Elaine Pasqualini

Proposta de Adaptação do Sistema de Gerenciamento de Aprendizagem COL ao Modelo de Padronização SCORM

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia.

São Paulo
2005

Elaine Pasqualini

Proposta de Adaptação do Sistema de Gerenciamento de Aprendizagem COL ao Modelo de Padronização SCORM

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia.

Área de Concentração:
Engenharia Naval

Orientador:
Prof. Dr. Oscar B. Augusto

São Paulo
2005

"Hay hombres que luchan un día y son buenos
Hay otros que luchan un año y son mejores
Hay quienes luchan muchos años y son muy buenos
Pelo hay los que luchan toda la vida
Esos son los imprescindible".
(Bertold Brecht)

Agradecimentos

Primeiramente, agradeço a Deus por ter dado forças para eu começar, prosseguir e terminar este trabalho.

À minha família, pelo fato de entender meu objetivo e ter me incentivado e motivado para a conclusão de mais esta jornada na minha vida.

Ao meu orientador Dr. Oscar B. Augusto e a Dra. Regina M. Silveira, que souberam conduzir o trabalho, sempre prontos para auxiliar, sugerindo idéias inovadoras.

Ao meu amigo Paulo H. Chíxaro, que compartilhou comigo várias experiências de aprendizado por todos estes anos de trabalho juntos.

Aos professores Doutores, Mardel B. de Conti, Cláudio Sampaio, Marco Brinatti e Rui Botter, que demonstraram a importância do saber e do conhecimento.

À minha amiga Eunice C. Sanches Belloti pela atenção e carinho dedicado a minha pessoa.

E a todos os meus amigos, que participaram de forma direta ou indireta, meu muito obrigado.

Resumo

O trabalho trata de uma proposta de adaptação do Sistema de Gerenciamento de Aprendizagem COL (Cursos on-line) desenvolvido pelo Laboratório de Arquitetura de Redes de Computadores, da Universidade de São Paulo, ao modelo de padronização SCORM (Sharable Content Object Reference Model). O método apóia-se em referências, buscando validar a teoria desenvolvida, seguindo uma padronização para garantir aos objetos de aprendizagem interoperabilidade, reusabilidade e acessibilidade dos mesmos. Como resultado, reformulou-se a base de dados do COL, implementou-se uma arquitetura por meio de APIs (Application Program Interface) de acordo com o padrão SCORM, modelando-se a comunicação entre o COL e os objetos, visando obter chamadas e respostas remotas por meio do protocolo de transporte HTTP (Hipertext Transfer Protocol) e da linguagem ASP (Active Server Pages).

Abstract

This work presents a proposal to adapt the COL (Courses on-line) Learning Management System developed by the São Paulo University Computer Networks Architecture Laboratory to the SCORM (Sharable Content Object Reference Model) standardized model. The method is based on references, seeking to validate the theory developed by following standardization to warrant interoperability, reusability and accessibility to the learning objects. As a result, the COL data base was reformulated, was implemented an architecture through APIs (Application Program Interface) in conformity with SCORM and the communication between COL and learning objects were suggested aiming at getting remote calls and answers via HTTP (Hipertext Transfer Protocol) transport protocol and Active Server Pages (ASP) Language.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ADL – Advanced Distributed Learning

AICC - Aviation Industry CBT Committee

API - Application Program Interface

ASP – Active Server Pages

CAM - Content Aggregation Model

CBT – Computer Based Training

CMI – Computer Manager Instruction

COL - Sistema de Gerenciamento de Aprendizagem Cursos On-Line, desenvolvido pela Universidade de São Paulo

DOM – Document Object Model

FTP - File Transfer Protocol

HTML – HiperText Markup Language

HTTP - HiperText Transfer Protocol

IEEE – Institute of Electrical and Electronics Engineers

IMS - Instructional Management System

IIS - Internet Information Services

LARC - Laboratório de Arquitetura de Redes de Computadores

LMS – Learning Management System

LO – Learning Object

LOM - Learning Object Metadata

LTSC - Learning Technology Standards Committee Of The IEEE

ODBC - Open DataBase Connectivity.

PIF - Package Interchange File

RAM – Random Access Memory

RTE - Run-Time Environment

SCO - Sharable Content Object

SCORM – Sharable Content Object Reference Model

SMTP - Simple Mail Transfer Protocol

SQL - Structured Query Language

URL - Uniform Resource Locator

XML – Extensible Markup Language

W3C - World Wide Web Consortium

WWW – World Wide Web

Capítulo 1

Introdução

1.1 Contextualização

O presente trabalho se insere nas áreas de Educação, Tecnologia e Engenharia. O foco central é utilizar a tecnologia a serviço da comunidade Politécnica e outras instituições de ensino, criando novos meios de transmissão de informações e conhecimentos. Para isto, será pesquisada e proposta uma metodologia de adaptação do Sistema de Gerenciamento de Aprendizagem COL (Cursos On-Line) ao SCORM (Sharable Content Object Reference Model) em relação aos objetos de aprendizagem (conteúdos educacionais).

O COL é um sistema que permite centralizar as informações referentes a diversos cursos e possibilita a geração de dados para o acompanhamento da evolução do curso e do aluno. Reúne várias ferramentas necessárias para o docente criar cursos a distância, tais como: bate-papo, e-mail, fórum, lista de discussão, gerenciamento de conteúdo, avaliações, etc.

O padrão SCORM define um modelo de como se fazer e como se executar cursos baseados na Web. As normas do modelo são coleções de especificações, criando um grupo de habilidades de ensino via Web que permitem acessibilidade e reutilização de conteúdo educacional.

O estudo é elaborado com base em referências bibliográficas, partindo-se da teoria geral, buscando validar a metodologia de adaptação do COL ao padrão SCORM.

Segundo Moran; Masetto; Behrens (2000), na década de 1990, com a evolução dos computadores pessoais, houve a possibilidade dos usuários utilizarem mais os

computadores, tanto pelo barateamento, quanto pela facilidade de se trabalhar com os programas. Os computadores começaram também a ser utilizados em maior escala para a comunicação pessoal e acesso à informação.

Assim, cresceu o número de computadores interligados entre si para a transmissão e recepção de informações, denominada de redes. Dentre as redes, destaca-se a Internet. Criada nos anos 60 como suporte para pesquisa na área militar, foi também incorporada ao universo acadêmico, empresarial e outros. Com a criação em 1994 da tecnologia WWW (World Wide Web), incorporada à Internet, a facilidade de uso aumentou, o que fez com que mais pessoas a utilizassem.

A Internet disponibiliza, além da comunicação entre vários usuários, o acesso e o compartilhamento de dados como um serviço de informação ou de uma biblioteca.

Vários recursos de comunicação já estão disponíveis hoje nas escolas, como televisão, vídeo, computador, etc. Os Governos Estadual e Federal promovem a compra de computadores e outros meios de comunicação, visando, inclusive acesso à Internet, ampliando as possibilidades de conhecimento, como por exemplo, o intercâmbio de informação com outras escolas e bibliotecas. Além da Internet, algumas Instituições usam recursos ou tecnologias interativas que possibilitam contato em tempo real com alunos em vários locais espalhados geograficamente, como por exemplo, a videoconferência.

Outras escolas, já preparam materiais ou conteúdos educacionais (Learning Object - LO) baseados em uma estrutura multidisciplinar, visando aproveitar conteúdos de uma disciplina para outra, que podem ser acessíveis em qualquer lugar e hora, adaptando as necessidades e habilidades dos alunos.

Um LO é um instrumento que pode trazer novas possibilidades no desenvolvimento de material didático a ser usado na educação. A aplicação dos conceitos de modularização e reutilização, torna mais fácil a atualização de conteúdos

didáticos, reduzindo tempo e custo de desenvolvimento, testes, documentação e manutenção, permitindo possivelmente melhorias na qualidade de ensino.

Esses objetos podem ser usados em um determinado contexto e depois utilizados em contextos similares. O uso de repositórios (bancos de dados) de LO devidamente identificados e catalogados disponibilizam recursos didáticos que podem ser compartilhados em qualquer parte do mundo que tenha acesso à Internet, facilitando a criação e o desenvolvimento de cursos, tutoriais e outras situações de ensino e aprendizagem.

Um exemplo de projeto é o RIVED (Rede Internacional Virtual de Educação), de acordo com MEC (2004). Este projeto é de cooperação internacional entre países da América Latina, em que atualmente trabalham de forma colaborativa Brasil, Peru e Venezuela. No Brasil, o Projeto é desenvolvido pelo Ministério da Educação por meio da Secretaria de Educação a Distância, SEED e da Secretaria de Educação Média e Tecnológica, SEMTEC.

O Projeto RIVED é uma iniciativa com o propósito de trabalhar as disciplinas de Ciências e Matemática no ensino médio das escolas públicas, aproveitando o potencial das Tecnologias de Informática e da Comunicação. O programa envolve o design instrucional de atividades de ensino/aprendizagem, a produção de material pedagógico multimídia, capacitação de pessoal, rede de distribuição de informação e estratégias de avaliação da aprendizagem e do programa.

Um dos objetivos do projeto RIVED é desenvolver módulos educacionais que estimulam o raciocínio e pensamento crítico, trazendo questões relevantes aos alunos do ensino médio.

O RIVED utiliza a tecnologia de objetos de aprendizagem na confecção de seus módulos a fim de otimizar a produção e a reutilização. A possibilidade de reutilização

dos objetos de aprendizagem oferece uma forma de readaptar atividades para diferentes tipos de alunos.

Mais um exemplo de projeto é o LabVirt (Laboratório Virtual), desenvolvido pela Escola do Futuro da Universidade de São Paulo (USP). O projeto consiste no desenvolvimento de situações e problemas, ligados ao cotidiano, que são transformadas por universitários em simulações e animações publicadas na Internet e estas são discutidas e reutilizadas por diversos grupos em escolas públicas.

Este projeto, de acordo com Coscarelli (2004), tem como maior benefício a criação de uma nova configuração educacional. É como uma "comunidade de aprendizagem", em que alunos e professores de escolas e universidades, se encontram, trocam e produzem recursos educacionais e experiências didáticas. A idéia principal do projeto é fazer com que o aluno seja cada vez mais criativo.

Outra iniciativa é o projeto CESTA, segundo UFRGS (2004). O projeto CESTA (Coletânea de Entidades de Suporte ao uso de Tecnologia na Aprendizagem) foi idealizado com o objetivo de sistematizar e organizar o registro dos objetos de aprendizagem que foram desenvolvidos pela equipe de Pós-Graduação em Informática na Educação e pelo CINTED (Centro Interdisciplinar de Novas Tecnologias na Educação) da Universidade Federal do Rio Grande do Sul, para cursos de capacitação em Gerência de Redes, Videoconferência e na Pós-Graduação Latu-Sensu em Informática na Educação.

Todos esses cursos têm sido desenvolvidos em modalidade a distância e considerável quantidade de material didático de apoio foi projetado e construído para apoiar as atividades de aprendizagem.

Mais uma iniciativa é o Projeto TIDIA (Tecnologias da Informação no Desenvolvimento da Internet Avançada), financiado pela Fapesp (Fundação de Amparo

à Pesquisa do Estado de São Paulo). Este projeto visa à criação de ferramentas de suporte e apoio ao ensino e aprendizagem com interações presenciais e a distância.

Dentre os objetivos desse projeto, conforme FAPESP (2004) destacam-se: gerenciamento administrativo de cursos envolvendo estatísticas de acesso, padrões de interoperabilidade e reutilização de conteúdos educacionais, criação, organização e manutenção de um repositório de objetos de aprendizagem, construídos para serem compartilhados no desenvolvimento de novos conteúdos de ensino e aprendizagem.

Outro exemplo de projeto é o EduSourceCanada, desenvolvido no Canadá, com o intuito de criar uma rede de repositórios de objetos de aprendizagem. Para isto, fez-se um levantamento de ferramentas, sistemas e protocolos para que um esquema de reutilização possa ocorrer, permitindo que várias instituições de ensino possam utilizar o material didático armazenado no repositório, conforme EduSourceCanada (2004).

Nesse contexto, será pesquisada e proposta uma metodologia de adaptação do COL em relação a uma padronização dos objetos de aprendizagem, seguindo um modelo internacional.

1.2 Justificativa

Possibilitar a disseminação da informação na Politécnica e em outras instituições de ensino, por meio da tecnologia com o auxílio de meios interativos, em que professores e alunos possam participar de maneira colaborativa no processo de ensino e aprendizagem, gerando materiais didáticos e pedagógicos, que possam ser combinados, acessados independentemente de sua plataforma, garantindo interoperabilidade e reutilização.

1.3 Objetivo

Pesquisar e propor uma metodologia de adaptação do COL, desenvolvido pelo Laboratório de Arquitetura e Redes de Computação da Escola Politécnica da Universidade de São Paulo, em relação aos objetos de aprendizagem, seguindo uma padronização internacional para garantir interoperabilidade, reusabilidade, acessibilidade e armazenamento dos mesmos. Essa metodologia abrange a elaboração de um modelo de arquitetura cliente servidor, ajustando a atual base de dados do COL ao padrão SCORM, desenvolvendo do lado cliente um programa (API – Application Program Interface) que disponibiliza uma comunicação dos objetos de aprendizagem com o servidor. Também é desenvolvido um protótipo da implementação da API do lado servidor para atender as requisições do cliente.

1.4 Apresentação do trabalho

O trabalho é composto por seis capítulos e dois anexos.

No primeiro capítulo, descreve-se a contextualização, o objetivo e a justificativa do trabalho. No capítulo dois, têm-se algumas definições de objetos de aprendizagem, suas principais características e exemplos.

O capítulo três descreve modelos de padronização de objetos de aprendizagem. No quarto capítulo tem-se o conceito de LMS (Learning Management System) e o sistema de aprendizagem COL.

O capítulo cinco estabelece a metodologia de adaptação do COL ao modelo de padronização SCORM em relação aos objetos de aprendizagem.

O sexto capítulo narra as conclusões, aperfeiçoamentos e sugestões para a continuidade do trabalho.

No Anexo A, tem-se uma lista dos elementos do modelo de dados do SCORM com exemplos de uso.

O Anexo B define a implementação de vários programas para complementar a arquitetura elaborada e um modelo do arquivo de manifesto (imsmanifest.xml).

Capítulo 2

Objetos de Aprendizagem

2.1 Conceitos

Não existe uma definição de consenso entre os autores que pesquisam sobre objetos de aprendizagem.

Segundo LTSC (2002), um objeto de aprendizagem é qualquer entidade, digital ou não, que pode ser usada e re-utilizada para aprendizagem, educação e treinamento. Essa definição é generalista, pois a partir dela qualquer elemento educacional pode ser considerado um objeto de aprendizagem, como um livro ou um documento produzido por meio de um editor de texto.

Wiley (2004), apresenta a definição de LO como sendo qualquer recurso digital que possa ser reutilizado para apoio ao aprendizado. A principal idéia dos LO é quebrar o conteúdo educacional em pequenos blocos que possam ser reutilizados em diferentes ambientes de aprendizagem. Como exemplo, pode-se ter uma disciplina chamada Editor de Texto, que ensina como formatar textos. A estrutura do curso está representada na figura 2.1.

Editor de Texto	
Módulo 1 – Informações sobre a disciplina	**Módulo 3 – Formatação básica**
1.1 Objetivo	3.1 Fontes
1.2 Conteúdo programático	3.2 Parágrafos
1.3 Referências bibliográficas	
1.4 Pré-requisitos	**Módulo 4 – Formatação avançada**
1.5 Sistema de avaliação	4.1 Molduras
	4.2 Numeração de páginas
Módulo 2 – Seleção de texto	4.3 Figuras
2.1 Teclado	4.4 Tabelas
2.2 Mouse	

Fig. 2.1 Estrutura da disciplina de Editor de Texto.

Cada item de um módulo pode ter um ou mais LO. No módulo 3 da disciplina, item 3.1, Fontes, pode-se ter mais de um objeto, pois formatar fontes, implica na alteração de cor, tamanho, tipo de letras, etc., nos quais podem ser criados mais de um LO.

Para Sosteric; Hesemeier (2002), um LO é um arquivo digital (imagem, filme, texto, etc.) que pode ser utilizado para fins pedagógicos que incluem sugestões sobre o contexto apropriado para sua utilização, tais como: autor, título, palavra-chave, conteúdo, data de criação, formato, etc. Segundo os próprios autores, esta definição é incompleta, mas é um ponto de partida para a definição dos LO.

Como não há uma concordância sobre a definição de objetos de aprendizagem, para este trabalho será adotada a seguinte, fundamentada nas definições anteriores: um objeto de aprendizagem é uma entidade digital, baseado em conteúdo, que pode ser re-utilizado em ambientes educacionais e de aprendizagem, independentemente da plataforma de hardware ou do sistema operacional dos usuários, garantindo interoperabilidade, modularidade e facilidade no seu acesso, armazenamento e classificação.

2.2 Principais características

Independentemente da concordância do conceito de objetos de aprendizagem, eles apresentam as seguintes características, conforme Araújo (2003), EDUWORKS (2004) e Longmire (2000):

- **Reusabilidade**: como um objeto de aprendizagem pode ser utilizado em diversos contextos, seu uso pode ser estendido para muitas disciplinas com conteúdos semelhantes.

- **Interoperabilidade**: um objeto de aprendizagem deve ser capaz de ser utilizado em diversos tipos de hardware, sistemas operacionais, navegadores ou em outros ambientes de aprendizagem, proporcionando independência de plataforma.

- **Modularidade:** um conteúdo de aprendizagem, que pode ser uma disciplina, por exemplo, pode ser quebrado em pedaços ou blocos de aprendizagem, correspondendo aos LO, permitindo um aprendizado maior, por ter um objetivo menor, comparando-se ao conteúdo de uma disciplina. Cada pedaço ou módulo deve ser capaz de se comunicar com o sistema de aprendizagem, usando para isso um modelo de padronização, que será discutido no capítulo três. Os pedaços devem ter uma descrição que permitam procurar e encontrar o bloco correto para seu trabalho.

- **Facilidade de pesquisa, atualização e gerenciamento**: é possível obter informações, atualizar ou gerenciar objetos de aprendizagem por técnicas de seleção e filtragem, obtendo o conteúdo desejado e relevante para o devido contexto de utilização.

- **Acessibilidade:** corresponde à possibilidade de acessar ou usar os objetos de aprendizagem em um local remoto e utilizá-los em muitos outros locais.

Além dessas características, segundo Bettio (2003), os objetos de aprendizagem podem ser divididos em três partes:

- **Objetivo:** esta parte do objeto de aprendizagem tem como finalidade demonstrar ao aluno o que ele poderá aprender a partir do estudo desse objeto.

- **Conteúdo instrucional:** corresponde ao material didático necessário, para que no término, o aluno possa atingir os objetivos.

- **Prática e feedback:** possibilidade do aluno verificar se o seu desempenho atingiu o objetivo. Caso não, o aprendiz pode ter a liberdade de voltar e utilizar novamente o objeto.

Nesta divisão, podem ser incluídas outras partes ou categorias como: nome do LO, data de criação, autor, relação com outras disciplinas, habilidades e conhecimentos exigidos para a sua utilização, etc.

Conforme Campos (2003), os objetos de aprendizagem têm sido apontados pela literatura como uma possível solução para os problemas de redução de tempo e custo de desenvolvimento de conteúdo. Hoje, existem vários repositórios de objetos de aprendizagem.

O interesse na utilização de repositórios de objetos de aprendizagem engloba desde universidades ou instituições que oferecem cursos on-line até empresas que possuem departamentos de treinamento.

Estes repositórios são vistos como facilitadores na montagem de cursos on-line e devem possuir alguns requisitos, segundo Campos (2003), tais como:

- **Armazenamento de conteúdos de LO** - Conteúdos de LO dizem respeito ao material propriamente dito. Seu armazenamento e acesso devem ser oferecidos de maneira segura, eficiente e padronizada.

- **Armazenamento de dados sobre LO** – Os objetos de aprendizagem podem armazenar diversos dados, como seu nome, autor, data de criação, objetivo, etc. Essas informações podem ser classificadas, a fim de facilitar seu acesso.

- **Integração com sistemas de LMS** - Sistemas de gerenciamento de aprendizagem (Learning Management System), como o COL, são responsáveis pela disponibilização de conteúdo educacional on-line. É importante que o repositório de LO ofereça uma interface para carga ou acesso de seu conteúdo aos LMS.

- **Segurança** - Repositórios devem prover acesso a seu conteúdo mediante uma autorização ou autenticação do usuário.

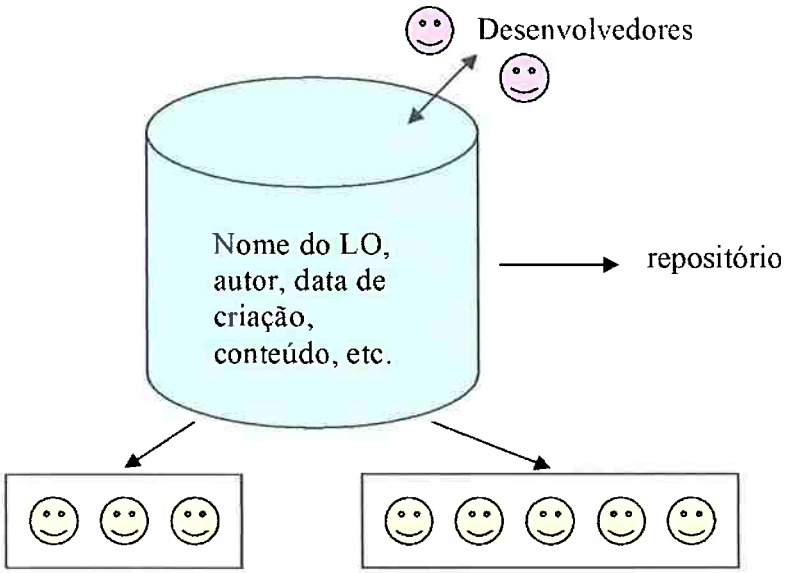

Fig. 2.2 Repositório de LO.

Os desenvolvedores podem criar os LO, armazenar, acessar e modificar os mesmos no repositório. Uma vez criados e armazenados, os LO podem ser acessados por várias pessoas em diversos cursos, desde que tenham a devida permissão.

2.3 Exemplos de LO

Seguem-se alguns exemplos de LO, de acordo com UNIVERSIABRASIL (2004):

- Exemplo 1

Nome do objeto: Minigramática eletrônica do português brasileiro.

Data de criação: 03/05/2004.

Autor: Universidade de São Paulo – São Carlos.

Objetivo: com a Minigramática espera-se que o usuário tenha, via Internet, um recurso de aprendizagem do português escrito. Com essa proposta, o indivíduo é levado a aprender sozinho as regras da língua escrita, uma vez que se trata de um ambiente de navegação (hipertexto).

Conteúdo instrucional: normas gramaticais do português escrito. Alguns tópicos relevantes: crase, colocação pronominal, concordância (nominal e verbal), acentuação, pontuação, etc.

Prática, feedback: a Minigramática pode ser utilizada por professores com seus alunos em uma atividade em grupo. A partir de um texto redigido, parte-se dos erros cometidos pelos alunos em busca de auto-aprendizado das regras do português escrito. Nesse caso, ao invés de material impresso, o aluno é estimulado a buscar o tema na Minigramática, aprofundando-se no problema por meio dos links que a página em questão lhe ofereceu.

Relação com outras disciplinas: a Minigramática é de uso multidisciplinar, uma vez que, em várias disciplinas, os alunos são levados a produzir textos.

Habilidades e outros conhecimentos exigidos: conhecimento sobre o funcionamento da Internet.

- Exemplo 2

Nome do objeto: Esforços internos em vigas.

Data de criação: 05/07/2004.

Autor: Instituto de Ensino Superior de Brasília – IESB.

Objetivo: espera-se que o usuário consiga analisar os esforços internos em elementos submetidos a carregamentos transversais. Ou seja, o usuário deve ser capaz de entender o que acontece em uma viga e qual é o comportamento das principais variáveis ao longo desta viga. O usuário terá que responder perguntas como: em qual ponto a viga vai quebrar e o que se faz para ela não quebrar.

Conteúdo instrucional: são abordados equilíbrio de um ponto material, equilíbrio de um corpo rígido, esforços internos, diagramas de força cisalhante e momento fletor, cálculo diferencial e integral e problemas de otimização.

Prática, feedback: busca-se fornecer as ferramentas para que o aluno escolha a seu tempo o grau de conhecimento que deseja obter. Coloca-se a disposição do estudante a teoria básica do tema de aula proposto juntamente com exemplos ou exercícios interativos para a aplicação da teoria. Completados esses dois passos: teoria e exercícios interativos, o usuário toma a decisão de participar de fóruns, ler mais sobre o assunto (capítulos de livros com exercícios para download), revisar assuntos anteriores ou aprofundar na matéria (artigos científicos). Os arquivos introduzidos foram estudados de maneira que o aluno decida o percurso a ser feito para aprender o conteúdo.

Relação com outras disciplinas: o objeto pode ser utilizado em disciplinas de várias áreas da Engenharia, tais como: Civil, Naval, Mecânica, etc.

Habilidades e outros conhecimentos exigidos: para uma completa compreensão de todos os elementos é necessário um conhecimento básico na utilização da Internet, em matemática (cálculo) e física (mecânica-estática).

2.4 LOM (Learning Object Metadata)

Para que os LO possam ser utilizados, eles precisam ser encontrados. Não é uma tarefa fácil encontrar informações em um meio com muitos dados, como a Internet. Uma possível solução é armazenar não somente os conteúdos dos LO, mas também sua descrição. Nessa descrição, incluem-se informações como título, autor, assunto, palavras-chave, data de criação, formato, tamanho, requisitos técnicos e educacionais, etc., segundo EDUWORKS (2004).

Conforme Anido et al (2002), para descrever essas informações utilizam-se metadados educacionais (LOM), definidos como sendo dados sobre dados. Eles fornecem informações sobre a acessibilidade e a organização dos LO, possibilitando a classificação e a indexação por meio de uma das informações escolhidas.

