ADILSON VAHLDICK

CELINE: UM MODELO PARA UTILIZAÇÃO E ADAPTAÇÃO DE CONTEÚDO SCORM EM AMBIENTES INTELIGENTES DE APRENDIZAGEM

São José (SC), dezembro de 2008

UNIVERSIDADE DO VALE DO ITAJAÍ

CURSO DE MESTRADO ACADÊMICO EM COMPUTAÇÃO APLICADA

CELINE: UM MODELO PARA UTILIZAÇÃO E ADAPTAÇÃO DE CONTEÚDO SCORM EM AMBIENTES INTELIGENTES DE APRENDIZAGEM

por

Adilson Vahldick

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Computação Aplicada.
Orientador: André Luís Alice Raabe, Dr.

São José (SC), dezembro de 2008

“CELINE: Um componente para utilização e adaptação de conteúdo SCORM em ambientes inteligentes de aprendizagem”.

ADILSON VAHLDICK

‘Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Computação Aplicada, Área de Concentração Computação Aplicada e aprovada em sua forma final pelo Programa de Mestrado em Computação Aplicada da Universidade do Vale do Itajaí’.

_____
Prof. André Luís Alice Raabe, Dr.
Orientador

_____
Prof. Cesar Albenes Zeferino, Dr.
Coordenador do Programa Mestrado em Computação Aplicada

Apresentado perante a Banca Examinadora composta pelos Professores:

_____
Prof. André Luís Alice Raabe, Dr. (UNIVALI)
Presidente

_____
Profa. Anita Maria da Rocha Fernandes, Dra. (UNIVALI)
Membro

_____
Prof. Raimundo Celeste Ghizoni Teive, Dr. (UNIVALI)
Membro

_____
Prof. Ricardo Azambuja Silveira, Dr. (UFSC)
Membro

São José (SC), 15 de dezembro de 2008.

Às minhas fontes de inspiração: Bruce e Celine.

A curiosidade constante pela resolução de novos problemas
é atributo seguro do homem altamente inteligente.

Dr. José Reis

AGRADECIMENTOS

Muita gente fez parte da minha vida durante o desenvolvimento desse trabalho, e que de alguma forma, contribuiu em alguma linha de código, ponto de vista ou pensamento. Alguns nem vou lembrar, e outros nem posso citar. Mas do fundo do coração, agradeço muito a todas essas pessoas.

De forma especial, reconheço a importância da família, principalmente meus pais, Lauro e Edla, que forneceram abrigo e atenção em momentos complicados nos dois anos de mestrado.

Ao meu amigo, André Luis Alice Raabe, que desempenhou o papel de artesão, permitindo o meu aperfeiçoamento como mestre, combinando elogios e reprovações na medida certa pensando sempre em minha integridade.

Agradeço o companheirismo e ombro amigo de todos os colegas da primeira turma, em especial, os grandes *javafesteiros* Jackson e Jonny.

Uma desonra se eu não citasse meus valorosos professores Anita, Raimundo, Adhemar e Eros, à Lurdes sempre disposta a nos atender na maior brevidade, e ao coordenador Cesar sempre preocupado no encaminhamento de nossas atividades.

CELINE: UM MODELO PARA UTILIZAÇÃO E ADAPTAÇÃO DE CONTEÚDO SCORM EM AMBIENTES INTELIGENTES DE APRENDIZAGEM

Adilson Vahldick

Dezembro / 2008

Orientador: André Luís Alice Raabe, Dr.

Área de Concentração: Computação Aplicada

Palavras-chave: SCORM. Adaptabilidade. Padrões em tecnologias educacionais. Ambientes Inteligentes de Aprendizagem.

Número de páginas: 140

RESUMO

O esforço em manter conteúdos em AIA (Ambientes Inteligentes de Aprendizagem) foi a inspiração para o desenvolvimento desta dissertação. Observou-se que os conteúdos a serem ensinados são fortemente dependentes desses ambientes, dificultando seu compartilhamento com outros sistemas. Além disso, esses materiais são criados com ferramentas internas, que oferecem poucos recursos ao projetista instrucional. Com o intuito de permitir que conteúdos mais ricos em recursos possam ser explorados por esse tipo de ambiente, esta dissertação apresenta o desenvolvimento de uma infra estrutura para gerenciamento, adaptação e apresentação de conteúdo no formato SCORM (Sharable Content Object Reference Model). Nesta monografia é apresentada uma visão geral sobre padrões e especificações em tecnologias educacionais, descrito o modelo SCORM, comentado sobre a Engenharia de Software Baseada em Componentes e resumida a tecnologia JEE (Java Enterprise Edition). A infra estrutura está disponibilizada como um componente, batizado como CELINE, que pode ser acoplado às aplicações web com JEE. Ela foi dividida em duas partes: um framework para execução de pacotes SCORM, e outro para criação desses ambientes web. São fornecidas tags customizadas para a confecção das páginas JSP, permitindo o acesso aos recursos do componente. Quanto ao gerenciamento, é possível importar e manter pacotes SCORM, cadastrar usuários e visualizar dados sobre os conteúdos acessados pelos usuários. O componente permite a execução de pacotes SCORM na versão 2004 3rd Edition, que é a mais nova versão até a data do desenvolvimento deste trabalho. Quanto a adaptação, os ambientes

conseguem montar cursos com pedaços de outros cursos, assim como alterar a estrutura do curso enquanto o aluno interage com ele. A persistência dos dados pode ser realizada através de arquivos XML, banco de dados relacional ou alguma criada pelo desenvolvedor. Neste trabalho está especificada a arquitetura e seu processo de implementação. Durante a execução desta dissertação, foram desenvolvidos quatro ambientes de aprendizagem: BRUCE (Basic Reference of Using the CELINE Component), que serve como documentação e referência de utilização do componente; SAGAA, um STI (Sistema Tutor Inteligente) desenvolvido como TCC (Trabalho de Conclusão de Curso) com intuito de inspirar e validar recursos para adaptação do conteúdo; ALICE 2.0, um ambiente de aprendizagem utilizado na UNIVALI, pelos alunos de Algoritmos I; e o ASTRO, um ambiente de aprendizagem utilizado na FURB pelos alunos em três turmas de duas disciplinas nos cursos de Licenciatura em Computação, Ciências da Computação e Sistemas de Informação. Os quatro ambientes também estão descritos neste trabalho. Além disso, o componente foi aplicado com testes de conformidade disponibilizados pela ADL (Advanced Distributed Learning) que é a entidade responsável pela manutenção do SCORM. Os resultados dos testes são apresentados nesta monografia, demonstrando que os ambientes que utilizarem o componente podem receber o selo “LMS SCORM 2004 3rd Edition Conformant”. Ao final desta dissertação, são descritas as conclusões no desenvolvimento e utilização dos ambientes testados pelos alunos, e sugestões para trabalhos futuros.

CELINE: A MODEL FOR USE AND ADAPTATION OF SCORM CONTENT IN INTELLIGENT LEARNING ENVIRONMENTS

Adilson Vahldick

December / 2008

Advisor: André Luís Alice Raabe, Dr.

Area of Concentration: Applied Computer Science.

Keywords: SCORM. Adaptability. Standards in educational technology. Intelligent Learning Environments.

Number of pages: 140

ABSTRACT

The effort to maintain contents in ILE (Intelligent Learning Environments) was the inspiration behind the development of this work. It was observed that the contents to be taught are heavily reliant on these environments, which makes their sharing with other systems difficult. In addition, these materials are created with internal tools, which offer few resources for the instructional designer. In order to enable more content-rich resources to be exploited by this type of environment, this work presents the development of an infrastructure for managing, adapting and presenting content in the SCORM (Sharable Content Object Reference Model). This work presents an overview of standards and specifications in educational technologies, describes the SCORM model, comments on Component Based Software Engineering, and summarizes the JEE (Java Enterprise Edition) technology. The infrastructure is made available as a component, called CELINE, which can be coupled to the JEE web applications. It was divided into two sections: a framework for execution of SCORM packages, and another for generating these web environments. Customized tags are provided for the construction of JSP pages, enabling access to the resources of the component. In terms of management, it is possible to download and keep SCORM packages, register users, and view data on the content accessed by users. The component allows the execution of SCORM packages in 2004 3rd Edition, which was the latest version at the time this paper was written. In relation to adaptation, the environments can set up courses from parts of other courses, as well as change the structure of the course as the student interacts with it. Data persistence can be achieved through XML files, relational database or some other tool created by the developer. This

work specifies the architecture and its implementation process. During the execution of this work, four learning environments were developed: BRUCE (Basic Reference for Use of the CELINE component) which serves as a reference and documentation for the use of the component; SAGAA, an ITS (Intelligent Tutor System) developed as CCW (Course Conclusion Work) in order to inspire and validate resources for adaptation of content; ALICE 2.0, a learning environment of UNIVALI used by students of Algorithms I, and ASTRO, a learning environment of FURB used by students in three classes of two disciplines in the Bachelor in Computing, Computer Science and Information Systems. The four environments are also described in this work. The component has also been submitted to compliance tests provided by ADL (Advanced Distributed Learning), which is responsible for maintaining SCORM. The test results are presented in this work, demonstrating that environments that use the component can receive the "LMS SCORM 2004 3rd Edition Conformant" seal. At the end of this work, the conclusions of the development and use of the environments tested by the students are described, and suggestions are given for future work.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADL	Advanced Distributed Learning
AIA	Ambiente Inteligente de Aprendizagem
AICC	Aviation Industry Computer-Based Training Committee
ALICE	Algorithm Learning Internet Based Computer Environment
API	Application Programming Interface
BDR	Banco de Dados Relacional
BRUCE	Basic Reference of Using the CELINE Component
CAM	Content Aggregation Model
CC	Common Cartridge
CMI	Computer Managed Instruction
CTS	Conformance Test Suite
DBC	Desenvolvimento Baseado em Componentes
DCMI	Dublin Core Metadata Initiative
ED	Engenharia de Domínio
ESBC	Engenharia de Software Baseada em Componentes
FURB	Fundação Universidade Regional de Blumenau
HTML	Hypertext Markup Language
IA	Inteligência Artificial
IEEE	Institute of Electrical and Electronics Engineers
IEEE LTSC	IEEE Learning Technology Standards Comittee
IMS GLC	Instructional Management System Global Learning Consortium
JAR	Java Archive
JDBC	Java Database Connectivity
JCP	Java Community Process
JEE	Java Enterprise Edition
JSF	Java Server Faces
JSP	Java Server Page
JSTL	Java Standard Tag Library
LMS	Learning Management Systems
LOM	Learning Object Metadata
OA	Objeto de Aprendizagem
PDF	Portable Document Format
PIPG	Programa de Integração Pós-Graduação e Graduação
RTE	Run-Time Environment
SAGAA	Sistema de Aprendizagem do Gerenciador de Arquivos e Armazenamento
SBIE	Simpósio Brasileiro de Informática na Educação
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
SHA	Sistema de Hipermídia Adaptativa
SN	Sequencing and Navigation
SQL	Structured Query Language
SSP	Shareable State Persistence
STI	Sistema Tutor Inteligente
TLD	Tag Library Descriptor
UNIVALI	Universidade do Vale do Itajaí

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

As tecnologias educacionais têm se desenvolvido muito na última década. Observa-se um acréscimo na quantidade e qualidade de ambientes de ensino e aprendizagem, tanto nos recursos disponíveis quanto na forma em que os objetos de aprendizagem são oferecidos e organizados (ABEL et al, 2007).

Os objetos de aprendizagem (OA) são conceituados como entidades digitais, ou não, que podem ser reusadas durante o ensino com suporte tecnológico (IEEE LTSC, 2002; WILEY, 2000). Por esse conceito, se conclui que qualquer material é aceito como OA, desde que ele possa ser inserido em uma atividade de ensino. Porém, devido a popularização da internet, e para orientar as pesquisas na área de tecnologias educacionais, se têm aceitado como OA os materiais que possam ser renderizados em navegadores, e entregues aos alunos por sistemas específicos para o gerenciamento do ensino-aprendizagem.

Um desses tipos de sistemas de apoio é o LMS (Learning Management Systems) que, segundo Irlbeck e Mowat (2007), tem como objetivo ajudar a administrar as atividades relacionadas ao aprendizado. Com ele montam-se cursos agrupando e organizando os OA em uma estrutura que representa a ordem em que o aluno deve acessá-los. O sistema disponibiliza o material seguindo essa seqüência, que é baseada na interação de cada aluno, avaliando os materiais já visitados e o desempenho nas avaliações. Os LMS constituem uma forma mais recente de sistemas, que se diferenciam dos tradicionais ambientes de aprendizagem por possuírem integração com outros tipos de sistemas de apoio, rastreamento e apresentação sofisticada do desempenho do estudante nas atividades de aprendizado, centralização dos dados do aluno, recursos de colaboração e adaptabilidade na distribuição do conteúdo (ADL, 2006b).

Para promover a reutilização dos OAs, estes precisam ser portáveis de um LMS para outro. Grupos internacionais se organizaram para estabelecer especificações de como estruturar e organizar o material instrucional para que os LMS possam selecionar e fornecer os objetos para o usuário, assim como se comunicar com esses objetos. Esses grupos, que têm a finalidade de estabelecer regras visando a padronização na utilização de tecnologias para o ensino e aprendizado, são o IMS GLC (Instructional Management System Global Learning Consortium), AICC (Aviation

Industry Computer-Based Training Comittee) e ADL (Advanced Distributed Learning). Além disso, as especificações consolidadas se transformam em normas certificadas pelo IEEE LTSC (Learning Technology Standards Comittee).

A ADL mantém uma especificação chamada SCORM (Sharable Content Object Reference Model) formada por um conjunto de padrões e especificações de outros grupos (IMS GLC, AICC e IEEE LTSC) com a finalidade de definir um ambiente completo para execução de OAs, e promover a criação de conteúdo reusável. Ela serve como guia para o empacotamento do material instrucional, com vistas a definir uma seqüência de execução desse conteúdo e regras de como esse material se comunica com o LMS e este com o material (ADL, 2006b).

Ao estabelecer um formato padrão para reunir conteúdo e estrutura, permitindo que ele seja reusado pelos LMS, fomenta-se a pesquisa e criação de ferramentas de autoria (p.e. MOS Solo, eXe Learning, Reload, HotPotatoes, CourseLab) e processos (p.e. LODAS (WILEY, 2000)) de composição do material, promovendo a formação de equipes multidisciplinares para elaboração de conteúdo instrucional.

A adaptabilidade é um dos requisitos da ADL no desenvolvimento da especificação SCORM (ADL, 2006b). A adaptação é prevista pelo autor do material, também chamado de designer instrucional (FILATRO, 2004), através de condicionais como, por exemplo, se o aluno acertou abaixo de 60% das questões então é direcionado por um caminho, senão é direcionado por outro. A ordem do que será ensinado já vem estabelecida no próprio conteúdo. Os LMS apresentam o conteúdo seguindo essas regras e o aluno interage exclusivamente com o material, sem interferência de outros atores no seqüenciamento. Esse processamento desconsidera a individualidade do aluno, ou seja, suas características e necessidades. É como se o professor, em uma aula presencial, não tivesse a possibilidade de usar estratégias diferentes para esclarecer um assunto que alguns alunos não entenderam.

Por outro lado, existe uma categoria de ambientes que conseguem adaptar o ensino respeitando a individualidade dos alunos, que são os AIA (Ambientes Inteligentes de Aprendizagem) (HATZILYGEROUDIS e PRENTZAS, 2006). Dentro dessa categoria, existem outras categorias como Sistemas de Hipermídia Adaptativa (SHA) e Sistemas Tutores Inteligentes (STI). Esses ambientes consideram o histórico, características e preferências do aluno para personalizar e adaptar a seqüência e o conteúdo a ser apresentado. Porém, nesses casos, o conteúdo

e ambiente costumam ser dependentes entre si, ou seja, não é possível incorporar o conteúdo desenvolvido por outros de forma facilitada como acontece nos LMS.

Através de análise de anais, literatura e a própria experiência dos pesquisadores desse trabalho, observa-se que os criadores do conteúdo desses ambientes são normalmente o próprio desenvolvedor ou o professor. Esse conteúdo é elaborado usando ferramentas incorporadas ao ambiente, e que normalmente são simplistas quanto a recursos de apresentação, ou se referem a links a conteúdos disponíveis na internet, ou cópia de arquivos (p.e. PDF, Flash, imagens) para o sistema. Essa forma de compor o conteúdo dificulta o compartilhamento das aulas e cursos entre a comunidade externa.

O presente trabalho oportuniza que esses ambientes inteligentes possam utilizar material no modelo SCORM através de uma infra estrutura baseada em componentes. O componente oferece recursos para administrar e executar conteúdo SCORM, e inclusive, acompanhar toda a interação dos usuários com esses conteúdos. Além disso, com um mecanismo também proporcionado pelo componente, os ambientes podem adaptar a estrutura e seqüência do conteúdo utilizando os dados do acompanhamento em conjunto com suas próprias estratégias pedagógicas e regras de inferência. A adaptação empenhada pelos AIA, de acordo com Brusilovsky e Vassileva (2003 apud KARAMPIPERIS e SAMPSON, 2006), pode acontecer em dois momentos: quando ele é apresentado ao aluno – e para isso se baseia no conhecimento inicial do aluno (geração adaptativa de conteúdo) – e durante a interação do aluno com o curso (geração dinâmica de conteúdo). Seguindo esses princípios, o mecanismo oferece (i) a possibilidade do ambiente gerar um curso quando o aluno entra nele, e (ii) permite alterar a estrutura e as regras de seqüenciamento durante a interação do aluno com o conteúdo.

O ambiente que utilizar essa infra estrutura conseguirá administrar seu conteúdo independente do desenvolvedor do sistema, reusando material já criado por outros, e estimulando a produção de novo material. Seguindo um padrão internacional, tal como o SCORM, agrega-se mais valor ao ambiente, pois permite que mais pessoas participem no processo de desenvolvimento do material instrucional, facilitando a atualização e manutenção do conteúdo apresentado.

A Figura 1 apresenta uma visão geral do componente proposto nessa dissertação. Observa-se a interação da aplicação web e a associação com os pacotes de conteúdo SCORM.

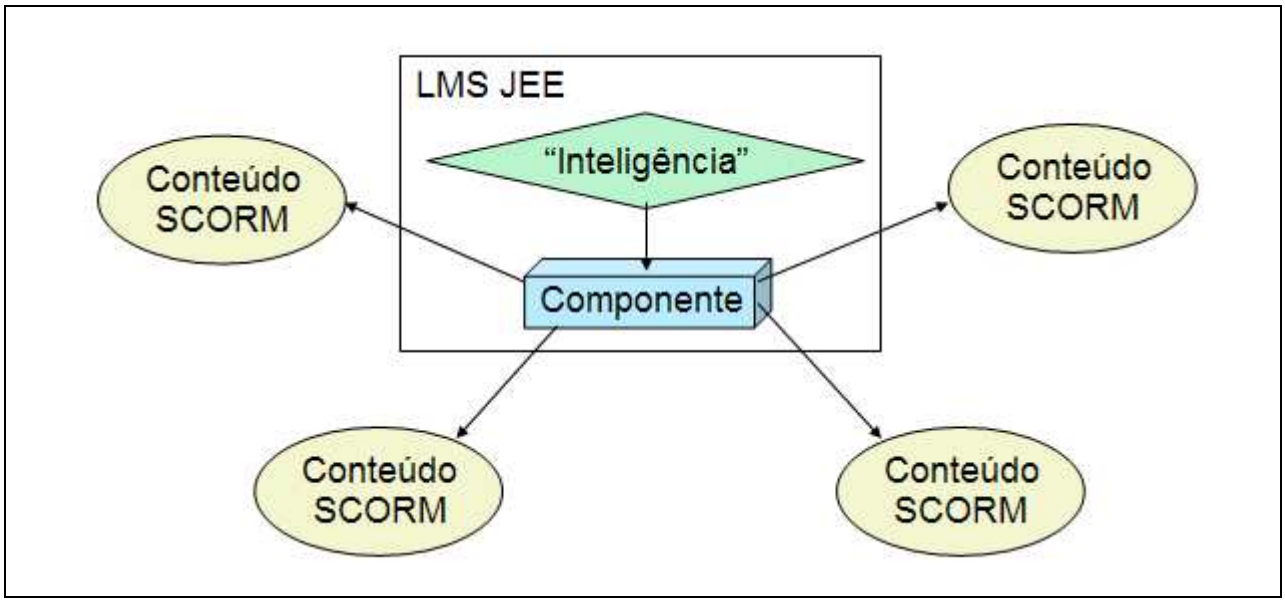


Figura 1. Visão geral do componente conectado à aplicação web

Um componente é um artefato de software que tem um conjunto de entradas e saídas bem definidas, e documentadas, e que pode ser acoplado facilmente às aplicações (MCCLURE, 2001, p. 4; BARROCA et al, 2005, p. 4). O componente é naturalmente desenvolvido para ser reusado. As entradas configuram o comportamento do componente, ou seja, ele pode ser adaptado às necessidades da aplicação. As saídas podem ser dados apresentados à aplicação, e direcionadas, por exemplo, à interface com o usuário ou a um mecanismo de persistência. A granularidade de um componente é muito baixa, pois ele pode ter uma solução completa de um problema, e são as entradas que garantem o quanto do componente será utilizado. Devido a essa granularidade um componente não tem a propriedade da estensibilidade, isto é, não é possível a partir dele adicionar comportamentos a não ser que seja desenvolvido um outro componente e o original seja acoplado a ele. Conclui-se daí que o acoplamento é a característica que diferencia um componente de um objeto (MCCLURE, 2001, p. 5).

O componente foi batizado como CELINE e segue a especificação JEE, ou seja, um componente acoplável às aplicações web em Java. A JEE (Java Enterprise Edition) é uma especificação para o desenvolvimento de aplicações robustas e distribuídas, por exemplo, aplicações web. Essa especificação garante um ambiente de execução, denominado servidor, padronizado e seguro para as aplicações. A especificação determina como uma aplicação deve ser estruturada, implantada, definida para que tenha recursos de autenticação e autorização de usuários,

como utilizar um ambiente transacional, distribuir seus objetos e assim por diante (BOND et al, 2003).

Esse trabalho apresenta também uma análise de trabalhos similares: dois ambientes que utilizam pacotes SCORM, e artigos que adaptam os conteúdos com a necessidade do aluno. Os ambientes avaliados serviram para levantar requisitos para a infra estrutura que o CELINE deveria oferecer. Nenhum deles pode ser classificado como inteligentes, e por essa razão, não possuem a capacidade de adaptar os pacotes. Porém, da análise dos artigos verificou-se que eles estendem a especificação SCORM, ou adicionam recursos para adaptar os cursos. Entre esses trabalhos, existem plugins para ambientes de ensino específicos, ferramentas que dependem de intervenção do professor, ou que avaliam o desempenho dos alunos e apresentam revisões da estrutura do conteúdo, além de arquiteturas de sistemas com o uso de agentes.

Quatro ambientes foram desenvolvidos com o CELINE, e dois deles foram colocados à prova por alunos. Os desenvolvedores desses dois ambientes estavam totalmente alheios à especificação SCORM, demonstrando que o componente conseguiu abstrair a complexidade do SCORM. Os conteúdos foram produzidos com ferramentas de autoria distintas, ambas com facilidade de uso, e recursos abundantes para explorar a interatividade. Com os experimentos realizados em “sala de aula”, pôde-se perceber que os alunos estavam motivados durante a utilização do material produzido.

1.2 PROBLEMA DE PESQUISA

A possibilidade de adicionar pacotes de conteúdo SCORM em Ambientes Inteligentes de Aprendizagem permite que outros não envolvidos no projeto do sistema possam elaborar o material instrucional, além de permitir novas fontes de conteúdos aos mantenedores desses ambientes, através dos repositórios existentes na Internet. Porém, não existem disponíveis meios para incorporar a implementação da especificação nesses ambientes, e sua realização é bastante complexa.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Especificar e desenvolver uma infra estrutura para que ambientes de aprendizagem possam escolher, adaptar e seqüenciar objetos de aprendizagem no formato SCORM.

1.3.2 Objetivos Específicos

- Pesquisar e descrever os padrões e especificações de Objetos de Aprendizagem dando ênfase ao modelo SCORM;
- Avaliar as contribuições do modelo SCORM em Ambientes Inteligentes de Aprendizagem;
- Identificar e descrever características dos Objetos de Aprendizagem que explorem as potencialidades da comunicação com os ambientes que seguem o SCORM;
- Pesquisar o paradigma de desenvolvimento baseado em componentes;
- Desenvolver um componente JEE que atenda a especificação SCORM viabilizando a integração de Objetos de Aprendizagem em LMS;
- Fornecer mecanismos no componente para que AIA possam adaptar o conteúdo SCORM;
- Utilizar esse componente para desenvolver LMS e experimentá-los com alunos.

1.4 METODOLOGIA

O desenvolvimento desta dissertação seguiu os passos enumerados a seguir.

Primeiramente, efetuou-se um estudo da especificação SCORM, através da documentação fornecida pela ADL. Além disso, foi experimentado o ambiente de execução de referência da implementação, também disponível no site da ADL. Para compreensão aprofundada das regras de sequenciamento e navegação, foi utilizada a ferramenta de autoria MOS Solo.

Em seguida, foi implementada a especificação SCORM, fornecendo recursos para leitura e interpretação de pacotes SCORM, sem agregar recursos de aplicações web. Para isso, foram necessárias pesquisas e estudos de frameworks para a interpretação do arquivo de manifesto.

Para propor o modelo de componente, foram estudadas a Engenharia de Software Baseada em Componentes, assim como trabalhos correlatos para relacionar os requisitos do componente. Nesse estudo foi feito o levantamento de artigos publicados que envolvem a especificação SCORM, assim como trabalhos que propõe adaptação em pacotes SCORM. Também foram feitas simulações com o Moodle.

De posse desses dados, foi o especificado e desenvolvido o componente. A implementação aconteceu em paralelo com o desenvolvimento do ALICE 2.0. Os testes dos recursos de adaptação de pacotes SCORM foram fomentados com o SAGAA. Para testar o ambiente com desenvolvedores inexperientes, foi criado o ambiente ASTRO, envolvendo bolsistas sem conhecimento de SCORM, e três turmas de graduação utilizaram-no. Esses ambientes estão descritos no quarto capítulo desta dissertação.

Por fim, foi executado o teste de conformidade, também fornecida pela ADL, com o objetivo de resolver possíveis erros não percebidos com o desenvolvimento dos ambientes.

1.5 ORGANIZAÇÃO DO DOCUMENTO

O presente trabalho foi estruturado em cinco capítulos. O primeiro capítulo apresenta a contextualização e os objetivos da dissertação. O segundo capítulo expõe a fundamentação teórica necessária para o desenvolvimento desta dissertação: relaciona padrões e especificações relevantes em tecnologias para o processo educacional, resumo da especificação SCORM, descreve o processo de engenharia de software baseada em componentes, enumera as tecnologias JEE, e por último cita os trabalhos correlatos que conduziram o desenvolvimento do modelo proposto nessa dissertação. O terceiro capítulo apresenta o modelo desenvolvido, com toda a sua arquitetura e tecnologias utilizadas. O quarto capítulo traz os resultados dessa implementação, relacionando os ambientes que foram desenvolvidos com o componente. Por fim, o último capítulo descreve as conclusões do trabalho e propostas para trabalhos futuros.

2 REVISÃO DA LITERATURA

2.1 Padrões e Especificações em Tecnologias no Processo Educacional

A popularização da Internet promoveu a disseminação de vários serviços e novas tecnologias. Um dos grandes beneficiados é a Educação. Ao mesmo tempo em que a Internet é considerada uma biblioteca gigantesca, onde é possível encontrar sobre quase tudo, e sobre vários pontos de vista, a infra-estrutura da web impulsionou o aperfeiçoamento do ensino a distância.

Atualmente existem vários ambientes de aprendizagem proprietários (Blackboard, WebCT, Angel, Enhance) e de código livre (Moodle, Learnloop, Teleduc), com recursos como fóruns, chats, correio eletrônico e área para disponibilização de material. Com a adoção desses ambientes pelas instituições de ensino, os professores se transformaram em autores de material instrucional digital. Em geral esse conteúdo é criado na forma de slides (p.e. com MS Power Point), páginas HTML, textos e outros formatos diversos. Entretanto, esses conteúdos são versões eletrônicas de livros, revistas e textos que antes os professores disponibilizavam aos seus alunos por meio impresso. Esse material não se encontra num contexto automatizado de aprendizado em que os ambientes possam inferir sobre ele, e fornecer ao aluno uma seqüência de materiais segundo um planejamento instrucional.

Observando essa necessidade, algumas instituições e fabricantes desses ambientes se uniram e formaram um consórcio para desenvolver um padrão aberto e baseado no mercado para o ensino online, incluindo especificações de metadados para o conteúdo instrucional (METADATA ADVISORY ..., 2004 *apud* MITCHELL e FARHA, 2007). Esse consórcio foi chamado de IMS GLC. Mais tarde, o ARIADNE, que era um repositório de objetos de aprendizagem, e o IMS GLC, baseado no metadados do Dublin Core, sugeriram a criação de um padrão para descrição do conteúdo instrucional, o que acabou gerando um novo projeto no IEEE batizado de 1484 com objetivos de definir padrões em tecnologias educacionais. Em 2002 foi definido então o padrão LOM (*Learning Object Metadata*) no documento Std 1484.12.1-2002. Inclusive, nesse documento foi definido o conceito de objeto de aprendizagem referenciado na maioria dos trabalhos sobre o assunto que é “qualquer entidade, digital ou não, que é usado para o aprendizado, educação ou treinamento”. Foi criado em 2000 o IEEE LTSC (*Learning Technology Standards Commitee*) para a definição de padrões para a tecnologia educacional.

O *Dublin Core Metadata* (DCMI) é uma iniciativa de estabelecer um padrão de metadados para qualquer recurso disponível na internet. Nele são definidos os elementos sintáticos, mas não o conteúdo que esses elementos devem possuir (MITCHELL e FARHA, 2007).

Na última década foram despendidos esforços para estabelecer e difundir o uso de padrões nos metadados para facilitar a procura por eles nos repositórios. Nos últimos cinco anos houve uma atenção maior em outra direção: em criar uma arquitetura para execução desses objetos de aprendizagem, e que o projeto instrucional pudesse contribuir no desenvolvimento desses objetos.

Encontrar o conteúdo já não é o suficiente. Os sistemas de apoio tecnológico à Educação, chamados de LMS, desejam gerenciar e conduzir a distribuição dos objetos para os alunos. A definição de uma arquitetura comum permite a portabilidade desses objetos entre os produtos LMS, e ainda, favorece a comunicação entre sistemas diferentes.

O sucesso dessa automatização depende do seqüenciamento dos objetos, isto é, em que condições um objeto pode ser apresentado e fornecido ao aluno (WILEY, 2000). Para que o sistema possa selecionar e seqüenciar os objetos, se faz necessário a descrição das relações entre esses objetos. Nesse sentido, foram consideradas experiências de entidades renomadas, como o AICC, que define como devem ser montados os cursos para treinamento de seu pessoal. O AICC é uma organização internacional que promove a padronização de tecnologias educacionais na aviação. O AICC definiu um guia de nome CMI (*Computer Managed Instruction*) Guideline for Interoperability onde estão os modelos de como o conteúdo deve se comunicar com os sistemas.

O IEEE LTSC aprovou padrões baseados no CMI que são o Std 1484.11.1-2004 (define um modelo de dados para comunicação entre objeto de aprendizagem e o LMS), Std 1484.11.2-2003 (API (*Application Programming Interface*) de comunicação entre os objetos de aprendizagem e o LMS) e Std 1484.11.3-2005 (mapeamento do modelo 1484.11.1-2004 em XML (*Extensible Markup Language*)). Por outro lado, o IMS GLC criou a especificação IMS Simple Sequencing (IMSSS) com a finalidade de criar a estrutura de seqüenciamento dos objetos de aprendizagem. Além disso, também definiram a especificação IMS Content Packaging que é o modelo de como agrupar o conteúdo com as regras de seqüenciamento.

Percebeu-se que existia uma lacuna na integração entre esses padrões e especificações para promover um sistema para a construção de conteúdo instrucional. A ADL, em participação com

outras organizações, desenvolveu em 2000 a junção de várias especificações criando o SCORM. Essa especificação será descrita na Seção 2.2.

Enquanto isso, o IMS GLC trabalha em especificar outros processos e necessidades em ambientes educacionais, como os testes e avaliações, projeto instrucional, acessibilidade à pessoas de necessidades especiais, ambientes colaborativos, integração usando serviços web com outras naturezas de sistemas que uma instituição de ensino utiliza, além de guias de como desenvolver os conteúdos usando essas especificações, e desenvolver os sistemas garantindo a interoperabilidade.

Em 2006 o IMS GLC anunciou a criação de um novo grupo com propósito de firmar a interoperabilidade total entre o conteúdo, seqüenciamento e integração entre sistemas, e com vistas a definir uma nova especificação denominada *Common Cartridge* (CC). Até o momento a especificação não está liberada para acesso público, mas as empresas envolvidas na definição da CC já possuem produtos lançados seguindo essa especificação.

2.2 SCORM

SCORM (*Sharable Content Object Reference Model*) é o nome da especificação que visa promover a criação de conteúdo reusável para aprendizagem como objetos instrucionais com uma estrutura técnica universal para o aprendizado baseado em computador e na web (ADL, 2006b). Ela é mantida pela ADL (*Advanced Distributed Learning*) que adaptou outras especificações, padrões e guias. A ADL tem como parceiros a ARIADNE, AICC, IEEE LTSC e IMS GLC. A liberação das versões do SCORM com as principais mudanças são ilustradas na Tabela 1.

Tabela 1. Liberações do SCORM

Versão	**Data**	**Recursos**
1.0	Janeiro, 2000	Sharable Courseware Reference Model Formato de cursos (*Course Structure Format*) Ambiente de execução (*Run-Time Environment*) da AICC
1.1	Janeiro, 2001	Troca de Courseware para Content A especificação do AICC foi submetido para padronização pelo IEEE LTSC, e as mudanças nesse processo foram refletidas nessa versão
1.2	Outubro, 2001	Incorporada a especificação Content Packaging da IMS GLC
2004 1st edition	Janeiro, 2004	Atualizada a estrutura dos metadados para refletir a padronização do LOM pela IEEE LTSC Incorporado o Simple Sequencing da IMS GLC Dividiram em livros as especificações envolvidas no SCORM
2004 2nd edition	Julho, 2004	Revisões na documentação
2004 3rd edition	Outubro, 2006	Atualizado o Content Packaging para refletir a nova versão da IMS GLC

Fonte: ADL (2007b).

A especificação SCORM 2004 está dividida em três partes:

- *Content Aggregation Model* (CAM): o **modelo de agregação do conteúdo** define como estruturar num documento XML o conteúdo instrucional;
- *Run-Time Environment* (RTE): o **ambiente de execução** estabelece como o conteúdo instrucional se comunica com o LMS e vice-versa; e
- *Sequencing and Navigation* (SN): o **seqüenciamento e navegação** instituem como ocorre a seleção do conteúdo instrucional baseado nas interações do usuário e na definição do modelo de agregação.

O CAM é baseado em duas especificações da IMS GLC: Content Packaging v1.1.4 e o Simple Sequencing v1.0. A primeira é utilizada para definir as atividades de aprendizagem (*Learning Activity*) e a segunda para projetar as condições de satisfação para uma atividade ser iniciada ou encerrada.

O modelo é composto por entidades sintáticas em três níveis de granularidade:

- *assets*: os **recursos** são a menor unidade dentro do contexto instrucional. Cada recurso corresponde a um arquivo que será entregue ao usuário no seu navegador. Por exemplo, documentos HTML, arquivos de imagem (JPG, PNG), arquivos de áudio (WAV, MP3) e qualquer outro que possa ser executado num navegador web;

- *Sharable Content Object* (SCO): os **objetos compartilháveis de conteúdo** agrupam os *assets* para representar uma atividade completa com o usuário. Os SCOs são indivísiveis e completos. Isso implica que esses objetos devem evitar se referenciarem para promover a reusabilidade desses objetos. Os SCOS é que se comunicam com o sistema. É claro que a comunicação está implementada nos recursos, mas o ciclo da interação é delimitado pelo SCO. Esses objetos precisam sinalizar o sistema quando iniciam e finalizam. Assim todas as interações entre esses dois momentos serão representados como pertencendo ao mesmo SCO;
- *Content Organization*: a **organização de um conteúdo** representa um conjunto de SCOs, o que se subentende como sendo uma unidade instrucional de maior nível, por exemplo, uma disciplina, uma unidade do currículo ou mesmo uma aula, pois os objetos de conteúdo são representações de uma unidade pequena de conteúdo, com um objetivo bem determinado e uma aula poderia abranger mais de um objeto desses, tanto para completar o assunto, como para reforçar o aprendizado.

Essas três partes são fisicamente mapeadas dentro de um documento XML de nome `imsmanifest.xml`. Além disso, existem duas unidades conceituais:

- *Learning Activity*: as **atividades de aprendizagem** correspondem aos SCOs sendo executados, ou seja, representam as instâncias dos SCOs para um único usuário. Uma atividade pode corresponder a um SCO ou um conjunto de SCOs. Uma atividade que contém outras atividades é chamada de *Cluster*. Uma atividade que não possui subatividades é chamada de atividade folha e somente essas atividades são relacionadas aos SCOs;
- *Content Aggregation*: o **conteúdo agregado** representa a união do arquivo de manifesto com os arquivos de recursos necessários para a execução do processo de aprendizagem.

A Figura 2 representa o conteúdo agregado, ou seja, a relação entre as atividades e os recursos. O elemento `<item>` se refere às atividades de aprendizagem e somente os itens folha estão associados aos recursos, que por sua vez podem ser representados por SCOs ou *assets*. Os SCOs são compostos de *assets.*

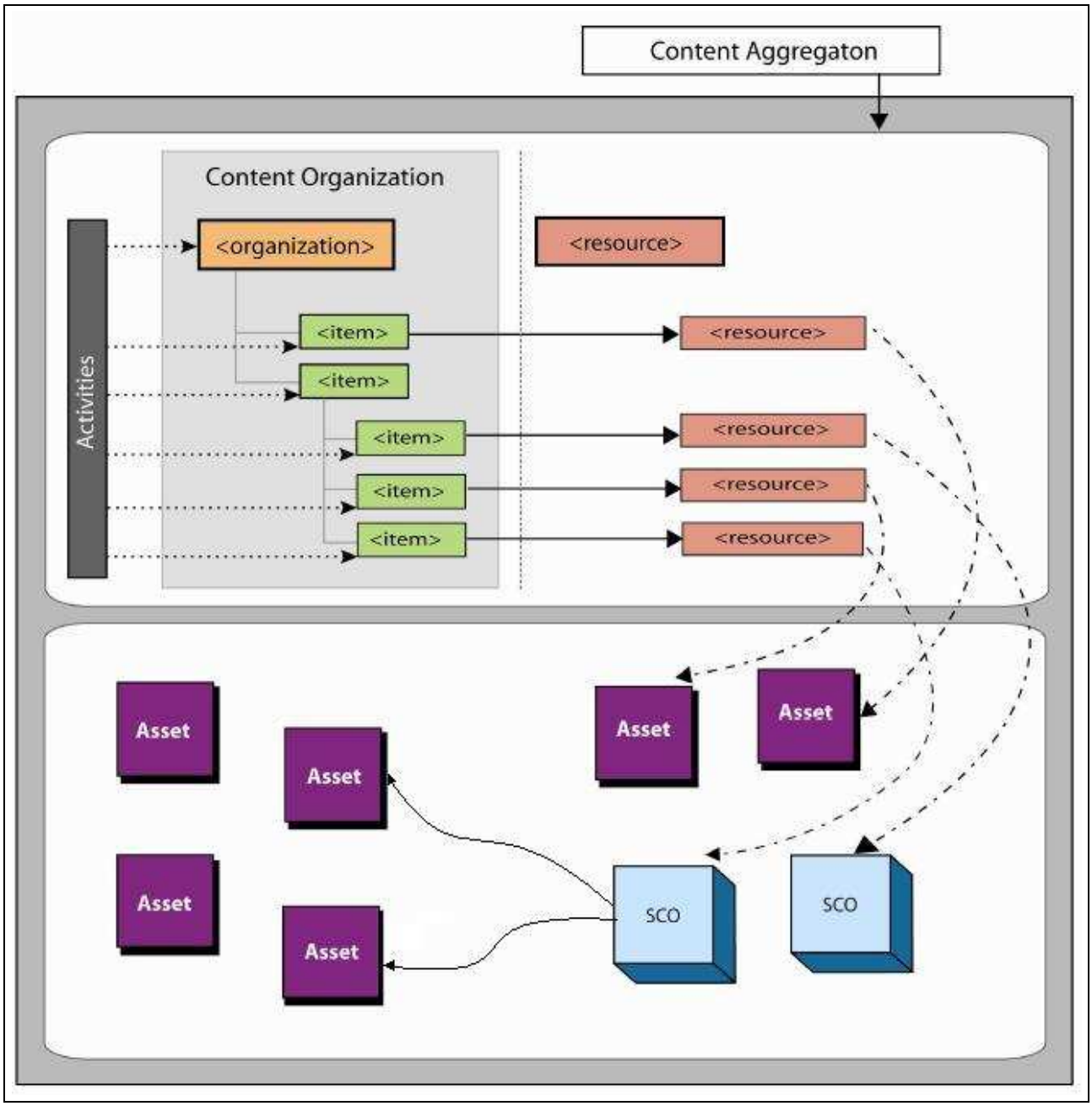


Figura 2. Conteúdo Agregado

Fonte: ADL (2006a, p. 26).

