Caio Moritz Ronchi

Estudo do padrão SCORM e proposta de implementação

Florianópolis - SC

2007

Caio Moritz Ronchi

Estudo do padrão SCORM e proposta de implementação

Orientador:
Ricardo Azambuja Silveira

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis - SC

2007

Trabalho de conclusão de curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Prof^{o}. Dr. Ricardo Azambuja Silveira
Orientador

Prof^{a}. Dra. Silvia Modesto Nassar
Universidade Federal de Santa Catarina

Prof^{a}. Bel. Julia Marques Carvalho da Silva
Universidade do Vale do Itajaí

Agradecimentos

Aos meus pais e irmãs, por serem quem são na minha vida.

A Thânia, que ilumina minha existência.

Ao meu orientador, Ricardo Silveira, por me incentivar a conhecer o padrão SCORM.

Aos membros da banca, por dedicarem seu tempo a estudar e avaliar este trabalho.

"*Os laços que prendem a alma ao corpo não se rompem senão aos poucos, e tanto menos rapidamente quanto mais a vida foi material e sensual.*"

O Livro dos Espíritos, n.º 155

Resumo

Este trabalho tem como objetivo principal estudar de forma abrangente os conceitos relacionados aos Objetos de Aprendizagem. Na primeira parte são apresentados os principais conceitos relacionados aos conteúdos criados sobre esse paradigma. Em seguida é desenvolvido um estudo sobre a especificação do padrão SCORM (Modelo de Referência para Objetos de Conteúdo Compartilháveis), uma definição de como os conteudistas podem criar conteúdos reutilizáveis e rastreáveis dentro de um LMS em conformidade com tal padrão. O trabalho aborda, ainda, a implementação do LMS Moodle para o padrão SCORM, incluindo uma visão geral de seu código-fonte e uma análise crítica de seus principais problemas de acordo com as melhores práticas da engenharia de software. Por fim, é proposta uma tecnologia para a implementação de um LMS capaz de suportar conteúdos no padrão SCORM: um *framework* MVC. Três *frameworks* escritos em diferentes linguagens de programação são analisados, e um deles é escolhido, sob os aspectos analisados, como a melhor opção para o desenvolvimento do sistema.

Palavras-chave: Objetos de Aprendizagem, SCORM, Moodle, *Frameworks* MVC

Abstract

This work aims to be a comprehensive study about Learning Objects and related concepts. The first part is dedicated to present the main concepts related to creation of content compliant to the Learning Objects approach. The second part provides a study about SCORM (Sharable Content Object Reference Model), which is a definition of how content designers can build trackable sharable content to be used in a SCORM-compliant LMS. Also, the work studies the Moodle LMS implementation for the SCORM standard, including an overview of its source code and a critical analysis of the main problems found in it, according to the best practices of software engineering. Last, the author proposes a technology to build an LMS that's capable of supporting SCORM compliant content: an MVC framework. Three frameworks written in three different programming languages are compared, and one of them is chosen as the best option for the system construction.

Keywords: Learning Objects, SCORM, Moodle, MVC Frameworks

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

1 Introdução

Vivemos em um mundo de constantes transformações. Parece ser da natureza do homem transformar o mundo à sua volta. Estamos inseridos na era da sociedade do conhecimento e das redes sociais. Há duas décadas, no entanto, isso parecia muito distante, ou sequer imaginado. Uma tecnologia inovadora pode dar à luz dezenas de outras: o computador deu origem ao processamento de dados e informações; a internet aproximou as pessoas e tornou o mundo mais dinâmico. A internet poderia não ter existido sem a presença do computador. A tecnologia de hoje será o suporte para as novas tecnologias de amanhã.

Com a educação essas transformações são similares. A educação formal, restrita à sala de aula, ainda continua firme por todo o mundo, mas já há muitos anos vem sendo acompanhada por outro tipo de educação, livre das barreiras físicas da sala de aula: a Educação a Distância (EAD). Saber onde e quando a EAD surgiu pode não ser o mais importante. Saber a razão de seu surgimento e porque continua a crescer, no entanto, nos ajuda a aprimorá-la.

No Brasil, especificamente, conhece-se a EAD desde a época da educação "por correspondência". Por que será que as pessoas optavam por esse tipo de meio? Pode-se pensar em alguns motivos que certamente observamos ainda hoje:

- Falta de tempo para freqüentar um curso presencial
- Gosto por aprender sozinho / falta de gosto pela sala de aula
- A faculdade/escola mais próxima pode ficar a quilômetros de distância

A EAD "por correspodência" baseou-se em tecnologias tais como "correio" e, possivelmente, o telefone. Naquela época o computador ainda não se fazia presente em nossos lares. Hoje tem-se outra situação: as crianças já nascem em lares com um ou mais computadores, dependendo de sua faixa social. Elas sabem mexer no computador, sabem se comunicar com os amigos da escola pela internet etc. A EAD também se beneficiou dessas tecnologias, utilizando a internet como meio de entrega de conteúdo, e as *aplicações web* como plataformas para gerência de cursos e aprendizes.

E qual a relevância da EAD para o nosso país?

1.1 Justificativa

> "Educação a Distância surgiu no Brasil, oficialmente, em dezembro de 1996, instituída pela Lei n. 9.394, com mais de cem anos de atraso em relação a outras iniciativas mundiais. Acabamos de completar, portanto, dez anos de existência legal. Mas essa cronologia diz respeito apenas à EAD de caráter oficial, antes disso, sua aplicação já era intensa e abrangente. Em todos os cantos do país, existiam cursos de ensino ou aprendizagem por correspondência."(ABED, 2007)

Não apenas os cursos na modalidade a distância vêm se profilerando no país, mas também os investimentos das iniciativas privadas e públicas.

> "Para o ano de 2007 foram previstos R$ 167 milhões no programa UAB, valor seis vezes maior do que o aplicado em 2006, o que demonstra a valorização crescente da aplicação dos métodos de Educação a Distância pelo Governo Federal."(ABED, 2007)

Entre as diversas tecnologias que suportam a EAD brasileira (e mundial) estão os Ambientes Virtuais de Aprendizagem (*Learning Management Systems*, referenciados daqui em diante apenas como LMS), softwares que permitem, entre outras funções, que o professor publique conteúdo para seus aprendizes e avalie o aprendizado por meio de testes desenvolvidos com o auxílio do mesmo software.

Os LMSs costumam ser ricos em funcionalidades e focados em usabilidade, principalmente para pessoas sem muita experiência na utilização de outros tipos de software. A criação e publicação de conteúdo também é facilitada e encorajada dentro desses sistemas. Com um clique o professor cria um novo documento HTML, e o confecciona utilizando um editor de textos embutido, ao estilo Microsoft Word. Quando considera seu conteúdo pronto, ele o salva e o publica. Seus aprendizes, então, passam a ter acesso a esse material. É simples para o professor criar conteúdo, pois ele não precisa sair do LMS. No entanto, essa prática cria um sério problema.

O problema é que o professor está amarrando seu conteúdo à plataforma LMS em uso. Todas as facilidades que o LMS dá para criar conteúdo tornam-se *dificuldades* quando se necessita transportar esse conteúdo para um outro LMS. Não é possível prever como esse mesmo conteúdo será tratado em outro ambiente, ou sequer prever se será possível visualizá-lo.

É necessário, portanto, que nos preocupemos com a forma como *pretendemos manter nossos conteúdos independentes de plataforma LMS*. Visando resolver esse problema tem surgido,

há vários anos, padrões internacionais para produção e compartilhamento de conteúdo instrucional. Dentre esses padrões, o padrão *de facto* é o SCORM (WILEY, 2001). Esse padrão prevê não apenas um forma para compartilhamento de conteúdo, mas define ainda formas de monitorar os caminhos que o aprendiz percorre dentro do conteúdo: quais suas deficiências em termos de conhecimento, qual o tempo que leva para alcançar determinados objetivos, entre outras informações.

É um temor do autor, portanto, que o Brasil, e principalmente as intituições públicas de ensino, estejam criando conteúdos específicos a um determinado LMS, em vez de torná-los independentes de plataforma. Uma vez que a tecnologia está sempre criando soluções melhores para resolver os mesmos problemas, é possível e provável que surjam novas gerações de LMS, e que as instituições precisem atualizar seus sistemas, não apenas para uma versão superior, mas possivelmente para um outro fabricante. Uma vez que os conteúdos sejam projetados e desenvolvidos respeitando-se padrões internacionais, e uma vez que os LMSs suportem esses padrões, tem-se garantida a portabilidade do conteúdo produzido. Isso significa investimento mais consciente do dinheiro público, evitando futuros gastos com retrabalho.

1.2 Objetivos

1.2.1 Objetivo Geral

Propor novas tecnologias para a implementação de um ambiente virtual de ensino-aprendizagem que suporte a utilização de objetos de aprendizagem desenvolvidos de acordo com o padrão SCORM, e assim aumentar a quantidade de opções existentes no atual mercado de LMSs (AVA).

1.2.2 Objetivos Específicos

- Investigar as principais visões a respeito do conceito de Objetos de Aprendizagem.
- Investigar as potencialidades do padrão SCORM.
- Levantar quais alternativas, construídas sob o paradigma do software livre, existem para a criação de cursos baseados em conteúdo SCORM.
- Analisar e documentar a implementação do módulo SCORM implementado pelo LMS Moodle.

- Analisar alguns *frameworks* para desenvolvimento web, e a partir dessa análise propor a ferramenta que possua as características mais adequadas para suportar a criação e manutenção de um sistema LMS compatível com o modelo SCORM.

1.3 Metodologia

Este trabalho foi conduzido como um estudo exploratório: seu escopo foi sendo modificado diversas vezes em função das descobertas feitas pelo autor. Se em sua forma final ele se apresenta como uma "história" com começo, meio e fim, é porquê houve um árduo trabalho para que todas as suas partes pudessem se encaixar de forma lógica.

No início do trabalho foi considerada a implementação de um pequeno LMS que suportasse o padrão SCORM. Esse sistema permitiria, através de um Modelo de Aluno, verificar qual o estilo de aprendizado do aprendiz, e, assim, tentar facilitar seus estudos, modificando suas estratégias pedagógicas de acordo com o *feedback* recebido do aprendiz.

Esse trabalho, no entanto, mostrou-se demasiado complexo para ser realizado no prazo proposto. Essa complexidade não partia nem das técnicas de inteligência artificial necessárias para alimentar o Modelo do Aluno, e nem da dificuldade de se construir uma aplicação web rapidamente: foi a própria complexidade do padrão SCORM que acabou por invialibizar a idéia inicial.

Uma vez que a própria complexidade do padrão SCORM parecia ser um fator que inviabilizaria a finalização do trabalho em prazo aceitável, optou-se por utilizar o único LMS sob licença livre que implementa o SCORM em sua totalidade: o Moodle. O autor dedicou então, um longo tempo ao estudo do código-fonte dessa implementação. Esse código, embora aparentemente bem-estruturado, carecia de documentação adequada. De todas as suas funções, estima-se que 90% delas não possuíam documentação de sua finalidade e nem de seus parâmetros de entrada e de saída. Se o módulo não apresentasse nenhum problema de funcionamento esses detalhes poderiam ter sido desconsiderados, mas não era o caso: o autor verificou, dentro do software de gerência do projeto Moodle (MOODLE-TRACKER, 2007), diversas notificações de problemas. O principal desenvolvedor do módulo, Roberto Pinna, parecia estar com pouco tempo para dedicar o projeto, e não chegou a responder a dois emails enviados pelo autor, oferecendo ajuda para manutenção do módulo e documentação.

O autor considerou estranho e perigoso que todas as instituições de ensino que dependem de software livre no país possuam à sua disposição uma única implementação do padrão SCORM (aquela fornecida pelo Moodle), e que a mesma não esteja recebendo tanta atenção quanto

poderia por parte de alguns dos mantenedores Moodle.

Partiu-se, então, do pressuposto de que uma alternativa à implementação do Moodle é necessária. Analisou-se, então, 3 *frameworks* de desenvolvimento web, os quais garantem ao desenvolvedor maior agilidade e melhor infra-estrutura para criação de testes de software. O resultado dessa análise, em verdade, auxilia a escolha de um framework não só para a implementação do padrão SCORM, mas de muitas outras aplicações baseadas na web.

O framework escolhido, Ruby on Rails, já é bastante popular em países como Estados Unidos, mas sua adoção no Brasil segue lenta.

1.4 Estrutura do documento

No capítulo 2 são relacionadas as principais teorias criadas a respeito de Objetos de Aprendizagem; no capítulo 3 é abordado, de maneira abrangente, o padrão SCORM; no capítulo 4 é apresentado um estudo da implementação do módulo SCORM do Moodle; no capítulo 5 é apresentada uma comparação entre *frameworks* de desenvolvimento web, e um desses é escolhido como melhor solução para implementação de um LMS com suporte de qualidade ao padrão SCORM; no capítulo 6 o trabalho é concluído.

2 *Objetos de Aprendizagem*

2.1 Definições

Para Jacobsen (2001), um Objeto de Aprendizagem (referenciado a partir de agora simplesmente como OA) é "uma coleção discreta e reusável de conteúdo utilizado para apresentar e suportar um único objetivo de aprendizado".

> "Objetos de Aprendizagem são elementos de um novo tipo de instrução baseada em computador, sustentado pelo paradigma de orientação a objetos da Ciência da Computação [...] Esta é a idéia fundamental por trás dos objetos de aprendizagem: designers instrucionais podem criar pequenos (em relação ao tamanho de um curso completo) componentes instrucionais que podem ser reutilizados um grande número de vezes em diferentes contextos de aprendizagem. Além disso, objetos de aprendizagem são, em geral, entendidos como entidades digitais entregues através da Internet, o que implica que qualquer número de pessoas pode acessá-los e utilizá-los simultaneamente."(WILEY, 2001)

Ao mesmo tempo que o tamanho do objeto promove sua reutilização, fica claro que, uma vez que diferentes pessoas compartilhem seus objetos em diferentes cursos, localizando tais objetos em um mesmo repositório, a atualização do conteúdo de um desses objetos implicará na atualização do mesmo em todos os diferentes contextos nos quais ele possa estar sendo utilizado.

Quando professores têm um primeiro contato com material instrucional, eles geralmente o quebram em suas partes constituintes. Em seguida eles reagrupam essas partes de modo a suportar seus objetivos instrucionais individuais (WILEY, 2001 apud REIGELUTH; NELSON, 1997). Com base nesse preceito, Wiley detalha ainda mais a forma prática do reuso de objetos de aprendizagem:

> "[...] Isso sugere uma razão pela qual componentes instrucionais reusáveis, ou objetos de aprendizagem, podem promover benefícios instrucionais: se os professores recebessem recursos como componentes individuais, esse passo inicial de quebra poderia ser eliminado, potencialmente aumentando a velocidade e eficiência da construção de conteúdo instrucional."(WILEY, 2001)

As definições destes e de outros autores, embora claras e objetivas, costumam ser um pouco teóricas. A definição de objeto de aprendizagem dada pelos autores do padrão SCORM, no entanto, é muito mais concreta (mas nem por isso mais correta) e voltada claramente para a visão de objetos de aprendizagem como pequenos componentes instrucionais dependentes, exclusivamente, de uma outra plataforma, o *Learning Management System* (LMS). Para maiores detalhes sobre o SCORM vide o Capítulo 3.

2.2 Origens

De acordo com Jacobsen o termo "objeto de aprendizagem" apareceu no início da década de 1990, e costuma-se creditar a autoria do termo a Wayne Hodgins.

> "Em 1992 Wayne estava vendo um de seus filhos brincar com peças de Lego, enquanto ele refletia sobre alguns problemas relacionados a estratégias de ensino. Wayne concluiu, naquele momento, que a indústria precisava de peças (blocos) para conteúdos de aprendizagem estilo *plug and play*. Ele chamou tais blocos de objetos de aprendizagem" (JACOBSEN, 2001).

Ainda de acordo com Jacobsen, no período entre 1992 e 1995 diversos grupos não relacionados passaram a trabalhar com o conceito de objetos de aprendizagem, entre eles o Learning Object Metadata Group (LOM). Entre o período de 1994 e 1996, segundo o autor, foi a vez de grupos como IEEE, IMS e ARIADNE iniciarem seus trabalhos na área de objetos de aprendizagem. Tom Kelly e Chuck Barrit lançaram, em 1998, o *white paper* da Cisco Systems sobre Objetos de Aprendizagem Reusáveis.

2.3 Motivação

Uma das grandes motivações para a utilização de objetos de aprendizagem reusáveis é o custo de produção (DOWNES, 2001). Se considerarmos que uma aula sobre o algoritmo de ordenação *Bubble Sort* (um conteúdo que não evolui através dos anos), por exemplo, custa mil reais para ser produzida por uma instituição, o compartilhamento da mesma aula por mil instituições transformaria esse investimento em um real para cada instituição. Por outro lado, se cada uma dessas mil instituições produzisse seu próprio conteúdo, o custo somado seria de um *milhão* de reais.

2.4 Metáforas relacionadas

Desde o seu surgimento, a comunidade em torno dos objetos de aprendizagem tem utilizado metáforas para explicar o conceito de objetos de aprendizagem para os não-iniciados (WILEY, 2001). Entre as metáforas ligadas a objetos de aprendizagem encontradas na literatura pode-se destacar as seguintes.

2.4.1 Lego

Uma das metáforas mais famosas é a analogia do brinquedo Lego: criar pequenos pedaços de instrução (a peça do Lego propriamente dita) que podem ser construídos (grudados uns nos outros) em uma estrutura instrucional (castelo) e reusados em outras estruturas instrucionais (uma espaço-nave, por exemplo) (WILEY, 2001). Wiley considera um problema o fato de essa metáfora ter, segundo ele, ganhado vida própria, pois essa forma de explicar o conceito de objetos de aprendizagem acabou tornando-se regra até para os profissionais mais importantes da área.

Wiley aponta para o perigo da comparação de objetos de aprendizagem com o brinquedo Lego:

- Todas as peças Lego são combináveis entre si
- As peças do Lego podem ser combinadas de qualquer maneira que se desejar
- As peças do Lego são tão divertidas que até mesmo crianças podem combiná-las

Para Wiley, um sistema de objetos de aprendizagem com essas três propriedades *não pode produzir* nada mais *instrucionalmente útil* do que as peças de Lego podem.

2.4.2 Átomo

Uma metáfora mais completa para explicar o conceito de objeto de aprendizagem é a comparação com o átomo. O átomo é um pedaço de matéria extremamente pequeno que, combinado com outros átomos, forma (possivelmente) tudo que a ciência conhece. Para Wiley, essa metáfora é superior à do LEGO nas seguintes maneiras:

- Os átomos não são todos combináveis entre si

- Átomos podem apenas ser combinados em certas estruturas prescritas pela sua própria estrutura interna
- Um certo grau de treinamento é necessário para que se possa combinar átomos

2.5 Granularidade

Assim como há diversas definições, de diferentes autores, para o conceito de objetos de aprendizagem, a maioria dessas definições não explicita qual a granularidade (tamanho) necessária para um objeto, de modo que ele possa manter-se reusável em diferentes contextos instrucionais. Por se tratar de uma área relativamente nova do conhecimento, a construção de objetos de aprendizagem ao longo dos anos deve gerar respostas mais precisas no que diz respeito à granularidade desses componentes.

Não obstante, no trabalho de Holzinger, Smolle e Reibnegger, os membros do projeto VMC-Graz (VMC-GRAZ. . . , 2007) descrevem sua experiência prática com objetos de aprendizagem, onde resolveram suas questões de granularidade da seguinte maneira: todos os OA produzidos podem ter qualquer granularidade dentro da duração máxima de 45 minutos. Os OAs do VMC-Graz formam *unidades atômicas*, as quais são agrupadas em unidades de lição (em outras palavras aulas de duração de 45 minutos), grupos temáticos (tópicos, temas) e módulos. O OA não possui restrições técnicas em termos da sua quantidade de dados. Ou seja, um objeto é composto por outros, menores, que, por sua vez, são compostos por outros, e a menor unidade possível é o módulo.

2.6 Reusabilidade

Outro ponto de discordância entre os autores diz respeito ao próprio conceito de reusabilidade de objetos de aprendizagem. Para alguns autores o reuso está relacionado a diferentes instituições utilizando os mesmos conteúdos, enquanto para outros o reuso está mais próximo da idéia de recombinar diferentes objetos de modo a formar um novo objeto, maior e mais completo.

Seguindo a linha de pensamento da metáfora do átomo, por exemplo, WILEY afirma que, de um ponto de vista construtivista, os objetos de aprendizagem devem ser internamente contextualizados até certo grau - um grau que promova sua contextualização (combinação) com um conjunto limitado de outros objetos de aprendizagem, enquanto, simultaneamente, previne sua combinação com outros objetos de aprendizagem.

2.7 Taxonomia

Para melhor teorizar as questões de reusabilidade inerentes a objetos de aprendizagem, Wiley (2001) propõe uma taxonomia para os objetos, na qual os identifica em cinco tipos principais. Sua definição, posteriormente, foi adotada em um caso real e expandida por Holzinger, Smolle e Reibnegger (2005).

- *OA fundamental*: pode incluir como conteúdo uma imagem (JPEG, PNG etc.), um documento (DOC, TEX, PDF etc.), um filme (MPEG, AVI etc.).
- *OA de combinação fechada*: um vídeo acompanhado de áudio, por exemplo.
- *OA de combinação aberta*: um link externo para uma página web, combinando dinamicamente, por exemplo, arquivos JPEG e QuickTime combinados com material textual de fonte externa.
- *OA de apresentação produtiva*: um *applet* Java, por exemplo.
- *OA de instrução produtiva*: *neste item ambos os autores citaram um terceiro e não propuseram uma definição clara para o autor deste trabalho*.

2.8 Metadados

Metadados são importantes no contexto de OA pois possibilitam aos LMSs implementar e prover mecanismos de busca adequados aos metadados utilizados por um determinado repositório de OAs. Não se deve, no entanto, associar metadados a um objeto de forma aleatória, pois de tal modo perde-se completamente a interoperabilidade desse objeto com outros sistemas.

O desenvolvimento mais relevante na área de metadados foi a finalização, em 2002, do padrão *IEEE LTSC Learning Object Metadata* (LOM). A tecnologia interna mais utilizada para troca de instâncias LOM é baseada em XML (DUVAL, 2004).

Deve-se, de maneira geral, obedecer a especificação de metadados provida por alguma entidade padronizadora de objetos de aprendizagem (vide próxima seção).

2.9 Padrões abertos para criação de objetos

O objetivo principal de muitas iniciativas de padrões abertos, em qualquer área tecnológica, é tornar-se não apenas o padrão *de jure*, mas também o padrão *de facto* (DUVAL, 2004). Enquanto um padrão *de jure* é uma especificação formal estabelecida por um órgão oficial de padronização (como o ISO ou o IEEE), um padrão *de facto* é uma especificação criada por uma empresa ou entidade independente, cuja aceitação e disseminação pelo mercado, mesmo sem nenhum plano formal, a transforma em um padrão (ROSSI, 2003).

Os conceitos de "padrões abertos" e "código aberto" não devem ser confundidos. Padrões abertos têm como foco tornar publicamente disponível a sua especificação, construída em um processo aberto, de forma que qualquer parte interessada possa influenciar a evolução daquela especificação e produzir produtos e serviços baseados nela (DUVAL, 2004). O desenvolvimento "código aberto", por sua vez, é baseado na idéia de que o código-fonte de produtos e ferramentas também é feito público, de modo que as partes interessadas podem modificá-lo da forma que entenderem, uma vez respeitada a licença definida pelo autor.

No mundo do *e-learning*, há uma hierarquia de organizações que produzem e aprovam especificações, as quais podem ser divididas em dois níveis, sendo o nível 1 o de maior autoridade (DUVAL, 2004):

- *Nível 1*: Representado por organizações oficiais como IEEE LTSC e CEN/ISSS WSLT, as quais tem a obrigação explícita de atingir as necessidades e requisitos do domínio como um todo, e manter um processo justo e aberto para alcançar tais objetivos. É por isso que rascunhos dos padrões são disponibilizados ao público em estados iniciais do processo e ao longo dele: para que a comunidade possa influenciar o desenvolvimento do padrão.

