairs (among the TIDIA-Ae research groups). This process was important for the sharing of experience among the groups and for the maturation of the produced artifacts. In the case of the Portfolio tool, in addition to the artifacts anticipated in the UML-Components process, prototypes were developed of the user interface that were analyzed by the team, allowing its members to contribute more effectively to the functionalities and to the tool's user interface;
- **Need to define stages for the development of the user interface, including the elaboration of prototypes:** since the TIDIA-Ae project is composed of a multidisciplinary team, it is necessary that the development process makes possible the most effective participation of its members who are not specialists in computer science. We adopted the development of a user interface prototype (that mirrors, and makes evident to the user, the requirements, use cases and base conceptual model) in order to promote the collaboration of these participants.

The development of the Portfolio tool was part of the first cycle of tool development in the TIDIA-Ae project. The considerations presented above are intended to contribute to the refinement of the process of development and collaboration in the subsequent phases of the project.

⁵ http://tidia-ae.incubadora.fapesp.br/portal/laboratories/e-laboratorio-unicamp/workingdrafts/GT1/2005-09-13_gtARQ_CPS_eLabora_ScriptDeAnaliseDeCodigo_v0-1-0.zip

5. References

- [1] D. M. Beder, L.M.Casagrande, C.M.F.Rubira. Proposta de Arquitetura de Software para o Projeto TIDIA – Ae. Nov 2004. Available at: <http://tidia-ae.incubadora.fapesp.br/portal/laboratories/e-laboraunicamp/workingdrafts/GT1/>
- [2] M. Shaw & D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
- [3] J. Cheesman & J. Daniels. UML Components: A Simple Process for Specifying Component-based Software, Addison-Wesley, 2001.
- [4] M. Silva Júnior. COSMOS - Um Modelo de Estruturação de Componentes para Sistemas Orientados a Objetos, Master's Dissertation, Instituto de Computação, UNICAMP, 2003.
- [5] R. Kazman & L. Bass. Towards Deriving Software Architectures From Quality Attributes. Technical Report CMU/SEI94TR10, Software Engineering Institute, 1994.
- [6] E. Armstrong, D. Green, K. Haase et al. J2EE Tutorial, Addison-Wesley, 2004.
- [7] T. Marrs, S. Davis. JBoss at Work : A Practical Guide, O'Reilly; 1 edition, 2005.
- [8] C. Bauer & G. King. Hiberante in Action. Manning Publications Co. 2004.
- [9] T.N. Husted, C. Dumoulin, G. Franciscus & D. Winterfeldt. Struts in Action. Manning Publications Co. 2002.
- [10] D.Geary, C. Horstmann. Core JavaServer Faces. Prentice Hall PTR , 2004.
- [11] C.Walls, N. Richards. XDoclet in Action. Manning Publications, 2003.
- [12] J. Crupi, D. Malks & D. Alur. Core J2EE Patterns – Best Practices and Design Strategies, Prentice-Hall, 2003.
- [13] JSR 220: Enterprise JavaBeans™ 3.0. <http://jcp.org/aboutJava/communityprocess/edr/jsr220/index.html>
- [14] E. Gamma et al., Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
- [15] H.V. Rocha *et al.* "Projeto TelEduc: Pesquisa e Desenvolvimento de Tecnologia para Educação a Distância", In: IX Congresso Internacional de Educação a Distância da ABED. September, 2002