An MDA Domain Specific Architecture to Provide Interoperability Among Collaborative Environments

Rita Suzana Pitangueira Maciel[1,2], Carlos Guimarães Ferraz[1], Nelson Souto Rosa[1]

[1]Universidade Federal de Pernambuco, Centro de Informática
50732-970 Recife, Pernambuco, Brazil

[2]Faculdade Ruy Barbosa
41940-320 Salvador, Bahia, Brazil

e-mail {rspm, cagf, nsr@cin.ufpe.br},{rsuzana@frb.br}

Abstract. *A Domain Specific Architecture (DSA) is generic architecture for a family of application system, a problem or a task area. Middleware specific services are tailored to the requirements of particular domain. Through a development process that uses the Object Management Group Model-Driven Architecture (MDA), this paper presents the InterDOC, a domain-specific architecture that includes specific middleware services to provide interoperability in the collaborative authoring domain. The MDA's UML profile Enterprise Distributed Object Computing (EDOC) is used to achieve a platform-independent specification that could be mapped into various specific environments, and to domain-conceptual modelling.*

Resumo. *Uma Arquitetura para Domínio Específico é uma arquitetura genérica para uma família de aplicações, um problema ou uma determinada área. Serviços específicos de middleware satisfazem os requisitos de um determinado domínio. Através de um processo de desenvolvimento que utiliza MDA(Model-Driven Architecture) da OMG(Object Management Group), este artigo apresenta uma arquitetura para domínio específico baseada em serviços de middleware para prover a interoperabilidade no domínio da autoria colaborativa. O perfil UML EDOC(Enterprise Distributed Object Computing) é utilizado para obter uma especificação independente de plataforma que pode ser mapeada para vários ambientes específicos, e para a modelagem conceitual.*

1. Introduction

The number of tools to support software development has grown a lot in the last decade. However, the design and implementation of complex applications still continue to be expensive and the final users' satisfaction is still low compared to the accomplishment of the specified functional requirements. Much of the cost and effort resides in the continuous process of rediscovering and reinvention of core concepts and key components across the software industry. In other words, to each new application to be developed, little is reused in relation to the experience of the development of applications of the same domain (Fayad, 1999).

Several approaches for software development are being proposed to capture a domain and to reuse in future developments. Domain-specific architectures (Taylor, 1995) and specific middleware services (Schmidt, 2001) are some approaches proposed to facilitate the development of applications of a same domain. The modeling of the stable aspects of the domain is an important and fundamental aspect of these approaches.

Modeling is an essential and crucial issue in software construction. However, in most of software development methodologies, models are only used in the initial phases of the development process. Several models became outdated due to software evolution and maintenance. Another issue is that, as the technology becomes more and more complex, modeling is increasingly necessary in software development in order to be productive (Broy, 2004).

In Model-Driven Engineering (MDE) (Guelfi, 2003), models are not only used as software documentation, but also as a building tool. They became the development process backbone. Each development process phase takes a number of input models that produce some output models. Hence, the process of building an application can be seen as a series of model transformations until the final system. The Model-Driven Software Development (MDSD) (Frankel, 2004) enables reuse at the domain level, and potentially increases quality as models are successively improved, reduces costs when using an automated process, and increases software longevity. The aim is that models become assets instead of expenses (Mellor, 2004).

One of the most known initiatives in this scenario is the Model-Driven Architecture (MDA) (OMG, 2003), which is an approach to system specification and interoperability based on UML models. MDA proposes the construction of a given application based on generic models. Specific implementations should follow such a model. The construction of a distributed application based on MDA starts with the definition of a middleware-independent model called PIM (*Platform Independent Model*). Then, a model named PSM (*Platform Specific Model*) is defined according to the middleware the application will be implemented on (e.g., CORBA, CORBA/CCM, EJB, DCOM).

Although collaborative authoring is a much-studied domain, its supporting tools are not usually interoperable. Consequently, authors have to use a single environment to obtain the benefits of computer-supported cooperation. This aspect is extremely limiting, as authors need to adapt to new aspects of computer-aided collaboration while she/he have to learn new mechanisms (e.g., editors, e-mail tools, interfaces) that may not be familiar to them (Noel, 2004). In some situations, due to the huge effort to adapt to a new environment, organizations discard the use of cooperative work supporting environments in the authoring task.

The existence of a vast diversity of collaborative authoring tools makes the implementation of interoperability an expensive task, due to both (1) the heterogeneity of hardware and software platforms that host them and (2) the number of specific solutions (frameworks, protocols, APIs, components, etc) that some environments make available to enable this requirement (Li, 2003; Tietze, 2001). In addition, these solutions create platform- or language-dependent environments because they take domain information into little consideration, i.e., they are very much related to low-level issues such as windows system output, operating systems etc. A specific development and implementation process could be necessary for each new collaborative authoring

environment, in order to provide interoperability between them. In that case, the cost of the development to provide such interoperability becomes very high.