- *Nível 2*: Representado por consórcios como AICC (*Aviation Industry CBT (Computer-Based Training) Committee*), IMS, ADL (*Advanced Distributed Learning*) e ARIADNE (*Alliance for Remote Instructional Authoring and Distribution Networks for Europe*), são responsáveis pela *produção de especificações*. Esses documentos técnicos são baseados em um processo interno, para que atinjam as necessidades e requisitos dos membros da organização. No entanto, tais especificações não são padrões, pois as mesmas não precisam levar em conta os requisitos e necessidades do domínio educacional, incluindo educação escolar e acadêmica, treinamento corporativo e militar, aprendizagem formal e informal etc.

O padrão abordado com profundidade neste trabalho é o SCORM, desenvolvido a partir da

especificação da ADL.

3 SCORM

A edição atual do padrão SCORM (SCORM 2004 3ª Edição) foi lançada em resposta a melhorias identificadas pelo time de técnicos da ADL e pela comunidade ADL como um todo, bem como a atualizações em especificações e trabalhos sobre padrões desenvolvidos desde que o SCORM 2004 2ª Edição foi lançado, em julho de 2004 (ADL, 2006b).

> A ADL emprega um esforço estruturado, adaptativo e colaborativo entre os setores público e privado para desenvolver os padrões, ferramentas e conteúdo de aprendizagem para o ambiente de aprendizagem do futuro. (...) Implementar a iniciativa ADL requer colaboração dentro do Departamento de Defesa (DoD) dos Estados Unidos da América e do governo federal como um todo, bem como colaboração da indústria, academia, estados e entidades locais. (ADL..., 2007).

A estratégia da ADL é (ADL..., 2007):

- Trabalhar próxima ao governo, indústria e academia para promover especificações e padrões internacionais comuns e abertos que possibilitem o reuso e a interoperabilidade de conteúdo instrucional
- Promover ampla colaboração que satisfaça necessidades comuns
- Melhorar a performance com tecnologias emergentes de aprendizagem
- Promover um processo coordenado de implementação com incentivos para mudanças organizacionais e culturais

SCORM é a sigla para "Sharable Content Object Reference Model" ou, em português, "Modelo de Referência para Objetos de Conteúdo Compartilhável". Esse padrão descreve um "Modelo de Agregação de Conteúdo" e um "Ambiente de Tempo de Execução" para que objetos instrucionais suportem instrução adaptativa baseada nos objetivos, preferências e performance anterior do aprendiz, entre outros fatores. O padrão descreve, ainda, um modelo de "Seqüenciamento e Navegação" para a apresentação dinâmica de conteúdo baseado nas necessidades do aprendiz (ADL, 2006b).

Os pilares do padrão SCORM são os seguintes:

- Acessibilidade: a habilidade de alocar e acessar componentes instrucionais de uma localização remota e entregá-los para muitos outros destinos.
- Adaptabilidade: a habilidade de modificar uma instrução de acordo com necessidades individuais e organizacionais.
- Rentabilidade: a habilidade de se aumentar a eficiência e a produção ao reduzir o tempo e o custo envolvidos na entrega da instrução.
- Durabilidade: a habilidade de acompanhar a evolução e mudança de uma tecnologia sem arcar com reprojeto, reconfiguração ou recodificação.
- Interoperabilidade: a habilidade de capturar componentes desenvolvidos em um local com um conjunto de ferramentas ou plataforma e utilizá-los em outro local com um diferente conjunto de ferramentas ou plataforma.
- Reusabilidade: a flexibilidade para se incorporar componentes instrucionais em múltiplos contextos.

3.1 Modelo de Agregação de Conteúdo

O livro CAM (Content Aggregation Model) descreve os componentes SCORM utilizados em um objeto de aprendizagem, como empacotá-los para utilização de sistema para sistema, como descrever esses componentes para possibilitar a busca e descoberta e como definir o seqüenciamento para os componentes (ADL, 2006a).

3.1.1 Modelo de Conteúdo

O modelo de conteúdo define as unidades de conteúdo que um pacote SCORM suporta:

- Recurso (Asset): a unidade mínima de conteúdo. Exemplos de Recurso são arquivos de texto, imagem, som ou qualquer outro pedaço de dados que possa ser renderizado por um cliente Web e apresentado ao aprendiz. Um Recurso pode ser descrito por metadados, o que possibilita sua procura e descoberta dentro de repositórios.

- Objeto de Conteúdo Compartilhado (SCO): Um SCO é uma coleção de um ou mais Assets que representam um único recurso de aprendizagem inicializável. Um SCO representa o menor nível de granularidade para um recurso de aprendizagem que deseja ser rastreado pelo LMS.

- Atividades: toda e qualquer unidade significativa de instrução. Apesar do nome uma atividade não implica no envio de feedback ao LMS. Atividades podem ser compostas por recursos, SCOs, outras atividades (sendo então chamadas de agregados) ou uma combinação de todos esses.

- Organização de conteúdo: é uma representação ou mapa que define o uso pretendido de um conteúdo por meio de Atividades. Essa organização pode ser vista como uma árvore, onde a raíz é a Organização, os ramos são os Ítens e as folhas são os recursos. O LMS fica responsável por ler essa estrutura e traduzi-la em um menu navegacional. A vantagem de se ter a organização definida externamento ao conteúdo diz respeito ao reuso: um conteúdo auto-contido, sem ligações externas, é mais reusável do que outro que possui ligações explícitas a outros recursos.

- Agregação de conteúdo: esse conceito pode ser utilizado para descrever a ação de se agregar um conjunto de objetos de conteúdo relacionados para que possam ser aplicados como uma instrução. O termo pode ser ainda utilizado para descrever o pacote físico de conteúdo.

3.1.2 Modelo de Empacotamento

O propósito do Pacote de Conteúdo (Content Package) é prover uma forma padronizada para a troca de conteúdo de aprendizagem entre diferentes sistemas e ferramentas. O pacote de conteúdo também provê um local para se declarar a estrutura (ou organização) e o comportamento esperado de uma coleção de conteúdos de aprendizagem (ADL, 2006a).

Atualmente há duas maneiras de se testar um pacote SCORM: (i) importá-lo para dentro de um LMS ou (ii) utilizar uma ferramenta de autoria específica para desenvolvimento de conteúdo nesse padrão. Este trabalho aborda apenas a primeira forma, uma vez que a avaliação de ferramentas de criação de conteúdo não faz parte do escopo do trabalho.

Quando deseja-se importar o pacote para dentro de um LMS os arquivos que o compõe não podem ser importados um de cada vez. É necessário, antes de tudo, que eles sejam empacotados no formato PKZip v2.04g (a extensão atribuída ao arquivo pode ser .zip ou .pif). A única obrigatoriedade desse pacote é conter, internamente e na sua raíz, o arquivo imsmanifest.xml.

Os demais arquivos necessários para formar o objeto de aprendizagem são livres para estarem em diversos formatos e localizações dentro do pacote. O que o SCORM impõe, no entanto, é que cada um desses arquivos seja referenciado dentro do manifesto.

O arquivo de manifesto, sendo um arquivo de dados XML, possui algumas *tags* obrigatórias, descritas no Content Aggregation Model. Conhecer todas elas, no entanto, não é o objetivo deste trabalho: o essencial é conhecer as tags necessárias para criar pequenos objetos reutilizáveis. Visualizar, primeiramente, um exemplo de manifesto pode facilitar o entendimento do papel desse arquivo

Listagem 3.1: Esqueleto do Manifesto XML

```
1  <manifest>
2    <metadata>
3    </metadata>
4    <organizations>
5      <organization>
6        <title></title>
7        <item></item>
8      </organization>
9    </organizations>
10   <resources>
11     <resource>
12       <file></file>
13     </resource>
14   </resources>
15 </manifest>
```

No documento XML acima pode-se identificar o elemento raíz manifesto (`<manifest>`). O elemento `<manifest>` possui 3 descendentes diretos. Cada um desses descentes tem papel fundamental na estrutura SCORM, a saber:

- Metadados (`<metadata>`): esse elemento provê informações que descrevem o pacote como um todo. Internamente a esse elemento deve-se declarar, obrigatoriamente, outros dois: `<schema>` e `<schemaversion>`. O primeiro sempre deve ter o conteúdo "ADL SCORM", enquanto o segundo deve indicar a versão do padrão SCORM implementada pelo pacote ("2004 3rd Edition", por exemplo).

- Organizações (`<organizations>`): define uma ou mais árvores de naveção para os Recursos do pacote. Esse elemento, embora obrigatório, pode ser declarado na forma de elemento vazio (`<organizations />`), indicando que o pacote não possui uma estrutura

de navegação. Um pacote que deseja incluir uma árvore de navegação deve declarar, internamente a `<organizations>`, quantos elementos `<organization>` julgar necessários. Diferentes organizações têm o propósito de apresentar diferentes formas de navegar por um conteúdo – o pacote ou o LMS podem optar por uma organização ou por outra.

- Recursos (`<resources>`): define uma coleção de referências a recursos. Cada recurso é descrito pelo elemento `<resource>`. Não há limites para o número de elementos `<resource>` de um arquivo manifesto, ou seja, não há limite, também, para a quantidade de arquivos de um pacote SCORM. Um elemento `<resource>` precisa de um identificador único (`identifier`), de forma que possa ser referenciado, por exemplo, na árvore de navegação do pacote. Para declarar a localização física utiliza-se o atributo `href`.

3.1.3 Metadados

De acordo com o padrão SCORM, o objetivo dos metadados é descrever os componentes do Modelo de Conteúdo. Descrevê-los com metadados facilita a busca e descoberta desses componentes dentro de LMSs (ADL, 2006a). Metadados também são utilizados pelo padrão para definir regras de seqüenciamento dentro de uma Árvore de Atividades. A definição de metadados do padrão SCORM adere às definições de outro padrão, o *Learning Object Metadata* (IEEE LOM) (IEEE, 2007). O SCORM descreve como os metadados do padrão IEEE LOM podem ser mapeados para componentes do Modelo de Conteúdo.

Os aproximadamente 64 elementos do IEEE LOM são divididos em 9 categorias, de acordo com o tipo de informação que descrevem. *Todos* os elementos de metadados do LOM são *opcionais*. A listagem abaixo detalha cada uma dessas categorias.

1. *Geral*: pode ser utilizada para descrever informações gerais sobre um componente.
2. *Ciclo de Vida*: pode ser utilizada para descrever características ligadas ao histórico e ao estado atual de um componente e daqueles que afetaram o componente durante sua evolução.
3. *Meta-metadados*: pode ser utilizada para descrever os próprios metadados.
4. *Técnica*: pode ser utilizada para descrever requisitos técnicos de um componente.
5. *Educacional*: pode ser utilizada para descrever as características educacionais e pedagógicas de um componente.

6. *Direitos*: pode ser utilizada para descrever direitos de propriedade intelectual e condições de uso de um componente.

7. *Relação*: pode ser utilizada para descrever relações entre o componente em questão e outros componentes.

8. *Anotação*: pode ser utilizada para prover comentários sobre o uso educacional do componente e informações sobre quem criou tais comentários e quando o fez.

9. *Classificação*: pode associar uma categoria ao componente, no contexto de um conjunto particular de categorias.

Todas essas categorias, se definidas, devem estar dentro do elemento `<lom>` do arquivo de manifesto. Cada uma das nove categorias, se definidas, devem ser posicionadas dentro do elemento `<lom>`. Cada categoria possui elementos filhos, os quais podem ser consultados na referência ADL (2006a). A listagem 3.2 apresenta, de forma resumida, a estrutura XML correspondente a um manifesto contendo as nove categorias de metadados.

Listagem 3.2: As nove categorias de metadados dentro de um manifesto SCORM

```
<lom>
    <general> </general>
    <lifeCycle> </lifeCycle>
    <metaMetadata> </metaMetadata>
    <technical> </technical>
    <educational> </educational>
    <rights> </rights>
    <relation> </relation>
    <annotation> </annotation>
    <classification> </classification>
</lom>
```

De acordo com o padrão, portanto, cada organização, seja ela uma universidade, um departamento etc., pode definir os elementos da especificação IEEE LOM adequados ao seu contexto e utilizá-los para descrever seus pacotes SCORM.

Um conjunto de metadados pode ser associado aos seguintes elementos de um arquivo manifesto:

- *Manifesto*: quando o conjunto de metadados é filho do elemento-raiz `<manifest>`.

- *Organização de conteúdo*: quando o conjunto de metadados é filho de um elemento `<organization>`.
- *Atividades*: quando o conjunto de metadados é filho de um elemento `<item>`.
- *SCO*: quando o conjunto de metadados é filho de um elemento `resource` (com a propriedade `adlcp:scormType` definida como `sco`).
- *Asset*: quando o conjunto de metadados é filho de um elemento `resource` (com a propriedade `adlcp:scormType` definida como `asset`).

3.1.4 Seqüenciamento

Esta seção descreve os elementos XML do manifesto que possibilitam definir diferentes comportamentos de seqüenciamento para elementos `<organization>` e `<item>`. Para maiores informações sobre o processo geral de seqüenciamento consulte a seção 3.2.

Para criar informações relacionadas a seqüenciamento há dois elementos:

- `<sequencing>`: encapsula todas as informações necessárias para uma atividade.
- `<sequencingCollection>`: pode ser utilizado para colecionar conjuntos de informações de seqüenciamento, para que sejam reutilizadas por diversas atividades.

Atividades dentro um manifesto SCORM são representadas por elementos `<item>` ou `<organization>`. O elemento `<sequencing>` pode ser posicionado como filho direto de um desses dois elementos.

O elemento `<sequencing>` é complexo e possui como filhos os elementos a seguir:

- `<controlMode>`: indica que tipos de requisições de seqüenciamento são permitidas. Este elemento não possui descendentes, apenas atributos, entre os quais destacam-se:
  - `choice`: indica se uma requisição de seqüenciamento *Escolha* pode endereçar uma atividade filha desta.
  - `forwardOnly`: indica que o aprendiz não pode navegar "para trás" dentro das atividades filhas desta.
- `<sequencingRules>`: este elemento é o container para todos os elementos que descrevem as regras de seqüenciamento para uma atividade. As regras são avaliadas na forma

em que são declaradas. Este elemento não possui atributos mas possui três filhos diretos, a saber:

  - `<preConditionRule>`: este elemento é o container para a descrição de ações que controlam decisões de seqüenciamento e entrega de uma atividade específica. As regras que incluem tais ações são utilizadas para determinar se a atividade deve ser entregue.
  - `<postConditionRule>`: este elemento é o container para a descrição de ações que controlam decisões de seqüenciamento e entrega de uma atividade específica. As regras que incluem tais ações são aplicadas quando a tentativa na atividade termina.
  - `<exitConditionRule>`: este elemento é o container para a descrição de ações que controlam decisões de seqüenciamento e entrega de uma atividade específica. As regras que incluem tais ações são aplicadas depois que uma tentativa em uma atividade descendente desta termina.

- `<limitConditions>`: a ADL recomenda o uso deste elemento com precaução, e por isso ele não será abordado neste trabalho.

- `<auxiliaryResources>`: a ADL recomenda o uso deste elemento com precaução, e por isso ele não será abordado neste trabalho.

- `<rollupRules>`: é o container para um conjunto de regras de propagação. Por "propagar", neste contexto, entende-se o mecanismo pelo qual uma atividade-folha propaga para cima (até o elemento-raíz da Árvore de Atividades) mudanças ocorridas no conjunto de dados que rastreia a tentativa do aprendiz na atividade.

- `<objectives>`: é o container para o conjunto de objetivos que deve estar associado com uma atividade. As atividades devem possuir, minimamente, um objetivo primário, e podem possuir quantos objetivos desejarem.

- `<randomizationControls>`: é o container para as descrições de como os elementos-filho desta atividade devem ser ordenados durante o processo de seqüenciamento.

- `<deliveryControls>`: tem o mesmo objetivo que o elemento `<randomizationControls>`.

- `<adlseq:constrainedChoiceConsiderations>`: é o container para as descrições de como as requisições de navegação devem ser restringidas durante o processo de seqüenciamento. A restrição aplica-se apenas ao elemento para o qual esta propriedade é definida.

- `<adlseq:rollupConsiderations>`: é o container das descrições de quando esta atividade deve ser incluída no processo de propagação.

O elemento `<sequencing>` e seus descendentes, portanto, têm por objetivo garantir ao projetista do conteúdo um grande controle sobre as regras que regem o seqüenciamento de uma Árvore de Atividades como um todo e das atividades individualmente. Essas regras, em última instância, permitem o projeto de uma seqüência "inteligente" de entrega de atividades. Uma atividade não é entregue, por exemplo, a menos que se possa inferir que o aprendiz tem o conhecimento necessário (inferido a partir do modelo de dados de tempo de execução) para visualizá-la. Ou, então, dado o conhecimento do aprendiz, também por forma de inferência, pode-se eleger uma entre diversas atividades semelhantes como a melhor atividade para consolidar o conhecimento desse aprendiz. Desse modo dois aprendizes diferentes contariam com seqüências de atividades diferentes durante a experiência de aprendizado.

Para maiores detalhes sobre as possíveis regras de seqüenciamento consulte a referência (ADL, 2006d). Elas são bastante poderosas e devem ser estudadas com cuidado tanto pelo projetista do LMS quanto pelo projetista de conteúdos SCORM.

3.2 Seqüenciamento e Navegação

Esta seção descreve o funcionamento geral dos mecanismo de seqüenciamento e navegação. Para maiores informações sobre como definir as regras de seqüenciamento consulte a seção 3.1.4.

3.2.1 Conceitos de Seqüenciamento

Árvore de Atividades

O padrão SCORM utiliza como base para seu esquema de seqüenciamento a figura de uma Árvore de Atividades (*Activity Tree*). Essa árvore pode ser derivada diretamente do elemento *organization*, presente no arquivo de manifesto do pacote SCORM. O elemento *organization* representa a raíz dessa árvore, enquanto seus elementos *item* internos correspondem a uma atividade. O padrão assume que todo LMS deve ser capaz de traduzir a estrutura XML presente no manifesto em uma árvore de atividades. Essa árvore é base para a definição de todas as possíveis regras de seqüenciamento. A figura 3.1, adaptada de ADL (2006d), mostra uma árvore de atividades derivada da estrutura XML de um arquivo manifesto.

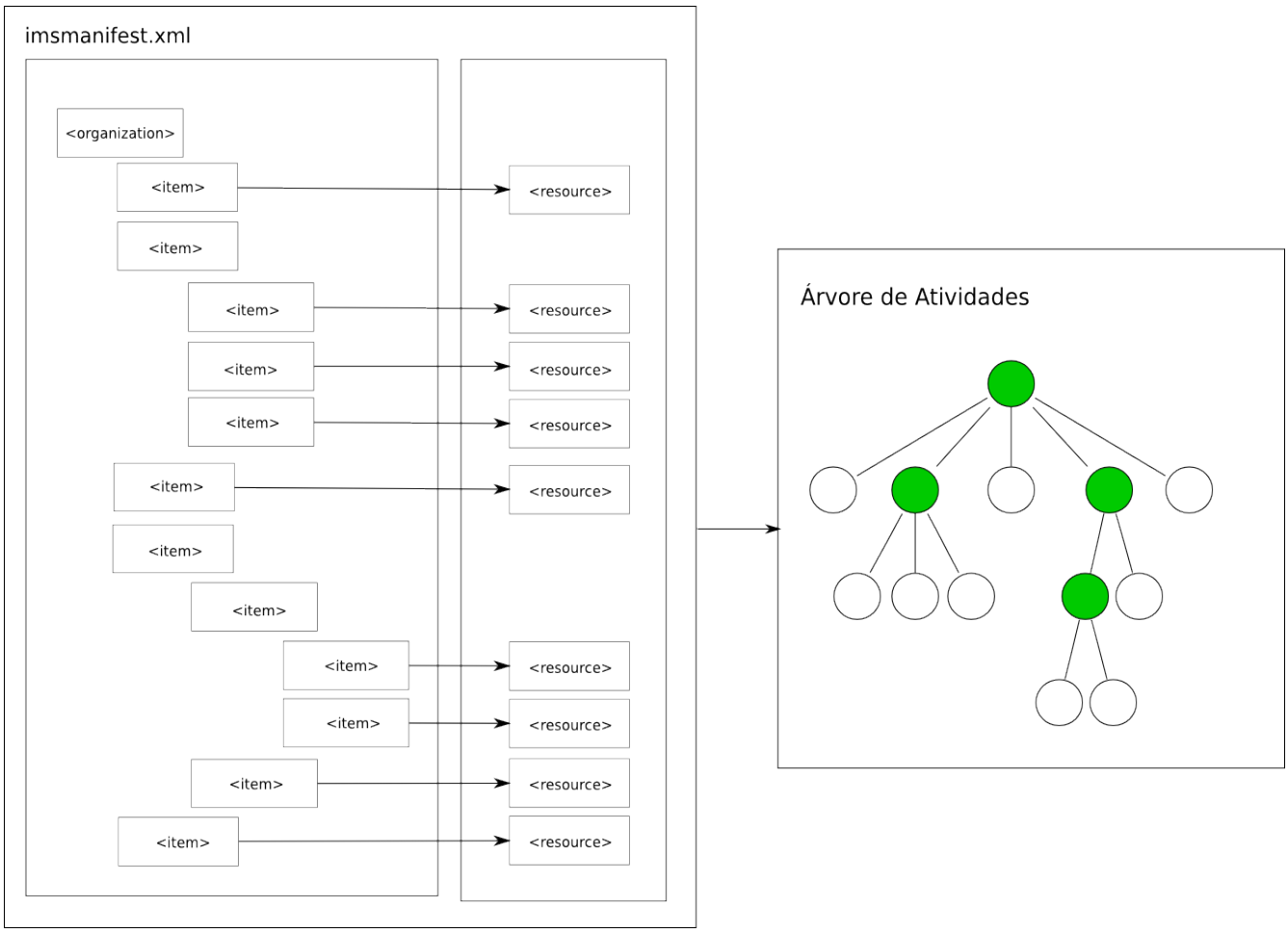

Figura 3.1: Árvore de Atividades derivada de um arquivo manifesto (adaptada de ADL (2006d))

Agregado

Um agregado (*cluster*) é toda a atividade de aprendizado que possui sub-atividades. Os filhos de um agregado ou são atividades-folha ou são outros agregados, ou seja, um agregado envolve no máximo dois níveis da árvore. Uma atividade-folha não é um cluster. Muitos elementos do Modelo de Definição de Seqüenciamento aplicam-se especificamente a agregados. A atividade-pai de um agregado contem informações sobre a estratégia de seqüenciamento para esse agregado. As atividades-folha do agregado possuem objetos de conteúdo (*content objects*) associados, os quais serão selecionados para entrega de acordo com a estratégia de seqüenciamento definida para o agregado. A figura 3.2, adaptada de ADL (2006d), ilustra de que maneira as atividades de uma árvore de atividades são agrupadas em pequenos agregados.