O Seqüenciamento e Navegação (SN) é a possibilidade de criar regras para condicionar a disponibilização das atividades ao aluno (ADL, 2006d). Por exemplo, uma determinada atividade somente pode ser visível se outro conjunto de atividades for concluído, ou o aluno obteve no mínimo 60% de acerto num questionário. O SN determina a existência de valores em memória que são alterados e acessados durante a interação do usuário. O SCORM estabelece as seguintes possibilidades de configuração do seqüenciamento (ADL, 2006d):

1. **Controle de Requisição:** como se comporta um grupo de atividades em relação às requisições de navegação do usuário. Uma requisição é a seleção de uma atividade que parte do usuário, seja clicando diretamente na atividade, ou pressionando um botão de próximo ou anterior. As configurações de controle de navegação determinam o fluxo de navegação, por exemplo, se pode voltar uma atividade, ou escolher uma atividade dentro de um grupo de atividades;
2. **Regras de Seqüenciamento:** é um conjunto de regras que podem ser usadas para habilitar/desabilitar ou mostrar/esconder uma atividade avaliando o estado das outras atividades. Por exemplo, uma atividade só é apresentada se outra atividade em específico foi concluída;
3. **Restrições de Acesso:** usado para limitar quantas vezes ou por quanto tempo uma atividade pode ser acessada;
4. **Regras de Retropropagação:** indica se o desempenho das subatividades reflete nas suas atividades de nível superior na árvore. Por exemplo, se uma atividade completa deve refletir que a atividade de nível acima também deve ser considerada completa;
5. **Descrição de Objetivos:** um objetivo é um recurso de uma atividade, visto como uma pequena área de dados, que o projetista instrucional se utiliza para criar o seqüenciamento das atividades. Por exemplo, o objetivo em uma atividade pode ser a quantidade de páginas visitadas dentro dela, ou a pontuação que o usuário obteve com os testes. Outras atividades podem ter regras de seqüenciamento que se baseiam no objetivo dessa atividade. Por exemplo, uma atividade “A1” armazena o resultado de um conjunto de questões em um de seus objetivos; outra atividade “A2” pode iniciar como escondida, e quando o objetivo da atividade “A1” chegar a 70%, ela então pode estar disponível ao usuário.

Uma atividade pode ter um ou mais objetivos. Esses objetivos podem ser primários ou secundários. Cada atividade tem somente um objetivo primário, e o valor dele é alimentado automaticamente pelo LMS, com a mensuração do desempenho do aluno nos exercícios e com a informação se a atividade foi visitada por completa. Os objetivos secundários são aqueles em que atividades podem manipular. Além disso, existem os objetivos locais e os globais. Os objetivos locais tem como escopo de visibilidade a própria atividade. As páginas das atividades conseguem acessar somente esse tipo de objetivo. O objetivo local precisa ser mapeado para um objetivo global

se outra atividade quiser acessá-lo. Existem duas possibilidades de relacionamento entre um objetivo local com o global: o local pode obter seu valor de um global, e/ou, o local pode gravar em um objetivo global. Por exemplo, se o objetivo local “Obj1” de uma atividade precisa ser acessado por outra atividade, então mapeia-se o objetivo “Obj1” como gravação para um objetivo global “Geral1”; na outra atividade, pode-se mapear um novo objetivo local “Obj2” para ler do global “Geral1”.

6. **Controle de Aleatoriedade:** com essas opções se configura para que o sistema possa selecionar aleatoriamente um conjunto de atividades.

Todas essas configurações permitem um conteúdo que seja selecionado dinamicamente de acordo com as condições das próprias atividades. A Figura 3 apresenta um modelo do processo de SN.

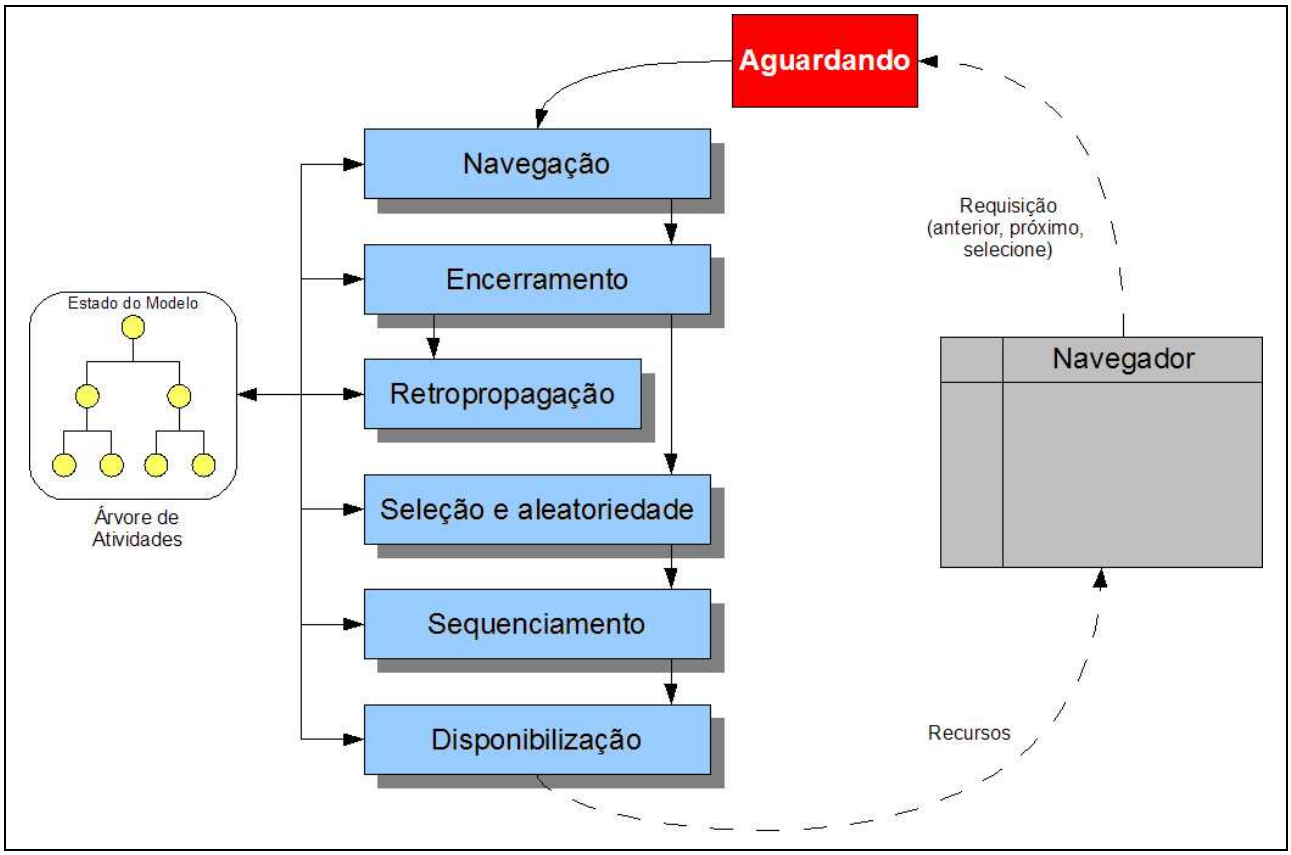


Figura 3. Modelo conceitual do processo completo de seqüenciamento

Fonte: Adaptado de ADL (2006d, p. 102).

Do lado esquerdo da figura existe uma estrutura de árvore que o modelo se refere à árvore de atividades. Essa árvore é instanciada a partir do pacote de conteúdo, acessada e alterada durante o processo de interação. Como o SCORM é uma especificação voltada para aplicações web, o processo considera dessa forma: o usuário a partir do navegador interage com o sistema (representado pela requisição), e o sistema lhe devolve o conteúdo (representado pelos recursos).

Quando o sistema recebe uma requisição, por exemplo, a seleção de uma atividade, ou a continuação ou retrocesso de uma atividade, ele trata essas requisições na fase de navegação. Neste momento é decidido o encerramento de atividades ou sessões e a seleção da próxima atividade.

O encerramento não implica necessariamente no término de uma atividade, mas a marcação de término na utilização de um SCO, que por sua vez pode indicar o término de uma atividade. Um SCO pode ser encerrado antes de concluí-lo. Uma sessão é o intervalo entre o início e o término de utilização de um SCO. Então o encerramento de uma sessão pode ou não afetar o estado das atividades, ou no caso das regras de retropropagação, o novo estado de uma atividade pode se propagar para as atividades de nível superior.

A fase de seleção e aleatoriedade analisa o estado atual das atividades e reorganiza a ordem de apresentação das atividades. A fase de seqüenciamento utiliza os dados da requisição junto da ordem de apresentação das atividades e decide pela próxima atividade. Por último, o processo de disponibilização decide pelos recursos a serem enviados ao usuário baseado na atividade selecionada.

A definição do controle desse seqüenciamento, assim como a estrutura do conteúdo, é feita através da criação de um arquivo XML. A Figura 4 apresenta o trecho do arquivo de manifesto de um curso disponível no próprio site da ADL. O elemento raiz é o `<manifest>`. Esse contém três elementos: `<metadata>` que são informações para descrever o próprio arquivo de manifesto; `<organizations>` que representa a estrutura do conteúdo agregado; e `<resources>` com a lista de SCOs e recursos que compõe os SCOs.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest ...>
  <metadata>
    <schema>ADL SCORM</schema>
    <schemaversion>2004 3rd Edition</schemaversion>
  </metadata>
  <organizations default="ORG-EC37F452-C3E9-DE62-A307-41F607300701">
    <organization identifier="ORG-EC37F452-C3E9-DE62-A307-41F607300701">
      <title>SCORM 2004 3rd Edition Plug-in Technologies Example</title>
      <item identifier="ITEM-WELCOME" identifierref="RES-WELCOME">
        <title>Welcome</title>
        <adlnav:presentation>
          <adlnav:navigationInterface>
            <adlnav:hideLMSUI>previous</adlnav:hideLMSUI>
            <adlnav:hideLMSUI>continue</adlnav:hideLMSUI>
          </adlnav:navigationInterface>
        </adlnav:presentation>
        <imsss:sequencing>
          <imsss:controlMode choiceExit="false" />
          <imsss:objectives>
            ...
          </imsss:objectives>
        </imsss:sequencing>
      </item>
      <item identifier="ITEM-DIRECTOR" identifierref="RES-DIRECTOR">
        <title>Director</title>
        ...
      </item>
    </organization>
  </organizations>
  <resources>

    <resource identifier="RES-DIRECTOR" adlcp:scormType="sco" type="webcontent"
href="Director%20Example/dswmedia/pluginexample.htm">
      ...
      <dependency identifierref="SHARED_FILES" />
    </resource>
    <resource identifier="RES-WELCOME" href="Welcome/welcome_01.html"
adlcp:scormType="sco" type="webcontent">
      <file href="Welcome/welcome_01.html" />
      <file href="Welcome/welcome_02.html" />
      ...
      <file href="Welcome/welcome_08.html" />
      <dependency identifierref="SHARED_FILES" />
    </resource>
  </resources>
</manifest>
```

Figura 4. Exemplo de arquivo manifesto

O elemento `<organizations>` contém nenhum ou vários elementos `<organization>`. Esse elemento representa a organização de um conteúdo, e por sua vez contém um ou mais elementos `<item>` que representa uma atividade de aprendizagem. A existência do atributo `identifierref` no elemento `<item>` indica que ele é uma atividade folha. Esse atributo relaciona a atividade com um recurso que pode ser um SCO ou um *asset*. Nesse exemplo não está representado, mas um `<item>` pode conter outros elementos `<item>`. Nesse caso o `<item>` mais externo não tem associação com recursos.

A definição do sequenciamento é feita dentro de cada elemento de atividade de aprendizagem, através do elemento `<imsss:sequencing>`.

Dentro desse elemento, as possibilidades de configuração são:

1. **controle de requisição:** indicam como as requisições de navegação podem afetar um *cluster*. Pode-se configurar que seja proibida a seleção de uma atividade fora do *cluster*, ou que não possa solicitar o encerramento de uma atividade, ou se pode retornar uma atividade já concluída. Na Figura 5 está representado que não é possível selecionar as atividades em qualquer ordem (choice="false"), que o usuário não pode selecionar outras atividades fora do *cluster* "PRETEST1" (choiceExit="false"), que a execução é sequencial (flow="true") e o usuário não pode retroceder as atividades (forwardOnly="true");

```
<item identifier="PRETEST1">
   <title>Module 1 --  Pretest</title>
   <item identifier="PRETEST_QUESTION1" isvisible = "false"
         identifierref="RESOURCE QUESTION1">
      <title>Question 1</title>
   </item>
   <item identifier="PRETEST QUESTION2" isvisible = "false"
         identifierref="RESOURCE QUESTION2">
      <title>Question 2</title>
   </item>
   <imsss:sequencing>
      <imsss:controlMode choice="false" choiceExit="false"
            flow="true" forwardOnly = "true"/>
   </imsss:sequencing>
</item>
```

Figura 5. Exemplo de controle de requisição
Fonte: ADL (2006a, p. 186).

2. **regras de seqüenciamento:** é a montagem de regras (no estilo se-então) que são testadas e executadas antes, após e na saída de uma atividade. Por exemplo, é possível esconder e mostrar atividades de acordo com a satisfação de um objetivo de aprendizagem, que pode ser mensurado e comparado (p.e. a quantidade de acertos do aluno foi maior que 70%). A Figura 6 contém um exemplo onde a atividade PRETEST1 é desabilitada (action="disabled") se ela estiver satisfeita (condition="satisfied");

```
<item identifier="PRETEST1">
   <title>Module 1 --  Pretest</title>
   <item identifier="PRETEST QUESTION1" isvisible = "false"
         identifierref="RESOURCE_QUESTION1">
      <title>Question 1</title>
   </item>
   <item identifier="PRETEST_QUESTION2" isvisible = "false"
         identifierref="RESOURCE QUESTION2">
      <title>Question 2</title>
   </item>
   <imsss:sequencing>
      <imsss:sequencingRules>
         <imsss:preConditionRule>
            <imsss:ruleConditions>
               <imsss:ruleCondition condition = "satisfied"/>
            </imsss:ruleConditions>
            <imsss:ruleAction action = "disabled"/>
         </imsss:preConditionRule>
      </imsss:sequencingRules>
   </imsss:sequencing>
</item>
```

Figura 6. Exemplo de regras de seqüenciamento

Fonte: ADL (2006a, p. 189).

3. **restrições de acesso:** de tentativas e de tempo. É possível configurar para que uma atividade fique disponível durante uma quantidade de tempo, ou até atingir uma quantidade de tentativas. Na Figura 7 é definida que a atividade somente pode ser feita uma única vez (attemptLimits=“1”);

```
<item identifier="INTRO" identifierref="RESOURCE INTRO">
   <title>Photoshop Introduction</title>
   <imsss:sequencing>
      <imsss:limitConditions attemptLimit="1"/>
  </imsss:sequencing>
</item>
```

Figura 7. Exemplo de restrições de acesso

Fonte: ADL (2006a, p. 196).

4. **regras de retropropagação:** são as regras para indicar o estado de uma atividade baseado nas subatividades A Figura 8 indica que, se em todas as atividades (childActivitySet=“all”) de PRETEST2 o usuário ter feito pelo menos uma tentativa (condition=“attempted”) então a atividade MODULE2 deve ser considerada completa (action=“completed”);

```
<item identifier="MODULE2">
   <title>Module 2 -- Enhancing Images</title>
   <item identifier="PRETEST2">
      <title>Module 2 -- Pretest</title>
       <item identifier="PRETEST QUESTION4" isvisible = "false"
             identifierref="RESOURCE QUESTION4">
          <title>Question 4</title>
       </item>
       <item identifier="PRETEST QUESTION5" isvisible = "false"
             identifierref="RESOURCE_QUESTION5">
          <title>Question 5</title>
       </item>
       <item identifier="PRETEST_QUESTION6" isvisible = "false"
             identifierref="RESOURCE QUESTION6">
          <title>Question 6</title>
       </item>
       <imsss:sequencing>
          <imsss:rollupRules >
             <imsss:rollupRule childActivitySet = "all">
                <imsss:rollupConditions>
                   <imsss:rollupCondition condition = "attempted"/>
                </imsss:rollupConditions>
                <imsss:rollupAction action = "completed"/>
             </imsss:rollupRule>
          </imsss:rollupRules>
       </imsss:sequencing>
    </item>
</item>
```

Figura 8. Exemplo de configuração de regras de retropropagação
Fonte: ADL (2006a, p. 200).

5. **descrição de objetivos:** é possível associar objetivos com as atividades, e existem várias condições que são satisfeitas quando os objetivos atingem um valor determinado. Na está representado que o objetivo PRIMARYOBJ é considerado satisfeito baseado na mensuração dele (satisfiedByMeasure=“true”) e quando ele atingir 60% (minNormalizedMeasure=“0.6”). Também é configurado que as atividades podem alterar o *status* (writeSatisfiedStatus=“true”) do objetivo “obj_module_1” mas não podem ler o valor numérico dele (readNormalizedStatus=“false”);

```
<item identifier="POSTTEST1">
   <title>Module 1 --  Posttest</title>
   <item identifier="POSTTEST QUESTION1" isvisible = "false"
         identifierref="RESOURCE QUESTION1">
      <title>Question 1</title>
   </item>
   <item identifier="POSTTEST QUESTION2" isvisible = "false"
         identifierref="RESOURCE QUESTION2">
      <title>Question 2</title>
   </item>
   <item identifier="POSTTEST QUESTION3" isvisible = "false"
         identifierref="RESOURCE QUESTION3">
      <title>Question 3</title>
   </item>
   <imsss:sequencing>
      <imsss:objectives>
         <imsss:primaryObjective objectiveID = "PRIMARYOBJ"
                satisfiedByMeasure = "true">
            <imsss:minNormalizedMeasure>0.6</imsss:minNormalizedMeasure>
            <imsss:mapInfo targetObjectiveID = "obj_module_1"
                      readNormalizedMeasure = "false"
                      writeSatisfiedStatus = "true" />
         </imsss:primaryObjective>
      </imsss:objectives>
   </imsss:sequencing>
</item>
```

Figura 9. Exemplo de configuração dos objetivos de uma atividade

Fonte: ADL (2006a, p. 211).

6. **controle de aleatoriedade:** a Figura 10 demonstra como configurar para que a seleção das atividades seja aleatória. No elemento `<randomizationControls>` está configurado para que a cada nova tentativa (selectionTiming=“onEachNewAttempt”) serão selecionadas duas atividades da lista do POSTTEST1 (selectCount=“2”).

```
<item identifier="POSTTEST1">
   <title>Module 1 --  Posttest</title>
      <item identifier="POSTTEST QUESTION1" isvisible = "false"
            identifierref="RESOURCE_QUESTION1">
         <title>Question 1</title>
      </item>
      <item identifier="POSTTEST_QUESTION2" isvisible = "false"
            identifierref="RESOURCE QUESTION2">
         <title>Question 2</title>
      </item>
      <item identifier="POSTTEST QUESTION3" isvisible = "false"
            identifierref="RESOURCE QUESTION3">
         <title>Question 3</title>
      </item>
   <imsss:sequencing>
      <imsss:randomizationControls selectCount="2"
                                   selectionTiming="onEachNewAttempt" />
   </imsss:sequencing>
</item>
```

Figura 10. Exemplo de configuração de aleatoriedade
Fonte: ADL (2006a, p. 215).

A comunicação entre o SCO e o LMS é determinada pelo RTE. Ele segue os padrões Std 1484.11.1-2004 e Std 1484.11.2-2003 do IEEE LTSC. Esses padrões determinam como o SCO se comunicará utilizando JavaScript, e quais são os nomes dos valores a serem trocados entre SCO e LMS.

A Figura 11 apresenta as interfaces que o SCO deve considerar como disponíveis. Uma instância da API precisa ser criada antes do SCO ser visualizado. Essa instância precisa ter o nome API_1484_11. Existem três possibilidades de implementar o lado cliente do LMS para exposição de um SCO: utilizando frames, abertura de uma nova janela (janela popup) ou ambas. Em qualquer uma delas, a instância precisa ser localizada, e existe um algoritmo disponibilizado pelo IEEE para a localização dessa instância.

As funções da API podem ser classificadas em três tipos (ADL, 2006c):

- sessões: que delimitam o início e fim da comunicação entre SCO e LMS;
- transferência de dados: com a finalidade de enviar e solicitar dados entre SCO e LMS;
- suporte: são funções para capturar o estado do LMS em relação ao SCO.

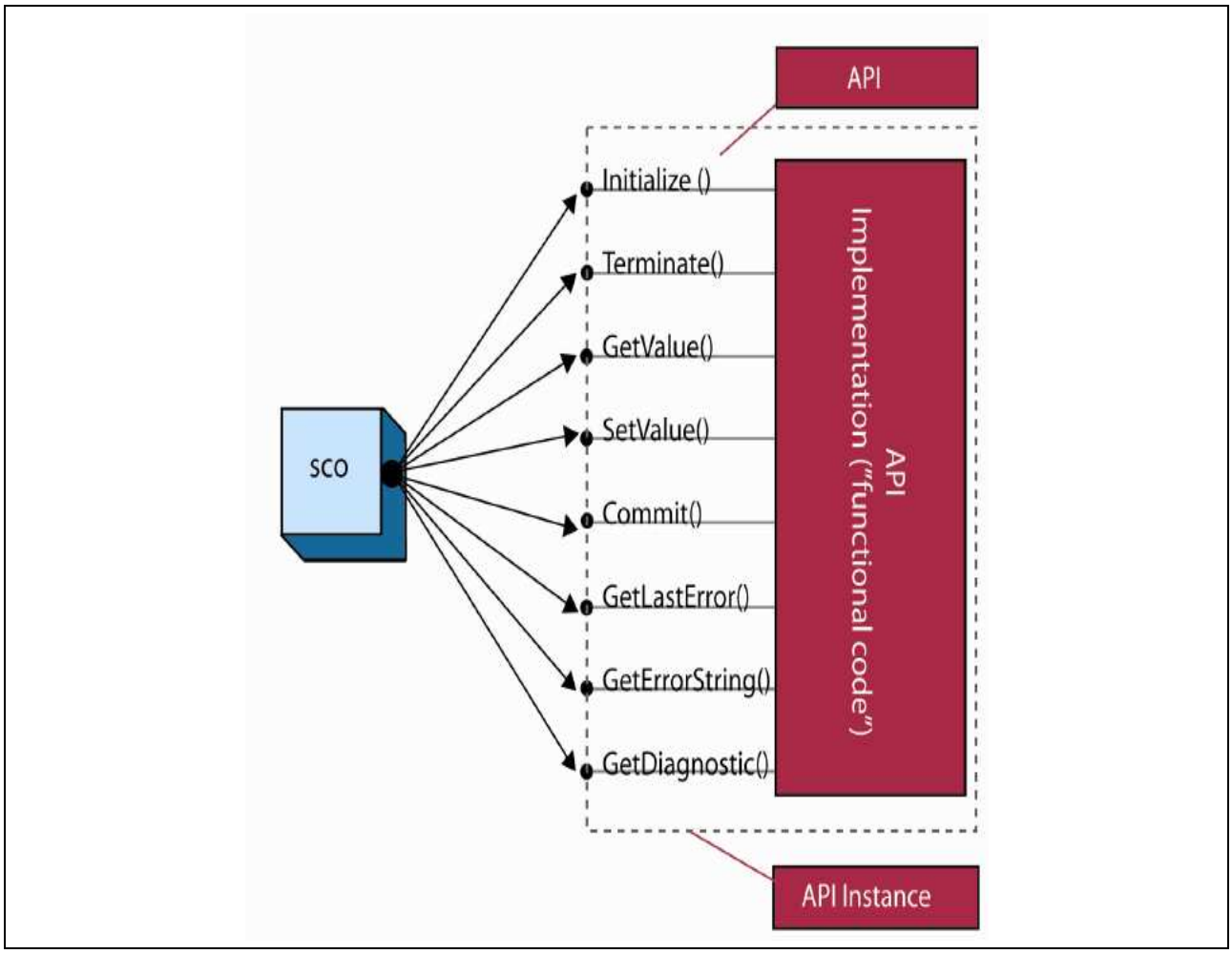


Figura 11. API de comunicação SCO e LMS
Fonte: ADL (2006c, p. 32).

Os métodos de sessão são:

- status = Initialize(“”): para iniciar uma sessão de comunicação com o LMS. O valor de retorno pode ser “true” ou “false”;
- status = Terminate(“”): encerra a sessão com o LMS.

Os métodos de transferência são:

- return_value = GetValue(*Data Model Element*): solicita um valor para o LMS. O parâmetro *Data Model Element* identifica o valor a ser solicitado. Por exemplo, para descobrir a identificação do usuário atual, utiliza-se como parâmetro `cmi.learner_id`;

- return_status = SetValue(*Data Model Element*, *value*): solicita ao LMS para armazenar um valor no elemento identificado no primeiro parâmetro, e retorna como texto o resultado da operação: “true” ou “false”. Por exemplo, status = SetValue(“`cmi.completion_status`”, “completed”), define que o SCO pode ser considerado como completado pelo usuário:
- return_status = Commit(“”): exige ao LMS para armazenar todos os valores solicitados através do método SetValue.

Os métodos de suporte são:

- error_code = GetLastError(): retorna um número inteiro entre 0 e 65535 representando o erro atual no LMS;
- error_text = GetErrorString(“número do erro”): retorna o texto representando o número do erro passado como parâmetro; e
- diagnostic_text = GetDiagnostic(“parâmetro”): retorna informações sobre o diagnóstico detalhado de alguma situação do LMS. Por exemplo, pode ser passado como parâmetro o código de um erro e o LMS retornar mais detalhes sobre o erro.

Segundo ADL (2006c), o *Data Model Element* é um identificador de alguma informação que o SCO possa acessar ou alterar através do LMS. A definição desses identificadores garante uma forma padrão de comunicação entre ambos. O SCORM institui 27 identificadores:

1. `cmi._version`: retorna a versão do *Data Model* que o LMS suporta. A versão atual é 1.0;
2. `cmi.comments_from_learner`: permite que o usuário acesse ou forneça comentários sobre o SCO;
3. `cmi.comments_from_lms`: devolve comentários armazenados no LMS para o SCO corrente;
4. `cmi.completion_status`: define se a interação do usuário com o SCO pode ser considerada como completa ou não;

5. `cmi.completion_threshold`: determina o valor de `cmi.progress_measure` para ser considerado como completo;

6. `cmi.credit`: retorna se o usuário pode acessar o SCO;

7. `cmi.entry`: indica se é a primeira vez que o SCO está sendo acessado, ou se é a continuação de uma sessão anterior;

8. `cmi.exit`: é o modo de saída do SCO, se normal ou suspenso;

9. `cmi.interactions`: esse *Data Model* indicado para manter o histórico dos dados que o usuário forneceu ao SCO, por exemplo, em atividades de exercício ou avaliação;

10. `cmi.launch_data`: são dados que o SCO pode utilizar para usa inicialização. São definidos no pacote de conteúdo;

11. `cmi.learner_id`: representa um identificador (por exemplo, um código, uma chave, ou a matrícula) do usuário corrente;

12. `cmi.learner_name`: identifica o nome do usuário corrente;

13. `cmi.learner_preference`: são as preferências do usuário em relação ao áudio, idioma e velocidade de acesso;

14. `cmi.location`: é utilizado pelo SCO como marcador de páginas;

15. `cmi.max_time_allowed`: é a quantia de tempo que o usuário tem para interagir com o SCO;

16. `cmi.mode`: solicita o modo que o SCO será apresentado: se vai ou não registrar as interações com o usuário, ou se vai apresentar uma interação passada;

17. `cmi.objectives`: são objetivos disponíveis no SCO, e utilizados nas regras de seqüenciamento. Os objetivos podem ser testados se estão ou não completos e se estão ou não aprovados;

18. `cmi.progress_measure`: determina o valor de progresso do usuário em relação a quantidade de conteúdo que ele acessou do SCO;

19. `cmi.scaled_passing_score`: é a pontuação que o usuário precisa alcançar para ser considerado aprovado no SCO;

20. `cmi.score`: é a pontuação que o usuário obteve no SCO;

21. `cmi.session_time`: é o tempo consumido pelo usuário na interação corrente do SCO;

22. `cmi.success_status`: indica se o usuário foi aprovado pelo SCO;

23. `cmi.suspend_data`: permite que o SCO armazene e recupere informações caso a interação seja suspensa;

24. `cmi.time_limit_action`: obtém a informação do que o SCO precisa fazer quando `cmi.max_time_allowed` é excedido;

25. `cmi.total_time`: é a soma de todos os valores `cmi.session_time` na sessão corrente do usuário;

26. `adl.nav.request`: determina uma requisição de navegação que deve ser executada quando o SCO encerrar. Por exemplo, ele pode solicitar para ir ao SCO anterior ou posterior;

27. `adl.nav.request_valid`: retorna se uma determinada requisição de navegação pode ser executada.

2.3 Engenharia de Software Baseada em Componentes (ESBC)

O componente é um módulo independente que provê informações sobre o que ele faz e como usá-lo através de suas interfaces públicas, enquanto encapsula as suas tarefas internas (MCCLURE, 2001, p. 4; BARROCA *et al*, 2005, p. 4). Conforme esses autores, algumas características básicas são requisitadas para que um componente atenda os seus princípios básicos:

- interface bem definida: como elas definem os serviços que o componente atende, elas devem ser claras, explicitas e fornecer uma documentação;
- facilidade de ser substituído: a interação entre os componentes deve ser única e exclusivamente através de suas interfaces. A sua especificação precisa ser clara sobre suas funcionalidades e relações com outros componentes; e
- interoperabilidade com outros componentes: aqui entram os serviços de infra-estrutura do ciclo de vida e comunicação dos componentes, que é referenciada como *middleware* (BARROCA *et al*, 2005, p. 21). Existem padrões abertos (como o CORBA da OMG) e proprietários (COM/DCOM da Microsoft e RMI do Java da SUN).

O conceito de componentes é muitas vezes confundido, ao de objetos. A diferença marcante entre ambos está em sua granularidade. Um objeto atende a uma responsabilidade bem limitada, abstrai um conceito menor, deve prover alta coesão e baixo acoplamento (LARMAN, 2004). Em contrapartida, um componente encapsula um número maior de responsabilidades e funcionalidades sendo, portanto, maiores (MCCLURE, 2001, p. 5). Eles podem requerer interação de muitos outros componentes para atingir seus objetivos.

Para esclarecer essa diferença, será exemplificado o módulo de Diário de Classe de um ambiente de aprendizagem. Quando se fala em objetos (e classes), as necessidades envolvidas nesse módulo são divididas em vários conceitos: aluno, dia, disciplina, avaliação, regras de distribuição de pesos para a nota final, entrada de dados, persistência dos dados, etc; e cada um desses conceitos é atendido por uma ou mais classes. A equipe de desenvolvimento vai se preocupar no projeto, implementação e utilização dessas classes. Os projetos futuros podem (e deveriam) reutilizá-las. O usuário das classes tem acesso a cada uma delas, podendo estendê-las, ou rearranjá-las da forma que melhor convir. Ele monta a aplicação com essas pequenas partes.

Numa visão de componente, existe o módulo “Diário de Classe” como uma única entidade. Ela possui uma interface que o montador da aplicação vai usufruir para acoplá-la no sistema. Nessa interface constam configurações de cálculo de aprovação, as regras de distribuição dos pesos, como conectar-se a um banco de dados para armazenar os dados do módulo, etc. O usuário do componente tem acesso ao módulo somente através dessas interfaces. Ele não consegue estender o componente, ou modificar alguma lógica, se esta não está fornecida através das interfaces.

Enquanto a montagem de uma aplicação com objetos é um trabalho enfadonho, pois requer a disposição e ligação de pequenos blocos para realizar uma funcionalidade, a montagem com componentes é mais produtiva, porque se juntam blocos completos para a mesma tarefa, inclusive, o próprio bloco já pode ter toda a funcionalidade inerente.

Portanto, um componente é um conjunto de artefatos de software (p.e., classes, bancos de dados, arquivos de mídia) encapsulados, e que fornece uma interface para publicar os serviços que ele oferece, assim como que serviços que ele também requer para executar as suas funcionalidades.

A ESBC é muitas vezes confundida com o próprio desenvolvimento baseado em componentes (DBC). Esse trabalho utilizará a definição de Pressman (2002), onde a ESBC é dividida pela modelagem de domínio (ou engenharia de domínio (ED)) e modelagem de especificação (ou DBC).

Na ED existem duas perspectivas (WERNER e BRAGA, 2005, p. 60): na primeira se pretende entender o problema de negócio, definindo modelos conceituais, se situando sobre as características do negócio. Na segunda, o objetivo é conhecer sistemas e aplicações já existentes sobre o domínio dessa aplicação, além de “identificar, construir, catalogar e disseminar um conjunto de componentes de software” (PRESSMAN, 2002, p. 707) sobre esse domínio.

Essa engenharia é composta de duas atividades: análise, que define os requisitos, delimita o domínio, define características naquele domínio; e projeto, onde se generaliza uma arquitetura de software para atender as especificações do domínio, delimitando o quão reutilizável será o componente, o que implica na definição das interfaces.

A implementação é responsabilidade do DBC. Nesse processo buscam-se por componentes para utilizá-los, ou se desenvolvem novos. Então o DBC é composto pelas tarefas de (PRESSMAN, 2002, p. 714): (i) qualificação de componentes (analisá-lo comparando com os requisitos funcionais e não-funcionais, se integrável com as ferramentas de desenvolvimento legadas, etc), (ii) adaptação de componentes (com três possibilidades: alterar o código interno, utilizar alguma API ou mecanismo de extensão do componente, e introdução de pré ou pós processamentos das interfaces. Se não houver a adaptabilidade, recorre-se a implementação de um novo componente), (iii) composição de componentes (juntar os componentes para atender a arquitetura estabelecida na ED), e (iv) interoperação (adequação dos componentes dentro de um middleware).

Em relação à modelagem de componentes, a UML 2.0 apresentou mudanças significativas na modelagem do diagrama de componentes em relação à versão 1.4 (PENDER, 2004, p. 425; FOWLER, 2005, p. 135). Conforme Fowler (2005), nas versões anteriores os componentes eram tratados como DLLs, e nessa nova versão eles são considerados artefatos.

Pender (2004) comenta que essas modificações facilitam a especificação das interfaces e de como um componente é composto, e são as seguintes:

- um componente é modelado com a mesma notação de uma classe, adicionando o estereótipo <<component>> ou uma ícone de componente que era utilizado na versão 1.4 (Figura 12);
- é possível representar a dependência entre os componentes, sem necessariamente explicitar as interfaces que eles dependem, através de linhas tracejadas (Figura 12.a);
- as interfaces fornecidas são representadas por um círculo. E as requeridas (ou exigidas) por um meio círculo (Figura 12.b). Essas interfaces também podem ser representadas por classes (Figura 12.c);
- a conexão entre as interfaces de dois componentes é representada por um ícone onde o círculo está dentro do meio círculo (Figura 12.d).

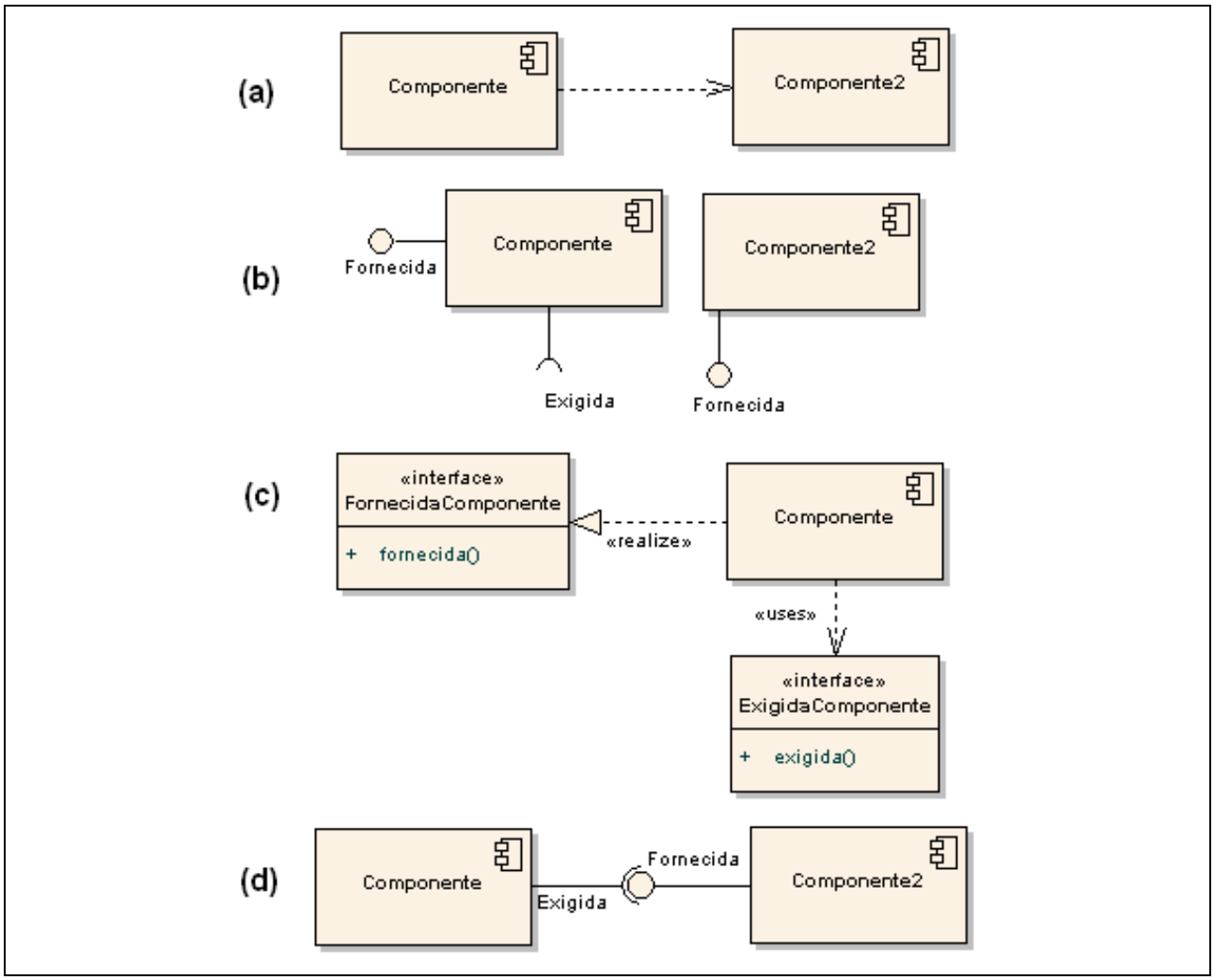


Figura 12. Diagrama de componentes
Fonte: Adaptado de Pender (2004).

2.4 Especificação JEE

A JEE é um padrão que especifica quais os serviços que os servidores devem prover para a produção de aplicações seguras, escaláveis e altamente disponíveis (BOND *et al*, 2003, p. 13). Ela foi lançada e distribuída pela Sun em 1999 (FREIRE e GUAPO, 2004, p. 31) quando se resolveu dividir as versões do Java em três: JSE (*Java Standard Edition*, que contém a API básica de todos os programas em Java), JEE (conjunto de especificações para atender a arquitetura de sistemas corporativos) e JME (*Java Micro Edition*, contendo a API para desenvolvimento de aplicativos móveis, tais como, celulares, handhelds e PDA (*Personal Digital Assistant*)).

O ambiente em que a aplicação será executada dentro do servidor JEE, e por conseqüência utilizar os serviços de infra-estrutura, é chamado de contêiner. A arquitetura de um servidor JEE é

dividida em dois contêineres (BOND *et al*, 2003, p. 14; YUNG, 2004, p. 29), como ilustrado na Figura 13:

- Contêiner WEB: nesse, os componentes interagem com o cliente utilizando o protocolo HTTP. Exemplos: Servlets, JSP e WebServices. O cliente mais utilizado para interagir com esses componentes são os navegadores;
- Contêiner EJB (*Enterprise Java Bean*): são aceitos três tipos de componentes (ou EJBs): (i) *beans* de sessão, para executarem tarefas de controle das operações de negócios; (ii) *beans* de entidade, responsáveis por persistir os objetos; e (iii) *beans* de mensagem, que permitem troca de mensagens assíncronas entre os componentes. Para acessar os componentes desse contêiner, é preciso implementar clientes específicos.

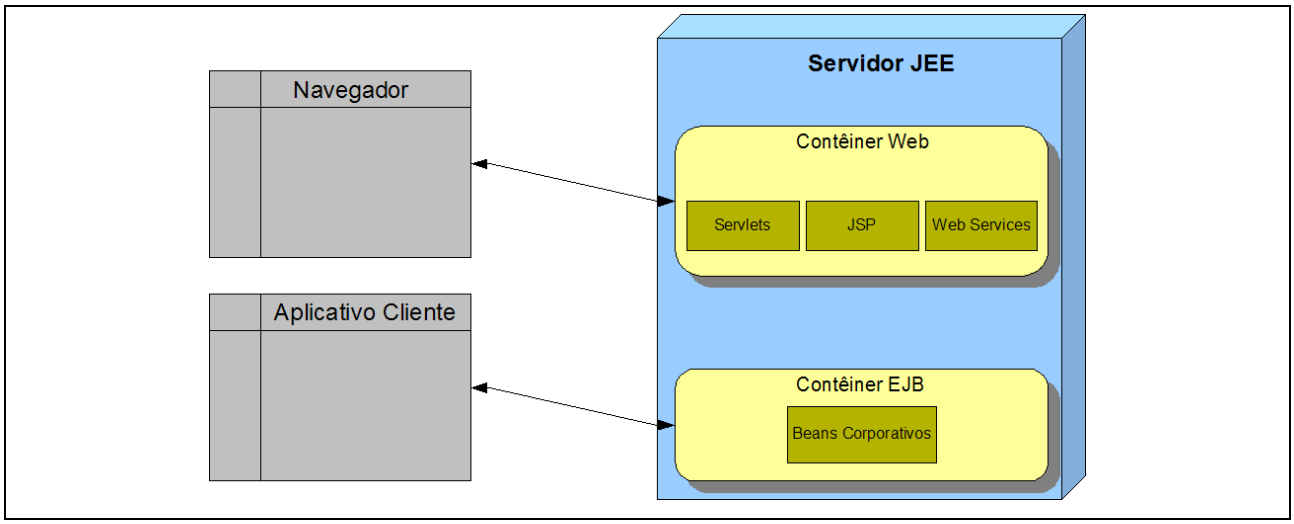


Figura 13. Arquitetura JEE
Fonte: Adaptado de Adaptado de Bond *et al* (2003, p. 14) e Yung (2004, p. 29).

Os *servlets* são componentes convenientes para a criação de páginas dinâmicas (BOND *et al*, 2003, p.455). Eles interagem com os clientes através do protocolo HTTP: recebem uma requisição do cliente, e retornam uma resposta (normalmente uma página HTML criada por ele). A Figura 14 ilustra esse processo de interação.

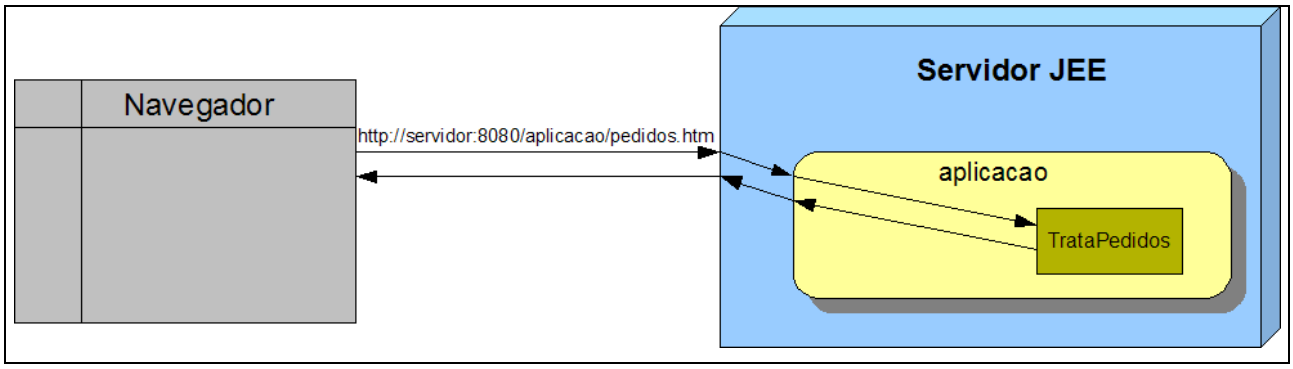


Figura 14. Tratamento de uma requisição por um *servlet*
Fonte: Adaptado de Bond *et al* (2003, p. 455).

Os *servlets* apresentam muitas vantagens em relação às tecnologias concorrentes. Kurniawan (2002, p. 5) cita o desempenho, a portabilidade, a facilidade em (re)utilizar códigos de bibliotecas, robustez e a aceitação difundida pelo mercado. Mas os *servlets* mostram-se falhos quando o assunto é a criação das páginas. Essa tarefa é essencialmente feita por webdesigners, que normalmente não tem (e nem deveriam ter) acesso ao código dos *servlets*.

Para suprir essa dificuldade, foi desenvolvida uma nova tecnologia, para estender as capacidades dos *servlets*, chamada JSP (*Java Server Pages*). As JSPs são páginas HTML dinâmicas. Nelas podem ser incorporados código Java, JavaBeans, biblioteca de *tags*, EL (*Expression Languages*), e outras mais tecnologias JEE.

Um JSP é convertido em *servlet* quando o servidor precisa executar o script, transformando a página em uma classe javax.servlet.HttpJspBase (KURNIAWAN, 2002, p. 224).

A JSTL (*JSP Standard Tag Library*) é uma biblioteca de *tags* que torna mais acessível a criação de páginas JSP (BAYERN, 2003), melhora a legibilidade e facilita a manutenção dessas páginas. Ela encapsula em *tags* funcionalidades comuns a todas as aplicações, como condicionais e laços (ARMSTRONG *et al*, 2004). A Figura 15 mostra o exemplo de duas *tag*s: o condicional <if>, e a saída <out>. Esse código fica exposto na página tal como fosse uma *tag* HTML. A JSTL vem para substituir os *scriptlets*, que são trechos de código Java na página.