This paper presents the domain-specific architecture (DSA) of InterDOC (Environment for Supporting Interoperability among Collaborative Document Authoring Tools). This architecture defines domain-specific middleware services to provide interoperability in the collaborative authoring domain. To attain a generic and platform-independent model, the reference architecture is specified as an MDA PIM. The models of the InterDOC architecture are described based on MDA's UML Profile for Enterprise Distributed Object Computing (EDOC) (OMG 2002), using the concepts *of* viewpoints defined in RM-ODP (Open Distributed Processing-Reference Model) (ISO, 2005).

The rest of this paper is organized as follows: next section introduces the DSA development process. Section 3 gives an overview of the Collaborative Authoring. Section 4 MDA views of the InterDOC reference architecture (PIM). A platform-specific model of InterDOC and its practical experiences is given in Section 5. In the last section, we summarize the contribution this work together some steps for further developments.

2. The DSA Development Process

Several proposals use MDA to model a specific domain (Wang, 2003; Guelfi, 2003a) Even though they use MDA, they do not use the OMG profiles like EDOC, EAI (Enterprise Application Integration, OMG), etc. The use of the profiles proposed by the OMG guarantees, in a more extensive fashion, the interoperability of MDA-based models. Methodologies and tools that assist in the development of applications and use EDOC or other sub-profiles proposed by the OMG will have the same conceptual and notational framework. This aspect facilitates the reading of models both by development teams and by process automation tools.

The DSA is specified as an MDA PIM and EDOC is the profile used. EDOC profile is based on UML 1.4, and its objective is to simplify the development of component systems. EDOC presents a modelling framework that provides a platform-independent and recursive collaboration based modelling approach that can be used at different levels of granularity and different degrees of coupling, for both business and systems modelling. A set of sub-profiles composed EDOC profile. The Business, Entities and Component Collaboration Architecture (CCA) were the sub-profiles chosen to compose our development process.

EDOC optionally proposes the use of the concept of viewpoint defined in the RM-ODP to achieve a specification that addresses a number of distinct sets of concerns (e.g. system behaviour, structure, semantics and infrastructure). RM-ODP divides a system into five viewpoint specifications, namely Enterprise, Information, Computational, Engineering and Technology views. The Enterprise view is concerned with the business activities of the specified domain. It describes the purpose, scope and policies of the domain where the specific middleware service should be used. The Information view defines the information that needs to be stored and processed in the middleware services. Computational View is concerned with the description of the system as a set of objects that interacts at interfaces enabling system distribution. This view describes the service's interfaces and the components that provide the service's functionalities. The Engineering view is a viewpoint on the domain and its environment that focuses on the mechanisms and functions required to support distributed interaction

between objects in the system. The Technology View focuses on the system and its environment that focuses on the choice of technology in that system (ISO 1995).

Although EDOC and RM-ODP framework offer a set of tools that permit the specification of a system based on high abstraction models, they do not offer a guide to orient developers in the application of these concepts. Together they provide a set of definitions and notations, but not a process with well-defined steps to be followed (Gervais, 2002). In order to guide the InterDoc development, a development process was defined (Maciel, 2005). Three categories of model should be specified: Domain, Design and Operational (Figure 1).

The Domain Model corresponds to the context (collaborative authoring domain) in which the service should be applied. The scope and responsibilities of the application are defined in this model via the functional requirements initially established for the application. The services to be offer and the information handled by the application should be defined in this model. The Design Model describes the Computational view for the specified Domain model. This model identifies the service's interface and the components that provide the requirements established in the domain model, independently of the platform. The non-functional requirements of the application should be observed on this occasion to identify elements for the performance thereof. The Domain and Design models form PIM as described in MDA.

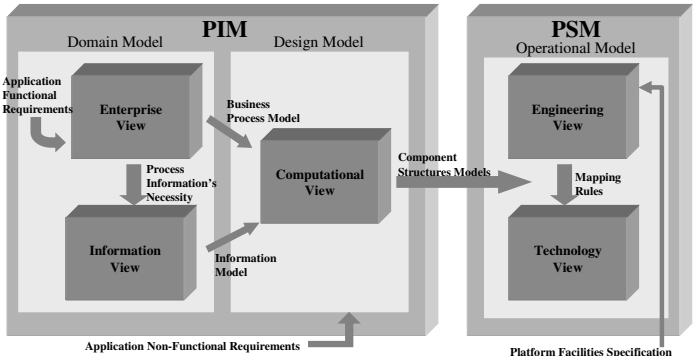

Figure 1. The Development Process Model (Maciel, 2005)

The Operational Model describes the application execution environment on a specific platform and corresponds to the specification of Engineering and Technology Views. The characteristics of the implementation platform chosen are considered in this model to reflect the real execution environment. The Operational Model form PSM as described in MDA. Our current InterDOC version is implemented in *CORBA Component Model* (CCM), so the PSM reflects this choice. PSM will be detailed in Section 4.

Our development process uses a subset of EDOC sub-profiles meta-models. The application's objectives, politics and restrictions specification, that is part of the Enterprise view concepts, should be specified. EDOC doesn't provide mechanisms to represent enterprise community behaviour at highest levels of abstraction. Hence, we use a set of UML use case diagram proposed in (ISO 2004), which identify the possible process to model this aspect.