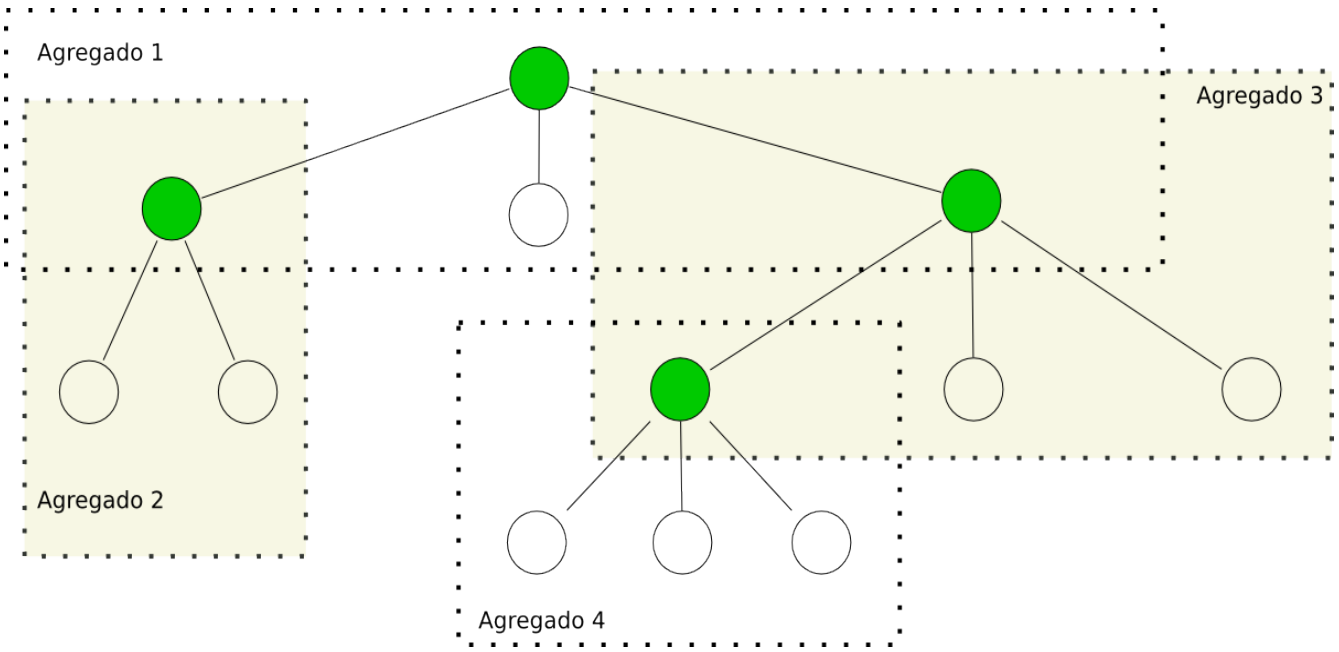

Figura 3.2: Agregado de atividades (adaptada de ADL (2006d))

Atividade de Aprendizado

Para o SCORM uma atividade de aprendizado (*learning activity*) é qualquer quantidade significativa de instrução. Essa atividade pode consistir em, por exemplo, exibir ao aprendiz algum conteúdo e em seguida aplicar um teste para verificar a assimilação desse conteúdo. Nesse caso, ainda, poderia-se pensar na atividade como sendo "Realizar o teste". Essa atividade englobaria duas sub-atividades, sendo uma responsável por exibir o conteúdo e a outra por aplicar o teste de aprendizado. É importante notar que o aprendiz sempre realiza sub-atividades *no contexto de* uma atividade-pai.

Não há restrição quanto ao aninhamento de atividades (a hierarquia de atividades pode ter tantos níveis quanto se julgue necessário), e toda atividade que não possui sub-atividades é chamada de atividade-folha. Toda atividade-folha possui um objeto de conteúdo associado.

Tentativa

Uma tentativa (*attempt*) é definida pelo padrão SCORM como o esforço empregado pelo aprendiz para completar uma atividade. *Tentativas em atividades sempre ocorrem dentro do contexto das tentativas nas atividades-pai*. Enquanto uma atividade está sendo tentada, todas as atividades-pai dessa são consideradas *em progresso*.

Uma tentativa começa quando uma atividade é identificada para entrega e termina enquanto o LMS identifica a próxima atividade para entrega. Nem sempre será possível para um aprendiz

terminar uma tentativa de uma única vez: ele poderá suspender a tentativa. Quando o aprendiz volta a ver uma atividade com a qual ele possui uma tentativa suspensa, essa tentativa é continuada, não sendo considerada uma nova tentativa.

Sessão de Seqüenciamento

Uma sessão de seqüenciamento (*sequencing session*) corresponde ao intervalo de tempo entre o início de uma tentativa na atividade-raíz de uma Árvore de Atividades até o momento em que essa tentativa termina. Quando essa sessão é iniciada (o aprendiz acessou um objeto de aprendizagem dentro de um LMS, por exemplo), o LMS deve gerar uma requisição de navegação Iniciar. Para uma tentativa que foi terminada por uma requisição Suspender Todos, a mesma deve ser continuada pelo LMS com uma requisição Continuar Todos, e não uma requisição Iniciar. Uma sessão de seqüenciamento termina quando uma requisição de navegação Sair é processada a partir da atividade-raíz da Árvore de Atividades.

3.2.2 Rastreamento do Estado de um Atividade

O padrão SCORM depende de valores armazenados dentro do Modelo de Rastreamento de Estados (*Tracking Status Model*) para controlar o comportamento do seqüenciamento (ADL, 2006d). Para cada tentativa do aprendiz em uma atividade devem ser gerados dados de rastreamento do estado referentes àquela tentativa naquela atividade. O padrão prevê que apenas SCOs podem comunicar-se com o LMS e, portanto, apenas esses possuem papel ativo no rastreamento do estado. É responsabilidade do SCO informar ao LMS sobre o progresso do aprendiz ocorrido dentro do seu conteúdo.

Outras atividades podem ser tentadas enquanto uma está suspensa, e pode haver mais de uma atividade suspensa para o mesmo aprendiz. Se uma tentativa é concluida por um aprendiz em um SCO, e outra tentativa é iniciada por ele no mesmo SCO, o LMS não é obrigado a salvar o estado da tentativa original. Optar pelo salvamento de tentativas anteriores é decisão do LMS.

3.2.3 Modelo de Definição do Seqüenciamento

De acordo com ADL (2006d), este modelo define um conjunto de elementos que pode ser utilizado por criadores de conteúdo para definir suas intenções de comportamento em termos de seqüencia da entrega de atividades. O modelo é aplicado a atividades da Árvore de Atividades, e cada uma dessas possui uma valor padrão para a propriedade do modelo. Esse valor é utilizado caso o criador do conteúdo não tenha definido seu valor explicitamente.

Modos de Controle do Seqüenciamento

Estes modos permitem ao criador do conteúdo interferir na forma como as requisições de navegações são aplicadas a um agregado, e como as atividades de um agregado são escolhidas durante o processamento dessas requisições. Os modos de controle são utilizados das seguintes maneiras (ADL, 2006d):

- Durante o processamento de uma *requisição de nevegação*, para determinar se a requisição será traduzida em uma *requisição de seqüenciamento* válida.
- Durante vários subprocessos da requisição de seqüenciamento, interferindo na forma como as atividades são consideradas para entrega.
- Durante comportamentos de seqüenciamento, para afetar a forma como as informações de rastreamento são gerenciadas.

A listagem abaixo relaciona os modos de controle existentes. Uma atividade possuirá um valor `true` ou `false` para cada uma dessas propriedades.

- Escolha (*Sequencing Control Choice*): indica que um aprendiz pode escolher qualquer atividade dentro de um agregado, em qualquer ordem, sem qualquer restrição. Por padrão, toda atividade, por toda a Árvore de Atividade, filha de uma atividade-pai cujo modo "Escolha" é definido como `true` é um alvo válido para uma requisição de navegação "Escolha". Este modo é aplicado apenas a atividades que possuem atividades-filha.
- Escolha para Sair (*Sequencing Controle Choice Exit*): indica que um aprendiz pode tentar acessar uma atividade fora do ramo de atividades que está visualizando, ou seja, gerar uma requisição "Escolha" para uma atividade que não seja descendente da atividade atual. Quando definida como `true`, a propriedade indica que o aprendiz *pode* acessar atividades que não as descendentes da atual; quando definida como `false` que apenas as atividades descendentes podem ser acessadas. O valor padrão é `true`.
- Fluxo (*Sequencing Control Flow*): indica se o fluxo de navegação entre atividades pode ser controlado pelo LMS. Ou seja, se definida como `true`, a propriedade indica que o LMS deve exibir controles visuais (botões ou links) que permitam ao aprendiz pular de uma atividade para a próxima. Por outro lado, se definida como `false`, então o LMS não pode prover controles para a próxima atividade e para a anterior, ficando a cargo do SCO decidir isso. O valor padrão é `false`.

- Apenas para Frente (*Sequencing Control Forward Only*): indica que não deve ser permitido, dentro de um agregado de atividades, ao aprendiz "caminhar para trás", ou seja, acessar uma atividade anterior àquela que está acessando. Fica à cargo do LMS não prover os elementos visuais que permitiriam ao aprendiz violar essa regra. Quando definida como `true` essa regra fica ativada, e definida como `false` fica desativada (o aprendiz pode se mover para a próxima atividade e também para a anterior). O valor padrão é `false`.

- Utilizar informações dos objetivos da tentativa atual: indica que as informações relacionadas aos objetivos para a atividade atual devem ser utilizadas. O valor padrão é `true`.

- Utilizar informações do progresso da tentativa atual: indica que as informações relacionadas ao progresso na tentativa atual deve ser utilizada. O valor padrão é `true`.

3.2.4 Comportamentos de Seqüenciamento

O padrão SCORM define os seguintes modelos conceituais para separar por responsabilidades as várias e complexas tarefas e decisões envolvidas no controle do seqüenciamento de atividades.

- Modelo de Rastreamento (*Tracking Model*): captura informações provenientes da interação do aprendiz com os objetos de conteúdo associados às atividades. Este é um modelo utilizado em tempo de execução.

- Modelo de Estado das Atividades (*Activity State Model*): utilizado pelo LMS para gerenciar o estado da Árvore de Atividades durante a sessão do aprendiz. Esse modelo aborda as atividades de forma individual e também a árvore como um todo. Este é um modelo de tempo de execução.

- Modelo de Definição do Seqüenciamento (*Sequencing Definition Model*): descreve como os vários processos de seqüenciamento utilizam e interpretam informações do Modelo de Rastreamento. Este não é um modelo de tempo de execução, ou seja, é definido dentro do pacote SCORM de forma estática.

Modelo de Rastreamento

De acordo com ADL (2006d), para possibilitar o *seqüenciamento condicional* de atividades faz-se necessário manter e gerenciar informações associadas às interações de um aprendiz com

os objetos de conteúdo com os quais ele interage. O conjunto desses elementos é chamado Modelo de Rastreamento.

Alguns elementos do Modelo de Dados do Ambiente de Tempo de Execução correspondem diretamente a elementos do Modelo de Rastreamento. A figura 3.3, adaptada de (ADL, 2006d), mostra a relação conceitual entre a Árvore de Atividades, as informações de rastreamento exclusivas de uma atividade, o objeto de conteúdo associado àquela atividade e o conjunto de dados de tempo de execução associados aquele objeto de conteudo.

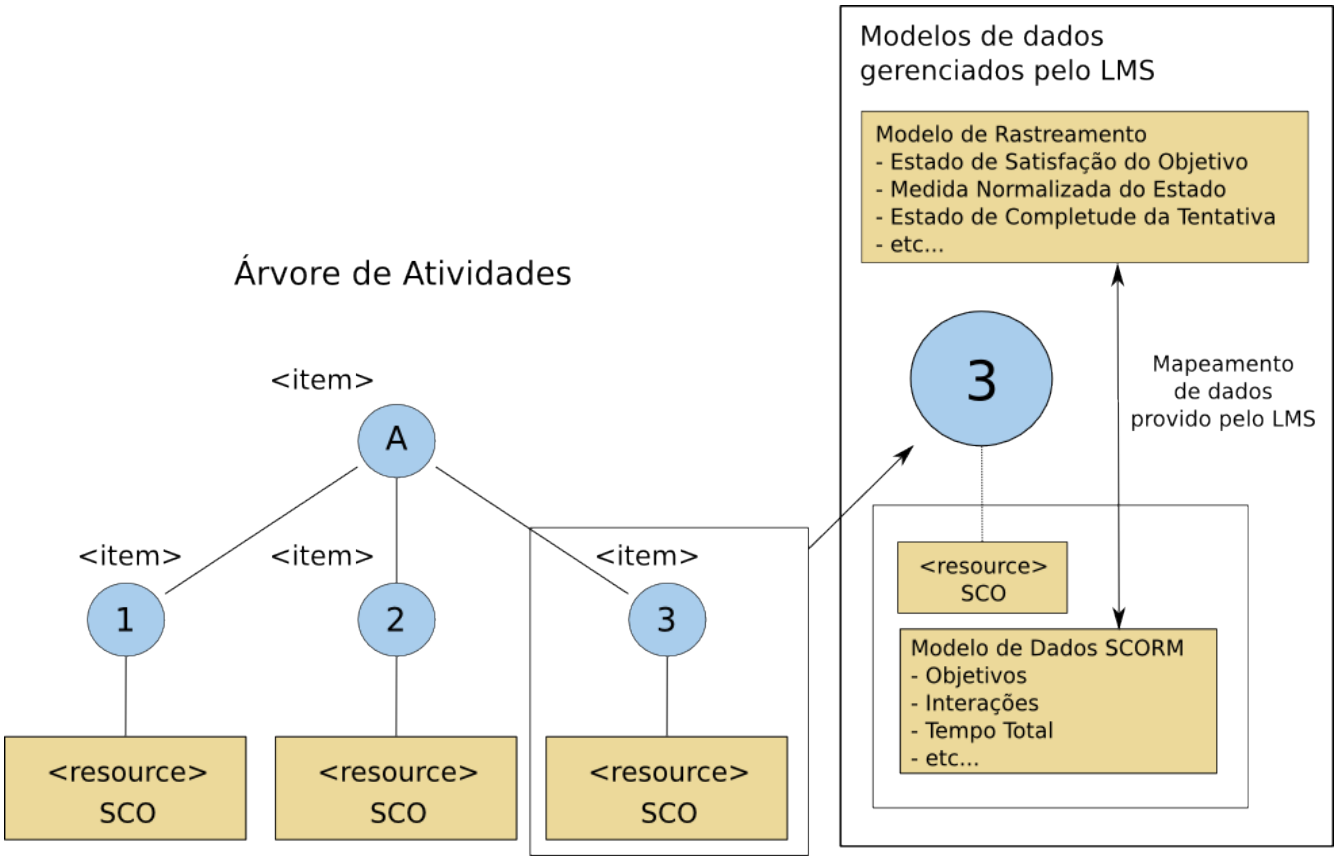

Figura 3.3: Relacionamento entre modelos de dados SCORM (adaptada de ADL (2006d))

O padrão SCORM define quatro conjuntos de informações relacionadas a rastreamento:

- Informações do Progresso do Objetivo (*Objective Progress Information*): Para cada tentativa em uma atividade, o aprendiz recebe um conjunto dessas informações para cada objetivo associado com a atividade. Essas informações são definidas pelos seguintes elementos:
  - Estado do Progresso do Objetivo (*Objective Progress Status*): indica que Estado de Satisfação do Objetivo possui informação válida.

  - Estado de Satisfação do Objetivo (*Objective Satisfied Status*): indica que o objetivo está satisfeito (`true` ou `false`).
  - Estado de Mensuração do Estado (*Objective Measure Status*): indica que Mensuração Normalizada do Objetivo possui informação válida.
  - Mensuração Normalizada do Objetivo (*Objective Normalized Measure*): uma nota, normalizada entre -1.0 e 1.0, para indicar o progresso do objetivo.

- Informações do Progresso da Atividade (*Activity Progress Information*): as informações que duram por todas as tentativas de um aprendiz em uma mesma atividade. Essas informações são definidas pelo seguintes elementos:
  - Estado do Progresso da Atividade (*Activity Progress Status*): indica que as demais variáveis possuem informações válidas. Este valor é definido como `true` quando a primeira tentativa na atividade tem início.
  - Duração Absoluta da Atividade (*Activity Absolute Duration*): é a duração total de todas as tentativas na atividade, contando inclusive o tempo que o aprendiz deixou de interagir com ela (pois se desconectou do LMS, por exemplo).
  - Duração Real da Atividade (*Activity Experienced Duration*): é o tempo total que o aprendiz realmente interagiu com a atividade (períodos *offline*, por exemplo, são desconsiderados).
  - Quantidade de Tentativas na Atividade (*Activity Attempt Count*): indica quantas tentativas o aprendiz realizou nessa atividade. O valor 0 indica nenhuma tentativa, 1 indica uma tentativa e assim por diante.

- Informações do Progresso da Tentativa (*Attempt Progress Information*): para cada tentativa em uma atividade o aprendiz ganha o seguinte conjunto de informações:
  - Estado de Progresso da Tentativa (*Attempt Progress Status*): indica que as demais variáveis possuem informações válidas.
  - Grau de Completude da Tentativa (*Attempt Completion Amount*): em uma escala de 0..1 (1 indicando que a tentativa está completa) indica o quão completa está a tentativa.
  - Estado de Completude da Tentativa (*Attempt Completion Status*): indica se a tentativa está completa ou não (`true` ou `false`).
  - Duração Absoluta da Tentativa (*Attempt Absolute Duration*): é o tempo total que a tentativa levou, contando inclusive o tempo que o aprendiz deixou de interagir com ela.

  - Duração Real da Tentativa (*Attempt Experienced Duration*): é o tempo total que o aprendiz realmente interagiu com a atividade.

- Informações do Estado da Atividade (*Activity State Information*): o padrão define os seguintes elementos adicionais utilizados pelo modelo para rastrear cada atividade para cada aprendiz:

  - Atividade está Ativa (*Activity is Active*): indica que uma tentativa está em curso para essa atividade (ou essa atividade é ancestral de uma atividade em curso). Este valor é definido como `true` quando a atividade inicia, e como `false` quando a tentativa nessa atividade termina. Apenas uma atividade-folha pode ter esse valor definido como `true` em um dado momento.
  - Atividade está Suspensa (*Activity is Suspended*): indica que essa atividade está suspensa.
  - Filhas Disponíveis (*Available Children*): uma lista indicando a ordem das atividades-filhas disponíveis para essa atividade. Este elemento só se aplica a uma atividade que representa um agregado.

Há, ainda, dois elementos globais de informação:

- Atividade Atual (*Current Activity*): identifica a atividade-folha na qual o aprendiz está realizando uma tentativa. Todas as atividades ancestrais dessa devem possuir a propriedade *Atividade está Ativa* definida como `true`.
- Atividade Suspensa (*Suspended Activity*): identifica a atividade-folha que foi a *Atividade Atual* na sessão anterior. Todas as atividades ancestrais dessa devem possuir a propriedade *Atividade está Suspensa* definida como `true`.

Processo Geral de Seqüenciamento

O Processo Geral de Seqüenciamento, descrito em ADL (2006d), define como os vários comportamentos de seqüenciamento são aplicados dentro do contexto de uma sessão. O ponto de entrada do processo é um *requisito de navegação* provocado pelo LMS (esses requisitos costumam ser gerados em resposta a um evento de navegação provocado pelo usuário, como "seguir para a próxima atividade"). O ponto de saída para o processo é ou a identificação de uma "próxima" atividade a ser entregue, ou uma exceção. Essa atividade a ser entregue pode significar ainda nenhuma atividade, dependendo do resultado da aplicação das regras definidas

pelos diferentes comportamentos de seqüenciamento. A figura 3.4, adaptada de (ADL, 2006d), ilustra a relação entre os vários comportamentos de seqüenciamento *entre si*, a Árvore de Atividade e o modelo de rastreamento.

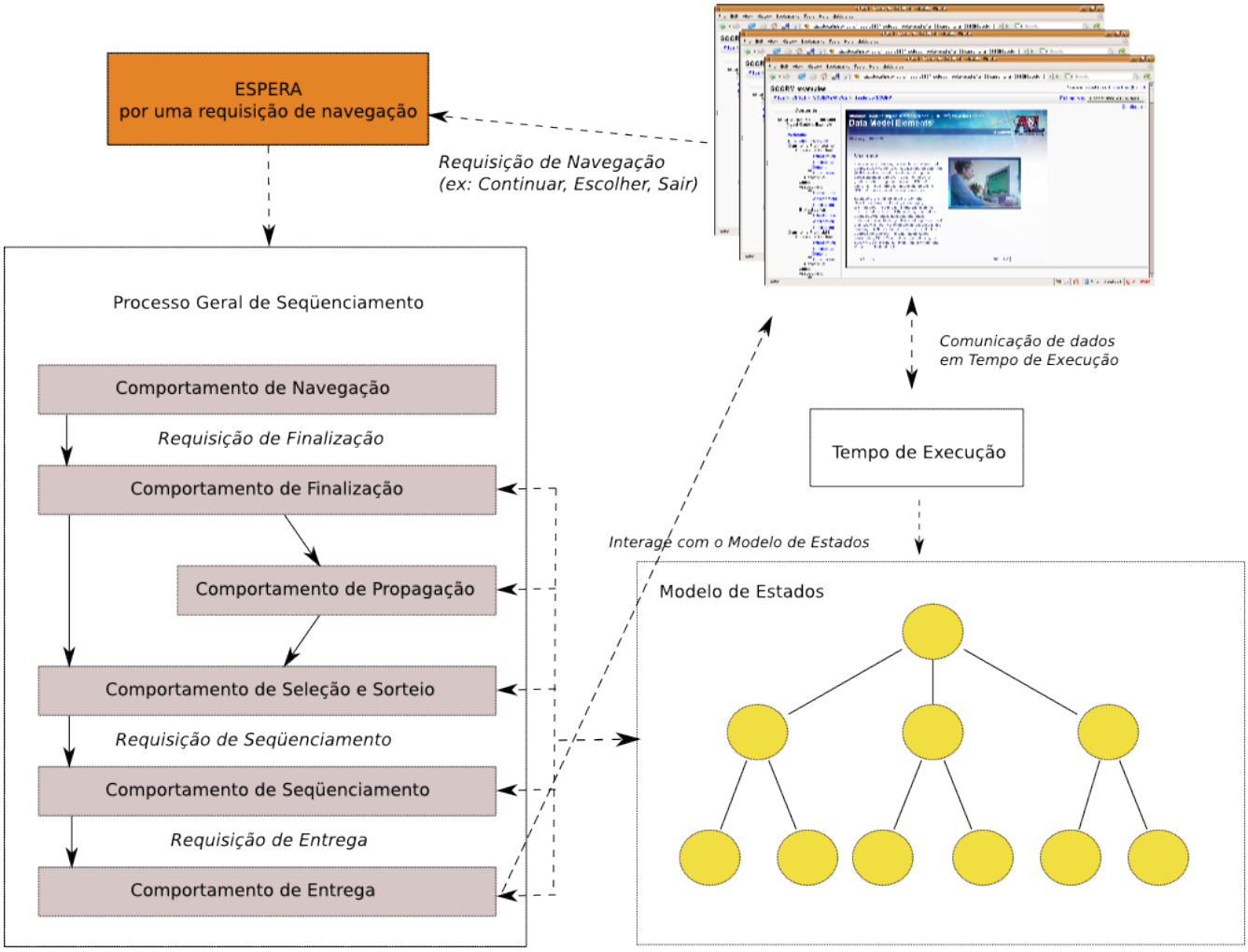

Figura 3.4: Processo Geral de Seqüenciamento (adaptada de ADL (2006d))

- Comportamento de Navegação (*Navigation Behavior*): é o ponto de partida do processo. É responsável por traduzir os eventos de navegação (gerados, em geral, pela interação do aprendiz com elementos da interface) em requisições de navegação.

- Comportamento de Finalização (*Termination Behavior*): enquanto uma requisição de navegação é processada a atividade atual pode precisar ser terminada, caso exista uma próxima atividade válida. O comportamento de navegação é responsável por controlar essa etapa do processo, que inclui a definição da próxima *Atividade Atual*, por exemplo.

- Comportamento de Propagação (*Rollup Behavior*): como não há informações de rastreamento associadas diretamente a agregados de atividades, o LMS deve utilizar os dados das

atividades-folha para atualizar o estado de suas atividades-pai, até chegar à atividade-raíz da árvore.

- Comportamento de Seleção e Sorteio (*Selection and Randomization Behavior*): descreve quando as atividades-filha de um agregado são selecionados e quando (se for o caso) esse conjunto é reordenado. De fato, é possível associar mais atividades em um agregado do que o número necessário para cumprir os objetivos desse agregado. Com isso é possível selecionar, para cada aprendiz, conteúdos diferentes a serem apresentados, baseados em uma lógica definida pelo criador do conteúdo.