```
<c:if test=”${user.education == ‘doctorate’}”>
Dr.
</c:if>
<c:out value=”${user.name}”/>
```

Figura 15. Exemplo de uso da JSTL
Fonte: Adaptado de Bayern (2003, p. 80).

As *tags* customizadas são recursos que encapsulam código dinâmico em *tags* (ARMSTRONG *et al*, 2004). A JSTL é um exemplo de *tags* customizadas. As *tags* customizadas são distribuídas em uma biblioteca de *tags*, que é um arquivo com extensão TLD (*Tag Library Descriptor*), onde se define qual a classe que implementa a *tag*, quais são os atributos e restrições da *tag*. Existem dois tipos de *tags*: a simples, que não tem corpo (p.e. a *tag*
 do HTML), e a clássica, que tem corpo podendo conter textos, outras *tags* e *script*. Para implementar uma *tag* é preciso que a classe seja descendente de javax.servlet.jsp.tagext.SimpleTagSupport ou javax.servlet.jsp.tagext.BodyTagSupport.

Outra tecnologia existente na especificação JEE são os filtros. Eles são recursos disponíveis desde a versão 2.3 (BOND *et al*, 2003, p. 487; KURNIAWAN, 2002, p. 195; LEME, 2005, p. 24). Eles permitem que uma requisição seja processada antes de chegar ao *servlet*. Também é possível interceptar as respostas dos *servlets*. A Figura 16 apresenta o funcionamento do filtro, interceptando as requisições e respostas da aplicação Web.

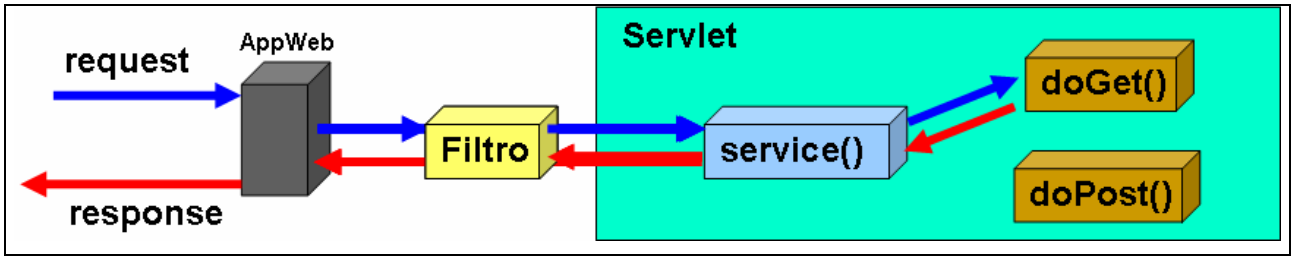


Figura 16. Filtro tratando requisições e respostas
Fonte: Adaptado de Leme (2005, p. 24).

Eles são uma aplicação do padrão *Intercepter Filter* do Core J2EE Pattern (LEME, 2005, p. 24), e ainda utilizam o padrão *Chain of Responsability* (GAMMA *et al*, 2000, p. 212) para encadeamento entre si, já que é permitido que coexistam um ou mais filtros.

Algumas utilidades para os filtros: (i) formatação de dados; (ii) controle de autorização e bloqueio de requisições; (iii) auditoria; (iv) criptografia e decodificação; (v) compressão; (vi) validação e (vii) componentização.

Os filtros podem ser adicionados na aplicação sem necessidade de recompilação ou alteração de quaisquer classes, sejam *servlets* ou não. Eles são desacoplados dos *servlets* e tem vida própria (LEME, 2005, p. 24).

Os filtros implementam a interface javax.servlet.Filter que possui três métodos:

- init(): é invocado quando a aplicação WEB é inicializada;
- destroy(): quando ela a aplicação é finalizada; e
- doFilter(): que é responsável pela tarefa de filtragem e encadeamento dos filtros. Esse método é chamado no momento que a aplicação recebe uma requisição.

Um filtro é exemplificado na Figura 17. Ele mostra na saída de console o nome da URL acessada.

```
1.  package monografia;

2.  // imports

3.  public class MostraURL implements Filter {

4.      public void init(FilterConfig config) throws ServletException { }
5.      public void destroy() throws ServletException {}

6.      public void doFilter(ServletRequest request, ServletResponse response,
              FilterChain chain) throws ServletException {

7.          String url = ((HttpServletRequest)request).getServletPath();
8.          System.out.println(url); // pré-processamento
9.          chain.doFilter(request, response);
10.         // pós-processamento

11.     }

12. }
```

Figura 17. Exemplo de filtro

O pré-processamento é o código antes da linha 9 quando nenhum *servlet* ainda recebeu a solicitação. O pós-processamento é o código após essa linha, quando todos os *servlets* já trataram a solicitação.

Os filtros são adicionados e configurados no arquivo `web.xml` com o elemento `<filter>`. Na Figura 18 é exemplicado o trecho do arquivo `web.xml` necessário para inserir um filtro na aplicação. No exemplo dessa figura foram criados dois mapeamentos (`<filter-mapping>`) para o mesmo filtro: o primeiro mapeia para um *servlet* especifico, e no caso aquele com nome Contador; e o segundo é baseado num padrão para a URL: o filtro é executado sempre que uma requisição com extensão `mon` vir para a aplicação. Por exemplo: http://192.168.200.1/monografia/umteste.mon.