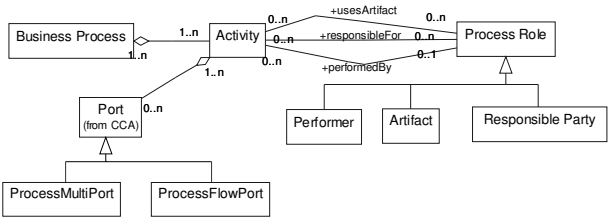

Figure 2.Business Profile Meta-model Subset.

The Business Process Profile is the EDOC sub-profile that should be used to model the Enterprise view (Figure 2). This sub-profile permits design models that present the structure and behaviour of the application in the environment where it is inserted. The domains functionalities are described in terms of *Business Process* (BP) that specifies a complete business task. A BP may contain *Activities,* which are the pieces of work required to complete a task. Data flows connect different BPs, and *Activities* in a BP, defining temporal and data dependencies between these elements. These data flows are modelled as Port. A *ProcessFlow*Port represents data used in Activity input/output. A *ProcessMultiPort* represents a set of related data. A *ProcessRole* defines a placeholder for behaviour in a context. Performer, *Artefacts* and *ResponsibilityPart*y extends ProcessRole. Performer is specifically for identifying an entity that performer the Activity to which it is associated. *Artefacts* are an entity that is need by an Activity as a resource. *ResponsabiltyParty* is an entity that has responsibility for the Activity to which it is associated.

The Entities Profile is the EDOC sub-profile that should be used to model the Information View. A central concept of the Entities Profile is an *EntityData.* An *EntityData* is a structure of data that represents one definitive concept of the application domain. An *EntityData* is equivalent to an entity or a relation in the relational model. *CompositeData* is a primitive data type (e.g. integer, string, char,etc) composed of others types in the form of attributes.

Component Collaboration Architecture (CCA) is the EDOC sub-profile that should be used for model Computational View (Figure 3).

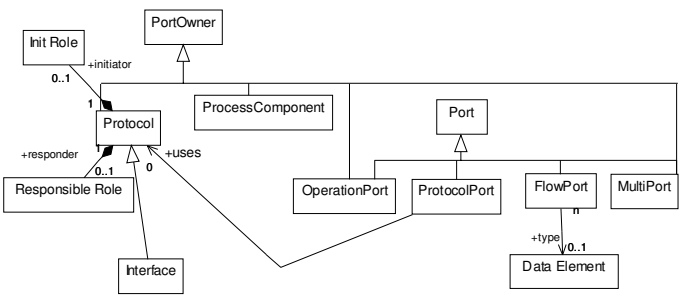

Figure 3 CCA Profile Subset.

PortOwner is an abstract metaclass used to group the meta-class that own ports. *ProcessComponents* and *Protocol*. A *Process Components* (PC) represents an active

processing unit that has a group of ports to interact with another PC. Each component is detailed in terms of ports and its protocols. A *Protocol* specifies the messages the component sends and receives in the collaboration with another component. A port defines a point of interaction between *ProcessComponents*.

A *Port* is a generalization of *FlowPort*, *ProtocolPort* and *OperationPort*. The simpler form of port is the *FlowPort,* which may produce or consume a single data type. A *FlowPort* defines a data flow in or out on the behalf of the component or protocol. More complex interactions between components use a *ProtocolPort*, which refers and define the use of a *Protocol*, a complete conversation between components. Since a Protocol has two roles (initiator and responder), the direction is used to determine which role the *ProtocolPort* is taken on. *OperationPorts* represents request-reply semantics.

An *Interface* is a *Protocol* specialization that represents a standard object interface. Each *OperationPort* or *FlowPort* in the *Interface* will map into a method. A *ProtocolPort* that initiates the *Interface* will call the interface. A *ProtocolPort* that responds will implement the interface.

The development process also suggests a set of diagrams that should be made to specify the RM-ODP views and consequently the proposed models (Table I).

Table I – The Development Process Diagrams

Enterprise View	Standard UML and *Business Profile*
Objective Definition	Use Case Diagram
Main Policies and Restriction	Text
Business Process Composition	Class Diagram
Activity Composition	Class Diagram
Activity Port Composition	Class Diagram
Business Process and Activity behavior	Activity Diagram
Information View	Entity Profile
Entities Data Definition	Class Diagram
Composite Data Definition	Class Diagram
Computational View	CCA Profile
Architecture Structure	Class Diagram
Component Structure	Class Diagram
Protocol Specification	Class Diagram
Protocol Structure	Class Diagram
Protocol Description	Sequence or Collaboration Diagram
Protocol Choreography	Activity Diagram

3. InterDOC: Interoperating Heterogeneous Collaborative Writing Environments

Several studies reveal that people write in a collaborative way because it is more productive. Among the advantages of collaborative writing there are getting several viewpoints, getting different expertise, reducing error and obtaining a better, more accurate text. The disadvantages are related to the management of the task that involves a group (Kim, 2001). Usually in a group a leader is revealed, that initially accomplishes a fast task planning. In this planning are defined the group's memberships, their roles (reviser, comment, etc), document's format, document control (centralized, relay, shared, etc) and even a writing strategy (single writer, separate writer, joint writing,

scribe, etc). In most of the processes all these aspects can be modified along the production of the text.