- Comportamento de Seqüenciamento (*Sequencing Behavior*): seu propósito é, dado o estado atual da Árvore de Atividade, tentar determinar a "próxima" atividade a ser entregue ao aprendiz, ao percorrer a árvore a partir da Atividade Atual, ou simplesmente encontrar a primeira atividade da Árvore, para iniciar uma nova sessão.

- Comportamento de Entrega (*Delivery Behavior*): define o último passo do processo de seqüenciamento. O propósito desse comportamento é, dada uma requisição de entrega, validar tal requisição e, se essa for válida, entregar ao aprendiz o objeto de conteúdo apropriado.

O comportamento de seqüenciamento é descrito pelo padrão SCORM em 12 passos, sendo que do passo 4 ao passo 12 ocorre um ciclo (*loop*). Abaixo esses passos são abordados de forma superficial. Para maiores detalhes deve-se recorrer à especificação de Seqüenciamento e Navegação do padrão (ADL, 2006d).

1. O aprendiz acessa o LMS (realiza uma autenticação, por exemplo) e, de algum modo, acesso um pacote SCORM.

2. O LMS inicia uma sessão de seqüenciamento, e dispara uma requisição de navegação *Iniciar*, *Continuar Todos* ou *Escolher*, dependendo da forma como terminou a última sessão do aprendiz.

3. O Comportamento de Navegação traduz uma dessas requisições de navegação em uma requisição de seqüenciamento, e a processa. Uma sessão será iniciada oficialmente quando uma atividade é identificada para entrega.

4. Baseado em informações presentes no modelo de rastreamento e na requisição de seqüenciamento o Comportamento de Seqüenciamento percorre a Árvore de Atividades buscando a próxima atividade a ser entregue ao aprendiz. Se nenhuma atividade pode ser

entregue o processo pára e passa a aguardar uma nova requisição de navegação (pula para o Passo 12).

5. O Comportamento de Entrega determina se a atividade identificada para entrega pode ser entregue, e a prepara. Se a mesma não puder ser entregue o processo pára e passa a aguardar uma nova requisição de navegação (pula para o Passo 12).

6. O aprendiz interage com o objeto de conteúdo. O processamento fica ocioso à espera de uma requisição por parte do aprendiz.

7. O objeto de conteúdo pode comunicar atualizações de valores do modelo de rastreamento enquanto o aprendiz interage com ele.

8. O aprendiz, ou o objeto de conteúdo ou o sistema invoca um evento de navegação (como *Continuar* ou *Escolher*).

9. O LMS transforma o evento de navegação em uma requisição de navegação.

10. O Comportamento de Navegação traduz a requisição de navegação em uma requisição de finalização (que terminará a sessão na atividade corrente) e em uma requisição de seqüenciamento.

11. Se o objeto de conteúdo conseguir, nesta etapa, se comunicar com o LMS, ele pode atualizar dados do modelo de rastreamento, e em seguida a tentativa acaba. O Comportamento de Propagação é invocado para determinar os efeitos de quaisquer mudanças que ocorreram durante a interação do aprendiz com o objeto. O Comportamento de Propagação atualiza o estado de todas as atividades, a partir da atividade recém-terminada até a raíz da Árvode de Atividade.

12. O ciclo de repete a partir do passo 4, até que a sessão termine.

3.2.5 Navegação

O conceito de navegação do SCORM baseia-se nos elementos navegacionais da interface que o LMS e um SCO disponibilizam ao aprendiz durante o processo de aprendizagem. Tais elementos devem permitir que o aprendiz navegue, dentro de um objeto de aprendizagem, entre os diferentes SCOs disponíveis. Nesse sentido, o padrão deixa claro que não impõe restrições à navegação interna a um SCO: apenas a navegação entre SCOs é especificada (ADL, 2006d).

O padrão SCORM descreve, ainda, uma nomenclatura para os tipos de eventos possíveis durante a experiência de navegação do aprendiz.

- Iniciar (*Start*): indica a vontade de identificar a primeira atividade disponível na Árvore de Atividades. Costuma ser gerado pelo LMS quando o aprendiz inicia uma nova tentativa na raíz da Árvore de Atividades. Apenas o LMS pode gerar este evento.

- Continuar Todos (*Resume All*): indica a vontade de continuar uma tentativa anteriormente suspensa na raíz da Árvore de Atividades. Apenas o LMS pode gerar este evento.

- Continuar (*Continue*): indica a vontade de identificar a próxima atividade (em relação à atual) disponível na Árvore de Atividades. Tanto o LMS quanto o SCO podem gerar este evento.

- Anterior (*Previous*): indica a vontade de identificar a atividade anterior (em relação à atual) disponível na Árvore de Atividades. Tanto o LMS quanto o SCO podem gerar este evento.

- Escolher (*Choose*): indica a vontade de pular diretamente para uma atividade específica na Árvore de Atividades. Tanto o LMS quanto o SCO podem gerar este evento.

- Abandonar (*Abandon*): identifica o desejo de terminar abruptamente a *tentativa* atual dentro do SCO atual sem desejo de continuá-la. Este evento não provoca, no entanto, a terminação das atividades-pai da atividade atual. Tanto o LMS quanto o SCO podem gerar este evento.

- Abandonar Todos (*Abandon All*): identifica o desejo de terminar abruptamente a *tentativa* atual na atividade-raíz da Árvore de Atividades, sem desejo de continuá-la. Este evento termina a tentativa atual tanto na atividade-raíz quanto em todas as suas atividades-filhas. Tanto o LMS quanto o SCO podem gerar este evento.

- Suspender Todos (*Suspend All*): identifica o desejo de suspender a tentativa atual na atividade-raíz da Árvore de Atividades. Este evento suspende todas as tentativas filhas da atividade-raíz. Tanto o LMS quanto o SCO podem gerar este evento. O evento *Continuar Todos* funciona como o oposto deste, pois permitir continuar todas as atividades suspensas para uma Árvore de Atividades. Tanto o LMS quanto o SCO podem gerar este evento.

- Saída Não-Qualificada (*Unqualified Exit*): indica que a tentativa atual na atividade atual foi terminada normalmente, e que essa terminação não foi engatilhada por outro evento navegacional como *Continuar*, *Anterior* ou *Escolher*. Tanto o LMS quanto o SCO podem gerar este evento.

- Sair de Todos (*Exit All*): indica que a tentativa atual na atividade-raíz da Árvore de Atividades foi terminada normalmente. Este evento termina a tentativa atual na atividade-raíz e em todas as suas filhas ativas. Tanto o LMS quanto o SCO podem gerar este evento.

Todos os eventos de navegação listados geram uma *requisição de navegação* correspondente. Em geral essa requisição possui um nome similar ao nome do próprio evento. O evento *Escolher*, por exemplo, engatilha a requisição de navegação *Escolha*.

Processamento de Requisições de Navegação

Quando uma requisição de navegação é processada pelo LMS, um de três resultados são possíveis:

- Se o efeito da requisição é encerrar a tentativa atual na Árvore de Atividades, o LMS processa a requisição *Sair de Todos*, que termina a tentativa atual e retorna o controle para o LMS.
- Depois de avaliar informações relacionadas ao seqüenciamento da árvore atual o LMS decide que o processamento da requisição não deve ser completado (o LMS ignora a requisição).
- Depois de avaliar informações relacionadas ao seqüenciamento da árvore atual o LMS decide que o processamento da requisição deve ser realizado. O resultado é um dos seguintes:
  - Uma atividade é identificada para entrega
  - Nenhuma atividade é identificada para entrega
  - Uma exceção ocorre durante a escolha da atividade

Modelo de Apresentação da Informação

Embora o padrão SCORM não imponha requisitos sobre a aparência e disposição dos elementos de interface que permite ao aprendiz gerar eventos navegacionais (tais como *Continuar*, *Escolher*, *Abandonar*), o LMS deve, obrigatoriamente, prover tais elementos. Um botão HTML com a palavra “Próximo”, por exemplo, pode ser utilizado para permitir ao aprendiz gerar o evento *Continuar* na atividade atual.

Além disso, o padrão permite ao SCO tanto apresentar, ele mesmo, elementos visuais para navegação quanto solicitar ao LMS que esconda um ou mais desses. Os seguintes elementos visuais podem ser "escondidos" pelo LMS quando assim determinar um SCO (ADL, 2006d).

- *previous*: permite engatilhar o evento *Anterior*.
- *continue*: permite engatilhar o evento *Continuar*.
- *exit*: permite engatilhar o evento *Sair*.
- *exitAll*: permite engatilhar o evento *Sair de Todos*.
- *abandon*: permite engatilhar o evento *Abandonar*.
- *abandonAll*: permite engatilhar o evento *Abandonar Todos*.
- *suspendAll*: permite engatilhar o evento *Suspender Todos*.

Comunicação de Requisições de Navegação

Uma vez que um SCO pode optar por apresentar ou não apresentar determinados elementos visuais de navegação, é importante que o mesmo possa questionar o LMS a respeito do contexto navegacional da atividade atual: "há uma próxima atividade?"; "há uma atividade anterior?". Desse modo o SCO pode prover os elementos visuais que proporcionam a navegação mais adequada ao aprendiz (provendo, por exemplo, um link para a próxima atividade *a menos que* uma próxima atividade não esteja disponível).

Um SCO pode, ainda, informar suas intenções navegacionais para o LMS. O SCO pode definir uma, e apenas uma, requisição de navegação a ser processada pelo LMS no momento do término do SCO. O SCO pode definir, por exemplo, que a requisição *Sair* seja processada quando o aprendiz finalizar o conteúdo.

Essa comunicação é realizada com base no Modelo de Dados de Navegação do SCORM, e ocorre em tempo de execução por intermédio da API (*Application Programming Interface* ou Interface para Programação de Aplicativos) SCORM.

Modelo de Dados de Navegação em Tempo de Execução

O modelo de dados de navegação é bastante simples, e composto por apenas dois elementos principais.

- adl.nav.request: utilizado para definir qual requisição de navegação o LMS deve processar após o término da atividade atual. Este valor pode ser definido durante todo o tempo que o aprendiz estiver interagindo com a atividade. O LMS deve implementar esta propriedade como escrita e leitura (pode ser lida e redefinida em tempo de execução).

- adl.nav.request_valid: como o SCO não pode saber, sozinho, o seu contexto navegacional (se há ou não atividades antes e depois de si), ele deve recorrer ao LMS para obter tal informação. De posse dela, pode optar por exibir certos elementos de interface (como um link para a atividade anterior). Este elemento é uma coleção, e seus reais elementos são os seguintes:

  - continue: indica se há ou não uma próxima atividade.
  - previous: indica se há ou não uma atividade anterior.
  - choice.{TARGET=STRING}: indica se o processamento de uma requisição *Escolha* na atividade identificada por TARGET resultaria em uma atividade identificada para entrega.

Os elementos acima são lidos e atualizados com o suporte das funções GetValue e SetValue providas pela API SCORM. Abaixo estão alguns exemplos de uso dessas propriedades:

- SetValue(“adl.nav.request”, “continue”)
- GetValue(“adl.nav.request_valid.continue”)
- GetValue(“adl.nav.request_valid.previous”)
- GetValue(“adl.nav.request_valid.choice{target=intro}”)

3.3 Ambiente de Tempo de Execução

A especificação do SCORM para o Ambiente de Tempo de Execução (RTE) detalha os requisitos para se iniciar objetos, estabelecer a comunicação entre os LMSs e os SCOs e gerenciar o rastreamento das informações que podem trafegar entre os SCOs e os LMSs. Enquanto um Asset não pode se comunicar com um LMS em tempo de execução, um SCO pode (ADL, 2006c).

Um objeto que se comunica com o LMS seguindo o padrão SCORM poderá ser movido de uma plataforma para outra sem precisar ter sua lógica de comunicação reprogramada, uma vez

que o padrão define o protocolo entre os SCOs e o LMS. Isso acaba resultando em alta portabilidade e durabilidade, que diminuem o custo de desenvolvimento, instalação e manutenção.

A especificação do ambiente de execução do SCORM começa do ponto em que o LMS lançou o objeto de aprendizagem para o cliente – aquele ponto em que o aprendiz acabou de começar a sua interação com os diferentes conteúdos do objeto. Faz parte do escopo do padrão, portanto:

- Entrega de objetos de conteúdo para o navegador Web do aprendiz;
- Se necessária, como se dará a comunicação entre objeto e LMS e;
- Que informações são rastreadas por um objeto e como o LMS as gerencia

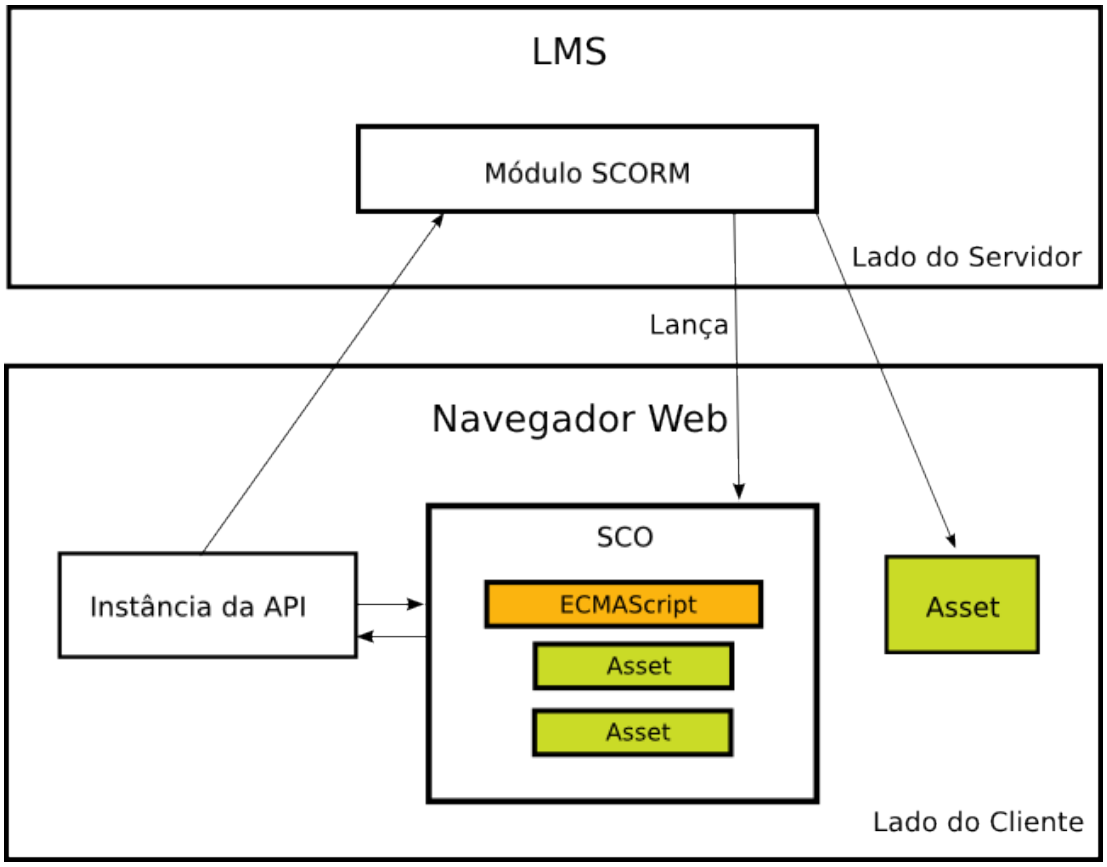

Figura 3.5: Arquitetura da comunicação entre o LMS e o objeto SCORM (adaptada de ADL (2006d))

Antes de entrar em detalhes sobre o protocolo de comunicação é preciso que se entenda o essencial: no fim das contas a comunicação entre um objeto de aprendizagem e um LMS é realizada com a utilização da linguagem ECMAScript (nos navegadores Web ela é implementada

com o nome de JavaScript). A API implementada pelo LMS é escrita nessa linguagem, e isso implica que ela seja localizada dentro do navegador Web do aprendiz.

A única forma de um objeto de aprendizagem se comunicar com a API desse LMS, portanto, é utilizar a mesma linguagem. Ou seja, o projetista do SCO deve implementar diversos "ouvintes" de evento. Cada evento (aprendiz respondeu "Sim" para a pergunta "1", por exemplo) engatilhará a ação de um ouvinte, implementado em JavaScript pelo OA. Esse ouvinte irá, por sua vez, comunicar-se diretamente com a API do LMS (solicitando, por exemplo, que a propriedade "pontuação" para aquele aprendiz seja incrementada em 1 ponto). Apesar de o SCO estar se comunicando com a API do LMS, essa API está, fisicamente, dentro do navegador web do aprendiz. Ela precisa, ainda, gerar uma invocação remota para o lado servidor do LMS (implementado em alguma linguagem de programação de alto nível, como PHP, Python ou Ruby).

Quando o servidor processa a requisição ele pode realizar diversas tarefas que não fazem parte do escopo do SCORM: armazenar a pontuação no banco de dados, criar um log para a atividade do usuário etc. O lado servidor, em seguida, precisa retornar algum tipo de resposta para a sua API. Essa API, dependendo da resposta que recebeu do servidor, modificará seu estado interno, e possibilitará ao SCO "perguntar" se, por exemplo, a operação foi realizada com sucesso e, em caso negativo, qual foi o problema. Faz parte do escopo do SCORM definir quais problemas a API deve relatar, e que códigos ela utiliza para relatá-los.

3.3.1 Responsabilidades do LMS

O SCORM requer que o LMS disponibilize uma API que atenda às solicitações do SCO. Essa API protege o objeto dos detalhes envolvidos no processo de comunicação cliente-servidor. A maneira como essa API trocará dados com o servidor não diz respeito ao SCORM (ADL, 2006c). Abaixo seguem os requisitos que um LMS deve atender para garantir compatibilidade com o padrão.

Instância da API

O SCORM requer que o LMS disponibilize ao SCO uma API implementada de acordo com as funcionalidades descritas na seção 3.3.3. Para que um SCO utilize a instância da API algumas restrições são impostas a respeito de onde ela pode ser posicionada no documento HTML. Para garantir que possa localizá-la, essa API deve ser acessível a partir da árvore HTML como um objeto de nome `API_1484_11`.

A implicação prática dessas regras é a seguinte: dentro do documento que o navegador web disponibiliza ao aprendiz há um objeto JavaScript `API_1484_11` com a seguinte definição:

Listagem 3.3: Representação mínima da API SCORM

```
1 var API_1484_11 = (function() {
2   // diversos metodos que devem ser definidos obrigatoriamente
3 })();
```

Para que o SCO possa encontrar esse objeto é preciso que o LMS lance o SCO em uma hierarquia bem definida. Esse objeto deve estar dentro de uma janela filha ou de um *frame* HTML filho da janela do LMS que contêm a instância da API. Três posições são permitidas:

- Em uma cadeia de nodos-pai: neste caso o SCO precisa subir a árvore do documento até encontrar a API. Um caso típico dessa abordagem é lançar o SCO dentro de um *Frameset* HTML e fazê-lo subir apenas um nível para encontrar a API.
- Na "janela-mãe" (*opener*): neste caso o SCO precisa buscar a API dentro da janela que abriu a janela onde ele se encontra (sua "janela mãe" ou *opener*). Um caso típico: tem-se o *opener* como a janela principal do LMS, e o SCO está contido em uma janela "pop-up".
- Em uma cadeia de nodos-pai do *opener*: neste caso o *opener* da janela onde está o SCO não contém a API. Ele próprio é um *frame* dentro de uma janela, e a API está localizada em um de seus nodos-pai.

O SCORM também exige que a instância da API contenha um atributo chamado "version". O valor desse atributo representa a versão da API, e deve ser uma cadeia de caracteres iniciada por "1.0", onde as aspas não fazem parte de seu conteúdo. Esse requisito pode ser traduzido em mais algumas linhas dentro do código JavaScript definido anteriormente:

Listagem 3.4: Representação da API SCORM com atributo "versão"

```
1 var API_1484_11 = (function() {
2   // diversos metodos que devem ser definidos obrigatoriamente
3   this.version = '1.0'
4
5   return {
6     'version': this.version
7   };
8 })();
```

3.3.2 Responsabilidades do SCO

SCOs devem, primariamente, ser capazes de localizar a instância da API. Essa é uma das principais razões pelas quais há restrições a respeito de onde, na hierarquia DOM, o LMS posiciona a instância e porquê existe a imposição de um único nome para a mesma (ADL, 2006c).

Encontrando a instância da API

Como visto na seção 3.3.1 há certas restrições a respeito de onde posicionar a instância da API. Uma vez que essas restrições sejam conhecidas e obedecidas, o seguinte algoritmo deve ser seguido, pelo SCO, a fim de encontrar essa instância dentro da hierarquia DOM.

- Buscar na cadeia de pais da janela atual, se algum existir, até que o elemento raíz seja alcançado
- Buscar na "janela-mãe" ou *opener*, se existir
- Buscar na cadeia de pais da "janela-mãe", se existir, até que o elemento raíz seja alcançado

O SCO deve buscar pela instância dessa maneira, e parar a busca no momento em que encontrá-la. Caso a encontre, o SCO deve invocar, no mínimo, dois métodos dessa instância durante o tempo em que permancer ativo (sua janela não for fechada, por exemplo): `Initialize()` e `Terminate()`.

Requisitos de utilização da API e regras gerais

Um desenvolvedor de SCOs deve ter em mente que, durante a interação do aprendiz com o SCO circunstâncias inesperadas podem provocar o fim da comunicação entre o SCO e o LMS. Embora inesperadas, elas se enquadram em um das seguintes categorias:

- Saída acidental: o aprendiz fechou seu navegador sem querer, por exemplo;
- Ação deliberada: o aprendiz decidiu interromper a interação com o objeto clicando o link que o leva à página inicial do sistema, por exemplo;
- Fim catastrófico: o navegador do usuário terminou sua execução inesperadamente

A maneira pela qual um SCO transmite dados para o LMS é através do método `SetValue()`, fornecido pela instância da API. No entanto, não é possível saber se a instância está acumulando esses dados para posterior envio ou se gera uma requisição toda vez que `SetValue()` é invocada. Caso ela esteja acumulando, o SCO pode forçar a API a submeter as requisições invocando o método `Commit()`. Para evitar-se a perda de dados não submetidos, desenvolvedores devem aderir às seguintes recomendações de programação (ADL, 2006c):

- Invocar `Commit()` sempre que algo significante ocorrer na interação Aprendiz-SCO.
- Não invocar `Commit()` após cada invocação de `SetValue()`, mas somente após um número razoável de chamadas a `SetValue()`.
- Invocar `Terminate()` antes de um SCO ser "retirado" do aprendiz: os navegadores suportam um evento que indica que a página está sendo descarregada, ou seja, que outra requisição está sendo processada e a página atualmente sendo visualizada dará lugar a uma nova. Esse evento é o *onbodyunload*, e deve ser rastreado pelo SCO.

3.3.3 API

A utilização de uma API comum concretiza muitos dos requisitos do SCORM para interoperabilidade e reuso. Ela provê uma maneira padronizada para que SCOs se comuniquem com LMSs, além de proteger o desenvolvedor de SCOs de detalhes a respeito da comunicação entre essas duas entidades. O SCORM não se preocupa com a maneira como é realizada a comunicação entre SCO e LMS, ou seja, uma vez que se respeite os requisitos mínimos da interface da API têm-se garantidos interoperabilidade e reuso (ADL, 2006c).

O padrão SCORM prevê que durante a execução de um SCO é o SCO quem atua ativamente, iniciando a comunicação. O papel do LMS é apenas dar respostas, mas nunca iniciar a comunicação. Em outras palavras, o SCO conhece e invoca métodos da API, enquanto a API não conhece e não invoca métodos do SCO.

Os métodos que a API do LMS deve expor são divididos em três categorias, de acordo com sua utilidade:

- Métodos de sessão: marcam o início e o fim de uma sessão de comunicação entre SCO e LMS;
- Métodos de transferência de dados: são aqueles que o SCO pode utilizar para atualizar valores do modelo de dados;

- Métodos de apoio: utilizados como auxílio para a comunicação (para verificar a ocorrência de erros de transferência, por exemplo).

Todas as chamadas que um SCO realiza a métodos da API inicializam um código de erro dentro da API (exceto pelos métodos de apoio). Por exemplo, uma tentativa de definir-se o tempo que o aprendiz levou para realizar um exercício acabará gerando um código de erro, ainda que esse código tenha valor zero, indicando que nenhum erro ocorreu.

Também é importante ter-se em mente que todos os valores trocados entre SCO e LMS são do tipo string (seqüência de caracteres). Em outras palavras, valores numéricos devem ser sempre cercados por aspas duplas (utiliza-se "1", e não 1, por exemplo).

Para todos os métodos de sessão e transferência de dados, sempre que um erro ocorrer durante o processamento a API deve inicializar seu código de erro interno, o qual poderá ser acessado pelo SCO através do método `GetLastError()`. Ainda mais detalhes podem ser recuperados invocando o método `GetDiagnostic()`.

Métodos de sessão

Um SCO utiliza métodos de sessão para iniciar e terminar a comunicação entre ele e uma instância da API (ADL, 2006c).

- `Initialize()`: invocado para permitir que o LMS realize suas operações de inicialização (que fogem ao escopo do SCORM). O método deve retornar um de dois valores:
  - `true`: indica que a comunicação foi estabelecida com sucesso.
  - `false`: indica que houve problemas durante o estabelecimento da comunicação.
- `Terminate()`: invocado quando o SCO decide que não precisa mais se comunicar com o LMS. Essa chamada deve provocar o envio de quaisquer dados que a instância da API tenha deixado acumulados (através de uma chamada implícita a `Commit()`). Uma vez que este método é invocado o SCO pode continuar apenas invocando métodos de apoio. O método retorna um de dois valores:
  - `true`: indica que a comunicação foi finalizada com sucesso.
  - `false`: indica que a comunicação não pôde ser concluída com sucesso.

Métodos de transferência de dados

Os métodos de transferência de dados são utilizados por um SCO para recuperar (get) e armazenar (set) valores para elementos do modelo de dados SCORM. Os seguintes métodos compõe esta categoria:

- `GetValue()`: solicita dados ao LMS. É possível verificar, entre outras, coisas, (i) o valor de um elemento do modelo de dados suportado pelo LMS, (ii) verificar a versão do modelo de dados do LMS e (iii) verificar se um elemento é suportado pelo modelo de dados do LMS. O método recebe o seguinte parâmetro:
  - Elemento: a identificação completa de um elemento do modelo de dados.

  E retorna um de dois valores:
  - O valor do elemento solicitado.
  - Uma string vazia, se um erro ocorrer. Neste caso o código de erro será definido para o valor adequado.
- `SetValue()`: utilizado para solicitar a definição de um novo valor para um elemento do modelo de dados. O método recebe dois parâmetros:
  - Elemento: a identificação completa de um elemento do modelo de dados.
  - Valor: o valor que se deseja atribuir ao elemento informado.

  E retorna um de dois valores:
  - `true`: se o LMS aceita o valor solicitado para o elemento.
  - `false`: se ocorrer um erro durante a definição do valor do elemento.
- `Commit()`: envia para o LMS quaisquer dados que estejam acumulados na instância da API desde a última invocação de `Initialize()` ou `Commit()`, dependendo de qual foi invocado mais recentemente. O LMS deve, então, definir o valor do código de erro para 0 (zero) e retornar a string true. O método recebe um único parâmetro:
  - Uma string vazia

  E retorna um de dois valores:
  - `true`: se os dados foram enviados e persistidos com sucesso.
  - `false`: se ocorrer um erro durante o envio ou persistência dos dados.

Métodos de apoio

Os métodos de apoio da API existem para que um SCO possa diagnosticar e gerenciar adequadamente erros de comunicação ocorridos em invocações a métodos de sessão ou de transferência de dados. Esses métodos apenas recuperam valores definidos internamente pela API, sem modificá-los:

- `GetLastError()`: retorna o valor atual do código de erro da instância da API. O valor de retorno é uma string (que pode ser convertida para um inteiro entre 0 e 65536). O significado de cada código é definido pelo padrão SCORM.
- `GetErrorString()`: invocado a fim de se conseguir uma descrição textual de um código de erro. O método recebe um único parâmetro, que é o código do erro que se deseja traduzir. A string retornada pelo método (i) não deve possuir mais de 255 caracteres e (ii) caso o código passado como parâmetro não seja parte do padrão SCORM uma string vazia deve ser retornada.
- `GetDiagnostic()`: invocado pela próprio LMS. O padrão SCORM não deixa muito claro qual o seu propósito, uma vez que os outros dois métodos de apoio parecem ser suficientes para a gerência de erros. O único parâmetro do método é uma string de valor livre (que não deve ultrapassar 255 caracteres) e o valor de retorno é também uma string de no máximo 255 caracteres.

3.3.4 Modelo temporal

Para poder especificar com mais clareza algumas variáveis do modelo de dados o padrão SCORM conta com um modelo temporal. Esse modelo define os seguintes conceitos:

- Tentativa do aprendiz (*learner attempt*)
- Sessão do aprendiz (*learner session*)
- Sessão de comunicação (*communication session*)
- Sesssão autenticada (*login session*)

Uma *tentativa* por parte do aprendiz começa quando uma atividade é identificada para entrega (ou seja, o LMS a disponibiliza no navegador web do usuário). Assim que o aprendiz recebe o conteúdo em seu navegador uma *sessão do aprendiz* se inicia. Se o conteúdo em questão

é um SCO (ao contrário de um *Asset*), então essa sessão deve ser rastreada pelo LMS, e nesse momento é iniciada uma *sessão de comunicação*. Uma sessão de comunicação termina quando acaba a comunicação com o LMS. Sessões do aprendiz podem terminar deixando o SCO em um estado suspenso (o aprendiz não concluiu o SCO), ou deixando-o em um estado normal. Para um SCO, a tentativa do aprendiz termina quando o mesmo consegue satisfazer todos os requisitos definidos pelo SCO. Já para um *Asset*, a tentativa termina quando o mesmo é retirado do aprendiz (após um redirecionamento por parte do LMS, por exemplo) (ADL, 2006c).

É importante notar que os dados definidos pelo modelo de dados SCORM estão sempre associados à uma tentativa iniciada por um único aprendiz. Se um SCO for projetado de forma a permitir múltiplas tentativas (e não bloquear o SCO para o aprendiz após a conclusão de uma tentativa), então um novo conjunto de dados deve ser criado para cada nova tentativa iniciada por esse aprendiz. O padrão deixa claro, no entanto, que não é obrigação de um LMS armazenar conjuntos de dados para diferentes tentativas de um aprendiz em um SCO. Apenas uma tentativa deve ser armazenada obrigatoriamente.

3.3.5 Modelo de dados

O Modelo de Dados do ambiente de tempo de execução do SCORM provê um conjunto de elementos que pode ser rastreado por um SCO. Tais elementos podem ser utilizados para registrar, entre outros dados, o estado, a pontuação, as interações e os objetivos de um aprendiz durante uma *tentativa* em um SCO. A tabela 3.1 sumariza a finalidade de cada um dos elementos do modelo de dados. São 24 elementos ao todo.

A descrição dos elementos conta com termos especiais do modelo temporal SCORM, tais como *tentativa* e *sessão*. Para maiores informações consulte a seção 3.3.4.

Comentários do aprendiz

Para ocasiões onde o projetista do conteúdo considera relevante rastrear os comentários do aprendiz para cada SCO com o qual interage, o SCORM prevê a existência do elemento "Comentários do aprendiz" (cmi.comments_from_learner). Não faz parte do escopo do padrão, no entanto, definir quando, onde e nem como os comentários coletados serão utilizados. Fica a cargo do LMS prover meios para análise desses comentários. O comentário do aprendiz refere-se a um único SCO dentro do pacote SCORM, e também fica registrado, junto dessa propriedade, a data em que o comentário foi feito (na forma de um *timestamp*).

Tabela 3.1: Conjunto de elementos do modelo de dados SCORM

Elemento	**Nome**	**Descrição**
cmi.comments_from_learner	Comentários do aprendiz	Contém texto o aprendiz
cmi.comments_from_lms	Comentários do LMS	Contém comentários e anotações que se pretende disponibilizar ao aprendiz
cmi.completion_status	Completo	Indica se o aprendiz completou o SCO
cmi.completion_threshold	Limiar de completude	Identifica um valor contra o qual a medida do progresso que o aprendiz fez tentando completar o SCO pode ser comparada, para determinar se o SCO deve ser considerado completado ou não
cmi.credit	Crédito	Indica se o aprendiz receberá crédito pela performance no SCO em questão
cmi.entry	Acesso	Contém a informação que afirma se o aprendiz já acessou o SCO
cmi.exit	Saída	Indica como ou porquê o aprendiz deixou o SCO
cmi.interactions	Interações	Define informações referentes a uma interação, com propósito de medida ou avaliação
cmi.launch_data	Dados de inicialização	Provê dados específicos a um SCO que o mesmo pode utilizar na sua inicialização
cmi.learned_id	Identificador do aprendiz	Identifica em nome de qual aprendiz a instância do SCO foi inicializada
cmi.learner_name	Nome do aprendiz	Representa o nome do aprendiz
cmi.learner_preference	Preferência do aprendiz	Representa um conjunto de preferências do aprendiz
cmi.location	Localização	Representa a localização (avanço) do aprendiz dentro do SCO
cmi.max_time_allowed	Máximo tempo permitido	Indica a quantidade de tempo acumulado que o aprendiz pode utilizar durante a tentativa
cmi.mode	Modo	Identifica os modos nos quais o SCO pode ser apresentado ao aprendiz
cmi.objectives	Objetivos	Especifica objetivos de aprendizado ou performance associados ao SCO
cmi.progress_measure	Medida de progresso	Identifica uma medida do progresso que o aprendiz obteve tentando completar o SCO
cmi.scaled_passing_score	Pontuação mínima	Indica a pontuação mínima para considerar o SCO concluído para o aprendiz
cmi.score	Pontuação	Indica a pontuação atual do aprendiz
cmi.session_time	Tempo da sessão	Indica o tempo já gasto pelo aprendiz em suas sessões com o SCO
cmi.success_status	Estado de sucesso	Indica se o aprendiz foi aprovado ou reprovado no SCO
cmi.suspend_data	Dados suspensos	Armazena dados suspensos entre interações do aprendiz com o SCO
cmi.time_limit_action	Ação no tempo limite	Indica que ação será tomada quando o tempo limite para conclusão do SCO for alcançado
cmi.total_time	Tempo total	Indica o tempo utilizado pelo aprendiz para conluir a tentativa

Comentários do LMS

A especificação do padrão não deixa muito clara, na opinião do autor, a finalidade deste elemento. Segundo o padrão, o elemento cmi.comments_from_lms contém comentários e anotações que se pretende apresentar para todos os aprendizes que visualizam o SCO em questão. O formato dessa informação não é definido pelo padrão SCORM.

Por se tratarem de "comentários", o autor deste trabalho acredita que sua finalidade seja associar textos que sirvam como comunicação direta entre o instrutor e o aprendiz, mas que não necessariamente façam parte do conteúdo. Por exemplo, "Esse é um conceito muito importante para a prova 2" não se adequa como conteúdo de um SCO, mas sim como comentário feito pelo instrutor para o aprendiz no contexto de um determinado curso, em um determinado semestre.

Completo

Para dar ao projetista de conteúdo uma forma de considerar que o aprendiz concluiu (completou) um SCO o SCORM disponibiliza o elemento cmi.completion_status. É de responsabilidade do projetista de conteúdo definir quando um SCO foi completado. Possíveis valores para esse elemento são "completo" (*complete*), "incompleto" (*incomplete*), "não tentado" (*not attempted*) e "desconhecido" (*unknown*).

Limiar de completude

A propriedade "Limiar de completude" (elemento cmi.completion_threshold) deve ser utilizada em conjunto com a propriedade "Medida de progresso" (elemento cmi.progress_measure). É responsabilidade do projetista do conteúdo associar ao SCO um "Limiar de completude". Durante a interação do aprendiz do SCO, esse limiar é utilizado como o valor que define se o SCO deve ser considerado completo ou não. Por exemplo, se tivermos um limiar igual a seis, o SCO será considerado completo apenas se a medida de progresso do aprendiz for maior ou igual a seis.

Crédito

O padrão define que a propriedade "Crédito" (elemento cmi.credit) pode ser utilizada para especificar se o aprendiz receberá crédito por sua performance durante a interação com um SCO. Isso significa dizer que se o SCO está definido como "sem crédito" (*no credit*), então os valores que determinam o sucesso ou a pontuação do aprendiz não devem ser interpretados pelo

LMS. O valor padrão para um SCO é “crédito”, e seu valor oposto é “sem crédito”.

Acesso

Esta propriedade (elemento cmi.entry) indica se o aprendiz está acessando um SCO pela primeira vez ou se já o acessou ao menos uma vez e está retomando uma sessão de aprendizado. No primeiro caso o valor dessa propriedade é definido como “ab-initio”, e no segundo é definido como “resume” (se o elemento cmi.exit tem como valor “suspend” ou “logout”). Para quaisquer outros casos esse elemento terá um valor vazio (“”).

Saída

Esta propriedade (elemento cmi.exit) indica *como* ou *porquê* o aprendiz deixou um SCO incompleto. Se um aprendiz deixa um SCO incompleto durante sua interação, a única chance de os dados dessa interação permanecerem disponíveis quando ele voltar a interagir com o SCO é se o LMS definiu a propriedade “Saída” com o valor “suspensa” (*suspended*). Para outros valores possíveis todos os dados da interação não estarão disponíveis quando o aprendiz retomar a interação.

Interações

A propriedade “Interações” é uma das mais complexas do modelo de dados SCORM (junto com a propriedade “Objetivos”). Ela é representada pelo elemento cmi.interactions, e tem por finalidade definir uma *coleção* de respostas do aprendiz que podem ser transmitidas do SCO para o LMS.

Essas interações podem ser rastreadas sob duas abordagens distintas:

- Completo (*journaling*): sob este esquema cada interação do aprendiz ganha uma nova entrada na coleção cmi.interactions, o que possibilita futuras análises da evolução do desempenho do aprendiz, e também uma comparação do desempenho de um conjunto de aprendizes.
- Compacto (*status*): sob este esquema apenas a primeira interação do aprendiz ganha uma nova entrada na coleção cmi.interactions - novas interações irão apenas substituir os dados da primeira. Não é possível sequer saber quantas interações o aprendiz gerou durante a sessão.

Cada entrada da coleção cmi.interactions deve armazenar os seguintes dados:

- identificador (*id*): identifica exclusivamente a interação do aprendiz.
- tipo (*type*): indica qual tipo de resposta está sendo gravada e como essa deve ser interpretada. Valores possíveis são:
  - Verdadeira ou falso (*true-false*): a interação possui apenas duas respostas possíveis (*true* ou *false*).
  - Escolha (*choice*): a interação possui um conjunto de duas ou mais respostas possíveis.
  - Texto curto (*fill-in*): a interação requer que o aprendiz preencha um campo com parte de uma palavra, uma palavra ou algumas palavras.
  - Texto longo (*long-fill-in*): a interação requer como resposta um texto com várias palavras.
  - Likert[1] (*likert*): a interação espera que o aprendiz escolha uma entre um conjunto discreto de escolhas em uma escala.
  - Relação (*matching*): a interação consiste em relacionar elementos de um conjunto com elementos de outro.
  - Performance (*performance*): a interação requer que o aprendiz execute uma tarefa em múltiplos passos.
  - Ordenação (*sequencing*): a interação requer que o aprendiz identifique uma ordem lógica para elementos de uma lista.
  - Numérica (*numeric*): a resposta consiste em um número (inteiro ou decimal).
  - Outra (*other*): qualquer outro tipo de interação não definida pelo padrão.
- objetivos (*objectives*): armazena uma coleção de objetivos (vide seção 3.3.5).
- data (*timestamp*): representa o ponto no tempo em que foi dada ao aprendiz a possibilidade de gerar uma resposta.
- respostas_corretas (*correct_responses*): este elemento é uma coleção de outros elementos. Cada elemento dessa coleção possui um elemento "pattern", que corresponde a uma resposta correta dependendo do tipo de resposta definida para a respectiva interação.
- peso (*weighting*): corresponde ao peso dado à resposta, o qual pode ser utilizada para computar uma nota final.

- resposta_do_aprendiz (*learner_response*): corresponde à resposta do aprendiz. Essa resposta deverá atender ao formato referente ao *tipo* desta interação.
- resultado (*resultado*): um julgamento para a resposta do aprendiz. Permite ao SCO, em tempo de execução, avaliar se a resposta é:
  - Correta
  - Incorreta
  - Inesperada
  - Neutra (nem correta nem incorreta)
  - Real (uma estimativa numérica do quão correta é a resposta)
- latência (*latency*): representa quanto tempo o aprendiz levou para gerar uma resposta (desde o momento - indicado por *timestamp* - em que o mesmo tornou-se apto a responder).
- descrição (*description*): uma descrição textual sobre a interação, definida em tempo de execução pelo próprio SCO.

Alguns dos elementos descritos possuem uma grande quantidade de detalhes referentes ao formato permitido para seus valores. Para maiores detalhes a especificação SCORM para o Ambiente de Tempo de Execução (ADL, 2006c) deve ser consultada.

Dados de inicialização

A propriedade "Dados de inicialização" (elemento cmi.launch_data) é definida pelo padrão SCORM como um único valor que deve ser passado para o SCO sempre que esse é inicializado. O padrão não especifica que valores essa propriedade pode assumir, e também não fornece exemplos de uso que poderiam explicar melhor sua finalidade.

Identificador do aprendiz

A propriedade "Identificador do aprendiz" (elemento cmi.learner_id) identifica o aprendiz para o qual o SCO em questão foi lançado. Todos os demais valores de todos os elementos do modelo de dados, portanto, dizem respeito a um único aprendiz. A maneira como esta

[1] A palavra Likert se refere a um formato de questionários criado por Rensis Likert em 1932 (http://en.wikipedia.org/wiki/Likert)

propriedade é atribuída está fora do escopo do padrão. Um LMS pode, por exemplo, utilizar como identificador do aprendiz o mesmo identificador que esse possui, no banco de dados, enquanto usuário do sistema.

Nome do aprendiz

A propriedade "Nome do aprendiz" (elemento cmi.learner_name) permite ao LMS definir um nome para o aprendiz que interage com o SCO em questão. Esse nome pode ser utilizado, por exemplo, para que certas orações dentro do conteúdo dirijam-se diretamente ao aprendiz pelo seu primeiro nome. Não faz parte do escopo do SCORM definir como essa propriedade é inicializada.

Preferência do aprendiz

A propriedade "Preferência do aprendiz" (elemento cmi.learner_preference) permite a um SCO identificar certas preferências a seu respeito. A forma como essas preferências são inicializadas não faz parte do SCORM. Elas podem ser inicializadas, por exemplo, com o preenchimento de um formulário externo ao SCO, provido pelo LMS.

As preferências possíveis são:

- nível do som (*audio_level*)
- linguagem (*language*)
- velocidade de entrega (*delivery_speed*)
- legenda para áudio (*audio_captioning*)

Localização

A propriedade "Localização" (elemento cmi.location) indica a localização do aprendiz dentro do SCO. O termo localização, neste contexto, pode ser entendido como ponto de parada. O valor desta propriedade faz sentido apenas se o aprendiz está retomando a interação com o SCO, pois na sua primeira interação não há ainda um ponto de parada.

O padrão SCORM define que o LMS não deve modificar o valor desta propriedade, cabendo ao SCO gerenciá-lo.

Máximo tempo permitido

A propriedade "Máximo tempo permitido" (elemento cmi.max_time_allowed) indica a quantidade máxima de tempo que um aprendiz pode levar para concluir sua *tentativa* com o SCO. Esse tempo é cumulativo, o que significa que se o aprendiz interrompe sua seção com o SCO antes que o tempo máximo se esgote, o tempo já decorrido é salvo pelo LMS e restaurado no momento em que o aprendiz inicia uma nova interação (retomando a tentativa anterior).

Modo

A propriedade "Modo" (elemento cmi.mode) indica as formas como um SCO pode ser disponibilizado para o aprendiz. Esta propriedade não pode ser modificada pelo SCO em tempo de execução (é somente-leitura). Os possíveis valores são:

- navegação: indica que o SCO é apresentado sem guardar informações sobre a interação do aprendiz.
- normal: indica que o SCO é apresentado de forma normal (gravando as interações do usuário).
- revisão: indica que o SCO é apresentado sem guardar informações sobre a interação do aprendiz *pois esse já completou o SCO em questão*.

O padrão SCORM impõe como responsabilidade do LMS manter esta propriedade em sincronia com a propriedade "Crédito".

Objetivos

A propriedade "Objetivos" (elemento "cmi.objectives") é definida pelo padrão SCORM para que designers instrucionais possam associar quaisquer tipos de objetivos às atividades que criam. O padrão, no entanto, não procura definir *o que é um objetivo*, e nem define restrições quanto ao seu uso. O padrão define, no entanto, como os estados desses objetivos podem ser utilizados para modificar as avaliações utilizadas pelos mecanismos de seqüenciamento SCORM.

Objetivos são associados apenas a SCOs, e um SCO pode possuir zero ou mais objetivos.

Objetivos são rastreados como um conjunto de informações. O padrão SCORM define os seguintes dados associados a um objetivo:

- Identificador (*Identifier*): um identificador para o conjunto de informações.
- Pontuação (*Score*): uma pontuação (se aplicável). Este elemento possui quatro filhos:
  - `scaled`: um valor normalizado entre -1 e 1 (inclusive).
  - `raw`: um valor entre `min` e `max`.
  - `min`: o menor valor válido para `raw`.
  - `max`: o maior valor válido para `raw`.
- Estado de sucesso (*Success Status*): uma indicação do estado de sucesso para um objetivo (se aplicável). Valores possíveis são "passed", "failed" e "unknown" (passou, falhou e desconhecido, respectivamente).
- Estado de completude (*Completion Status*): uma indicação do estado de completude para um objetivo (se aplicável). Valores possíveis são "completed", "incomplete", "not attempted" e "unknown" (completo, incompleto, não tentado e desconhecido, respectivamente).
- Medida do progresso (*Progress Measure*): uma indicação de que o aprendiz está progredindo em direção ao alcance do objetivo. Seu valor é real e varia de 0 a 1.
- Descrição (*Description*): uma breve informação descritiva do objetivo.

Um SCO é livre para atualizar os dados relacionados aos objetivos durante a interação do aprendiz. Além disso, é de responsabilidade do designer de conteúdo definir quando esses valores devem ser alterados: o padrão SCORM, propositalmente, não especifica quando esses valores são modificados, nem porquê.

Quando um aprendiz inicia uma tentativa em um SCO, é responsabilidade do LMS gerenciar o carregamento do conjunto de objetivos definido em XML para aquele SCO. Entre os dados que o LMS inicializará estão `cmi.objectives.n.id`, `cmi.objectives.n.success_status` e `cmi.objectives.n.score.scaled`, onde "n" denota o n-ésimo objetivo associado ao SCO.

Alguns exemplos de uso dos objetivos, por parte do SCO, é mostrado a seguir:

- `GetValue('cmi.objectives.0.id')`
- `SetValue('cmi.objectives.0.score.scaled', '0,75')`
- `SetValue('cmi.objectives.0.success_status', 'passed')`

Medida de progresso

A propriedade “Medida de progresso” (elemento cmi.progress_measure) indica se o aprendiz já concluiu o SCO em questão. O valor zero significa que esse aprendiz ainda não realizou nenhuma tentativa; valores entre zero e um indicam uma tentativa em andamento (e o grau de progresso dessa tentativa, sendo que quanto mais próximo de um, maior foi esse progresso), e o valor um indica que o SCO foi concluído.

Pontuação mínima

A propriedade “Pontuação mínima” (elemento cmi.scaled_passing_score) indica a pontuação mínima para a qual é considerado que o aprendiz concluiu o SCO. Essa pontuação deve ser normalizada entre -1 e 1.

Pontuação

A propriedade “Pontuação” (elemento cmi.score) é dividida em 4 atributos que indicam a pontuação atual do aprendiz dentro do SCO em questão:

- *scaled*: a pontuação em uma escala de -1 a 1.
- *raw*: a pontuação real, com um valor variando de *min* a *max*.
- *min*: o valor mínimo possível para a pontuação *raw*.
- *max*: o valor máximo possível para a pontuação *raw*.

Tempo de sessão

A propriedade “Tempo de sessão” (elemento cmi.session_time) permite ao SCO rastrear quanto tempo o aprendiz leva para completá-lo. O padrão SCORM prevê que é função do próprio SCO gerenciar esse tempo, implementando seu próprio cronômetro, e solicitar ao LMS a atualização do valor dessa propriedade quando julgar necessário.

Estado de sucesso

A propriedade “Estado de sucesso” (elemento cmi.success_status) indica se o aprendiz dominou completamente o SCO. Como o SCO determina isso está fora do escopo do padrão

SCORM. Os possíveis valores para a propriedade são “passou” (*passed*), “reprovou”, (*failed*), e “desconhecido” (*unknown*).

Dados suspensos

A propriedade “Dados suspensos” (elemento cmi.suspended_data) tem por objetivo permitir ao SCO armazenar dados referentes à interação de um aprendiz entre duas sessões diferentes. O LMS não deve alterar o valor dessa propriedade, e o padrão SCORM define como livre o formato e tipo de conteúdo dessa propriedade.

Ação no tempo limite

A propriedade “Ação no tempo limite” (elemento cmi.time_limit_action) indica o que o SCO deve fazer quando o tempo máximo de *tentativa* do aprendiz com o SCO for alcançado. As seguintes ações são permitidas:

- Forçar saída e exibir mensagem ao aprendiz.
- Continuar a execução, mas alertar o aprendiz a respeito do tempo.
- Forçar saída e não exibir mensagem alguma.
- Continuar a execução e não exibir mensagem alguma.

Tempo total

A propriedade “Tempo total” (elemento cmi.total_time) armazena a soma de todo o tempo gasto pelo aprendiz em todas as suas sessões durante uma tentativa.

4 O módulo SCORM do Moodle

4.1 O que é o Moodle

> ”O Moodle é um sistema de gerenciamento de cursos (CMS) - um pacote de software livre e de código aberto, projetado sob a ótica de princípios pedagógicos, para ajudar educadores a criarem comunidades efetivas de aprendizagem online. (...) O Moodle possui uma grande e diversa comunidade de usuários, com mais de 200.000 pessoas registradas em seu site, falando mais de 75 línguas em 175 países.”(MOODLE, 2007)

4.2 Por que tratar do Moodle

Uma das pré-condições deste trabalho era tratar, sempre que possível, de softwares livres. Não em virtude de sua melhor ou pior qualidade quando comparados com alternativas de software proprietário, e sim pela possibilidade que apenas o software livre e de código aberto dá às pessoas a oportunidade de o estudarem, aprenderem com ele e criarem coisas novas.

Ainda assim não fica explicado porque o Moodle foi escolhido, entre todos os LMSs de código aberto existentes. Três LMSs livres foram analisados, além do Moodle, conforme a listagem a seguir.

- Claroline (CLAROLINE, 2007): segundo sua documentação implementa o padrão SCORM versão 1.2 (CLAROLINEDOC..., 2007).
- Sakai (SAKAI, 2007): sua documentação é de difícil acesso (necessita a criação de uma conta para que se possa visualizá-la), e seu site principal não indica que o padrão SCORM é totalmente implementado.
- TelEduc (TELEDUC, 2007): não possui documentação de suas características em seu site principal, e sites externos a respeito do assunto não foram encontrados pelo autor da pesquisa.

De acordo com seus sites oficiais, os projetos Claroline e Sakai têm interesse em implementar o suporte ao SCORM, mas ainda não atingiram resultados tão notáveis quanto os do Moodle.

4.3 Visão geral do Moodle

O Moodle possui um grande número de funcionalidades. Com ele é possível administrar centenas de cursos e usuários, definindo-se categorias para esses cursos e permissões de acesso para esses usuários. Um dos conceitos mais visíveis do Moodle, enquanto software, é seu projeto modular. Um desenvolvedor pode extender o núcleo do sistema, sem afetar seu funcionamento normal, nos seguintes aspectos:

- Blocos: a navegação do sistema é baseada em conteúdo central e blocos. Os blocos envolvem, em sua forma mais simples, uma lista de links navegacionais, que permitem ao usuário navegar pelo sistema.
- Filtros: compreendem filtros de texto. O suporte do Moodle à linguagem LaTeX, por exemplo, é alcançado por meio de filtros.
- Módulos de atividades: são os módulos que permitem ao professor avaliar o aprendiz. Entre os módulos nativos destacam-se Fórum, Tarefa, Wiki e SCORM.

A figura 4.1 ilustra a localização do módulo SCORM dentro da arquitetura do Moodle.

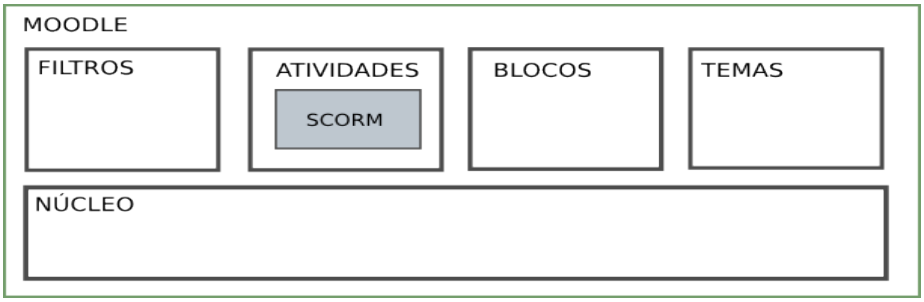

Figura 4.1: Arquitetura Moodle

Módulos de Atividades

Os módulos de atividades se integram ao Moodle por meio de pequenos pacotes de software. Esses pacotes precisam obedecer às normas do Moodle, que são, resumidamente, as seguintes:

- O módulo deve ser criado como um diretório, o qual possui como conteúdo imediato (fora de quaisquer diretórios aninhados) arquivos PHP de nomes pré-definidos pelo Moodle. O nome do diretório indica o nome do módulo.
- Os arquivos que o Moodle impõe ao desenvolvedor do módulo precisam implementar um conjunto mínimo de *funções* PHP.

Na prática o desenvolvedor do módulo possui uma grande liberdade para desenvolver funcionalidades dentro de um módulo de atividades. Do ponto de vista da organização do código, no entanto, essa liberdade custa caro. O desenvolvedor acaba sendo obrigado a misturar código referente à lógica de negócio com o código referente à apresentação dos dados, e isso torna a manutenção bastante complicada.

4.4 Módulo SCORM

As subseções a seguir analisam a implementação do Moodle para o padrão SCORM 2004 3ª Edição.

4.4.1 Adição de um novo pacote

Para utilizar o suporte a SCORM dentro do Moodle é necessário, antes de tudo, adicionar um pacote SCORM a um curso. Os dados mais importantes cadastrados no sistema são os seguintes.

- Gerais
  - Pacote SCORM: o usuário deve escolher um pacote SCORM para realizar o upload.
- Outras configurações
  - Formas de avaliação: define, dentre todas as notas que os SCOs podem dar ao aprendiz, aquela que o Moodle interpretará como nota final do aprendiz nesse pacote.

Não é possível associar metadados com o pacote SCORM. A implementação assume que o arquivo de manifesto do pacote contém todos os metadados apropriados.

4.4.2 Recursos de interface

Uma vez que se tenha cadastrado um pacote SCORM a um curso do Moodle, pode-se começar a interação com ele. O Moodle mantém na página principal do curso a relação de pacotes SCORM disponíveis, com links associados a cada um.

A figura 4.2 apresenta a interface do Moodle quando o aprendiz visualiza um pacote SCORM.

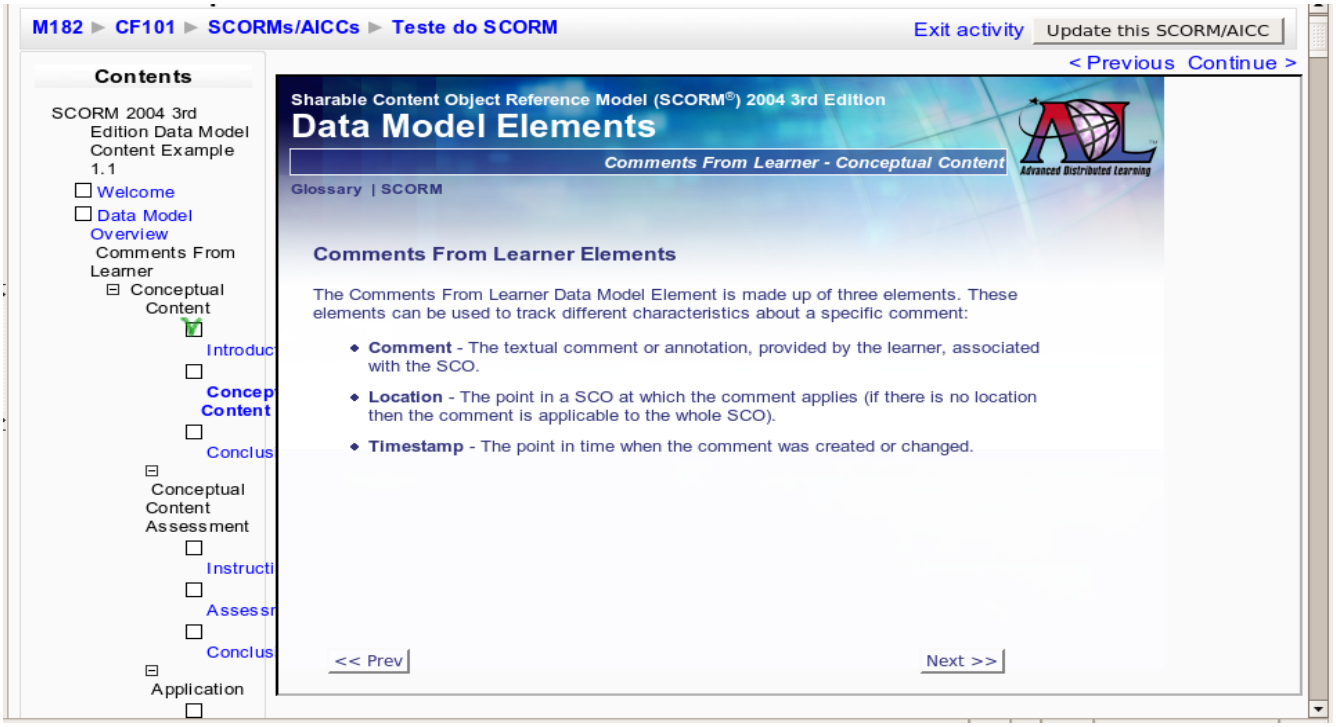

Figura 4.2: Interface com o usuário no módulo SCORM do Moodle

O pacote SCORM que a figura retrata foi obtido do site oficial da ADL, como um pacote que ensina os conceitos do próprio SCORM. Esse pacote possui uma grande quantidade de conteúdo, e a barra de rolagem lateral do navegador dá uma idéia de como o Moodle renderiza uma Árvore de Atividades. Os pontos mais importantes da figura são os seguintes:

- O Moodle utiliza um *frame* HTML como container do objeto de conteúdo (SCO ou Asset) sendo visualizado pelo aprendiz. Essa é uma das maneiras previstas pelo padrão, ou seja, essa prática está em perfeita conformidade com a especificação.

- O Moodle traduz a Árvore de Atividades do pacote em uma lista HTML de atividades aninhadas. Essa lista possui controles que permitem exibir e esconder todos os ramos da árvore. Enquanto as atividades não-folha são exibidas como texto simples, as atividades-folha da árvore são exibidas como links para seu respectivo objeto de conteúdo.

- Dentro da Árvore de Atividades, a referência (link HTML) para a atividade atual é destacada com negrito.

- Uma atividade cujos objetivos já foram cumpridos pelo aprendiz é marcada de forma diferenciada dentro da Árvore de Atividades: um "V" na cor verde acompanha o título da atividade.

- O Moodle apresenta, acima do conteúdo central, o link *Exit activity*, que permitirá ao aprendiz cancelar a tentativa atual na atividade.

- O Moodle apresenta, acima do conteúdo central, links para auxiliar a navegação do usuário, *Previous* e *Next* (anterior e próximo, respectivamente). A especificação SCORM prevê que um LMS deve prover esses elementos visuais mínimos de forma a melhor a usabilidade do aprendiz.

4.4.3 Modelo lógico do banco de dados

A figura 4.3 representa o modelo lógico do módulo SCORM para a versão 1.8.2 do Moodle. A documentação oficial do Moodle não incluía esse modelo. O autor deste trabalho, portanto, decidiu criá-lo e disponibizá-lo para a comunidade [1].

Entre as tabelas pode-se ver que estão incluídos os seguintes dados:

- Rastreamento de valores do modelo de dados do SCORM para cada tentativa do usuário em cada SCO (tabela scorm_scoes_track).

- Regras para definição do seqüenciamento de um Árvore de Atividade (tabelas que possuem "cond" como parte do nome).

- Regras para propagação (*rollup*) dos dados de uma atividade-folha para todas as suas atividades-pai até a atividade-raíz (tabelas que possuem "rollup" no nome).

Não é possível saber, a partir desse modelo lógico do banco de dados, se a implementação do Moodle para o SCORM contempla toda a sua complexidade, mas pode-se assumir que a preocupação dos criadores é suportá-lo por inteiro, uma vez que as estratégias de seqüenciamento são a parte mais complexa do padrão e nem por isso deixaram de ser implementadas.

[1] A imagem 4.3 foi disponibilizada pelo autor no fórum oficial do Moodle para o módulo SCORM, disponível no endereço http://moodle.org/mod/forum/discuss.php?d=81665.

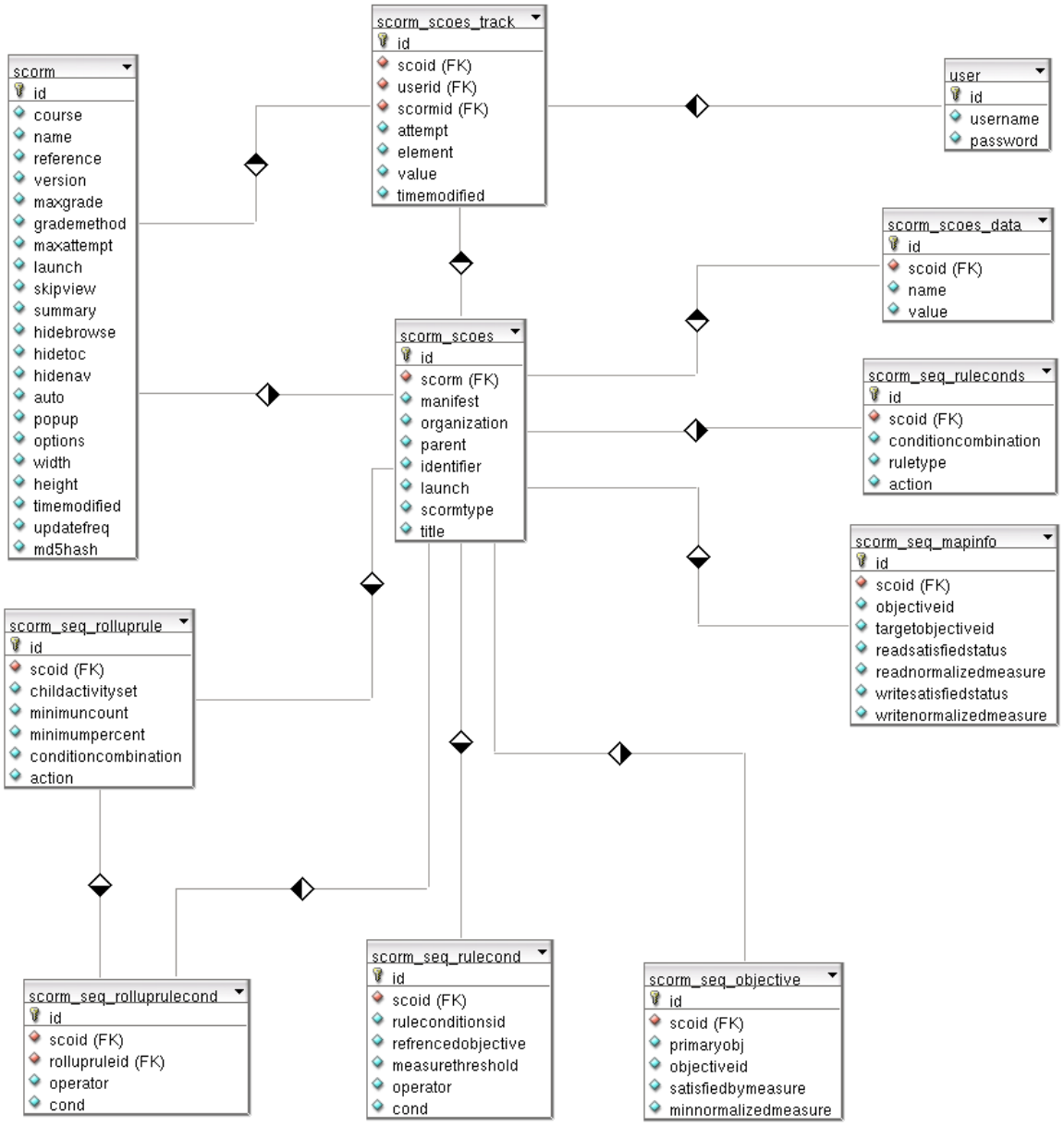

Figura 4.3: Modelo lógico do módulo SCORM para a versão 1.8.2 do Moodle

4.4.4 Scripts que compõe o módulo

A listagem a seguir relaciona os principais scripts presentes no módulo SCORM do Moodle.

- `datamodels/scorm_13lib.php`: Contém funções relacionadas ao processamento do pacote SCORM em tempo de execução.
- `datamodels/scorm_13.js.php`: Contém a implementação da API JavaScript de Tempo de Execução, obrigatória em qualquer LMS de acordo com o padrão.

- `datamodels/sequencinglib.php`: Contém funções relacionadas ao gerenciamento das regras de seqüenciamento das atividades do pacote.
- `index.php`: Script que imprime a listagem de todas as instâncias do módulo associadas ao curso corrente.
- `lib.php`: Contém todas as funções necessárias para tornar este um módulo compatível com o Moodle.
- `locallib.php`: Uma das bibliotecas de funções do módulo.
- `view.php`: Script que exibe uma instância do módulo SCORM. Durante seu processamento uma instância (que se traduz em um pacote SCORM) será exibida ao aprendiz. A árvore de atividades é construída durante esse processamento.
- `request.js`: Pequena biblioteca JavaScript que implementa a função que permite à instância da API SCORM, em tempo de execução, invocar o lado servidor da implementação para comunicar dados. É um problema recorrente em qualquer aplicação web que utilize o paradigma de requisições AJAX, e o Moodle preferiu não utilizar outras bibliotecas que também já implementam essa mesma função.

4.4.5 Código-fonte

O código-fonte do módulo está presente nos scripts relacionados na seção anterior, e em outros no mesmo diretório mas não citados. O código é composto por bibliotecas PHP em sua grande maioria e pequenas bibliotecas JavaScript.

Há muito processamento de XML dentro do código PHP, necessário para que se possa vasculhar o arquivo de manifesto do pacote. O módulo, como possui uma base de dados associada, também manipula procedimentos de instalação, os quais envolvem criação de tabelas com a linguagem SQL.

Em sua grande maioria o código não é documentado. O cabeçalho de suas funções não possui documentação. Muitas funções manipulam estruturas bastante complexas, as quais são passadas de função para função.

4.4.6 Documentação

A listagem abaixo relaciona os tipos de documentação vinculadas ao módulo que o autor tentou encontrar, e o respectivo resultado da procura.

- **Diagramas UML**: diagramas de seqüência seriam úteis para explicar o funcionamento geral do módulo. Nenhum desses diagramas foi encontrado.

- **Modelo lógico do banco de dados**: o modelo lógico do banco de dados, na pior das hipóteses, deixa explícito que informações um sistema armazena. O site docs.moodle.org, que é o wiki oficial do Moodle, possuía o modelo da versão 1.7 do SCORM, mas não da 1.8. Esse modelo, portanto, foi elaborado manualmente pelo autor a partir da versão 1.7, e em seguida foi disponibilizado no fórum referente ao módulo SCORM dentro do site moodle.org.

- **Documentação do código**: a documentação do código-fonte, se não inexistente, é mínima. Em geral algumas instruções são documentadas (como desvios condicionais e atribuições de variáveis), mas as funções não são devidamente documentadas (o que incluiria uma descrição de seu propósito, de seus valores de entrada e de seus possíveis valores de saída).

O autor deste trabalho tentou entrar em contato por duas vezes com o autor do módulo SCORM, Roberto Pinna, para oferecer ajuda na ampliação da documentação, mas *não obteve resposta*.

4.4.7 Problemas observados na implementação

Antes de realizar uma análise da implementação do padrão é preciso que fique claro que o autor não pôde verificar se toda a complexidade do SCORM foi implementada. Esta seção, portanto, trata o módulo com um certo nível de abstração.

A respeito da interface com o usuário a implementação parece muito boa. A comunicação entre o SCO e o LMS funciona (ao menos para os casos mais simples de rastreamento). A Árvore de Atividades é montada de forma bastante intuitiva para o usuário. Os elementos de interface que permitem a navegação para a próxima atividade e para a atividade anterior também estão presentes. Um problema observado foi a sobreposição do *frame* HTML sobre a Árvore de Atividades quando a janela do navegador web é reduzida horizontalmente.

No que diz respeito ao código-fonte, no entanto, os problemas são muitos, e sérios, a saber:

- **Documentação mínima**: as funções do módulo não são documentadas de forma adequada. Uma análise superficial de uma função do módulo não nos permite saber, exatamente, o quê a função faz, nem quais são os tipos de seus parâmetros de entrada e nem o tipo de seus valores de retorno.

- **Passagem de valores por referência**: muitas estruturas utilizadas dentro da implementação são tuplas (variáveis que guardam mais de um valor). Quando essas variáveis são passadas de uma função para a outra, há momentos em que novas propriedades são anexadas à variável, transformando-a numa estrutura difícil de rastrear: que informações ela possui *neste exato momento?*. Essa prática prejudica muito o entendimento do código, e a falta de documentação, neste caso, torna o trabalho de estudar o código um pesadelo.

- **Funções que geram efeitos colaterais**: há funções dentro da implementação que realizam tarefas das quais outras funções dependem. Por exemplo, no corpo da função A é invocada a função B, que deve validar um pacote SCORM. Na linha seguinte, ainda no contexto A, movem-se arquivos para um diretório que, antes da função B ser invocada, não existia. Ou seja, a função B, que deveria estar apenas realizando validações, criou um diretório no meio do seu processamento (esse é o chamado "efeito colateral"). Esse tipo de estruturação de código prejudica enormemente o entendimento do código, pois cria dependências não-explícitas entre funções.

- **Falta de uma arquitetura bem definida**: este problema está presente em todos os módulos do Moodle: há trechos de código que executam lógica de negócio, acesso ao banco de dados e geração de HTML. Isso torna o código difícil de manter, pois não se sabe exatamente onde determinada funcionalidade foi implementada, uma vez que todo script parece ser uma local possível. Paradigmas que forçam a separação de responsabilidades, como o MVC, contribuem para a melhor estruturação do código.

5 *Proposta de implementação do padrão SCORM*

Enquantos os demais capítulos do presente trabalho foram dedicados ao estudo do conceito de Objetos de Aprendizagem, do padrão SCORM e do módulo SCORM do Moodle, este propõe-se a estudar uma alternativa de implementação do padrão. Para "implementação do padrão SCORM" pode-se atribuir dois significados:

- Implementar um curso (conteúdo) em conformidade com as especificações do padrão SCORM
- Implementar um sistema (LMS) em conformidade com as especificações do padrão SCORM

Deste ponto em diante, portanto, o autor utiliza a expressão "implementação do padrão SCORM" como sinônimo para "implementação de um LMS em conformidade com as especificações do padrão SCORM".