```
   ...
   <filter>
           <filter-name>Mostrar a URL</filter-name>
           <filter-class>monografia.MostrarURL</filter-class>
   </filter>
   <filter-mapping>
           <filter-name>Mostrar a URL</filter-name>
              <servlet-name>Contador</servlet-name>
   </filter-mapping>
   <filter-mapping>
           <filter-name>Mostrar a URL</filter-name>
           <url-pattern>*.mon</url-pattern>
   </filter-mapping>
...
```

Figura 18. Mapeamento para o filtro

A execução dos filtros vai de acordo com a ordem em que eles estão definidos no arquivo `web.xml` (BOND *et al*, 2003, p. 487; LEME, 2005, p. 25)

2.5 TRABALHOS CORRELATOS

Essa dissertação apresenta mecanismos para que conteúdo SCORM possa ser gerenciado e adaptado pelos LMS. Visando elencar os requisitos que um ambiente precisa contemplar para satisfazer o primeiro objetivo (de gerenciar conteúdo), estudaram-se dois LMS: de referência da ADL e o Moodle. Para o segundo objetivo (adaptação do conteúdo), buscaram-se trabalhos produzidos e publicados por outros autores. Essa seção foi dividida em duas partes: na primeira são descritos os LMS, e na segunda são apresentados os trabalhos focados em melhorar a especificação SCORM para facilitar o seu uso e adaptação.

2.5.1 LMS que gerenciam conteúdo SCORM

Os ambientes selecionados para relacionar os requisitos de gerenciamento de conteúdo SCORM foram o SCORM 2004 3rd Edition Sample Run-Time Environment (ADL, 2008) e o Moodle (MOODLE, 2008). O primeiro foi selecionado por tratar-se da implementação de referência

do SCORM e nele são exemplificados o RTE e o SN. O Moodle foi adotado por ser um dos ambientes de aprendizagem livre mais popular e utilizado, e contém um *plugin* para execução de pacotes SCORM.

2.5.1.1 SCORM Sample RTE

Esse ambiente desenvolvido em Java está disponível para download no site da ADL, incluindo todo o seu código fonte. Ele exemplifica a implementação do SN e das APIs para comunicação entre SCO e LMS.

A Figura 19 apresenta o diagrama de casos de uso relacionando os recursos disponíveis no SCORM Sample RTE. Existem somente dois tipos de atores no sistema: o administrador e o usuário.

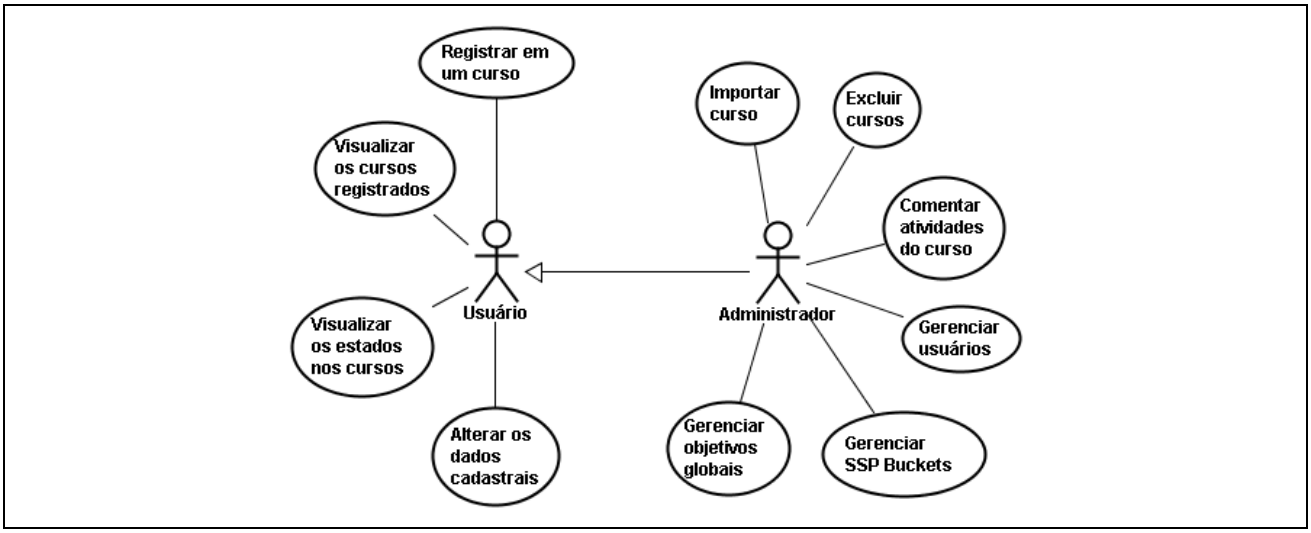


Figura 19. Casos de uso do SCORM Sample RTE

As atividades realizadas exclusivamente pelo administrador são:

- Importar curso: é uma página onde se informa o arquivo zip a ser importado no LMS. Inclusive, existe uma opção para consistir se o arquivo está em conformidade com SCORM antes de importá-lo;
- Excluir cursos: são apresentados todos os cursos disponíveis. O administrador seleciona quais deles são excluídos;

- Comentar atividades do curso: permite que o administrador insira comentários para cada uma das atividades do curso. Esses comentários podem ser acessados pelo SCO através do *Data Model* `cmi.comments_from_lms`;

- Gerenciar usuários: podem ser cadastrados dois tipos de usuários: o administrador e o usuário comum. Cada usuário mantém uma identificação, o nome completo e a senha;

- Gerenciar SSP Buckets[1]: permite adicionar e excluir *data buckets*;

- Gerenciar objetivos globais: é possível definir que o curso compartilhe objetivos com outros cursos. Esses tipos de objetivos podem ser consultados, zerados e excluídos.

Existem atividades comuns a todos os usuários:

- Registrar em um curso: para o usuário poder acessar um curso, exige-se que ele primeiramente se registre nele;

- Visualizar os cursos registrados: a partir dessa opção é possível acessar os cursos registrados, ou somente a árvore de atividades. A Figura 20 ilustra o acesso de um curso no ambiente. A partir do curso é possível suspender a interação, o que implica em continuar em outro momento, ou sair do curso, e nesse caso a interação não é armazenada. Também é possível ativar uma área da janela para acompanhar todas as mensagens trocadas entre SCO e LMS;

- Visualizar os estados nos cursos: para cada curso apresenta se ele está satisfeito, se está completo e a pontuação obtida;

[1] De acordo com IMS Global ... (2004), a especificação Shareable State Persistente (SSP) define modelos e APIs de acesso às áreas de dados (*data buckets*) compartilháveis entre SCOs e cursos distintos. A especificação SCORM determina que todos os dados armazenados por um SCO são acessíveis somente por ele. Porém, a SSP estende essa capacidade, permitindo que o SCO armazene dados que podem ser recuperados mais tarde por ele, ou por outros SCOs, e que esses possam acessá-los e alterá-los. *Data buckets* podem ser especificados no pacote de conteúdo, assim como em tempo de execução.

- Alterar os dados cadastrais: permite que o usuário configure as preferências disponíveis no *Data Model* `cmi.learner_preference`.

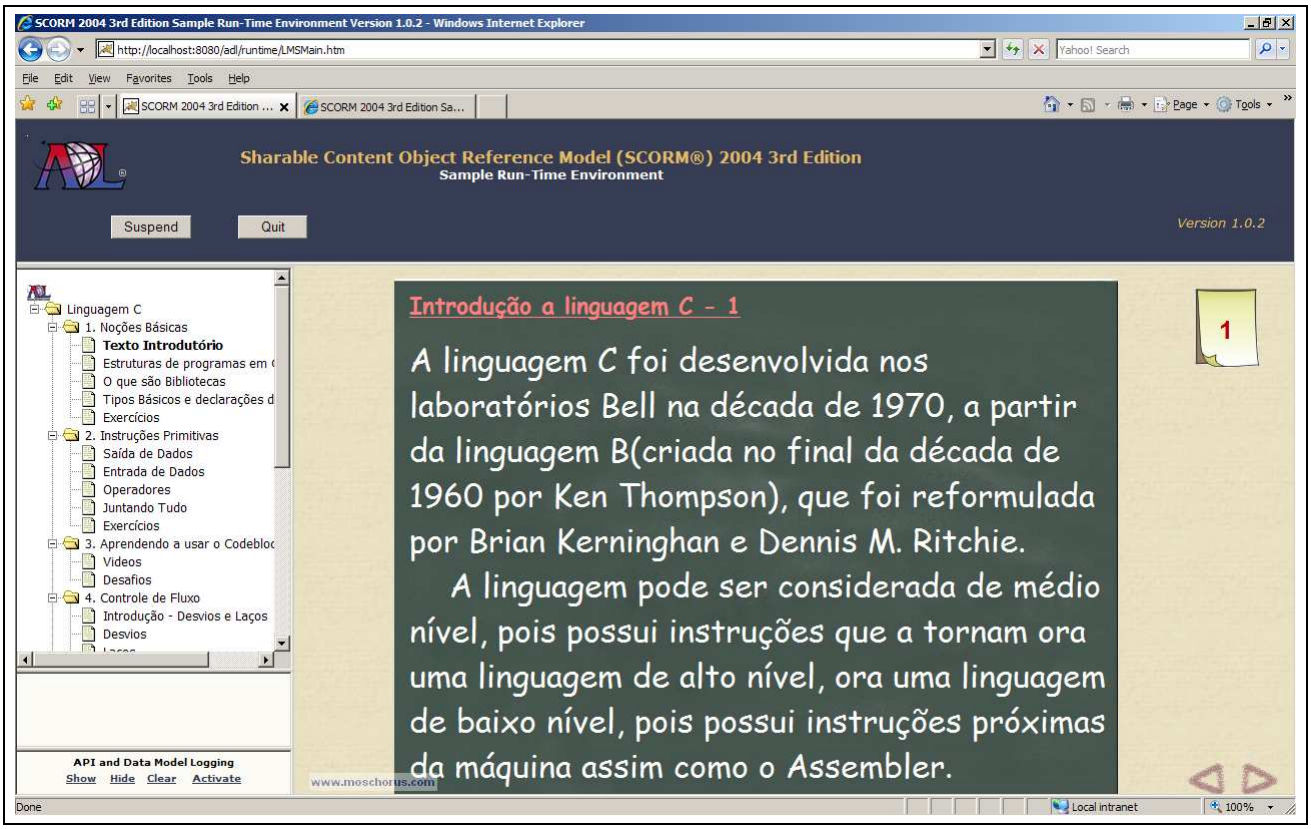


Figura 20. Acessando um curso no SCORM Sample RTE

Os dados do sistema são armazenados com Microsoft Access utilizando ODBC. São mantidos três arquivos: SampleRTE (mantém os dados gerais), SSPManager (dados compartilháveis entre SCOs de cursos distintos) e GlobalObjectives (para armazenar os objetivos compartilháveis entre os cursos).

No arquivo SampleRTE são armazenados os usuários, os cursos cadastrados, os cursos registrados por cada usuário, se o curso está suspenso, a organização do conteúdo de cada curso, e os comentários dos SCOs (aqueles possíveis de serem alimentados pelos administradores).

A instância da API (API_1484_11) de comunicação foi implementada com uma *applet*. A árvore de atividades é renderizada através de um componente de terceiro denominado Morten's JavaScript Tree Menu.

2.5.1.2 Moodle

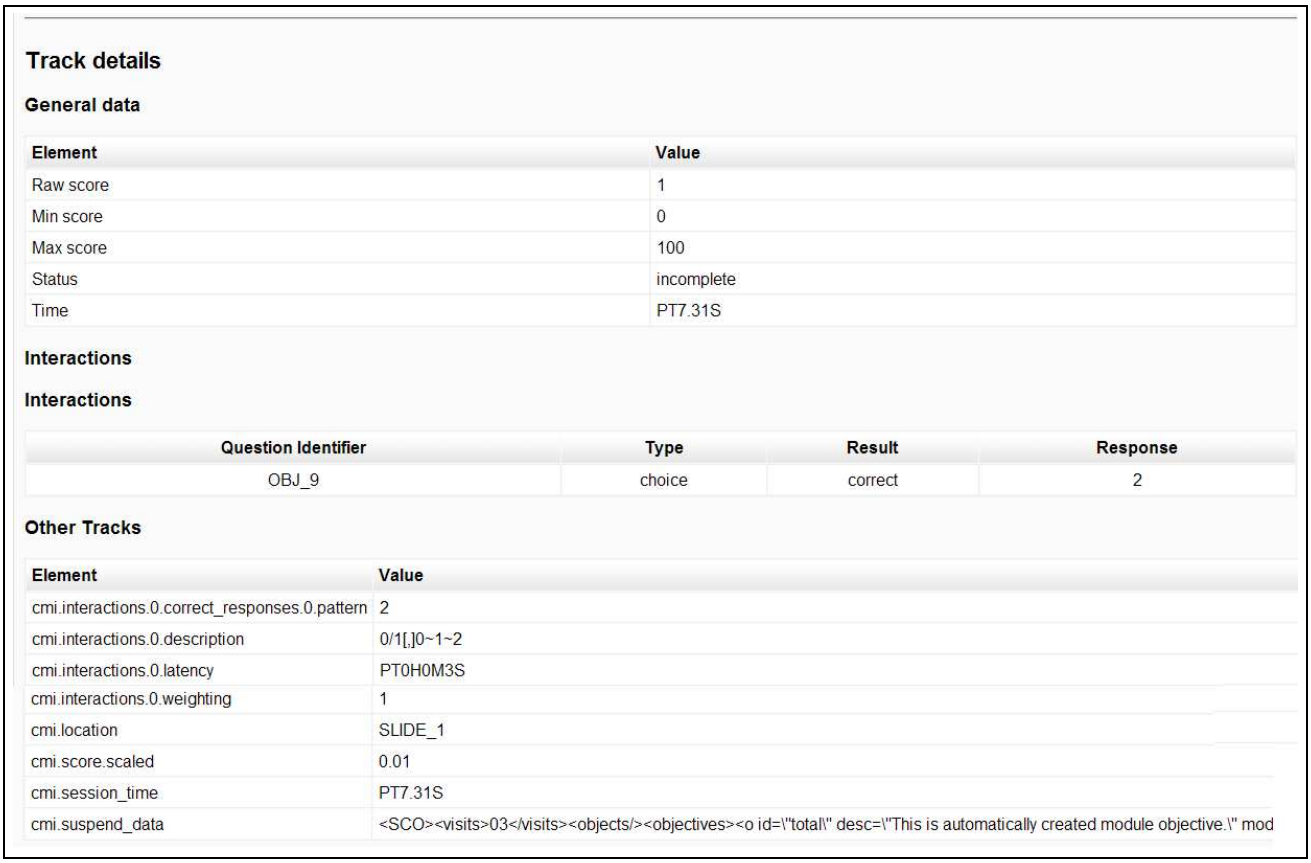
Esse ambiente desenvolvido em PHP, que utiliza o banco de dados MySQL, está disponível em Moodle (2008), e também oferece o seu código fonte. O gerenciamento de pacotes SCORM é desempenhado por um *plugin* que implementa a versão 2004 da especificação.

O Moodle tem seis tipos de usuários: administrador, criador de curso, professor, professor que não altera os cursos, estudantes e visitantes. Os três primeiros conseguem definir as atividades disponíveis em um curso. Entre essas atividades, existe a possibilidade de adicionar um pacote SCORM.

Quando o usuário adiciona uma atividade com pacote SCORM, o sistema solicita um nome para o curso e o arquivo zip. Além disso, opcionalmente, existem disponíveis várias configurações, como por exemplo, o tamanho da área que exibirá o conteúdo, se abre em uma nova janela, e a posição da árvore de atividades.

Na exibição do curso, cada atividade folha é representada por um ícone de acordo com o estado dela: visitado/não visitado, completo/incompleto, aprovado/reprovado, asset e suspensa.

Com exceção dos alunos e visitantes, os demais usuários conseguem visualizar relatórios de acessos dos conteúdos. Primeiramente são exibidos todos os cursos e quantos usuários experimentaram cada um deles. Ao selecionar um dos cursos, são apresentados os usuários que acessaram, quando foi o último acesso, quantas vezes acessaram e a sua pontuação. Ao escolher um desses usuários são então exibidas todas as atividades com o seu estado (se foi tentado, completo ou incompleto), o tempo consumido e a pontuação. Ainda é possível obter os detalhes de cada atividade. No detalhamento são destacados os elementos `cmi.interactions` e todos os elementos *Data Model* atribuídos para o SCO, com seus respectivos valores. A Figura 21 ilustra esse relatório.



Track details

General data

Element	Value
Raw score	1
Min score	0
Max score	100
Status	incomplete
Time	PT7.31S

Interactions

Interactions

Question Identifier	Type	Result	Response
OBJ_9	choice	correct	2

Other Tracks

Element	Value
cmi.interactions.0.correct_responses.0.pattern	2
cmi.interactions.0.description	0/1[,]0~1~2
cmi.interactions.0.latency	PT0H0M3S
cmi.interactions.0.weighting	1
cmi.location	SLIDE_1
cmi.score.scaled	0.01
cmi.session_time	PT7.31S
cmi.suspend_data	<SCO><visits>03</visits><objects/><objectives><o id=\"total\" desc=\"This is automatically created module objective.\" mod

Figura 21. Exemplo de relatório detalhando uma atividade no Moodle

2.5.2 Trabalhos relacionados ao modelo SCORM

Nesta seção são relacionados os trabalhos publicados em revistas e anais internacionais quanto ao uso do SCORM. Os artigos foram reunidos em dois grupos: o primeiro quanto a adaptação dos pacotes para que eles sejam personalizados segundo as necessidades e características do usuário; e o outro grupo refere-se a trabalhos que visam melhorar as tarefas de produção do material instrucional e também quanto à arquitetura das aplicações, objetivando estruturar um ambiente com facilidades de gerenciamento e integração com outras tecnologias.

2.5.2.1 Adaptação de conteúdo SCORM

MOT (My Online Teacher) é uma ferramenta de autoria de hipermídia adaptativa (CRISTEA e STEWART, 2006) que segue o framework LAOS (*Layered Adaptative Hypermedia System Authoring-Model and Operators*) formado por cinco camadas, cada uma representando um modelo na elaboração do material: modelo de domínio, de objetivos e restrições, do usuário, de adaptação e de apresentação. O MOT consegue gerar automaticamente os modelos de uma camada se baseando nas camadas antecedentes, inferindo das relações dos elementos desta camada. O

LAOS é bem rico em recursos para definir adaptação do material: em uma camada são elaboradas as estratégias e técnicas pedagógicas; noutra se definem estratégias baseadas na característica do aluno; e outra se baseia nas inteligências múltiplas. O objetivo dos autores é criar ferramentas que exportem o material produzido no MOT para Sistemas de Hipermídia Adaptativa. Uma dessas ferramentas exporta o material para o formato SCORM (POWER *et al*, 2005; CRISTEA e STEWART, 2006). Os autores definem que o conceito de adaptação no SCORM apenas é ilusório, pois o material é pré-adaptado (com as regras de seqüenciamento) e o usuário tem a impressão de adaptação. O material é produzido baseado no modelo de objetivos e restrições do LAOS, que é exatamente o que o seqüenciamento do SCORM segue: utiliza regras que comparam objetivos, e de acordo com o desempenho habilita novas atividades.

Na mesma linha de utilização de ferramentas de autoria, está o trabalho de Limongelli, Sampietro e Temperini (2007). Nesse trabalho foi implementado um *plugin* para o Moodle gerar automaticamente os cursos. O aluno resolve um conjunto de questões iniciais, em conjunto com o conhecimento a ser adquirido e o plano de ações, para o *plugin* gerar um curso no formato SCORM personalizado para o usuário. Para gerar esse curso, o *plugin* se baseia num conceito chamado Componente de Aprendizagem. Esse componente contém definições do conteúdo a ser ensinado (o conjunto de páginas HTML), os pré-requisitos (conhecimento que o aluno precisa ter para acessar o conteúdo) e o conhecimento adquirido (o que o aluno aprendeu após interagir com o componente). Uma ferramenta transforma esse componente em um SCO, montando um arquivo de metadados e o manifesto. O plano de ações é definido por uma linguagem de script através de uma ferramenta chamada BlackBox. Nesse plano são listados o conhecimento inicial e o objetivo, e por fim que ações são necessárias para convergir entre esses extremos. Cada ação representa um módulo didático, que exige pré-requisitos, e a lista de conhecimentos adquiridos na conclusão do módulo.

Quanto a proposta de estender a especificação SCORM, foram encontrados três trabalhos: Silva *et al* (2006) que propõem uma modificação no arquivo de manifesto, substituindo os SCOs por arquivos de metadados, Shin, Lee e Baik (2007) que adicionam um novo elemento no arquivo de metadados, e Rey-Lopez *et al* (2008) que apontam mudanças tanto no arquivo de manifesto quanto no SCO.

Existem várias classificações de estilos de aprendizagem, e uma delas, a classificação de Kolb, é utilizada por Silva *et al* (2006) para gerar automaticamente a estrutura e seqüência de um curso. A partir de um questionário o estudante é avaliado em uma das quatro categorias dessa

classificação (convergente, assimilador, divergente e acomodado). Além disso, os autores propõem o uso da Teoria do Entrelaçamento, para criar um mapa conceitual baseado nos valores de atributos dos conceitos. Nessa teoria estabelecem-se atributos comuns a todos, e de acordo com os valores de domínio desses atributos criam-se as relações entre os conceitos agrupando-os por aqueles valores dos atributos. Um mapa exemplificado no artigo está ilustrado na Figura 22. Para esse mapa foi utilizado o conceito dos planetas do sistema solar, e os atributos foram distância do Sol, tamanho do planeta e se ele possui luas. Observa-se que na metade superior do mapa encontram-se os atributos e na metade inferior a relação dos planetas com os atributos. Esse mapa permite uma navegação bidirecional: a partir de um planeta descobrir os seus conceitos, ou a partir dos conceitos descobrirem o conjunto de planetas que atendem os valores.

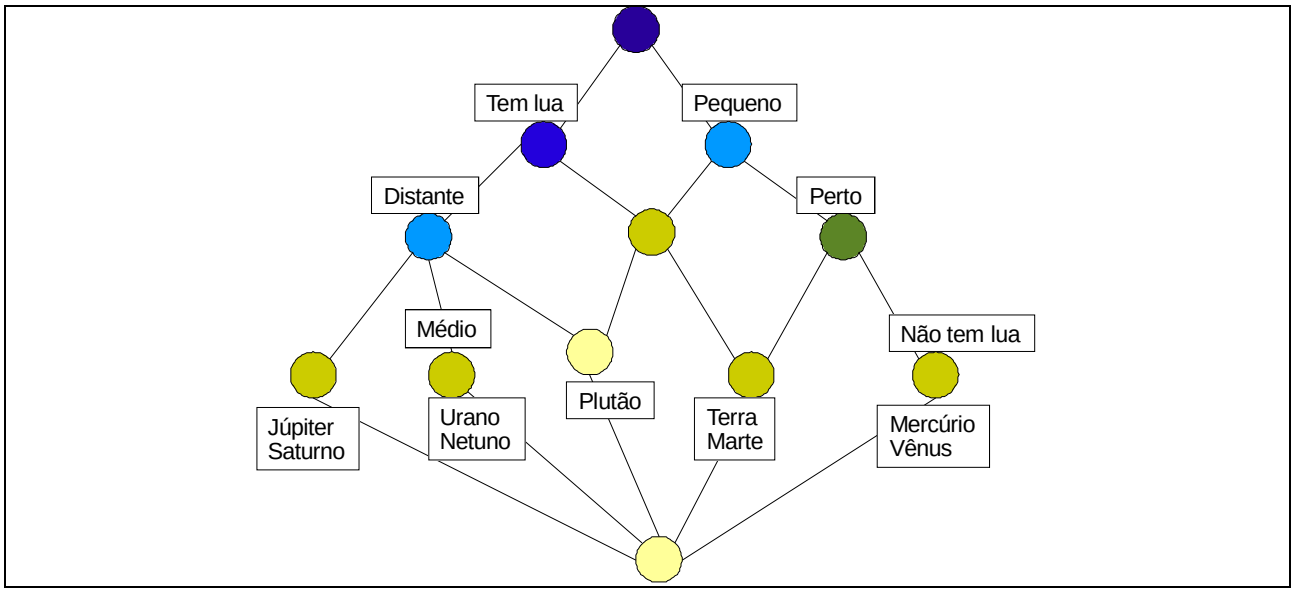


Figura 22. Exemplo de mapa conceitual utilizando a Teoria do Entrelaçamento
Fonte: Adaptado de Silva *et al* (2006).

Essa possibilidade de navegação é que define como o conteúdo será apresentado de acordo com os estilos de Kolb. Segundo essa classificação, os alunos convergentes e assimiladores conseguem aprender baseado nas propriedades dos planetas, ou seja, eles têm um poder de abstração melhor que o outro grupo. Nesse caso, o conteúdo pode ser estruturado de cima para baixo no mapa.

Para implantar essa idéia no modelo SCORM, e criar a estrutura de conteúdo, os autores propõem utilizar arquivos de metadados e relacioná-los em vez de utilizar SCOs. Esses arquivos utilizam o elemento `keyword` para definir os atributos e os conceitos, e um novo elemento

chamado `navigation` que forma a relação entre esses arquivos. A estrutura da organização aponta somente para um desses metadados junto da definição de um dos quatro estilos de aprendizagem. Com base nisso, o LMS pode inferir a direção de navegação de acordo com o estilo do aluno.

Shin, Lee e Baik (2007) apresentam um sistema que gera automaticamente um curso baseado no histórico das atividades. Inicialmente, os professores propõem uma seqüência de conteúdos a serem estudados, para então resolver um problema. Os alunos, após o estudo e resolução do problema, analisam a relevância dos conteúdos para a solução, aumentando ou diminuindo a utilidade de cada SCO. Durante o estudo, o aluno pode acessar livremente outros SCOs, e incluí-los na avaliação. Os conteúdos também são avaliados negativamente quando o aluno não consegue resolver o problema. Todas essas informações são armazenadas em um elemento de metadados do SCO. Assim, quando um novo aluno precisar resolver um problema que outros já tiveram, o sistema montará o conjunto e a seqüência de atividades de acordo com as avaliações feitas pelos outros alunos.

Os níveis de extensão que ocorrem no trabalho de Rey-Lopez *et al* (2008) são na estrutura do conteúdo, ou a nível de atividade, e no próprio SCO.

O SCO é auto-adaptável de acordo com as características do aluno. Essa adaptação ocorre na personalização de como o conteúdo é apresentado: cores, fontes, disposição do texto, adição de recursos multimídia ou figuras. Para isso, foram adicionados alguns elementos no Data Model para que o SCO obtenha as características do usuário. O SCO é montado sobre um *template*. Existem um ou mais arquivos de configuração que armazenam as formas e opções que o SCO pode aparentar. E por final, um arquivo com as regras de adaptação contendo os dados a serem solicitados pelo LMS, e de acordo com os resultados seleciona um dos arquivos de configuração.

A adaptação a nível de atividades acontece para elencar que atividades realmente serão proporcionadas ao aluno. O designer instrucional pode definir um curso com várias atividades, e regras de adaptação para que determinadas atividades sejam dispensáveis para o aluno que já tenha conhecimento em determinado assunto, o que tornaria repetitivo, ou selecionar o conjunto de atividades, ou a sua seqüência, de acordo com as preferências do aluno. O trabalho definiu um novo elemento no arquivo de manifesto para dar suporte a essas regras. Esse elemento pode ser inserido em qualquer nível de atividade.

Os cursos com essas características são montados com uma ferramenta que os autores desenvolveram, chamado SCOCreator, ou através de um *plugin* na ferramenta Reload, também implementado por eles.

Kazanidis e Satratzemi (2007) apresentam os requisitos do LMS que eles pretendem desenvolver com capacidades de SHA. Eles citam os objetivos do trabalho deles como suportar e gerenciar cursos no formato SCORM, adaptar os cursos de acordo com o conhecimento e objetivos do aluno, e permitir o acompanhamento dos tutores sobre o desempenho dos alunos. O modelo do aluno manterá os cursos já feitos e os resultados deles. Os tutores poderão interferir nos resultados apurados pelo sistema, fazendo com que o aluno faça de novo uma atividade caso o tutor tenha percebido que ele não aprendeu. Baseado no modelo do aluno, o sistema pode apresentar cursos que sejam de interesse do aluno, também permitindo a intervenção do tutor, para que ele adicione algum curso que o sistema não tenha percebido. A árvore de atividades sofre mudanças na apresentação dos seus itens quanto as necessidades do usuário. O aluno terá objetivos e divisões desses objetivos a serem cumpridos, e poderá visualizar o seu progresso a qualquer nível. Os alunos poderão questionar os tutores, e junto das respostas, serão armazenados num banco de perguntas. Também estará disponível um recurso para que o aluno faça anotações sobre os conteúdos apresentados.

A geração adaptativa do curso, no trabalho de Chang *et al* (2007), é executada com base na análise do desempenho dos alunos nas avaliações. O ambiente utiliza uma tabela tridimensional que contém conteúdos, alunos e questões. As células são preenchidas com os resultados das questões: acertos e erros. O sistema, com essa tabela, infere se os cursos atendem as avaliações. Ele avalia a quantidade de conteúdos abrangidos por uma questão, a quantidade questões que avaliam um conteúdo, as questões com menor índice de acerto, e assim por diante. Com essas informações, o sistema pode adicionar mais conteúdos, ou alterar os conteúdos que estão sendo aplicados. Para executar estas tarefas de adição e alteração, o sistema utiliza alguns campos de metadados: interactivityType (active, expositive e mixed), interactivityLevel (very low, low, medium, high e very high), learningResourceType (exercise, simulation, questionnaire, diagram, etc) e typicalLearningTime.

O uso de sociedades de agentes em ambientes de aprendizagem é fortemente justificado pela capacidade nata de adicionar novos agentes tanto para atender necessidades de escalabilidade e principalmente por novos recursos (SHEMSHADI, SOROOR e TAROKH, 2008). A seguir serão relatados três artigos analisados quanto ao uso de agentes para garantir um ambiente adaptável.

A sociedade de agentes – proposta por Silva, Bavaresco e Silveira (2007) – apresenta uma perspectiva diferente em relação à adaptabilidade. Os trabalhos relatados anteriormente propõem mudanças na definição dos pacotes de conteúdo ou mesmo no ambiente. Já para esses autores, a capacidade de adaptação pode ser alcançada quando os próprios objetos de aprendizagem são inteligentes. Para isso, eles apresentam uma sociedade composta por agentes que desempenham a função de LMS, de repositório, de aluno e do próprio objeto de aprendizagem. Quanto ao SCORM, um agente tem a função de marcar o início e fim dos SCOs. Além disso, quando um novo pacote de conteúdo é adicionado ao repositório, o agente de repositório interpreta o arquivo de manifesto para que seja indexado segundo as necessidades do sistema.

A arquitetura de LMS baseada em agentes proposta por Shemshadi, Soroor e Tarokh (2008) destaca-se pela presença de um agente agregador de conteúdo (CAAgent). Além disso, está envolvido mais um agente, responsável por coordenar a tarefa do LMS (LMSAgent), e outro para representar o papel do aluno. Esse agente solicita uma requisição de conteúdo para o LMSAgent, que por sua vez processa a árvore de atividades, e solicita o conteúdo ao CAAgent, que cria o arquivo de manifesto, monta o seqüenciamento e empacota o conteúdo, devolvendo-o ao LMSAgent.

Bitonto e Roselli (2008) apresentam um agente que pode ser incorporado a um LMS com a finalidade de alterar o seqüenciamento baseado nas características do usuário, onde esse perfil está armazenado segundo o IMS Learning Information Package (IMS LIP). O LIP é uma especificação que guarda as características, objetivos e habilidades do usuário, utilizando para isso 11 categorias principais, que por sua vez tem suas subdivisões. O agente é composto por três componentes. O primeiro é usado quando ao LMS é solicitado o carregamento de um pacote de conteúdo. O componente analisa o arquivo de manifesto (imsmanifest.xml). Dentro do agente, o segundo componente é acionado para acessar a base de conhecimento com a finalidade de adaptar o seqüenciamento do conteúdo. Essa adaptação depende do estilo do aluno, e para obter essa informação, são solicitados a um novo componente os dados do usuário, obtidos a partir da análise de um arquivo LIP.

2.5.2.2 Outros trabalhos correlatos

Nesta seção é apresentado um estudo com objetivo de relatar pesquisas realizadas por outros autores quanto ao uso do padrão SCORM, que não seja no âmbito da adaptabilidade.

O projeto de seqüenciamento do SCORM é pouco natural, pois as atividades são representadas como uma estrutura de árvore, que facilita o encadeamento hierárquico de atividades, mas dificulta o controle do encadeamento de atividades entre nós distintos. Alguns trabalhos (SU *et al*, 2005; KIM, YOO e KIM, 2005; LIN *et al*, 2005; KIM, KIM e PAIK, 2008) propõe a utilização de grafos como representação do seqüenciamento. E outros (CHANG *et al*, 2006) propõe melhorias no processo de projeto.

Assim como no desenvolvimento de software existe a atividade de testes, ou seja, verificar e comparar o proposto e o realizado, essa atividade também é necessária quando se trata de desenvolvimento de cursos. Esses testes podem ser para constatar apresentação do conteúdo dentro de um SCO, entradas e saídas dos formulários, e no seqüenciamento verificando se todas as atividades podem ser alcançadas, ou se não existe uma seqüência cíclica de atividades. Essa tarefa pode não ser trivial quando o curso possuir dezenas de atividades e as possibilidades de caminhos do usuário pode ser bem variado. Um algoritmo para detectar inconsistências no seqüenciamento foi desenvolvido por Chang *et al* (2006) baseado na análise do elemento **Controle de Requisição**. Para validar o algoritmo, foi adicionado um recurso de detecção em uma ferramenta de autoria e em um LMS, ambos desenvolvidos pelo grupo de pesquisa deles.

Uma ferramenta de autoria é proposta por Su *et al* (2005), onde o seqüenciamento das atividades é produzido através de Objetos de Seqüenciamento (SO), seguindo cinco modelos inspirados em estruturas de programação. Essas estruturas são compostas de nós e arestas direcionadas. Os nós podem ser compostos por outros nós ou atividades dos pacotes de conteúdo SCORM. Os cinco modelos de estrutura são: linear (um nó após o outro), condicionais (linear, que permite continuar quando o usuário atender uma condição; seleção, que define um caminho para cada tipo de condição), laço (um nó com duas condições: uma para manter no mesmo nó e outra para se dirigir a outro nó) e saída (encerrar um nó). As estruturas linear e condicional linear são ilustradas na Figura 23 com os objetos de seqüenciamento e seus correspondentes no modelo de seqüenciamento do SCORM.

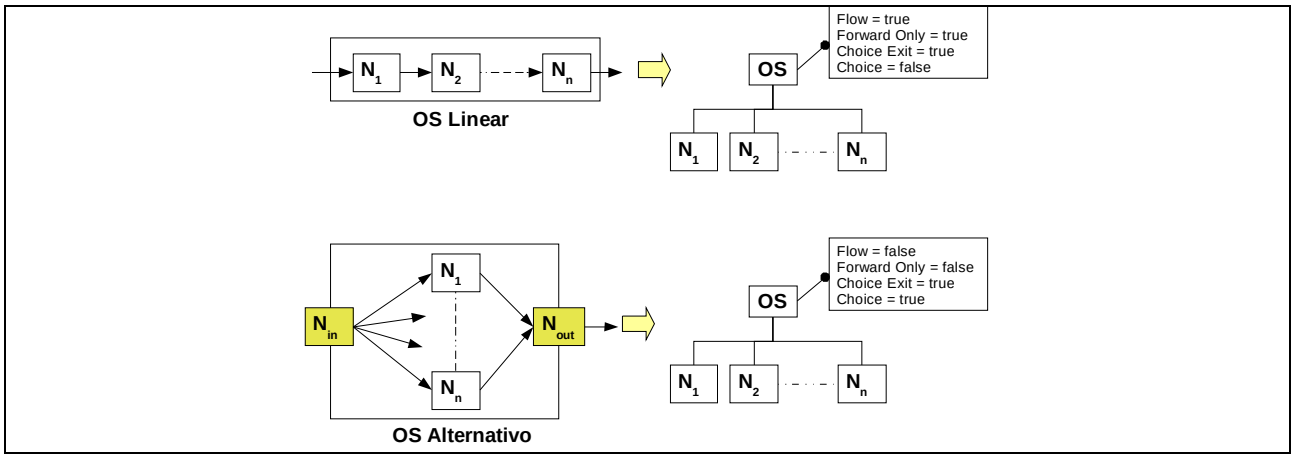


Figura 23. Objetos de seqüenciamento com o modelo de seqüenciamento correspondente
Fonte: Adaptado de Su *et al* (2005).

Assim como em Su *et al* (2005), outro trabalho que procura alternativas na representação do seqüenciamento é apresentado Kim, Yoo e Kim (2005) e Kim, Kim e Paik (2008). Esse trabalho estabelece a formalização e representação gráfica do processo de ensino, que por conseqüência resulta na geração de um curso. Essa representação é baseada em grafos direcionados, mas que possui equivalência total com o modelo SCORM. O modelo proposto pelos autores se concentra em criar uma estrutura baseada no processo de ensino, em que conteúdo e seqüência são indissociáveis, ao contrário do que acontece na especificação SCORM, onde o seqüenciamento é aplicado sobre a estrutura do conteúdo. A Figura 24 apresenta o exemplo de um curso montado seguindo o modelo proposto. Cada vértice representa uma tarefa dentro do processo e está associado a um conjunto de SCOs. Os pequenos círculos representam um conector OR e os quadrados são os repositórios em que estão armazenados os SCOs identificados em cada vértice.

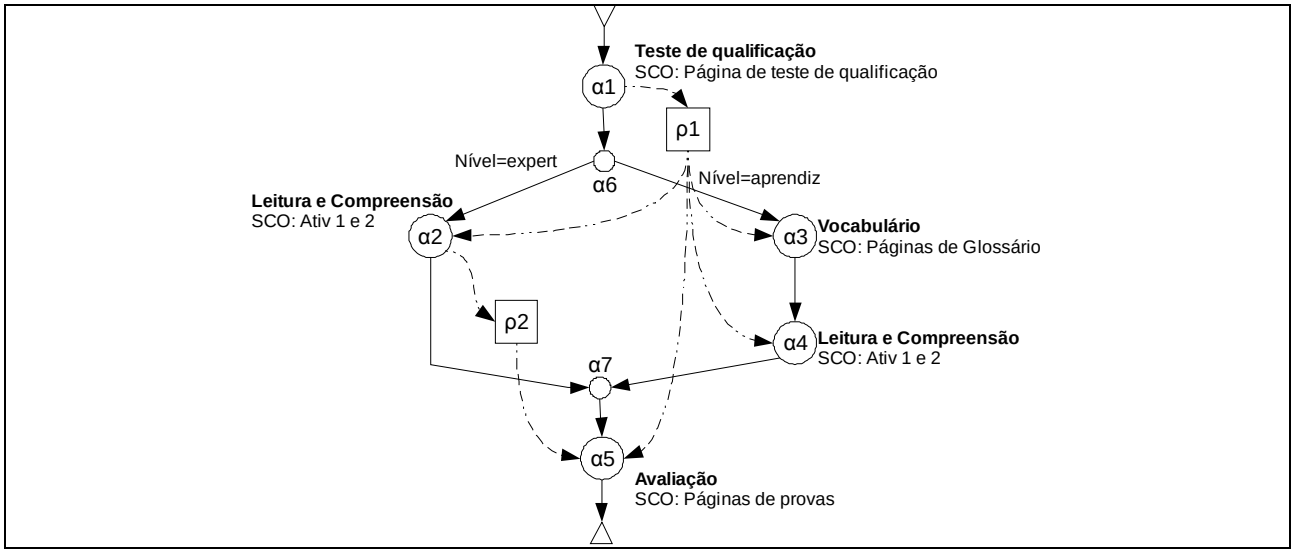


Figura 24. Representação gráfica do processo de ensinar inglês
Fonte: Adaptado de Kim, Yoo e Kim (2005).

Lin *et al* (2004) e Lin *et al* (2005) apresentam um modelo e uma ferramenta para o projeto de seqüenciamento utilizando Redes de Petri, que é representado por um grafo direcionado. Além disso, propõe a possibilidade de um ambiente colaborativo usando SCORM. A proposta deles inclui um seqüenciamento em que alunos desempenham papéis diferentes, e por isso seguem caminhos diferentes. Existem pontos em que a seqüência é suspendida para que aconteça uma sessão de chat entre os envolvidos. Essa seqüência pode juntar os alunos em certo momento para executarem uma atividade em conjunto. A Figura 25 representa um exemplo dessa seqüência: existem três papéis (representados por *p1*, *p2* e *p3*). De acordo com o resultado de um exame, o estudante pode ser classificado como *p1* ou *p2*. Em certo momento, eles se agrupam, e a seqüência continua após o dois alunos com o papel *p1* e outro com o papel *p2* alcançarem os objetivos da atividade.

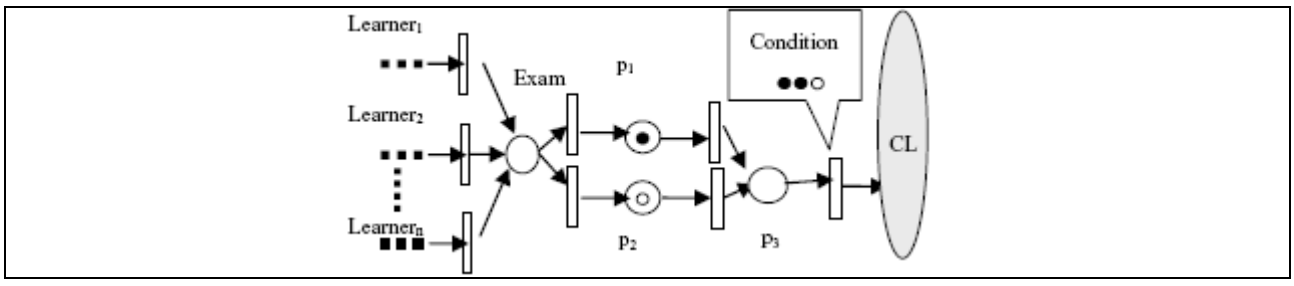


Figura 25. Representação gráfica de um fluxo para aprendizagem colaborativa
Fonte: Adaptado de Lin *et al* (2005).

Outra linha de pesquisa em relação ao SCORM é a execução *off-line* ou em dispositivos móveis. Isso se faz necessário, pois os navegadores dos dispositivos móveis são limitados quanto aos recursos que podem ser renderizados ou não suportam JavaScript. Alguns desses trabalhos são relatados a seguir.

Nakabayashi *et al* (2007) apresentam uma extensão do SCORM, onde no arquivo de manifesto pode ser definido se o conteúdo é apresentado em computador, dispositivo móvel, ou em ambos. Nos dois últimos casos, o LMS disponibiliza o conteúdo para download. Os pesquisadores desenvolveram um navegador que renderiza HTML simples, e uma linguagem de script que é encapsulada em tags de comentário HTML. Esse script contém comandos da API do RTE para se comunicar com o LMS. Esse navegador interpreta o script e coordena a comunicação.

Nordmann e Neumann (2008) desenvolveram duas ferramentas com intuito de interagir e visualizar off-line os pacotes SCORM. Uma delas é uma ferramenta de autoria, que gera um pacote SCORM, e artefatos adicionais para esses pacotes, como glossário, legendas e thumbnail (pequenas figuras representando o SCO). Essas informações adicionais podem ser vistas somente pela ferramenta de visualização desses autores, que utiliza o motor do Microsoft Internet Explorer. Os pacotes SCORM podem ser acessados off-line por essa ferramenta.

A utilização de serviços web (*web services*) também é foco de várias pesquisas. O intuito desses trabalhos é prover a interoperabilidade quanto ao uso da linguagem de programação ou plataforma de servidor web. Assim ambientes de aprendizagem podem adicionar novos recursos sem haver compatibilidade alguma entre as tecnologias empregadas com os novos serviços.

Uma arquitetura para montar um LMS totalmente baseado em Web Services é apresentado em Chun *et al* (2004). Naturalmente o LMS contém um serviço de acompanhamento do usuário, que se utiliza de informações para dirigir o conteúdo ao usuário, e armazena as interações dele com o ambiente. A proposta desses autores é que esse serviço permaneça local, e que a ele seja adicionada a função de coordenar os demais WS, como o serviço de *profile* do usuário (que pode estar armazenado em mais de um servidor) e um serviço de gerenciamento de conteúdo (que armazena e oferece cursos para o usuário).

Chien e Kao (2005) propuseram a integração de um LMS com vários LCMS (Learning Content Management System) através de *broker* que utiliza Web Services. Ao selecionar um

conteúdo, o *broker* define qual servidor deve entregar o conteúdo. Como o conteúdo apresentado encontra-se em um servidor (LCMS que proveu aquele conteúdo) e o LMS pode encontrar-se em outro, é necessário sobrescrever a implementação do adaptador da API do RTE que o pacote de conteúdo fornece, pois as chamadas à API seriam todas direcionadas para o LCMS, quando quem trata é o LMS.

O trabalho de Vossen e Westerkamp (2006) segue na mesma direção de Chun *et al* (2004). Baseado no ambiente denominado Learnserve projetado pelos próprios autores, definiram uma versão SCORM construída totalmente de serviços web. A diferença entre os dois trabalhos, é que de Chun *et al* (2004) o seqüenciamento é realizado pelo orquestrador de serviços, e no trabalho de Vossen e Westerkamp (2006), o seqüenciamento é um novo serviço que pode ser incorporado ao servidor.

Abdullah e Davis (2005) apresentam uma arquitetura de hipermídia adaptativa que permite expandir o conteúdo SCORM. Quando o usuário seleciona um item da árvore de atividades, o título desse item é passado a um servidor de *links* personalizados, que busca novas URLs para esse assunto, e devolve essa lista. Eles apresentaram uma adaptação do LMS ADL SCORM RTE (implementação de referência do SCORM) onde à página do usuário foi adicionado um novo frame com os links adicionais.

3 DESENVOLVIMENTO

A especificação SCORM divide em dois papéis a responsabilidade pelo material instrucional: um deles é a elaboração do próprio material, feito por ferramentas de autoria; e outro é o ambiente que executa e gerencia esses pacotes. O material é elaborado independente do ambiente em que for executado, inclusive sem criar vínculos com o mesmo. Essas ferramentas possuem funções que permitem a elaboração de materiais com recursos de multimídia oferecendo muita interatividade. Uma vez que os Ambientes Inteligentes de Aprendizagem estejam em conformidade com a especificação SCORM, eles poderiam se beneficiar dos materiais elaborados por essas ferramentas. Além disso, permite o uso de repositórios de conteúdo, podendo aproveitar material elaborado por outras instituições.

Essa dissertação propõe uma infra estrutura, distribuída como componente, para possibilitar o desenvolvimento de aplicações web em Java, visando fornecê-los recursos para gerenciamento e execução de pacotes de conteúdo SCORM.

Na Figura 26 é apresentado como o componente se relaciona com as outras partes da aplicação. O componente fornece recursos para elaboração das páginas JSP, que são a entrada e saída do sistema. Essas páginas listam e abrem os cursos registrados no sistema. Esses cursos são pastas acessíveis para o componente. É possível interceptar as ações do componente, permitindo adicionar recursos inteligentes, que consigam gerar e a adaptar a estrutura dos cursos.

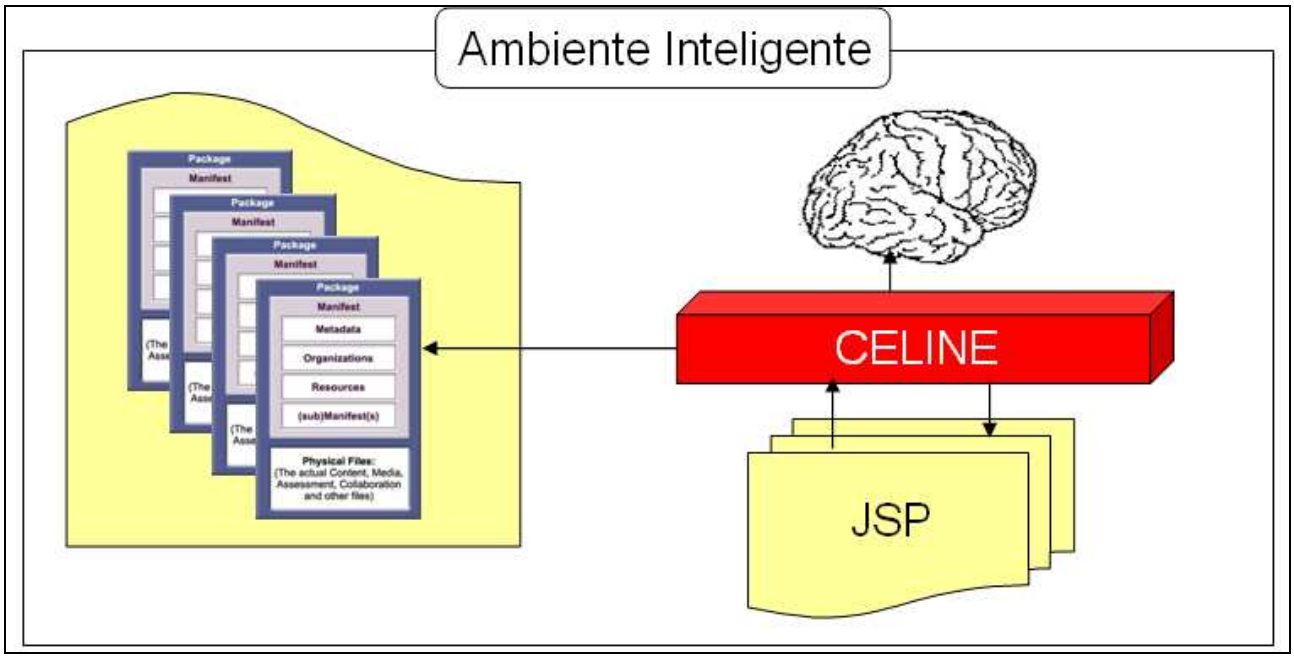


Figura 26. Visão geral do componente integrado à aplicação

O componente, pelo seu conceito, é um artefato de software acoplável aos sistemas, que possui um conjunto de objetivos bem específicos. Ele possui entradas e saídas bem determinadas, onde as entradas são as configurações do comportamento do componente, e as saídas são os resultados esperados do componente conforme o comportamento configurado.

O desenvolvimento do componente foi dividido em dois módulos: SCORM e LMS. O primeiro módulo objetiva a implementação da especificação SCORM e o segundo são os recursos para o desenvolvimento do ambiente para utilização dos pacotes SCORM. Esse capítulo foi dividido em três seções: na primeira são relacionados os requisitos do componente; na segunda está a especificação e implementação do módulo SCORM e na última seção estão a especificação e implementação do módulo LMS.

3.1 REQUISITOS

A seguir estão relacionados os requisitos funcionais baseando-se nos ambientes relacionados na Seção 2.5.1 e na possibilidade de adaptação do conteúdo:

1. Importação de pacotes SCORM por intermédio de um arquivo zip;
2. Atualização e exclusão do pacote SCORM;
3. Cadastramento de usuários, diferenciando entre os administradores e usuários;

4. Registrar o pacote para interagir com ele, assim como desvincular-se a ele;
5. Visualizar a árvore de atividades do pacote;
6. Interação com o conteúdo;
7. Suspender e retornar a interação com o conteúdo;
8. Relatórios de utilização dos pacotes SCORM;
9. Todos os acessos exigem usuários autenticados;
10. Criar um curso dinamicamente baseando-se em partes de outros cursos;
11. Alterar a árvore de atividades enquanto o usuário interage com o curso.

Os requisitos não-funcionais a seguir foram elencados visando flexibilizar e facilitar a utilização do componente:

1. Execução dos pacotes em conformidade com a versão 2004 3ª edição do SCORM;
2. Persistência dos dados em arquivos XML, banco de dados relacional ou por customização do desenvolvedor;
3. Desenvolvido em Java;
4. Disponibilizar *tags* customizadas em JSP para acesso aos recursos do componente;
5. Ser disponibilizado através de arquivos JAR.

3.2 SCORM

Esta seção apresenta o desenvolvimento do módulo SCORM, primeiro apresentando toda a especificação para em seguida descrever alguns detalhes sobre a implementação.

3.2.1 Especificação

A especificação do SCORM é dividida em três partes (CAM, RTE e SN) que foram norteadores para a divisão das tarefas na especificação e desenvolvimento do CELINE.

Quanto ao CAM, é necessária a leitura e interpretação do arquivo de manifesto. A Figura 27 apresenta parte das classes envolvidas no processo de interpretação do arquivo de manifesto. `ContentPackageReader` é responsável por coordenar a leitura do arquivo e instanciar um objeto

`ContentPackage` resultante dessa interpretação. Cada nó do arquivo de manifesto tem uma classe equivalente nesse diagrama, e é instanciado um objeto agregado no `ContentPackage`. A estrutura desse arquivo de manifesto foi apresentada na Figura 4. O mapeamento foi realizado seguindo o modelo de agregação de conteúdo (ADL, 2006a).

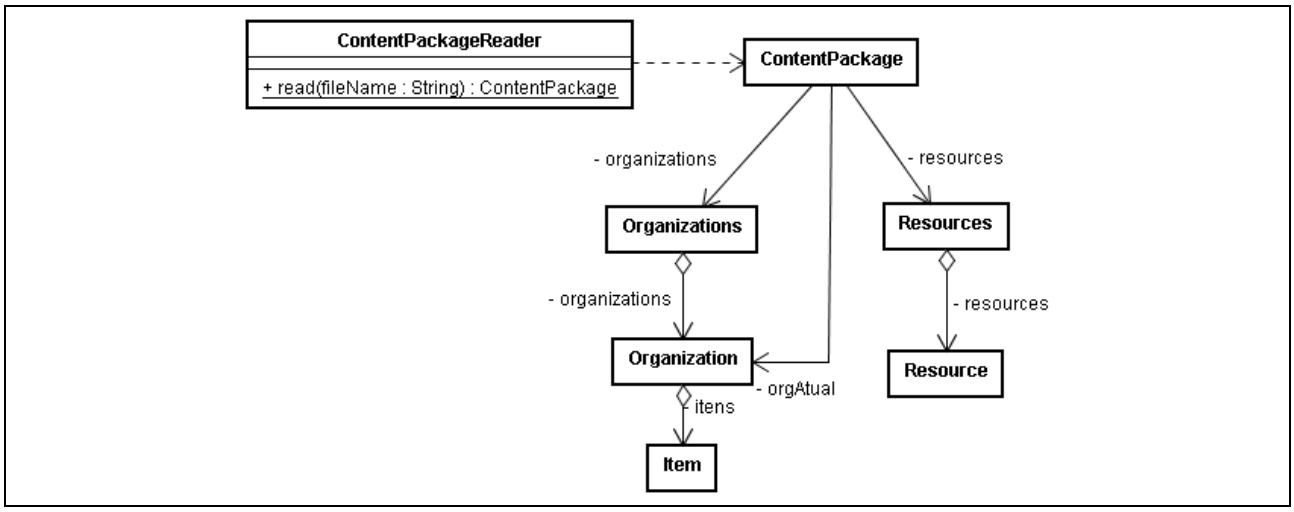


Figura 27. Diagrama das classes do modelo de agregação de conteúdo

O processo de seqüenciamento e navegação (SN) segue conforme ADL (2006d) e já apresentado na Seção 2.2. A Figura 28 apresenta a arquitetura de classes que gerenciam a criação do processo. Cada fase do processo de seqüenciamento (p.e. navegação, encerramento, retropropagação, etc) foi mapeado para um pacote contendo as classes para resolver a fase. O objeto `ProcessProvider` contém a relação com todas as fases. Quando uma fase passa para a outra, ela solicita ao `ProcessProvider` um objeto dessa outra fase. Essa estrutura foi concebida visando a facilidade de manutenção nas próximas versões do SCORM. Quando alterar a especificação de alguma fase do seqüenciamento, somente as classes relacionadas a essa fase deverão ser repensadas, e por ventura estendidas das que existem. O responsável por instanciar essas relações é a classe `BuildProcessProvider20043rdEdition`, que como o próprio nome sugere, cria o relacionamento de acordo com a versão 2004 3ª edição do SCORM. Quando a ADL lançar uma nova versão, cria-se então uma nova classe que será responsável por instanciar e montar um `ProcessProvider` para atender a nova versão.

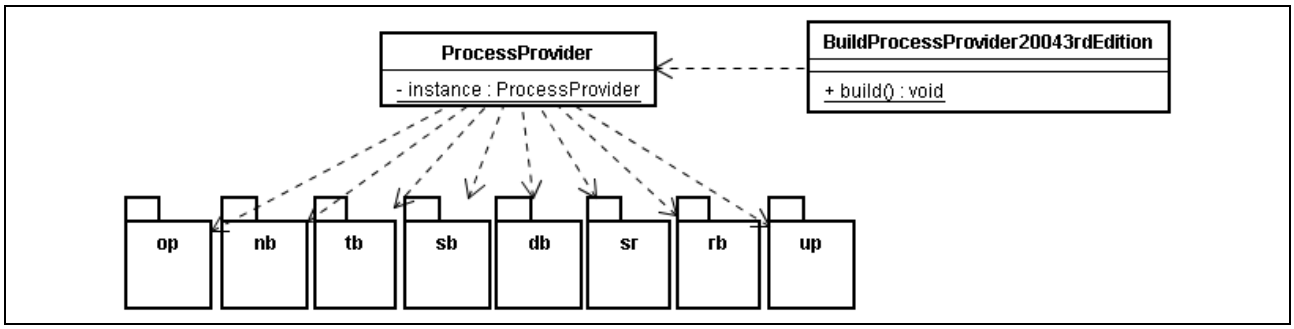


Figura 28. Diagrama das classes com a arquitetura de criação do processo de seqüenciamento

Cada pacote apresentado na figura anterior representa uma fase do processo, conforme descrito na Tabela 2.

Tabela 2. Descrição dos pacotes de seqüenciamento

Pacote	**Descrição**
`op`	Centralizador do processo de execução.
`nb`	Navegação. Processa as requisições de navegação vindas do usuário. Por exemplo, iniciar ou encerrar um SCO, ir ao próximo ou anterior, e assim por diante.
`tb`	Encerramento. Coordena as tarefas quanto ao encerramento de um SCO.
`sb`	Seqüenciamento. Analisa a árvore de atividades e altera o estado das atividades de acordo com as regras de seqüenciamento e navegação.
`DB`	Disponibilização. Define que recurso será enviado ao navegador.
`sr`	Seleção e aleatoriedade. Conforme as regras de seqüenciamento, define que SCO deve ser selecionado. Os objetos desse pacote são referenciados dentro dos objetos do pacote sb.
`rb`	Retropropagação. Essa fase acontece somente quando o SCO é encerrado. Os objetos desse pacote são acessados a partir do pacote tb.
`up`	São as classes utilitárias para o processo.

O processo de seqüenciamento atua sobre a árvore de atividades, que é uma instância do pacote de conteúdo para um usuário. As classes apresentadas na Figura 27, que correspondem ao pacote de conteúdo, são a base para se criar a árvore de atividades. A Figura 29 ilustra a representação em classes da árvore de atividades. A árvore é uma instância de `ActivityTree`, criada baseando-se no pacote de conteúdo passado como parâmetro no método `build()` da classe `ActivityTreeFactory`. O objeto `ActivityTree` é formado por um conjunto de objetos `LearningActivity`. Cada instância desse objeto corresponde a uma atividade de aprendizagem. Nele são armazenados dados referentes à interação do usuário com o conteúdo, como a quantidade de vezes acessada, tempo utilizado e assim por diante. O objeto `ActivityTree` mantém a atividade atual (aquela em que o usuário está interagindo) e a atividade alvo. Essa última é utilizada

quando o usuário clica em uma atividade a partir de um menu, por exemplo, em uma árvore que fica normalmente disponível na lateral esquerda da página.

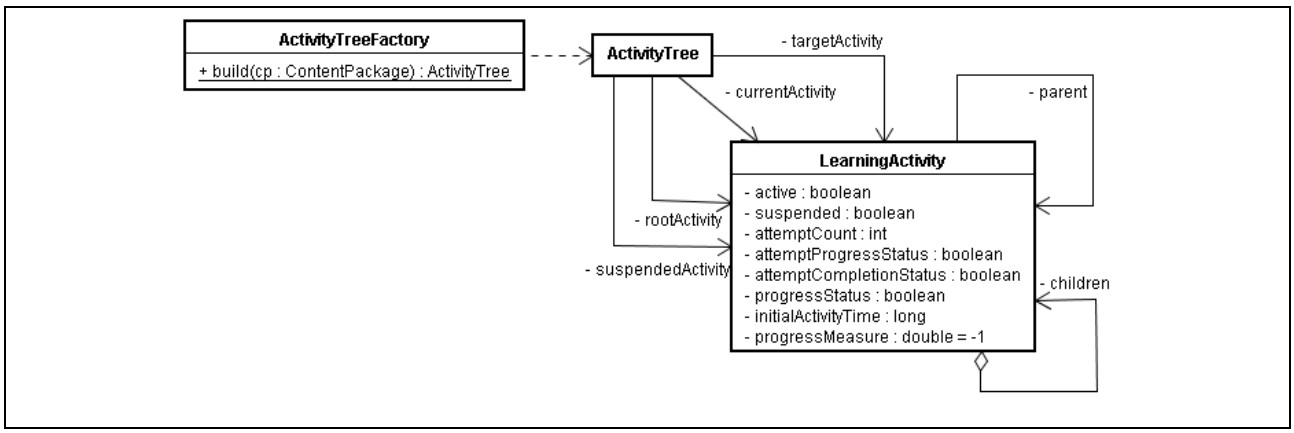


Figura 29. Diagrama das classes representando a árvore de atividades

A Figura 30 apresenta a classe, do pacote `op`, que coordena e centraliza o seqüenciamento. A partir dela, são acionados os objetos dos demais pacotes, apresentados na Tabela 2, das fases do processo de seqüenciamento. Toda a requisição que for feita para o sistema, será tratada pelo objeto `OverallSequencingProcess`. Ele possui dois métodos: um para executar a requisição e outra para verificar a validade da requisição. Os tipos de requisição podem ser iniciar, suspender, encerrar, próxima atividade, e assim por diante. Ainda em ambos os métodos, o segundo parâmetro é a árvore de atividades onde será realizada a ação da requisição.

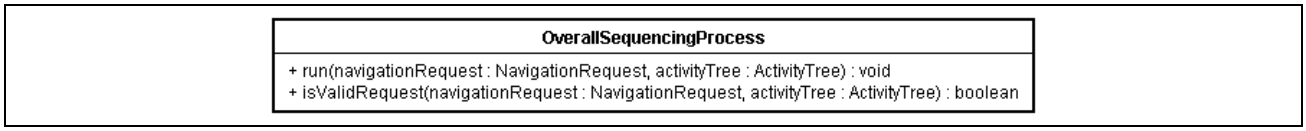


Figura 30. Classe que gerencia o processo de seqüenciamento

A última parte especificada foi o ambiente de execução (RTE). Para atendê-la é necessário implementar a API de comunicação (Figura 11) e os 27 *Data Models*. A Figura 31 apresenta o diagrama de classes com a solução especificada. A interface `API` contém os métodos combinando com as funções exigidas da API de comunicação, e a classe `APIImplementation` realiza essa interface. Ela controla as interações dos SCOs com a árvore de atividades, coordenando os ciclos de vida dos SCOs.

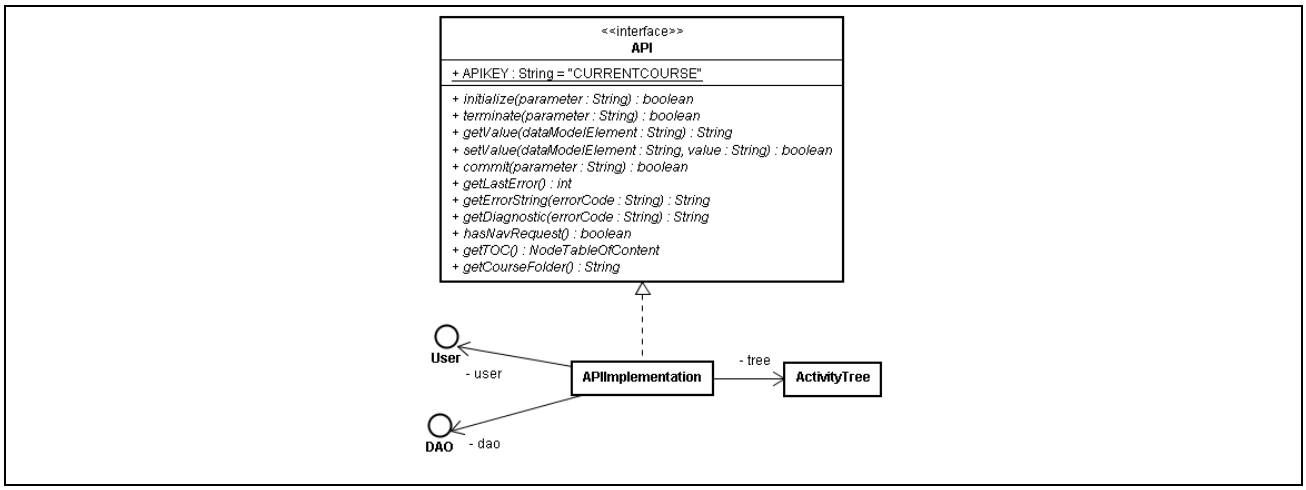


Figura 31. Diagrama de classes da API de comunicação SCO com LMS

Baseando-se na API de comunicação, os SCOs utilizam a função GetValue() para obter valores do LMS, e a função SetValue() para atribuir ou alterar valores. Ambas as funções necessitam do *Data Model* que identifica o tipo de informação que pode ser obtido ou alterado. Os 27 elementos do *Data Model* foram mapeados para classes, todas realizando a interface `DataModelCommand`, como ilustrado na Figura 32. A interface `DataModelCommand` contém basicamente um método `getValue()` e outro `setValue()`.

A classe `DataModelCommandManager` organiza os 27 elementos, facilitando a manutenção quando uma nova versão do SCORM adicionar novos elementos. Quando o método `getValue()` de `APIImplementation` é invocado, vem como parâmetro o identificador do *Data Model*. Esse parâmetro é repassado para o método `getValue()` da classe `DataModelCommandManager`, e através desse identificador localiza o objeto `DataModelCommand`. Com a referência do objeto, então é invocado seu método `getValue()`. O mesmo princípio é seguido para executar o método `setValue()`.

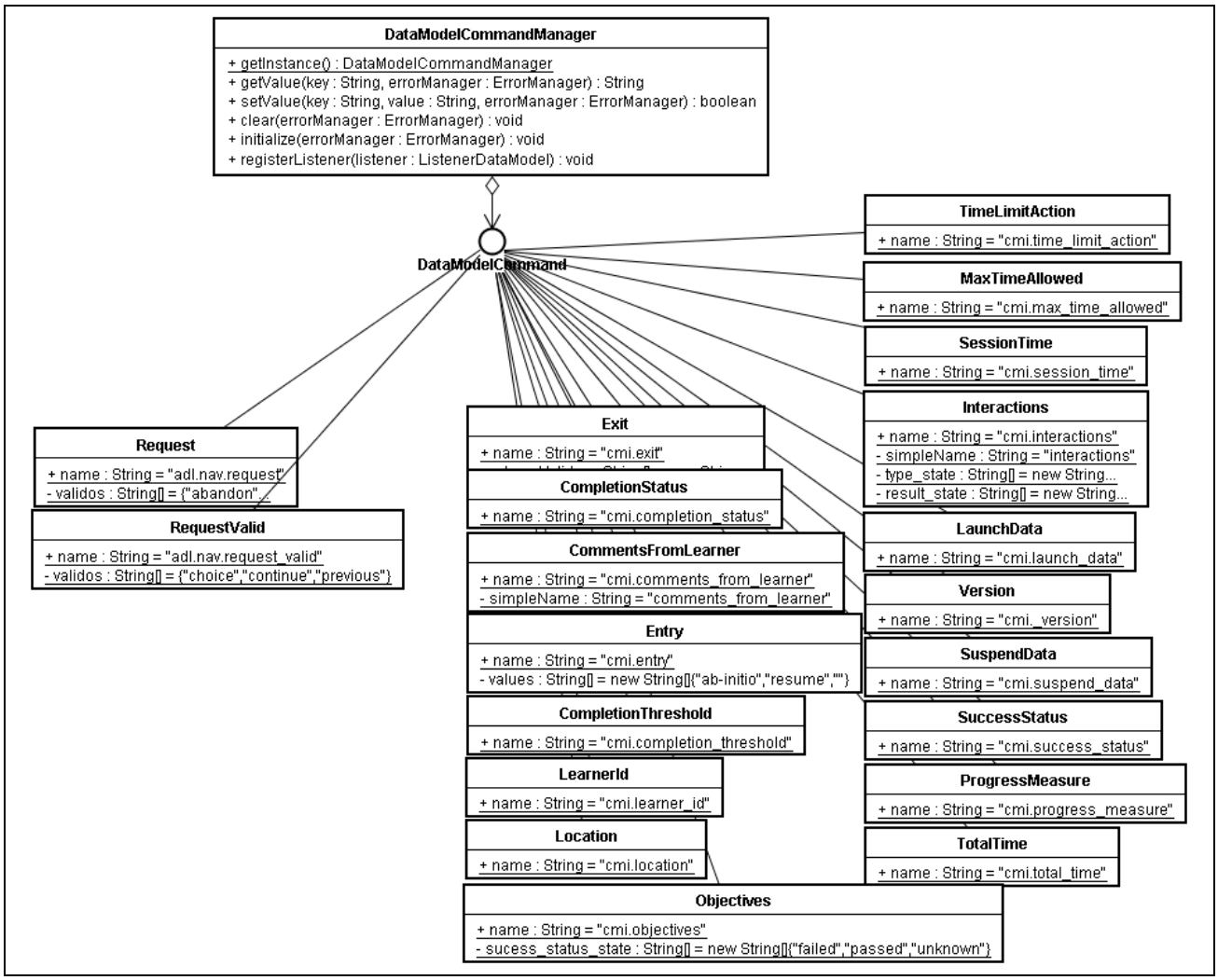


Figura 32. Diagrama de classes com a estrutura para gerenciamento do *Data Model*

Ainda na Figura 31, observam-se duas interfaces associadas à classe `APIImplementation`. Como essa classe também é responsável por coordenar a persistência da interação do usuário com o curso (através da função Commit()), tanto na gravação quanto no carregamento dos dados, ela precisa de um mecanismo de persistência. Esse mecanismo foi desenvolvido seguindo o padrão DAO (*Data Access Object*) que, segundo Sardagna e Vahldick (2008), "consiste em abstrair o mecanismo de persistência utilizado na aplicação. A camada de negócios acessa os dados persistidos sem ter conhecimento se os dados estão em um banco de dados relacional ou um arquivo XML". Para isso foi especificada a interface `DAO` que possui métodos para persistir dados dos cursos, usuários e o estado das interações das atividades desses usuários. A interface `User` é utilizada para determinar as informações mínimas que um usuário precisa ter para seu

gerenciamento. A `APIImplementation` não cria ou busca o usuário, mas ele recebe um objeto que segue interface.

3.2.2 Implementação

Essa subseção apresenta alguns detalhes quanto às decisões de implementação do módulo SCORM.

Para realizar a interpretação do arquivo de manifesto foi utilizado o framework Apache Digester (APACHE COMMONS, 2007) que interpreta um arquivo XML, e instancia objetos, inicializa atributos e cria relacionamentos entre os objetos. A estrutura do arquivo de manifesto é bastante flexível, e por conseqüência exige complexidade na sua interpretação. Com o Apache Digester se especifica a máscara do elemento XML, e a regra que deve ser executada quando ela é encontrada no arquivo: a classe que ele deve instanciar, ou o método a ser invocado. A Figura 33 apresenta o trecho de código da classe `ContentPackageReader` com as regras de mapeamento do elemento `<item>`.