The collaborative authoring process is normally divided into the planning, authoring, proofreading, edition and closing phases among others. Due to the particularities of the phases that constitute this process, supporting the entire collaborative document creation process has been recognized as a complex task (Noel, 2004). Tools support either specific phases or tasks of collaborative authoring. For applications such as CSCW (*Computer Supported Cooperative Work*) and CSCL (*Computer Supported Collaborative Learning*), collaborative authoring is a secondary functionality among several others performed by the application. Nevertheless, the collaborative authoring of documents is the main activity in applications like cooperative editors. Applications that support any phase of the collaborative authoring process will be referred to CASA (Collaborative Authoring Supporting Application).

Due to their quantity and diversity, CASAs are not usually interoperable. Hence, authors have to make a choice: to use one or several environments to write documents. By using a single environment, authors opt to obtain the benefits of computer-supported cooperation. However, this choice is extremely limiting, as authors need to adapt to new aspects of computer-aided collaboration while she/he have to learn new mechanisms (e.g., editors, e-mail tools, interfaces) that may not be familiar to them (Noel, 2004). By using several environments, all the context information (e.g. groups, author's roles, permissions, shared workspaces, realized activities, etc) has to be managed without an automated support, or inserted several and repeated times in each different environment.

Researches reveal that in the majority of collaborative authoring process (75%), an author writes the initial draft of the document, and the collaboration actually starts in the subsequent phases when other authors revise, make notes, make comments and/or edit the document (Kim, 2001). The collaboration effectively starts in the asynchronous activities of a document construction. Hence, the use of groupware tools is largely asynchronous (Steves, 2001).

InterDOC allows various asynchronous activities of the authoring process to be executed in different tools. InterDOC should be an intermediary layer positioned between the CASAs and their repositories, promoting interoperability among various environments. Groups of authors can hold synchronous or asynchronous sessions for planning, drafting, revision and editing documents in their favourite environments. In any phase of the authoring process, InterDOC should support the availability of documents to any group of authors that use distinct environments, or even an individual work tool.

Figure 4 shows a plan for the InterDOC model of use. Each domain (Domain 1 and Domain 2) represents a different community and may adopt a specific protocol. These communities are interconnected through a long-distance network. Five layers made up the model of use: CASA applications, Interaction with Applications, Services, Communication between Domains and Interaction with Repositories layer.

Each domain has its CASAs (Client A, B, C and D applications) that are not interoperable in terms of collaborative authoring tasks. In order to allow their interoperation, applications may interface to InterDOC (interfaces for registering/searching and for document retrieval and availability). The type A/C client-application illustrates the category of applications that have been developed according to the InterDOC reference model. The type B/D client-application illustrates the

category of legacy applications, where the interface components were subsequently developed and where external calls are necessary for communication with InterDOC.

The layer of Interaction with Applications is formed by components that implement the interfaces between the applications and InterDOC, for provisioning documents and information relating to the authoring tasks. The Service layer consists of InterDOC objects that implement functional requirements of InterDOC. InterDOC provides functionalities for controlling access to documents, supporting definition of authors and group information, supporting the authoring activity process and notifying authors of activities to be executed. The layer of Communication between Domains consists of elements that are responsible for communication between InterDOC instances hosted in different domains. When necessary, messages will be formatted for the interoperability protocol used by the domain (e.g. SOAP (Simple Object Access Protocol) and GIOP (General Inter-Orb Protocol)**.**

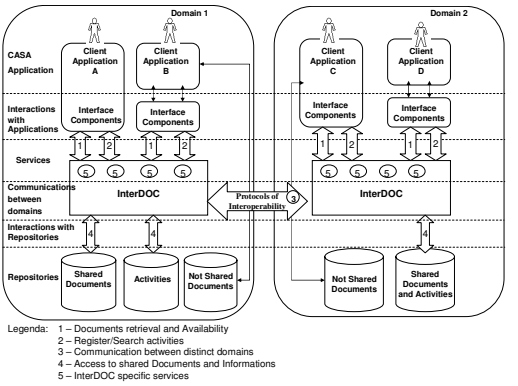

Figure 4. InterDOC Model of use (Maciel, 2004).

The layer of Interaction with Repositories has components for the interface with the repositories that are responsible for the validation and provisioning of information to be stored.

For InterDOC, collaborative authoring is divided into two main phases: Planning and Writing. The Planning phase refers to the time an author of the group provides basic information for the description of the authoring project of a document, e.g., group of authors, roles of authors in the group, location of shared repository and so on. The Writing phase is when authors make versions of the document available to other authors to perform activities on them.

4. InterDOC: The Domain Specific Architecture

This section presents DSA through some diagrams proposed by our approach. As an MDA PIM, the Domain and Project Model comprise the DSA.

4.1 Domain Model

The Enterprise and Information Views compose the Domain Model. The Enterprise view describes interactions among different InterDOCs and interactions between the

applications, repositories and the InterDOCs. The Information provides the description of objects that InterDOC handles.