Os principais propósitos dos LOM, de acordo com Steinacker; Ghavam; Steinmetz (2001), são:

- permitir que alunos e professores procurem, avaliem, acessem e utilizem os LO.

- permitir o compartilhamento dos LO por meio de qualquer sistema de gerenciamento de aprendizagem.

- permitir que desenvolvedores de LO possam usar blocos de aprendizagem, combinando-os e criando novos LO.

- permitir a documentação dos LO.

- prover recursos e modelos de padronização que suportem a classificação, indexação, pesquisa e o compartilhamento dos LO para que estes possuam um formato comum.

Portanto, para usar os LO na Internet, precisa-se de um formato padrão para que se possam encontrá-los e acessá-los.

Alguns modelos de padronização já foram criados para garantir que os LO possam ser localizados e utilizados, garantindo interoperabilidade e reusabilidade. Os modelos de padronização serão vistos no capítulo três, enfatizando-se os aspectos de como se pode unir ou agregar os mesmos, formando um curso e disponibilizando-os para os sistemas de gerenciamento de aprendizagem.

Capítulo 3

Modelos de Padronizações Aplicados aos LO

Para que se possa utilizar as características de reusabilidade e interoperabilidade, têm-se criado vários modelos ou padronizações para os LO no mundo, principalmente nos Estados Unidos.

Instituições e organizações como o Institute of Electrical and Electronics Engineers (IEEE) e o Departamento de Defesa norte-americano têm trabalhado nessa área para obter um consenso sobre protocolos, arquiteturas e modelos de dados para conteúdos educacionais.

Relatam-se aqui algumas instituições e organizações que criaram meios para padronizar os objetos de aprendizagem.

3.1 Alguns exemplos de padronização

3.1.1 AICC (Aviation Industry CBT Committee)

Segundo WEBAULA (2004), os esforços de padronizações mais antigos começaram na aviação em 1988, por meio de treinamento mediado por computador (Computer Based Training - CBT). O Aviation Industry CBT Committee estava estabelecido de maneira que montadoras como a Boeing, que compravam partes de aviões de vários fabricantes diferentes, poderiam ter certeza de que cada um dos cursos de treinamento de seus fornecedores trabalhava da mesma maneira que os outros.

As regras do padrão AICC foram aceitas por diversas empresas e indústrias, e seu selo de aprovação é buscado por algumas empresas de e-learning.

Um de seus padrões é voltado para o processo de agregação dos objetos de aprendizagem e sua disponibilização para os sistemas de gerenciamento de aprendizagem.

3.1.2 LTSC (Learning Technology Standards Committee Of The IEEE)

O Comitê de Padronização de Tecnologia Educacional (LTSC) do IEEE, desenvolve modelos, recomendações e guias para software, ferramentas e métodos de projeto para facilitar o desenvolvimento, implementação, manutenção e interoperabilidade de sistemas educacionais.

Esse comitê também define alguns padrões para o gerenciamento e execução dos LO.

Segundo Anido et al (2002), em 1997, o IEEE criou um documento base para o IEEE LOM, especificando um conjunto de atributos necessários para permitir que os LO pudessem ser criados, desenvolvidos, gerenciados, localizados, usados e avaliados. Estes atributos são título, autor, formato, etc. Eles permitem a catalogação e indexação dos LO, definidos na norma IEEE P1484, de acordo com LTSC (2004).

Dentre os atributos ou categorias do modelo, destacam-se:

- geral: agrupa informações gerais que descrevem o objeto, como título, endereço, idioma, autor, descrição, palavras-chave.

- ciclo de vida: reúne informações que descrevem as características relacionadas ao histórico e estado atual dos LO, como versão, data de criação ou alteração, entidades que contribuíram, etc.

- técnica: agrupa os requisitos e características técnicas, como formato (som, figura, etc.), tamanho, requisitos necessários para a visualização do objeto, tipo de tecnologia (sistema operacional, navegador) e duração.

- educacional: define as características educacionais e pedagógicas do objeto, como tipo de atividade (prática, teórica ou ambas), recurso de aprendizagem (exercício, exame, leitura, simulação, figura, diagrama, etc.), nível de interatividade indicando o grau que o aluno terá em participar de uma tarefa, influenciando seu estado (muito baixo, baixo, médio, alto, muito alto), faixa etária indicada, grau de dificuldade (muito fácil, fácil, médio, difícil, muito difícil), etc.

- direito: especifica os direitos de propriedade e as condições de uso do objeto.

O processo da IEEE é aberto para todos que desejam participar e os interessados podem acessar por meio de listas de discussão pela Internet.

3.1.3 IMS (Instructional Management System)

O IMS é um consórcio governamental, que reúne um grupo de padrões abertos que permite, de acordo com Sonntag (2004):

- o compartilhamento de conteúdos educacionais entre cursos, promovendo a reutilização, interoperabilidade e modularidade. Cada módulo de um curso pode se comunicar com o sistema de aprendizagem.

- a localização e uso de LO, por meio de metadados.

- a criação de um empacotamento padrão dos LO para tornar possível seu acesso. Para o IMS, um curso é um conjunto de pequenos objetos ou unidades, que agregados em forma de árvore (hierárquica) podem ser navegados até o nível desses mesmos objetos. Os níveis de agregação a utilizar são da responsabilidade do criador.

- o gerenciamento do desenvolvimento do aluno com relação aos cursos que ele freqüenta, criando um modelo de dados para tal.

3.1.4 ADL (Advanced Distributed Learning)

O governo dos Estados Unidos, por meio do seu Departamento de Defesa, uniu-se à indústria de tecnologia, em meados de 1997 para iniciar o movimento pela adoção de um padrão único para os sistemas de educação on-line, visando interoperabilidade e reusabilidade. No início de 1999, o ADL, espécie de consórcio de pesquisa, criado pelo Departamento de Defesa dos Estados Unidos e pelo Bureau de Tecnologia e Ciência, começou a desenvolver o projeto de padronização SCORM (Sharable Content Object Reference Model), reunindo alguns modelos já disponíveis no mercado. O ADL não competiu com outros esforços de padronização. Pelo contrário, buscou incorporar o trabalho já realizado pelo AICC, IMS e IEEE em conjunto. Desde então, o padrão SCORM vem passando por evoluções e aprimoramento, de acordo, WEBAULA (2004).

Segundo ADL (2004), a primeira versão oficial (versão 1.0) foi lançada em janeiro de 2000. Posteriormente, em janeiro de 2001, essa versão sofreu algumas alterações e aperfeiçoamentos, baseados nos feedbacks dos usuários, dando origem à versão 1.1. Em outubro de 2001, surgiu a versão 1.2.

Atualmente, o SCORM encontra-se na versão 1.3 (SCORM 2004), lançada em janeiro de 2004.

As normas do padrão são coleções de especificações que permitem interoperabilidade, acessibilidade e reutilização de conteúdos educacionais (objetos de aprendizagem) via Web.

Segundo Lucena (2003), surgiu com o intuito de ser um modelo de especificações para desenvolvimento, empacotamento e entrega de conteúdos educacionais e de treinamento.

O padrão SCORM recomenda que várias regras sejam seguidas quando se desenvolvem os LO. Cada LO é uma entidade autônoma. Ela não depende de outro LO

para funcionar, e não se refere especificamente a outro LO. O LO desconhece totalmente seu contexto: não precisa saber de que curso faz parte, em qual módulo está e se há outros LO como ele no curso.

Por ser um padrão importante no contexto do trabalho, será feita a seguir uma descrição mais detalhada deste padrão.

3.2 Características do modelo SCORM

Um dos principais objetivos do SCORM é empacotar os conteúdos educacionais ou de ensino, integrando os repositórios e sistemas de gerenciamento de aprendizagem.

O modelo provê o conteúdo e a classificação dos LO, como seu nome, autor, formato, data, etc., armazenando em um repositório para acesso e alteração, definindo um modelo de empacotamento dos LO e estabelecendo um meio de comunicação entre os LO e o sistema de gerenciamento de aprendizagem.

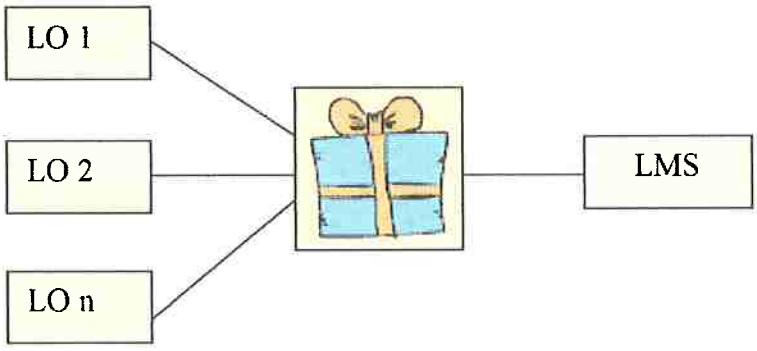

Fig. 3.1 Objetos de aprendizagem no SCORM, seu empacotamento e disponibilização para os LMS.

Cada conteúdo individual dos LO pode ser agregado a outros. Estes são empacotados criando o módulo de um curso, por exemplo, sendo acessado por um LMS (Learning Management System).

Os principais componentes do SCORM, segundo ADL (2004) e Tarouco (2003), são: modelo de agregação de conteúdo (Content Aggregation Model) e ambiente de

execução (Run-Time Environment). O modelo de agregação de conteúdo define a forma como os conteúdos de ensino devem ser criados e agrupados para que outros sistemas possam utilizá-los. Já o ambiente de execução define a forma como o LMS disponibiliza os LO e como é que estes podem se comunicar com o LMS.

3.2.1 Modelo de Agregação de Conteúdo

O modelo de agregação de conteúdo é composto pelo modelo de conteúdo (Content Model), metadado (Metadata) e empacotamento de conteúdo (Content Packaging), de acordo com ADL (2004).

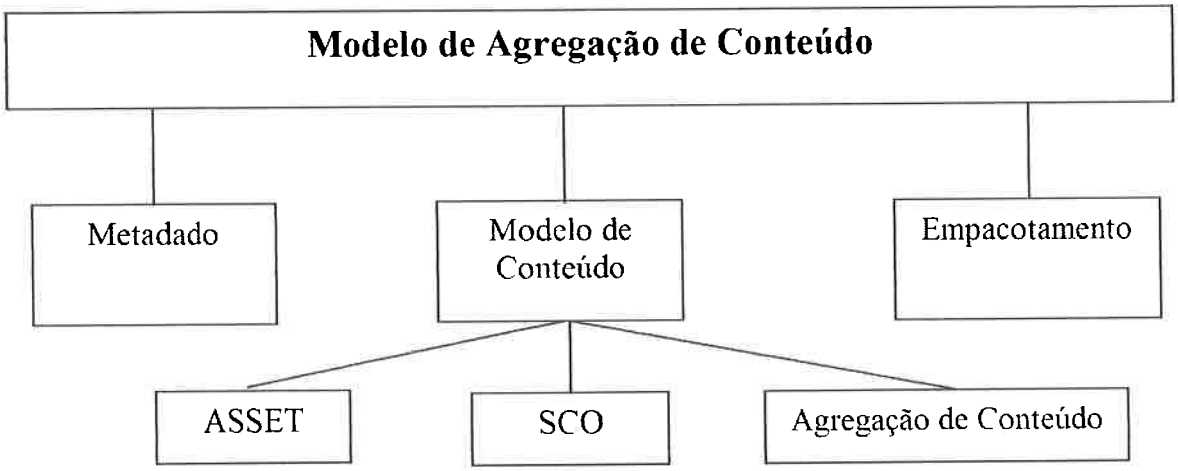

Fig. 3.2 Estrutura do Modelo de Agregação de Conteúdo do SCORM.

3.2.1.1 Modelo de conteúdo (Content Model)

Os componentes desse modelo são: partes de conteúdo (Assets), objetos compartilháveis de conteúdos (SCO) e agregação de conteúdo (Content Aggregation).

As partes de conteúdo (Assets) correspondem à representação eletrônica de dados (textos, imagens, sons, páginas Web, etc.) em diversos formatos como: TXT, JPEG, WAV, HTML, entre outros.

Os objetos compartilháveis de conteúdos (SCO) são conjuntos de um ou mais Assets. Embora os LMS possam apresentar os arquivos (Assets), estes devem ser inseridos em SCO para que os LMS possam acompanhar a evolução dos alunos por meio dos dados passados pelo SCO. Os SCO correspondem aos objetos de aprendizagem.

Os SCO devem ser representados em pequenas unidades de ensino para potenciar a sua reutilização em diferentes contextos de ensino. Quanto menor for o SCO mais facilmente poderá ser reintegrado em um outro contexto.

A agregação de conteúdo (Content Aggregation) é o processo de agregação de recursos (SCO e Assets) em uma unidade de instrução (curso, disciplina, módulo, etc.), criando um pacote de conteúdos.

É essa parte que define a seqüência pelo qual os conteúdos de ensino serão apresentados aos alunos. Cabe ao LMS interpretar a seqüência descrita e controlar a navegação.

A seqüência, como o próprio nome indica, cuida do comportamento do curso e a ordem de módulos que deve ser seguida para um determinado curso.

A navegação, por outro lado, permite que eventos de navegação possam ser iniciados, gerando a movimentação entre os módulos de aprendizagem.

Para tanto, o LMS deve prover um menu de escolha ou navegação para os módulos desejados dentro de um curso, dependendo do tipo de navegação projetada.

O SCORM estabelece uma estrutura hierárquica em forma de árvore para a seqüência e navegação de cursos.

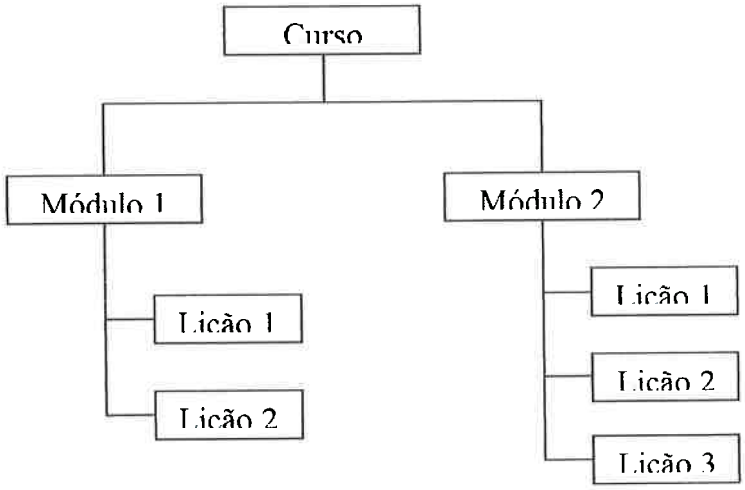

Fig. 3.3 Estrutura hierárquica do modelo de seqüência e navegação do SCORM.

Somente quando o aluno completar todas as lições de um módulo é que ele vai para o módulo seguinte, se assim for estipulado.

3.2.1.2 Metadado (Metadata)

Mecanismo para descrever e documentar os conteúdos educacionais, facilitando a busca e a seleção, por meio do autor, conteúdo, data, etc. Metadados fornecem os meios para acesso e recuperação de objetos em repositórios.

3.2.1.3 Empacotamento de conteúdo (Content Packaging)

Define o agrupamento dos conteúdos para permitir que sejam instalados em ambientes diferentes, facilitando a interoperabilidade.

Os pacotes, que representam uma unidade de ensino, são geralmente armazenados em um arquivo PIF (Package Interchange File) com o intuito de facilitar a sua distribuição pela Web. O PIF pode ter vários formatos, como por exemplo, ZIP, JAR, TAR, etc.

Os pacotes dividem-se em duas partes, que são: arquivo XML (Extensible Markup Language) de manifesto e arquivos físicos.

O arquivo XML de manifesto (imsmanifest.xml) é composto pela organização, recursos e metadados. Os recursos têm a função de indicar quais são os conteúdos educacionais que estão agrupados. A organização é responsável por apontar a estrutura dos conteúdos de ensino, dentro da uma determinada seqüência, elaborada na agregação de conteúdo. Metadados correspondem aos dados dos recursos educacionais armazenados, como descrito no item 3.1.2. No anexo B, tem-se uma estrutura simples do arquivo de manifesto.

Os arquivos físicos correspondem aos recursos de ensino na sua forma básica, tais como: vídeos, imagens, textos, etc.

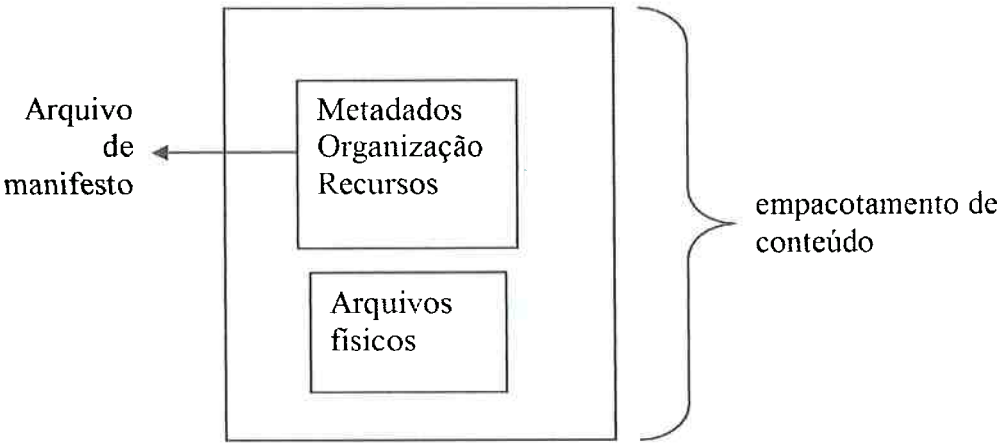

Fig. 3.4 Estrutura do empacotamento de conteúdo.

3.2.2 Ambiente de Execução

Segundo ADL (2004) e Barbeira; Santos (2004), o ambiente de execução tem como objetivo permitir que os conteúdos de ensino possam ser visualizados em diferentes LMS e tenham em todos o mesmo comportamento. Para que isto seja

possível, o ambiente de execução determina a forma como os SCO são enviados para o navegador (browser) e define o protocolo que os SCO e o LMS irão usar para se comunicarem entre si.

Este ambiente é composto pela Execução (Launch), Application Program Interface (API) e Modelo de dados (Data Model). Launch é responsável por lançar o SCO para o browser e efetuar operações para que o SCO possa se comunicar com o LMS. A API informará o atual estado do SCO (iniciado, em condição de erro, finalizado, etc.). O modelo de dados define um conjunto de informações referentes ao SCO que os LMS deverão usar.

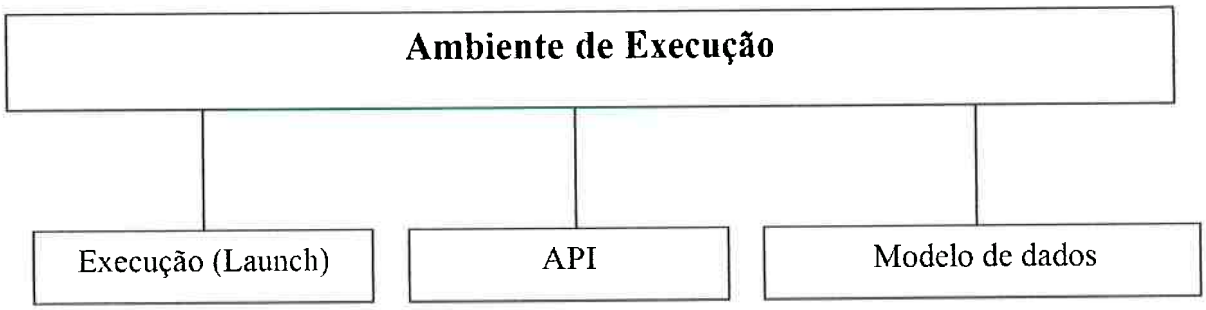

Fig. 3.5 Estrutura do Ambiente de Execução do SCORM.

3.2.2.1 Launch

Launch é um mecanismo comum para todos os LMS, responsável pelo envio dos recursos de ensino para o browser do usuário. A idéia de se criar um mecanismo comum para lançar os recursos de ensino tem por objetivo obter um comportamento igual em diferentes LMS independentemente da forma como os LMS estejam implementados.

Como já foi descrito, cabe ao LMS controlar a seqüência e a navegação entre diferentes recursos de ensino a partir da estrutura definida no Content Aggregation Model.

O LMS também baseia-se na interpretação de pré-requisitos presentes nos conteúdos de ensino, se estes existirem, para criar a seqüência pela qual os conteúdos serão enviados para o usuário.

Assim, a progressão por meio dos conteúdos de ensino, em um determinado curso ou contexto de ensino, pode ser feita de diferentes maneiras de acordo com o modelo de navegação que se quer construir. Essa construção, como também o envio de recursos de ensino para o browser fica a cargo do LMS.

As formas de navegações podem ser:

- seqüencial: o aluno navega progressivamente pelos recursos de ensino.

- escolhida pelo aluno: este escolhe a ordem dos recursos de ensino, selecionando-os por meio de um menu.

- adaptada: o LMS determina qual o conteúdo de ensino e a ordem que é entregue, conforme o desempenho do aluno.

Depois de selecionado o recurso de ensino a ser lançado, o LMS localiza-o por meio do URL (Uniform Resource Locator) descrito no Content Packaging e envia-o para o browser. O envio do recurso de ensino para o browser é realizado obrigatoriamente pelo protocolo http (HiperText Transfer Protocol).

Descreve-se resumidamente abaixo, o comportamento do LMS no Launch, para os Assets e para os SCO.

Assets

No lançamento de Assets, o único requisito a cumprir é que o Asset seja enviado para o browser por meio de http. Como o Asset não irá comunicar com o LMS não é necessário executar mais nenhuma operação, pois os Assets são arquivos físicos.

SCO

Um SCO só pode ser lançado pelo LMS. O LMS só pode lançar um SCO de cada vez e em cada momento apenas um SCO pode estar ativo. Para ser enviado para o browser, o LMS deve estabelecer uma comunicação com SCO, por meio da implementação de um programa denominado API Instance. Este programa usa as funções da API (Application Program Interface), descrito abaixo e é sempre fornecido pelo LMS, podendo ter implementações diferentes de LMS para LMS.

3.2.2.2 API (Application Program Interface)

É o mecanismo de comunicação que informa ao LMS o estado de um recurso educacional (iniciado, terminado, em condição de erro, etc.). A API pode ser considerada como um conjunto pré-definido de funções pelo SCORM, no qual o SCO tem a sua disposição.

As funções disponibilizadas pela API são:

- de sessão: estabelecem o início e o fim de uma sessão de estudo do aluno. Correspondem às funções: Initialize() e Terminate().

- de suporte: usadas para auxiliar na comunicação entre SCO e LMS se algum erro ocorrer. São usadas as funções GetLastError(), GetErrorString() e GetDiagnost().

- de transferência de dados: são usadas para passar dados do LMS para o SCO e vice-versa. As funções utilizadas são GetValue(), SetValue() e Commit().

Segue-se uma breve descrição de cada uma dessas funções:

- **Initialize(parameter)**: esta função indica ao LMS que o SCO pretende se comunicar. O SCO deve executar obrigatoriamente esta função antes de chamar qualquer outra. Retorna dois valores: true, se a comunicação foi bem sucedida ou false, se não houve sucesso.

- **Terminate(parameter)**: o SCO deve chamar esta função quando já não precisar mais se comunicar com o LMS. Retorna os mesmos valores da função Initialize().

- **GetValue(parameter)**: por meio desta função o SCO pode pedir dados ao LMS. O SCO especifica os dados que pretende, passando o parâmetro necessário. Os parâmetros pertencem aos campos do modelo de dados, descritos no item 3.2.2.3.

Exemplo: return_value = GetValue("cmi.learner_name");

O valor devolvido é o nome do aluno.

- **SetValue(parameter_1, parameter_2)**: com esta função, o SCO pode enviar uma informação para o LMS. Conforme a implementação da API Instance, a informação pode ser enviada diretamente para o LMS ou armazenada em uma memória cache (memória mais rápida que a RAM) e enviada mais tarde.

Em parameter_1, especifica-se um campo do modelo de dados e em parameter_2, especifica-se a informação que se pretende armazenar.

Exemplo: return_value = SetValue("cmi_learner_name", "João da Silva");

- **Commit(parameter)**: esta função existe para garantir ao SCO que a informação enviada ao LMS por meio de uma chamada SetValue() será persistida, ou seja, armazenada em um dispositivo físico. Nas implementações da API Instance que usam memória cache, quando recebem a chamada desta função devem enviar para o LMS todos os dados que tenham em memória e que ainda não tenham sido persistidos na base de dados do LMS. Se a API estiver implementada de forma a persistir os dados no fim de cada chamada SetValue(), a função Commit() torna-se redundante, devendo portanto, existir sempre para garantir conformidade com a norma SCORM. Neste caso, pode ser implementada como uma função oca, ou seja, não executa nenhuma operação e retorna sempre o valor true.

- **GetLastError(parameter)**: tem a função de retornar o número do erro para uma determinada função utilizada. Existem vários valores de erro, tais como: 0 (não ocorreu nenhum erro), 201 (argumento inválido), etc. O SCORM define uma tabela de códigos de erros com suas respectivas descrições. Os códigos de erros vão de 100 a 499. Há também a possibilidade do desenvolvedor criar seus próprios códigos de erros. Esses códigos podem ir de 1000 até 65535, segundo ADL (2004).

- **GetErrorString(parameter)**: retorna a descrição textual do erro ocorrido.

Exemplo: return_value = GetErrorString("201");

- **GetDiagnostic(parameter)**: esta função é criada dentro de cada LMS e permite acessar descrições específicas para erros ocorridos. Por exemplo, um curso começa a ser executado e o LMS retorna o erro 101, que é General Exception. Isso não ajuda a descobrir o motivo do erro, apenas indica o erro ocorrido. O curso poderia chamar a função GetDiagnostic para informar ao usuário o erro e este poderia ainda informar ao administrador do curso que tal erro aconteceu quando ele executou uma determinada ação.