```
...
d.addObjectCreate("*/item", Item.class);

d.addObjectCreate("*/item/adlnav:presentation", Presentation.class);
d.addSetNext("*/item/adlnav:presentation", "setAdlNavPresentation");

d.addObjectCreate("*/item/adlnav:presentation/adlnav:navigationInterface",
                    NavigationInterface.class);
d.addSetNext("*/item/adlnav:presentation/adlnav:navigationInterface",
                    "setNavigationInterface");

d.addCallMethod("*/item/adlnav:presentation/adlnav:navigationInterface"+
                    "/adlnav:hideLMSUI", "addHideLMSUI", 0);

addImssSequencing(d, "*/item");

d.addCallMethod("*/item/title", "setTitle", 0);

d.addCallMethod("*/item/adlcp:dataFromLMS", "setDataFromLMS", 0);
d.addCallMethod("*/item/dataFromLMS", "setDataFromLMS", 0);

d.addCallMethod("*/item/adlcp:timeLimitAction", "setTimeLimitAction", 0);
d.addCallMethod("*/item/timeLimitAction", "setTimeLimitAction", 0);

d.addSetProperties("*/item");
d.addSetNext("*/item", "addItem");

d.addCallMethod("*/item/adlcp:completionThreshold",
                    "setCompletionThreshold", 0);
...
```

Figura 33. Trecho de código do `ContentPackageReader`

A implementação das classes de SN foram baseadas no pseudocódigo que consta em ADL (2006d). Aquela documentação divide em módulos cada parte do processo de sequenciamento. Cada módulo desses foi convertido para uma classe. Quando um módulo precisa chamar outro, é utilizado o objeto `ProcessProvider`. Essa classe foi implementada seguindo o padrão de projeto Singleton (GAMMA, 2000), que apresenta uma solução quando se deseja ter um único objeto da classe. A classe `BuildProcessProvider20043rdEdition` é responsável pela inicialização do objeto `ProcessProvider`, e por isso ela deve ser invocada antes dele. Essa classe segue o padrão de projeto Builder (GAMMA, 2000), que coordena a criação e inicialização de outros objetos. Quando uma nova versão do SCORM for liberada, se desenvolve outra classe `BuildProcessProviderXXX`, que cria outros objetos para o `ProcessProvider`.

O módulo SCORM é empacotado no arquivo CELINESCORM.JAR e pode ser utilizado por aplicações Java independente do contexto (web, interface gráfica, console ou dispositivos móveis). Uma vantagem na adoção dessa estratégia é que podem ser automatizados os testes quanto ao processo de seqüenciamento, ou o desenvolvimento de aplicações em qualquer direção além do propósito do SCORM que são aplicações web como os LMS.

O CELINESCORM depende de três pacotes: commons-beanutils, commons-digester e commons-logging, que possuem a implementação do Apache Digester.

3.3 LMS

A seção anterior apresentou o módulo que atende a especificação SCORM. Através dele é possível executar um conteúdo SCORM, obedecendo a sua estrutura e seqüenciamento. Esse módulo não oferece ainda os recursos suficientes para a montagem de uma aplicação web que executa, gerencia e adapta pacotes SCORM. Essa seção apresenta o módulo LMS, que usa o módulo SCORM, que fornece recursos para o desenvolvimento de um LMS.

Essa seção foi dividida em duas subseções: na primeira é apresentada a especificação e dependências do módulo, e a segunda lista detalhes de implementação do módulo.

3.3.1 Especificação

O requisito funcional 6 é atendido pelo módulo SCORM, os demais são de responsabilidade do módulo LMS. Esses recursos são quase todos disponibilizados ao desenvolvedor por *tags*

customizadas. Elas são responsáveis por gerar código HTML nas páginas JSP, tanto para estruturar as informações a serem apresentadas, quanto prepararem as requisições para a aplicação web. Os requisitos 10 e 11 dizem respeito a modificações de estruturas de pacotes de conteúdo, e não são tratados pelas *tags*, porém esse papel deve ser feito antes de alcançar as *tags*, o que será explicado mais adiante.

O tratamento dessas requisições é apresentado no diagrama da Figura 34. Cada tipo de requisição, que representa uma ação a ser executada pelo componente, foi mapeado para uma classe que realiza a interface Command. O *servlet* LMS recebe as requisições, seleciona e executa o objeto Command de acordo com essa requisição. A Tabela 3 relaciona cada uma das classes Command com um resumo de suas responsabilidades e a associação com os requisitos funcionais.

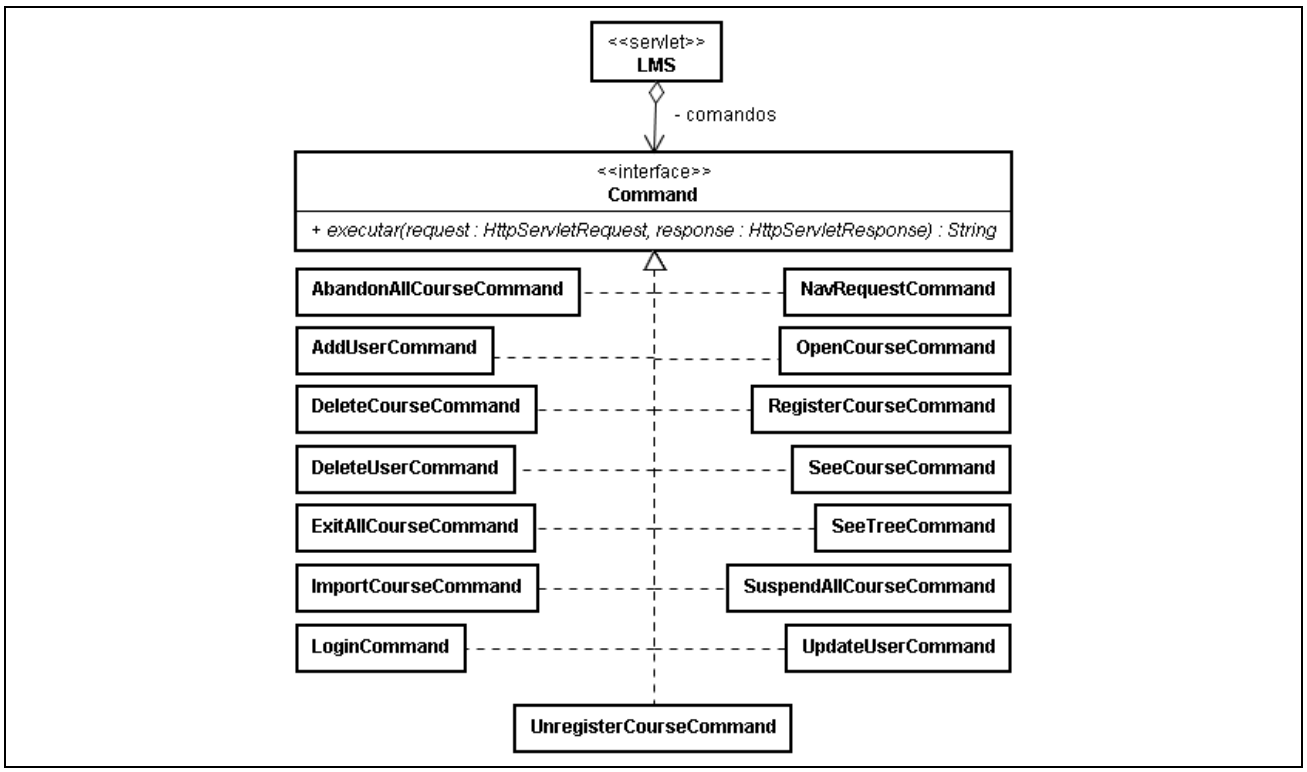


Figura 34. Diagrama de classes com a estrutura central do módulo LMS

Tabela 3. Rastreabilidade entre ações e requisitos funcionais

Ação	RF	Descrição
abandonallcourse	6	Requisição de navegação do tipo *AbandonAll*
exitallcourse	6	Requisição de navegação do tipo *ExitAll*
suspendallcourse	6, 7	Requisição de navegação do tipo *SuspendAll*
adduser	3	Na requisição vêm os dados de um usuário, e ele é adicionado no sistema.
deleteuser	3	Na requisição vêm os dados de um usuário, e ele é excluído do sistema.
updateuser	3	Na requisição vêm os dados de um usuário, e ele é alterado no sistema.
importcourse	1	Recebe um arquivo zip e adiciona ele ao sistema.
deletecourse	2	Marca como removido um curso. A exclusão definitiva acontece quando a aplicação é reiniciada.
opencourse	6, 7	Abre um conteúdo para o usuário interagir.
navrequest	6	Requisição para o conteúdo.
registercourse	4	Registra o usuário em um curso.
unregistercourse	4	Desfaz a relação entre o usuário e o curso.
seetreecourse	6	Apresenta a árvore de atividades de um curso.
login	9	Efetua a autenticação do usuário no sistema.

As três primeiras classes referem-se à requisições de encerramento do SN. Quando um conteúdo está sendo exibido, o usuário pode suspender (*suspendall*), abandonar (*abandonall*) ou encerrar (*exitall*). Ao suspender, todas as interações são salvas para o usuário continuar em outro instante. Ao abandonar, as interações da sessão atual não são salvas. Caso tenha vindo tenha vindo de uma seção suspensa, continua valendo os dados daquela sessão. A ação de encerrar implica em eliminar todos os dados sobre as interações.

A ação *opencourse* é executada quando o usuário deseja abrir um curso. Como resultado dessa ação, é apresentado ao usuário dois frames organizados horizontalmente: à esquerda fica a árvore de atividades, e à direita as páginas de conteúdo. A árvore de atividades é montada através da ação *seetreecourse*. Cada item representa um link de uma atividade na árvore. Quando um item é clicado, é disparada a ação *navrequest*. Essa mesma ação é disparada quando uma atividade envia a solicitação de uma requisição, por exemplo, para ir à atividade anterior ou posterior.

Para realizar essas requisições e, por conseqüência, disparar essas ações, o desenvolvedor precisaria conhecer as URLs na montagem das páginas JSP. Porém, visando facilitar a implementação dessas páginas, foram disponibilizadas *tags* customizadas, que trazem a vantagem de possuírem documentação embutida, e melhoram a legibilidade das páginas pois a sua sintaxe é baseada nas *tags* HTML. Ainda, na Tabela 3, podem se notar que não existe associação para os requisitos 5, 8, 10 e 11. Os dois primeiros também foram resolvidos com *tags* customizadas, e a solução dos outros dois será discutida mais adiante. A Tabela 4 apresenta o mapeamento de cada uma das ações da Tabela 3 com uma *tag* customizada.

Tabela 4. Mapeamento entre ações e *tags* customizadas

Ação	***Tag* Customizada**
abandonallcourse	`abandonAllCourse`
exitallcourse	`exitAllCourse`
suspendallcourse	`suspendAllCourse`
adduser	`saveUserForm`
deleteuser	`deleteUser`
updateuser	`saveUserForm`
importcourse	`formImportCourse`
deletecourse	`deleteCourse`
opencourse	`listCourses`
navrequest	`--`
registercourse	`registerCourse`
unregistercourse	`unregisterCourse`
seetreecourse	`--`
login	`loginForm`

Cada uma das *tags* está relacionada com uma classe que provê os recursos às páginas JSP. Com exceção da *tag* `listCourses`, as demais são descendentes da classe `NextURLBodyTagSupport`. Essa classe contém o atributo `nextURL` que é necessário para determinar a continuação do fluxo de navegação. Esse atributo serve para indicar qual a página a ser exibida após a ação ser executada. Por exemplo, ele determina qual a página a ser exibida após a ação *AddUser* ser executada.

As *tags* `abandonAllCourse`, `exitAllCourse` e `suspendAllCourse` também possuem um atributo `varLink`. Esse atributo contém o link formatado para a requisição relacionada à *tag*. A Figura 35 mostra o exemplo de utilização da *tag* `suspendAllCourse`. A página a ser executada após o sistema suspender o curso é `listcourses.jsp` e a variável `link` contém a URL pronta para realizar a requisição *suspendall*. Essa variável é embutida na *tag* `<a>`.

```
...
<celine:suspendAllCourse nextURL="listcourses.jsp" varLink="link">
   <a href="${link}" target="corpo">Salvar e sair</a>
</celine:suspendAllCourse>
...
```

Figura 35. Exemplo de utilização da *tag* `suspendAllCourse`

A *tag* `saveUserForm` monta um formulário com os dados do usuário. Ela é utilizada tanto para incluir quanto para alterar usuários. Contudo, para se alcançar na página que contém essa *tag* é necessário utilizar outro conjunto de *tags*.

Primeiramente, uma página lista todos os usuários cadastrados, e para isso se utiliza a *tag* `listUsers`. Ela possui dois atributos: `varName` e `varIndex`. O primeiro fornece à página o nome do usuário e o segundo a posição do usuário em relação ao primeiro da lista. Outra tag, `userInfo` pode ser utilizada para fornecer a senha e o tipo do usuário (se administrador ou normal).

Para determinar que o formulário da *tag* `saveUserForm` refere-se a uma alteração, deve-se alcançar essa página utilizando a *tag* `editUser`. Para a inclusão não faz-se tratamento algum. A Figura 36 ilustra a utilização desse conjunto de *tags*. Essa página se chama `manageusers.jsp` e `edituser.jsp` é a que possui o formulário, e não está representada no exemplo. Nesse exemplo também consta a *tag* `deleteUser` para criar o link que permite a exclusão do usuário.

```
...
<celine:listUsers varName="name" varIndex="index">

   <celine:editUser name="${name}" nextURL="edituser.jsp"
                                    varLink="linkEdit">

         <a href="${linkEdit}">${name}</a>

   </celine:editUser>

   <celine:deleteUser name="${name}" nextURL="manageusers.jsp"
                                      varLink="linkDelete">

         <a href="${linkDelete}">Excluir</a> 

   </celine:deleteUser>
</celine:listUsers>
...
```

Figura 36. Exemplo de listagem para acesso ao formulário de cadastro de usuários

De forma semelhante ao gerenciamento de usuários, também existe uma *tag* para listagem de cursos: `listCourses`. Ela elenca todos os cursos cadastrados fornecendo o link para requisitar a ação *opencourse*. Além disso, a *tag* provê informação para a página indicando se o curso está ou não registrado para o usuário. A Figura 37apresenta um exemplo de listagem de cursos, e nesse caso, somente mostra aqueles que o usuário corrente está registrado. Observa-se também o uso das *tag* `deleteCourse`. Essa *tag* povoa uma variável indicada no atributo `varLink` contendo a ação *deletecourse*. Assim que a ação for executada, a requisição é direcionada para a página `listcourses.jsp`. A *tag* `courseInfo` exibida no exemplo é utilizada para obter dados (nome da pasta e título) de um curso baseado na identificação do curso.

```
...
<celine:listCourses varId="id" varName="nome" varLink="link"
                                    varStatusCourseRegistered="status">
   <c:if test="${status == true}">
         <a href="${link}">${nome}</a><br/>
         <celine:deleteCourse courseId="${id}" nextURL="listcourses.jsp"
                                                varLink="linkDelete">
               <a href="${linkDelete}">Excluir</a>
         </celine:deleteCourse>
         <celine:unregisterCourse courseId="${courseId}" varLink="linkUnreg"
                                    nextURL="listcourses.jsp" >
               <a href="${linkUnreg}">Cancelar registro</a>
         </celine:unregisterCourse>
   </c:if>

   <c:if test="${status == false}">
         ${nome}
         <celine:registerCourse courseId="${courseId}" varLink="linkReg"
                                    nextURL="listcourses.jsp" >
               <a href="${linkReg}">Registrar</a>
         </celine:registerCourse>
   </c:if>

   <celine:courseInfo courseId="${id}" varFolderName="folder">
         ${folder}
   </celine:courseInfo>

</celine:listCourses>
...
```

Figura 37. Exemplo de listagem de cursos

A Figura 37 também exemplifica o uso das *tags* `registerCourse` e `unregisterCourse`. A condição "*status == true*" indica que o usuário corrente está registrado. Nesse caso, ele tem a opção de cancelar o registro. A criação do link é feita dentro de `unregisterCourse`. No outro teste, se o usuário estiver registrado, então é ilustrado o uso de `registerCourse`.

A *tag* `formImportCourse` gera um formulário HTML direcionando para ação *importcourse*. Esse formulário contém um campo para a digitação de uma identificação única, a ser usada pelo sistema internamente, o título do curso, e o arquivo zip. A identificação foi exemplificada na Figura 37, onde as tags tem o atributo `courseId`. Ainda, essa *tag* usa requisições AJAX para apresentar o progresso de upload do arquivo.

Para encerrar a lista de ações da Tabela 4, a *tag* `loginForm` produz um formulário HTML contendo os campos para digitação do nick e senha do usuário, e submissão para a ação *login*.

Além das *tags* listadas na Tabela 4, ainda existem outras que não geram ações, mas buscam dados do sistema para fornecer dados para a página formatá-los. Todas as *tags* estão listadas na

Tabela 5 com uma breve descrição delas, e aquelas marcadas com um "*" são já descritas nos parágrafos anteriores.

Tabela 5. *Tags* disponíveis no CELINELMS.

Nome da *Tag*	**Descrição**
`abandonAllCourse*`	Link para abandonar o curso.
`courseInfo*`	Fonte de dados para extrair informações sobre um curso.
`courseStatus`	Lista a árvore de atividades de um curso contendo informações sobre cada item, em relação às interações de um usuário.
`deleteCourse*`	Link para marcar um curso como removido.
`deleteUser*`	Link para exclusão do usuário.
`editUser*`	Link para uma página que tenha o `saveUserForm`, indicando que vai alterar um usuário.
`exitAllCourse*`	Link para sair do curso.
`formImportCourse*`	Formulário para importação de cursos.
`listCourses*`	Listagem de cursos.
`listCoursesPractice`	Lista os dados estatísticos sobre cursos que foram visitados por um usuário.
`listUsers*`	Listagem de usuários.
`listUsersPractice`	Lista os dados estatísticos sobre os usuários que já interagiram com cursos.
`loginForm*`	Formulário para autenticação.
`registerCourse*`	Link para o usuário registrar-se em um curso.
`saveUserForm*`	Formulário para cadastro de usuários.
`seeContentCourse`	Lista a árvore de atividades de um curso.
`suspendAllCourse*`	Link para suspender o curso.
`unregisterCourse*`	Link para o usuário cancelar o registro com o curso.
`user`	Fornece dados sobre o usuário corrente.
`userInfo*`	Fonte de dados para extrair informações sobre um usuário.

Conforme já descrito, ainda dois dos requisitos funcionais que utilizam *tags* não foram explanados. O primeiro deles é o requisito *5. Visualizar a árvore de atividades do pacote;*. Esse requisito é atendido pela *tag* `seeContentCourse`. Ela fornece uma listagem com toda a organização do conteúdo, e o desenvolvedor pode então fazer uma página que mostre essa lista em forma de árvore.

O outro requisito – *8. Relatórios de utilização dos pacotes SCORM;* – é atendido pelas *tags* `listUsersPractice`, `listCoursesPractice` e `courseStatus`, normalmente para serem usadas nessa seqüência.

A primeira *tag* lista todos os usuários que já interagiram com algum curso. Para cada usuário é listado o seu nome, quantos cursos acessou, quantia de tempo (em milisegundos) dispendido nesses acessos, e quando acessou pela última vez algum curso.

A *tag* `listCoursesPractice` apresenta os cursos acessados de um usuário. Para cada curso acessado é listado o nome dele, quantia de tempo (em milisegundos) dispendido, se ele foi visitado por completo, se ele foi satisfeito, e sua pontuação. Essas três informações dependem das regras de sequenciamento do curso.

E por último, a *tag* `courseStatus` lista todas as atividades de um curso, junto das seguintes informações: se está completo, se está satisfeito, sua pontuação, e a quantia de tempo (em milisegundos) dispendido.

Para a adaptação do conteúdo SCORM, e atender os requisitos funcionais 10 e 11, decidiu-se por adicionar uma classe que modificará o resultado de algumas ações. Essa classe deve realizar a interface `LMSIntegration`, representada na Figura 38.

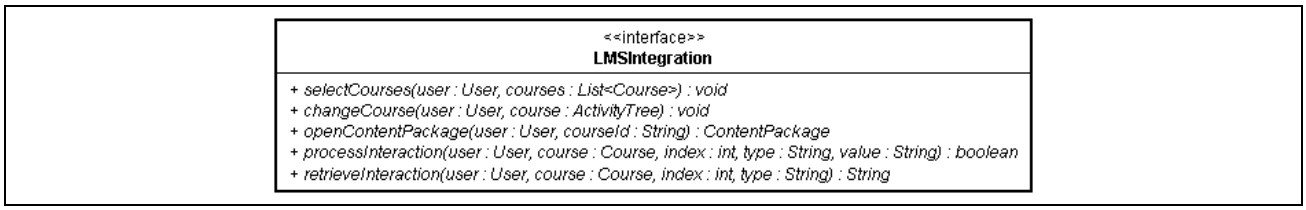


Figura 38. Interface a ser implementada para permitir adaptação do conteúdo

Os métodos a serem implementados atendem os seguintes objetivos:

- Listar cursos (`selectCourses`): fornece a lista de cursos disponíveis ao usuário. Ao método é entregue uma lista de todos os cursos registrados pelo usuário, porém, ela pode ser alterada para que apresente, por exemplo, somente aqueles que o usuário esteja apto a executar segundo o desempenho das atividades anteriores. Esse método é chamado pela *tag* `listCourses`;
- Abrir um curso (`openContentPackage`): esse método é executado antes de abrir um curso, e corresponde à geração adaptativa de conteúdo. Através dele, o ambiente monta um curso juntando partes de outros cursos. As atividades podem ser desabilitadas ou escondidas, para que em outro momento possam estar disponíveis ao aluno. Esse método é invocado na ação *opencourse*;
- Alterar a árvore de atividades atual (`changeCourse`): esse método corresponde à geração dinâmica de conteúdo. Esse método é chamado toda vez que o usuário encerra

uma atividade (SCO). O ambiente pode avaliar o desempenho do aluno, e habilitar ou apresentar novas atividades previstas quando da abertura do curso;

- `cmi.interactions` alterado (`processInteraction`) ou solicitado (`retrieveInteraction`): esse é o *Data Model* sugerido pelo modelo SCORM para manter dados específicos do curso. Quando um curso precisa guardar e obter informações específicas, por exemplo, as respostas de um questionário, o *Data Model* `cmi.interactions` é o identificador aconselhado para isso.

Para auxiliar na montagem do curso e na atualização da árvore de atividades, existem duas classes: `EasyContentPackage` e `EasyActivityTree`. Ambas possuem métodos que abstraem o entendimento do desenvolvedor em relação à especificação SCORM.

Através da classe `EasyContentPackage` pode-se adicionar pacotes SCORM, ou partes deles, e gerar um novo curso baseado nos pacotes previamente adicionados. Essa classe é utilizada para geração de um curso quando o usuário entra, ou seja, quando o sistema precisa abrir um curso.

A classe `EasyActivityTree` possibilita ao desenvolvedor modificar a estrutura e as regras de seqüenciamento da árvore de atividades. Essa modificação inclui a adição de novas atividades.

Com isso, encerra-se a especificação dos requisitos funcionais. Quanto aos requisitos não-funcionais, o requisito *1. Execução dos pacotes em conformidade com a versão 2004 3ª edição do SCORM;* é atendido pelo módulo CELINESCORM; os requisitos *3. Desenvolvido em Java;* e *4. Disponibilizar tags customizadas em JSP para acesso aos recursos do componente;* foram amplamente discutidos no decorrer dessa seção; e o requisito *5. Ser disponibilizado através de arquivos JAR.* implica na divisão dos módulos nos arquivos CELINESCORM.JAR e CELINELMS.JAR.

Para seguir o requisito *2. Persistência dos dados em arquivos XML, banco de dados relacional ou por customização do desenvolvedor;*, o módulo SCORM tem especificada a interface `DAO`, e o módulo LMS realiza essa interface com duas classes `XMLDAO` e `RDBDAO`, uma para cada tipo de persistência.

No caso do XML, os dados dos usuários, cursos e cursos registrados do usuário são armazenados em um arquivo XML, e um arquivo separado para cada interação de um curso para um usuário.

No caso de BDR (Banco de Dados Relacional), é necessária a criação de um esquema conforme o modelo entidade-relacionamento apresentado na Figura 39. A descrição de cada entidade está relacionada na Tabela 6.

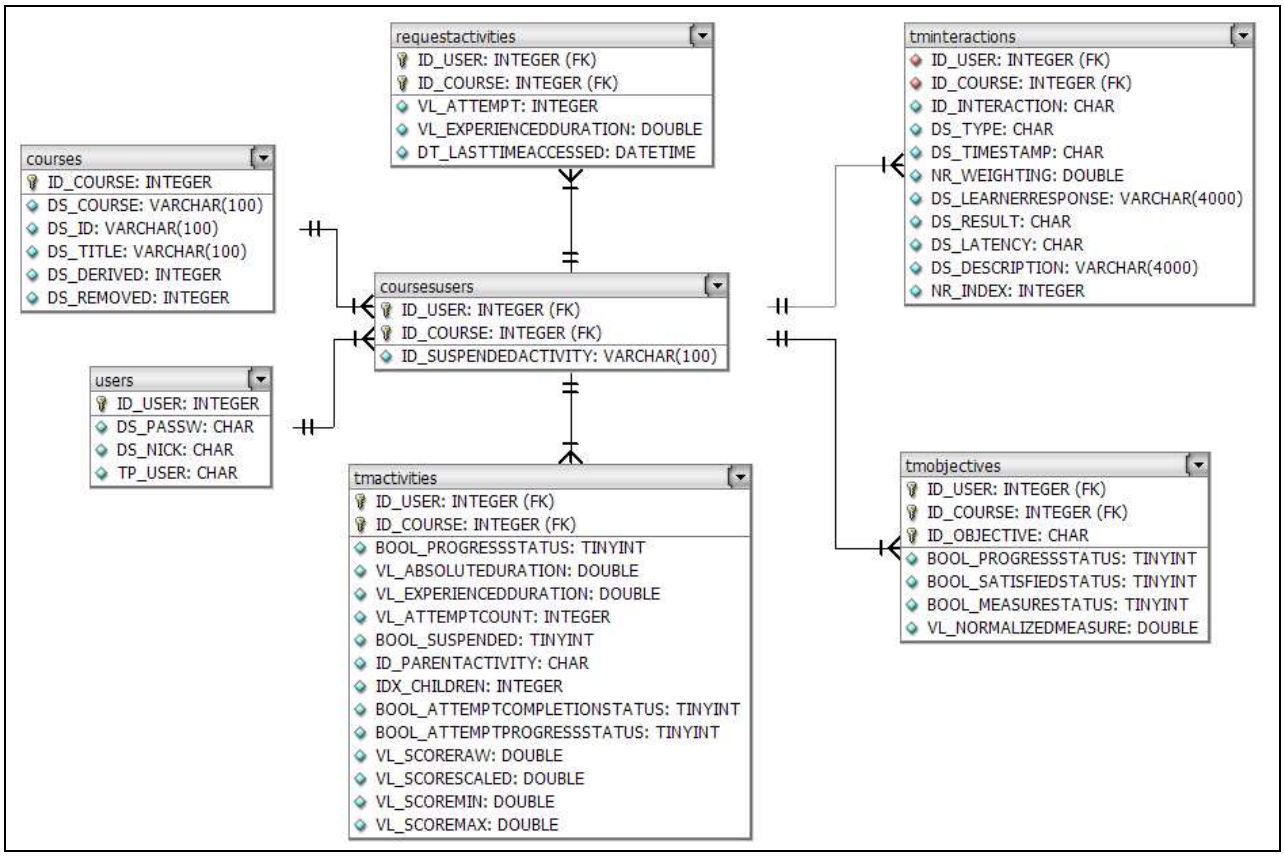


Figura 39. Modelo entidade-relacionamento para o mecanismo de persistência `RDBDAO`

Tabela 6. Dicionário de dados do CELINE.

Tabela	**Descrição**
`courses`	Pacotes de conteúdo SCORM importados.
`users`	Cadastro de usuários.
`courseusers`	Cursos registrados de cada usuário, e a atividade que foi suspensa.
`tmactivities`	Interação das atividades do usuário com o curso. Utilizado para manter a árvore de atividades enquanto não encerra o curso.
`tmobjectives`	Valores dos objetivos da interação com o curso. Mantém enquanto não encerra o curso.
`tminteractions`	Valores dos elementos `cmi.interactions`. É mantido enquanto não encerra o curso.
`requestactivities`	Mantém os dados estatísticos sobre as interações com os cursos.

Com todos os requisitos especificados, é necessário definir a arquitetura do componente. O desenvolvedor precisa definir o mecanismo de persistência e a pasta onde os arquivos zip precisam descompactar. A Figura 40 apresenta o diagrama de componentes do CELINE, representando essas configurações. Além disso, também pode ser configurado a classe que adapta o conteúdo SCORM. Para as *tags* customizadas as páginas precisam ter acesso ao TLD.

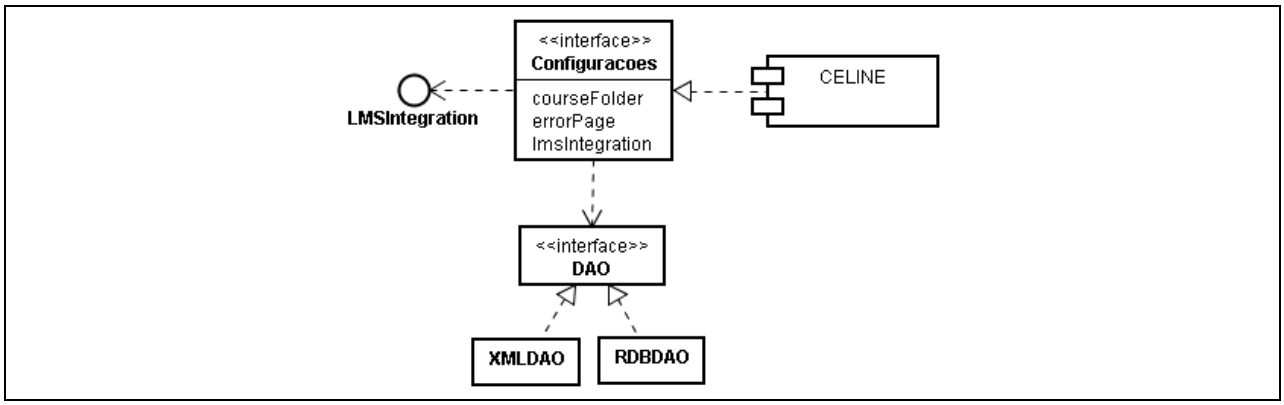


Figura 40. Diagrama de componentes do CELINE

3.3.2 Implementação

Essa subseção apresenta os detalhes de implementação do módulo LMS, ao mesmo tempo em que demonstra como acoplar o componente à aplicação. Todo o código implementado para esse módulo foi compactado em um arquivo chamado CELINELMS.JAR.

Para que a classe `LMS`, que é um *servlet*, possa receber as requisições da aplicação, é preciso que ela seja mapeada no arquivo `web.xml`, como demonstrado na Figura 41. Esse é o primeiro passo para conectar o componente CELINE à aplicação web.

```
...
<servlet>
   <servlet-name>lms</servlet-name>
   <servlet-class>br.univali.celine.lms.core.LMS</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>lms</servlet-name>
   <url-pattern>/lms</url-pattern>
</servlet-mapping>
...
```

Figura 41. Mapeando o *servlet* LMS na aplicação

A classe `LMS` tem aplicado o padrão Command (GAMMA, 2000): quando o objeto é iniciado, é povoado o atributo `comandos` (Figura 34), onde cada tipo de requisição possui uma classe associada e por conseqüência, um objeto é criado. Quando uma requisição é recebida por esse *servlet*, a distinção de que comando executar é baseado no parâmetro da requisição chamado `action`. Como resultado da requisição é retornado o conteúdo da página solicitada.

Existe um tipo de requisição chamado `file`, que solicita o conteúdo de um arquivo HTML ou JavaScript. Alguns arquivos desse tipo foram desenvolvidos sem serem respostas de requisições. Para que o desenvolvedor não precise copiá-los em uma pasta de sua aplicação, esses arquivos são compactados no CELINELMS.JAR. A classe `FileHandler` lê os arquivos solicitados e devolve o conteúdo da resposta.

Quando uma ação *opencourse* é solicitada, se cria um objeto `APIImplementation`, e ele é adicionado na sessão do usuário. Quando requisições do tipo *navrequest* ou *exitallcourse* são solicitadas, o objeto é obtido da sessão, e então, através dos *Data Models*, interagisse com ele.

No CELINE foi utilizada a arquitetura AJAX para implementar a API de comunicação entre LMS e SCO. A vantagem em utilizar essa solução é que o cliente se torna leve, sendo dispensado download prévio da implementação da API, ou exigência de software instalado (por exemplo, com uma *applet* o computador do cliente precisa ter a distribuição do Java instalado). Porém, como as chamadas entre os SCOs e o LMS não podem ser assíncronas, utilizou-se a estrutura do AJAX, mas a comunicação com o servidor foi adaptada para ser realizada de forma síncrona.

O AJAX foi implementado usando o framework DWR (GETAHEAD, 2004). A motivação é a facilidade em implementar AJAX com o DWR. Primeiro implementou-se uma classe `APIInstance` com todos os métodos da `API`. Essa classe carrega da sessão a `APIImplementation`, e repassa o respectivo método invocado pelo SCO. Para que o framework saiba que objeto precisa ser tratado com requisições AJAX, foi necessário mapear em um XML a classe `APIInstance` nomeando-o como `API_1484_11`, que é o nome exigido para o objeto, ou seja, o SCO procura por um objeto com esse nome para invocar seus métodos. Além disso, também precisa adicionar uma configuração no arquivo `web.xml`, para que o framework seja reconhecido. A Figura 42 apresenta o trecho de código que deve se adicionar ao arquivo `web.xml`.

```
...
<servlet>
   <servlet-name>dwr-invoker</servlet-name>
   <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>dwr-invoker</servlet-name>
   <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
...
```

Figura 42. Mapeando AJAX na aplicação

Quando o curso é aberto, é criada uma página com dois frames, como pode ser visto na Figura 43. O frame da esquerda possui uma árvore com todos os itens da organização de conteúdo. Ao clicar em um desses itens, é atribuído ao *Data Model* `adl.nav.request` o valor `choice` e o identificador do item. Com isso, o LMS faz o processamento substituindo a página do lado direito. Para a renderização da árvore foi utilizado o componente Treeview (GUBUSOFT, 2008), que é um conjunto de scripts em JavaScript.

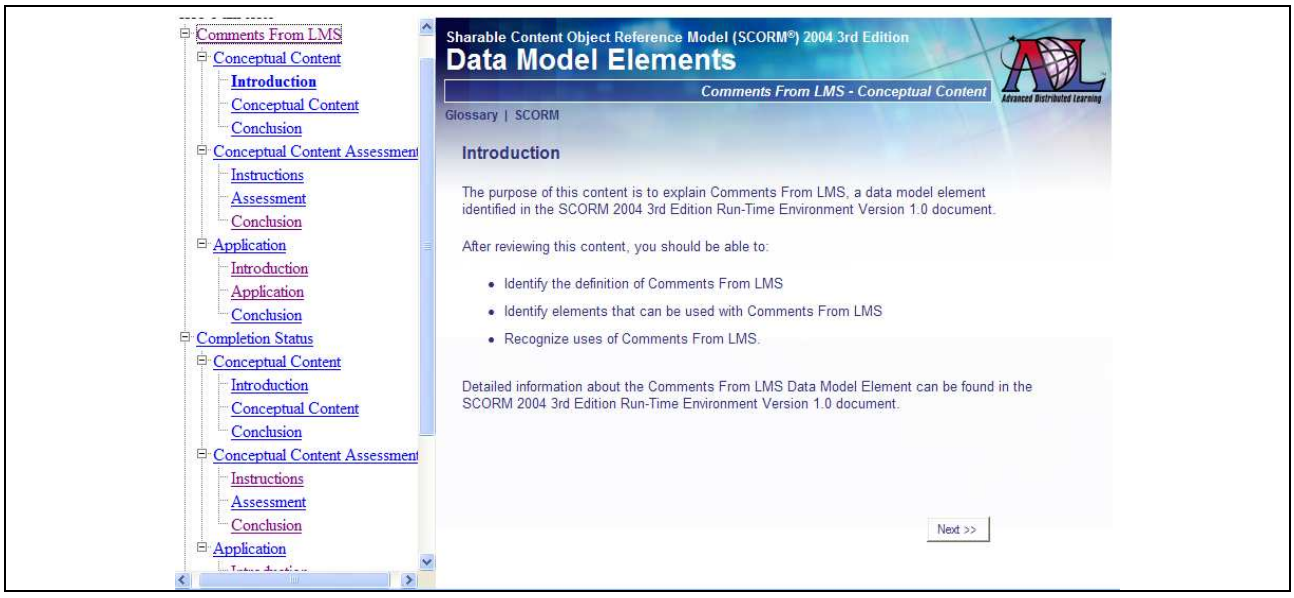


Figura 43. Exemplo de curso aberto

As configurações do componente, ilustrados na Figura 40, são feitas no arquivo XML de nome `celine-config.xml`, como exemplificado na Figura 44. Na *tag* `<courses-folder>` se determina a pasta onde os arquivos zip serão descompactados. A *tag* `<error-page>` define a página de erro, ou seja, quando algum erro acontecer no componente, essa é a página a ser exibida. Na *tag* `<lmsIntegration>` se fixa a classe que fará a adaptação do conteúdo. Essa é uma *tag*

opcional. E finalmente, na *tag* `<database-source>` se estabelece o mecanismo de persistência. No exemplo dessa figura é apresentado como utilizar o mecanismo XML: dentro da *tag* `<xml>` se coloca o arquivo de dados que a aplicação utilizará.

```
   <config>
      <courses-folder>cursos</courses-folder>
      <error-page>erro.jsp</error-page>
      <lmsIntegration>br.univali.bruce.ExemploLMSIntegration</lmsIntegration>
      <database-source>
            <xml>WEB-INF/celine.xml</xml>
      </database-source>
   </config>
```

Figura 44. Exemplo de arquivo de configuração do componente

Tanto as *tags*, quando o *servlet*, necessitam dessas informações. Por isso, o componente precisa ler esse arquivo o mais cedo possível. Essa leitura é efetuada por uma classe denominada `LMContextListener` que também precisa ser mapeada no arquivo `web.xml` da aplicação, como exemplificado na Figura 45.

```
...
<listener>
      <listener-class>
            br.univali.celine.lms.core.LMSContextListener
      </listener-class>
</listener>
...
```

Figura 45. Mapeando a classe invocada quando inicia a aplicação

Um método dessa classe é invocado quando o servidor inicia a aplicação. Esse método utiliza a classe `LMSConfig`, que segue o padrão Singleton (GAMMA, 2000), que lê o arquivo e inicializa seus atributos. Essa classe será solicitada sempre que outra classe precisar de alguma informação sobre as configurações do componente. A leitura do XML também é feita pelo framework Digester: a própria classe `LMSConfig` contém os atributos desse arquivo. A Figura 46 apresenta o trecho de código onde foi realizado o mapeado dessa classe.