4.1.1 Enterprise View

In this view the communities must be identified, as well their behaviour and the main constraints.

InterDOC has a single community type: the Authorship community. The Authorship community involves the authors and the applications used for writing documents. The main InterDOC's constraints refer to the community that registers a project to be the same that hosts the repository of documents for the group.

Based on several studies that propose collaborative authoring requirements (Cerrato, 1999; Kim, 2001; Noel, 2004; Ribeiro, 2004) as well several CASAs requirements, ten business-oriented objectives, representing the community behaviour, were identified: Create a Project, Define a Group, Define Author, Define Role, Define Activity, Delegate an Activity, Register an Activity, Notifies Authors, Retrieve Documents, Communication with other Domains.

Two scenarios, Design a Project and Writing, which specify each InterDOC phase (situations described in Section 3) was identified, and each business-oriented objective was modelled as a use case. Each use case represents one process, and focuses on a particular slice of the behaviour of the system being modelled, related to a particular scenario. The use cases: Create a Project, Define a Group, Define Author, Define Role, Define Activity belong to the Design a Project scenario.

The functionalities that InterDOC realized are described in terms of *Business Process*. The scenarios were mapped in two BPs: *Design a Project* and *Writing* (Figure 5). A BP may contain *Activities,* which are the pieces of work required to complete a task. An *Activity* may be composed by other activities or by input Data flows that are necessary to realize an activity, and output Data flows that is generate through the activity. This Data Flows are modelled as *ProcessFlow* ports.

The InterDOC BP and Activities are decomposed up all the objectives specified in the Enterprise View (use cases) were reached. InterDOC only possesses three levels of decomposition: (1) *Business Process* into *Activity*, (2) *Activity* into others *Activity* and (3) *Activity* into P*rocessFlowPorts*.

Figure 5 presents the first level decomposition, the Business Process Composition diagram of *Design a Project* and *Writing*. *Form a Group_Act* and *Register a Project_Act Activity* composes *Design a Project* BP. *Delegate Activity_Act*, *Execute Delegate Activity_Act* and *Notify_Act* composes *Writing* BP.

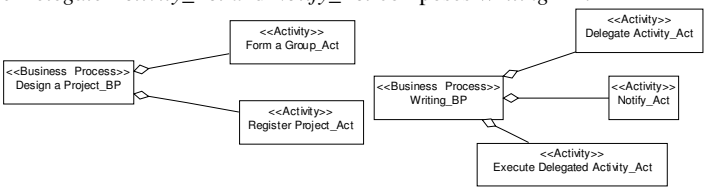

Figure 5. Design a Project and Writing Business Process Composition – Fist level.

Figure 6(a) presents the *Form a Group_Act* composition into three others Activities: *Search_Author Act, Search_Role_Act* and *Create_Group_Act.* Finally,

Figure 6(b) presents an example of the decomposition last level: the Activity Port Composition Diagram of *Create_Group_Act*.

Create a Group Activity has four *ProcessFlowPort*. As input, the activity needs Author and Role information's. As output, it generates the group information and identification.

Activity diagrams describe the dynamic aspects of the *Business Process* and *Activities* (out of scope of this paper). It shows a sequence of steps that should be accomplished for the execution of an activity.

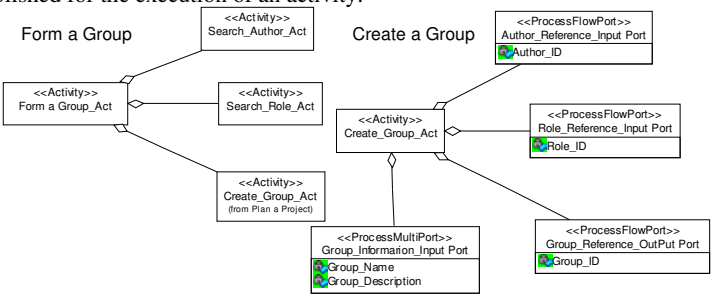

Figure 6 (a)Form a Group Activity Composition Diagram-Second Level.(b) Create a Group Activity Port Composition Diagram – Third Level.

4.1.2 Information View

The Information view models entities and their relationships. Entities represent concepts of the problem domain. The resulting models present the structure of used objects that represents the business concepts of the domain in the computational environment.

Entitydata and their relationships should be specified through a class diagram. InterDOC proposes the following *Entitydata*: Project (a certain authorship project); Repository (storage of project information); File (file stored in the repository); Document and Comments (specializations of File to represent documents and their respective comments); Activity (activities that the authors accomplish in a document); Author (user-author); and Group (authors' group and Role).

A project has a repository of associated files and belongs to a certain group of authors. Authors can play different roles in different projects. The role associates certain actions to them. Actions are registered in the environment. After revising a document, an author can save in the repository a new version of it. Authors revise documents producing comments (or annotations) or versions of these documents. Therefore, a document can be a version of another one, and a comment is associated to a document.

4.1.3 Computational View

The Computational view is a viewpoint on the system and its environments that enable distribution through functional decomposition of the system into objects that interact at interfaces. The component's structure, as well the interfaces and the data that these components handle, are described in the models of this view.