Depois que o LMS lançar o SCO para o browser invocando o mecanismo Launch, cabe ao SCO localizar a API Instance e iniciar a comunicação com o LMS.

A Fig. 3.6 representa um diagrama mostrando o estado de um SCO. Depois de lançado pelo LMS, o SCO deve localizar a API Instance previamente enviada para o browser pelo LMS.

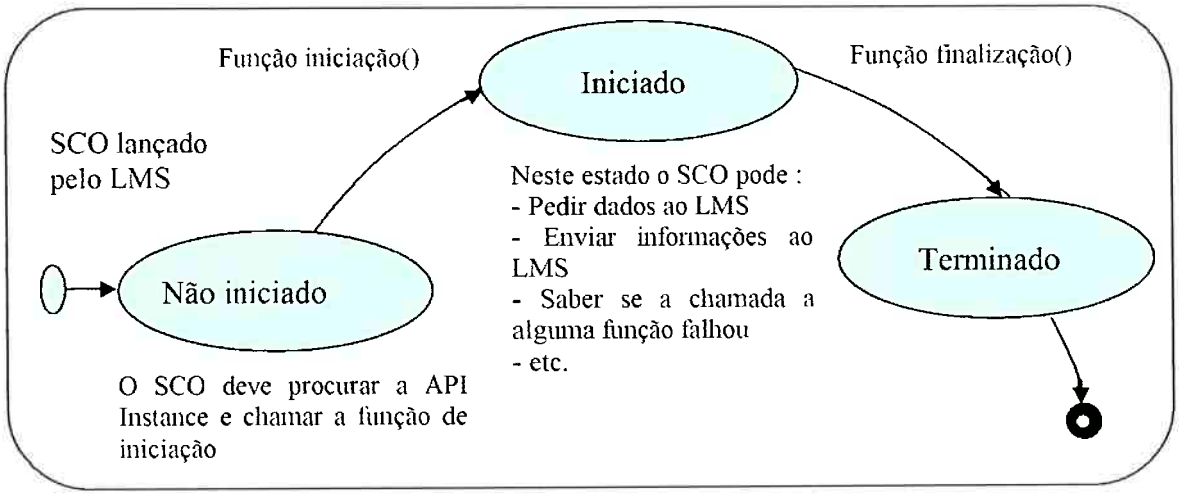

Fig. 3.6 Diagrama mostrando o estado de um SCO.

3.2.2.3 Modelo de dados (Data Model)

É um conjunto de elementos de dados padronizados pelo SCORM usados na comunicação do SCO com o LMS, estabelecendo um protocolo que irá permitir a comunicação entre ambos.

Para cada SCO é implementado um modelo de dados, isto é, cada SCO dispõe de campos específicos para armazenar seus dados. Os nomes dos campos do modelo de dados começam por "cmi" (Computer Manager Instruction).

Como na API Instance, a implementação fica por conta do LMS.

Abaixo, têm-se resumidamente os principais elementos ou campos do modelo de dados, segundo ADL (2004). No Anexo A, tem-se exemplos desses elementos descritos de maneira detalhada.

Identificação do aluno: cmi.learner_id.

Nome do aluno: cmi.learner_name.

Comentários do aluno: cmi.comments_from_learner.

Os elementos de dados contidos nessa parte do modelo, têm a função de coletar informações dos alunos sobre as experiências de aprendizagem para que depois se possa criar uma lista ou relatório desses comentários, avaliando o projeto e a estrutura do conteúdo.

Comentários do LMS: cmi.comments_from_lms.

Os elementos desse modelo de dados contém comentários realizados pelos desenvolvedores do SCO para serem visualizados pelos alunos.

Preferências: cmi.learner_preference.

Especificam as preferências do aluno com relação ao som, linguagem, etc.

Estado do SCO: cmi.completion_status.

Indica se o aluno completou ou não o SCO.

Sucesso: cmi.success_status.

Indica se o aluno obteve ou não sucesso (desempenho apropriado) utilizando o SCO.

Entrada: cmi.entry.

Aponta se o aluno já visitou ou acessou o SCO.

Saída: cmi.exit.

Apresenta o motivo ou a forma como o aluno abandonou a sessão ou o SCO.

Suspensão do SCO: cmi.suspend_data.

Durante uma sessão de aprendizagem, o aluno ou o SCO pode desejar suspender o uso do SCO e armazenar o ponto no qual ele foi suspenso. O campo cmi.suspend_data armazena e recupera os dados necessários.

Pontuação: cmi.score.

É o elemento do modelo de dados que corresponde à pontuação ou nota do aluno.

Tempo de Sessão: cmi.session_time.

Especifica o tempo que o aluno usou na sessão corrente do SCO.

Tempo total: cmi.total_time.

O valor de cmi.total_time define o tempo total utilizado pelo aluno em um dado SCO.

Tempo máximo permitido: cmi.max_time_allowed.

É o tempo total máximo permitido para um aluno usar o SCO.

Ação do limite de tempo do SCO: cmi.time_limit_action.

Indica o que o SCO pode fazer quando o tempo limite de uso exceder (cmi.max_time_allowed).

Credito: cmi.credit

Indica se o aluno tem crédito para executar o SCO. Há dois estados para este campo: credit e no_credit.

Modos de apresentação do SCO: cmi.mode.

Identifica os modos possíveis no qual o SCO pode ser apresentado ao aluno.

Interação: cmi.interactions.

Define um conjunto de respostas do aluno quando se tem questões ou tarefas que o professor quer registrar.

Objetivos do SCO: cmi.objectives.

Identifica o desenvolvimento de cada aluno relativamente aos objetivos de um determinado SCO. Inclui informações sobre os objetivos de cada SCO e o estado em que o aluno se encontra a esses objetivos.

3.3 Padronização do SCORM

O SCORM reuniu várias padronizações existentes, originados do AICC, IMS e LTSC, conforme já foi descrito. Tem-se na figura 3.7, os resultados das padronizações.

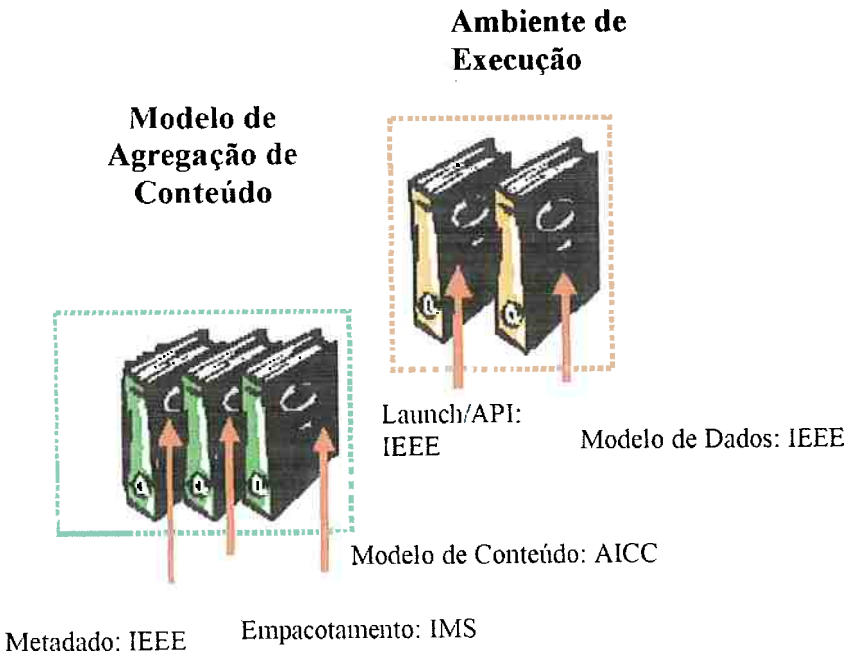

Fig. 3.7 Padronização do SCORM.

O modelo de agregação de conteúdo resultou do IEEE para o padrão de metadados, o empacotamento resultou do IMS e o modelo de conteúdo originou do AICC.

Para o ambiente de execução, o modelo de dados, Launch e API precederam do IEEE.

3.4 Fechamento do Capítulo

As normas do modelo SCORM estabelecem padrões para criar os LO ou os módulos de um curso, empacotando os mesmos, definidos no modelo de agregação de conteúdo.

Uma vez criados, é necessário executar os LO, lançando-os no browser para os alunos acessarem dentro de um sistema de gerenciamento de aprendizagem. Devem ser respeitados dentro do SCORM o modelo de dados e as respectivas chamadas dos elementos, por meio da API, que é realizado dentro do ambiente de execução.

Descreve-se, portanto, no capítulo cinco, os elementos ou campos que devem ser acrescentados no COL para respeitar o modelo de dados, como também um modelo de implementação das funções da API.

No capítulo 4, descrevem-se os conceitos de sistemas de gerenciamento de aprendizagem e o funcionamento atual do COL.

Capítulo 4

Sistemas de Gerenciamento de Aprendizagem (LMS)

4.1 Conceitos

Segundo Zaina (2002), os sistemas de gerenciamento de aprendizagem foram criados a partir das necessidades de centralizar as informações referentes a cursos pela Internet. Eles reúnem várias ferramentas para o docente criar e gerenciar um curso a distância e interagir com o aluno, tais como: bate-papo, e-mail, fórum, acompanhamento de avaliações do curso e do aluno, criação e gerenciamento de conteúdo educacional, etc.

Pode-se definir um LMS como um software que automatiza a administração de cursos on-line. O LMS entre outras funções, registra usuários, controla cursos e grava dados de alunos, de acordo com Lucena (2003).

Para ADL (2004) o termo LMS no SCORM implica em um serviço no qual os conteúdos de aprendizagem são gerenciados e liberados aos estudantes. Em outras palavras, no SCORM o LMS autoriza quando e quais conteúdos devem ser permitidos aos alunos utilizarem, mostrando o progresso e como os estudantes se desenvolvem durante o processo de aprendizagem.

O LMS não é apenas utilizado por alunos que irão acessar um curso e utilizar suas ferramentas de interação. O ambiente também é utilizado por diversos usuários, como tutores, professores e administradores que estarão criando novos cursos e escolhendo as ferramentas adequadas para cada tipo específico de disciplina ou módulo.

Genericamente, tem-se o seguinte mapeamento das tarefas realizadas pelos alunos, professores, tutores e administradores dentro do LMS:

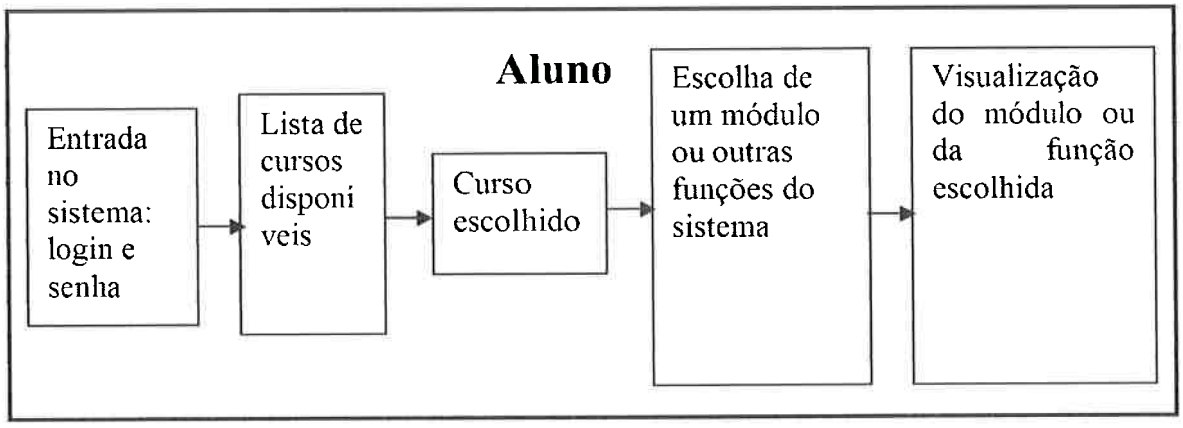

Fig. 4.1 Tarefas realizadas pelos alunos dentro de um LMS.

O aluno acessa o sistema de gerenciamento de aprendizagem com seu login (nome do usuário) e senha. Se aceitos, são apresentadas, entre outras informações, uma lista com os cursos disponíveis para ele cursar ou para fazer sua matrícula. O aluno pode escolher o curso e módulo desejado ou outra função oferecida pelo sistema para a sua visualização.

Caso o aluno ainda não tenha uma senha poderá solicitar ao administrador.

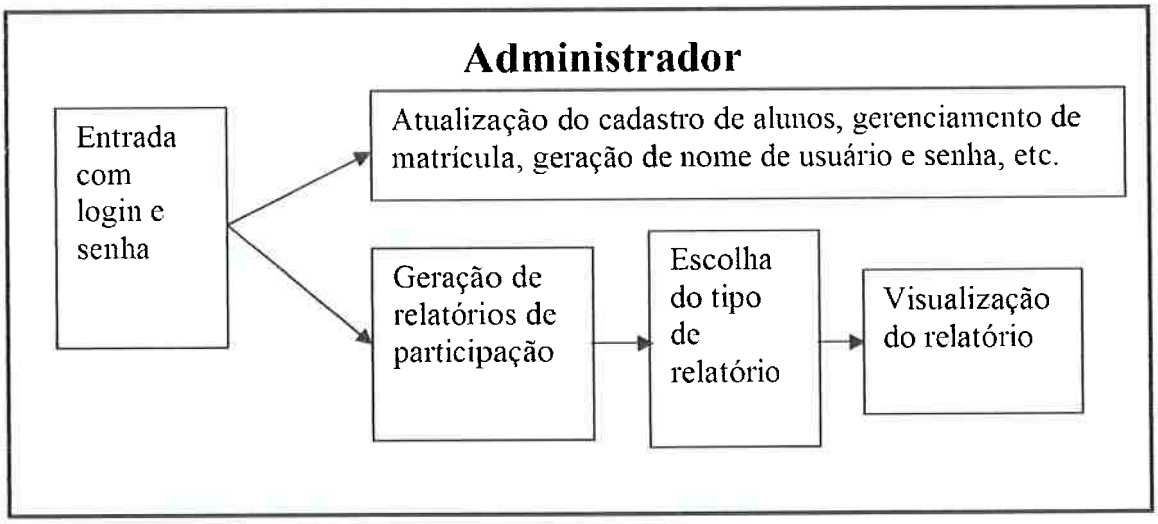

Fig. 4.2 Tarefas realizadas pelos administradores dentro de um LMS.

O administrador também faz sua entrada no sistema. No sistema, ele pode atualizar o banco de dados dos alunos (incluir, alterar, excluir, consultar), administrar matrículas, gerar nomes de usuários e senhas, etc. ou pode gerar relatórios de participação do aluno em um dado curso ou módulo, etc.

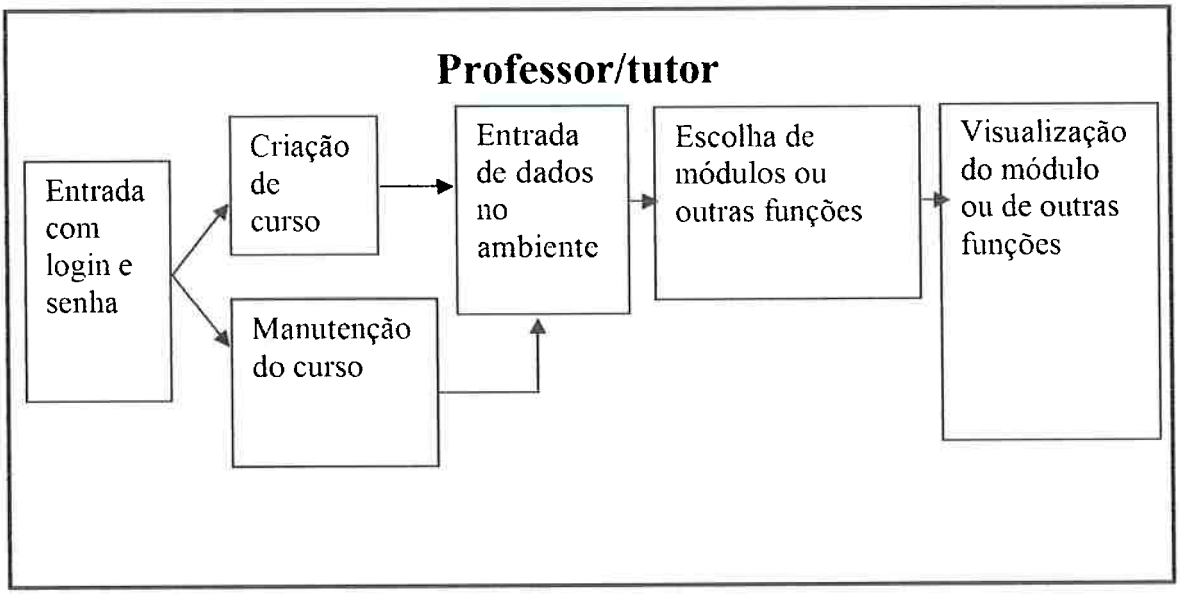

Fig. 4.3 Tarefas realizadas pelos professores e tutores dentro de um LMS.

A tarefa do professor e do tutor engloba a criação ou a manutenção do curso. Há necessidade de gerar os dados de entradas necessários, escolher os objetos de aprendizagem ou outras funcionalidades que o curso irá ter. Poderá também verificar o desempenho do aluno em um curso ou módulo, etc.

O mapeamento das funções para cada usuário é flexível, pois pode variar da implementação de cada LMS e da própria instituição de ensino. Foi descrita acima uma visão mais comum das tarefas realizadas.

Existem vários sistemas de gerenciamento disponíveis no mercado, sendo que todos têm a preocupação de gerenciar o conteúdo de cursos e disponibilizar ferramentas que permitam a interação entre alunos e professores. Este conteúdo pode ser um arquivo na forma de texto, imagem, som, filme, etc., segundo Castro (2001).

Neste trabalho será discutido o sistema de gerenciamento COL, desenvolvido pela Universidade de São Paulo.

4.2 Sistema de Gerenciamento de Aprendizagem COL

O COL é um sistema gerenciador de aprendizagem, desenvolvido pelo LARC, (Laboratório de Arquitetura de Redes de Computadores), do Departamento de Engenharia de Computação e Sistemas Digitais, da Escola Politécnica da Universidade de São Paulo.

O desenvolvimento do sistema surgiu da necessidade de disponibilizar material on-line para diversos cursos, centralizando o gerenciamento dos mesmos, de acordo com Zaina (2002).

É um sistema que se propõe a hospedar cursos baseados em páginas HTML e recursos multimídia, conforme Araújo (2003). O material desenvolvido pelo docente é independente da ferramenta COL. Este material pode ser constituído por vídeo, animação, texto, imagem, etc., sendo que o sistema gerencia este conteúdo que será disponibilizado ao aluno.

Possui várias ferramentas, tais como: e-mail, fórum, bate-papo, agenda, material on-line, testes que podem ser aplicados aos estudantes, gerenciamento de dúvidas dos alunos, atividades, etc. Uma atividade é o local onde o professor define uma tarefa extra aos alunos, como por exemplo, uma leitura de um texto que estará disponibilizado aos mesmos ou uma área para que os alunos coloquem seus arquivos para outros alunos. Para esta ferramenta, pode ser atribuída uma nota e um peso, como também para o fórum, bate-papo e teste (prova).

O COL encontra-se atualmente na versão 3.2, lançada em agosto de 2005.

Quando um usuário entra no sistema é requisitado seu nome de usuário (login) e senha, conforme a figura 4.4. Se válidos, o aluno pode requisitar matrícula ou acessar as ferramentas disponíveis pelo sistema.

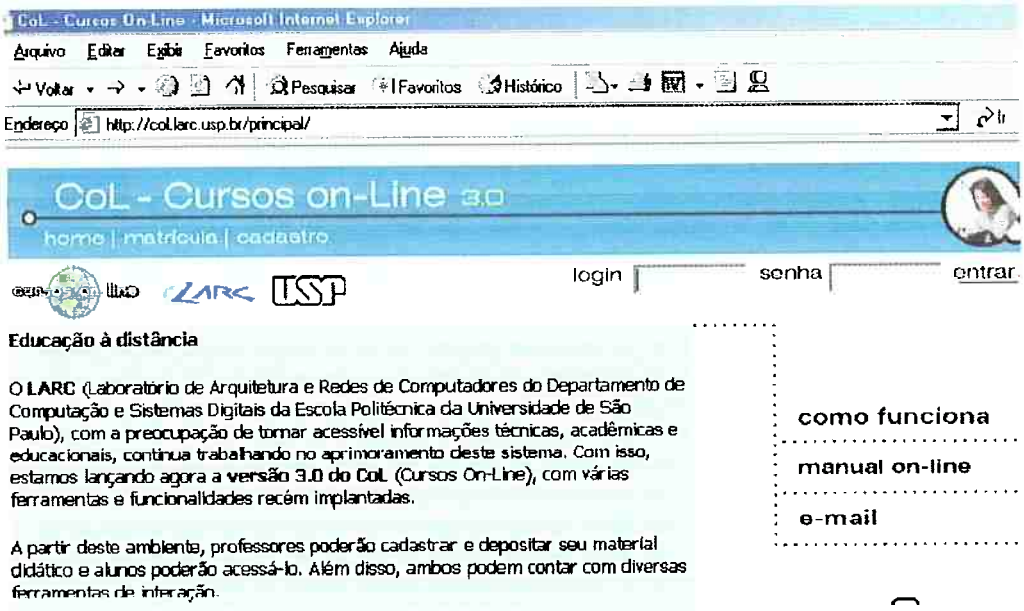

Fig. 4.4 Tela de abertura do COL.

Um curso no COL é formado por módulos, disciplinas e turmas. Uma turma pode ter várias disciplinas e uma disciplina pode ter vários módulos. Um módulo representa uma unidade de ensino. Um módulo pode ser formado por uma ou mais páginas HTML e pode conter qualquer tipo de arquivo, como vídeo, imagem, texto, etc., que deve estar disponibilizado para os alunos.

Como exemplo, tem-se um curso de Pós-graduação em Educação. Como turma poderia se considerar POS-EDU. Como uma das disciplinas, Tecnologias para Educação Virtual Interativa. Os módulos dessa disciplina poderiam ser: conceitos de educação a distância, jogos para educação, ambiente multimídia, objetos de aprendizagem, etc.

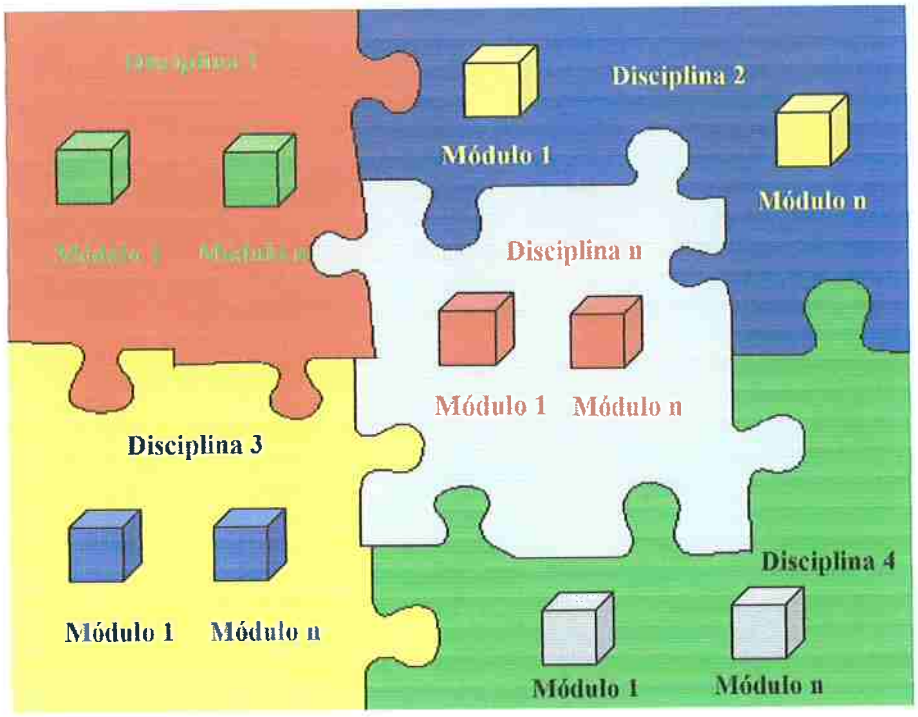

Fig. 4.5 Modelo do sistema COL: estrutura das disciplinas e módulos, segundo Silveira et al (2004).

Para disponibilizar um novo curso pelo COL, deve-se proceder da seguinte forma:

- **Criar módulos**. Um módulo deve ser formado por uma ou mais páginas HTML ligadas. A ligação dessas páginas é realizada por meio de links do HTML que precisam ser criados. O professor pode elaborar suas páginas utilizando um editor HTML, usando qualquer tipo de arquivo, desde que estes arquivos sejam copiados para o servidor que gerencia o COL. O upload do conteúdo do módulo é realizado por uma ferramenta que está integrada ao gerenciador de cursos e tem como finalidade copiar para o servidor que hospeda o COL o conteúdo de um módulo do curso. Sendo assim, um módulo pode ser utilizado em várias disciplinas diferentes.

- **Criar disciplinas**. Uma disciplina pode ser composta por um ou mais módulos. O cadastro de disciplinas não depende do cadastro de módulos, porém é necessário que

os módulos de uma disciplina estejam previamente cadastrados para que estes sejam associados a ela.

- **Atribuir os módulos às disciplinas**. Para atribuir os módulos a uma disciplina, é preciso selecionar os módulos que devem fazer parte dela e determinar como esses módulos se relacionam em termos de seqüência e de pré-requisito. Isto determina como o aluno vai cursar a disciplina, ou seja, a cada momento o aluno poderá cursar um ou mais módulos dependendo dos módulos que ele já fez (e foi aprovado em seu teste). Um módulo pode ter vários pré-requisitos e pode ser pré-requisito de vários módulos.