```
Digester d = new Digester();
d.push(config);

d.addCallMethod("*/courses-folder", "setCoursesFolder", 0);
d.addCallMethod("*/error-page", "setErrorPage", 0);

d.addCallMethod("*/lmsIntegration", "setLMSIntegration", 0);

d.addObjectCreate("*/database-source/rdb", RDBDAO.class);
d.addSetNestedProperties("*/database-source/rdb");
d.addSetNext("*/database-source/rdb", "setDAO");

d.addObjectCreate("*/database-source/xml", XMLDAO.class);
d.addCallMethod("*/database-source/xml", "setFileName", 0);
d.addSetNext("*/database-source/xml", "setDAO");

d.addObjectCreate("*/database-source/bean", "class", DAO.class);
d.addSetNext("*/database-source/bean", "setDAO");

d.addRule("*/database-source/bean/bean-attribute", new BeanSetterAttribute());

String fileName = path + "/celine-config.xml";
java.io.File srcfile = new java.io.File(fileName);
d.parse(srcfile);
```

Figura 46. Código que mapeia o arquivo de configuração com a classe `LMSConfig`

A classe `XMLDAO`, que corresponde a um dos mecanismos de persistência, utiliza o framework XStream (XSTREAM, 2008) para leitura e gravação dos arquivos XML. O Digester, utilizado nas outras situações, servia para a leitura de arquivos de certa complexidade, porém não cabia para gravação. O XStream é um framework que persiste objetos e ele se encarrega do mapeamento para XML. Para utilizá-lo, basta informar as classes que devem ser persistidas, como demonstrado na Figura 47. A classe `CelineXML` é o contêiner principal. Esse objeto é apresentado na Figura 48. Quando precisa ler ou gravar alguma informação, solicita-se ao objeto XStream para fazê-lo junto de um objeto `CelineXML`.

```
...
XStream xstream = new XStream();

xstream.alias("celine", CelineXML.class);
xstream.alias("user", UserImpl.class);
xstream.alias("registeredCourse", RegisteredCourse.class);
xstream.alias("course", CourseImpl.class);
...
```

Figura 47. Mapeando o arquivo de dados com mecanismo XML

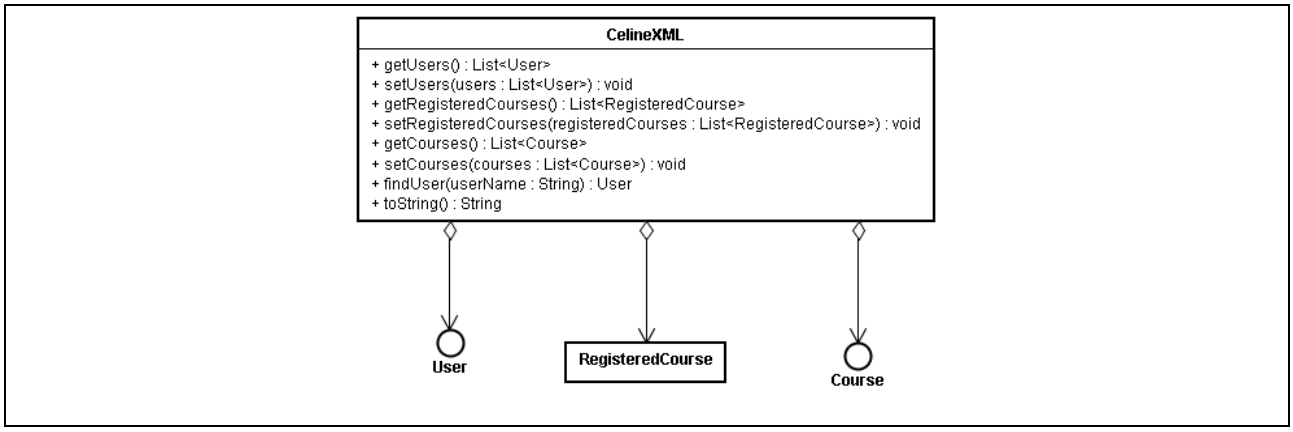


Figura 48. Classe contêiner do mecanismo XML de persistência

O outro mecanismo de persistência utiliza BDR, e para isso são necessárias a criação das tabelas apresentadas na Figura 39. O componente não cria essas tabelas automaticamente. O desenvolvimento desse mecanismo foi testado com MySQL, porém as expressões de consulta (SQL) seguiram o padrão ANSI. Além disso, como o acesso do Java é através de driver JDBC, e não nativamente ao banco, as expressões são ajustadas para o BD pelo próprio driver. A configuração no `celine-config.xml` é exemplificada na Figura 49. A *tag* `<driver>` determina o fabricante do BD que a aplicação utilizará. O driver é fornecido pelos fabricantes através de um arquivo JAR, que deve ser copiado para a pasta `WEB-INF\lib`. Na *tag* `<url>` se define o local do BD. As *tags* `<user>` e `<password>` são utilizadas para autenticação com o BD.

```
...
<database-source>
      <rdb>
            <driver>com.mysql.jdbc.Driver</driver>
            <url>jdbc:mysql://localhost:3306/celine</url>
            <user>root</user>
            <password>root</password>
      </rdb>
</database-source>
...
```

Figura 49. Exemplo de configuração para o mecanismo de persistência do tipo BDR

Além dos dois mecanismos citados, existe um terceiro passível de personalização pelo desenvolvedor. Esse tipo de mecanismo será utilizado pelo desenvolvedor quando ele já possui uma BD de sistemas legados, e deseja reutilizar para o ambiente com o CELINE. O desenvolvedor precisa realizar a interface `DAO`, que está no módulo CELINESCORM, e efetuar as configurações como exemplificado na Figura 50. No atributo `class` da *tag* `<bean>` se especifica o nome da classe que realiza a interface `DAO`. A *tag* `<bean-attribute>` é opcional, e será utilizada pelo desenvolvedor se ele desejar atributos configuráveis em sua classe. O atributo da classe é especificado pelo atributo `name` da *tag*, e o valor do atributo é definido no conteúdo da *tag*. No exemplo, são inicializados os atributos `adminUser` e `adminPassw` com os valores “adilson” e “mypassw” respectivamente.

```
...
<database-source>
      <bean class="br.univali.example.dao.LegacyDAO">
            <bean-attribute name="adminUser">adilson</bean-attribute>
            <bean-attribute name="adminPassw">mypassw</bean-attribute>
      </bean>
</database-source>
...
```

Figura 50. Exemplo de configuração para o mecanismo de persistência personalizável

A Tabela 7 apresenta todos os arquivos JAR necessários para utilização do componente, além dos arquivos CELINESCORM.JAR e CELINELMS.JAR, que são a própria implementação do componente. Foi omitido o arquivo de driver com o BD, que depende da própria aplicação.

Tabela 7. Dependências de bibliotecas.

Arquivos JAR	**Utilização no componente**
`commons-beanutils`	Leitura do arquivo de manifesto e `celine-config.xml`.
`commons-digester`	Leitura do arquivo de manifesto e `celine-config.xml`.
`commons-logging`	Leitura do arquivo de manifesto e `celine-config.xml`.
`commons-fileupload`	Importação dos pacotes SCORM.
`commons-io`	Importação dos pacotes SCORM.
`cos`	Importação dos pacotes SCORM.
`Dwr`	Comunicação entre SCO e LMS.
`xpp3_min`	XMLDAO.
`xstream`	XMLDAO.

4 AMBIENTES DESENVOLVIDOS

Esse capítulo apresenta as experiências na construção de ambientes de aprendizagem com o CELINE. Durante o desenvolvimento desta dissertação, foram desenvolvidos quatro ambientes, e dois deles utilizados em sala de aula, em substituição às aulas expositivas: os alunos estudaram baseando-se no material elaborado no modelo SCORM. Ao final do capítulo é apresentada a bateria de testes de conformidade com o SCORM, comprovando que o CELINE está totalmente de acordo com a especificação.

4.1 BRUCE

Com o propósito de fornecer uma aplicação de exemplo e documentação para os desenvolvedores que desejarem utilizar o CELINE, foi implementado o BRUCE (Basic Reference of Using the CEline component). Essa seção apresenta passo-a-passo o desenvolvimento de uma aplicação utilizando o CELINE. O resultado dessa aplicação é o próprio BRUCE. O código de algumas páginas serão omitidas, devido a sua simplicidade, e irrelevância para a utilização do componente.

Após criar a estrutura de uma aplicação web, deve-se copiar os arquivos listados na Tabela 7, mais os arquivos CELINESCORM.JAR e CELINELMS.JAR para a pasta `WEB-INF\lib`. Todos os arquivos citados para cópia nessa seção estão disponíveis conforme citado no Apêndice A.

O arquivo `dwr.xml` deve ser copiado para a pasta `WEB-INF`. Na mesma pasta, o arquivo `web.xml` deve conter as linhas já apresentadas em várias figuras na Seção 3.3.2 . Para facilitar, a Figura 51 agrupa todos os mapeamentos necessários nesse arquivo. Pode-se observar a existência do *servlet* de nome `bruce`. Esse *servlet* foi criado para ilustrar o uso de páginas JSP em área protegida na aplicação, ou seja, manter as páginas dentro da pasta `WEB-INF`. Todas as requisições com a URL terminando com a extensão “`do`” são tratadas por esse *servlet*. O código fonte do *servlet* está listado na Figura 52. Através da classe `UserAdministration`, que faz parte do componente, é verificado se existe um usuário autenticado na sessão corrente. Se não houver um usuário autenticado, a aplicação é encaminhada para a página `dontlogin.jsp`. Em caso positivo, a URL é reutilizada substituindo a extensão “`.do`” por “`.jsp`”. Essas páginas estão localizadas na pasta `WEB-INF\views`. Com isso, o usuário não consegue acessá-las diretamente no navegador, ou seja,

somente a aplicação fornece essas páginas através do repasse da requisição, como ilustrado no código fonte.

```
...
      <listener>
            <listener-class>
                  br.univali.celine.lms.core.LMSContextListener
            </listener-class>
      </listener>

      <servlet>
            <servlet-name>lms</servlet-name>
            <servlet-class>br.univali.celine.lms.core.LMS</servlet-class>
      </servlet>

      <servlet>
            <servlet-name>bruce</servlet-name>
            <servlet-class>br.univali.bruce.core.BruceWeb</servlet-class>
      </servlet>

      <servlet>
            <servlet-name>dwr-invoker</servlet-name>
            <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
      </servlet>

      <servlet-mapping>
            <servlet-name>dwr-invoker</servlet-name>
            <url-pattern>/dwr/*</url-pattern>
      </servlet-mapping>

      <servlet-mapping>
            <servlet-name>lms</servlet-name>
            <url-pattern>/lms</url-pattern>
      </servlet-mapping>

      <servlet-mapping>
            <servlet-name>bruce</servlet-name>
            <url-pattern>*.do</url-pattern>
      </servlet-mapping>
...
```

Figura 51. Configuração do arquivo `web.xml`

```
public class BruceWeb extends HttpServlet {

   protected void service(HttpServletRequest request, HttpServletResponse
            response) throws ServletException, IOException {

      String nextURL = "";
      User user = UserAdministration.getUser(request);
      if (user == null) {
          nextURL = "dontlogin.jsp";
      } else {
          nextURL = request.getServletPath().replaceFirst(".do", ".jsp");
      }

      RequestDispatcher rd = request.getRequestDispatcher("WEB-INF/views/" +
                                                          nextURL);
      rd.forward(request, response);

   }
}
```

Figura 52. Listagem da classe `BruceWeb`

A Figura 53 representa a configuração do componente através do arquivo `celine-config.xml` utilizando o mecanismo XML de persistência. Esse arquivo deve ser criado na pasta `WEB-INF`. O arquivo `celine.xml`, referenciado nessa listagem, deve ser iniciado com pelo menos um usuário administrador, conforme a Figura 54.

```
<config>
      <courses-folder>courses</courses-folder>

      <error-page>error.do</error-page>

      <database-source>
            <xml>WEB-INF\celine.xml</xml>
      </database-source>
</config>
```

Figura 53. Listagem do arquivo de configuração `celine-config.xml`

```
<celine>
  <users>
    <user>
      <name>adilson</name>
      <passw>123</passw>
      <admin>true</admin>
    </user>
  </users>
  <courses>
  </courses>
  <registeredCourses>
  </registeredCourses>
</celine>
```

Figura 54. Listagem do arquivo de dados `celine.xml`

A primeira página da aplicação é aquela onde o usuário digita os seus dados para autenticação. A Figura 55 apresenta o código fonte dessa página, usando a *tag* customizada apropriada para isso. Uma vez autenticado, a página a ser apresenta é `welcome.jsp`, que precisa estar na pasta `WEB-INF\views`. Essa conversão de `welcome.do` para `welcome.jsp` é feita pelo *servlet*, conforme já demonstrado.

```
<%@taglib prefix=”celine” uri="http://www.univali.br/celine/tags" %>
<html>
   <head>
      <title>CELINE - Login</title>
   </head>

   <body>

      <celine:loginForm nameFieldName="User :" nameFieldPassw="Password :"
                                                 nextURL="welcome.do"/>

   </body>
</html>
```

Figura 55. Listagem da página `main.jsp`

A página `welcome.jsp` apresenta um menu para que o usuário possa (i) acessar os cursos registrados ou (ii) registrar-se ou cancelar registros nos cursos. Caso ele for um administrador, ainda aparecem as opções para (i) gerenciamento de cursos ou (ii) usuários. A Figura 56 lista o código fonte dessa página. A *tag* `<user/>` armazena os dados do usuário corrente na página, que é acessada em seguida pela *tag* condicional `<if>` verificando se o usuário é administrador.

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

You're welcome to BRUCE.<br/><br/>
Select below the action what do you do want now.<br/><br/>
<celine:user/>

<div>
<a href='listcourses.do'>Your registered courses</a><br/>
<a href='regcourses.do'>Register to a new course</a><br/>
</div>

<c:if test="${useradmin}">
   <a href='managecourses.do'>Manage courses</a><br/>
   <a href='manageusers.do'>Manage users</a><br/>
</c:if>
```

Figura 56. Listagem da página `welcome.jsp`

A página `manageusers.jsp` lista os usuários cadastrados permitindo adicionar, alterar ou excluir usuários. O seu código é listado na Figura 57. A *tag* `<listUsers>` é um iterador, ou seja, para cada usuário cadastrado ele repetirá o código no seu corpo. No exemplo da figura, são criadas linhas de uma tabela. Além disso, também existem outras duas *tags* – `<editUser>` e `<deleteUser>` – que produzem links. Na *tag* `<deleteUser>` está configurado para retornar para a própria página após a execução da ação. A página `edituser.jsp` é o destino para adicionar e alterar usuários. A Figura 58 lista o código dessa página, onde pode-se observar a *tag* `<saveUserForm/>`, com suas configurações dos nomes dos rótulos para os campos e botões. Após a execução dessa ação, a requisição é repassada para a listagem de usuários.

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
    <h1>Managing Users</h1>
    <table border="1" width="500px">
       <tr>
          <td align="center" width="100%" colspan="3">
             <b>Users</b>
          </td>
       </tr>
       <celine:listUsers varName="username" varIndex="index" >
          <tr>
             <td width="350px">
                <a href="viewuserinfo.do?userName=${username}">${username}</a>
             </td>

             <td align="center" width="75px">
               <celine:editUser name="${username}" nextURL="edituser.do"
                                                          varLink="linkEdit">
                        <a href="${linkEdit}">Edit</a>
               </celine:editUser>
             </td>
             <td align="center" width="75px">
               <celine:deleteUser name="${username}" nextURL="manageusers.do"
                                                       varLink="linkDelete">
                  <a href="${linkDelete}">Delete</a>
               </celine:deleteUser>
             </td>
          </tr>
       </celine:listUsers>
    </table>
    <a href="edituser.do">New User</a>
</body>
</html>
```

Figura 57. Listagem da página `manageusers.jsp`

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<html>
<body>
      <celine:saveUserForm nextURL="manageusers.do"
            nameFieldName="User name"
            nameFieldPassw="Passw"
            nameFieldAdmin="Is admin"
            label="Save"/><br/>
      <a href="manageusers.do">Back</a>
</body>
</html>
```

Figura 58. Listagem da página `edituser.jsp`

A Figura 59 mostra o código fonte que lista todos os cursos cadastrados. A *tag* `<listCourses>` também é um iterador, e cria uma linha da tabela para cada curso cadastrado. A partir da *tag* `<deleteCourse/>` se cria um link, e após a execução da ação, é encaminhada de

volta para essa página. A Figura 60 ilustra a *tag* que destaca os dados de um curso. Na página da Figura 61 é apresentada a *tag* que gera um formulário para importação de pacotes SCORM.

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
   <h1>Managing Courses</h1><br/>
   <table border="1" width="500px;">
      <tr>
         <td align="center" width="100%" colspan="2"><b>Courses</b></td>
      </tr>

      <celine:listCourses varLink="courseLink" varName="courseName"
                                                varId="courseId">
         <tr>
            <td width="250px">
               <a href="viewcourseinfo.do?courseId=${courseId}">${courseName}</a>
            </td>
            <td align="center">
               <celine:deleteCourse courseId="${id}" nextURL="managecourses.do"
                                                     varLink="linkDelete">
                  <a href="${linkDelete}">Delete</a>
               </celine:deleteCourse>
            </td>
         </tr>
      </celine:listCourses>
   </table>
   <a href="importcourses.do">Import Course</a><br/><br/>
</body>
</html>
```

Figura 59. Listagem da página `managecourses.jsp`

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<html>
<body>

   <h1><b>Course Info</b></h1><br/>
   <celine:courseInfo courseId="${param.courseId}" varFolderName="folder"
                                                   varTitle="title">
       Folder : ${folder} <br/>
       Title : ${title}
   </celine:courseInfo>

   <a href="managecourses.do">Back</a>

</body>
</html>
```

Figura 60. Listagem da página `viewcourseinfo.jsp`

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<html>
<body>
      <h1>Import Course</h1>

      <celine:formImportCourse nextURL="welcome.do"/><br/>

      <a href="managecourses.do">Back</a>

</body>
</html>
```

Figura 61. Listagem da página `importcourses.jsp`

Até aqui foram apresentadas as páginas de administração. As próximas páginas são acessíveis para os usuários comuns. Antes que o usuário possa abrir algum curso, é necessário registrar-se a ele. O processo de registro em cursos é listado na Figura 62. Ela mostra outra aplicação da *tag* `<listCourses>`. É possível verificar se o usuário corrente está registrado para cada curso da lista. Na página é feito um teste: se está registrado no curso, então mostra um link para cancelar o registro; senão adiciona um link para registrar-se no curso. Esses links são criados com as *tags* `<registerCourse>` e `<unregisterCourse>` .

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<table>
   <celine:listCourses
      varLink="courselink"
      varName="coursename"
      varStatusCourseRegistered="courseregistered"
      varId="courseid">
      <tr>
         <td><a href='seetoc.do?id=${courseid}'>${coursename}</a></td>
         <td>
            <c:if test="${courseregistered}">
              <celine:unregisterCourse courseId="${courseid}"
                              nextURL="regcourses.do" varLink="unregister">
                <a href="${unregister}">Unregister</a>
              </celine:unregisterCourse>
            </c:if>

            <c:if test="${courseregistered == false}">
              <celine:registerCourse courseId="${courseid}"
                                  nextURL="regcourses.do" varLink="register">
                <a href='${register}'>Register</a>
              </celine:registerCourse>
            </c:if>
         </td>
      </tr>
   </celine:listCourses>
</table>
```

Figura 62. Listagem da página `regcourses.jsp`

Na Figura 63 está o código da página que lista os cursos que estão registrados para o usuário corrente. Para cada curso, é criado um link que permite abrir o curso.

```
<%@taglib prefix="celine" uri="http://www.univali.br/celine/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<h1>Your registered courses</h1>

<table>
<celine:listCourses varLink="courselink"
                    varName="coursename"
                    varStatusCourseRegistered="courseregistered">

      <c:if test="${courseregistered}">
            <tr>
                  <td><a href='${courselink}'>${coursename}</a></td>
            </tr>
      </c:if>

</celine:listCourses>
</table>
```

Figura 63. Listagem da página `listcourses.jsp`

A partir do que foi exposto nessa seção, é possível desenvolver as aplicações web utilizando os recursos do componente para o gerenciamento e uso de pacotes SCORM. A diferença entre utilizar o mecanismo XML e banco de dados, está na mudança da configuração do arquivo na Figura 53 para usar o exemplificando em Figura 49. As próximas seções apresentam aspectos que diferenciam uma aplicação da outra, e como o componente consegue se adaptar a essas especificidades.

4.2 SAGAA

Para experimentar o CELINE com os recursos de adaptação, o pesquisador desta dissertação orientou um TCC que desenvolveu um STI utilizando sistemas *fuzzy* como mecanismo de inferência. Esse sistema foi batizado como SAGAA (SCHAPPO, 2008).

O STI utiliza uma técnica conhecida como **modelo de sobreposição** (MAGOULAS, PAPANIKOLAOU e GRIGORIADOU, 2001; KAVCIC, 2004), que consiste na sobreposição do modelo de domínio[2] com o modelo de aluno[3]. Esse modelo utiliza uma estrutura de grafo direcionado, onde cada vértice corresponde a um conceito a ser aprendido, e as arestas são as relações entre os conceitos. Definem-se quais os vértices iniciais do grafo, que correspondem aos assuntos que o aluno primeiro precisa aprender. Nas arestas são atribuídas condições para ativar o próximo vértice. Essas condições se referem ao grau de conhecimento que o aluno precisa ter no vértice de origem, que podem ser: desconhece, conhece pouco, conhece, conhece bem, aprendeu ou aprendeu bem. A Figura 64 ilustra o modelo de domínio utilizado no trabalho de Schappo (2008). Os nomes dos conceitos foram omitidos da figura e são irrelevantes para citar nesta monografia.

[2] Segundo Pozzebon e Barreto (2002), o modelo de domínio é fundamentalmente uma base de conhecimento, contendo informações de um determinado domínio, que é organizado para representar o conhecimento de um especialista ou professor.
[3] Conforme Pozzebon e Barreto (2002), o modelo do aluno contempla o conhecimento e o comportamento do aluno que possa trazer algum tipo de conseqüência na aprendizagem.

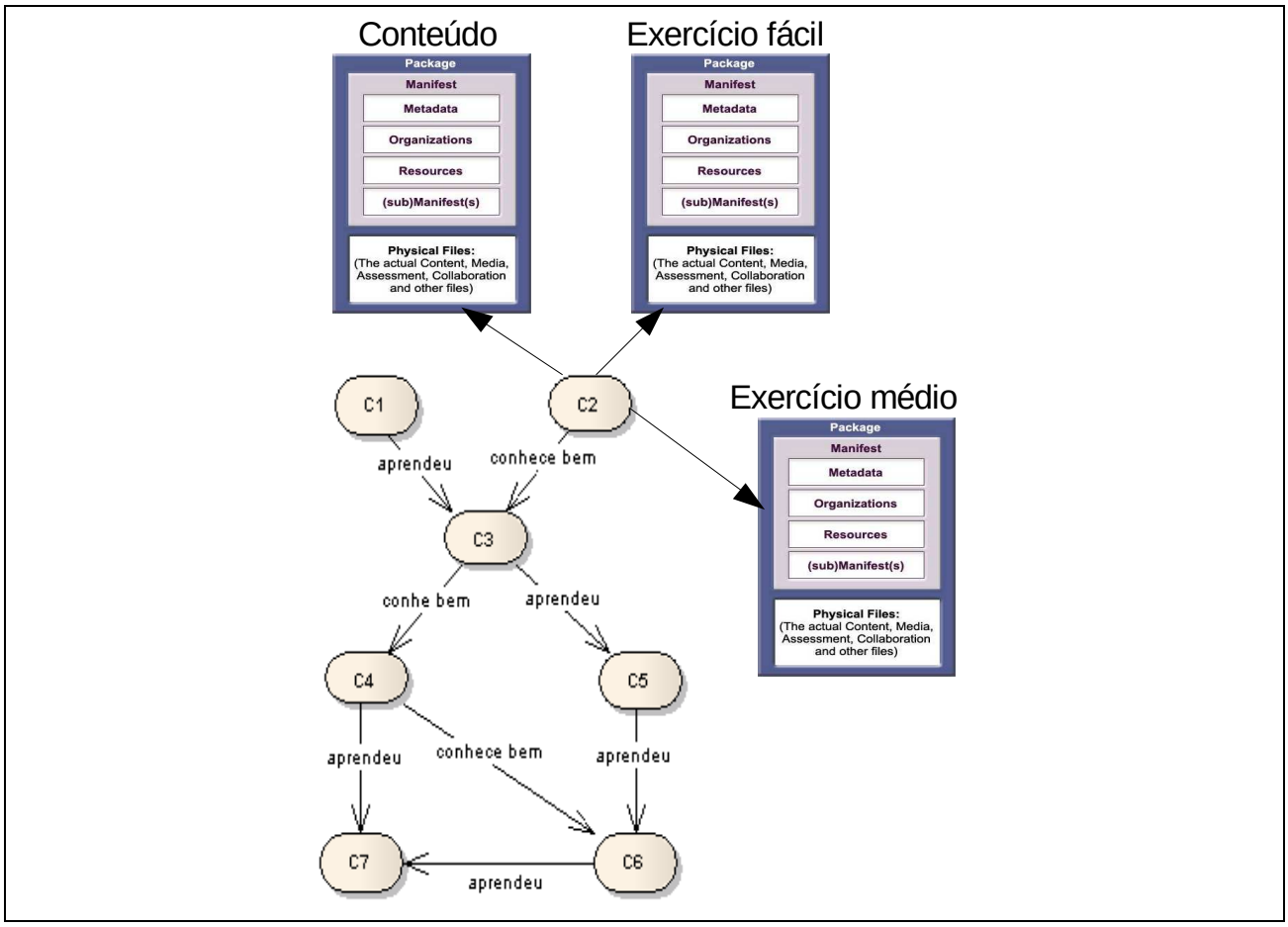


Figura 64. Modelo de domínio do SAGAA

Para o modelo de aluno – que é uma cópia do domínio – cada vértice representa o grau de conhecimento que ele tem do assunto. No primeiro instante, todos os vértices são inicializados como o aluno desconhecendo o assunto.

O SAGAA permite ao professor importar pacotes SCORM para cada assunto. O pacote SCORM pode ser classificado entre conteúdo ou exercício. Ainda, para o exercício, o professor pode classificá-lo como fácil, médio ou difícil. Na Figura 64 está exemplificado o relacionamento entre o conceito C2 com três pacotes: um classificado como conteúdo e os demais como exercício. Essas classificações são utilizadas nas regras de inferência do STI. Por exemplo, se o aluno desconhece o assunto, então os primeiros exercícios são de nível fácil, e à medida que ele evolui no grau de conhecimento, os exercícios vão sendo dificultados.

Em relação ao CELINE, foi desenvolvida uma classe que realiza a interface `LMSIntegration`. Para o SAGAA, um curso corresponde a um vértice do modelo de domínio.

Quando o aluno entra no sistema, os cursos listados para ele são os vértices habilitados. Essa decisão influencia na implementação do método `selectCourses()` da classe de integração. Esse método devolve os cursos (ou seja os assuntos) que ele precisa aprender.

Quando o usuário abre um curso, no método `openContentPackage()` são agrupados todos os pacotes de conteúdo do assunto selecionado. Também é escolhido um pacote de exercício, ainda não praticado pelo aluno, com nível de dificuldade de acordo com as regras de inferência. No sistema não foi implementada a adaptação dinâmica, ou seja, aquela em que altera a estrutura da árvore enquanto o aluno interage com o curso. Por essa razão, o método `changeCourse()` não foi modificado.

Nesse trabalho também foi desenvolvida uma *applet* contendo uma ferramenta para realização dos exercícios. Essa *applet* foi incorporada em pacotes de conteúdo, e ela fazia a comunicação com o LMS utilizando-se do *Data Model* `cmi.interactions`, com o propósito de enviar os resultados do exercício ao STI. A *applet* também solicita dados ao LMS, em caso de continuação de um exercício suspenso. A Figura 65 ilustra um exercício com essa *applet* aberto no SAGAA.

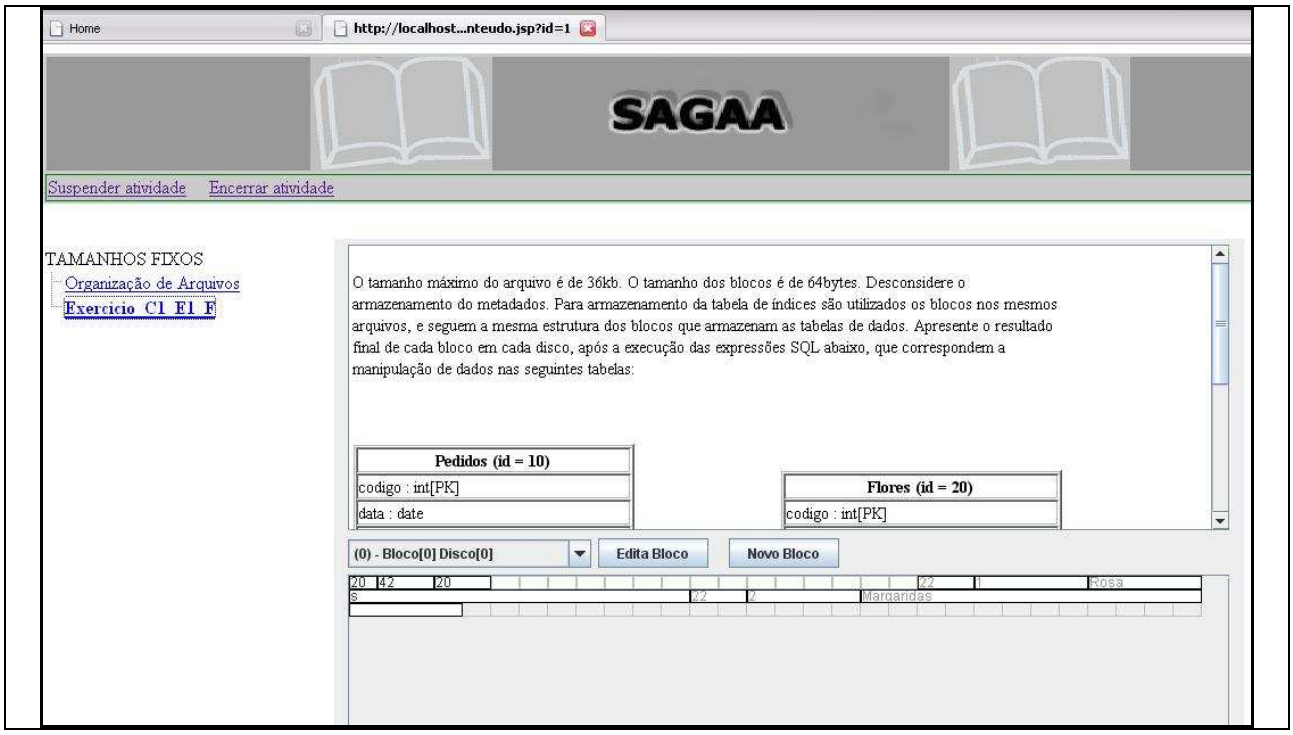


Figura 65. Página do SAGAA aberto com um exercício

O sistema utilizou banco de dados PostgreSQL e tecnologias JSF. Desse experimento concluiu-se que as *tags* customizadas do CELINE não eram compatíveis com as páginas JSF[4]. Por essa razão, foi criada uma classe provedora de dados (segundo o JSF, uma classe do tipo *form bean*), com objetivo de suprir a *tag* `<listCourses/>`. O código dessa classe está listado na Figura 66. Pode-se observar que a URL para disparar a ação de abrir o curso é montada nessa classe: `lms?action=opencourse`. A página na Figura 67 utiliza esse objeto como fonte de dados para criar as linhas de uma tabela.

[4] Segundo Jacobi e Fallows (2007), JSF (*Java Server Faces*) é um *framework* de componentes de interface com o usuário, que segue a arquitetura MVC (*Model View Controller*). Por essa razão, a implementação com JSF exige a separação bem definida entre as classes de apresentação e regras de negócios.

```
public class ListCoursesBean {

   public List<CourseBean> getItens() throws Exception {
      LMSConfig config = LMSConfig.getInstance();
      List<Course> lista = config.getDAO().listCourses();
      HttpServletRequest request = (HttpServletRequest)
           FacesContext.getCurrentInstance().getExternalContext().getRequest();

      UserImpl user = (UserImpl) request.getSession().getAttribute("user");
      LMSIntegration lmsInt = config.getLMSIntegration();
      try {
            lmsInt.selectCourses(user, lista);
      } catch (Exception e) {
            throw new JspException(e);
      }
      List<CourseBean> l =  new ArrayList<CourseBean>();
      for (Course c:lista) {

            boolean userRegistered = false;
            boolean hasRights = false;
            try {
                  userRegistered =
                     config.getDAO().userIsRegisteredAtCourse(user, c.getId());
                  hasRights = config.getDAO().userHasRightsAtCourse(
                                                            user, c.getId());
            } catch (Exception e) {
                  e.printStackTrace();
            }
            l.add(new CourseBean(c.getTitle(),
                              "lms?action=opencourse&id=" + c.getId(),
                              c.getId(), c, userRegistered, hasRights));
      }
      return l;
   }
}
```

Figura 66. Listagem do provedor de dados `ListCoursesBean`

```
...
<h:dataTable value='#{ListCoursesBean.itens}' var='item' border="0"
      cellpadding="2" cellspacing="0" width="100%">
   <h:column>
      <h:outputLink value="conteudo.jsp?id=#{item.courseId}" target="_blank">
                  <h:outputText value="#{item.title}"/>
      </h:outputLink>
   </h:column>
</h:dataTable>
...
```

Figura 67. Listagem da página que lista os cursos

4.3 ALICE 2.0

Os alunos das disciplinas de Algoritmos I e II do curso de Bacharelado em Ciências da Computação da UNIVALI vem utilizando nos últimos sete semestres o ALICE (*Algorithm Learning Internet Based Computer Environment*), que é um STI que incorpora o professor no processo de aprendizagem dos alunos, diferente do que acontece nos modelos tradicionais de STI. O professor passa a atuar como parceiro do sistema, fornecendo informações sobre os alunos e recebendo em troca análises do desempenho deles nas seções de uso da ferramenta. Esta característica em especial tem promovido melhorias significativas no desempenho dos alunos, índices de desistência e repetência na disciplina (RAABE, 2005).

O sistema foi desenvolvido em PHP, e o objetivo é utilizar o sistema em outras disciplinas. A elaboração do conteúdo é feita diretamente no sistema, através de cadastros, ou via código fonte. Esses dois aspectos dificultam o emprego do ALICE. Por essa razão, decidiu-se desenvolver uma nova versão, para que possa utilizar pacotes SCORM.

A versão 2.0 do ALICE foi especificada pelo pesquisador desta dissertação, junto do seu orientador, e implementado pelo mesmo bolsista envolvido na codificação do módulo CELINELMS. Por essa razão, o sistema possui bem aplicados os recursos documentados no BRUCE.

Como a versão atual do ALICE foi utilizada em vários semestres, desejava-se reaproveitar partes consolidadas do sistema, como o cadastro de exercícios. Existiam duas alternativas: produzi-los novamente dentro de pacotes SCORM, ou fazer com que o ambiente crie dinamicamente a estrutura das atividades com esses exercícios. Decidiu-se pela segunda opção, visto que o trabalho a demandar seria muito inferior.

Para atender essa necessidade, implementou-se uma classe de integração (realizando a interface `LMSIntegration`). Quando o usuário abre o curso, o sistema cria uma atividade chamada “Exercícios” no final da árvore de atividades. Acessando o banco de dados, são selecionadas cinco questões, e assim criadas novas atividades agrupadas dentro de “Exercícios”. Ao acessar uma dessas atividades, o sistema monta dinamicamente a página, seguindo a especificação SCORM quanto às obrigações de um SCO.

Existem três tipos de questões: **digitação de algoritmo**, **verdadeiro/falso** ou **alternativas**. Cada questão é composta de no máximo uma página, determinada por um *template*. Nesse *template* existem partes que são trocadas com informações da questão na montagem da página. A Figura 68 apresenta o *template* para o tipo de questão **digitação de algoritmo**. Os campos a serem trocados encontram-se contornados com “[“ e “]”.

```
<html>
  <head><meta http-equiv="Content-Type"
                                  content="text/html; charset=ISO-8859-1">
  <title>Questão de Algoritmo</title></head>

  <script>

      [scripts necessarios]

      function submitAnswer() {

        var resp = document.getElementById("resposta").value;

        if (resp.length > 0) {

            if (confirm("Deseja realmente enviar sua resposta?\nVocê não poderá
                                           alterá-la posteriormente!!")) {

                  AddInteraction("idQuestao", "numeric", [idQuestao]);
                  AddInteraction("idAluno", "numeric", [idAluno]);
                  AddInteraction("resposta", "long-fill-in", resp);

                  document.getElementById("sendButton").disabled = true;
                  document.getElementById("resposta").disabled = true;

                  alert("A sua resposta foi enviada com sucesso!!");

            }

        } else alert("Você deve digitar uma resposta!!");

      }

  </script>

  <!-- Chama as funções LoadPage() e UnloadPage(), responsáveis pela
                                                    comunicação com o LMS -->

  <body onload="LoadPage();" onunload="return UnloadPage();">

      <textarea readonly="readonly" style="border: 0; overflow: hidden;"
                        rows="[enunciado.linhas]" cols="[enunciado.colunas]"
                  name="enunciado">[enunciado]</textarea>
      <br/><br/>
      <textarea id="resposta" cols="60" rows="25"/></textarea><br/><br/>
      <input type="button" value="Responder" id="sendButton"
                                  onclick="javascript:submitAnswer();"/>

  </body>

</html>
```

Figura 68. Listagem do *template* para criação de questões de digitação de algoritmos

Foi realizado um experimento com 34 alunos da disciplina de Algoritmos I do primeiro semestre do curso de Ciências da Computação da UNIVALI, ministrada pelo professor orientador desta dissertação. O curso desenvolvido era sobre o assunto “Linguagem C”. Ele foi confeccionado

por um bolsista usando a ferramenta MOS Solo. O ambiente é acessível somente dentro da instituição. O experimento foi realizado em três aulas: nos dias 7, 12 e 19 de novembro. Os alunos foram cadastrados pelo bolsista, onde o usuário e senha eram os códigos de matrículas dos alunos. A Figura 69 e Figura 70 apresentam as páginas disponíveis para os alunos: a primeira é a página de autenticação e a segunda apresenta o curso aberto.

Figura 69. Página de entrada no sistema

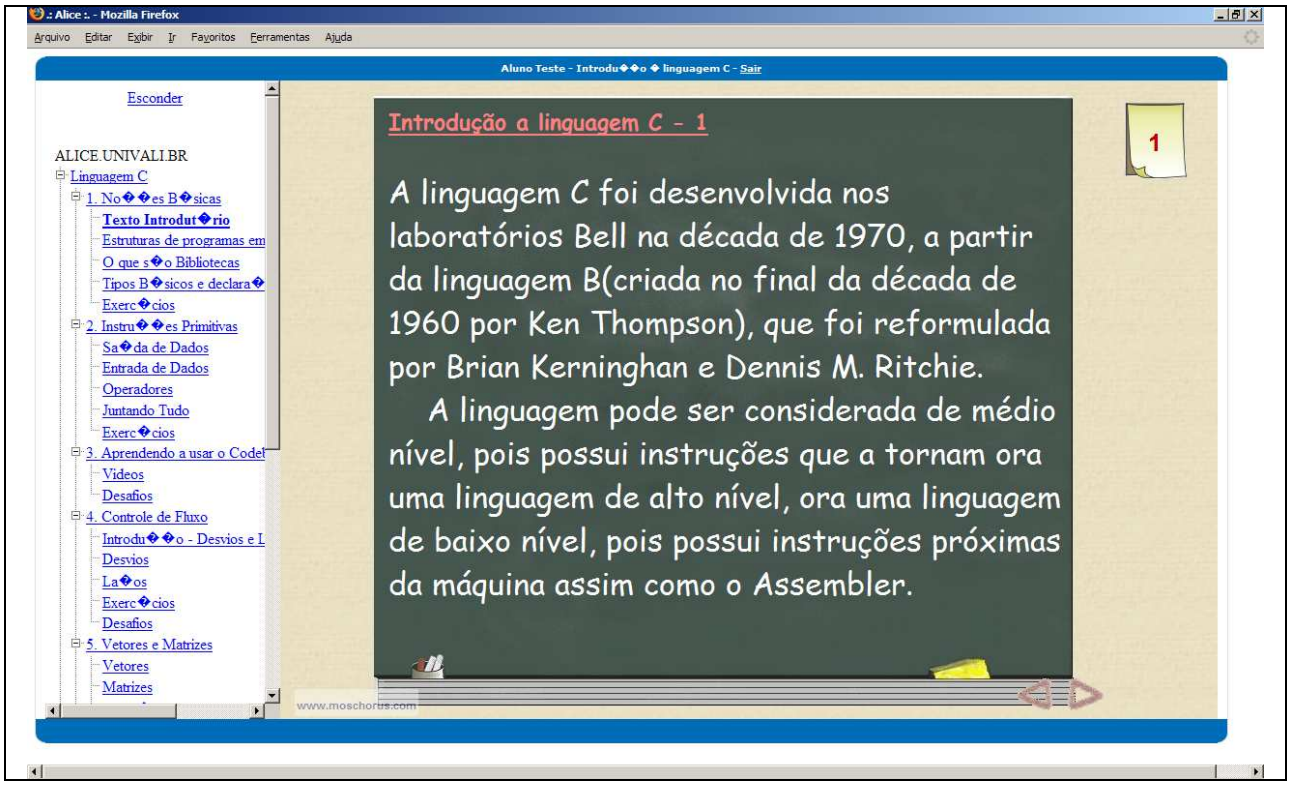


Figura 70. Página com o curso aberto

Conforme relato do professor, os alunos estavam motivados em ler o conteúdo e executar as atividades propostas no material. O assunto não foi exposto pelo professor, ou seja, a fonte de aprendizado desses alunos foi exclusivamente o conteúdo. Os resultados em termos de notas ainda não puderam ser publicados na data desta dissertação, visto que as provas não tinham sido realizadas.