As DSA defines middleware specific-services, the first decision to be taken, before computational view modelling process, is the definition of which services should be offered to the CASA applications. Based, on the Enterprise View, all Activity that composes the Business Process (second level activities) is mapped into a middleware

service. Therefore, InterDOC has five services Form a Group, Register Project, Notification, Delegate Activity and Execute Delegate Activity services.

Figure 7 shows an overview of the InterDOC's component structure model in terms of *Process Components* (PC).Each PC will execute one or more *services* identified. Initially four PC was identified:

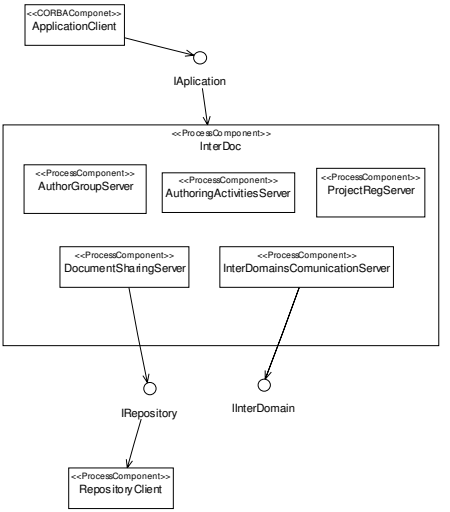

Figure 7 InterDoc Component Structure.

- *ProjectManagerServer*: perform register a project service;
- *AuthoringActivitiesServer*: perform the execute and delegate activities services;
- *ActivitiesNotificationServer:* perform the notification service;
- *AuthorGroupService:* perform the form a group service;

Later, two others PC, that perform InterDOC non-functional requirements, were identified:

- *InterDomainCommunicationServer*: it is a wrapper service component that formats messages for the interoperability protocol used by the domain;
- *DocumentSharingServer*: it manages the access to the information that is stored in the repositories.

Further, the InterDOC reference architecture defines two client-side components that use its services: the CASA components *(ApplicationClient ProcessComponent*) and the Repository component (*RepositoryClient ProcessComponent*).

In InterDOC, *Interface* defines the interactions between InterDOC components and external elements (CASA, Repository and another InteDOCs). *Protoco*l was used for interactions between InterDOC components. The interfaces are conceived through the Facade design pattern.

For each service there is an EDOC Interface and each service has a *ProtocolPort* that performs a third level activity defined in the Enterprise View.

The Protocol structured diagram describes the data involved in a *Protocol* and the ports type. This diagram details the protocol's internal view. Each port has an

association to a *Composite Data* derived from the *Entities Data* defined in the Information View. Figure 13 shows the protocol structure for the *Create_New_Project*. This *Protoco*l is composed of four *Flows Ports*. In this diagram shows the components that initiates (*ApplicationClient*) and responds (*ProjectManagerServer*) the *Protocol.*

Collaboration, Sequence and Activity diagrams should be use to describe the dynamic aspects of a protocol (not discussed in this paper). It is necessary to describe the messages sequence between two or more components, and describes the protocol choreography to put the actions of the ports within one component in a sequence.

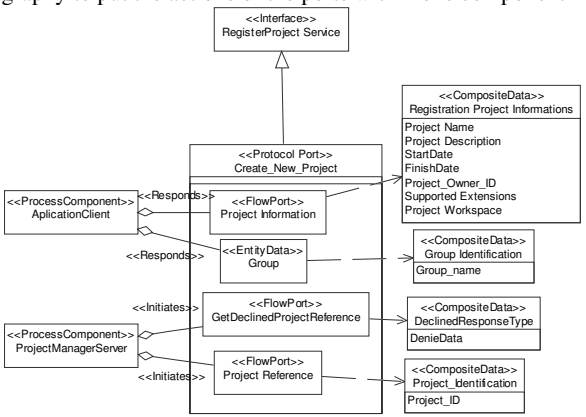

Figure 8 Create_New_Project Protocol Structure Diagram.

4.2 InterDOC PSM – The Service Current Implementation

The current implementation platform for InterDOC is *CORBA Component Model* (CCM). CCM was chosen because it is an OMG's open standard and, like EDOC, utilizes components as the unit for software development.

4.2.1 Engineering and Technological Views

The specification of the Engineering View is derived from the Computational View and it defines the rules for mapping components selected into CCM. The UML profile for CORBA/CCM, up to the moment of writing this paper, is in the process of final revision to become an OMG standard. CCM stereotypes and some mapping rules are based on this profile proposal.

The Technological View specifies the structure of the components in the given technology. Each identified component should be modeled to present the services that it offers to the other components. Figure 9 shows one of the existing relationships between the components *ApplicationClient* and *PlanningServer*. Therefore, in according with Table 1, the *RegisterProject Interface* becomes a *CORBA Interface.* Its *ProtocolPort* is mapped as *CORBAUses* (application client-side) and *CORBAProvides* (InterDOC server-side) that compose the protocol.

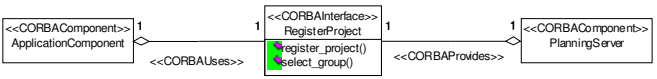

Figure 9 Register a Project CCM Interface.

The last step to be taken in the Operational Model, before the coding phase, is the mapping of CCM interface diagrams into an interface definition language (IDL).

5. Pratical Experiences

Initially, in order to test InterDOC services, application client components for two CASAs were developed. One of these applications is a collaborative editor that has the interoperability as one of its initial non-functional requirements. InterDOC provided the achievement of this requirement. The other application, a legacy one, is also a collaborative editor (Schunemann, 2003) implemented in CORBA 2.3 and it did not provide interoperability functionalities (Figure 10).

In the two situations, it was possible to observe that the developers needed to discuss which information that their applications manipulated has to be mapped onto the InterDOC's services request. Having made this, the components were developed according to the architecture and made available for use in the applications. No discussion was needed about internal InterDOC´s mechanisms or platform-specific characteristics. Hence, developers only focused on domain-specific information instead of platform-implementation aspects.

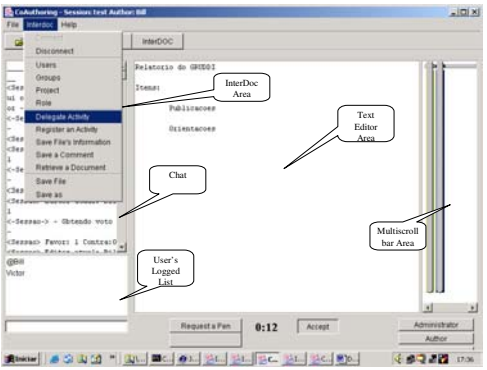

Figure 10 InterDOC Service's Interface in a Collaborative Editor

Several experiments were realized to test InterDOC services. One of this experiments was collaborative writing project whose objective was produce a final report of publications and activities accomplished in a certain research project. Two students and one advisor made up the group. Domain A is the university local area network and the Domain B is each author's homes. Collaborative on-line sessions, was realized in the editor showed in figure 12. The index of the report was written by all the three participants. The invitations (dates and time) for on-line sessions were sent through the InterDOC's notification service. This service has used the community e-mail system.

After defining the report index, the topics were distributed among the authors, so that they could be able to write them in asynchronous sessions. The delegation and the registration of these activities were accomplished through InterDOC. As the collaborative editor only accepts document in text format, the report was formatted in MS-Word. A Java application that integrates all InterDOC's interfaces and that calls

MS-Word was used in those asynchronous sessions. Even with an individual work tool, it could be observed that the context stored at the time of the project registration (group, authors, roles, repository, etc), and at the subsequent phases of the activities (document versions, accomplished activities, documents blocked, etc), was preserved.

6. Conclusion and Future Works

This paper presented the InterDOC's DSA that provides middleware domain-specific services for the interoperability among CASAs. The goal (to final users) is to enable the authoring process for ones who use different collaborative applications. Meanwhile, the proposed architecture also decreases the cost in the execution of this process. The goal (to application developers) is to provide mechanisms to achieve interoperability without increasing the application development life cycle. Since, both DSA and middleware specific-services embody knowledge of a domain, they have the potential to increase system quality and decrease the cycle-time and effort required to develop particular types of distributed applications.

The specification of middleware domain-specific services using a model-driven architecture, as a mechanism for attaining interoperability among heterogeneous environments, instead of frameworks or simply APIs, makes the model more open. Consequently, evolutionary and extendable due to the fact those developers of applications know their external (interfaces) and internal mechanisms (specific services). The reference architecture permits different implementations to be obtained in heterogeneous computer environments. The adoption of a common architecture, for the provisioning of interoperability, also avoids the need to develop specific interoperability mechanisms for each different application involved in the process, and for each new application that a group member wishes to use.

The modeling of the well-established aspects of a domain is an important and fundamental aspect to develop solutions that reuse developers´ experiences, components, frameworks, middleware specific-services, or any other artifact relative to the software development.

Specific-services domain modeling requests a great effort that certainly would not be rewarded if the use was restricted to a specific middleware platform. The approach of specifying those services through an MDA PIM, allows the description of the domain that the service can be applied to, enabling wider service applicability. Since the service specification is based on MDA, the interoperability is reached at the design level. Portability is reached once a PIM can be translated into several PSMs and platform-specific coding.

The approach of specifying those services through several views (RM-ODP views), allows the description of the domain that the service can be applied to, enabling larger service applicability. In addition, using EDOC allow information and behaviour of the target domain to be seen as high-level business process components and activities from the initial models, facilitating the domain decomposition from the service perspectives.

The use of profiles proposed by OMG guarantees, in a more extensive fashion, the interoperability of models proposed by MDA. Methodologies and tools, that assist in the development of applications and use EDOC or other sub-profiles, will have the same conceptual and notational framework. This aspect facilitates the understanding of models both by development teams and by process automation tools.

The development process proposed was used in the components implementation of two Collaborative Authoring Supporting Applications (CASAs). In this experience it was possible to verify that, in spite of EDOC being an extensive UML profile, the combination of the sub-profiles used, as well as the diagrams chosen to comprise the process, resulted in a smooth process that does not overburden the developer.