Esta proposta apresenta e compara três frameworks para desenvolvimento de aplicações web, e por fim escolhe um desses como a melhor alternativa para a implementação do padrão SCORM.

5.1 Motivações

Entre as motivações para a criação de um novo LMS capaz de suportar o padrão SCORM destacam-se as seguintes:

- Os problemas encontrados na implementação do Moodle para o padrão (vide capítulo 4).
- Popularizar o suporte ao padrão SCORM entre LMSs de código aberto (considerando que o LMS construído como resultado desta proposta seja também um software de código aberto).

- Verificar possíveis problemas dentro do padrão, que de outro modo não poderiam ser verificadas. Por "problemas" entende-se a definição de uma característica para o sistema que não pode ser implementada por um LMS em função de:
  - Conflito com outra definição da mesma especificação
  - Falta de clareza na sua definição
  - Não-definição de um comportamento essencial, o que pode levar a implementações distintas

5.2 Fundamentos da proposta

Propor a implementação de um sistema novo para realizar tarefas que outros sistemas já realizam nem sempre é uma boa idéia, mesmo no meio acadêmico. O tempo e os custos envolvidos na produção de um sistema de médio porte podem ser altos demais se as tecnologias corretas não forem utilizadas. Ou seja, o novo sistema pode se tornar um "novo problema", em vez de uma nova solução.

Pode-se reduzir drasticamente o custo e o tempo de execução de um projeto de software ao se optar pela reutilização de recursos existentes. Entre esses recursos, que podem ir desde reutilização de conhecimento a reutilização de projetos gráficos, esta proposta se fundamenta e se foca apenas na reutilização do ponto de vista da engenharia de software.

Ao longo da história recente da engenharia de software a questão da reutilização ganhou bastante atenção por atender necessidades inerentes à maioria dos projetos de software:

- Reduzir os custos de mão-de-obra vinculados à criação de código-fonte
- Reduzir o tempo gasto com a criação de código-fonte
- Reduzir a quantidade de *bugs* presentes nos sistemas, ao reutilizar-se codigo-fonte testado em projetos anteriores

O reuso de software acabou motivando a (ou talvez tenha sido resultado da) evolução das próprias linguagens de programação modernas. O reuso, em princípio, estava expresso na possibilidade de se criar funções, as quais podem ser invocadas diversas vezes dentro de um mesmo programa. Com o advento do paradigma de programação orientada a objetos as linguagens passaram a suportar também o uso de classes, as quais encapsulam funções que operam sobre dados internos a uma instância dessa classe. A orientação a objetos, embora por si só não torne

o reuso de código maior, foi mais um passo importante. Ela foi, provavelmente, a grande responsável pela proliferação da técnica que permite a reutilização de uma arquitetura inteira de software: o framework.

> Um projeto abstrato orientado a objetos, também conhecido como framework, consiste de uma classe abstrata para cada componente significativo. As interfaces entre os componentes do projeto são definidas em termos de um conjunto de mensagens. (...) A habilidade dos frameworks de permitir a extensão de componentes já existentes em bibliotecas é uma de suas principais forças. (...) Frameworks possibilitam o reuso de código na mais alta granularidade e costumam ser específicos a uma área de aplicações (FOOTE; JOHNSON, 1991).

5.3 A arquitetura Model View Controller

No domínio das aplicações baseadas na web, onde se inclui o LMS, pode-se encontrar hoje diversos frameworks, escritos em diversas linguagens de programação, alicerçados em diferentes princípios. Um desses princípios é o da divisão de uma arquitetura de software em três partes distintas: uma responsável pela lógica de negócio (conhecida como Controller ou controle), outra responsável pela apresentação de informações e captura de eventos do usuário (conhecida como View ou visão) e outra responsável por controlar o acesso aos dados do sistema (conhecida como Model ou modelo). Esse conceito é conhecido como Model View Controller (Modelo Visão Controle, referenciado a partir de agora apenas como MVC) e, embora antigo na computação, ganhou mais atenção dos desenvolvedores web apenas nos anos 2000, com a popularização de frameworks projetados sobre esse conceito.

> No MVC, o elemento do domínio é chamado de modelo. Objetos do modelo desconhecem a interface com o usuário. (...) A parte de apresentação do MVC é composta pelos dois elementos restantes: visão e controle. A função do controle é capturar dados gerados pela interação do usuário com a interface e descobrir o que deve ser feito com isso (FOWLER, 2006).

A definição proposta por Fowler é fundamentada na implementação do MVC dentro de um ambiente Smalltalk, e por isso une controle e visão como “apresentação”. Não é exatamente esse o conceito de MVC que os frameworks web implementam. Esses frameworks separam os papéis desempenhados pelo View e pelo Controller como camadas distintas.

5.4 Frameworks MVC para desenvolvimento web

A arquitetura MVC é uma das mais utilizadas, senão a mais utilizada, para o desenvolvimento de sistemas baseados na web. Sua disseminação foi tão grande que tornou-se difícil escolher entre todas as alternativas existentes, em diversas linguagens. Esta seção tem por objetivo apresentar e destacar as principais características de três frameworks dessa categoria.

Os critérios de escolha desses frameworks foram os seguintes:

- São estáveis (estando próximos ou além da versão 1.0)
- Seus criadores os liberaram como software livre
- Possuem extensa documentação de sua API e de sua forma de uso, não apenas em seu site principal, mas em blogs de programadores ao redor do mundo
- São desenvolvidos sobre linguagens populares
- São desenvolvidos sobre linguagens de tipagem dinâmica (mais especificamente, onde o programador não precisa declarar os tipos das variáveis que utiliza em seu programa)
- São desenvolvidos sobre linguagens liberadas, pelos seus criadores, como software livre
- Possuem bom suporte para metodologias de desenvolvimento orientadas por testes

Os frameworks escolhidos para análise foram:

- CakePHP, versão 1.1
- Django, versão 0.96
- Ruby on Rails (referenciado deste ponto em diante apenas como Rails), versão 1.2.3

Todos os frameworks analisados possuem diversos conceitos comuns, que são destacados nesta seção para evitar repetições nas seções que descrevem suas características individuais.

- Todos possuem integração com uma biblioteca de software que implementa o padrão de projeto Active Record (FOWLER, 2003), que busca tratar cada *tabela* do banco de um banco de dados relacional como uma *classe* dentro de um paradigma orientado a objetos, e cada *registro* de uma tabela como um *objeto* dentro desse mesmo paradigma.

  - Para utilizar a biblioteca em cada um dos frameworks é preciso que se crie, de forma geral, uma classe para representar cada tabela do banco de dados.
  - Essa classe passa a compor a camada de modelos (*Model*) dos frameworks, e possui métodos que permitem, por exemplo, buscar registros dentro do banco de dados.
  - Uma vez que um registro é encontrado por uma classe *Model*, esse registro é transformado na classe correspondente, e ao longo do código (seja ele Ruby, Python ou PHP) passa a ser tratado como um objeto, o qual possui métodos capazes de modificar seus dados dentro do banco de dados (no caso do CakePHP, embora o projeto do framework seja fundamentado no MVC, essa transformação pode não ocorrer em determinadas circunstâncias).

- Django e Rails, após instalados no sistema operacional, tornam global um comando que permite ao desenvolvedor criar uma aplicação mínima dentro do framework. No caso do Rails esse comando cria uma grande estrutura de diretórios e arquivos, e para o Django poucos arquivos são criados. Uma vez que o programador se familiarize com a estrutura de diretórios de seu framework favorito, criar novas aplicações já não significa mais criar uma aplicação a partir do nada, e sim criá-la a partir de uma estrutura bem definida. No caso do CakePHP esse comando não fica disponível pois o framework não é, de fato, instalado (sempre que se deseja criar uma aplicação nova é preciso descompactar a estrutura padrão disponível no site do projeto).

  - A estrutura de diretórios *força* a localização física de certos trechos de código. Tome-se como exemplo o caso do Rails. Para qualquer aplicação Rails a configuração do acesso ao banco de dados estará sempre no mesmo local, a declaração dos *Controllers* estará sempre no mesmo local etc. O programador deve estudar a documentação do framework para saber *onde* declarar cada parte de sua aplicação.

- Todos possuem uma forma padronizada de transformar requisições web (na forma de URLs) em uma ação executada dentro da camada de lógica de negócio. Essa camada pode ser dividida em classes ou arquivos. Por exemplo, para o Rails, a classe UsersController pode ficar encarregada de encapsular a lógica de negócio associada a usuários. Parâmetros adicionais da URL podem ser convertidos diretamente em *métodos* dessa classe, como um método que carrega um usuário e o prepara para ser formatado pelo respectivo *View*, por exemplo.

- Todos possuem uma forma simples de "instalar a aplicação", que no caso de uma aplicação web significa trazer o banco de dados para um estado onde todas as tabelas definidas na

aplicação estejam criadas, de acordo, no banco de dados. Essa definição pode estar mais ou menos vinculada à linguagem SQL, e pode ter uma forma mais ou menos simples para gerenciar mudanças na definição dessas tabelas (sejam elas modificações em propriedades de tabelas já existentes ou adição de novas tabelas dentro da aplicação). A facilidade para tratar essa *evolução do banco de dados* é vital em um framework, pois indica:

  - Se o framework força o programador a definir as tabelas em termos de SQL, e assim força a escolha de um Sistema Gerenciador de Banco de Dados (SGBD) antes mesmo de se iniciar a construção da aplicação. Alternativamente, o framework pode prover uma outra linguagem para definição de tabelas, e tal linguagem passa a abstrair a linguagem SQL, tornando a aplicação *portável*: qualquer SGBD pode ser utilizado pela aplicação.
  - Se o framework facilita a criação de *versões do esquema do banco de dados*. Esse é um fator vital, pois pode significar o alívio ou o pesadelo do programador que mantém o sistema. Um framework que possui suporte a esse versionamento facilita a criação de novas versões da aplicação.

- Todos fundamentam-se na filosofia "Não se Repita" ou DRY (WIKIPEDIA, 2007b)

5.4.1 CakePHP

O CakePHP é, de acordo com seu site (CAKEPHP, 2007), um framework para desenvolvimento rápido que utiliza padrões de projeto comumente conhecidos como Active Record, Association Data Mapping, Front Controller e MVC. Seu objetivo principal é prover um framework estruturado que permita a programadores PHP de todos os níveis desenvolvedor rapidamente aplicações web robustas sem perda de flexibilidade.

O framework é escrito sobre a linguagem PHP (PHP, 2007) e, de acordo com Wikipedia (2007a), seu projeto foi inspirado nos conceitos do Ruby on Rails.

Nomenclatura do framework

Para o CakePHP a nomenclatura original do MVC se aplica sem exceções, a exemplo do Rails.

Arquitetura básica

A arquitetura básica do processamento de requisições do CakePHP é descrita a seguir:

- Uma requisição a uma aplicação CakePHP (na forma de um endereço web ou URL) é processada por um arquivo de rotas, o qual define, por meio de regras no formato string, como uma URL é traduzida na invocação de um Controller. O importante a notar aqui é que o framework provê regras default que, em geral, são exatamente o que o desenvolvedor precisa. A URL meusite.com/usuarios/ver/1, por exemplo, seria traduzida, sem necessidade de configuração, na invocação do método "ver" da classe "UsuariosController", tendo como variável local "id", contando com o valor 1 (o mesmo comportamento é observado no Rails).

- Dentro do método de um Controller a lógica de negócio acontece. Os parâmetros provindos da URL ganham um significado (como o número 1 da url meusite.com/usuarios/ver/1, que dentro da função *ver* poderia ser encarado como o identificador do usuário dentro do banco de dados). Os métodos dentro de um Controller ficam responsáveis por interagir com o banco de dados através de uma *camada de abstração*. Os dados são preparados para que um View os formate dentro de uma estrutura HTML. Todos os dados inicializados em um método do Controller devem ser passados *explicitamente* para um View.

- Agora um View, que consiste em um arquivo HTML com algumas características especiais, fica encarregado de colocar entre as tags HTML as propriedades dos objetos inicializados em seu respectivo método do Controller.

Definindo tabelas

Até sua versão 1.1, que é a última versão estável, as tabelas de uma aplicação CakePHP precisam ser definidas em SQL puro. O site oficial indica que a versão em desenvolvimento, 1.2, inclui o recurso de migrações. Para maiores informações sobre migrações consulte a seção 5.4.3.

Definindo Models

As regras para definição de modelos do CakePHP são similares às do Rails. Consulte a seção 5.4.3 para maiores detalhes.

Versionamento do banco de dados

Como a definição de tabelas no CakePHP é feito em SQL puro, o controle de versionamento não fica integrado ao framework, cabendo ao desenvolvedor criar um esquema de ver-

sionamento diferente para cada projeto criado com o framework. Na prática, portanto, não há versionamento do banco de dados no CakePHP.

Suporte a testes de software

O suporte a testes de software no framework consiste em testes de Models. Não há suporte direto, ainda, para testes de Controllers e Views.

Programação Interativa

O PHP, até sua última versão, não conta com um Interpretador Interativo. O CakePHP também não o implementa e, portanto, a programação interativa não é suportada nesse framework.

5.4.2 Django

O Django é, de acordo com seu site (DJANGO, 2007), um framework de alto nível escrito com a linguagem de programação Python (PYTHON, 2007), que encoraja o desenvolvimento rápido e projeto limpo e pragmático.

Nomenclatura do framework

As três camadas da arquitetura MVC original ganham nomes diferentes dentro do Django, a saber:

- Um *Controller* do MVC é chamado de *View* no Django
- Um *View* do MVC é chamado de *Template* no Django

Um Model do MVC continua sendo chamado de Model dentro do Django.

Arquitetura básica

A arquitetura básica do processamento de requisições do Django é descrita a seguir:

- Uma requisição a uma aplicação Django (na forma de um endereço web ou URL) é processada por um arquivo de rotas, o qual define, por meio de expressões regulares, como

uma URL é traduzida na invocação de um Django View. Por exemplo, a URL meusite.com/usuarios/ver/1 pode ser traduzida como "invoque a função *ver* declarada no arquivo ver/views.py, dentro do diretório principal da aplicação".

- Dentro de uma função do Django View, a lógica de negócio acontece. Os parâmetros provindos da URL ganham um significado (como o número 1 da url meusite.com/usuarios/ver/1, que dentro da função *ver* poderia ser encarado como o identificador do usuário dentro do banco de dados). As funções dentro de um Django View ficam resposáveis por interagir com o banco de dados através de uma *camada de abstração*. Os dados são preparados para que um Django Template os formate dentro de uma estrutura HTML. Todos os dados inicializados em um Django View devem ser transmitidos explicitamente para um Django Template, caso contrário não estarão disponíveis fora do contexto do Django View.
- Agora um Django Template, que consiste em um arquivo HTML com algumas características especiais, fica encarregado de colocar entre as tags HTML as propriedades dos objetos inicializados em seu respectivo Django View.

Definindo tabelas e Models

As tabelas de uma aplicação Django não são definidas em termos de SQL, e sim na própria linguagem Python. A listagem (**??**) exemplifica a definição de duas tabelas a partir de duas classes Python, as quais estendem obrigatoriamente models.Model.

Listagem 5.1: Definição de Models no Django

```
1 from django.db import models
2
3 class User(models.Model):
4     username = models.CharField(maxlength=20)
5     password = models.CharField(maxlength=32)
6     last_access = models.DateTimeField('last access')
7
8 class SCORMPackage(models.Model):
9     name = models.CharField(maxlength=60)
```

Uma vez definidas as classes Model, um subcomando do Django *converte* as definições de classe em tabelas de um banco de dados relacional. Como essas definições são feitas em código Python, a aplicação fica independente de SGBD, sendo que esse passa a ser escolhido a partir da configuração de uma das propriedades da aplicação Django em questão.

Em resumo, *tabelas e Models são definidos, em Django, com o mesmo trecho de código*.

Versionamento do banco de dados

Uma conseqüência da conversão de um Django Model em uma tabela do banco de dados é a perda do versionamento do banco de dados. O Django ainda não suporte uma evolução continua para o projeto, capaz de associar mudanças nos Models a versões do projeto. Essa, no entanto, é uma preocupação dos criadores do framework, e deve ser atendida nas próximas versões.

Suporte a testes de software

O suporte a testes no Django inclui testes de Models, de Django Views e, aparentemente, de Django Templates, a partir das seguintes formas:

- Doctests: são trechos de código embutidos nas *docstrings* Python de uma função
- Unittest: neste caso os testes são definidos dentro de métodos de uma classe herdeira de unittest.TestCase

Programação interativa

Os pacotes de software que incluem a linguagem Python também incluem um Interpretador Interativo, o qual avalia expressões Python de forma interativa. Dentro do Django é possível carregar o interpretador, e em seguida carregar as definições dos Models para dentro do ambiente de execução do interpretador. Em seguida pode-se instanciar as classes e testar seu comportamento interativamente.

5.4.3 Rails

Ruby on Rails é, de acordo com seu site oficial (RUBY..., 2007a), um framework web de código aberto otimizado para promover a felicidade do programador e uma produtividade sustentável. Ele lhe permite escolher código belo ao promover *convenção em vez de configuração*.

O framework é desenvolvido sobre a linguagem Ruby (RUBY..., 2007b).

Nomenclatura do framework

Para o Rails a nomenclatura original do MVC se aplica sem exceções. Portanto, quando um Controller é mencionado, sabe-se que o significado original foi mantido (ao contrário do

framework Django, por exemplo).

Arquitetura básica

A arquitetura básica do processamento de requisições do Rails é descrita a seguir:

- Uma requisição a uma aplicação Rails (na forma de um endereço web ou URL) é processada por um arquivo de rotas, o qual define, por meio de regras no formato string, como uma URL é traduzida na invocação de um Controller. O importante a notar aqui é que o framework provê regras default que, em geral, são exatamente o que o desenvolvedor precisa. A URL meusite.com/usuarios/ver/1, por exemplo, seria traduzida, sem necessidade de configuração, na invocação do método "ver" da classe "UsuariosController", tendo como variável local "id", contando com o valor 1.

- Dentro do método de um Controller a lógica de negócio acontece. Os parâmetros provindos da URL ganham um significado (como o número 1 da url meusite.com/usuarios/ver/1, que dentro da função *ver* poderia ser encarado como o identificador do usuário dentro do banco de dados). Os métodos dentro de um Controller ficam resposáveis por interagir com o banco de dados através de uma *camada de abstração*. Os dados são preparados para que um View os formate dentro de uma estrutura HTML. Todos os dados inicializados em um método do Controller como variáveis de instância passam a estar automaticamente disponíveis no View, sem necessidade de passá-los explicitamente.

- Agora um View, que consiste em um arquivo HTML com algumas características especiais, fica encarregado de colocar entre as tags HTML as propriedades dos objetos inicializados em seu respectivo método do Controller.

Definindo tabelas

Tabelas são definidas no Rails utilizando puro código Ruby, através de uma biblioteca específica do Rails. As tabelas de um projeto não precisam ser todas concebidas ao mesmo tempo. O programador utiliza as chamadas migrações (*migrations*) para definir novas tabelas. As migrações são numeradas, e representam a versão na qual o banco de dados se encontra. A listagem 5.2 ilustra a definição de uma tabela.

O Rails conta com o suporte do programa Rake para registrar *tarefas* dentro de um projeto. Quando uma nova migração é criada, uma nova tarefa é automaticamente registrada. Quando o programador roda o Rake, o programa compara se a versão do banco de dados é menor que a

versão da última versão disponível. Se isso for verdade, todas as migrações são executadas, até que a versão do banco de dados se igual ao número de versão da última migração disponível.

Listagem 5.2: Definição de tabelas no Rails

```
1  class CreateUsers < ActiveRecord::Migration
2    def self.up
3      create_table :users do |t|
4        t.column "username",     :string,   :limit => 20
5        t.column "password",     :string,   :limit => 32
6        t.column "last_access", :datetime
7      end
8    end
9
10   def self.down
11     drop_table :users
12   end
13 end
```

A migração conta com dois métodos, um que define o que acontece quando a versão está sendo instalada (self.up) e outro que define o que acontece quando a versão está sendo revertida (self.down). Reverter uma migração significa voltar a versão do banco de dados, por exemplo, da 5 para a 4. Isso costuma ser útil durante o desenvolvimento.

Definindo Models

Em Rails, um modelo pode ser definido com a simples declaração de uma classe. A convenção do framework é que o nome de uma classe deve ser definido no singular, e automaticamente o framework assumirá uma correspondência com uma tabela do banco de dados, a qual possui o mesmo nome que a classe, porém na forma plural. A listagem (**??**) demonstra esse comportamento. A classe "User" corresponde à tabela "users" do banco de dados.

Listagem 5.3: Definição de Models no Rails

```
1  class Users < ActiveRecord::Base
2  end
```

Embora o código não pareça fazer muito coisa, o Rails se encarrega de definir como atributos da classe, dinamicamente, todas as colunas da respectiva tabela.

Versionamento do banco de dados

O versionamento do banco de dados é garantido pelas migrações. Outro benefício do uso de migrações para a definição de tabelas é a independência de SGBD. Uma simples configuração indica ao Rails que SGBD se deseja utilizar antes de executar todas as migrações.

Suporte a testes de software

O Rails suporta testes de software por meio de diversas bibliotecas, entre elas a biblioteca RSpec (RSPEC, 2007). Essa biblioteca permite ao programador definir testes por meio de especificações. Essa metodologia ganhou o nome de BDD (*Behavior Driven Development*), e ainda não ganhou implementações em outras linguagens de programação além de Ruby.

No Rails é possível testar todas as 3 camadas do MVC. Utilizando a biblioteca de testes embutida no Rails, os testes das camadas de Controller e View são chamados *testes funcionais*, e os testes da camada Model são chamados *testes unitários*.

A utilização da biblioteca RSpec permite testar as 3 camadas de forma independente, e dessa maneira isolar melhor as intenções de cada teste.

Programação interativa

Os pacotes de software que incluem a linguagem Ruby incluem um Interpretador Interativo, chamado IRB, o qual avalia expressões Ruby de forma interativa. Dentro do Rails é possível abrir uma versão customizada do IRB (*Interactive Ruby Shell*), a qual possui disponível todos os Models e Controllers, os quais podem ser testados de forma interativa.

5.5 Comparativo entre os frameworks

Uma vez que se tem uma descrição das características principais dos três frameworks, é possível compará-los sob certas óticas e decidir qual deles é o mais apropriado para a implementação de um LMS adequado ao padrão SCORM. Entre as características comparadas, todas possuem, para o autor, a mesma importância. O autor procurou incluir no comparativo as características que considerou mais relevantes para a tomada de decisão. Análises sobre outras óticas poderiam, possivelmente, apontar uma escolha final diferente da escolha do autor.

A tabela 5.1 apresenta o resultado da comparação dos frameworks CakePHP, Django e Rails.

Tabela 5.1: Comparativo entre os frameworks CakePHP, Django e Rails

	CakePHP	**Django**	**Ruby on Rails**
Definição de tabelas	SQL Puro	Python	Ruby
Versionamento do banco de dados	Sem suporte	Sem suporte	Com suporte
Suporte a testes de software	Apenas Models	Models e Controllers	Models, Controllers e Views e suporte a BDD
Programação interativa	Sem suporte	Com suporte (Python Console)	Com suporte (Interactive Ruby Shell)
Livros sobre o assunto	0	1 (online)	Mais de 15

O quadro comparativo mostra que o framework que mais deixa a desejar é o CakePHP, com as seguintes desvantages aparentes:

- Definição de tabelas com SQL puro, dificultando o alcance da independência de SGBD
- Suporte ainda fraco ao desenvolvimento orientado por testes (apenas os Models podem ser testados)
- Sem suporte a programação interativa (pois não foi criado um "PHP Console" ou um "Interactive PHP Shell")
- A ausência de um livro sobre o framework

Dessa forma resta ainda a comparação dos dois frameworks mais completos, Django e Rails. Em todas as categorias analisadas houve vantagem para o Rails ou empate.

- Definição de tabelas: no Django é feito com código Python, mas esse código também é utilizado como definição do Model. No Rails a definição das tabelas é feita em Ruby, e o código de definição do Model é localizado em outro ponto. Para o autor essa é uma vantagem do Rails, pois contribui com o versionamento e com a separação de propósitos.
- Versionamento do banco de dados: enquanto o Django ainda não o suporta diretamente, o Rails o suporta e incentiva.
- Suporte a testes de software: o suporte do Django é bom, mas o do Rails é mais completo.
- Programação interativa: nesta categoria há um empate, já que as soluções de ambos são boas.
- Livros sobre o assunto: o Rails possui muitos livros mais que o Django, embora os dois frameworks tenham quase o mesmo tempo de vida.

O autor considera, portanto, como opção mais adequada para a criação de um LMS compatível com SCORM o framework Ruby on Rails.

6 Conclusões

A pesquisa identificou que o conceito de Objetos de Aprendizagem surgiu, essencialmente, da necessidade de se construir conteúdos de aprendizagem reutilizáveis. A reutilização tem como um de seus principais objetivos diminuir os custos financeiros envolvidos na construção de materiais de ensino-aprendizagem.

Entre os padrões desenvolvidos para definição de Objetos de Aprendizagem a pesquisa identificou o padrão SCORM (Modelo de Referência para Objetos de Conteúdo Compartilháveis) como o padrão *de facto*. Esse padrão, em sua versão 2004 3ª Edição, possui três grandes especificações: um Modelo de Agregação de Conteúdo (*Content Aggregation Model*), que descreve como um pacote SCORM deve ser definido em termos de organização de conteúdos digitais e metadados que identificam cada um desses conteúdos e seus relacionamentos; um Ambiente de Tempo de Execução (*Run-Time Environment*), que permite a troca de informações entre um Objeto de Conteúdo SCORM e um LMS (o sistema de software que disponibiliza esses conteúdos ao aprendiz) através de uma API (*Application Programming Interface*) definida na linguagem de programação ECMAScript; e uma especificação para Seqüenciamento e Navegação (*Sequencing and Navigation*), a qual define como um pacote SCORM pode ser organizado de modo a aceitar diferentes esquemas de seqüenciamento para aprendizes diferentes, e como os Objetos de Conteúdo e o LMS podem atuar mutuamente para oferecer os elementos de interface (botões, links) adequados a uma rica experiência de aprendizagem.

O LMS Moodle foi identificado como uma das poucas opções de código aberto a implementar a maior parte da especificação SCORM. Sua implementação trata um pacote SCORM como uma instância de um módulo de atividades, em função de sua arquitetura modular. Embora, para o usuário final, a implementação pareça ser impecável, do ponto de vista do código-fonte ela apresenta problemas sérios, os quais dificultam não apenas a compreensão de seu funcionamento, mas também a reestruturação de partes desse código e testes de software.

Com os problemas observados na implementação do Moodle para o padrão SCORM, passa a ser relevante a idéia de se implementar a especificação sob uma nova abordagem. Para tor-

nar o trabalho, no entanto, menos custoso do ponto de vista do tempo de desenvolvimento são analisados três *frameworks* de desenvolvimento de aplicações web projetados sob o padrão arquitetural MVC (*Model-View-Controller*). Após a comparação entre CakePHP, Django e Ruby on Rails, o último foi escolhido como a melhor escolha para o trabalho.

6.1 Contribuições e Trabalhos futuros

Este trabalho, em virtude de se tratar de uma pesquisa ampla sobre diferentes áreas da engenharia de software, gerou contribuições diversas, a saber:

- Apresentou diferentes perspectivas teóricas sobre a área de objetos de aprendizagem
- Abordou a especificação do padrão SCORM 2004 3ª Edição de maneira abrangente, a qual ainda carece de referências na língua portuguesa
- Definiu de forma objetiva a arquitetura modular do LMS Moodle, sua implementação para o padrão SCORM na forma de um módulo de atividades e os problemas encontrados no código-fonte dessa implementação
- Documentou o modelo lógico do banco de dados relacional referente à implementação do Moodle para o padrão SCORM e também uma parcela do código-fonte desse módulo
- Apresentou três frameworks web (CakePHP, Django e Ruby on Rails) construídos sobre a arquitetura MVC, e ofereceu uma comparação entre os três projetos, levando em consideração o ganho de produtividade que conferem ao programador e a qualidade que podem oferecer ao software durante seu ciclo de vida
- Apontou o framework MVC Ruby on Rails como a melhor solução entre as 3 estudadas para a construção de um LMS que suporte objetos de aprendizagem criados de acordo com o padrão SCORM.

Este trabalho pode ser utilizado como ponto de partida para trabalhos da área de Informática na Educação e também de Engenharia de Software. O autor sugere os seguintes temas para pesquisa:

- Estudo dos custos envolvidos na construção de conteúdos sob o padrão SCORM em toda a sua potencialidade;

- Implementação mínima de um LMS capaz de suportar o padrão SCORM e avaliação das dificuldades envolvidas;
- Implementação de melhorias para o código-fonte do módulo SCORM do Moodle;
- Estudo das possibilidades pedagógicas provindas da mineração dos dados gerados na interação dos aprendizes com objetos de conteúdo adequados ao padrão.

APÊNDICE A – Documentação do módulo SCORM do Moodle

O fator que mais dificultou o entendimento da implementação do módulo SCORM foi a falta de documentação das funções. A grande maioria das funções não possuía descrição (fosse breve ou detalhada), nem informava quais eram os parâmetros de entrada e quais os possíveis valores de retorno. Muitas dessas funções, para dificultar ainda mais a compreensão, recebem tuplas (registros) como valores de entrada. Isso significa que um parâmetro se desdobra, na prática, em vários outros, um para cada propriedade do registro.

As próximas seções procuram documentar da forma mais clara possível as funções mais importantes do módulo, na medida em que foi possível para o autor compreendê-las.

A.1 scorm_add_instance()

Parâmetros de entrada:

- `$scorm`: um registro PHP contendo como propriedades todas as colunas da tabela `scorm`, entre outras propriedades. As propriedades inicializadas serão aquelas que o usuário definiu ao preencher o formulário. Entre elas pode-se destacar:
  - `reference`: o nome do pacote ou URL de referência ao pacote a ser processado.
  - `grademethod`: indica a forma como o módulo retornará ao Moodle a nota do aprendiz.
  - `maxgrade`: dependendo do valor da propriedade `grademethod` esta propriedade pode estar inicializada com um valor inteiro representando a nota máximo possível para um aprendiz dentro desse objeto de aprendizagem.
  - `maxattempt`: indica o número máximo de tentativas que o aprendiz poderá fazer com esse objeto de aprendizagem.

Possíveis valores retornados:

- `id`: O identificador do pacote recém adicionado à tabela `scorm`.

Esta função realiza formatações nos campos inicializados pelo usuário, além de descompactar o pacote selecionado por ele e verificar se o mesmo é válido. *Após* inserir o registro do novo pacote SCORM/AICC no banco de dados realiza o parsing do pacote: para um pacote SCORM cada um dos seus SCOs e Assets será registrado no banco de dados.

A.2 scorm_check_package()

Parâmetros de entrada:

- `$data`: uma variável que contém uma abstração do arquivo enviado pelo usuário durante a submissão do formulário de adição de pacote SCORM. Essa variável é um registro PHP com os seguintes atributos:
  - `course`: o identificador desse pacote dentro dos módulos de atividades do Moodle.
  - `reference`: O caminho na web para esse pacote. Será uma URL ou um nome de arquivo (sem o caminho precedendo-o).
  - `instance`: o identificador desse pacote dentro da tabela de pacotes SCORM/AICC (só estará inicializado no caso de atualização do pacote).

Possíveis valores de retorno (um entre esses):

- `null`: se a validação falhar.
- `registro`: um registro PHP (`$validation`) com os seguintes atributos inicializados:
  - `launch`: ?
  - `pkgtype`: o tipo de pacote sendo validado (possíveis valores são "SCORM" e "AICC").
  - `datadir`: o diretório onde o pacote descompactado (ou manifesto) foi deixado.
  - `result`: aplica-se apenas a pacote SCORM. Será `true` se o arquivo de manifesto existir, e `false` caso contrário.
  - `errors`: um hash. Quando `result` for `false`, então a chave `reference` deste hash terá uma mensagem associada, explicando o erro ocorrido.

Observações:

- Esta função gera efeitos colaterais.

Valida um pacote SCORM (arquivo .zip ou .pif). O algoritmo assume que esse arquivo pode significar a criação de uma nova instância do módulo ou uma atualização de uma instância já existente.

Verifica-se, primeiramente, onde, fisicamente, está o pacote escolhido. Ele pode estar em um repositório IMS ou no próprio diretório de arquivos do curso. Sabendo essa localização define-se o diretório-base onde o pacote será descompactado e testado.

Se a instância do módulo estiver sendo criada ocorre o seguinte processo.

- Se se trata de um arquivo de extensão .pif ou .zip então cria-se um diretório temporário, para onde esse arquivo é copiado. Lá dentro o pacote é descompactado e em seguida valida-se o arquivo manifesto (em caso de pacote SCORM) utilizando a função `scorm_validate_manifest()`. Note que o novo diretório criado permanece: apenas o pacote é removido. Neste momento a propriedade `pkgtype` do registro `$validation` é definida como "SCORM" ou "AICC".

- Senão, se se trata de um arquivo XML de nome imsmanifest.xml então a propriedade `pkgtype` de `$validation` é definida como "SCORM". Se esse for um arquivo externo, é criado um diretório temporário, e então esse arquivo é posicionado lá para ser validado. Caso contrário (o arquivo não é externo) simplesmente valida-se o arquivo referenciado-o de forma direta no sistema de arquivos.

- Para qualquer um dos casos acima, se a validação falhar o diretório temporário eventualmente criado é eliminado. Caso contrário

  - Se `$data` se trata de um arquivo manifesto não-externo, a propriedade `$datadir` de `$validation` passa a ser igual ao nome do diretório (não inclui caminho completo) onde esse arquivo está posicionado.
  - Senão essa mesma propriedade passa a ser igual ao nome do diretório temporário criado após a descompactação do pacote (não inclui caminho completo).

Se a instância do módulo estiver sendo atualizada ocorre processo semelhante ao que ocorre durante a criação. A diferença básica é que um arquivo de extensão .xml não será aceito.

A.3 scorm_delete_files()

Parâmetros de entrada:

- `$directory`: uma string representando o caminho completo do diretório a ser removido.

Possíveis valores de retorno (um entre esses):

- `boolean`: `true`, caso o diretório tenha sido recursivamente removido, ou `false`, caso contrário.

Observação: esta função é recursiva.

Esta função remove recursivamente o diretório `$directory`.

A.4 scorm_external_link()

Parâmetros de entrada:

- `$link`: uma URL associada a um pacote SCORM/AICC sendo processado pelo módulo.

Possíveis valores de retorno (um entre esses):

- `$result`: `true` se o endereço se parece com uma URL externa e `false` caso contrário. Uma URL é considerada externa se é iniciada por "http://", "https://" ou "www.".

Verifica se um link é externo ou não.

A.5 scorm_get_manifest()

Este método é um dos mais longos da biblioteca, com um total de 436 linhas. As variáveis de entrada da função são as seguintes:

- `$blocks` armazena uma representação em array do manifesto do pacote.
- `$scoes` será inicializada durante a execução da função (o autor não descobriu com que valor) e será utilizada no contexto que invocou esta função. Ela é passada como parâmetro

(em vez de ser apenas retornada ao fim da execução da função) pois esta se trata de uma *função recursiva*. A cada nova invocação a variável `$scoes` é passada para frente, para receber mais dados.

Esta função é totalmente baseada no padrão SCORM publicado pela ADL (ou tenta ser). Para cada nodo imediatamente abaixo da raiz da árvore será verificado se esse é (um de dois):

- `metadata`
- `organizations`

O algoritmo também procura por elementos `manifest`. O uso desses é, até a 3ª edição do SCORM 2004, desencorajado.

O conhecedor do padrão SCORM 2004 sentirá falta do elemento *resources* sendo analisado junto aos outros dois mencionados acima. O autor deste documento não conseguiu entender porquê isso ocorre, mas os recursos são analisados em outra função, `scorm_get_resources()`.

Adicionalmente, para cada um desses dois elementos da árvore o algoritmo desce a partir desses elementos procurando, entre seus filhos, por elementos obrigatórios, e ainda validando se esses filhos obrigatórios estão no formato e/ou quantidade corretos.

A.6 scorm_get_toc

Esta função constrói uma TOC (Tabela de Conteúdos) para o pacote SCORM em questão.

A.7 scorm_option2text()

Parâmetros de entrada:

- `$scorm`: um registro PHP com diversas propriedades anexadas.

Possíveis valores de retorno:

- O mesmo registro `$scorm` que chegou, porém com a propriedade `options` inicializada.

Se `$scorm` representar um objeto de aprendizagem que deve ser exibido em uma janela popup (sua propriedade popup será igual a 1), todas as propriedades relacionadas a popup serão

colocadas na propriedade `options`, na forma
"propriedade1=valor1,propriedade2=valor2,propriedadeN=valorN".

As propriedades relacionadas a popup definidas pelo módulo são as seguintes:

- `resizable`
- `scrollbars`
- `directories`
- `location`
- `menubar`
- `toolbar`
- `status`

A.8 scorm_optionals_data()

Essa função recebe dois parâmetros de entrada:

- `$item`: o item (SCO ou Asset) a ser analisado, na forma do elemento `resource` do manifesto convertido para um array associativo (hash) PHP.
- `$standarddata`: um array de strings, onde cada registro representa os dados considerados "padrão" para o respectivo `$item`.

Sua saída é um array de elementos opcionais para o `$item` em questão. O autor deste trabalho não compreendeu o o formato interno desse array.

A.9 scorm_parse()

Parâmetros de entrada:

- `$scorm`: um registro PHP contendo diversas propriedades (possivelmente todos os campos da tabela scorm).

Possíveis valores de retorno:

- `launch`: ?

Esta função verifica com que tipo de pacote está lidando: SCORM ou AICC. Se for um pacote AICC retorna o resultado da função `scorm_parse_aicc()`. Se for um pacote SCORM retorna o resultado da função `scorm_parse_scorm()`.

A.10 scorm_parse_scorm()

Parâmetros de entrada:

- `$pkgdir`: o caminho completo para o diretório onde o pacote está localizado.
- `$scormid`: o identificador do pacote na tabela `scorm`.

Possíveis valores de retorno:

- `launch`: ?

Após verificar se o arquivo de manifesto do pacote em questão existe os passos são os seguintes.

- Lê o conteúdo do arquivo manifesto e o armazena como uma grande string.
- Troca alguns caracteres dessa string por espaços em branco (o autor deste trabalho não descobriu porque).
- Utiliza a classe `xml2Array` para gerar um objeto capaz de realizar o parsing da string. Esse processo gera um grande array, representando toda a árvore XML do manifesto (essa árvore é armazenada em `$manifests`).
- A função `scorm_get_scoes()` é invocada, e a ela são passadas `$manifests` (árvore do manifesto) e uma variável vazia, `$scoes`, que deve ser inicializada dentro dessa função. Essa variável inicializa, curiosamente, não apenas SCOs, mas também Assets.
- Para cada um dos `$scoes` tenta-se salvar os seguintes dados:
  - `scorm`: o identificador que o Moodle vinculou ao pacote que inclui esse SCO
  - `manifest`: o nome do manifesto para esse SCO (em caso de o pacote possuir mais de 1 manifesto)

  - `organization`: ?
  - `parent`: ?
  - `identifier`: ?
  - `launch`: ?
  - `scormtype`: ?
  - `title`: ?

- Dados referentes a SCOs são atualizados (pois provavelmente esta função é utilizada tanto para atualizar um pacote quanto para apresentá-lo pela primeira vez ao Moodle).
- Se há dados opcionais associados a um SCO então eles são coletados e armazenados na tabela `scorm_scoes_data`. Esses dados são coletados a partir da função `scorm_optionals_data()`.
- Se o SCO possui regras de seqüenciamento definidas então elas são tratadas e armazenadas em duas tabelas: `scorm_seq_ruleconds` e `scorm_seq_rulecond`.
- Há um tratamento para rolluproles.
- Há um tratamento para objetivos.

A.11 scorm_tempdir()

Parâmetros de entrada:

- `$strPath`: o diretório dentro do qual o diretório temporário deve ser criado.

Possíveis valores de retorno (um deles):

- `false`: caso `$strPath` não seja um diretório.
- `string`: uma string representando o caminho para o novo diretório criado.

Cria um diretório temporário dentro do diretório passado como parâmetro (`$strPath`).

A.12 scorm_scandir()

Parâmetros de entrada:

- `$directory`: uma string representando o caminho completo do diretório a ser removido.

Possíveis valores de retorno (um entre esses):

- `array`: um array contendo todos os arquivos e diretórios presentes em `$directory`. Cada índice armazena apenas o nome do arquivo ou diretório, e não seu caminho completo.

Retorna um array contendo todos os arquivos e diretórios presentes em `$directory`.

A.13 scorm_view_display()

A variável "versão", adquirida do SCO, dirá qual modelo de dados está sendo utilizado (1.2, 1.3 etc.). De posse dessa informação o modelo correto será carregado (será um dos modelos disponíveis no diretório `datamodels`). Esse modelo incluirá a função `scorm_get_toc()`, a qual será invocada para gerar a tabela de conteúdos.

Após imprimir a TOC (Tabela de Conteúdos) verifica-se se o aprendiz já concluiu os objetivos do pacote SCORM em questão e se ele pode realizar novas tentativas. Se for permitido, então um link apropriado lhe é apresentado ("Tentar novamente").

A.14 scorm_validate()

Parâmetros de entrada:

- `$data`: um registro PHP contendo como propriedades os campos presentes no formulário de adição/atualização de uma instância do módulo.

Possíveis valores de retorno:

- `$validation`: um registro PHP contendo dois valores:
  - `result`: será `true` se os dados submetidos são todos válidos, e `false` caso contrário.

  - `errors`: se result for `false`, então errors será um hash de mensagens de erro, onde as chaves correspondem a campos do formulário, e os valores são as mensagens que devem ser apresentadas ao usuário.

Esta função valida os diversos campos do formulário de criação de uma nova instância. Não valida, no entanto, se um pacote SCORM possui um manifesto de formato válido, apenas se esse manifesto existe.

A.15 scorm_validate_manifest()

Parâmetros de entrada:

- `$manifest`: o caminho para um arquivo de manifesto.

Possíveis valores de retorno:

- `$validation`: um registro PHP com os seguintes atributos inicializados:
  - `result`: será `true` se o arquivo `$manifest` existir, e `false` caso contrário.
  - `errors`: um hash. quando `result` for `false`, então a chave `reference` deste hash terá uma mensagem associada.

Retorna um registro PHP informando se o parâmetro de entrada `$manifest` é um arquivo válido.

Referências Bibliográficas

ABED. *Anuário Brasileiro Estatístico de Educação Aberta e a Distância*. [S.l.: s.n.], 2007.

ADL. *Sharable Content Object Reference Model (SCORM) 2004 3rd Edition Content Aggregation Model*. [s.n.], 2006. Disponível em: <http://www.adl.net>.

ADL. *Sharable Content Object Reference Model (SCORM) 2004 3rd Edition Overview*. [s.n.], 2006. Disponível em: <http://www.adlnet.org>.

ADL. *Sharable Content Object Reference Model (SCORM) 2004 3rd Edition Run-Time Environment*. [s.n.], 2006. Disponível em: <http://www.adl.net>.

ADL. *Sharable Content Object Reference Model (SCORM) 2004 3rd Edition Sequencing and Navigation*. [s.n.], 2006. Disponível em: <http://www.adl.net>.

ADL (Advanced Distributed Learning). 2007. Disponível em: <http://www.adlnet.org>. Acesso em: 22 de outubro de 2007.

CAKEPHP. 2007. Disponível em: <http://cakephp.org>. Acesso em: 22 de outubro de 2007.

CLAROLINE. 2007. Disponível em: <http://www.claroline.net>. Acesso em: 22 de outubro de 2007.

CLAROLINEDOC - How do I create SCORM content? 2007. Disponível em: <http://www-.claroline.net/doc/en/index.php/How_do_I_create_SCORM_content%3F>. Acesso em: 22 de outubro de 2007.

DJANGO. 2007. Disponível em: <http://www.djangoproject.com>. Acesso em: 22 de outubro de 2007.

DOWNES, S. Learning objects: Resources for distance education worldwide. 2001. Disponível em: <http://www.irrodl.org/index.php/irrodl/article/view/32/378>. Acesso em: 22 de outubro de 2007.

DUVAL, E. Learning technology standardization: Making sense of it all. 2004.

FOOTE, B.; JOHNSON, R. E. Designing reuseable classes. 1991.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: Addison-Wesley Professional, 2003.

FOWLER, M. *GUI Architectures*. 2006. Disponível em: <http://www.martinfowler.com-/eaaDev/uiArchs.html>. Acesso em: 22 de outubro de 2007.

HOLZINGER, A.; SMOLLE, J.; REIBNEGGER, G. Learning objects (lo): an object-oriented approach to manage e-learning content. 2005. Disponível em: <http://user.meduni-graz.at-/andreas.holzinger/holzinger/holzinger-publications.html>. Acesso em: 22 de outubro de 2007.

IEEE. *Learning Object Metadata (LOM)*. 2007. Disponível em: <http://ltsc.ieee.org/wg12/>. Acesso em: 22 de outubro de 2007.

JACOBSEN, P. Reusable learning objects: What does the future hold? 2001. Disponível em: <http://www.ltimagazine.com/ltimagazine/article/articleDetail.jsp?id=5043>. Acesso em: 22 de outubro de 2007.

MOODLE. 2007. Disponível em: <http://moodle.org>. Acesso em: 22 de outubro de 2007.

MOODLE-TRACKER. 2007. Disponível em: <http://tracker.moodle.org>. Acesso em: 22 de outubro de 2007.

PHP. 2007. Disponível em: <http://www.php.net>. Acesso em: 22 de outubro de 2007.

PYTHON. 2007. Disponível em: <http://www.python.org>. Acesso em: 22 de outubro de 2007.

REIGELUTH, C. M.; NELSON, L. M. A new paradigm of isd? 1997.

ROSSI, L. C. Standards: Do we really need them? 2003. Disponível em: <http://www-.masternewmedia.org/2003/12/26/standards_do_we_really_need.htm>. Acesso em: 22 de outubro de 2007.

RSPEC. 2007. Disponível em: <http://rspec.rubyforge.org>. Acesso em: 22 de outubro de 2007.

RUBY on Rails. 2007. Disponível em: <http://www.rubyonrails.com>. Acesso em: 22 de outubro de 2007.

RUBY Programming Language. 2007. Disponível em: <http://www.ruby-lang.org>. Acesso em: 22 de outubro de 2007.

SAKAI. 2007. Disponível em: <http://www.sakaiproject.com>. Acesso em: 22 de outubro de 2007.

TELEDUC. 2007. Disponível em: <http://teleduc.nied.unicamp.br/pagina/>. Acesso em: 22 de outubro de 2007.

VMC-GRAZ. Virtual Medical Campus. 2007. Disponível em: <http://vmc.meduni-graz.at/>. Acesso em: 22 de outubro de 2007.

WIKIPEDIA. *Artigo "CakePHP" na Wikipedia de língua inglesa*. 2007. Disponível em: <http://en.wikipedia.org/wiki/CakePHP>. Acesso em: 22 de outubro de 2007.

WIKIPEDIA. *Artigo "DRY" na Wikipedia de língua inglesa*. 2007. Disponível em: <http://en.wikipedia.org/wiki/DRY>. Acesso em: 22 de outubro de 2007.

WILEY, D. Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. 2001. Disponível em: <http://www.reusability.org/read>. Acesso em: 22 de outubro de 2007.