O sistema foi implantando em Linux, com Apache Tomcat e banco de dados MySQL.

4.4 ASTRO

Com intuito de experimentar o desenvolvimento de um ambiente com o CELINE, com programadores que não tiveram participação com o projeto do componente, foi organizada uma força tarefa com três bolsistas na instituição onde leciona: dois deles envolvidos com a programação do ambiente e outro com a produção do conteúdo em SCORM.

Os dois bolsistas eram alunos do segundo semestre de Ciências da Computação, e leigos no desenvolvimento com JEE. O pesquisador forneceu-lhes a especificação do sistema – batizada como ASTRO – conforme o Anexo I, material de aprendizado em desenvolvimento com JEE, e o

BRUCE. A dedicação deles eram 12 horas semanais, o que envolvia o aprendizado e a produção. Foram raras as intervenções do pesquisador.

A qualidade do código do ambiente é muito inferior ao ALICE, pelo fato dos bolsistas terem implementado com *scriptlets* diretamente nas páginas JSP, e por conseqüência não utilizaram as *tags* customizadas que o componente oferecia. Os cadastros foram todos implementados por eles, e não passavam pela consistência do CELINE. O ASTRO utilizou, única e exclusivamente, do componente a abertura e execução dos pacotes SCORM.

Enquanto isso, outro bolsista desenvolvia o material instrucional, no assunto “Vetores” em Java, com a ferramenta MOS Solo. O pesquisador desempenhou o papel de projetista instrucional fornecendo ao bolsista o roteiro a ser desenvolvido. O teste estava programado para ser aplicado com alunos da disciplina de Programação de Computadores, do primeiro semestre do curso de Ciências da Computação, lecionada pelo pesquisador.

A preocupação dos outros dois bolsistas era atender a especificação quanto a implementação. Na data da meta (19/11) o pesquisador aplicou estilos, e melhorou a aparência das págins, nas funcionalidades que os alunos precisavam acessar. O sistema foi implantado em um servidor Linux, com Apache Tomcat e banco de dados MySQL.

As Figuras 69 a 74 apresentam a sequência de páginas disponíveis aos alunos. A Figura 71 é a primeira página do sistema. Se o aluno não está cadastrado, ele precisa clicar no link [aqui], que é então direcionado para a página da Figura 72. Nessa nova página ele digita o nome de usuário e senha que pretende adotar para a autenticação. O cadastro não exige confirmação de nenhum outro usuário.

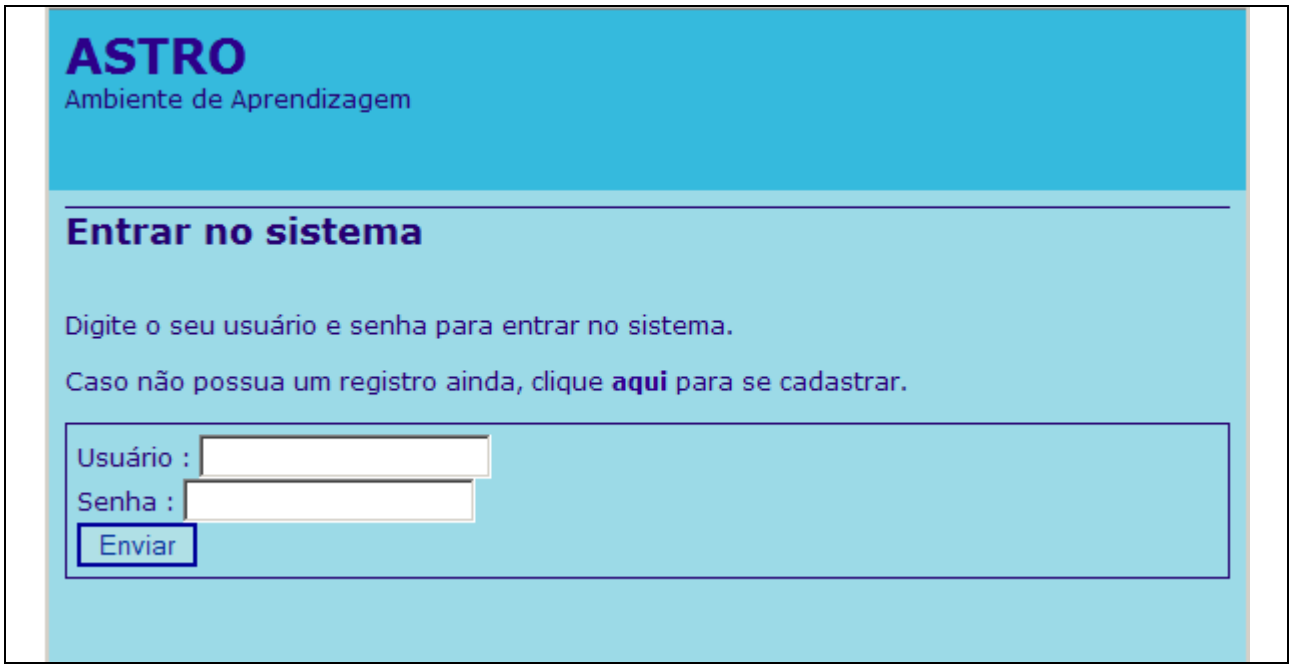


Figura 71. Página de entrada no sistema

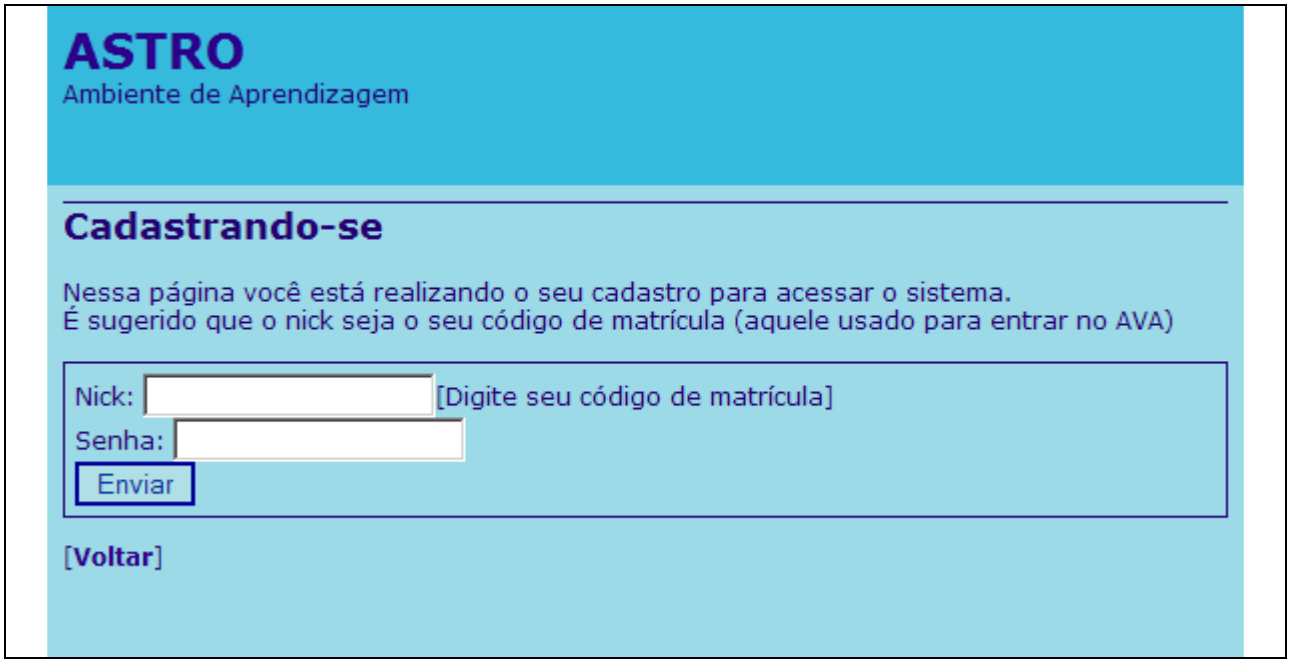


Figura 72. Página de registro no sistema

A Figura 73 apresenta a página que o usuário visualiza após autenticar-se no sistema. Para que possa acessar algum conteúdo, ele precisa se registrar em pelo menos uma turma. No ASTRO os conteúdos são organizados por turma. Ao clicar no link [aqui] o usuário é direcionado para a página apresentada na Figura 74. Nessa página o aluno marca as disciplinas que ele deseja (ou que ele pode) participar. A sua confirmação está condicionada pelo professor da disciplina, ou seja,

enquanto o professor não aceitar o registro, o aluno não consegue acessar os conteúdos da disciplina.

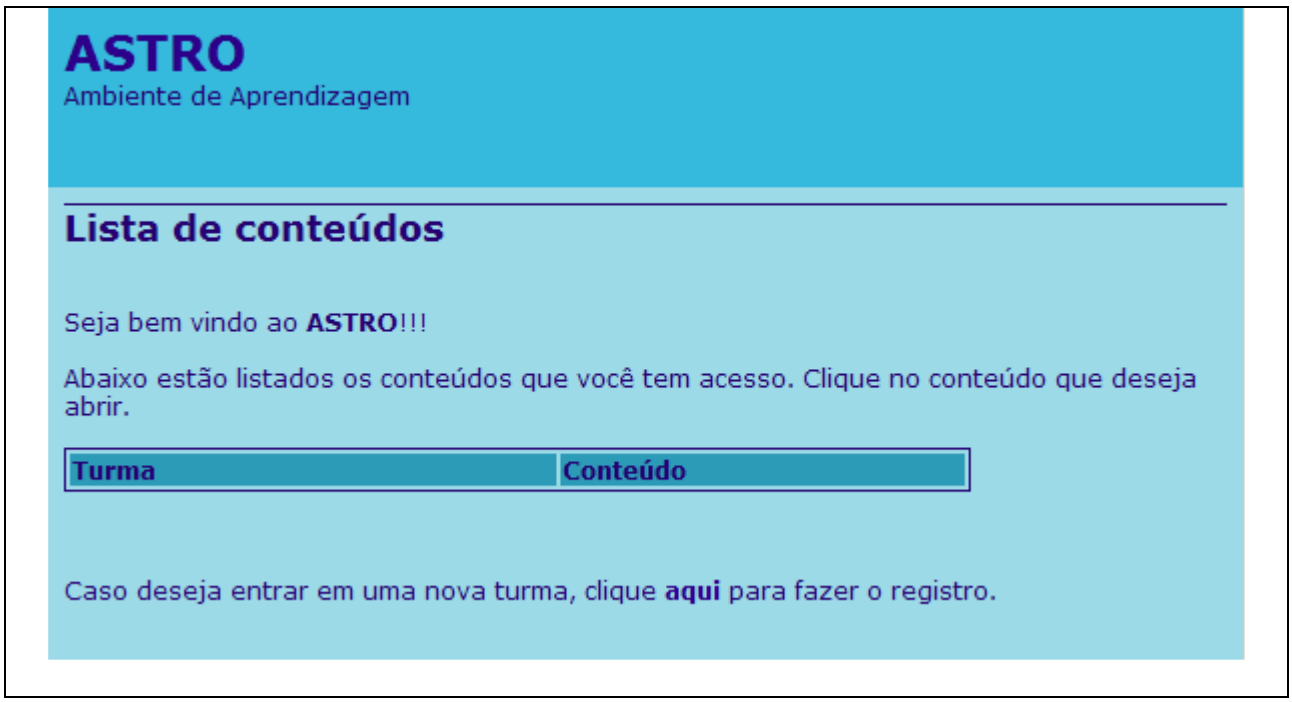


Figura 73. Página com a lista de conteúdo dentro do sistema

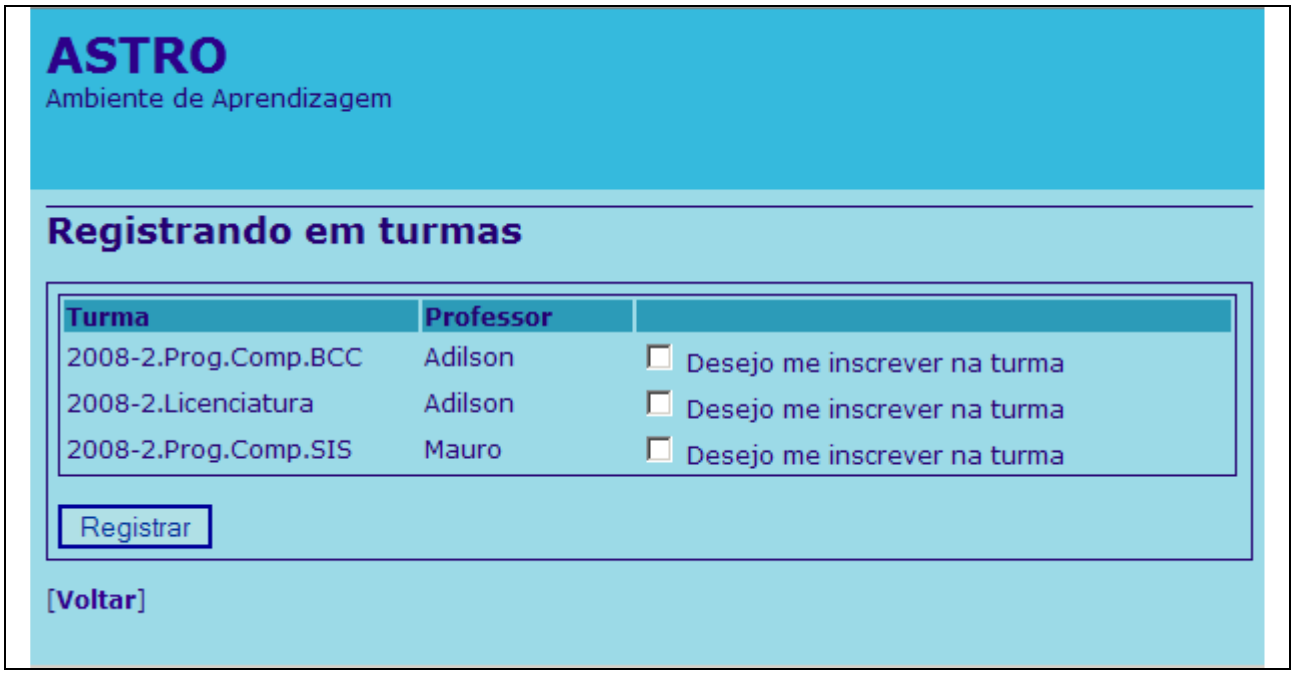


Figura 74. Página para registrar-se nas turmas

Após a confirmação do professor, o sistema lista os conteúdos da disciplina para o aluno. Cada conteúdo corresponde a um pacote SCORM. A Figura 75 apresenta os conteúdos após o registro na disciplina “2008-2.Prog.Comp.BCC” do professor “Adilson”.

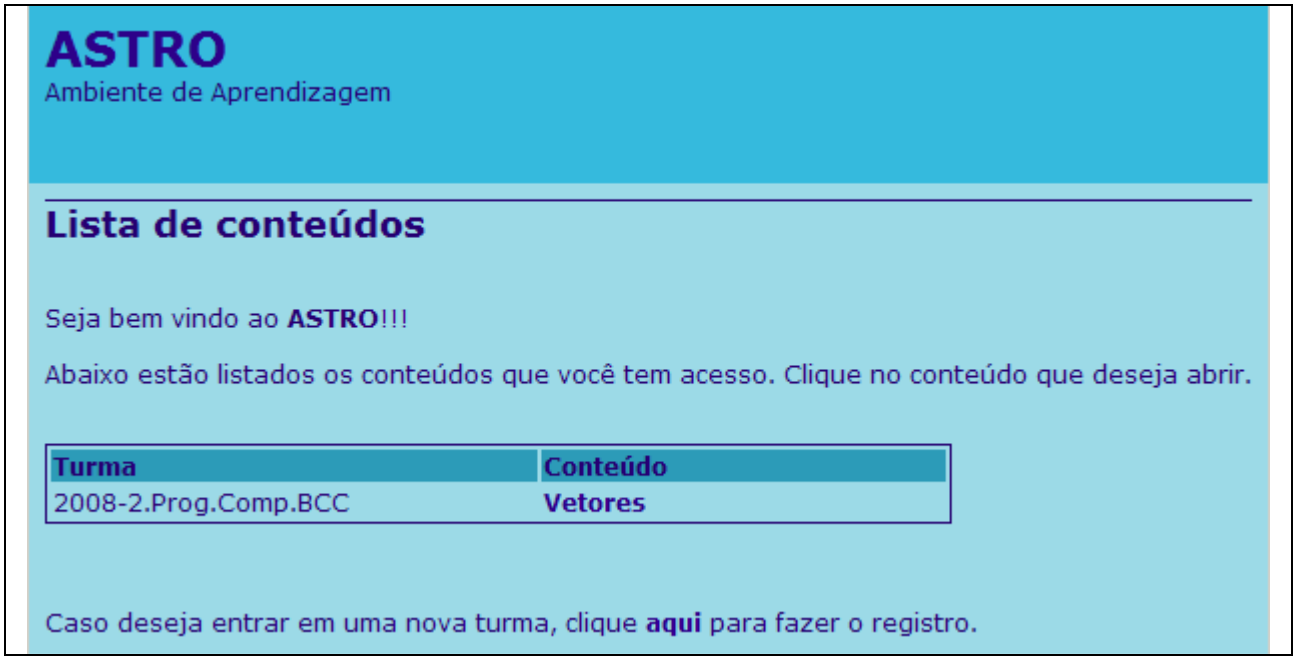


Figura 75. Página com a lista de conteúdo dentro do sistema depois de se registrar

Quando o aluno clica no link do conteúdo, então uma página como da Figura 76 é aberta, permitindo que ele interaja com o conteúdo. Essa figura apresenta a primeira página do curso “Vetores”.

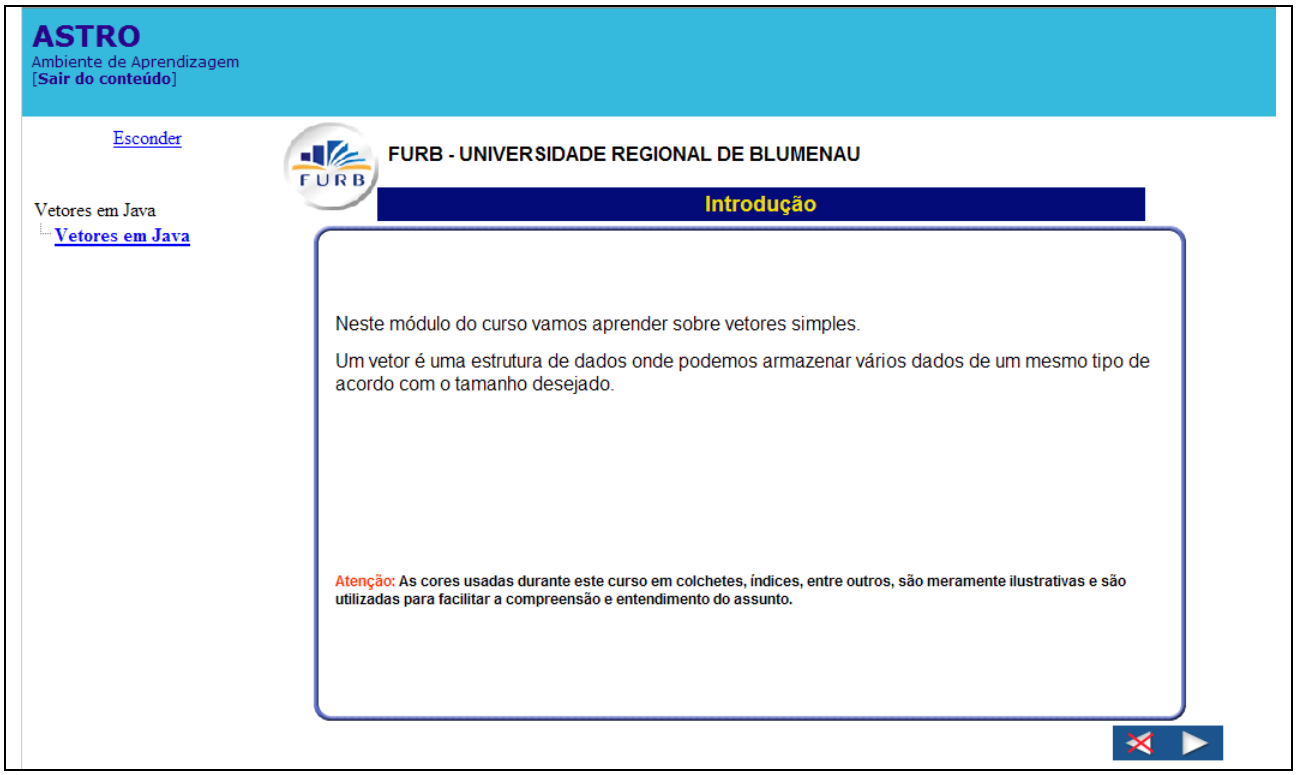


Figura 76. Página com o curso aberto

Esse experimento foi realizado com 31 alunos. O pesquisador forneceu o material do ambiente e o enunciado de um exercício a ser resolvido em um ambiente de programação. O assunto era novo para os alunos, e não foi explicado pelo professor. Eles se sentiram muito a vontade com a navegação e motivados em ler e entender o material. Era uma experiência nova para eles, pois nunca haviam interagido com material eletrônico organizado dessa forma.

Outro professor, da mesma disciplina, porém do curso de Sistemas de Informação, também fez uma aula de experimento com sua turma composta por 15 alunos, e a satisfação expressa por eles foi similar da primeira turma. Apesar dos alunos não terem sido direcionados a acessassem o conteúdo extra-classe, entre as duas turmas, cinco alunos consultaram o material fora do horário de aula.

O ASTRO também foi utilizado na disciplina de Objetos de Aprendizagem Computacional do curso de Licenciatura em Computação, onde o pesquisador também era professor. A turma era composta por três alunos, e a tarefa deles foi a elaboração de material instrucional utilizando as ferramentas CourseLab e MOS Solo. O professor implantou os cursos para os alunos acessarem os

materiais uns dos outros, visando avaliarem e reconhecerem as suas próprias falhas na confecção de material instrucional.

A disciplina tinha como ementa “Padrões de Objetos de Aprendizagem Computacional. Repositórios de Objetos de Aprendizagem. Projeto instrucional. Inserção no cotidiano escolar da Educação Básica.” Durante o semestre os alunos conheceram a especificação SCORM, e duas ferramentas de autoria: MOS Solo e CourseLab. Haviam um trabalho individual a ser produzido, com tema livre, mas tinha como requisito existirem 10 páginas de conteúdo, 10 questões de exercício e mais 10 de avaliação. O trabalho foi organizado em duas versões: na primeira eles produziram sem as técnicas de projeto instrucional; e na segunda o objetivo era a refatoração com as observações dos colegas, do professor e de projeto instrucional. A primeira versão do trabalho foi implantada no ASTRO, onde os alunos puderam visualizar o seu conteúdo disponível na Internet, e ocorreu a avaliação entre pares. A segunda versão foi avaliada exclusivamente pelo professor. Devido a problemas climáticos, as aulas tiveram que ser interrompidas no fim do semestre, o que comprometeu o andamento (não só) da disciplina.

4.5 TESTE DE CONFORMIDADE DE LMS COM SCORM

A ADL disponibiliza em seu site um conjunto de pacotes e ferramentas para execução de testes no intuito de garantir a conformidade entre o especificado no SCORM com o implementado. Esse conjunto, denominado *Advanced Distributed Learning (ADL) Sharable Content Object Reference Model (SCORM®) 2004 3rd Edition Conformance Test Suite*, permite testar LMS, SCOs e pacotes de conteúdo.

Os testes de concordância a nível de LMS são os mais apropriados para verificação do CELINE. Estes testes envolvem experimentar nos três aspectos da especificação CAM, RTE e SN.

No RTE são feitas as verificações para apresentar *assets* e SCOs; se todas as funções da API estão presentes; e se todos os elementos do *Data Model* são implementados com suas consistências.

Quanto ao CAM, são verificados se todos os elementos do arquivo de manifesto são reconhecidos e se inicializam os elementos do *Data Model*.

Para o SN, são averiguados os comportamentos das requisições e regras de seqüenciamento, considerando a atribuição e retorno dos valores nos *Data Models* e a apresentação da árvore de atividades.

Para a execução dos testes, é necessário baixar (i) os pacotes de conteúdo SCORM referentes aos testes (são 103 pacotes), e (ii) a ferramenta CTS (*Conformance Test Suite*) para coordenação dos testes. Os pacotes devem ser importados no LMS a ser testado. O BRUCE foi o ambiente utilizado, porém, qualquer ambiente feito com o CELINE poderia ser usado, pois os testes acontecem assim que o usuário abre o curso, e nesse momento a condução é totalmente feita pelo componente. É importante chamar a atenção, que não devem ser utilizado o recurso de adaptação de conteúdos, pois o conteúdo deve ser refletido conforme o original.

Para iniciar os testes, deve ser executada a CTS. Nele são cadastrados alguns dados iniciais, como a identificação do LMS e de dois usuários cadastrados no LMS. Feito isso, escolhem-se quais testes serão executados. A Figura 77 apresenta esse momento na CTS. No canto inferior esquerdo da página estão listados todos os testes possíveis. Caso sejam escolhidos somente alguns deles, a ferramenta adverte que o selo "*SCORM Conformance*" não será exibido se não forem executados todos os testes.

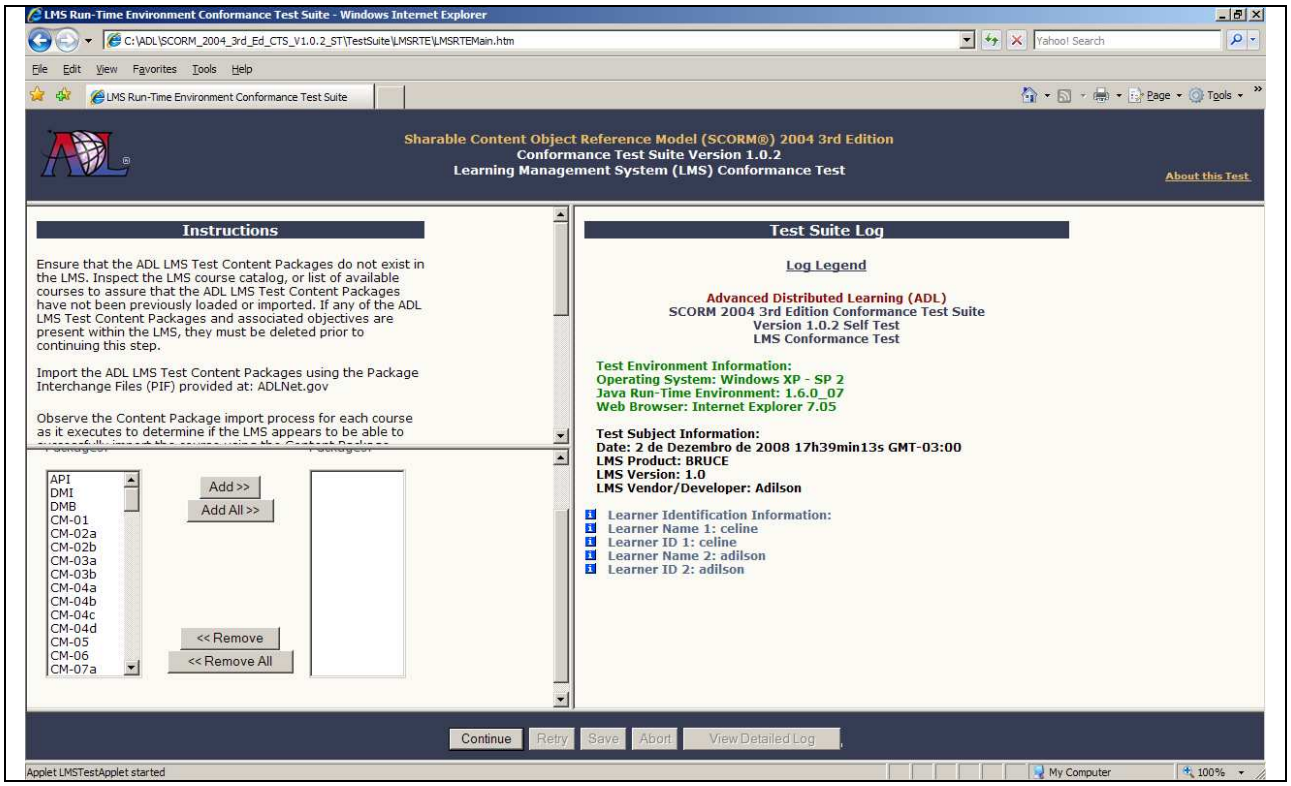


Figura 77. Página de configuração dos testes na CTS

Após selecionar os testes e prosseguir para a próxima página, a CTS fica em estado de espera, aguardando que os pacotes de conteúdo sejam iniciados pelo LMS. Ela sorteia o nome de um dos usuários, e solicita que com esse usuário seja autenticado no LMS.

Quando o pacote é aberto no LMS, então a CTS se comunica com o pacote e coordena os testes, apresentando os registros de todas as atividades realizadas durante o teste, e se houve sucesso ou falha. A Figura 78 apresenta a primeira fase do teste do pacote de nome API, que confere se o LMS consegue apresentar *assets*. A Figura 79 ilustra a aparência da página na CTS mostrando a aprovação nos testes do pacote API. A Figura 80 mostra uma parte dos detalhes nos testes efetuados nesse primeiro pacote.

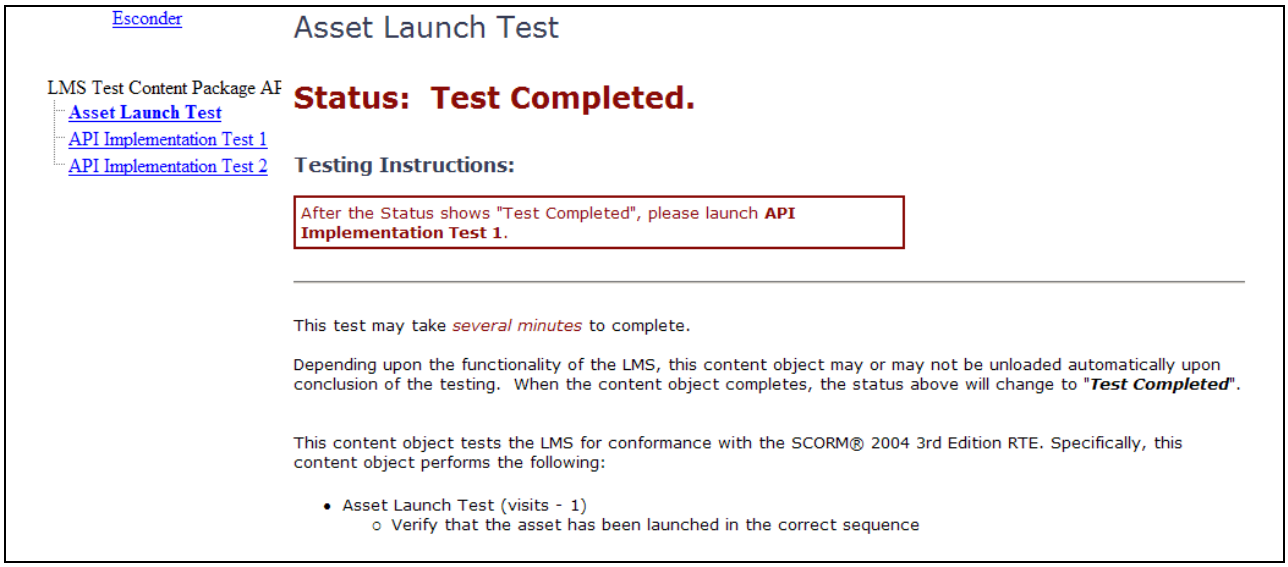


Figura 78. Página com o teste de apresentação de *asset*

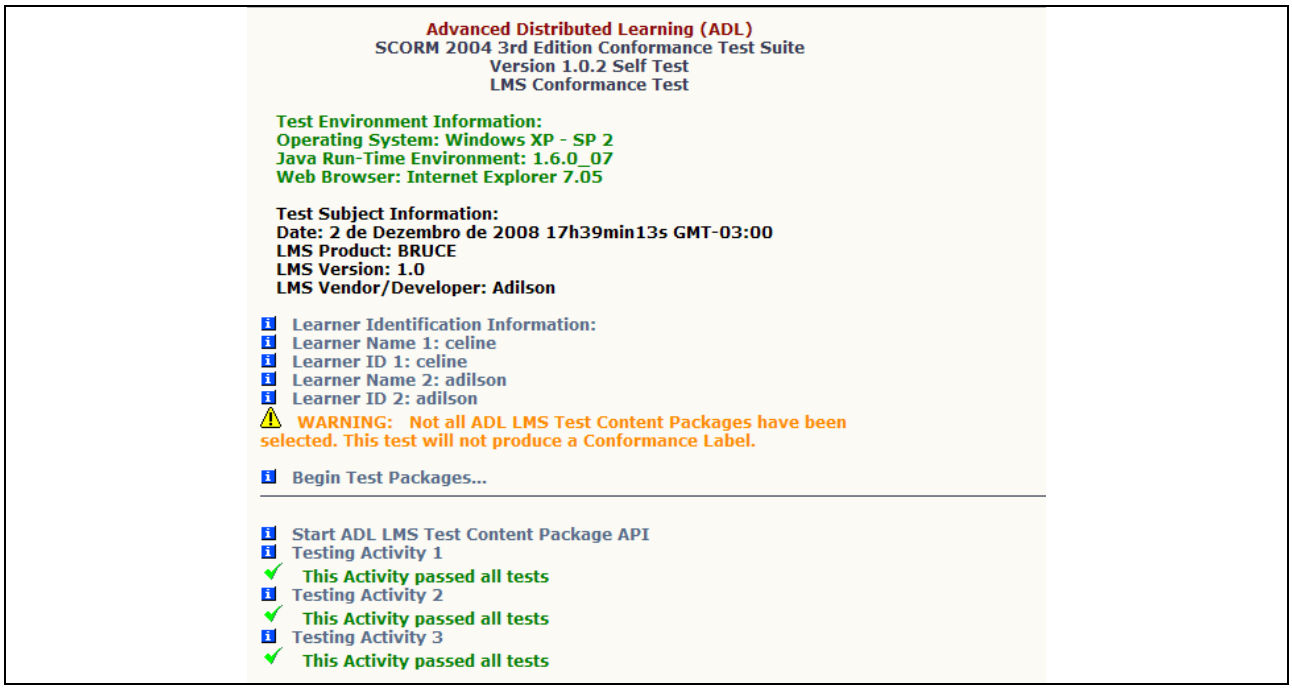


Figura 79. Página apresentando a aprovação dos testes do primeiro pacote

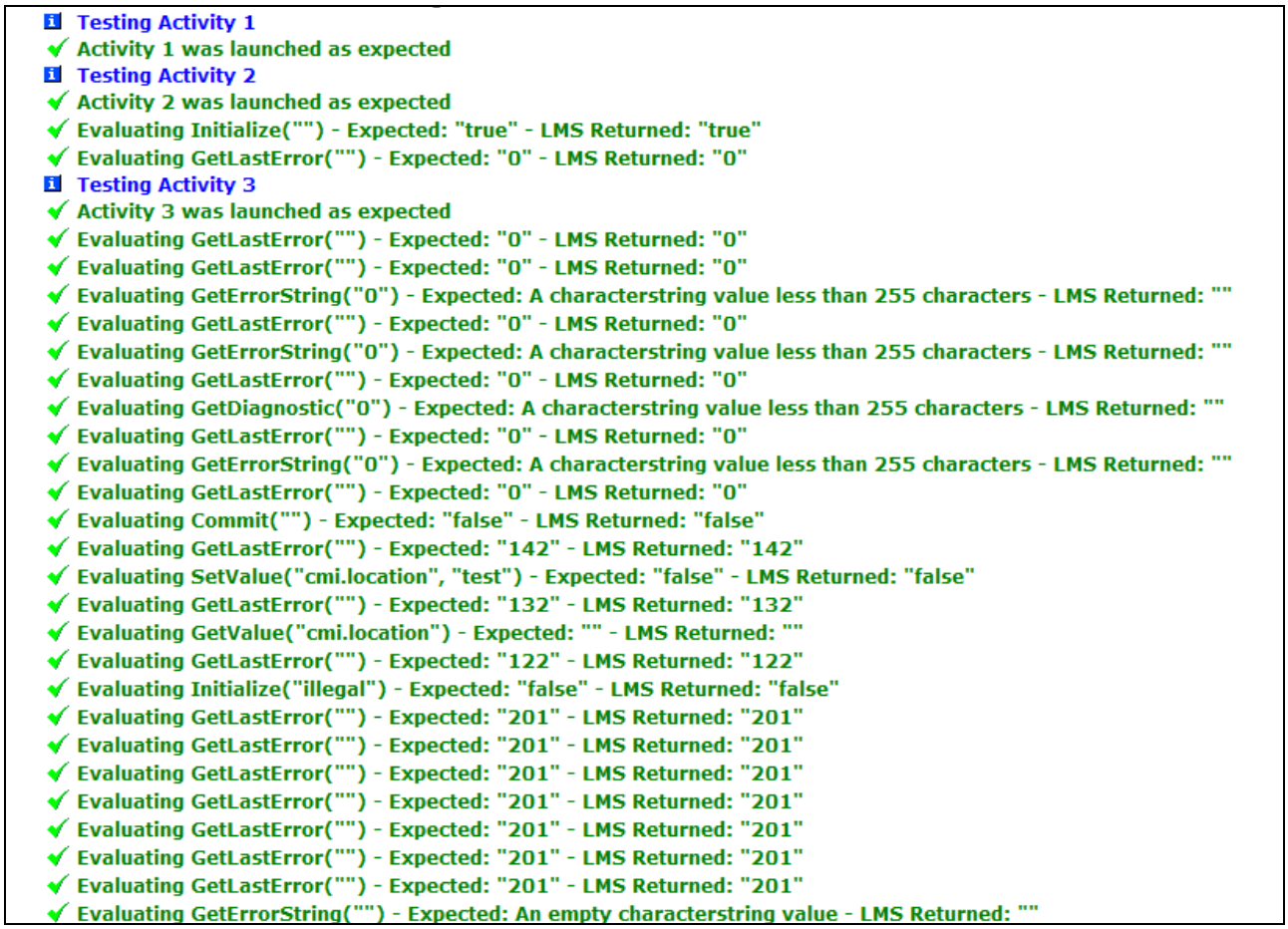


Figura 80. Página com uma pequena parte do registro detalhado das ações efetuadas no teste

O conjunto de testes verifica pequenos detalhes. Por exemplo, se na especificação havia um requisito que um determinado *Data Model* deveria aceitar no mínimo 200 itens, os testes incluem os 200 itens, inclusive atribuindo e verificando se foram atribuídos todos os valores desses 200 itens.

Para comparar o nível de detalhamento e *stress* dos testes, o segundo pacote, que testa a consistência de todos os elementos do *Data Model*, gera um log detalhado de 7Mb.

Foram realizados no BRUCE todos os testes da CTS e todas as operações foram realizadas com sucesso. Com isso, garante-se que os ambientes que forem desenvolvidos com o CELINE estão em conformidade com a especificação SCORM.

5 CONCLUSÕES

Um grande problema na continuidade do uso de Ambientes Inteligentes de Aprendizagem produzidos nos trabalhos acadêmico-científicos é a dificuldade na manutenção dos conteúdos, onde normalmente eles são produzidos no próprio ambiente, se referem a links externos, ou são cópias de arquivos como PDFs, DOCs ou imagens. O problema com essa solução está na reutilização do material produzido no ambiente e o reaproveitamento de conteúdos produzidos por outros.

Com intuito de fomentar o prolongamento da vida útil desses tipos de ambientes, essa proposta fornece uma infra estrutura para que esses ambientes utilizem conteúdo no formato SCORM. A vantagem de empregar conteúdo nesse formato é a independência do ambiente em que será executado, ou seja, a portabilidade desses arquivos. O material instrucional é produzido em ferramentas sem vínculo com ambiente algum, e uma vez que esses ambientes estejam em conformidade com a especificação, o material pode ser executado por eles.