The PSM specification in a J2EE platform has begun and as it is completed, tests will be carried out in different domains, with different InterDOC implementations to validate our ideas according to the InterDOC's model of use presented in this paper.

References

BAECKER, Ron et al. (1994) Sasse: The Collaborative Editor (video tape transcript). In: *Conference Companion on Human Factors in Computing Systems (CHI´94)*, April, Massachusetts.

BROY, Manfred. (2004) Architecture Driven Modeling in Software Development. In: *IEEE International Conference on Engineering Complex Computer Systems*. April. Italy. p. 3-12.

CERRATTO, Teresa. (1999) Instrumenting Collaborative Writing and Its Cognitive Tools. In: *Human Centered Process Conference*, September, France. p.141-147.

DOURISH, Paul. EDWARDS, Keith; LAMARCA, Anthony. (2000) Extending Document Management Systems with User-Specific Active Properties. *ACM Transactions on Information's Systems*, v.18, n.2, p.140-170.

FAYAAD, Mohamed; SCHIMDT, Douglas; et all. (1999) *Building Applications Frameworks – Object Foundations of Frameworks Design*. EUA.Willey.

FRANKEL, David S. (2004) Software Industrialization and the New IT: A Perspective on MDA. MDA Journal. January. http://www. mkpress.com/mda.

GERVAIS, Marie, (2002). Towards MDA-Oriented Methodology. In: *Annual International Computer Software and Applications Conference*, August, England, p 265-270.

GUELFI, Nicolas; RIES, Benoît; et all. (2003). MEDAL: A Case Tool Extension for Model-Driven Software Engineering. In: *IEEE International Conference on Software-Science, Technology Engineering***,** November, Israel, p. 33-44**.**

GUELFI, Nicolas; RAZAVI, Reza; et all. (2003). DRIP Catalyst: An MDE/MDA Method for Fault-tolerant Distributed Software Families Development. In: *OOPSLA Workshop on Best Practices for Model Driven Software Development.* Canada.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION - ISO. (1995) Basic Reference Model of Open Distributed Process, *ISO/IECIS 10746.* Parts 1-4.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION-ISO (2004). Use of UML for ODP system specification. Working Draft. ISSO/IEC JTC1/SC7.

KIM, Hee-Cheol; EKLUNDH, Kerstein. (2001) Reviewing Practices in Collaborative Writing. In: *Computer Supported Cooperative Work (10),* Netherlands: Kluver Academic Publishers. p 247-259.

LI, Du et all. (2003) Using Familiar Single-Users Editors for Collaborative Editing.

In: *Hawaii International Conference on System Sciences (HICSS'03)*, 36 th, 2003, Hawaii. January. 10 p.

NOEL, Sylvie; ROBERT, Jean-Marc. (2004) Empirical Study on Collaborative Writing: What Do Co-authors Do, Use, and Like?. In: *Computer Supported Cooperative Work (13),* Netherlands: Kluver Academic Publishers. p 63-89.

MELLOR. Stephen J. (2003) Model-Driven Development. *IEEE Software*. September v. 20, n. 5, p 14-18.

MACIEL, Rita S. P. (2004). A Model-Driven Architecture for Interoperable Collaborative Writing Environments. In: *X International Workshop on Groupware - Doctoral Colloquium*, San Carlos. Costa Rica.

MACIEL, Rita S. P. CARREIRO, Bruno, et al. (2005) An MDA-EDOC Based Development Process For Distributed Applications. In: *7th International Conference on Enterprise Information Systems (ICEIS 2005)*, p 3-11.

OBJECT MANAGEMENT GROUP. (2002) UML Profile for Enterprise Distributed Object Computing Specification. *OMG Adopted Specification* (ptc/02-02-05).

OBJECT MANAGEMENT GROUP. (2003) *MDA Guide Version 1.0*. 2003

RIBEIRO, Semíramis., MACIEL, Rita S.P. A Framework for Document-Based Applications. In: *ERBASE 2004 – 4th Regional School of Computing Bahia-Sergipe*. 2004. (In Portuguese).

SCHANTZ, R., SCHMIDT, D. (2001) *Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications*, Encyclopedia of Software Engineering, Wiley & Sons.

SHUENEMANN, Hermann, MACIEL, R. S. (2002). *A Tool for the Co-Authoring of XML Documents*. Undergraduate Final Report. Computer Science, Faculdade Ruy Barbosa. 82 p. (In Portuguese).

STIEMERLING, Oliver; CREMERS, Armim. (2000)The Evolve Project: Component-Based Tailorability for CSCW Applications. *AI & Society*, London. p. 120-141.

TAYLOR, Richard., TRACZ, Will. (1995) Software Development Using Domain-Specific Software Architectures. *SIGSOFT Software Engineering Notes*, v. 20. December, p 20-37.

TIETZE, Daniel. (2001) *A Framework for Developing Component-based Co-operative Applications*. 178 p. Ph. D. Dissertation (Computer Science), Technischen Universität Darmstadt, Germany.

WANG, H.; ZHANG, D. (2003). MDA-based Development of E-Learning System, In: *27th International Computer Software and Applications Conference*, IEEE Press, November, p. 684-689.