- **Criar uma turma**. Assim como no caso do cadastro da disciplina, o cadastro de uma turma não está vinculado ao cadastro de uma disciplina. Porém, para associar uma disciplina a uma turma, é necessário que a disciplina já esteja cadastrada.

- **Atribuir disciplinas à turma**. Uma turma deve ter uma ou mais disciplinas associadas a ela, caso contrário não ficará disponível para matrícula.

- **Associar um grupo de usuários à turma**. Uma turma deve ser associada à pelo menos um grupo de usuários (alunos de graduação, pós-graduação, etc.) para que ela possa ser visualizada.

A ordem da criação desses processos (módulos, disciplinas, turmas e grupos) não é importante. A associação entre os itens é que determina seu correto funcionamento.

Deve-se também incluir os alunos e suas respectivas matrículas. A inclusão de alunos e de matrículas poderá ser realizada pelo professor, administrador ou aluno.

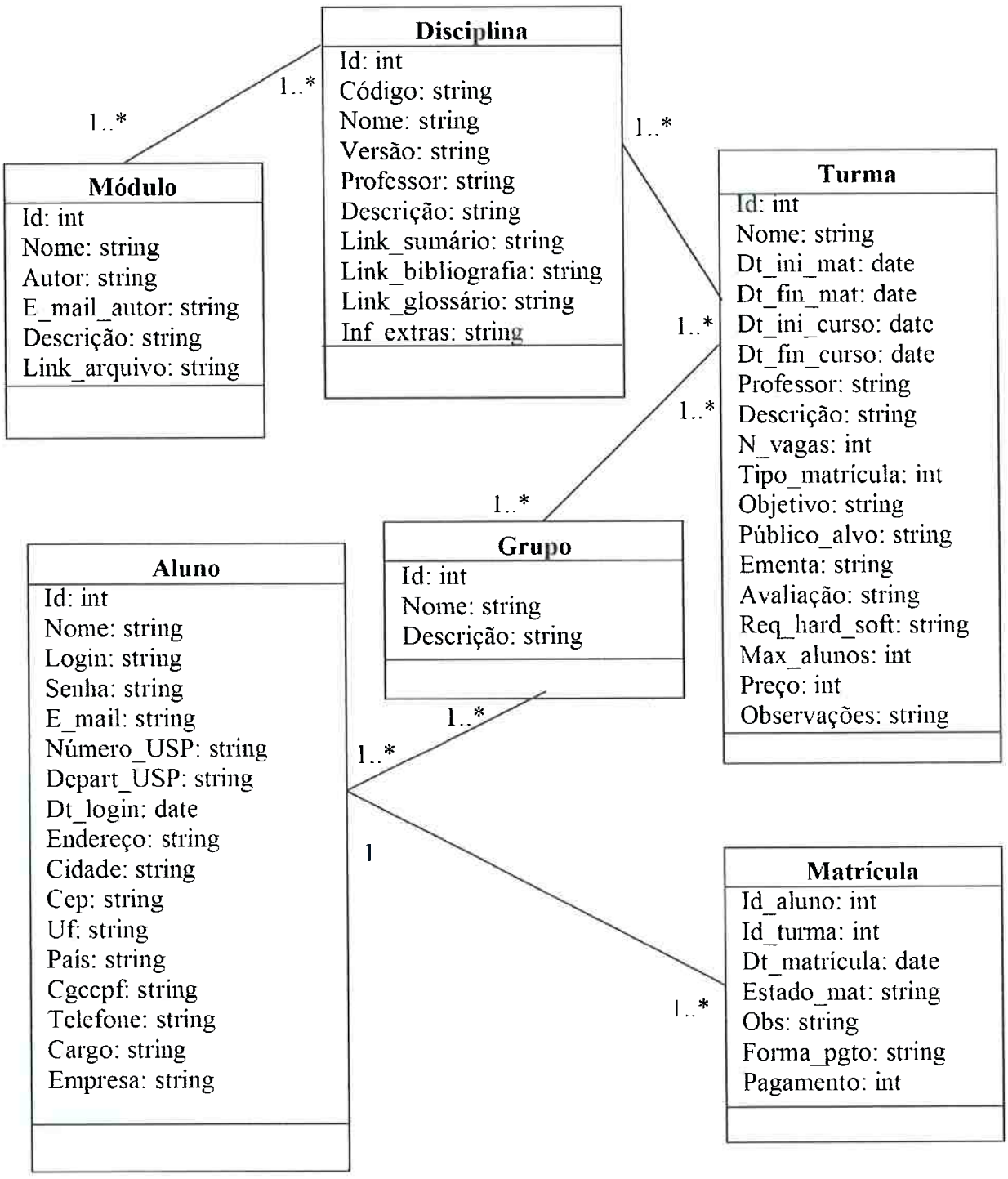

Fig. 4.6 Diagrama de classe parcial do COL.

Além do COL gerenciar o curso (módulos, disciplinas, turmas, alunos e matrículas), ele também permite a atribuição de notas aos testes dos módulos, da disciplina (prova com todo o conteúdo da matéria), ao fórum, ao bate-papo e às atividades. Para cada ferramenta o professor pode definir um respectivo peso.

Para atribuir notas aos testes dos módulos e da disciplina, o professor deve cadastrar as questões de múltipla escolha e a(s) alternativa(s) correta(s). Uma característica é o sorteio aleatório de questões ao aluno quando o mesmo está respondendo o teste, fazendo com que ele não responda as mesmas questões, quando utilizar o teste mais de uma vez. A nota do estudante é gerada automaticamente.

Em cada questão, uma alternativa errada anula uma correta. Se existirem três alternativas corretas, cada uma valerá 33,39%. Se o aluno marcar três corretas e uma errada, ele terá 66,66 % do valor da questão.

O valor de cada questão está relacionado ao número total de questões do teste. Se um teste possuir oito questões, cada uma valerá 12,5% do total do teste.

As notas do bate-papo são atribuídas pelo próprio professor aos alunos.

As notas do fórum podem ser geradas pelo sistema, por meio de alguns requisitos. Esses requisitos devem ser preenchidos pelo professor em relação ao aluno na forma de respostas do tipo Sim/Não para cada mensagem, tais como: demonstrou interesse, demonstrou entendimento, demonstrou educação e contribuiu com a sessão (escala a ser escolhida pelo professor de 1 a 10). O professor pode visualizar os dias e horários em que cada aluno acessou o fórum.

Supondo que uma avaliação no fórum seja formada apenas por um item Sim/Não e um item de escala analítica de 1 a 10, o conceito é definido da seguinte da maneira:

caso o item Sim/Não receba o valor "Sim", este recebe 100% de aproveitamento (valor 100). Caso o item que utiliza uma escala analítica receba como valor atribuído 7, então este recebe 70% de aproveitamento (valor 70). Somando-se estes valores (170) e dividindo-o pela pontuação máxima (200), chega-se ao conceito: 170 ÷ 200 = 0,85 (ou seja, 85% de aproveitamento) que corresponderá a 8,5 de conceito.

A nota das atividades é atribuída pelo professor responsável.

O cálculo da média final é feito pela média ponderada dos pesos atribuídos a cada ferramenta:

$$\text{Média Final} = \frac{(TD.P1 + MTM.P2 + MAT.P3 + MFO.P4 + MCH.P5)}{(P1 + P2 + P3 + P4 + P5)}$$, em que:

TD = nota do teste da disciplina

MTM = média dos testes dos módulos

MAT = média das atividades

MFO = média dos fóruns

MCH = média dos chats

P1, P2, P3, P4 e P5 representam os respectivos pesos para cada ferramenta de avaliação.

A figura 4.7 representa o diagrama de classe parcial do COL em relação ao cadastro de questões. Cada questão está associada a um ou mais módulo e vice-versa.

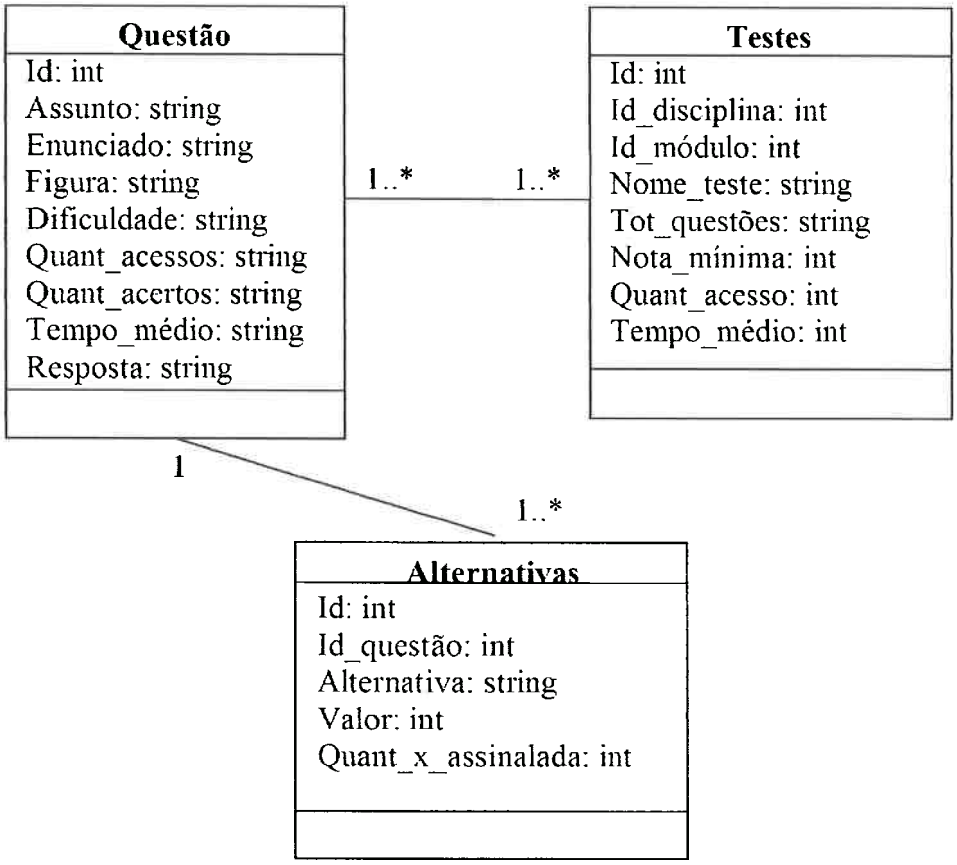

Fig. 4.7 Diagrama de classe parcial do COL em relação ao cadastro de questões.

Serão descritos, de acordo com Silveira et al (2004), os campos ou atributos de algumas classes do COL para que se possa no capítulo cinco estabelecer a relação desses campos ao modelo de dados do SCORM, isto é, será feita a verificação da existência ou não dos campos do COL ao modelo SCORM para garantir a padronização. A partir daí, será sugerido algumas mudanças no sistema COL, criando e alterando algumas de suas classes e atributos.

Eis algumas classes:

- **Módulo:** define os módulos para as diversas disciplinas.

Tabela 4.1 Classe Módulo

Nome	**Descrição**	**Tipo**
Id	Identificador do módulo	int
Nome	Nome do módulo	string
Autor	Autor ou criador do módulo	string
E_mail_autor	E-mail do autor	string
Descrição	Descrição textual do módulo	string
Link_arquivo	Corresponde ao endereço que contém o conteúdo do módulo	string

- **Grupo:** registro dos grupos.

Tabela 4.2 Classe Grupo

Nome	**Descrição**	**Tipo**
Id	Identificador do grupo	int
Nome	Nome do grupo	string
Descrição	Descrição do grupo	string

- **Disciplina:** descreve as disciplinas existentes nos cursos.

Tabela 4.3 Classe Disciplina

Nome	**Descrição**	**Tipo**
Id	Identificador da disciplina	int
Código	Código da disciplina. Cada curso tem um código específico dentro da instituição	string
Nome	Nome da disciplina	string
Versão	Versão da disciplina, ou seja, sua atualização	string
Professor	Nome do professor que ministra a disciplina	string
Descrição	Descrição da disciplina	string
Link_sumário	Endereço onde está o sumário da disciplina	string
Link_bibliografia	Endereço onde está a bibliografia da disciplina	string
Link_glossário	Endereço onde está o glossário da disciplina	string
Inf_extras	Informações adicionais sobre a disciplina	string

- **Discip_módulo:** associação da disciplina e seus respectivos módulos.

Tabela 4.4 Classe Discip_módulo

Nome	**Descrição**	**Tipo**
Id_disciplina	Identificador da disciplina	int
Id_módulo	Identificador do módulo	int

- **Turma:** representa a turma, ou seja, um determinado curso.

Tabela 4.5 Classe Turma

Nome	**Descrição**	**Tipo**
Id	Identificador da turma	int
Nome	Nome da turma	string
Dt_ini_mat, Dt_fin_mat	Data inicial e final de matrícula	date
Dt_ini_curso, Dt_fin_curso	Data de início e fim do curso	date
Professor	Nome do professor responsável pelo curso	string
Descrição	Descrição da turma	string
N_vagas	Número de vagas	int
Tipo_matrícula	Tipo de matrícula	int
Objetivo	Objetivo do curso	string
Publico_alvo	A quem se destina o curso	string
Ementa	Ementa do curso	string
Avaliação	Descrição de como serão as avaliações	string
Req_hard_soft	Requisitos de hardware e software	string
Max_alunos	Quantidade máxima de alunos por turma	int
Preço	Preço da matrícula	int
Observações	Comentários gerais	string

- **Tur_dis:** atribuição das disciplinas a uma determinada turma.

Tabela 4.6 Classe Tur-dis

Nome	**Descrição**	**Tipo**
Id_turma	Identificador da turma	int
Id_disciplina	Identificador da disciplina	int
Ordem	Seqüência no qual as disciplinas serão ministradas	int

- **Turmas_grupo:** associação das turmas a um grupo.

Tabela 4.7 Classe Turmas_grupo

Nome	**Descrição**	**Tipo**
Id_turma	Identificador da turma	int
Id_grupo	Identificador do grupo	int

- **Aluno:** cadastro de alunos.

Tabela 4.8 Classe Aluno

Nome	**Descrição**	**Tipo**
Id	Identificador do aluno	int
Nome	Nome do aluno	string
Login	Nome de usuário do aluno	string
Senha	Senha do aluno	string
E_mail	E-mail do aluno	string
Número_USP	Caso o aluno for da USP, seu número	string
Depart_USP	Departamento no qual o aluno está vinculado se for aluno da USP	string
Dt_login	Data do cadastro de senha	date
Endereço, Cidade, Cep, UF, Cgccpf, País, Telefone	Dados pessoais do aluno	string
Cargo, Empresa	Cargo e empresa do aluno, se este trabalha	string

- **Alunos_grupo:** associação dos alunos a um grupo de usuários.

Tabela 4.9 Classe Alunos-grupo

Nome	**Descrição**	**Tipo**
Id_aluno, Id_grupo	Identificadores do aluno e do grupo	int

- **Composição:** esta classe tem por objetivo acrescentar pré-requisito para que o aluno possa cursar corretamente a seqüência dos módulos.

Tabela 4.10 Classe composição

Nome	**Descrição**	**Tipo**
Id_disciplina, Id_módulo	Identificadores da disciplina e do módulo	int
Ordem	Indica a ordem do módulo para uma dada disciplina	int
Pai	Indica se há pré-requisito para se cursar um determinado módulo	string

- **Matrícula**: representa a matrícula dos alunos.

Tabela 4.11 Classe Matrícula

Nome	**Descrição**	**Tipo**
Id_aluno, Id_turma	Identificadores do aluno e da turma	int
Dt_matricula	Data de matrícula do aluno	date
Estado_mat	Estado da matrícula do aluno: regular, trancado, etc.	string
Obs	Observações gerais	string
Forma_pgto	Forma de pagamento da matrícula	string
Pagamento	Indica se o aluno efetuou ou não o pagamento, caso o curso for pago	int

- **Mf_critério:** atribuição de pesos ao teste da disciplina, dos módulos, bate-papo, fórum e atividade.

Tabela 4.12 Classe Mf_critério

Nome	**Descrição**	**Tipo**
Id_turma, Id_disciplina	Identificadores da turma e disciplina	int
Testedpeso	Peso atribuído ao teste da disciplina	real
Testesmpeso	Peso atribuído aos módulos da disciplina	real
Atividadespeso	Peso das atividades	real
Chatpeso	Peso aplicado ao bate-papo	real
Forumpeso	Peso dos fóruns	real

- **Questão:** cadastro de questões.

Tabela 4.13 Classe Questão

Nome	**Descrição**	**Tipo**
Id	Identificador da questão	int
Assunto	Assunto da questão	string
Enunciado	Enunciado da questão	string
Figura	Se a questão tiver figura, deve-se anexar o endereço	string
Dificuldade	Grau de dificuldade: fácil, intermediário ou difícil	string
Quant_acessos/ Quant_acertos	Quantidade de vezes que a questão foi acessada e quantidade de acertos relativos à questão	string
Tempo_médio	Tempo médio para se resolver a questão	string
Resposta	Corresponde a resposta correta da questão	string

- **Questão_módulo:** agregação da questão a um módulo.

Tabela 4.14 Classe Questão_módulo

Nome	**Descrição**	**Tipo**
Id_questão, Id_módulo	Identificadores da questão e do módulo	int

- **Testes:** produção do teste.

Tabela 4.15 Classe Teste

Nome	**Descrição**	**Tipo**
Id	Identificador do teste	int
Id_disciplina, Id_módulo	Identificadores da disciplina e do módulo	int
Nome_teste	Nome do teste	string
Tot_questões	Total de questões do teste	string
Nota_mínima	Nota mínima atribuída ao teste	int
Quant_acesso	Quantidade de vezes que o teste foi acessado	int
Tempo_médio	Tempo médio para se resolver o teste	int

- **Questão_teste:** ligação da questão com o teste.

Tabela 4.16 Classe Questão_teste

Nome	**Descrição**	**Tipo**
Id_questão, Id_teste	Identificadores da questão e do teste	int

- **Alternativas:** alternativas para cada questão.

Tabela 4.17 Classe Alternativas

Nome	Descrição	Tipo
Id	Identificador da alternativa	int
Id_questão	Identificador da questão	int
Alternativa	Descrição da alternativa	string
Valor	Valor atribuído para a respectiva alternativa	int
Quant_x_assinalada	Quantidade de vezes que a questão foi assinalada	int

- **Resp_alunos:** respostas do aluno para cada questão.

Tabela 4.18 Classe Resp_alunos

Nome	Descrição	Tipo
Id_aluno, Id_teste, Id_questão	Identificadores do aluno, do teste e da questão	int
Alternativa1, Alternativa2, Alternativa3, Alternativa4, Alternativa5	Indicação de qual alternativa o aluno assinalou como correta a resposta da questão	bit

Notas_teste: armazenamento das notas dos alunos nos testes, tanto para um módulo, como para toda uma disciplina.

Tabela 4.19 Classe Notas_teste

Nome	**Descrição**	**Tipo**
Id_teste, Id_turma, Id_aluno	Identificadores do teste, da turma e do aluno	int
Nota	Nota do aluno	real
Tempo_teste	Tempo que o aluno levou para realizar o teste	int
Data_hora	Data e horário que o aluno fez o teste	datetime

- **Tmp_estat:** armazena os dias e horários que o aluno acessou determinado módulo.

Tabela 4.20 Classe Tmp_estat

Nome	**Descrição**	**Tipo**
Id_aluno, Id_turma, Id_disciplina, Id_módulo	Identificadores do aluno, turma, disciplina e módulo	int
Data_hora	Data e hora de entrada do aluno no módulo	datetime
Permanência	Tempo utilizado pelo aluno nessa sessão de utilização	int

-**Mfnotas**: armazena a média final dos alunos para uma determinada disciplina.

Tabela 4.21 Classe Mfnotas

Nome	**Descrição**	**Tipo**
Id_aluno, Id_turma, Id_disciplina	Identificadores do aluno, da turma e da disciplina	int
Nota	Nota final do aluno	real

Todas essas classes e atributos do COL foram criadas utilizando o banco de dados SQL Server 2000 e o sistema foi desenvolvido em ASP (Active Server Pages), utilizando o protocolo de transporte http (HiperText Transfer Protocol).

Capítulo 5

Proposta de Adaptação do COL ao Padrão SCORM

O SCORM não define nenhum método ou tecnologia para a implementação do modelo de dados sendo esta da responsabilidade do LMS, tendo como única restrição a sua acessibilidade por meio das funções da API definido no Run-time Environment do SCORM.

O SCO controla certos elementos de dados, como por exemplo, cmi.session_time. O LMS também gerencia certos elementos de dados como cmi.time_limit_action, entre outros.

Assim sendo, as funções de leitura e gravação (GetValue e SetValue), se integram para controlar o fluxo de dados.

O LMS deve disponibilizar um modelo de dados coerente com a norma SCORM, isto é, deve permitir armazenar e acessar toda a informação desejada como é previsto no seu modelo. Para isto, a base de dados deve possuir uma estrutura adequada ao tipo de informação que se pretende guardar.

Já para os SCO, deve-se fornecer um programa (API Instance) capaz de implementar as funções da API do SCORM no lado cliente e mais uma outra API no lado servidor (LMS) com os recursos de comunicação com a base de dados necessários para que possa haver a troca de informações entre o SCO e o LMS em um modelo que garanta interoperabilidade.

Para isto, há necessidade de um estudo e análise de alguns componentes para a adaptação do COL ao modelo SCORM em relação ao SCO, tais como a base de dados e a implementação da API Instance no lado cliente.

Também será descrito um modelo de arquitetura da API no lado servidor.

A seguir relatam-se estes itens.

5.1 Base de Dados

A base de dados deverá estar armazenada no servidor e irá ter uma estrutura que permita guardar e acessar seus atributos ou campos de acordo com o modelo de dados da norma SCORM.

A arquitetura a ser implementada possui vários elementos e estes devem estar integrados de maneira que o sistema possa interpretá-los e responder às interações de cada usuário.

Esta arquitetura foi concebida ajustando as atuais bases de dados existentes no COL ao modelo de dados do SCORM, descrito no capítulo 3.

A seguir serão descritas as tabelas de dados (classes) reformuladas do COL, de acordo com o modelo de dados definido pelo SCORM.

A figura 5.1 mostra a relação das tabelas. Nessa figura, as classes e atributos representados em vermelho, correspondem aos elementos que ainda não existem e que serão incorporados ao sistema. Em preto, estão as classes e atributos já presentes no COL.

Os atributos ou elementos de dados que o SCORM define possuem um tipo apropriado de dados (inteiro, data, etc.). Neste trabalho, foram respeitados os tipos de dados, mantendo a integridade do modelo.

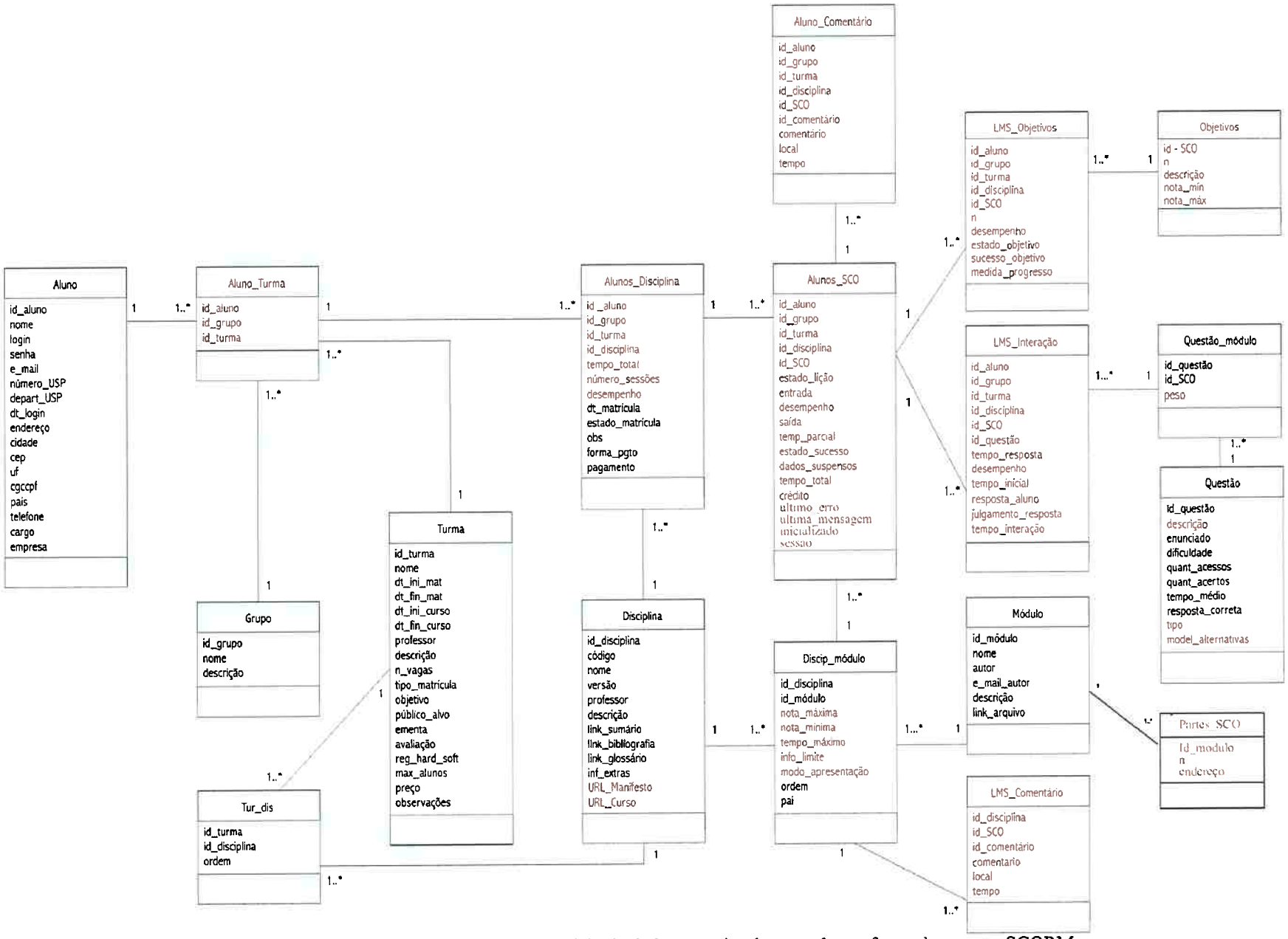

Fig. 5.1 Diagrama parcial das relações do modelo de dados a ser implementado conforme às normas SCORM.

A seguir, tem-se uma explicação detalhada da figura 5.1 com relação às classes e seus respectivos atributos.

As classes do COL Aluno, Grupo, Turma, Tur_dis, e Módulo não sofreram alterações.

- **Disciplina**: cadastro das disciplinas existentes nos cursos.

Além dos campos citados no capítulo quatro, acrescentam-se:

- URL_Manifesto e URL_Curso: endereço do manifesto e endereço dos arquivos físicos. Embora essas informações possam ser acessadas do próprio arquivo de manifesto, isso obrigaria o computador a procurar o arquivo a cada utilização do curso, no qual iria resultar em uma possível perda de eficiência no tempo de resposta do LMS. Optou-se, neste trabalho, por fazer a procura do Manifesto uma única vez, quando o curso é instalado no LMS, armazenando as informações extraídas do manifesto na base de dados correspondente. Estes dois atributos são do tipo string.

- **Aluno_Turma:** nova classe para controlar o grupo e a turma do aluno.

Tabela 5.1 Classe Aluno_Turma

Nome	**Descrição**	**Tipo**
id_aluno, id_grupo, id_turma	Identificadores do aluno, grupo e turma	int

- **Alunos_Disciplina:** relação dos alunos com as disciplinas.

Tabela 5.2 Classe Alunos_Disciplina

Nome	**Descrição**	**Tipo**
id_aluno, id_grupo, id_turma, id_disciplina	Identificadores do aluno, do grupo, da turma e da disciplina	int
tempo_total	Tempo total de acesso do aluno em uma determinada disciplina. O LMS irá calcular este tempo, baseado no tempo total de uso para cada SCO (módulo) para um determinado aluno	string
número_sessões	Número total de sessões do aluno em uma determinada disciplina. O LMS se responsabilizará em computar o valor deste campo, baseado no acesso do aluno na disciplina escolhida	int
desempenho	Desempenho do aluno em uma determinada disciplina. O LMS efetuará o cálculo da média do aluno para uma determinada disciplina	real

Os seguintes atributos do COL da classe Matrícula devem ser acrescentados na classe Alunos_Disciplina: dt_matrícula, estado_matrícula, obs, forma_pgto e pagamento. Portanto, a classe Matricula será eliminada.

- **Discip_módulo:** integração dos módulos com as disciplinas.

Tabela 5.3 Classe Discip_módulo

Nome	**Descrição**	**Tipo**
id_disciplina, id_módulo	Identificadores da disciplina e do módulo	int
nota_máxima	Nota máxima do SCO	real
nota_mínima	Nota mínima do SCO	real
tempo_máximo	Tempo máximo que o aluno dispõe para visualizar o SCO	string
info_limite	Armazena a informação sobre a forma como o LMS deve agir quando o tempo máximo de uso é excedido	string
modo_apresentação	Indica os modos no qual o SCO pode ser apresentado ao aluno, como no elemento de dados cmi.mode	string
ordem	Ordem do módulo para uma disciplina	int
pai	Indica se há pré-requisito para se cursar um módulo	string

Os atributos ordem e pai pertencem à classe Composição do COL. Esta classe deve ser excluída e seus campos ordem e pai devem ser inseridos na classe Discip_módulo. Esses campos permanecem para dar maior flexibilidade no sistema, pois o arquivo de manifesto pode transferir esses dados para a base de dados, sendo que o COL já faz o controle de seqüência e navegação dos LO.

- **Alunos_SCO**: relação dos alunos com os módulos.

Tabela 5.4 Classe Alunos_SCO

Nome	**Descrição**	**Tipo**
id_aluno, id_grupo, id_turma, id_disciplina, id_SCO	Identificadores do aluno, grupo, turma, disciplina e SCO	int
estado_lição	Corresponde ao estado da lição: completado, não completado, etc.	string
entrada	Indica se o aluno já visitou ou não o SCO	string
desempenho	Resultado da avaliação do aluno	real
saída	Indicação de como o aluno saiu do SCO	string
temp_parcial	Tempo gasto pelo aluno na última sessão de acesso (sessão corrente)	string
estado_sucesso	Indica se o aluno obteve desempenho apropriado com o SCO	string
dados_suspensos	Indicação da posição (local) do SCO, no qual o aluno suspendeu seu uso. O SCO é responsável por passar esse valor para o LMS	string
tempo_total	Tempo total de uso do SCO para um determinado aluno. É responsabilidade do LMS retornar o tempo total de uso	string
crédito	Verifica se o aluno tem crédito ou não para executar o SCO	string

Além desses campos , foram acrescentados mais quatro para controle de sessão, que são:

- ultimo_erro. Armazena caso houver, o código do último erro ocorrido na sessão.

- ultima_mensagem. Guarda uma mensagem ou um erro ocorrido durante a última sessão.

- inicializado. Indica se o SCO está iniciado ou não.

- sessão. Código da sessão gerada pelo servidor automaticamente para identificar cada usuário.

- **LMS_Comentário:** comentários gerais que o LMS pode fornecer aos alunos.

Tabela 5.5 Classe LMS_Comentário

Nome	**Descrição**	**Tipo**
id_disciplina, id_SCO	Identificadores da disciplina e do SCO	int
id_comentário	Identificador do comentário do LMS	string
comentário	Descrição do comentário do LMS	string
local	Indicação do local do comentário dentro de um SCO	string
tempo	Indicação de quando o comentário foi criado	datetime

Para todos esses elementos de dados, o SCORM não define um mecanismo para as devidas entradas ou valores.

- **Aluno_Comentário:** descreve os comentários dos alunos para um determinado SCO.

Tabela 5.6 Classe Aluno_Comentário

Nome	Descrição	Tipo
id_aluno, id_grupo, id_turma, id_disciplina, id_SCO	Identificador do aluno, grupo, turma, disciplina e SCO	int
id_comentário	Identificador do comentário do aluno	string
comentário	Descrição do comentário do aluno	string
local	Indicação do local do comentário	string
tempo	Indicação de quando o comentário foi criado	datetime

- **LMS_Objetivos:** relação do aluno com os objetivos de cada SCO.

Tabela 5.7 Classe LMS_Objectives

Nome	Descrição	Tipo
id_aluno, id_grupo, id_turma, id_disciplina, id_SCO	Identificação do aluno, grupo, turma, disciplina e SCO	int
n	Indica o número do objetivo do SCO	string
desempenho	Desempenho do aluno em um SCO	real
estado_objetivo	Estado da lição	string
sucesso_objetivo	Indica se o aluno compreendeu o objetivo	string
medida_progresso	Representa um número que mede o progresso do aluno	real

- **Objetivos:** define o objetivo de cada SCO.

Tabela 5.8 Classe Objetivos

Nome	**Descrição**	**Tipo**
id_SCO	Identificador do SCO	int
n	Especifica um determinado número do objetivo	string
descrição	Descrição do objetivo	string
nota_min	Nota mínima atribuída para um objetivo	real
nota_max	Nota máxima para um dado objetivo	real

- **Questão:** cadastro das questões ou interações.

Tabela 5.9 Classe Questão

Nome	**Descrição**	**Tipo**
id_questão	Identificador da questão	string
tipo	Tipo de questão, como por exemplo, verdadeiro/falso, múltipla escolha, etc.	string
descrição	Descrição da questão	string
model_alternativas	Modelo de alternativas para o aluno escolher, que irá depender do tipo de interação	string

Os atributos do COL enunciado, figura, dificuldade, quant_acessos, quant_acertos, tempo_médio e resposta_correta continuam presentes na classe Questão.

- **LMS_Interação:** ligação do aluno com as interações (provas, exames, etc.).

Tabela 5.10 Classe LMS_Interação

Nome	**Descrição**	**Tipo**
id_aluno, id_grupo, id_turma, id_disciplina, id_SCO	Identificação do aluno, grupo, turma, disciplina e SCO	int
id_questão	Indica o número da questão (interação)	string
tempo_resposta	Representa um ponto no tempo em que a interação do aluno terminou	datetime
desempenho	Desempenho do aluno	real
tempo_inicial	Representa um ponto no tempo em que a interação do aluno começou	datetime
resposta_aluno	Corresponde a resposta do aluno	string
julgamento_resposta	Indica se a resposta do aluno está correta	string
tempo_interação	Diferença entre o tempo de resposta e o tempo inicial	string

- **Questão_módulo**: associação da questão a um módulo ou SCO.

Tabela 5.11 Classe Questão_módulo

Nome	**Descrição**	**Tipo**
id_questão	Identificador da questão	string
id_SCO	Identificador do SCO	int
peso	Peso da interação	real

- **Preferências**: esta classe deve controlar certas preferências dos usuários. Os valores de seus atributos são válidos para todos os SCO na arquitetura proposta. O SCORM permite que as preferências sejam válidas para todos os SCO ou pode-se configurar separadamente cada um deles.

Tabela 5.12 Classe Preferência

Nome	**Descrição**	**Tipo**
áudio	Determina o nível de som	real
linguagem	Especifica a linguagem utilizada	string
velocidade_conteúdo	Indica a velocidade de conteúdo liberada	real

- **Partes_SCO**: indica a ordem de cada parte de um módulo, pois um módulo pode contém mais de uma página HTML.

Tabela 5.13 Classe Partes_SCO

Nome	**Descrição**	**Tipo**
id_modulo	Identificação do módulo	int
n	Ordem de cada parte do módulo	int
endereço	Endereço onde se encontra uma parte do módulo	string

A classe Alternativa deve ser excluída, pois não existe somente o tipo de interação de questões de múltiplas escolhas. O SCORM fornece vários tipos de questões ou interações.

Continua também a classe do COL, Mf_critério que atribui pesos à atividade, ao fórum, ao módulo e ao bate-papo. Deve ser excluído o atributo testedpeso, pois não há mais teste de disciplina. As provas (interações) serão aplicadas apenas aos módulos. O cálculo da média final do aluno, deve ser alterado, eliminando-se a nota do teste da disciplina.

Com as classes elaboradas, há necessidade de descrever como um SCO pede ou envia dados ao LMS, gerando o fluxo de dados para controlar o conteúdo dos atributos presentes nas classes.

5.2 Arquitetura cliente-servidor

Tal como acontece com o modelo de dados, a implementação da API Instance (API no lado cliente que estabelecerá a comunicação entre o LMS e o SCO, usando as funções do ambiente de execução) fica a cargo do LMS, tendo como obrigação a acessibilidade dessa API a partir de um navegador Web (browser) via Javascript, conforme as normas do SCORM. Para isto, cada LMS deve fornecer aos SCO uma API Instance que torne invisível o seu funcionamento interno (processos, funções, etc). A implementação da API Instance localiza-se no Anexo B.

A figura 5.2 mostra o funcionamento da API Instance e os pedidos do SCO.

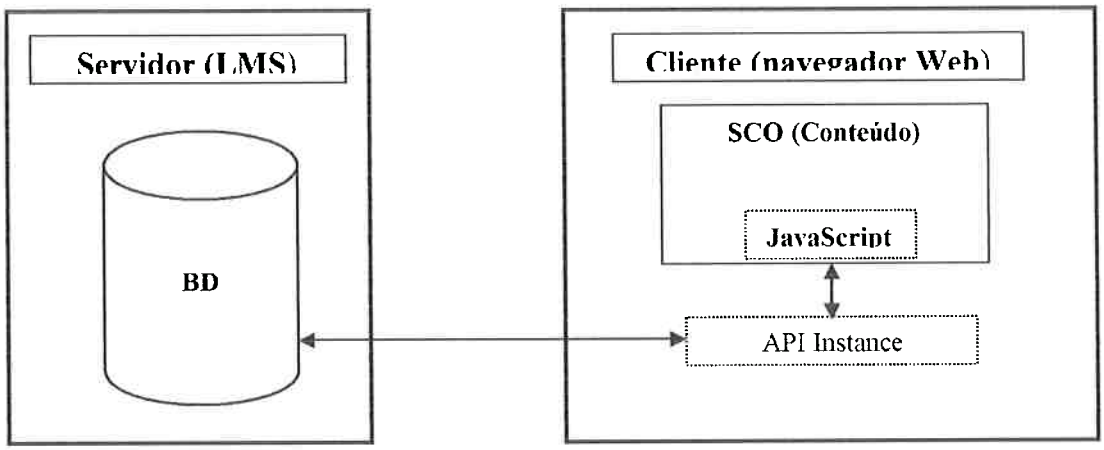

Fig. 5.2 Arquitetura da API Instance.

Com essa arquitetura, os processos são iniciados no lado do cliente, efetuando a comunicação com o servidor no momento de iniciar e terminar a sessão, quando o SCO pede e envia dados ao LMS, etc.

Desse modo, a cada solicitação do SCO, o servidor será chamado. O inconveniente dessa solução é a possível sobrecarga do servidor, porém as operações de chamadas são simples e transientes. A vantagem é que se houver uma falha na rede ou no servidor, grande parte das informações do cliente não serão totalmente perdidas, pois as atualizações serão feitas constantemente. Assim, os alunos não precisam fazer todas as operações novamente, caso aconteça algum problema com o servidor ou a rede.

Como existem chamadas remotas (cliente e servidor), as solicitações dos SCO podem ser atendidas de algumas maneiras. Neste trabalho, optou-se pela utilização do ASP, que o próprio COL já utiliza em suas rotinas. Esta parte corresponde à API no lado do servidor, no qual será implementada uma parte, pois sua codificação é fácil, porém extensa. A codificação está descrita no Anexo B.

Como já foi descrito, a API no lado cliente (API Instance) irá disponibilizar aos SCO as funções na norma SCORM, como Initialize(), Terminate(), GetValue(), SetValue(), Commit(), etc.

O LMS deverá então disponibilizar uma API no lado servidor para atender os pedidos do SCO. Isso significa que a API Instance terá que se comunicar com a API no lado servidor. Esta comunicação se dará por meio do protocolo de transporte http. Assim, a API Instance deverá disponibilizar de um lado uma interface acessível por JavaScript com as funções previstas no ambiente de execução do SCORM e de outro, uma interface de comunicação com a API no lado servidor.

Quando um SCO quiser se comunicar com o LMS irá chamar as funções da API Instance que por sua vez irá chamar, por meio do protocolo http, as funções da API no lado servidor. O resultado dessas funções será devolvido à API Instance por meio do protocolo http, que entregará ao SCO que desencadeou o processo de comunicação, como indica a figura 5.3:

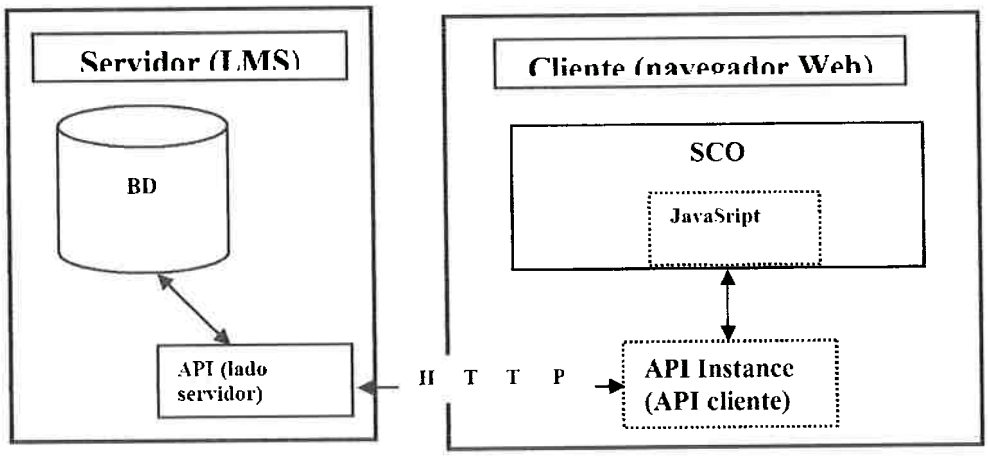

Fig. 5.3 Arquitetura da API Instance e da API no lado servidor.

5.2.1 Detalhando a API no lado cliente (API Instance)

A implementação da API Instance deverá ser codificada, segundo ADL (2004) como um objeto chamado "API_1484_11", criado em JavaScript. Neste objeto terão que estar todas as funções ou métodos do SCORM (Initialize, Terminate, etc.) e também as

funções de chamada para a API no lado servidor para que possa haver a comunicação entre a API Instance e a API no lado servidor.

Quando um SCO quiser por exemplo, chamar a função Initialize terá que localizar o objeto e usar o método Initialize desse objeto e se comunicar com a API no lado servidor.

Para um SCO encontrar a API Instance é necessário "navegar" em uma árvore hierárquica de dados para procurar o objeto "API_1484_11". A navegação é realizada por meio do DOM (Document Object Model).

Em linhas gerais, o DOM é um conjunto de especificações usadas para representar documentos XML na memória do computador, de acordo com Jorgensen (2002). Essa representação é realizada em forma de árvore, em que se tem uma estrutura hierárquica pai-filho.

O diagrama de uma árvore hipotética foi criado como mostra a figura 5.4 para exemplificar o conceito do DOM.

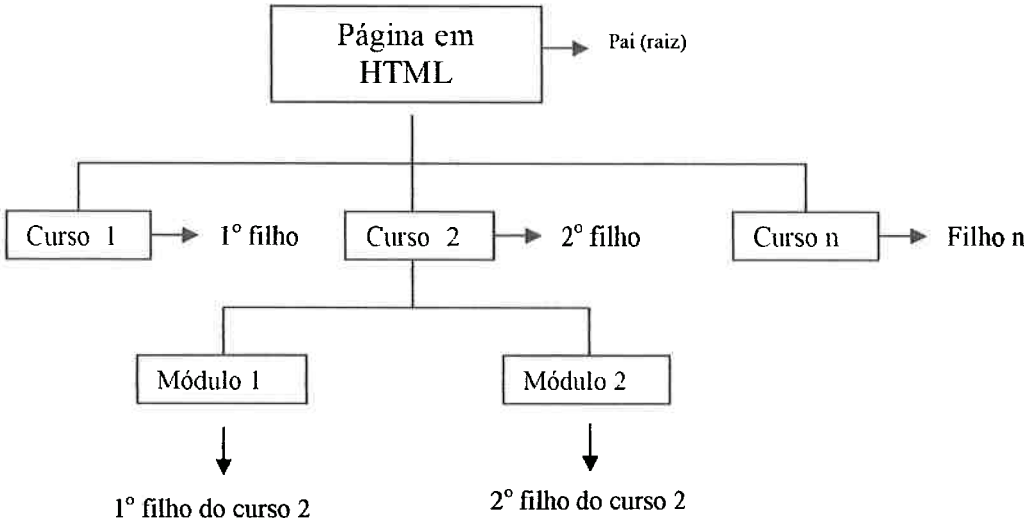

Fig. 5.4 Diagrama pai-filho de um documento DOM.

A API Instance deverá estar em uma janela especial. Essa janela deve ser colocada de maneira hierarquicamente superior à janela para onde o SCO será lançado, chamado de cadeia de pais (chain of parents). Dessa forma, os SCO poderão ser executados em uma estrutura em que a API Instance está inserida em um frame em HTML.

Por exemplo, uma janela de frame poderia ser dividida em duas. A primeira mostraria os módulos (SCO) existentes para o curso escolhido e a outra janela, seria a de visualização do conteúdo do SCO selecionado, conforme mostra a figura 5.5.

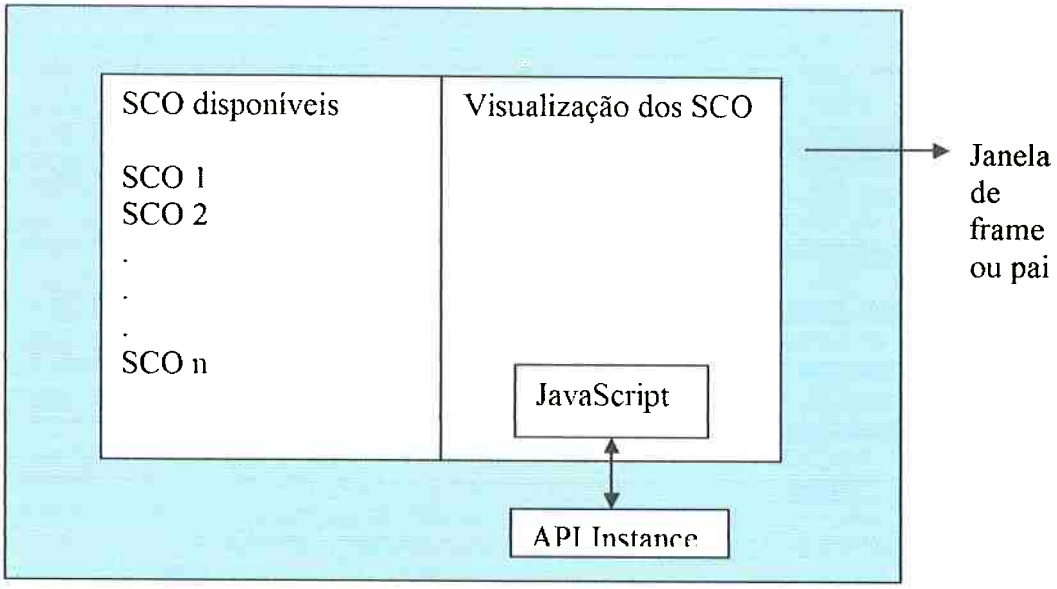

Fig. 5.5 Estrutura da janela de apresentação dos SCO e de seus conteúdos.

O algoritmo procurará pela API caminhando pela estrutura até encontrar ou até atingir um ponto onde não há outros frames pais.

Encontrada a API Instance, deve-se verificar as funções que os SCO exercem quando são chamados ou finalizados. É necessário, então, codificar as funções de cada um deles, dependendo do papel que cada SCO tem, fazendo a correlação com os elementos de dados.

Os elementos do modelo de dados que utilizam a função SetValue precisam ser calculados ou processados para que possam ser enviados para o LMS. Alguns desses elementos são os comentários realizados pelos alunos, preferência de som, tempo de sessão, resposta do aluno em relação a alguma questão ou interação, etc.

No Anexo B, encontra-se um exemplo de codificação de um SCO para se calcular o tempo de sessão, entre outras funções. Alguns elementos de dados não serão implementados vistos que a sua codificação é fácil, porém extensa porque existem muitos elementos de dados a serem testados.

5.2.2 Detalhando a API no lado servidor

A API no lado servidor será implementada, utilizando-se o ASP e o protocolo de comunicação http, no qual o COL já utiliza em seus procedimentos.

Serão descritos o ASP e também um resumo das funções para fazer a devida comunicação da API no lado servidor com a API Instance e vice-versa.

5.2.2.1 ASP

O ASP é uma linguagem que disponibiliza um conjunto de componentes para criar páginas dinâmicas e interativas na Internet. Tais páginas consistem em arquivos com a extensão .asp, em formato de texto, que contém combinações de tags (comandos) HTML e scripts (pequenos programas), ou seja, em uma página podem ter códigos escritos em HTML e em outras linguagens como JavaScript e VBScript, além da possibilidade de integração com XML (Extensible Markup Language).

Em páginas ASP, os scripts são executados no servidor Web e não no cliente (browser). É o próprio servidor que transforma os scripts existentes em uma página em HTML, fazendo com que qualquer browser seja capaz de acessar uma página que utilize ASP, segundo MICROSOFT (2005), garantindo assim a independência de plataforma.

Um servidor Web, que suporta ASP funciona da seguinte maneira:

Primeiramente o cliente solicita a visualização de uma página qualquer e o servidor abre a página e lê seu conteúdo. Se o servidor encontra tags HTML, ele envia os resultados direto ao cliente. Se encontra scripts, o servidor pára o envio e processa as instruções dos scripts e envia o resultado em HTML ao cliente, conforme a figura 5.6.

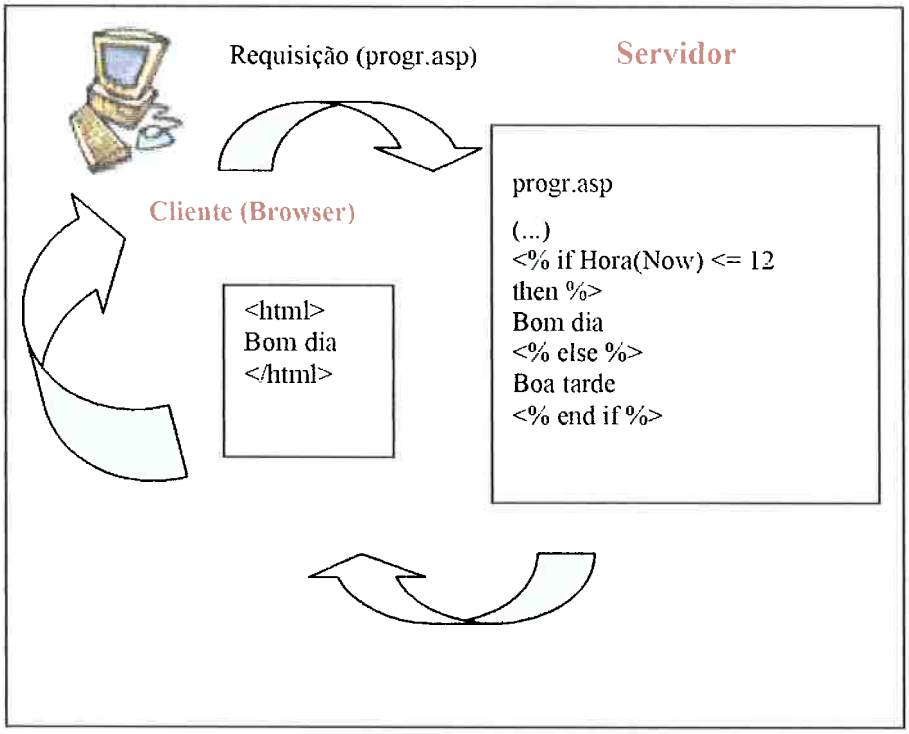

Fig. 5.6 Processamento de requisições em ASP.

O ASP possui uma série de características. Algumas delas, são descritas abaixo, de acordo com Martini (2002):

- Servidor Web: o ASP é compatível apenas com o IIS (Internet Information Services). O IIS é um conjunto integrado de serviços de rede para a plataforma Windows que permite publicar conteúdo e disponibilizar arquivos e aplicações em um ambiente Internet/Intranet. É integrado ao sistema operacional e possui uma interface para disponibilizar a hospedagem de web sites e o desenvolvimento de aplicações.

- Capacidade de integrar com vários bancos de dados: trabalha com quaisquer bancos de dados compatíveis com ODBC (Open DataBase Connectivity) que permite prover um padrão para conexão a bancos de dados, com a finalidade de tornar possível

acessar qualquer dado de qualquer aplicação independentemente do banco de dados usado.

- Interface padronizada aos bancos de dados: pode-se começar um projeto utilizando um determinado banco de dados e depois trocar de banco sem ter que alterar nenhuma linha de código, bastando apenas que se entre no sistema de ODBC para mudar as propriedades de conexão que existem lá.

5.2.2.2 Breve descrição das funções implementadas pelas APIs

No protótipo das APIs pode ser criado um método para cada uma das funções, como Initialize, Terminate, etc.

As listagens dos códigos estão descritas no Anexo B. Aqui será realizado um resumo de cada uma das funções desses métodos.

- **função de iniciação**: um determinado módulo carrega a função de iniciação. Nessa função, faz-se uma chamada para iniciar o processo de comunicação. É verificado se existe o objeto API_1484_11. Se o objeto for encontrado, deve-se preparar os dados a serem passados para o servidor, tais como a URL (endereço), a função a ser invocada (iniciação, finalização, etc.), quais parâmetros como curso, aluno, módulo, disciplina, etc. Deve ser verificado também se a comunicação com o servidor é possível. Sendo possível, na codificação da API no lado servidor, testa-se os tipos de erros que podem ocorrer de acordo com a norma SCORM, como por exemplo, se o SCO já está iniciado. Se não estiver, grava-se na tabela Alunos_SCO, no campo Inicializado o valor "inicializado".

- **função de fechamento**: um módulo qualquer descarrega a página e a função de finalização começa a ser executada. Procura-se o objeto API_1484_11. Encontrado, faz-se a preparação dos dados, como na função de iniciação para que estes possam ser passados para o servidor. Verifica-se se há alguma falha de comunicação com o servidor. Não ocorrendo nenhuma falha, na API lado servidor verifica-se conforme o modelo SCORM, se algum erro ocorreu, como por exemplo, finalizar a sessão sendo que

a mesma já foi encerrada. Deve-se atualizar os dados na tabela Alunos_SCO que o mesmo foi finalizado com o devido código de erro e sua respectiva descrição, caso ocorra alguma falha.

- **função para retornar o número do erro ocorrido**: retorna o código do último erro ocorrido, requisitado por uma função da API no lado cliente. Assim sendo, é necessário primeiramente fazer a comunicação com o servidor. Feito isso, é pesquisado o código do último erro na base de dados. Esse código é transportado para a função da API cliente que invocou o processo.

Na implementação da API no lado servidor, alguns exemplos de códigos de erros foram desenvolvidos neste trabalho, de acordo com as normas do SCORM, ADL (2004):

0 No Error: não ocorreu nenhum erro

102 General Initialization Failure: Erro ocorrido ao se iniciar a sessão

103 Already Initialized: SCO já iniciado

104 Content Instance Terminated: SCO já finalizado

112 General Termination Failure: o SCO quer terminar a sessão antes de iniciar

113 Termination After Termination: o SCO quer finalizar a sessão depois da sessão já ter sido encerrada

122 Retrieve Data Before Initialization: usado para testar se na função GetValue a recuperação dos dados ocorreu antes da iniciação

123 Retrieve Data After Termination: usado para testar se na função GetValue a recuperação dos dados ocorreu depois de finalizada a sessão

201 General Argument Error: parâmetro inválido passado para as funções

da API

132 Store Data Before Initialization: usado para testar se na função SetValue o armazenamento de dados ocorreu antes de se iniciar a sessão

133 Store Data After Terminate: usado para testar se na função SetValue o armazenamento dos dados ocorreu depois de finalizada a sessão

401 Undefined Data Model Element: elemento de dado não definido pelo SCORM

404 Data Model Element is Read Only: elemento de dado somente de leitura

405 Data Model Element is Write Only: elemento de dado somente de escrita

406 Data Model Element Type Mismatch: tipo de dado incompatível com o valor que se quer armazenar

- **função para retornar a descrição do erro ocorrido**: mesmo processo realizado para a obtenção do código do último erro, porém agora recuperando a sua respectiva descrição.

- **função para que o SCO peça dados ao LMS**: tem uma implementação extensa porque tem uma estrutura que utiliza comandos de condição (case, por exemplo) para testar todos os elementos do modelo de dados que suportem a função GetValue, como cmi.success_status, cmi.entry, etc.

Seu funcionamento começa quando um módulo requisita um determinado elemento de dados. É verificado se existe o objeto API_1484_11. Se existe, preparam-se os dados a serem passados para o servidor, como na função de iniciação. Testa-se

também se a comunicação com o servidor é possível. Sendo possível, há a necessidade de averiguar se nenhum erro ocorreu com a função de recuperação de dados, como os elementos de dados somente de gravação, recuperação de dados antes da iniciação, etc. Se nenhum erro ocorrer, procura-se na base de dados a informação desejada e devolve-se para o processo que fez a requisição.

- **função para que o SCO envie dados ao LMS**: semelhante à função anterior, porém ao invés de passar o dado para a função da API no lado cliente, armazena-se na base de dados correspondente o valor passado por esta função, juntamente com o nome do elemento de dado.

- **função de persistência de dados**: caso haja necessidade de implementação dessa função deve-se primeiramente verificar se o objeto API_1484_11 existe. Se existir, preparam-se os dados para serem transportados para o servidor. Se não houver falha na comunicação, a função de persistência da API no lado servidor é executada. Testa-se se ocorreu algum erro relacionado com essa função, como por exemplo, o SCO quer persistir os dados depois da comunicação ter sido encerrada, etc. Se não houver erros, armazena-se o dado na devida tabela. Nesse trabalho, não houve a necessidade de implementação dessa função, pois a cada chamada SetValue, os dados já são armazenados na base de dados.

- **função de diagnóstico:** essa última função é criada para acessar descrições mais detalhadas sobre erros ocorridos. Não está implementada, pois nessa fase de projeto não será desenvolvida essa função.

Capítulo 6

Considerações Finais

6.1 Conclusão

Com relação aos objetivos propostos por este trabalho, foi desenvolvida uma proposta de adaptação do COL ao modelo de padronização SCORM.

Primeiramente foi realizado um estudo e análise da atual situação do COL. Esse estudo baseou-se no conhecimento das funções e características do mesmo, tais como: suas ferramentas, processo de funcionamento dos objetos de aprendizagem, sua base de dados, comunicação de dados enviados entre cliente e servidor, etc.

Paralelamente foi pesquisado o modelo de padronização SCORM, como suas características e seus requisitos de funcionamento.

A partir daí, foi estudada e pesquisada uma arquitetura para que os objetos de aprendizagem pudessem ser adaptados a esta padronização.

A padronização envolveu a reformulação da base de dados do COL no que se refere aos objetos de aprendizagem, alunos, etc. e a implementação da API no lado cliente para que as funções do SCORM (Initialize, Terminate, GetValue, etc.) pudessem ser utilizadas na comunicação entre cliente e servidor.

Por último, foi sugerido um modelo de API no lado servidor, utilizando ASP e o protocolo de comunicação http, para garantir interoperabilidade, ou seja independência de plataforma.

6.2 Continuidade do trabalho

Com a modelagem dos dados (alteração da base de dados) e as funções disponibilizadas pela API Instance do SCORM, deve-se primeiramente complementar a API no lado servidor, pois neste trabalho foram implementadas algumas de suas funções.

Deve-se programar todos os elementos de dados das funções GetValue e SetValue, como também das funções de erro GetLastError e GetErrorString.

A função GetDiagnostic deverá ser também desenvolvida.

Posteriormente, muitas alterações e implementações no modelo de padronização SCORM ainda serão realizadas. Dessas implementações podem estar envolvidos novos ambientes de execução, novos modelos de arquitetura de dados, etc., incorporando aspectos de simulações, tutoriais inteligentes e jogos educacionais.

O LMS terá de ser capaz de incorporar esses aspectos avaliando os alunos e o sistema de aprendizagem, sendo necessário para isso criar uma arquitetura compatível para a realização de tais tarefas.

Anexo A

Lista dos Elementos do Modelo de Dados do SCORM

Neste Anexo, serão descritos os campos ou elementos do modelo de dados do SCORM, incluindo exemplos das funções GetValue e SetValue da API, segundo ADL (2004). Quando não exemplificada uma das funções é porque sua utilização não é permitida no elemento.

Os elementos são:

1. Identificação do aluno: cmi.learner_id.

Corresponde a um identificador único do aluno. Por meio deste elemento, o SCO pode pedir ao LMS o identificador do aluno.

Exemplo: GetValue("cmi.learner_id").

2. Nome do aluno: cmi.learner_name.

Nome do aluno, sendo também um elemento no qual o SCO pede ao LMS o nome do aluno.

Exemplo: GetValue("cmi.learner_name")

3. Comentários do aluno: cmi.comments_from_learner.

Os elementos de dados contidos nessa parte do modelo, têm a função de coletar informações dos alunos sobre as experiências de aprendizagem para que depois se possa criar uma lista ou relatório desses comentários, avaliando o projeto e a estrutura do conteúdo.

Os elementos de cmi.comments_from_learner são:

3.1 cmi.comments_from_learner._children: retorna uma lista de todos os campos ou elementos contidos em cmi.comments_from_learner.

Comments_from_learner._children deverá ser implementado utilizando a função GetValue.

3.2 cmi.comments_from_learner._count: descreve o número de comentários feitos pelos alunos para um dado SCO.

Exemplo: GetValue("cmi.comments_from_learner._count")

3.3 cmi.comments_from_learner.n.comment: descreve de forma textual o comentário do aluno, em que **n** é o número do comentário. Este campo aceita as funções GetValue e SetValue:

Exemplos:

GetValue("cmi.comments_from_learner.0.comment")

SetValue("cmi.comments_from_learner.0.comment", "this course is difficulty")

3.4 cmi.comments_from_learner.n.location: indica o ponto no SCO no qual o comentário foi aplicado. Se este campo não tiver um valor, então o comentário é aplicado como um todo para o SCO.

Exemplos:

GetValue("cmi.comments_from_learner.0.location")

SetValue("cmi.comments_from_learner.0.location", "PAGE1SECTION1")

3.5 cmi.comments_from_learner.n.timestamp: indica um ponto no tempo no qual o comentário foi criado ou alterado.

Exemplos:

GetValue("cmi.comments_from_learner.0.timestamp")

SetValue("cmi.comments_from_learner.0.timestamp", "2004-10-25T03:00:00")

4. Comentários do LMS: cmi.comments_from_lms.

Os elementos desse modelo de dados contém comentários realizados pelos desenvolvedores do SCO para serem visualizados pelos alunos. Todos os campos desse modelo são utilizados pelo SCO para pedir informações ao LMS.

Os seguintes campos fazem parte de comments_from_lms:

4.1 cmi.comments_from_lms._children: mostra uma lista de campos ou elementos suportados por este modelo.

4.2 cmi.comments_from_lms._count: é utilizado para mostrar o número de comentários do LMS.

4.3 cmi.comments_from_lms.n.comment: descreve o comentário associado ao SCO.

Exemplo: GetValue("cmi.comments_from_lms.0.comment")

4.4 cmi.comments_from_lms.n.location: indica o ponto no SCO no qual o comentário foi inserido.

Exemplo: GetValue("cmi.comments_from_lms.0.location")

4.5 cmi.comments_from_lms.n.timestamp: indica a hora e a data no qual o comentário foi criado ou alterado.

5. Preferências: cmi.learner_preference.

Especificam as preferências do aluno com relação ao som, linguagem, etc.

5.1 cmi.learner_preference._children: mostra uma lista de elementos do modelo.

Exemplo:

GetValue("cmi.learner_preference._children")

5.2 cmi.learner_preference.audio_level: especifica o nível de som (áudio).

Exemplos:

GetValue("cmi.learner_preference.audio_level")

SetValue("cmi.learner_preference.audio_level", "2") → amplificação do som.

5.3 cmi.learner_preference.language: especifica a linguagem utilizada pelo SCO. O valor padrão é em Inglês.

Exemplos:

GetValue("cmi.learner_preference.language")

SetValue("cmi.learner_preference.language, "") → string vazia corresponde a língua inglesa.

5.4 cmi.learner_preference.delivery_speed: especifica a preferência do aluno relativa a velocidade de conteúdo liberada.

Exemplos:

GetValue("cmi.learner_preference.delivery_speed")

SetValue("cmi.learner_preference.delivery_speed", "0.5")

6. Estado do SCO: cmi.completion_status.

Indica se o aluno completou ou não o SCO. Cmi.completion_status pode ter os seguintes estados: completed, incomplete, not_attempted e unknown.

- completed: indica que o aluno tem experiência ou conhecimento suficiente sobre o SCO.

- incomplete: significa que o aluno não tem experiência suficiente sobre o SCO.

- not_attempted: expressa que o aluno ainda não usou o SCO.

- unknown: indica que não se sabe o estado atual do SCO para um determinado aluno.

Exemplos:

GetValue("cmi.completion_status")

SetValue("cmi.completion_status", "incomplete")

7. Sucesso: cmi.success_status.

Indica se o aluno obteve ou não sucesso ou desempenho utilizando o SCO. Seus estados são:

- "passed": o aluno obteve sucesso utilizando o SCO.

- "failed": o aluno não obteve sucesso ou um desempenho desejado.

- " unknown": não há uma afirmação conhecida que possa ser indicada para esse campo.

Exemplos:

GetValue("cmi.success_status")

SetValue("cmi.success_status","passed")

8. Entrada: cmi.entry.

Aponta se o aluno já visitou ou acessou o SCO. Há três valores para o campo entrada:

- "ab-inibio": indica que o aluno nunca acessou o SCO.

- "resume": indica que o SCO já foi acessado.

- "" (string vazia): representa que o aluno já completou o SCO e resolveu retornar ao mesmo.

Exemplo: GetValue("cmi.entry")

9. Saída: cmi.exit.

Apresenta o motivo ou a forma como o aluno abandonou a sessão ou o SCO. Possui cinco estados:

- time_out: o SCO finalizou porque o tempo limite de utilização acabou.

- suspend: o aluno saiu do SCO com a intenção de voltar para o mesmo ponto em que ele deixou na última sessão.

- logout: o SCO terminou um conjunto de atividades de aprendizagem, ou seja, uma parte do SCO acabou.

- normal: o SCO terminou normalmente, isto é, ele foi completado inteiramente.

- "": condição de saída indeterminada.

Este elemento somente pode ser usado com a função SetValue.

Exemplo: SetValue("cmi.exit", "suspend")

10. Suspensão do SCO: cmi.suspend_data.

Durante uma sessão de aprendizagem, o aluno ou o SCO pode desejar suspender o uso do SCO e armazenar o ponto no qual ele foi suspenso. O campo cmi.suspend_data armazena e recupera os dados necessários.

Exemplo:

SetValue("cmi.suspend_data","A1;B2;C11")

11. Pontuação: cmi.score.

É o elemento do modelo de dados que corresponde à pontuação do aluno. Este modelo está dividido em outros elementos. São eles:

11.1 cmi.score._children: apresenta uma lista dos elementos desse modelo, que são: min, max e raw.

11.2 cmi.score.raw: é um número que irá refletir a performance do aluno.

Exemplos:

GetValue("cmi.score.raw")

SetValue("cmi.score.raw","0.75")

11.3 cmi.score.max: valor máximo que o aluno pode obter em um determinado SCO.

Exemplos:

GetValue("cmi.score.max")

SetValue("cmi.score.max","1.0")

11.4 cmi.score.min: valor mínimo que o aluno pode obter em um dado SCO.

Exemplos:

GetValue("cmi.score.min")

SetValue("cmi.score.min","1.0")

12. Tempo de sessão: cmi.session_time.

Especifica o tempo usado pelo aluno na sessão corrente do SCO.

Exemplo:

SetValue("cmi.session_time","1H5M")

13. Tempo total: cmi.total_time.

O valor de cmi.total_time define o tempo total utilizado pelo aluno em um dado SCO.

Exemplo: GetValue("cmi.total_time")

14. Tempo máximo permitido: cmi.max_time_allowed.

É o tempo total máximo permitido para um aluno usar um SCO.

Exemplo: GetValue("cmi.max_time_allowed")

15. Ação do limite de tempo do SCO: cmi.time_limit_action.

Indica o que o SCO pode fazer quando o tempo limite de uso exceder (cmi.max_time_allowed). O elemento cmi.time_limit_action possui quatro estados:

- "exit, message": o aluno é forçado a sair do SCO. O SCO provê uma mensagem para o aluno, indicando que o tempo máximo permitido foi excedido.

- "continue, message": o aluno continua a usar o SCO e este emite uma mensagem ao aluno, indicando que o tempo máximo permitido excedeu.

- "exit, no message": o aluno é forçado a sair do SCO, sem nenhuma mensagem de aviso.

- "continue, no message": o aluno continua a usar o SCO e não é apresentada nenhuma mensagem sobre o fato do tempo máximo permitido exceder. Esse é o valor padrão.

Exemplo: GetValue("cmi.time_limit_action")

16. Credito: cmi.credit.

Indica se o aluno tem crédito para executar o SCO. Há dois estados para este campo: credit e no_credit. O valor padrão é credit.

Utiliza-se a função GetValue: GetValue("cmi.credit")

17. Modos de apresentação do SCO: cmi.mode.

Identifica os modos possíveis no qual o SCO pode ser apresentado ao aluno. Seus estados são:

- "browse": o SCO é apresentado sem a intenção de gravar as informações sobre a sessão do aluno.

- "normal": o SCO é apresentado com a intenção de registrar as informações sobre a sessão do aluno.

- "review": o SCO armazena informações sobre o aluno, mas não atualiza as informações da sessão corrente.

Exemplo: GetValue("cmi.mode")

18. Interação: cmi.interactions.

Define um conjunto de respostas do aluno quando se tem questões ou tarefas que o criador quer registrar. Os campos desse modelo são:

18.1 cmi.interactions._children: mostra uma lista de elementos contidos nesse modelo. Deve-se usar a função GetValue para obter a lista de campos.

18.2 cmi.interactions._count: exibe o número de interações que foram armazenadas, pois este modelo é do tipo composto, permitindo armazenar vários registros sob a forma de uma lista (vetores). Por exemplo, se um SCO tiver três interações, podem ser armazenadas as três interações do aluno. Utiliza-se a função GetValue para obter o número de interações.

18.3 cmi.interactions.n.id: indica um rótulo para a interação, já que o campo é do tipo composto. O número **n** deve ser único, isto é, não pode ser repetido.

Exemplos:

GetValue("cmi.interactions.0.id")

SetValue("cmi.interactions.0.id","obj1")

18.4 cmi.interactions.n.type: identifica qual é o tipo de interação. O tipo de interação determina como a resposta do aluno deve ser interpretada. Os tipos são:

- "true_false": a interação tem duas possíveis respostas: verdadeira ou falsa.

Exemplos:

GetValue("cmi.interactions.n.type")

SetValue("cmi.interactions.n.type","true_false")

- "multiple_choice": a interação tem um conjunto de duas ou mais respostas possíveis.

- "fill_in": a interação deve ser interpretada com uma resposta curta de apenas algumas palavras.

- "long_fill_in": a resposta é interpretada por meio de várias palavras.

- "matching": a resposta do aluno deve ser interpretada como uma questão com dois conjuntos de itens em que cada item do primeiro conjunto se relaciona com nenhum ou mais de um item do outro.

- "likert": permite ao aluno selecionar um conjunto de valores sobre uma determinada escala.

- "performance": interpreta a resposta do aluno como um conjunto de passos que devem ser tomados.

- "sequencing": identifica uma ordem lógica para os itens de uma lista. O aluno deve indicar a ordem correta.

- "numeric": requer uma resposta numérica do aluno.

- "other": qualquer outro tipo não definido acima.

18.5 cmi.interactions.n.objectives._count: mostra o número de objetivos que há para cada interação.

Exemplo: GetValue ("cmi.interactions.0.objectives._count")

18.6 cmi.interactions.n.objectives.n.id: é um rótulo para o objetivo associado com a respectiva interação. Este rótulo deve ser único dentro do SCO.

Exemplos:

GetValue ("cmi.interactions.0.objectives.0.id")

SetValue ("cmi.interactions.0.objectives.0.id", "id-0001")

18.7 cmi.interactions.n.timestamp: corresponde a um ponto no tempo no qual a interação foi iniciada pelo aluno.

Exemplos:

GetValue ("cmi.interactions.0.timestamp")

SetValue ("cmi.interactions.0.timestamp", "2004-10-25T10:00:00")

18.8 cmi.interactions.n.correct_responses._count: exibe o número de respostas corretas do aluno para cada interação.

Exemplo:

GetValue ("cmi.interactions.0.correct_responses._count")

18.9 cmi.interactions.n.correct_responses.n.pattern: define a resposta correta para cada interação. O formato desse modelo depende do elemento cmi.interactions.n.type.

Exemplos:

GetValue("cmi.interactions.0.correct_responses.0.pattern")

SetValue("cmi.interactions.0.correct_responses.0.pattern","true")

18.10 cmi.interactions.n.weighting: é o peso dado para uma determinada interação.

Exemplos:

GetValue("cmi.interactions.0.weighting")

SetValue("cmi.interactions.0.weighting", "1.0")

18.11 cmi.interactions.n.learner_response: é a resposta que o aluno dá para cada interação.

Exemplos:

GetValue("cmi.interactions.0.learner_response")

SetValue("cmi.interactions.0.learner_response","true")

18.12 cmi.interactions.n.result: é o julgamento da resposta do aluno. Possui vários estados, tais como:

- "correct": a resposta do aluno está correta.

- "incorrect": a resposta do aluno está incorreta.

Exemplos:

GetValue("cmi.interactions.0.result")

SetValue(cmi.interactions.0.result","correct")

18.13 cmi.interactions.n.latency: corresponde ao tempo de interação iniciada pelo aluno e o tempo da sua resposta, ou seja, a diferença de cmi.interactions.n.timestamp e o tempo de resposta.

Exemplos:

GetValue("cmi.interactions.0.latency")

SetValue(cmi.interactions.0.latency","5M")

18.14 cmi.interactions.n.description: descreve a interação ou questão para o aluno.

Exemplos:

GetValue("cmi.interactions.0.description")

SetValue("cmi.interactions.0.description","Which of the following are red")

19. Objetivos do SCO: cmi.objectives.

Identifica o desenvolvimento de cada aluno relativamente aos objetivos de um determinado SCO. Inclui informações sobre os objetivos do SCO e o estado em que o aluno se encontra a esses objetivos. Este campo é do tipo composto.

Possui os seguintes elementos:

19.1 cmi.objectives._children: exibe a lista dos elementos desse modelo.

Exemplo: GetValue("cmi.objectives._children")

19.2 cmi.objectives._count: lista o número de objetivos de um SCO.

Exemplo: GetValue("cmi.objectives._count")

19.3 cmi.objectives.n.id: corresponde a um identificador do objetivo, no qual deve ser único dentro do SCO.

Exemplos:

GetValue("cmi.objectives.0.id")

SetValue("cmi.objectives.0.id","obj1")

19.4 cmi.objectives.n.score._children: apresenta uma lista de elementos de cmi.objectives.n.score. São eles: raw, min, max.

- cmi.objectives.n.score.raw: corresponde a uma nota do aluno para um determinado objetivo, ou seja, seu desempenho.

Exemplos:

GetValue("cmi.objectives.0.score.raw")

SetValue("cmi.objectives.0.score.raw","0.75")

- cmi.objectives.n.score.min: é o valor mínimo atribuído a um aluno para um determinado objetivo.

Exemplos:

GetValue("cmi.objectives.0.score.min")

SetValue("cmi.objectives.0.score.min","0.75")

- cmi.objectives.n.score.max: é o valor máximo atribuído para um determinado objetivo.

Exemplos:

GetValue("cmi.objectives.0.score.max")

SetValue("cmi.objectives.0.score.max","1.0")

19.5 cmi.objectives.success_status: indica se o aluno compreendeu o objetivo proposto. Seus estados são:

- "passed": o aluno compreendeu o objetivo.

- "failed": o aluno não compreendeu o objetivo.

- " unknown": não se sabe se o aluno compreendeu ou não o objetivo.

Exemplos:

GetValue("cmi.objectives.n.success_status")

SetValue("cmi.objectives.n.success_status","passed")

19.6 cmi.objectives.n.completion_status: indica se o aluno completou o objetivo associado ao SCO. Os possíveis estados são:

- "completed": o aluno tem experiência o suficiente do objetivo em questão.

- "incomplete": o aluno não tem experiência o suficiente para atingir o objetivo proposto.

- unknown: não pode ser inferido nenhum dos estados citados, pois não se têm os julgamentos necessários para tal.

Exemplos:

GetValue("cmi.objectives.0.completion_status")

SetValue("cmi.objectives.0.completion_status","incomplete")

19.7 cmi.objectives.n.progress_measure: representa um número que mede o progresso do aluno. Este número está no intervalo de $0 \leq$ cmi.objectives.n.progress_measure ≤ 1. Zero indica que o aluno ainda não iniciou o objetivo e um, que já completou.

Exemplos:

GetValue("cmi.objectives.0.progress_measure")

SetValue("cmi.objectives.0.progress_measure","0.75")

19.8 cmi.objectives.n.description: provê uma descrição do objetivo.

Exemplos:

GetValue("cmi.objectives.0.description")

SetValue("cmi.objectives.0.description","upon completion of this unit, the learner shall be able to distinguish...")

Anexo B

O arquivo de manifesto, os programas para a implementação das APIs (lado cliente e servidor) e a codificação de um SCO para iniciar e terminar a sessão, entre outras funções

Neste Anexo serão descritos alguns programas necessários para a implementação das APIs (lado cliente e servidor), a estrutura de um arquivo de manifesto simples e a codificação de um SCO para iniciar e terminar a sessão, entre outras funções.

B.1. Arquivo de manifesto

O arquivo de manifesto é usado para a instalação de um curso. É útil para a documentação e pode ser utilizado como transporte de um sistema para outro.

O arquivo XML de manifesto (imsmanifest.xml) é composto pela organização, recursos e metadados. Os recursos têm a função de indicar quais são os conteúdos educacionais que estão agrupados. A organização é responsável por apontar a estrutura dos conteúdos de ensino, dentro da uma determinada seqüência. Metadados correspondem aos dados sobre dados dos recursos educacionais armazenados.

Tem-se abaixo um exemplo simples da estrutura de um arquivo de manifesto:

```
<?xml version = "1.0"?>
<! -- o <manifest> é o elemento nó do arquivo que define a organização,
        metadado e recurso para garantir sua reusabilidade -- >
<manifest identifier="SAMPLE" version="1.3"
<! -- xmlns e xsi, correspondem aos namespaces ou tags, que são conjuntos
de nomes únicos e esquemas de dados definidos pelos endereços (URL)
especificados -- >
        xmlns="http://www.imsglobal.org/xsd/imscp_v1p1"
        xmlns:adlcp="http://www.adlnet.org/xsd/adlcp_v1p3"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.imsglobal.org/xsd/imscp_v1p1
                imscp_v1p1.xsd
                http://www.adlnet.org/xsd/adlcp_v1p3
                adlcp_v1p3.xsd">
```

```
<metadata>
< ! -- descreve o esquema que define e controla o manifesto: ADL SCORM e
a versão do CAM - Content Aggregation Model -- >
<schema>ADL SCORM</schema>
<schemaversion>CAM 1.3</schemaversion>
< ! -- define o atributo geral título do LO baseado nos padrões da IEEE -- >
< ! -- os outros atributos, como ciclo de vida, informações técnicas,
educacionais e de direito são definidos de maneira similar -- >
<lom xmlns="http://ltsc.ieee.org/xsd/LOM">
<general>
<title>
Título do pacote
</title>
</general>
</metadata>

< ! -- a organização descreve quais SCO estão envolvidos no curso -- >
<organizations>
<organization identifier="nome"> <! -- nome do curso -- >
< ! -- detalhando os SCO -- >
<item identifier="ITEM1" identifierref="sco0">
<title> Primeiro SCO do curso </title>
</item>
<item identifier="ITEM2" identifierref="sco1">
<title> Segundo SCO do curso </title>
</item>
</organization>
</organizations>

< ! -- os recursos descrevem quais SCO estão agrupados e seus respectivos
endereços -->
<resources>
<resource identifier="sco0" type="webcontent" href="lesson1.html">
<file href="lesson1.html"/>< ! -- se houver mais de um arquivo para cada
SCO, deve-se especificar em file href -- >
<file href="fig1.swf"/>
</resource>
<resource identifier="sco1" type="webcontent" href="lesson2.html">
<file href="lesson2.html"/>
</resource>
</resource>
</manifest>
```

B.2 API Instance

A API Instance irá estabelecer a comunicação com o servidor e disponibilizar as funções da norma SCORM, como Initialize(), SetValue(), GetValue(), Terminate(), etc.

Estas funções terão que estar todas em um objeto com o nome de "API_1484_11" (este nome é atribuído em função da norma SCORM que define a API) criadas em JavaScript. Assim quando o SCO quiser, por exemplo, chamar a função Initialize(), terá que localizar o objeto "API_1484_11" que deverá estar em uma janela pai daquela em que o SCO se encontra e utilizar o método Initialize desse objeto.

Os programas que serão apresentados se dividem em duas partes. O primeiro programa (APIWrapper.js) tem a função de procurar o objeto API_1484_11 e fazer a interface para disponibilizar as funções do SCORM. O segundo programa (API.js) tem como objetivo estabelecer a comunicação do SCO com a API no lado servidor.

B.2.1 Programa APIWrapper.js:

Este programa foi desenvolvido baseado em algumas codificações realizadas pela ADL, ADL (2004).

```
// criação das variáveis globais
var apiHandle = null;
var findAPITries = 0;
var noAPIFound = "false";
var terminated = "false";
var _debug = false;

/*************************************************************************
** Esta função procura pelo objeto API_1484_11 nas janelas pais existentes
*************************************************************************/
function findAPI (win)
{
 while ( (win.API_1484_11 == null) && (win.parent != null) && (win.parent != win) )
   {
    findAPITries++;
    if ( findAPITries > 500 )
    {
      alert( "Error finding API" );
```

```
      return null;
    }
    win = win.parent;
  }
return win.API_1484_11;
}

/*****************************************************************
** Esta função invoca uma chamada para procurar o objeto API na hierarquia
** das janelas de frames
*****************************************************************/
function getAPI()
{
  var theAPI = findAPI( window );
  if ( (theAPI == null) && (window.opener != null) && (typeof(window.opener) !=
"undefined") )
  {
    theAPI = findAPI(window.opener);
  }
  if (theAPI == null)
  {
    alert( "Unable to locate the LMS's API Implementation.\n" +
"Communication with the LMS will not occur." );
    noAPIFound = "true";
  }
  return theAPI
}

/*****************************************************************
** Faz a chamada para procurar o objeto API_1484_11 se necessário
*****************************************************************/
function getAPIHandle()
{
  if ( apiHandle == null )
  {
    if ( noAPIFound == "false" )
    {
      apiHandle = getAPI();
    }
  }
  return apiHandle;
}
```

```
/*****************************************************************
** Inicia a comunicação com o LMS chamando o método Initialize() que
** está localizado no arquivo API.js
*****************************************************************/
function initializeCommunication()
{
  var api = getAPIHandle();
  if ( api == null )
  {
    return "false";
  }
  else
  {
    var result = api.Initialize("");
    if ( result != "true" )
    {
      var errCode = retrieveLastErrorCode();
      displayErrorInfo( errCode );
    }
  }
  return result;
}

/*****************************************************************
** Função utilizada para "dizer" ao LMS que a sessão deve ser finalizada
*****************************************************************/
function terminateCommunication()
{
  var api = getAPIHandle();
  if ( api == null )
  {
    return "false";
  }
  else
  {
    if ( terminated != "true" )
    {
      var result = api.Terminate("");
      if ( result != "true" )
      {
        var errCode = retrieveLastErrorCode();

        displayErrorInfo( errCode );
      }
      else  // terminate was successful
```

```
      {
        terminated = "true";
      }
    }
  }
  return result;
}

/*******************************************************************
** Invoca o método GetValue do arquivo API.js para fazer a comunicação
** com o servidor, pedindo um determinado dado ao LMS
*******************************************************************/
function retrieveDataValue( name )
{
   if ( terminated != "true" )
  {
    var api = getAPIHandle();
    if ( api == null )
    {
      return "";
    }
    else
    {
      var value = api.GetValue( name );
      var errCode = api.GetLastError();
      if ( errCode != "0" )
      {
        var errCode = retrieveLastErrorCode();
        displayErrorInfo( errCode );
      }
      else
      {
        return value;
      }
    }
  }
  return;
}

/*******************************************************************
** Função para chamar o método SetValue do arquivo API.js para que os dados
** possam ser salvos na base de dados
*******************************************************************/

function storeDataValue(name, value)
```

```
{
  if ( terminated != "true" )
  {
    var api = getAPIHandle();
    if ( api == null )
    {
      return;
    }
    else
    {
      var result = api.SetValue( name, value );
      if ( result != "true" )
      {
        var errCode = retrieveLastErrorCode();
        displayErrorInfo( errCode );
      }
    }
  }
  return;
}

/*****************************************************************************
** Chama o método GetLastError do arquivo API.js
*****************************************************************************/
function retrieveLastErrorCode()
{
  var api = getAPIHandle();
  if ( api == null )
  {
    return "";
  }
  else
  {
    return api.GetLastError();
  }
}

/*****************************************************************************
** Função para invocar o método GetErrorString do arquivo API.js
*****************************************************************************/
function retrieveErrorInfo( errCode )
{
  var api = getAPIHandle();
  if ( api == null )
  {
```

```
    return "";
  }
  else
  {
    return api.GetErrorString(errCode);
  }
}

/*****************************************************************************
** Função para invocar o método GetDiagnost do arquivo API.js
*****************************************************************************/
function retrieveDiagnosticInfo(error)
{
  var api = getAPIHandle();
  if ( api == null )
  {
    return "";
  }
  else
  {
    return api.GetDiagnostic(error);
  }
}

/*****************************************************************************
** Função para invocar o método Commit do arquivo API.js
*****************************************************************************/
function persistData()
{
  if ( terminated != "true" )
  {
    var api = getAPIHandle();
    if ( api == null )
    {
      return "";
    }
    else
    {
      return api.Commit();
    }
  }
  else
  {
    return "";
  }
```

```
}

/*******************************************************************
** Função para exibir o código e a descrição do último erro
*******************************************************************/
function displayErrorInfo(errCode)
{
  if ( _debug )
  {
    var errString = retrieveErrorInfo( errCode );
    var errDiagnostic = retrieveDiagnosticInfo( errCode );
    alert( "ERROR: " + errCode + " - " + errString + "\n" + "DIAGNOSTIC: " +
errDiagnostic );
  }
}
```

B.2.2 Programa API.js

Este programa estabelece a comunicação do cliente com o servidor passando os dados necessários para tal. Essa comunicação é estabelecida por meio de JavaScript e XML

```
function ParsedResponse(returnValue, errorCode, text)
{
        this.returnValue = returnValue;
        this.errorCode = errorCode;
        this.text = text;
}

function CommunicationsFailureResponse()
{
        this.returnValue = "false";
        this.errorCode = 1001;
        this.text = "Failed to open a HTTP channel to the LMS.";
}

function ParseFailureResponse()
{
        this.returnValue = "false";
        this.errorCode = 1002;
        this.text = "Failed to parse the response.";
}

function GenericFailureResponse(description)
```

```
{
	this.returnValue = "false";
	this.errorCode = 1003;
	this.text = description;
}

/**********************************************************************
** A função Transport tem como objetivo fazer uma chamada HTTP ao
** servidor, criando um URL no momento que passa os dados da chamada (qual
** função invocar: iniciar, terminar, etc., qual curso, aluno, disciplina, modulo, etc).
**********************************************************************/
function Transport()
{
	this.aborted = false;
	this.abortMessage = "";
	this.abort = function(response)
	{
		this.aborted = true;
		this.abortMessage = response.text;
		return response;
	}
	this.parseResponse = function(data)
	{
		if (data.documentElement == null)
		{
			return this.abort(new ParseFailureResponse());
		}
		var returnValue = data.documentElement.getAttribute("return");
		var errorCode = data.documentElement.getAttribute("error");
		var text = data.documentElement.childNodes[0].data;
		return new ParsedResponse(returnValue, errorCode, text);
	}

	this.get = function(url)
	{
		if (this.aborted)
		{
			alert("This session has been aborted. Please, contact your support
staff.\n\nCause: " + this.abortMessage);
			throw { description: this.abortMessage };
		}
		var xml;
		if (window.XMLHttpRequest)
		{
			xml = new XMLHttpRequest();
```

```
            }
            else
            {
                xml = new ActiveXObject("MSXML2.XMLHTTP");
            }

            try
            {
                // método de requisição da página por XML HTTP
                xml.open("GET", url, false);
                xml.send("");
                if (xml.status != "200")
                {
                    return this.abort(new CommunicationsFailureResponse());
                }
                return this.parseResponse(xml.responseXML);
            }
            catch(e)
            {
                return this.abort(new GenericFailureResponse(e.description));
            }
        }
}

/*****************************************************************
** O objeto APIImplementation fornece a implementação da API usando parâmetros
** que o LMS passa para ele quando vai instanciar um módulo qualquer.
** Cada função está especificada na declaração do objeto: Initilize(), Terminate(), etc.
*****************************************************************/
function APIImplementation()
{
    this.url = "";
    this.sessionId = "";
    this.sco = "";
    this.transport = new Transport();
    this.lastResponse = null;
    this.prepareUrl = function(apiFunction, parameters)
    {
        var i;
        var url = "";
        for (var i = 0; i < parameters.length; i++)
        {
            url += url + "&p" + i + "=" + escape(parameters[i]);
        }
```

```
            url = url + "&sid=" + this.sessionId + "&sco=" + this.sco +
        "&fapi=" + apiFunction;
            return this.url + url;
        }

        this.Initialize = function(parameter)
        {
            this.lastResponse = this.transport.get(this.prepareUrl("Initialize",
[parameter]));

            return this.lastResponse.returnValue;
        }

        this.Terminate = function(parameter)
        {
            this.lastResponse = this.transport.get(this.prepareUrl("Terminate",
[parameter]));
            return this.lastResponse.returnValue;
        }

        this.GetValue = function(parameter)
        {
            this.lastResponse = this.transport.get(this.prepareUrl("GetValue",
[parameter]));
            return this.lastResponse.text;
        }

        this.SetValue = function(parameter, value)
        {
            this.lastResponse = this.transport.get(this.prepareUrl("SetValue",
[parameter, value]));
            return this.lastResponse.returnValue;
        }

        this.Commit = function(parameter)
        {
            this.lastResponse = this.transport.get(this.prepareUrl("Commit",
[parameter]));
            return this.lastResponse.returnValue;
        }

        this.GetLastError = function()
        {
            this.lastResponse = this.transport.get(this.prepareUrl("GetLastError", []));
            return this.lastResponse.text;
```

```
        }

        this.GetErrorString = function(parameter)
        {
                this.lastResponse = this.transport.get(this.prepareUrl("GetErrorString",
[parameter]));
                return this.lastResponse.text;
        }

        this.GetDiagnostic = function(parameter)
        {
                this.lastResponse = this.transport.get(this.prepareUrl("GetDiagnostic",
[parameter]));
                return this.lastResponse.text;
        }
}
```

Toda vez que é chamado cada um dos métodos (Initialize, Terminate, etc.), a API Instance irá se comunicar com a API no lado servidor para que este execute as operações pedidas pelo SCO.

B.3 API no lado servidor

Neste trabalho, para a implementação da API no lado servidor, optou-se pela utilização do ASP, que o próprio COL já utiliza em suas rotinas.

A implementação aqui descrita corresponde apenas a uma parte, pois sua codificação é fácil, porém extensa. Tem-se o arquivo API.asp com suas instruções.

B.3.1 API.asp

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' cabecalho.asp: tem por objetivo configurar a página no que diz respeito ao
' seu carregamento
' bd.asp: arquivo com as conexões da base de dados
' util.asp: configuração das mensagens exibidas aos usuários
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
<!--#include file="lib/cabecalho.asp"-->
<!--#include file="lib/bd.asp"-->
<!--#include file="lib/util.asp"-->

<%
        Dim IdAluno
```

```
        Dim IdGrupo
        Dim IdTurma
        Dim IdDisciplina
        Dim IdSessao
        Dim IdSco
        Dim FuncaoApi
        Dim Parametro
        Dim ValorParametro
        IdAluno = Request("aid")
        IdGrupo = Request("gid")
        IdTurma = Request("tid")
        IdDisciplina = Request("did")
        IdSessao = Request("sid")
        IdSco = Request("sco")
        FuncaoApi = Request("fapi")
        Parametro = Request("p0")
        ValorParametro = Request("p1")

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' verifica os erros ocorridos segundo a norma SCORM
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function RecuperarDescricaoErro(ByVal Erro)
        Select Case Erro
                Case 101
                        RecuperarDescricaoErro = "General Exception"
                Case 102
                        RecuperarDescricaoErro = "General Initialization Failure"
                Case 103
                        RecuperarDescricaoErro = "Already Initialized"
                Case 104
                        RecuperarDescricaoErro = "Content Instance Terminated"
                Case 111
                        RecuperarDescricaoErro = "General Termination Failure"
                Case 112
                        RecuperarDescricaoErro = "Termination Before Initialization"
                Case 113
                        RecuperarDescricaoErro = "Termination After Termination"
                Case 122
                        RecuperarDescricaoErro = "Retrieve Data Before Initialization"
                Case 123
                        RecuperarDescricaoErro = "Retrieve Data After Termination"
                Case 132
                        RecuperarDescricaoErro = "Store Data Before Initialization"
                Case 133
                        RecuperarDescricaoErro = "Store Data After Termination"
```

```
            Case 142
                    RecuperarDescricaoErro = "Commit Before Initialization"
            Case 143
                    RecuperarDescricaoErro = "Commit After Termination"
            Case 201
                    RecuperarDescricaoErro = "General Argument Error"
            Case 301
                    RecuperarDescricaoErro = "General Get Failure"
            Case 351
                    RecuperarDescricaoErro = "General Set Failure"
            Case 391
                    RecuperarDescricaoErro = "General Commit Failure
            Case 401
                    RecuperarDescricaoErro = "Undefined Data Model Element"
            Case 404
                    RecuperarDescricaoErro = "Data Model Element is Read Only"
            Case 405
                    RecuperarDescricaoErro = "Data Model Element is Write Only"
            Case 406
                    RecuperarDescricaoErro = "Data Model Element Type Mismatch"
                    ' existem outros códigos de erros que devem ser acrescentados
            Case Else
                    RecuperarDescricaoErro = "Unknown Error Code " & Erro
        End Select
End Function

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' recupera o nome do aluno
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function RecuperarNomeAluno
        RecuperarNomeAluno = RecuperarLista("select nome from aluno where
id_aluno = " & IdAluno)(0, 0)
End Function

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' recebe como parâmetro o sucesso da operação, o código do erro e sua descrição
' e salva na tabela Alunos_SCO o último erro e sua respectiva mensagem
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub CriarResposta(ByVal Sucesso, ByVal Resposta, ByVal CodigoErro, ByVal
DescricaoErro, ByVal SalvarErro)
        If IsNumeric(DescricaoErro) Then
                DescricaoErro = RecuperarDescricaoErro(DescricaoErro)
        End If
                If SalvarErro Then
                        SalvarItem "ultimo_erro", CodigoErro
```

```
				SalvarItem "ultima_mensagem", DescricaoErro
			End If
		' cria uma resposta XML para retorno com base nos parâmetros passados
			Response.ContentType = "text/xml"
			Response.Write "<?xml version=""1.0"" encoding=""iso-8859-1""?>"
			Response.Write "<response return=""" & IIf(Sucesso, "true", "false") &
""" error=""" & CodigoErro & """>"
			Response.Write "<![CDATA[" & Replace(Replace(Resposta, """",
"&quot;"), "<", "&gt;") & "]]>"
			Response.Write "</response>"
End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'' seleciona o devido campo da tabela Aluno_SCO pelo parâmetro passado
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function RecuperarItem(ByVal Nome)
		Dim Sql
		Dim Dados
		Sql = _
			" select coalesce(" & Nome & ", '')" & _
			" from alunos_sco " & _
			" where " & _
				" id_aluno = " & IdAluno & " and " & _
				" id_turma = " & IdTurma & " and " & _
				" id_grupo = " & IdGrupo & " and " & _
				" id_disciplina = " & IdDisciplina & " and " & _
				" id_sco = " & IdSco & " and " & _
				" id_aluno = " & IdAluno
		Dados = RecuperarLista(Sql)
		If Not IsNull(Dados) Then
			RecuperarItem = Dados(0, 0)
		Else
			RecuperarItem = ""
		End If
End Function

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' atualiza a tabela Aluno_SCO de acordo com o nome do campo e o valor passado
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub SalvarItem(ByVal Nome, ByVal Valor)
		Dim Sql
		Sql = _
				" update alunos_sco " & _
				" set " & Nome & " = '" & Replace(Valor, "'", "''") & "' " & _
				" where " & _
```

```
                    " id_aluno = " & IdAluno & " and " & _
                    " id_turma = " & IdTurma & " and " & _
                    " id_grupo = " & IdGrupo & " and " & _
                    " id_disciplina = " & IdDisciplina & " and " & _
                    " id_sco = " & IdSco & " and " & _
                    " id_aluno = " & IdAluno
    ExecutarSql(Sql)
End Sub

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' recebe como parâmetro o nome do elemento de dados.
' Dependendo de qual é o elemento de dados, atribui-se um respectivo valor
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function RecuperarItemAPI(ByVal Nome, ByRef Valor)
    Dim Temporario1
    Dim Temporario2
    Valor = ""
    RecuperarItemAPI = 0
    Select Case Nome
        Case "cmi.learner_id"
            Valor = IdAluno
        Case "cmi.learner_name"
            Valor = RecuperarNomeAluno
        Case "cmi.completion_status"
            Valor = RecuperarItem("estado_licao")
            If Valor <> "completed" And Valor <> "incomplete" And Valor
<> "not attempted" And Valor <> "unknown" Then
                Valor = "unknown"
            End If
        Case "cmi.entry"
            Temporario1 = RecuperarItem("entrada")
            Temporario2 = RecuperarItem("saida")
            If Temporario1 = "" Then
                Valor = "ab-initio"
            ElseIf Temporario2 = "suspend" Or Temporario2 = "logout" Then
                Valor = "resume"
            Else
                Valor = ""
            End If
        Case Else
            RecuperarItemAPI = 401
    End Select
End Function

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
' recebe como parâmetro um determinado elemento de dado que deve
' ser de escrita e verifica se há algum tipo de erro
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function SalvarItemAPI(ByVal Nome)
	SalvarItemAPI = 0
	Select Case Nome
		Case "cmi.learner_id", "cmi.learner_name", "cmi.entry"
			SalvarItemAPI = 404
		Case "cmi.completion_status"
			If ValorParametro = "completed" Or ValorParametro = "incomplete" Or ValorParametro = "not attempted" Or ValorParametro = "unknown" Then
				SalvarItem "estado_licao", ValorParametro
			Else
				SalvarItemAPI = 406
			End If
		Case "cmi.success.status"
			SalvarItem "estado_sucesso", ValorParametro
		Case "cmi.exit"
			SalvarItem "saida", ValorParametro
		Case "cmi.score.raw"
			SalvarItem "desempenho", ValorParametro
		Case Else
			SalvarItemAPI = 401
	End Select
End Function

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' rotina para começar o processo de iniciação do SCO
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub ResponderInicializar
	Dim SituacaoInicializacao
	SituacaoInicializacao = RecuperarItem("inicializado")
	If Parametro <> "" Then
		CriarResposta False, "", 201, 201, True
	ElseIf SituacaoInicializacao = "inicializado" Then
		CriarResposta False, "", 103, 103, True
	ElseIf SituacaoInicializacao = "terminado" Then
		CriarResposta False, "", 104, 104, True
	ElseIf SituacaoInicializacao = "" Then
		SalvarItem "inicializado", "inicializado"
		CriarResposta True, "", 0, "", True
	Else
		CriarResposta False, "", 102, "O estado da API está incorreto", True
	End If
```

```
End Sub
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'rotina para iniciar o processo de finalização do SCO
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub ResponderTerminar
        Dim SituacaoInicializacao
        SituacaoInicializacao = RecuperarItem("inicializado")
        If Parametro <> "" Then
                CriarResposta False, "", 201, 201, True
        ElseIf SituacaoInicializacao = "terminado" Then
                CriarResposta False, "", 113, 113, True
        ElseIf SituacaoInicializacao = "" Then
                CriarResposta False, "", 112, 112, True
        ElseIf SituacaoInicializacao = "inicializado" Then
                SalvarItem "inicializado", "terminado"
                CriarResposta True, "", 0, "", True
        Else
                CriarResposta False, "", 111, "O estado da API está incorreto", True
        End If
End Sub
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' rotina para iniciar o processo de persistência de dados
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub ResponderPersistir
        Dim SituacaoInicializacao
        SituacaoInicializacao = RecuperarItem("inicializado")
        If Parametro <> "" Then
                CriarResposta False, "", 201, 201, True
        ElseIf SituacaoInicializacao = "terminado" Then
                CriarResposta False, "", 143, 143, True
        ElseIf SituacaoInicializacao = "" Then
                CriarResposta False, "", 142, 142, True
        ElseIf SituacaoInicializacao = "inicializado" Then
                CriarResposta True, "", 0, "", True
        Else
                CriarResposta False, "", 111, "O estado da API está incorreto", True
        End If
End Sub

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' rotina para iniciar o processo de recuperação de dados: GetValue
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Sub ResponderRecuperarValor
        Dim Resultado
        Dim Valor
        Dim SituacaoInicializacao
        SituacaoInicializacao = RecuperarItem("inicializado")
        If Parametro = "" Then
                CriarResposta False, "", 401, 401, True
        ElseIf SituacaoInicializacao = "terminado" Then
                CriarResposta False, "", 123, 123, True
        ElseIf SituacaoInicializacao = "" Then
                CriarResposta False, "", 122, 122, True
        ElseIf SituacaoInicializacao = "inicializado" Then
                Resultado = RecuperarItemAPI(Parametro, Valor)
                If Resultado = 0 Then
                        CriarResposta True, Valor, 0, "", True
                Else
                        CriarResposta False, "", Resultado, Resultado, True
                End If
        Else
                CriarResposta False, "", 111, "O estado da API está incorreto", True
        End If
End Sub

‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
‘ rotina para iniciar o processo de gravação de dados (SetValue)
‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’

Sub ResponderSalvarValor
        Dim Resultado
        Dim SituacaoInicializacao
        SituacaoInicializacao = RecuperarItem("inicializado")
        If Parametro = "" Then
                CriarResposta False, "", 401, 401, True
        ElseIf SituacaoInicializacao = "terminado" Then
                CriarResposta False, "", 133, 133, True
        ElseIf SituacaoInicializacao = "" Then
                CriarResposta False, "", 132, 132, True
        ElseIf SituacaoInicializacao = "inicializado" Then
                Resultado = SalvarItemAPI(Parametro)
                If Resultado = 0 Then
                        CriarResposta True, "", 0, "", True
                Else
                        CriarResposta False, "", Resultado, Resultado, True
                End If
        Else
                CriarResposta False, "", 111, "O estado da API está incorreto", True
```

```
        End If
        End Sub

‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
‘ rotina que controla o código do último erro ocorrido
‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
Sub ResponderUltimoErro
        CriarResposta True, RecuperarItem("ultimo_erro"), 0, "", False
End Sub

‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
‘ rotina que controla a descrição do último erro ocorrido
‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
Sub ResponderDescricaoUltimoErro
        CriarResposta True, RecuperarDescricaoErro(RecuperarItem("ultimo_erro")), 0,
"", False
End Sub

‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
‘ rotina que controla a função de diagnóstico
‘’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’’
Sub ResponderDiagnostico
        If Parametro = "" Then
                CriarResposta True, RecuperarItem("ultima_mensagem"), 0, "", False
        Else
                CriarResposta True, RecuperarDescricaoErro(Parametro), 0, "", False
        End If
End Sub

        IniciarTransacao
        If RecuperarItem("sessao") <> IdSessao Then
                SalvarItem "sessao", IdSessao
                SalvarItem "inicializado", ""
                SalvarItem "ultimo_erro", "0"
                SalvarItem "ultima_mensagem", ""
        End If
        Select Case FuncaoApi
                Case "Initialize"
                        ResponderInicializar
                Case "Terminate"
                        ResponderTerminar
                Case "Commit"
                        ResponderPersistir
                Case "GetValue"
                        ResponderRecuperarValor
```

```
            Case "SetValue"
                  ResponderSalvarValor
            Case "GetLastError"
                  ResponderUltimoErro
            Case "GetErrorString"
                  ResponderDescricaoUltimoErro
            Case "GetDiagnostic"
                  ResponderDiagnostico
      End Select
      If Err.Number <> 0 Then
            Response.Clear
            CriarResposta False, Err.Description, 101, "", False
            CancelarTransacao
      Else
            FinalizarTransacao
      End If
%>
```

B.4 Codificação de um SCO para iniciar e terminar a sessão, entre outras funções

Tem-se abaixo a codificação com as funções necessárias para que um SCO possa iniciar e finalizar, calculando o tempo total de sessão, entre outras funções. Este programa deve ser inserido em todos os SCO, dentro de sua página para carregá-lo e assim utilizar as funções contidas dentro do programa.

Este programa (SCOFunction.js) foi implementado pela ADL, ADL (2004) e utilizado uma parte de suas funções no presente trabalho.

```
/*******************************************************************
** variáveis globais
*******************************************************************/
var IntraNavigation = false
var startDate;

/*******************************************************************
** Esta função tem por objetivo ir para uma outra página de um mesmo SCO,
** pois um SCO pode ter vários arquivos físicos. Deve ser passado como parâmetro
** a página desejada
*******************************************************************/
function GoToPage(page)
{
      IntraNavigation = true;
      window.location.replace(page);
```

```
}

/*****************************************************************
** Função que está relacionada ao sequenciamento dos SCO, no qual verifica se existe
** um SCO ou Asset anterior. Se existir, deve-se exibir o botão "previous"
** Retorna "true" se existir um SCO ou Asset anterior
*****************************************************************/

function renderPreviousButton()
{
  var value = retrieveDataValue( "adl.nav.request_valid.previous" );
  return value;
}

/*****************************************************************
** Função que está relacionada ao seqüenciamento dos SCO, no qual verifica se existe
** um próximo SCO ou Asset para continuação de um curso.
** Se existir, deve-se exibir um botão de continuação
** Retorna "true" se existir um próximo SCO
*****************************************************************/
function renderContinueButton()
{
  var value = retrieveDataValue( "adl.nav.request_valid.continue" );
  return value;
}

/*****************************************************************
** Função para armazenar o local em que o SCO foi finalizado
*****************************************************************/
function onUnexpectedExit(value)
{
        storeDataValue( "cmi.location", value );
        if ( IntraNavigation == false )
        {
          terminateCommunication();
        }
        else
        {
                IntraNavigation = false;
        }
}

/*****************************************************************
** Esta função propicia o início de comunicação entre o SCO e o LMS,
** usando o arquivo APIWrapper.js
```

```
**************************************************************************/
// showNext é um parâmetro opcional, que assume os valores "true" ou "false"
function Initialize(showNext)
{
        if ( !(entryStatus == "resume") )
        {
                initializeCommunication();
        }
        // if there is somewhere to go next and previous enable the buttons
        if ( renderContinueButton() == "true")
        {
                if ( showNext != "false" )
                document.getElementById("nextBtn").style.visibility = "visible";
        }
        if ( renderPreviousButton() == "true" )
        {
                document.getElementById("prevBtn").style.visibility = "visible";
        }
        var entryStatus = retrieveDataValue( "cmi.entry" );
}

/**************************************************************************
** Faz a devida comunicação para finalizar a sessão, utilizando o arquivo
** APIWrapper.js
**************************************************************************/
function Terminate()
{
  if (!IntraNavigation)
  {
    storeDataValue( "cmi.completion_status", "completed" );
    storeDataValue( "cmi.exit", "" );
    terminateCommunication();
  }
}

/**************************************************************************
** Função para acessar o tempo inicial de sessão de um aluno
**************************************************************************/
function startTimer()
{
  startDate = new Date().getTime();
}

/**************************************************************************
** Função para acessar o tempo final de sessão de um aluno
```

```
**************************************************************************/
function computeTime()
{
  if ( startDate != 0 )
  {
    var currentDate = new Date().getTime();
    var elapsedSeconds = ( (currentDate - startDate) / 1000 );
    var formattedTime = convertTotalSeconds( elapsedSeconds );
  }
  else
  {
    formattedTime = "00:00:00.0";
  }
}

/**************************************************************************
** Função calcular o tempo total de sessão de um aluno
**************************************************************************/
function convertTotalSeconds(ts)
{
  var sec = (ts % 60);
  ts -= sec;
  var tmp = (ts % 3600);  //# of seconds in the total # of minutes
  ts -= tmp;              //# of seconds in the total # of hours
  sec = Math.round(sec*100)/100;
  var strSec = new String(sec);
  var strWholeSec = strSec;
  var strFractionSec = "";
  if (strSec.indexOf(".") != -1)
  {
    strWholeSec =  strSec.substring(0, strSec.indexOf("."));
    strFractionSec = strSec.substring(strSec.indexOf(".")+1, strSec.length);
  }
  if (strWholeSec.length < 2)
  {
    strWholeSec = "0" + strWholeSec;
  }
  strSec = strWholeSec;
  if (strFractionSec.length)
  {
    strSec = strSec+ "." + strFractionSec;
  }
  if ((ts % 3600) != 0 )
    var hour = 0;
  else var hour = (ts / 3600);
```

```
  if ( (tmp % 60) != 0 )
    var min = 0;
  else var min = (tmp / 60);
  if ((new String(hour)).length < 2)
    hour = "0"+hour;
  if ((new String(min)).length < 2)
    min = "0"+min;
  var rtnVal = hour+":"+min+":"+strSec;
  return rtnVal;
}

var teste = startTimer ();
```

Com os programas codificados (API.js, APIWrapper.js, API.asp e SCOFunction.js), deve-se inseri-los nos locais corretos.

Quando um aluno escolhe um determinado módulo (SCO) para cursar, deve-se anexar os programas APIWrapper.js e SCOFunctions na página HTML deste módulo.

Já o arquivo API.js é introduzido na página em que o aluno seleciona o curso, a disciplina e o módulo. Neste arquivo, é criado o objeto API_1484_11.

O programa API.asp é chamado pelo programa API.js

Têm-se abaixo as listagens de alguns programas e páginas criadas para testar as codificações dos programas das APIs, como a página para a escolha do curso, da disciplina e do módulo, como também as configurações básicas da base de dados. Esses arquivos são listados no trabalho já que nos programas SCOFunctions, APIWrapper.js, API.js e API.asp fazem referências a alguns métodos e funções das páginas e configurações citadas.

1. Arquivo: bd.asp (banco de dados)

```
<%
    Dim Conexao
    Dim Sql
```

```
Sub InicializarConexao(StringConexao, Usuario, Senha)
        Set Conexao = Server.CreateObject("ADODB.Connection")
        Conexao.Open StringConexao, Usuario, Senha
End Sub

Sub FinalizarConexao()
        Conexao.Close
        Set Conexao = Nothing
End Sub

Sub ExecutarSql(Sql)
        Conexao.Execute(Sql)
End Sub

Sub IniciarTransacao
        Conexao.BeginTrans
End Sub

Sub FinalizarTransacao
        Conexao.CommitTrans
End Sub

Sub CancelarTransacao
        Conexao.RollbackTrans
End Sub

Function RecuperarLista(Sql)
        Dim Rs
        Set Rs = Server.CreateObject("ADODB.Recordset")
        Rs.Open Sql, Conexao, 1, 1, 1
        RecuperarLista = Null
        If Not Rs.State = 0 Then
                If Not Rs.BOF And Not Rs.EOF Then
                        RecuperarLista = Rs.GetRows
                End If
        End If
        Set Rs = Nothing
End Function

InicializarConexao Application("StringConexao"), Application("Usuario"),
Application("Senha")

%>
```

2. lms.asp (página para a escolha do curso, disciplina e módulo)

```
<!--#include file="lib/cabecalho.asp"-->
<!--#include file="lib/bd.asp"-->
<!--#include file="lib/util.asp"-->

' os arquivos cabecalho.asp e util.asp são arquivos de configurações para o
' banco de dados e para a emissão de mensagem padrão para usuários
<%
        Const ID_ITEM = 0
        Const NOME_ITEM = 1
        Const EXTRA_ITEM = 2
        Dim I
        Dim IdAluno
        Dim NomeAluno
        IdAluno = Session("IdAluno")
        NomeAluno = Session("NomeAluno")
        Dim Grupos
        Dim Turmas
        Dim Disciplinas
        Dim Modulos
        Dim GrupoSelecionado
        Dim TurmaSelecionada
        Dim DisciplinaSelecionada
        GrupoSelecionado = CLng("0" & Request("Grupo"))
        TurmaSelecionada = CLng("0" & Request("Turma"))
        DisciplinaSelecionada = CLng("0" & Request("Disciplina"))
        Grupos = Null
        Turmas = Null
        Disciplinas = Null
        Modulos = Null
        Grupos = RecuperarLista("select id_grupo, nome from grupo order by nome")
        If GrupoSelecionado <> 0 Then
                Sql = " select t.id_turma, t.nome " & _
                                " from " & _
                                        " turma t " & _
                        " inner join aluno_turma at on at.id_turma = t.id_turma " & _
                                " where " & _
                                " at.id_grupo = " & GrupoSelecionado & " and " & _
                                        " at.id_aluno = " & IdAluno & _
                                " order by " & _
                                        " t.nome "
                                Turmas = RecuperarLista(Sql)
        End If
        If TurmaSelecionada <> 0 Then
                Sql = " select d.id_disciplina, d.nome " & _
                                " from " & _
```

```
                                        " disciplina d " & _
        " inner join alunos_disciplina ad on ad.id_disciplina = d.id_disciplina " & _
                                " where " & _
                                " ad.id_turma = " & TurmaSelecionada & " and " & _
                                " ad.id_grupo = " & GrupoSelecionado & " and " & _
                                        " ad.id_aluno = " & IdAluno & _
                                " order by " & _
                                        " d.nome "
                        Disciplinas = RecuperarLista(Sql)
        End If
        If DisciplinaSelecionada <> 0 Then
                Sql =   " select m.id_modulo, m.nome, m.link_arquivo " & _
                                " from " & _
                                        " módulo m " & _
                " inner join discip_modulo dm on dm.id_modulo = m.id_modulo " & _
                                " where " & _
                                        " dm.id_disciplina = " & DisciplinaSelecionada & _
                                " order by " & _
                                        " m.nome "
                                Modulos = RecuperarLista(Sql)
                End If
%>

<html>
<head>
<title>LMS</title>
        <! O programa menu popup.js faz o tratamento de janelas>
        <script language="JavaScript" src="lib/popup.js"></script>
        <script language="JavaScript" src="api.js"></script>
        <script language="JavaScript">
        /* Criaçao do objeto API_1484_11 */
        window.API = null;
        window.API_1484_11 = null;
        var modulos = [];
        <%
                If Not IsNull(Modulos) Then
                        For I = LBound(Modulos, 2) To UBound(Modulos, 2)
        %>
                        modulos["<% = Modulos(ID_ITEM, I) %>"] = ["<% =
Modulos(EXTRA_ITEM, I) %>"];
        <%
                        Next
                End If
        %>
```

```
function executarModulo()
        {
                if (document.getElementById("modulo").value == "")
                {
                        alert("Por favor, selecione um módulo para abrir.");
                        return;
                }
                API.sco = document.getElementById("modulo").value;
                document.getElementById("sco").src = "about:blank";
                if (document.getElementById("telaCheia").checked)
                {
        abrirPopupCentro(modulos[document.getElementById("modulo").value][
0], "SCO", screen.availWidth, screen.availHeight);
                }
                else
                        {
                                document.getElementById("sco").src =
modulos[document.getElementById("modulo").value][0];
                        }
                }

        window.onload = function()
                {
                        window.API = new APIImplementation();
                        window.API.url = "<% = Application("Url")
%>/api.asp?aid=<% = IdALuno %>&gid=<% = GrupoSelecionado %>&tid=<% =
TurmaSelecionada %>&did=<% = DisciplinaSelecionada %>";
                        window.API.sessionId = "<% = Session.SessionID %>";
                        window.API_1484_11 = window.API;
                }
        </script>

</head>

<body>
<h1 style="float: left">LMS - Área de Cursos</h1>
<h2 style="float: right">

<%
        Dim Hora
        Hora = Hour(Now)
        If Hora >= 06 And Hora < 12 Then
                Response.Write("Bom dia, ")
        ElseIf Hora >= 12 And Hora < 18 Then
                Response.Write("Bom tarde, ")
```

```
            Else
                Response.Write("Bom noite, ")
            End If
            Response.Write NomeAluno
        %>

        </h2>
        <form method="post" action="lms.asp">
        <table border="0" width="100%" height="80%" cellpadding="0"
cellspacing="0" style="clear: both; margin-top: 25px">
            <tr>
                <td width="25%" valign="top">
                    <p>Escolha o grupo, curso, disciplina e módulo:</p>
                    <p>
                    Grupo:<br>
                    <select name="Grupo" style="width: 70%"
onchange="document.forms[0].submit()">
                    <option value="">Selecione...</option>
        <%
                If Not IsNull(Grupos) Then
                    For I = LBound(Grupos, 2) To UBound(Grupos, 2)
        %>
                                <option value="<% =
Grupos(ID_ITEM, I) %>" <% If GrupoSelecionado = Grupos(ID_ITEM, I) Then
%>selected<% End If %>><% = Grupos(NOME_ITEM, I) %></option>
        <%
                    Next
                End If
        %>
                    </select>
                    </p>
                    <p>
                    Turma:<br>
                    <select name="Turma" style="width: 70%"
onchange="document.forms[0].submit()">
                    <option value="">Selecione...</option>
        <%
                If Not IsNull(Turmas) Then
                    For I = LBound(Turmas, 2) To UBound(Turmas, 2)
        %>
                    <option value="<% = Turmas(ID_ITEM, I) %>" <% If
TurmaSelecionada = Turmas(ID_ITEM, I) Then %>selected<% End If %>><% =
Turmas(NOME_ITEM, I) %></option>
        <%
                    Next
```

```
                End If
        %>
                        </select>
                        </p>
                        <p>
                        Disciplina:<br>
                        <select name="Disciplina" style="width: 70%"
onchange="document.forms[0].submit()">
                        <option value="">Selecione...</option>
        <%
                If Not IsNull(Disciplinas) Then
                        For I = LBound(Disciplinas, 2) To UBound(Disciplinas, 2)
        %>
                                <option value="<% = Disciplinas(ID_ITEM, I)
%>" <% If DisciplinaSelecionada = Disciplinas(ID_ITEM, I) Then %>selected<% End
If %>><% = Disciplinas(NOME_ITEM, I) %></option>
        <%
                        Next
                End If
        %>
                        </select>
                        </p>
                        <p>
                        Módulo:<br>
                        <select name="Modulo" id="modulo" style="width:
70%">
                        <option value="">Selecione...</option>
        <%
                If Not IsNull(Modulos) Then
                        For I = LBound(Modulos, 2) To UBound(Modulos, 2)
        %>
                        <option value="<% = Modulos(ID_ITEM, I) %>"><% =
Modulos(NOME_ITEM, I) %></option>
        <%
                        Next
                End If
        %>
                        </select>
                        </p>
                        <p>
                        <input type="checkbox" name="telaCheia"
id="telaCheia"> Tela Cheia
                        <p>
                        </p>
```

```
                        <input type="button" value="Abrir"
onclick="executarModulo()">
                        </p>
                        </td>
                        <td valign="top">
                                <iframe id="sco" src="about:blank" width="100%"
height="100%" frameborder="0" style="border: 1px solid black"></iframe>
                        </td>
                </tr>
        </table>
        </form>
        </body>
</html>
```

3. Global.asa

O arquivo global.asa pode ser utilizado para configuração de uma aplicação. Application_OnStart é um evento que ocorre quando uma aplicação é iniciada. Este arquivo deve estar na pasta raiz da aplicação.

```
<script language="VBScript" runat="server">
        Sub Application_OnStart
                Application("StringConexao") = "Provider=SQLOLEDB.1;Integrated
        Security=SSPI;Persist Security Info=False;Initial Catalog=LMS;Data
        Source=LAZARUS"
                Application("Usuario") = "sa"
                Application("Senha") = "test"
                Application("Url") = "http://localhost/lms"
        End Sub
</script>
```

Referências Bibliográficas

ADL. Advanced Distributed Learning. **SCORM Overview**. Disponível em: <http://www.adlnet.org/index.cfm?fuseaction=scormabt>. Acesso em: 17 de ago. de 2004.

ANIDO, Luis E. et al. Educational metadata and brokerage for learning resources. **Computers & Education**. v. 38, n. 4, p. 351-374, May 2002.

ARAÚJO, Moysés. **Educação a Distância e a Web Semântica:** Modelagem Ontológica de Materiais e Objetos de Aprendizagem para a Plataforma COL. São Paulo, 2003. Tese (Doutorado). Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo.

BARBEIRA, Jacinto; SANTOS, Arnaldo. Portugal. **Desenvolvimento de conteúdos normalizados para ambientes de e-learnirg**: um estudo de caso na Pt Inovação. Disponível em: <http://lsm.dei.uc.pt/ribie/docfiles/txt2003729182959paper-234.pdf >. Acesso em: 29 de ago de 2004.

BETTIO, Rafael W. **Avaliações Gráficas e Dinâmicas Aplicadas a Ambientes Virtuais de Aprendizagem.** Florianópolis, 2003. Tese (Mestrado). Departamento de Engenharia de Produção, Universidade Federal de Santa Catarina.

CAMPOS, Gilda H. B. 2003. **Relato de uma experiência**. Disponível em: <http://www.timaster.com.br/revista/artigos/main_artigo.asp?codigo=879&pag=1>. Acesso em: 12 de ago. 2004.

CASTRO, Manuel et al. Examples of Distance Learning Projects in the European Community. **IEEE Transaction on Education**. v. 44, n. 4, p. 406-412, November 2001.

COSCARELLI, Crislaine. 2004. **Objetos para aprender fazendo.** Disponível em: <http://www.universiabrasil.net/ead/materia.jsp?id=3025>. Acesso em: 01 de ago. 2004.

COSTA, Carlos A. de Oliveira. **Análise de Requisitos do Sistema E-learning de um Centro de Formação Profissional e Desenvolvimento de Protótipo de Demonstração**. Disponível em: <http://paginas.fe.up.pt/~ee95135/relatorio_final.pdf>. Acesso em: 30 de ago. de 2004.

EDUSOURCECANADA. Canadá. **Canadian Network of Learning Object Repositories.** Disponível em: <http://www.edusource.ca/english/what_eng.html>. Acesso em: 01 de ago. 2004.

EDUWORKS. **All About Learning Objects.** Disponível em: <http://www.eduworks.com/LOTT/tutorial/learningobjects.html>. Acesso em: 20 de jun. 2004.

EL SADDIK, Abdulmotaleb; FISCHER, Stefan; STEINMETZ, Ralf. Reusable Multimedia content in Web-Based Learning Systems. **IEEE Multimedia**. v. 8, n. 3, p. 30-38, July-September 2001.

FAPESP, Fundação de Amparo à Pesquisa do Estado de São Paulo. **Projeto Tidia**. Disponível em: <http://www.tidia.fapesp.br/portal>. Acesso em: 13 de nov. de 2004.

FLANAGAN, David. **JavaScript:** o guia. 4. ed. Porto Alegre: Bookman, 2004.

JORGENSEN, David. **Desenvolvendo Serviços Web .NET com XML.** Rio de Janeiro, Alta Books, 2002.

LTSC, Learning Technology Standards Committee of The IEEE. New York, 2002. **Learning Object**. Disponível em: <http://ltsc.ieee.org/wg12/index.html>. Acesso em: 02 de ago. 2004.

LONGMIRE, W. **A primer on Learning Objects**. 2000. Disponível em: <http://www.learningcircuits.org/2000/mar2000/Longmire.htm>. Acesso em: 02 de ago. de 2004.

LUCENA. Beto. 2003. **Novas Tecnologias no E-learning: Desafios e Oportunidades para o Design.** Disponível em: <http://www.abed.org.br/publique/cgi/cgilua.exe/sys/start.htm?UserActiveTemplate=1por&infoid=883&sid=135>. Acesso em: 16 de ago. de 2004.

MARTINI, Júlio César. 2002. **Comparação entre as Linguagens ASP e PHP**. Disponível em: < http://www.imasters.com.br/artigo.php?cn=28&cc=44>. Acesso em: 07 de julho de 2005.

MEC, Ministério da Educação. Brasília. **Projeto Rived:** Red Internacional Virtual de Educación. Disponível em: <http://rived.proinfo.mec.gov.br>. Acesso em: 01 de ago. 2004.

MICROSOFT. **Active Server Pages**. Disponível em: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/activeservpages.asp>. Acesso em: 07 de julho de 2005.

MORAN, José Manuel; MASETTO, Marcos; BEHRENS, Marilda. **Novas Tecnologias e Mediação Pedagógica**. São Paulo, Papirus, 2000.

NUNES, César A. A. 2002. **Criação, produção e uso de objetos de aprendizagem**. Disponível em: <http://www.abed.org.br/congresso2002/ppcn.ppt >. Acesso em: 15 de ago 2004.

SANTOS, Osvaldo A.; RAMOS, Fernando M. S. Proposal of a framework for Internet based licensing of learning objects. **Computers & Educations**. v. 42, n. 3, p. 227-242, April 2004.

SILVEIRA, R. M. et al. **COL – Ferramenta de apoio ao ensino**. LARC – Laboratório de Arquitetura e Redes de Computadores. PCS – Departamento de Engenharia de Computação e Sistemas Digitais, EPUSP – Escola Politécnica da USP, 2004.

SONNTAG, Alexandre. **Padrões de conteúdo para o ensino a distância**. Disponível em: <http://www.elearningbrasil.com.br/congresso/2002/posevento/resumo/d17s7/alexandre.asp.>. Acesso em: 13 de ago. 2004.

SOSTERIC, M.; HESEMEIER, S. Canadá, 2002. **When is a Learning object not an Object:** A first step towards a theory of learning objects. Disponível em: <http://www.irrodl.org/content/v3.2/soc-hes.html>. Acesso em: 02 de ago. 2004.

STEINACKER, Achim; GHAVAM, Amir; STEINMETZ, Ralf. Metadata Standards for Web-Based Resources. **IEEE Multimedia**. v. 8, n. 1, p. 70-76, March 2001.

TAROUCO, Liane; FABRE, Marie J. M.; DUTRA, Renato. 2003. **Interoperabilidade entre objetos educacionais e sistemas de gerenciamento de aprendizagem**. Disponível em: <http://www.cinted.ufrgs.br/ppt/interopObjEduc/index.htm>. Acesso em: 15 de ago. 2004.

UFRGS, Universidade Federal do Rio Grande do Sul. **Projeto Cesta**. Disponível em: <http://www.cinted.ufrgs.br/CESTA/cestadescr.html>. Acesso em: 02 de ago. 2004.

UNIVERSIABRASIL. **Prêmio ABED para Objetos e Recursos de Aprendizagem**. Disponível em: < http://www.universiabrasil.net/premioabeduniversia/#>. Acesso em 03 de ago. 2004.

W3C, World Wide Web Consortium. **Web Services Activity Statement**. Disponível em: <http://www.w3.org/2002/ws/Activity>. Acesso em: 24 de dez. 2004.

WEBAULA. **Portal e-learning.** Disponível em: <http://portal.webaula.com.br>. Acesso em 12 de ago. 2004.

WILEY, David A. **Connecting learning objects to instructional design theory:** A definition, a metaphor, and a taxonomy. Disponível em: <http://reusability.org/read/chapters/wiley.doc>. Acesso em: 02 de ago. 2004.

ZAINA, Luciana A. Martinez. **Acompanhamento do Aprendizado do Aluno em Cursos a Distância Através da Web**: Metodologias e Ferramenta. São Paulo, 2002. Dissertação (Mestrado). Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo.

Elaine Pasqualini

Proposta de Adaptação do Sistema de Gerenciamento de Aprendizagem COL ao Modelo de Padronização SCORM

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia.

São Paulo
2005