Como os Ambientes Inteligentes de Aprendizagem, em sua natureza, operam com adaptação de estratégias pedagógicas e conteúdos, respeitando a individualidade do aluno, a infra estrutura aqui fornecida também provê meios para que os pacotes SCORM possam ser personalizados por esses ambientes.

Para a realização desse trabalho, inicialmente realizou-se uma pesquisa sobre as especificações e padrões existentes para o processo instrucional. A especificação SCORM foi a escolha pela completude de seus requisitos: ela trata de aspectos desde a criação e estruturação do material instrucional, comunicação do conteúdo com o ambiente, até as regras para a ordem e condições para a apresentação do assunto. Por essa razão, precisou de um estudo aprofundado sobre essa especificação e que consumiu a maior parte do tempo durante o trabalho, inclusive experimentando ferramentas de autoria e ambientes de execução. Além disso, para a especificação e implementação da infra estrutura, estudou-se o paradigma do desenvolvimento de componentes e como a especificação JEE poderia ser encaixado. A decisão pela componentização ocorreu pelas suas características em fornecer funcionalidades encapsuladas, isolando os detalhes para o usuário. Assim, a infra estrutura proposta foi realizada na forma de um componente, batizado de CELINE. O Java foi escolhido por duas razões: primeiro pelo domínio desse pesquisador no assunto, e segundo,

verificou-se que a maioria dos Ambientes Inteligentes de Aprendizagem, principalmente, os Sistemas Tutores Inteligentes, são desenvolvidos com essa linguagem.

Os trabalhos correlatos foram importantes para relacionar os recursos que um ambiente necessita para gerenciamento de pacotes SCORM, além de conhecer as melhorias e adaptações na especificação que vem sendo realizado nas pesquisas internacionais, o que determinou o desenvolvimento da adaptabilidade no componente. A partir dessas pesquisas pôde-se concluir que nenhum outro trabalho explorou a possibilidade de fornecer recursos para execução de conteúdo SCORM como um módulo a ser anexo em um sistema. Todos os trabalhos tinham a implementação da especificação vinculada ao próprio ambiente.

Baseando-se nesses levantamentos, foi possível então especificar a arquitetura do componente, através de casos de uso e diagramas de classes, o que levou a decisão de dividir em dois módulos, um responsável pela especificação SCORM e outro pelo LMS. O primeiro atende a especificação independente do contexto da aplicação (web, interface gráfica ou dispositivos móveis). Além disso, com a divisão pôde-se contar com a ajuda de um bolsista para a programação do módulo LMS, que foi implementado utilizando durante um ano o PIPG (Programa de Integração Pós-Graduação e Graduação) dessa instituição de ensino.

O pesquisador orientou um TCC, que produziu um Sistema Tutor Inteligente, com intuito de experimentar uma versão inicial do componente, onde a proposta era avaliar a execução de pacotes de SCORM e a condução da adaptabilidade desses pacotes. Durante o desenvolvimento desse TCC, foram resolvidos erros de implementação e melhorados aspectos de interface entre o componente e o ambiente.

Também foi desenvolvido um ambiente, batizado de BRUCE, para valer-se como documentação na utilização do componente. Nele estão relacionados todos os recursos de interfaceamento e exemplos de criação das páginas do sistema.

Outros dois ambientes foram desenvolvidos com o uso do componente, inclusive colocados a prova em “sala de aula”. O primeiro, chamado ALICE, foi utilizado por 31 alunos do primeiro semestre do curso de Ciências da Computação. O segundo ambiente, batizado como ASTRO, foi desenvolvido com programadores que não estiveram envolvidos com o projeto do componente, podendo assim experimentar a dificuldade e complexidade na utilização do CELINE. Com

dedicação semanal de 12 horas, no prazo de um mês o ambiente estava desenvolvido e implantando. Ele foi utilizado por 46 alunos da disciplina do primeiro semestre de Ciências da Computação e Sistemas de Informação, e mais uma turma de 3 alunos do curso de Licenciatura em Computação.

Nos experimentos realizados com esses dois ambientes, foram desenvolvidos conteúdos com as ferramentas CourseLab e MOS Solo. Ficou evidente nas turmas a cativação em utilizar os materiais instrucionais, tanto pela sua organização como pelos seus recursos. Essas experiências mostram que, mesmo se não houver a reusabilidade do material em outros cursos, instituições ou ambientes, o que é o apelo em seguir o modelo SCORM, vale a sua aplicação devido aos recursos de interatividade que as ferramentas de autoria oferecem na confecção do material.

Os alunos de Licenciatura, todos também professores em escolas públicas, mostraram interesse em implantarem nas escolas onde lecionam. Como responsáveis pela informática em suas instituições, eles estavam habituados em produzir o material instrucional usando MS Power Point. Comparando às possibilidades com o SCORM, um ponto que eles colocaram como evidente é o acompanhamento das atividades com os alunos. A interação dos alunos com os materiais com MS Power Point não consegue ser rastreado, como acontece com o SCORM. Porém, para essa rastreabilidade acontecer é necessário um computador para executar como servidor, recursos que a maioria das escolas públicas não dispõe.

Ambos os ambientes se comportaram sem erros durante os experimentos em sala de aula e com um desempenho adequado para não perturbar o foco dos alunos. A efetivação desses experimentos oferece possibilidades para a continuidade nos próximos semestres letivos, tanto na elaboração de novos conteúdos para as disciplinas, como para a aplicação em outras disciplinas.

O experimento com o ASTRO comprovou que, apesar da falta de experiência e conhecimento dos programadores, com um tempo reduzido foi possível oferecer um ambiente que execute e acompanhe a interação entre aluno e conteúdo.

Apesar do sucesso na utilização de conteúdo no formato SCORM, e ter provado a eficácia da infra estrutura proposta nesse trabalho, o ponto fraco desses experimentos é que ambos não ofereceram recursos de adaptação, o que não pôde ser provado pelo trabalho. O STI SAGAA promoveu a adaptação nos pacotes SCORM, porém ele não foi experimentado em sala.

Para comprovar a compatibilidade do componente com a especificação SCORM, foram realizados os testes de conformidade a nível de LMS (*Learning Management System*) com o *SCORM 2004 3rd Edition Conformance Test Suite 1.0.2*. O resultado apresentou total compatibilidade com a especificação, garantindo ao ambiente que utilizar o componente o selo "*LMS SCORM 2004 3rd Edition Conformant*".

5.1 CONTRIBUIÇÕES DA DISSERTAÇÃO

As contribuições diretas desse trabalho são:

- Um *framework* em Java, CELINESCORM, para gerenciamento de pacotes SCORM. Assim, é possível implementar qualquer tipo de aplicação em Java, deixando a responsabilidade do SCORM para o *framework*;
- Um componente, CELINE, que oferece a infra estrutura para desenvolvimento de aplicações web que gerenciam, adaptam e apresentam conteúdo SCORM;
- Dois ambientes, implantados em duas instituições de ensino diferentes, que permitem aos professores utilizarem conteúdo SCORM com suas turmas.

Com a execução e realização desse trabalho, foram publicados dois artigos. O primeiro foi resultado do estudo e experimentação de uma ferramenta de autoria SCORM. O artigo foi apresentado no X Ciclo de Palestras da Revista Novas Tecnologias na Educação, em 2007, com o título "Aplicação das Técnicas de Projeto Instrucional 4C/ID na Produção de Objetos de Aprendizagem em Conformidade com o SCORM usando um Software Livre como Ferramenta de Autoria", sendo os co-autores o mestrando Rafael de Santiago e o orientador dessa dissertação. O segundo artigo intitulado "Adaptação de Conteúdo SCORM em Ambientes Inteligentes de Aprendizagem", como co-autor também o orientador, foi publicado no XIX Simpósio Brasileiro de Informática na Educação (SBIE 2008), onde foram demonstrados os recursos do CELINE e as experiências com o desenvolvimento dos ambientes.

Pelos resultados apresentados, os objetivos deste trabalho foram plenamente alcançados, visto que, com a conclusão dessa dissertação, a comunidade tem a sua disposição a infra estrutura para desenvolver ambientes que usufruam das vantagens em utilizar conteúdos no formato SCORM, e ainda podendo adaptá-los sem comprometer a especificação.

5.2 TRABALHOS FUTUROS

As possíveis continuações, aprimoramentos ou utilizações desse trabalho são:

- Implementar um Sistema Tutor Inteligente para uso dos alunos;
- Especificar uma arquitetura que forneça a estrutura do conteúdo tal como expostos em Su *et al* (2005) e Kim, Yoo e Kim (2005), para facilitar o desenvolvimento na adaptação de conteúdo;
- Incorporar recursos para aprendizagem colaborativa como em Lin *et al* (2005);
- Usar em computação ubíqua com o *framework* CELINESCORM;
- Fornecer dados sobre as interações (`cmi.interactions`);
- Armazenar e fornecer dados sobre os caminhos da aprendizagem (a ordem em que as atividades foram clicadas);
- Converter as *tags* customizadas para JSF (*Java Server Faces*);
- Incorporar recursos sobre as especificações da IMS GLC e LOM;
- Fazer um componente que disponibilize infra estrutura para desenvolvimento de Repositórios de Objetos de Aprendizagem;
- Desenvolver uma ferramenta para criação de Ambientes Inteligentes de Aprendizagem;
- Manter o componente para as atualizações das versões do SCORM.

REFERÊNCIAS

ABDULLAH, N.A.; DAVIS, H.C. A real-time personalization service for SCORM. p. 61-63. In: **V IEEE International Conference on Advanced Learning Technologies**, 2005.

ABEL, R.; HUMES, L.; MATTSON, L.; McKELL, M.; RILEY, K.; SMYTHE, C. **Achieving Learning Impact 2007**: Annual Report from IMS Global Learning Consortium on the Latest Trends in Learning, Technology & Standards. Disponível em: <http://www.imsglobal.org/learningimpact2007/li2007report.cfm>. Acesso em: 14 set. 2007.

ADL. **SCORM® 2004 3rd Edition Content Aggregation Model (CAM) Version 1.0**. ADL: Virginia, 2006a.

____. **SCORM® 2004 3rd Edition Overview Version 1.0**. ADL: Virginia, 2006b.

____. **SCORM® 2004 3rd Edition Run-Time Environment (RTE) Version 1.0**. ADL: Virginia, 2006c.

____. **SCORM® 2004 3rd Edition Sequencing and Navigation (SN) Version 1.0**. ADL: Virginia, 2006d.

____. **Certified Products.** Disponível em: <http://www.adlnet.gov/scorm/certified/index.aspx>. Acesso em: 24 out. 2007a.

____. **Previous Versions**. Disponível em: <http://www.adlnet.gov/scorm/history/index.aspx>. Acesso em: 24 out. 2007b.

____. **SCORM® 2004 3rd Edition Sample Run-Time Environment.** Disponível em: <http://www.adlnet.gov/scorm/20043ED/SRTE.aspx>. Acesso em: 21 nov. 2008.

APACHE COMMONS. **Commons Digester**. Disponível em: <http://commons.apache.org/digester/>. Acesso em: 11 nov. 2007.

ARMSTRONG, E.; BALL, J.; BODOFF, S.; CARSON, D.B.; EVANS, I.; GREEN, D.; HAASE, K.; JENDROCK, E. **The J2EE 1.4 Tutorial**. Sun Microsystems, 2004.

BARROCA, L. *et al*. Conceitos Básicos**.** p 1-26. In: **Desenvolvimento Baseado em Componentes**. Itana M.S. Gimenes e Elisa H.M. Huzita (org). Rio de Janeiro: Editora Ciência Moderna Ltda. 2005.

BAYERN, S. **JSTL in Action**. Manning Publications, 2003.

BITONTO, P. DI; ROSELLI, T. Dynamic Tailoring of SCORM Sequencing and Navigation Rules. p. 373-375. In: **VIII IEEE International Conference on Advanced Learning Technologies**, 2008.

BOND, M.; *et al*. **Aprenda J2EE em 21 dias**. São Paulo: Pearson Education do Brasil, 2003.

CHANG, H.-P.; WANG, C.-C.; JAN, K. H.; SHIH, T.K. SCORM sequencing testing for sequencing control mode. In: **XX International Conference on Advanced Information Networking and Applications**, Volume 2, 2006.

CHANG, W.-C.; HSU, H.-H.; SHIH, T.K.; CHUNG, L.-Y.; YEN, Y-W. Using SPC Table to Strengthen SCORM Compliant Assessment. p. 825-830. In: **XXI International Conference on Advanced Information Networking and Applications**, 2007.

CHIEN, T.-C.; KAO, F.-C. The design of load-balancing LMS based on decomposition structure. p. 783-787. In: **V IEEE International Conference on Advanced Learning Technologies**, 2005.

CHUN, C.-P.; CHANG, C.-P.; YEH, C.-W.; YEH, Y.-F. A Web-service oriented framework for building SCORM compatible learning management systems. p. 156-161. In: **International Conference on Information Technology: Coding and Computing**, Volume 1, 2004.

CRISTEA, A.I.; STEWART, C. Automatic Authoring of Adaptative Education Hypermidia. In: **Web-Based Intelligent E-Learning Systems: Technologies and Applications.** Zongmin Ma (ed). Information Science Publishing, 2006.

FILATRO, A. **Design Instrucional Contextualizado: educação e tecnologia**. São Paulo: Editora Senac São Paulo, 2004.

FOWLER, M. **UML essencial: um breve guia para a linguagem-padrão de modelagem de objetos**. 3.ed. Porto Alegre; Bookman, 2005.

FREIRE, H.; GUAPO, M. A. **Java Timeline**. p. 28-35. Mundo Java, n. 7, 2004.

GAMMA, E. et al. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GETAHEAD. **DWR**: easy AJAX for Java, 2004. Disponível em: <http://getahead.org/dwr/overview/dwr>. Acesso em: 20 maio 2008.

GUBUSOFT. **Treeview: JavaScript DHTML Tree Menu.** Disponível em: <http://www.treeview.net>. Acesso em: 29 nov. 2008.

HATZILYGEROUDIS, I.; PRENTZAS, J. Knowledge Representation in Intelligent Educational Systems. In: **Web-Based Intelligent E-Learning Systems: Technologies and Applications.** Zongmin Ma (ed). Information Science Publishing, 2006.

IEEE LTSC. **1484.12.1. IEEE Standard for Learning Object Metadata.** New York: IEEE, 2002.

IMS GLOBAL LEARNING CONSORTIUM. **IMS Shareable State Persistence SCORM® Application Profile:** version 1.0 Final Specification. IMS/GLC, 2004.

IRLBECK, S. MOWAT, J. Learning Content Management Systems (LCMS). In: **Learning Objects: Standards, Metadata, Repositories & LCMS**. Keith Harman e Alex Koohang (eds). California: Informing Science Press, 2007.

JACOBI, J.; FALLOWS, J. R. **Pro JSF e AJAX**: construindo componentes ricos para

internet. Tradução Cleuton Sampaio de Melo Jr. Rio de Janeiro: Ciência Moderna, 2007.

JCP. **The Java Community Process Program**. Disponível em: <http://www.jcp.org/en/ home/index>. Acesso em: 24 out. 2007.

KAVCIC, A. Fuzzy user modeling for adaptation in educational hypermedia. In: **IEEE Transactions On Systems, Man, And Cybernetics – Part C: Applications And Reviews**, Vol. 34, No. 4, November, 2004.

KARAMPIPERIS, P; SAMPSON, D. Automatic Learning Object Selection and Sequencing Web-Based Intelligent Learning Systems. p. 56-71. In: **Web-based intelligent e-learning systems: technologies and applications**, 2006.

KAZANIDIS, I.; SATRATZEMI, M. Combine Adaptive Education Hypermedia and SCORM compliant Learning Management Systems. p. 330-331. In: **VII IEEE International Conference on Advanced Learning Technologies**, 2007.

KIM, H-A.; KIM, K.-H.; PAIK, S.-K. SCO Control Net for the Process-Driven SCORM Content Model. p. 1977-1982. In: **X International Conference on Advanced Communication Technology**, Volume 3, 2008.

KIM, K.-H.; YOO, H.-J.; KIM, H.-S. A Process-Driven e-Learning Content Organization Model. p. 328-333. In: **IV Annual ACIS International Conference on Computer and Information Science**, 2005.

KURNIAWAN, B. **Java para a Web com Servlets, JSP e EJB**. Rio de Janeiro: Editora Ciência Moderna Ltda, 2002.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise a ao projeto orientados a objetos. 2a edição. Porto Alegre: Bookman, 2004.

LEME, F. **Programando Servlets: Conceitos Avançados**. p. 24-30. Java Magazine, n. 20, 2005.

LIMONGELLI, C.; SAMPIETRO, G., TEMPERINI, M. Configuration of Personalized e-Learning Courses in Moodle. p. 2680-2686. In: **EUROCON, The International Conference on "Computer as a Tool"**, 2007.

LIN, H.W.; CHANG, W.-C.; YEE, G.; SHIH, T.K.; WANG, C.-C.; YANG, H.-C. Applying Petri nets to model learning sequence with SCORM specification in collaborative learning. p. 181-186. In: **Proceedings of the 2005 International Conference on Active Media Technology**, 2005.

LIN, H.W.; SHIH, L.K.; CHANG, W.-C.; YANG, C.-H.; WANG, C.-C. A Petri nets-based approach to modeling SCORM sequence. p. 1247-250. In: **IEEE International Conference on Multimedia and Expo**, Volume 2, 2004.

MAGOULAS, G.D.; PAPANIKOLAOU, K.A.; GRIGORIADOU, M. Neuro-fuzzy synergism for planning the content in a web-based course. In: **Informatica,** 25. p. 39-48, 2001.

McCLURE, C. **Software Reuse**: a standards-based guide. Los Alamitos: IEEE, 2001.

MITCHELL, J.L.; FARHA, N. Learning Object Metadata: Use and Discovery. In: **Learning Objects: Standards, Metadata, Repositories & LCMS**. Keith Harman e Alex Koohang (eds). California: Informing Science Press, 2007.

MOODLE. **A Free, Open Source Course Management System for Online Learning**. Disponível em: <http://moodle.org/>. Acesso em: 21 nov. 2008.

NAKABAYASHI, K.; HOSHIDE, T.; HOSOKAWA, M.; KAWAKAMI, T.; SATO, K. Design and Implementation of a Mobile Learning Environment as an Extension of SCORM 2004 Specifications. p. 369-373. In: **VII IEEE International Conference on Advanced Learning Technologies**, 2007.

NORDMANN, M.; NEUMANN, J. Learning Application Suite Creating and Playing SCORM Compatible Web and Computer Based Training. p. 572-573. In: **VIII IEEE International Conference on Advanced Learning Technologies**, 2008.

PENDER, T. **UML, A Bíblia**. Rio de Janeiro: Elsevier, 2004.

POWER, G.; DAVIS, H.C.; CRISTEA, A.I.; STEWART, C.; ASHMAN, H. Goal oriented personalisation with SCORM. p. 467-471. In: **V IEEE International Conference on Advanced Learning Technologies**, 2005.

POZZEBON, E; BARRETO, J. **Inteligência artificial no ensino com tutores inteligentes**. Porto Alegre, 2002. Disponível em: <http://www.das.ufsc.br/%7Eeliane/artigos/pozzebon02i.pdf>. Acesso em: 14 set. 2007.

PRESSMAN, R. S. **Engenharia de Software**. 5a. edição. São Paulo: McGraw-Hill, 2002.

RAABE, A.L.A. **Uma proposta de arquitetura de Sistema Tutor Inteligente baseada na Teoria das Experiências de Aprendizagem Mediadas**. 2005. 152f. Tese (Doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Informática na Educação.

REY-LÓPEZ, M.; DÍAZ-REDONDO, R. P.; FERNÁNDEZ-VILAS, ANA; PAZOS-ARIAS, J.J.; GARCÍA-DUQUE, J.; GIL-SOLLA, A.; RAMOS-CABRER, M. An extension to the ADL SCORM standard to support adaptivity: The t-learning case-study. In: **Computer Standards & Interfaces**, 2008. No Prelo.

SARDAGNA, M.; VAHLDICK, A. Aplicação do Padrão Data Access Object (DAO) em Projetos Desenvolvidos com Delphi. In: **IV Escola Regional de Banco de Dados**, Florianópolis, 2008.

SCHAPPO, J.C. **Sistema Tutor Inteligente para o Ensino do Gerenciador de Armazenamento e Arquivos**. 2008. Trabalho (Conclusão de Curso) Bacharel em Ciências da Computação, FURB, Blumenau.

SHEMSHADI, A.; SOROOR, J.; TAROKH, M.J. An Innovative Framework for the New Generation of SCORM 2004 Conformant E-Learning Systems. p. 949-954. In: **V International Conference on Information Technology: New Generations**, 2008.

SHIN, S.-O.; LEE, J.-O.; BAIK, D.-K. Self-learning System Based on Metadata Management Module (MMM) for Providing Self-learning Service. p. 100-105. In: **I IEEE International Symposium on Information Technologies and Applications in Education**, 2007.

SILVA, L.; STRINGHINI, D.; MUSTARO, P.N.; SILVEIRA, I. Adaptive Learning through Conceptual Lattice-based SCORM Meta-Objects. p. 739-748. In: **VII International Conference on Information Technology Based Higher Education and Training**, 2006.

SILVA, J.M.C; BAVARESCO, N.; SILVEIRA, R.A. Proposta de um Sistema Multi-agentes para a aplicação de Objetos Inteligentes de Aprendizagem seguindo o padrão SCORM. p. 264-274. In: **XVIII Simpósio Brasileiro de Informática na Educação**. São Paulo: SBC, 2007.

SU, J.-M.; TSENG, S.-S.; WENG, J.-F.; CHEN, K.-T.; LIU, Y.-L.; TSAI, Y.-T. An Object based Authoring Tool for Creating SCORM Compliant. p. 209-214. In: **XIX International Conference on Advanced Information Networking and Applications**, Volume 1, 2005.

SUN. **Core J2EE patterns: data access object**. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>. Acesso em: 15 abr. 2007.

VOSSEN, G.; WESTERKAMP, P. Towards the Next Generation of E-Learning Standards: SCORM for Service-Oriented Environments. p. 1031-1035. In: **VI International Conference on Advanced Learning Technologies**, 2006.

WERNER, C. M. L; BRAGA, R. M.M. A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes. In: **Desenvolvimento Baseado em Componentes**. Itana M.S. Gimenes e Elisa H.M. Huzita (org). Rio de Janeiro: Editora Ciência Moderna Ltda. 2005.

WILEY, D. A. **Learning Object Design and Sequencing Theory.** 2000. 142f. Thesis (Doctor of Philosophy) – Departament of Instructional Psychology and Technology, Brigham Young University.

XSTREAM. **XStream – About XStream.** Disponível em: <http://xstream.codehaus.org>. Acesso em: 29 nov. 2008.

YUNG, L. **J2EE Visão Geral**. p.26-30. Mundo Java, n. 4, 2004.

APÊNDICES

A CÓDIGO FONTE

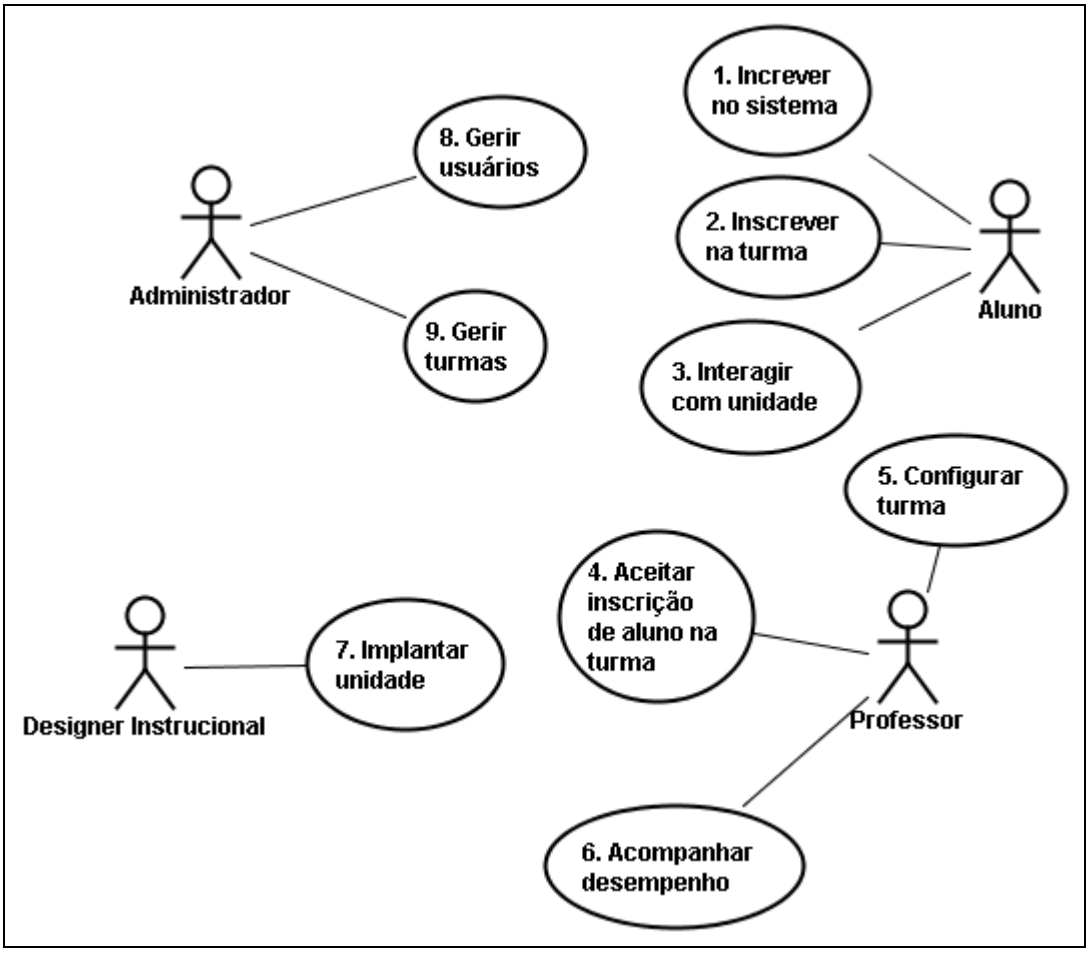
A implementação do componente, assim como todas as dependências, e os ambientes BRUCE e ASTRO, encontram-se disponíveis no CD entregue à secretaria do programa de mestrado.

ANEXOS

A ESPECIFICAÇÃO DO ASTRO

ASTRO – Ambiente de Gerenciamento de Aprendizagem

CASOS DE USO



Os casos de uso abaixo estão listados na ordem de implementação.

1. Inscrever no sistema
O usuário se cadastra no sistema, informando um nick (como um e-mail), nome e senha.
Cenário: 1. O usuário digita os dados necessários (nick, nome e senha) 2. O sistema inclui o usuário se o nick não existir.

2. Inscrever na turma
O aluno solicita inscrição para turmas.
Cenário: 1. O aluno deseja se inscrever em uma ou mais turmas. 2. O sistema apresenta a lista de turmas abertas, mostrando em cada uma delas se o usuário já

está ou não registro.
3. O aluno seleciona as turmas que deseja participar e confirma a inscrição.
4. O sistema avalia as turmas que foram marcadas ou desmarcadas em relação à lista original. No caso de turmas que ele acabou de marcar precisa colocar a inscrição numa lista de espera para aprovação ao professor.

Nota: veja o caso de uso "Gerir turma" para entender o que é uma turma aberta.

4. Aceitar inscrição de aluno na turma

O professor aceita a solicitação de um aluno fazer parte de sua turma

Cenário:
1. O professor deseja verificar se existem novos alunos solicitando inscrição nas turmas.
2. O sistema mostra todas as inscrições pendentes, apresentando o nome do aluno e a turma.
3. O professor seleciona os alunos e confirma a aceitação ou rejeição.
4. Para cada aluno selecionado o sistema inscreve o aluno na turma e envia e-mail a ele.

3. Interagir com unidade

O aluno abre uma unidade e interage.

Cenário:
1. O aluno deseja interagir com alguma unidade de conteúdo.
2. O sistema lista todos os conteúdos disponíveis para o aluno, baseando-se nas turmas em que ele está inscrito.
3. O aluno seleciona a unidade para abrir.
4. O sistema registra a data e hora do acesso e abre o curso.

Nota: a abertura e execução da unidade são de responsabilidade do componente CELINE

5. Configurar turma

O professor seleciona os pacotes disponíveis para a turma.

Cenário:
1. O professor deseja habilitar ou desabilitar conteúdos para a turma.
2. O sistema mostra todas as turmas do professor.
3. O professor seleciona uma turma.
4. O sistema mostra todos os conteúdos implantados diferenciado àqueles relacionados ou não com a turma.
5. O professor marca ou desmarca os conteúdos e confirma a configuração.
6. O sistema avalia os conteúdos que foram marcados ou desmarcados em relação à lista original, e adiciona ou remove os conteúdos da turma.

6. Acompanhar desempenho

O professor deseja visualizar os detalhes de interação dos alunos com seus conteúdos.

Cenários:
1. O professor deseja visualizar os desempenhos dos alunos das turmas.
2. O sistema mostra todas as turmas do professor.
3. O professor seleciona uma turma.
4. O sistema mostra uma lista de conteúdos da turma. Para cada conteúdo é apresentada a quantidade e a participação percentual de alunos que acessaram, e o tempo médio gasto por aluno.
5. O professor seleciona um conteúdo.

6. O sistema mostra todos os alunos que acessaram o conteúdo. Para cada aluno é apresentada a primeira data e a última data, tempo total gasto e percentual de conteúdo visualizado.
7. O professor seleciona um aluno.
8. O sistema mostra todas as atividades da unidade diferenciado àquelas que o aluno já acessou ou não.

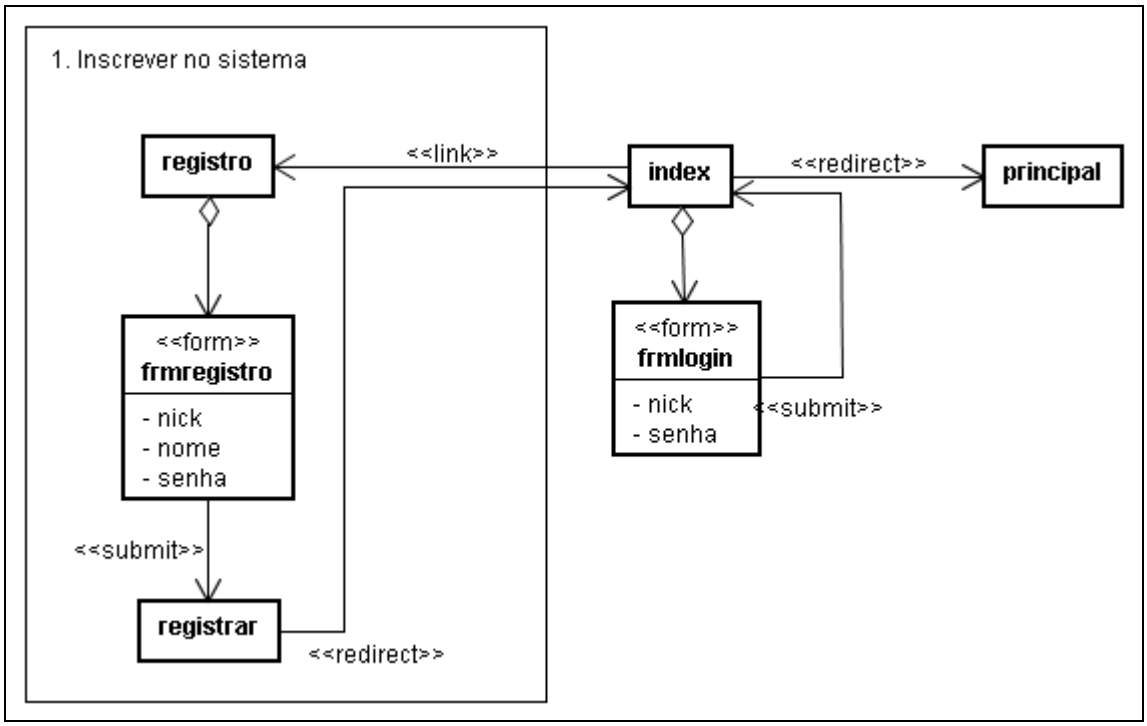
7. Implantar conteúdo
O designer instrucional implanta um pacote SCORM.
Nota: essa função é totalmente suportada pelo componente CELINE.

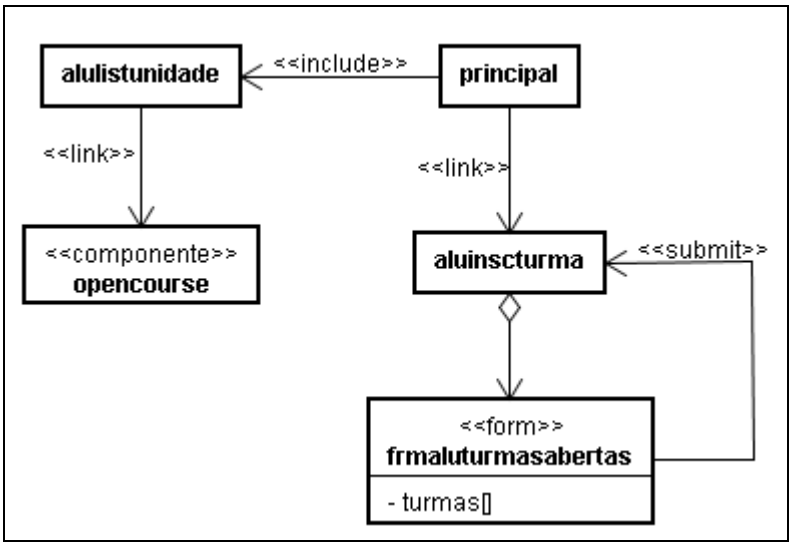
8. Gerir usuários
O administrador gerencia o cadastro de usuários do tipo administrador, professor e designer instrucional. O usuário tem nick, nome, senha e tipo.

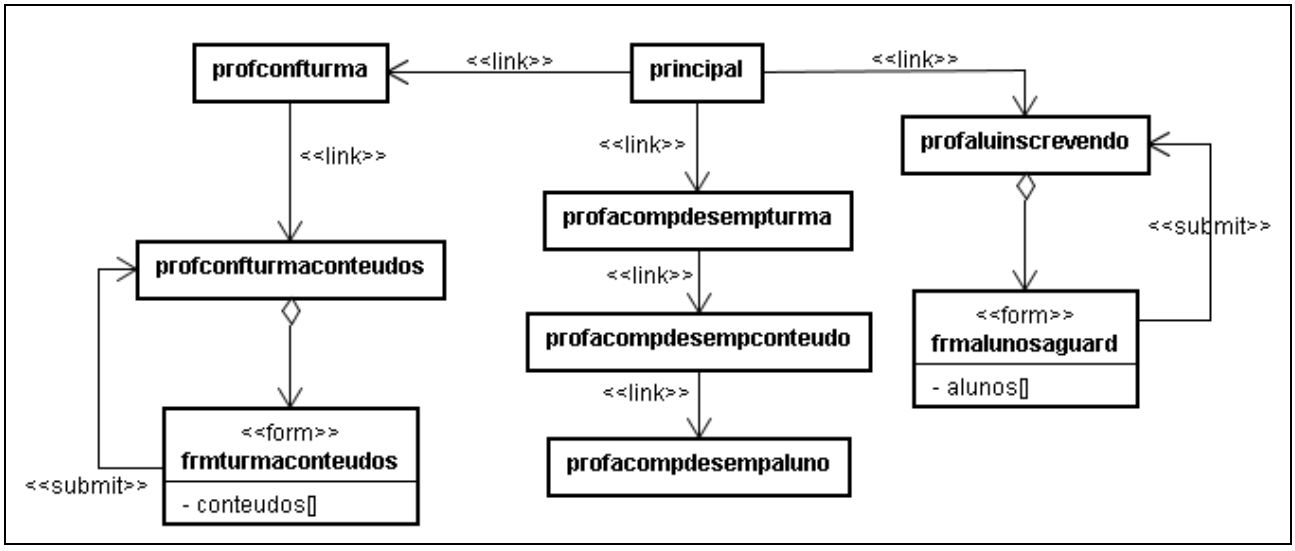
9. Gerir turmas
O administrador gerencia o cadastro de turmas. Uma turma tem uma descrição, data de abertura, fechamento e professor responsável.

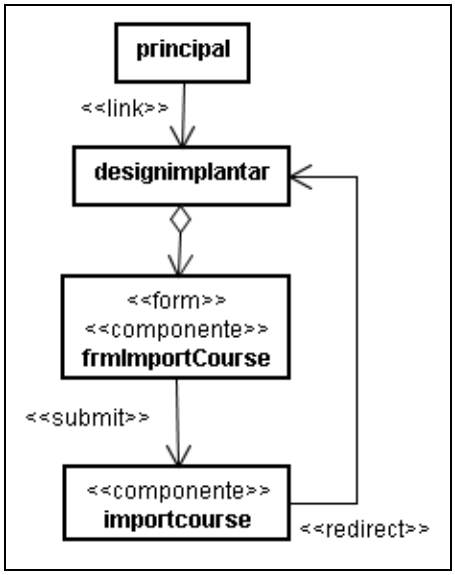
NAVEGABILIDADE

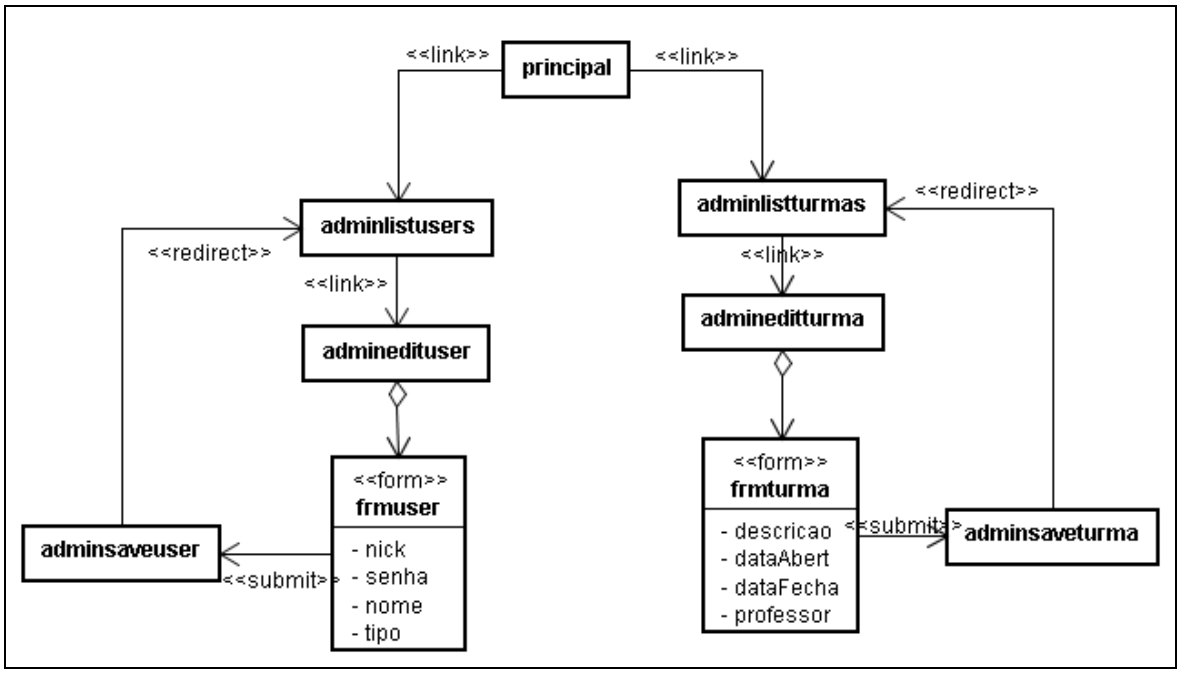
Cada diagrama está dividido por papéis. O primeiro se refere à página inicial do sistema. Em seguida, a página de utilização pelo aluno, depois o professor, o designer e o último o administrador.




alulistunidade
<<include>>
principal
<<link>>
<<link>>
<<componente>>
opencourse
aluinscturma
<<submit>>
<<form>>
frmaluturmasabertas
- turmas[]


profconfturma
<<link>>
principal
<<link>>
<<link>>
<<link>>
profaluinscrevendo
<<link>>
profconfturmaconteudos
profacompdesempturma
<<submit>>
<<link>>
<<form>>
frmalunosaguard
- alunos[]
profacompdesempconteudo
<<link>>
<<form>>
frmturmaconteudos
- conteudos[]
<<submit>>
profacompdesempaluno


principal
<<link>>
designimplantar
<<form>>
<<componente>>
frmImportCourse
<<submit>>
<<componente>>
importcourse
<<redirect>>


<<link>>
principal
<<link>>
adminlistusers
<<redirect>>
<<link>>
adminedituser
<<form>>
frmuser
- nick
- senha
- nome
- tipo
adminsaveuser
<<submit>>
adminlistturmas
<<redirect>>
<<link>>
admineditturma
<<form>>
frmturma
- descricao
- dataAbert
- dataFecha
- professor
<<submit>>
adminsaveturma

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo