UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

CELINE LIP: UM FRAMEWORK QUE UTILIZA O MODELO IMS LIP EM APLICAÇÕES WEB JEE

MARCELO GONZAGA

BLUMENAU
2009

2009/2-14

MARCELO GONZAGA

CELINE LIP: UM FRAMEWORK QUE UTILIZA O MODELO IMS LIP EM APLICAÇÕES WEB JEE

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Adilson Vahldick , Msc – Orientador

BLUMENAU
2009

2009/2-14

CELINE LIP: UM FRAMEWORK QUE UTILIZA O MODELO IMS LIP EM APLICAÇÕES WEB JEE

Por

MARCELO GONZAGA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Adilson Vahldick Orientador, M.Sc. – Orientador, FURB
Membro:	Prof. Alexander Roberto Valdameri, M.Eng. – FURB
Membro:	Prof. Dalton Solano dos Reis, M.Sc . – FURB

Blumenau, 9 de dezembro de 2009

Dedico este trabalho aos meus pais, a minha namorada, meu orientador e todos os amigos, pela dedicação e compreensão para que meus objetivos fossem alcançados.

.AGRADECIMENTOS

A Deus, pela graça da vida.

Aos meus familiares, que sempre estiveram presentes.

A minha namorada, Gesilene Cardoso, pelo apoio e compreensão para elaboração deste trabalho.

Aos meus amigos Greg Mercine Camilo, Jonathan Kraemer e Marcos Paulo Peron, que me auxiliaram nos momentos de dúvidas.

A empresa Uninformare Informática LTDA, que me deu todo o suporte necessário para a conclusão do trabalho.

Ao meu orientador, Adilson Vahldick, por ter acreditado na conclusão deste trabalho.

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

Este trabalho tem por objetivo apresentar o desenvolvimento de um *framework* que facilite a criação de aplicações WEB JEE para gerenciamento de ensino quanto à administração de usuários de cursos de EaD, segundo os padrões de estrutura de dados propostos pelo modelo IMS LIP. Através de páginas JSP, o desenvolvedor tem a possibilidade de incorporar a sua aplicação funcionalidades como listagens, consultas e manipulação de dados dos usuários de uma maneira transparente, apenas invocando em meio ao seu código fonte as *tags* definidas na documentação do *framework*. A implementação do *framework* foi realizada com tecnologias Java como JSP e Servlets, e complementada com a utilização da biblioteca JSTL para facilitar a montagem das páginas. Foi imprescindível a utilização do XStream para realizar a leitura e gravação de arquivos XML que atendessem a especificação IMS LIP. Esse *framework* é acoplável ao componente CELINE, que tem o propósito de fornecer recursos para o desenvolvimento de aplicações de gerencia de ensino utilizando o SCORM.

Palavras-chave: Ambiente web. Java. XML.

ABSTRACT

This paper aims to present the development of a framework that facilitates the creation of JEE web applications for education management regarding the administration of users of DL (Distance Learning) courses at the standards of data structure proposed by the model IMS LIP. Through JSP pages, the developer has the possibility of incorporating the application features such as lists, queries and manipulation of user data in a transparent manner, relying only in the midst of source code tags defined in the framework. The implementation of the framework was done with Java technologies like JSP and Servlets, and complemented with the use of the JSTL library to facilitate the assembly of the pages. It was essential to use XStream for reading and writing XML files that conform to IMS LIP specification. This framework is attachable to the component CELINE, which aims to provide resources for the development of manages teaching applications using SCORM.

Keywords: Environment web. Java. XML.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AJAX - *Asynchronous Javascript And XML*

API - *Application Programming Interface*

BDR - Banco de Dados Relacional

BNF - *Backus-Naur form*

DAO - *Data Access Object*

DOM - *Document Object Model*

EaD - Ensino à Distância

EJB - *Enterprise Java Beans*

GLC - *Global Learner Consortium*

HTML - HyperText Markup Language

IMS - *Instructional Management Systems*

IMS LIP - *Instructional Management Systems Learner Information Package*

JAR - *Java ARchive*

JDBC - *Java DataBase Connectivity*

JEE - *Java Enterprise Edition*

JSP - *JavaServer Pages*

LMS - *Learning Management System*

MER - *Modelo Entidade e Relacionamento*

MVC - *Model View Controller*

PAPI - *Personal And Private Information standard*

QCL - Qualificações, Certificações e Licenças

RF - Requisito Funcional

RNF - Requisito Não Funcional

SCORM - *Sharable Content Object Reference Model*

SQL - *Structured Query Language*

TLD - *Tag Library Descriptor*

UML - *Unified Markupt Language*

URL - *Uniform Resource Locator*

WTP - *Web Tools Plataform*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Nos tempos atuais, a Educação a Distância (EaD) é uma realidade. Cada vez mais esta modalidade educacional vem sendo utilizada nas escolas, empresas, iniciativas públicas e privadas e universidades. Um dos fatores chave para a sua expansão é o fato de estar associada às crescentes necessidades educacionais, mais especificamente à ausência de professor e aluno em uma sala de aula física, que não podem ser satisfeitas pelos sistemas tradicionais de ensino e, também, pelo desenvolvimento de tecnologias da informação e comunicação cada vez mais poderosas e presentes em nosso cotidiano, como a expansão dos recursos de multimídia e a popularização da internet. De acordo com Niskier (1999), a EaD pode ser considerada a tecnologia da esperança.

Diante da dimensão e complexidade apresentada pela EaD, facilmente torna-se possível subdividí-la e trabalhá-la em menores partes como: o gerenciamento dos cursos oferecidos pelo ambiente de EaD, o gerenciamento dos conteúdos dos cursos, o gerenciamento dos alunos e a interação dos alunos com os cursos.

No que se refere ao gerenciamento dos alunos, cada sistema poderia definir uma forma específica para controle dos mesmos, pelo fato de não existir uma padronização para esta situação. Ferreira (1995, p. 318) define o conceito de padronização como: “Uniformização do comportamento dos indivíduos segundo modelos aceitos por um grupo ou impostos pela criação de novos hábitos [...]”.

Dentre os modelos de gerenciamento de usuários em cursos existentes destaca-se o *Instructional Management Systems Learning Information Package* (IMS LIP) por ser bastante robusto e difundido. Este modelo tem por essência o gerenciamento de características do aluno como, por exemplo: histórico, objetivos de aprendizagem e aspectos de acessibilidade. Também possui uma característica muito forte de facilidade de integração com outros sistemas.

Este modelo foi desenvolvido pela *Instructional Management Systems* (IMS), com a finalidade de atender a uma grande diversidade de requisitos e também tornar simples o mapeamento entre a própria empresa e as demais especificações de componentes de EaD existentes. O IMS LIP prevê a divisão dos dados armazenados em onze categorias, sendo elas: identificação; objetivo; Qualificações, Certificações e Licenças (QCL); atividade, transcrito, interesse, competência, filiação, acessibilidade, chave de segurança e relacionamento.

Existem especificações que modelam a gerência de outro tipo de informação dentro do

ambiente de EaD, que não necessariamente estejam ligadas a dados do aprendiz, como por exemplo a especificação *Sharable Content Object Reference Model* (SCORM)[1], que diz respeito a parte do ambiente que realiza a gerência de conteúdos dos cursos. Existem componentes que realizam esta função dentro do ambiente de EaD como por exemplo o CELINE.

O CELINE é um componente que provê recursos para o desenvolvimento de ambientes de aprendizagem, atendo-se ao gerenciamento de conteúdo baseado no formato SCORM.

Visto o acima, este trabalho propõe desenvolver um *framework*[2] estendendo as funcionalidades do CELINE, para realizar a especificação IMS LIP. Esta ferramenta proverá recursos para gerenciamentos de usuários seguindo o modelo acima citado. Tais recursos referem-se à *tags* customizadas e formas de persistência e acesso aos dados.

A implementação do CELINE LIP ocorreu através da especificação *Java Enterprise Edition* (JEE), que é uma especificação para o desenvolvimento de aplicações de caráter distribuído, o qual garante um ambiente de execução padronizado e seguro para as aplicações. Os ambientes que atendem esta especificação são conhecidos como servidores de aplicação.

A especificação JEE prevê a definição de como uma aplicação deve ser estruturada, implantada e definida para que tenha recursos de autenticação e autorização de usuários, como utilizar um ambiente transacional, distribuir seus objetos e assim por diante (BOND *et al*. 2003, p. 13).

Por fim, como recurso facilitador na operação do *framework* utilizou-se do conceito de *tags* customizadas. Tratam-se de recursos acoplados a linguagem *Java Server Pages* (JSP) que tem por princípio auxiliar os desenvolvedores na eliminação da utilização de *scriptlets* e redução de redundância de código. A *tag* customizada é criada para a geração de um conteúdo dinâmico e utilizada em quantas páginas forem necessárias.

---

[1] SCORM é uma especificação técnica que se propõe a definir como um *e-learning* é criado e apresentado aos alunos.

[2] Appleton (1997, p. 15, tradução nossa) diz que "um *framework* de software é uma mini-arquitetura reutilizável que fornece a estrutura e comportamentos genéricos para uma família de abstrações de software, comum contexto que especifica suas interações e uso dentro de um determinado domínio".

1.1 OBJETIVOS DO TRABALHO

Este trabalho tem por objetivo disponibilizar um *framework* que juntamente ao componente CELINE disponibilize recursos de controle de usuários que atendam a especificação IMS LIP.

Os objetivos específicos do trabalho são:

a) fornecer uma *Application Programming Interface* (API) para gerenciamento dos dados no modelo IMS LIP em documentos, baseada em *eXtensible Markup Language* (XML) e em Banco de Dados Relacionais (BDR), a princípio compatível com MYSQL;
b) permitir a criação de acesso personalizado que permita o gerenciamento dos dados;
c) disponibilizar um conjunto de *tags* personalizadas para a montagem das páginas JSP (*Java Server Pages*) que permita o acesso aos recursos da API;
d) oferecer uma API para pesquisa e agrupamento dos dados de acordo com as categorias do modelo IMS LIP;
e) organizar as APIs em um *framework*;
f) integrar o *framework* no componente CELINE.

1.2 ESTRUTURA DO TRABALHO

O trabalho apresentado está divido em quatro capítulos, conforme descrito a seguir. O primeiro capítulo consiste na apresentação da contextualização, a apresentação dos objetivos e justificativas para o desenvolvimento do trabalho. O segundo capítulo apresenta a fundamentação das teorias relacionadas ao trabalho, apresentação do padrão IMS LIP, do componente CELINE, da plataforma JEE e a apresentação dos trabalhos correlatos.

O terceiro capítulo apresenta os requisitos da ferramenta, assim como sua especificação, implementação e resultados. São utilizados os diagramas de caso de uso, diagrama de atividades, diagrame de classe e o de modelo de entidade e relacionamento. A operacionalidade da ferramenta também é apresentada durante este capítulo.

O quarto capítulo apresenta as conclusões finais e sugestões para elaboração de futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os aspectos teóricos referentes à especificação IMS LIP. Na seção 2.1 IMS LIP serão definidas as características técnicas e detalhes de implementação das especificações. Na seção 2.2 é apresentado o componente CELINE. Na seção 2.3 são descritas as partes da JEE relevantes a esta proposta, especialmente as *tags* customizadas. Na seção 2.4 são relacionados três trabalhos correlatos.

2.1 IMS LIP

O *Learner Information Package* (LIP) é uma especificação que foi definida por um consórcio de instituições que atuam nos ramos de educação, software e editoras, e juntos formam o *Global Learner Consortium* (GLC) IMS. Este consórcio objetiva "promover a disseminação de especificações que permitam ambientes de aprendizagem distribuídos" (IMS GLOBAL LEARNING CONSORTIUM, 2001). O IMS LIP foi especificado para atender a necessidade de colecionar informações sobre aprendizes (individuais ou grupo de aprendizes) ou de produtores de conteúdos de aprendizagem (criadores, provedores ou vendedores).

A especificação proporciona a criação de um modelo de dados que descreva a característica de um usuário e atenda a propósitos que estejam relacionados ao seu aprendizado. Com estas informações ligadas ao aprendiz, alguns objetivos específicos podem ser apontados como: traçar um perfil de aprendizagem baseando-se em aspectos como o seu histórico, objetivos e realizações, envolver o aluno em uma experiência de aprendizagem e identificar aspectos e situações que possam auxiliar o aprendiz durante o processo de aprendizagem.

> A especificação suporta o intercâmbio de informações entre aluno aprendendo sistemas de gestão, sistemas de recursos humanos, sistemas de informação do estudante, sistemas de e-learning, sistemas de gestão do conhecimento, repositórios de currículos, e outros sistemas utilizados no processo de aprendizagem. (IMS GLOBAL LEARNING CONSORTIUM, 2001, tradução nossa).

Segundo Musa e Oliveira (2007) "O LIP foi desenvolvido visando à interoperabilidade das informações de aluno, de modo que as mesmas possam ser trocadas facilmente entre sistemas que também adotaram LIP".

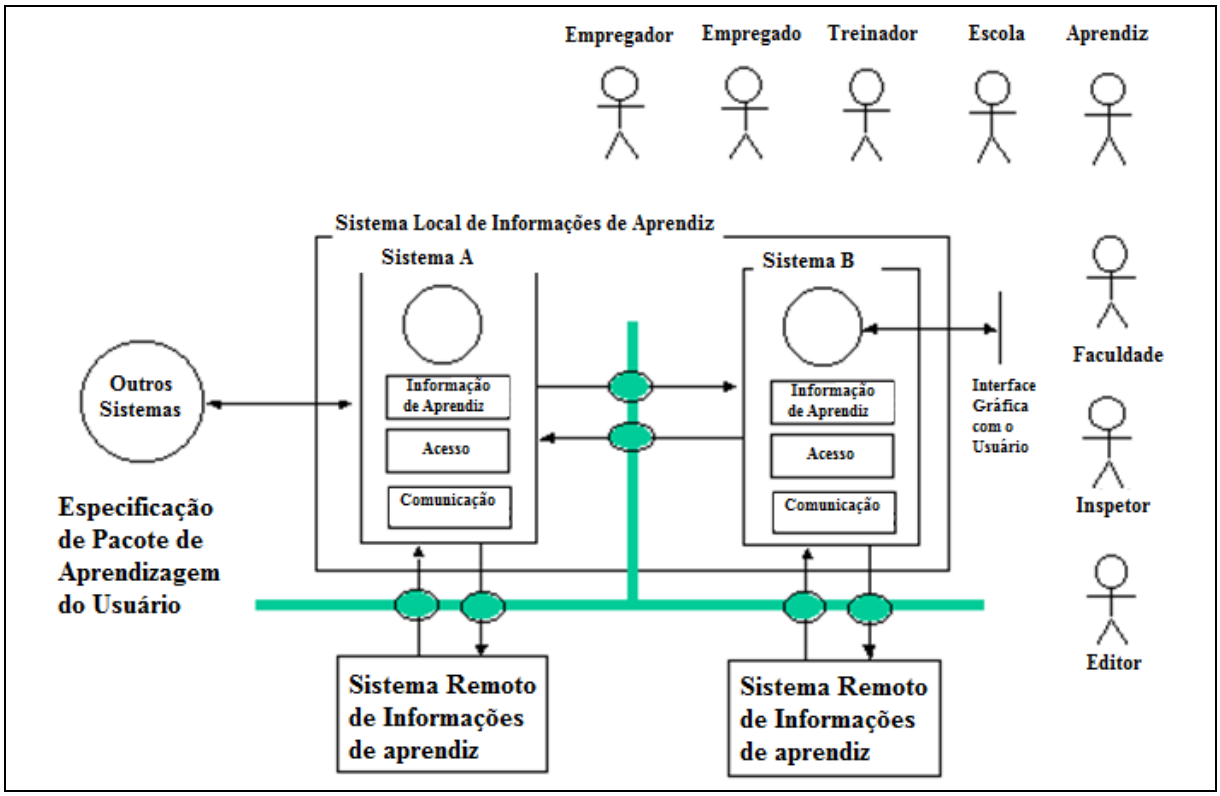
A intenção da especificação é definir um conjunto de pacotes que possam ser utilizados para manter, importar e extrair dados entre servidores compatíveis com IMS *Learner Information*.

Um dos focos da especificação do LIP foi baseado em definir requisitos facilitadores no processo de construção de sistemas de gerenciamento de aprendizes em cursos, dentre os quais, a quatro requisitos básicos:

a) distribuição de informação: "Um sistema de aprendizado de fato consiste de múltiplos sistemas distribuídos que compartilham ou armazenam informações através de um aprendiz modelo. Estas necessidades incluem a de adequar o acesso e o tempo de estampagem do sistema aprendiz e como este está dividido" (IMS GLOBAL LEARNING CONSORTIUM, 2001, tradução nossa);
b) escalabilidade: preocupar-se em atender softwares de grande escala e efetuar as trocas de informação de maneira sólida e consistente;
c) privacidade e proteção de dados: tornar o sistema capaz de implementar políticas de proteção dos dados e privacidade e garantir a integridade dos mesmos;
d) flexibilidade: "O sistema deve incluir muitas construções, tais como para objetivos e históricos de aprendizagem, que na prática deverão ser atendidas em diferentes estruturas e em diferentes contextos. Modelos de dados de informações de aluno devem ser suficientemente flexíveis para acomodar essas necessidades" (IMS GLOBAL LEARNING CONSORTIUM, 2001, tradução nossa).

Em relação à privacidade e proteção de dados é importante ressaltar as seguintes necessidades: manter a privacidade das informações dos alunos, proteger as informações de acesso inadequado, assegurar a integridade da informação e acomodar as exigências das políticas reguladoras e de diferentes jurisdições.

Na figura 1 é representada a integração de sistemas que utilizam e compartilham suas informações no formato LIP, onde é visualizada a validação dos quatro requisitos básicos.



Fonte: IMS Global Learning Consortium (2001).

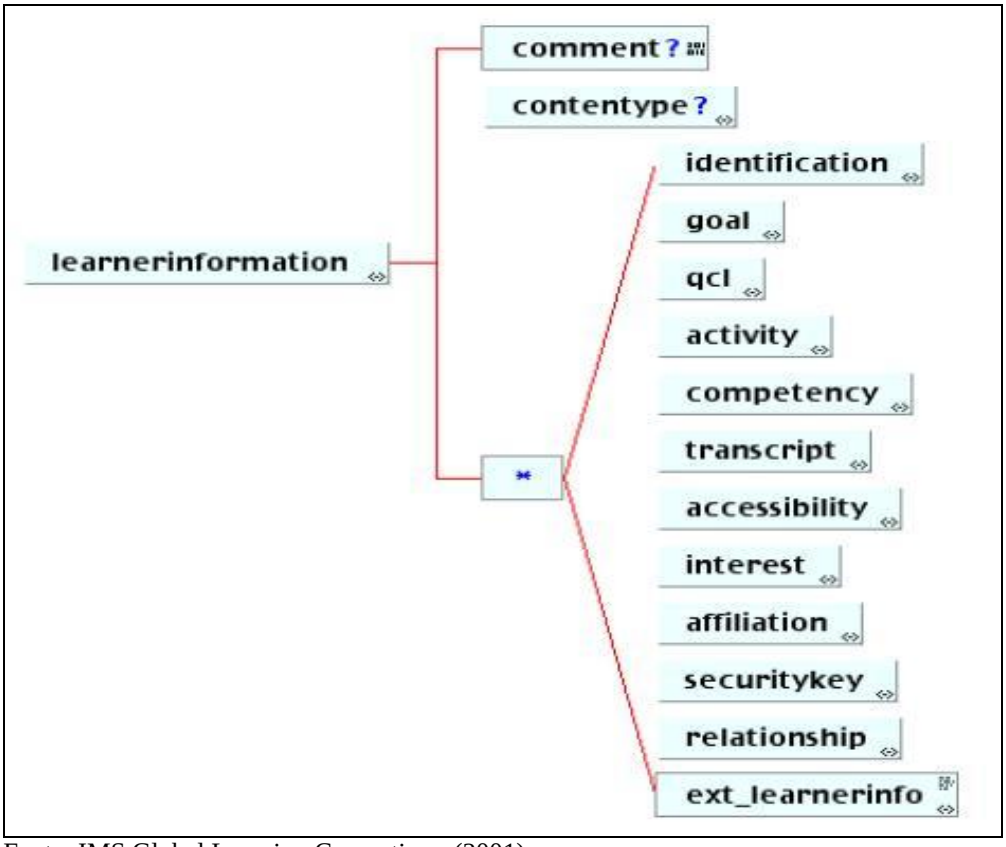
Figura 1 – Representação da comunicação de sistemas baseado em IMS LIP

No caso apresentado na Figura 1 é possível visualizar que, Sistema A e Sistema B, realizam a persistência dos dados, o controle de privacidade aos mesmos e a interação dos dados com outros sistemas. Observa-se também que várias situações de perfis de usuários são gerenciadas pelo mesmo sistema, atendendo desta maneira ao requisito de flexibilidade. A aplicação como um todo apresenta a validação do requisito de escalabilidade, pois o LIP é utilizado pelos pequenos sistemas em separado, mas também é utilizado de maneira distribuída, onde esta utilização pode ser compreendida como um sistema global de grande porte.

O modelo de armazenamento proposto pela especificação prevê manter dados e metadados sobre as informações do aluno, onde são definidos os campos dos dados e qual o gênero de informação que este mesmo campo de dados irá armazenar. “Metadados podem incluir informações de tempo; identificação e indexação; privacidade e proteção dos dados. Estes metadados estão disponíveis para cada campo do modelo de informação, diretamente ou via herança” (ASSIS, 2005, p. 36).

A estrutura de dados proposta pelo modelo é uma estrutura hierárquica que pode ser expressa em forma de árvore, em objetos de modelo ou como uma representação tabular. Tal modelo foi especificado, de maneira a atender onze categorias de informações pertinentes ao aprendiz, mais uma categoria para informações de extensão. “Estas estruturas foram

identificadas como as principais estruturas de dados que são necessários para apoiar a identificação do aluno. Esta abordagem composta significa que apenas precisam de informações necessárias para ser embalado e armazenado" (IMS GLOBAL LEARNING CONSORTIUM, 2001, tradução nossa). Na figura 2 é apresentada uma modelagem das informações do usuário, contendo estruturas para armazenar, comentários, tipos de conteúdos, as suas 11 categorias e mais a categoria de extensão.



Fonte: IMS Global Learning Consortium, (2001).

Figura 2 – Modelagem das informações dos usuários

As onze categorias são:

a) *identification* (identificação): consiste em reunir informações de caráter pessoal e demográfico, como por exemplo: nome, local de nascimento, número de documento e contatos;

b) *goal* (objetivo): tem por objetivo armazenar as metas e objetivos, tanto de aprendizado, como pessoais e profissionais. Pode ser utilizado, para acompanhar a evolução e o progresso do mesmo, identificar os aspectos que o aprendiz apresenta dificuldades;

c) *qualifications, certifications and licenses* (qualificações, certificações e licenças): agrupa informações sobre as qualificações, certificações e licenças (licença para exercer medicina, advocacia, etc.) conquistadas pelo aluno (GHELMAN, 2006, p.

37) ;

d) *activity* (atividade): conjunto das atividades que possuam relação com o aprendizado que está em andamento, como por exemplo: educação informal e formal, treinamento, experiência profissional e serviço militar ou civil. “Qualquer atividade de aprendizagem, já concluída ou em andamento, está referenciada na categoria *Activity*”. (MUSA; OLIVEIRA, 2007);
e) *transcript* (transcrito): guarda as informações de desempenho do aluno dentro da instituição;
f) *interest* (interesse): informações relacionadas a *hobbies* e atividades extras realizadas pelo aluno;
g) *competency* (competência): armazena o conjunto de experiências, conhecimentos e habilidades adquiridas pelo estudante, podendo possuir relação com qualificações, certificações, licenças e atividades já realizadas;
h) *affiliation* (filiação): organizações e entidades as quais o aluno possua relação profissional;
i) *accessibility* (acessibilidade): indica se o aluno possui deficiências, conhecimentos em outros idiomas, preferências em relação a aprendizagem ou ao uso de ferramentas/tecnologias específicas;
j) *securitykey* (chave de segurança): armazena o conjunto de senhas e chaves de segurança do estudante para transações com sistemas de informação de estudantes e serviços (ASSIS, 2005, p. 22);
k) *relationship* (relacionamento): descreve os relacionamentos existentes entre dados das categorias apresentadas.

Conforme apresentado, a estrutura especificada para o modelo permite sua representação de várias maneiras. A especificação do modelo LIP permite utilizar uma série de outras tecnologias para armazenamento, pelo fato de ser de fácil adaptação como banco de dados e arquivo XML.

“Documentos XML formam uma estrutura de árvore onde o nodo principal é a raiz e os demais elementos são folhas” (W3SCHOOLS, 2009, tradução nossa). Pelo fato do IMS LIP permitir que seu modelo seja representado em forma de árvore, o mesmo associa a sua utilização, a de arquivos XML para armazenar o banco de informações referentes a um *LearnerInformation*. No Quadro 1 é apresentado um arquivo de XML que armazena duas categorias de informações do aprendiz.

```
 1.<?xml version="1.0" standalone="no"?>
 2.<learnerinformation>
 3.
 4.   <contentype>
 5.         <referential>
 6.               <indexid>Entry</indexid>
 7.         </referential>
 8.   </contentype>
 9.   <identification>
10.         <formname>
11.               <typename>
12.                     <tyvalue>Preferred</tyvalue>
13.               </typename>
14.               <text>Mr Sherlock Holmes</text>
15.         </formname>
16.   </identification>
17.   <identification>
18.         <formname>
19.               <typename>
20.                     <tyvalue>Usual</tyvalue>
21.               </typename>
22.               <text>Mr Sherlock Holmes</text>
23.         </formname>
24.   </identification>
25.      <securitykey>
26.         <typename>
27.               <tysource sourcetype="imsdefault"/>
28.               <tyvalue>Password</tyvalue>
29.         </typename>
30.         <contentype>
31.               <referential>
32.                     <indexid>securitykey_1</indexid>
33.               </referential>
34.         </contentype>
35.         <keyfields>
36.               <fieldlabel>
37.                     <typename>
38.                           <tyvalue>PersonalPassword</tyvalue>
39.                     </typename>
40.               </fieldlabel>
41.               <fielddata>asits9</fielddata>
42.         </keyfields>
43.         <keyfields>
44.               <fieldlabel>
45.                     <typename>
46.                           <tyvalue>LMSPassword</tyvalue>
47.                     </typename>
48.               </fieldlabel>
49.               <fielddata>moriarty</fielddata>
50.         </keyfields>
51.   </securitykey>
52.
53.</learnerinformation>
```

Quadro 1 – Exemplo de arquivo XML

O XML permite que o modelo seja facilmente representado, pois sua estrutra apresenta, um nodo principal definido como *learnerinformation*, que por sua vez é composto de sub-nodos, como sendo suas categorias, dentre as quais no Quadro 1 são apresentadas

identification na linha 9 e *securitykey* na linha 25, que possuem sub-nodos que contemplam a representação das informações armazenadas pelo IMS LIP.

2.2 CELINE

O componente CELINE trata-se de um componente Java para aplicações web que tem por objetivo permitir a execução de pacotes SCORM e interferir em meio a interação do estudante com estes pacotes.

"O componente é um módulo independente que provê informações sobre o que ele faz e como usá-lo através de suas interfaces públicas, enquanto encapsula as suas tarefas internas" (MCCLURE, 2001, p. 4; BARROCA et al. 2005, p. 4 apud VAHLDICK, 2008, p. 32). Através da utilização de componentes torna-se possível que outros sistemas possam vir a utilizar as funcionalidades disponibilizadas pelo componente, pois é de baixo nível a complexidade de utilização.

Baseando-se na especificação do SCORM, o CELINE realiza as seguintes tarefas:

a) a interpretação do arquivo de manifesto: trata-se de um XML que é interpretado e a partir disso cria-se a estrutura dos objetos e seus relacionamentos;
b) a realização do processo de seqüenciamento: processo onde atua-se sobre árvore de atividades, que é uma definição lógica do pacote de conteúdo do usuário;
c) o ambiente de execução: onde as páginas de conteúdo comunicam-se com o *Learning Management System* (LMS).

Outra característica importante que o componente apresenta é a utilização do recurso de *tags* personalizadas. "[...] novos sistemas podem tirar proveito do componente, pois é muito baixo o nível de complexidade na sua utilização. O desenvolvedor precisa configurar o componente e montar páginas com *tags* disponibilizadas para listar e manter cursos e alunos" (VAHLDICK, 2008, p. 47).

O componente disponibiliza um arquivo de configuração descrito no formato XML chamado de `celine-config.xml`. "Fazem parte das configurações do componente: a pasta onde são armazenados os arquivos de conteúdo SCORM; a página padrão quando um erro acontece no componente; a classe que implementará a adaptabilidade do conteúdo; e o mecanismo de persistência" (VAHLDICK, 2008, p. 49).

Durante a configuração do componente, define-se o mecanismo de persistência. Três

tipos de persistência são suportados: XML, banco de dados relacional e um personalizável pelo desenvolvedor. Para atender a necessidade do componente de permitir o acesso a vários mecanismos de persistência de dados, utilizou-se o padrão de implementação DAO, onde criou-se uma *interface* que define um comportamento padrão para acesso e manipulação aos objetos de dados gerenciados pelo CELINE. As classes que vierem a realizá-la deverão implementar métodos para inserir, alterar, excluir e recuperar as informações no tipo de persistência desejado.

2.3 JEE (*JAVA ENTERPRISE EDITION*) E TAGS CUSTOMIZADAS

O JEE é uma plataforma Java voltada para redes, internet, intranets e semelhantes e aplicações distribuídas, pois contém bibliotecas desenvolvidas para o acesso a servidores, a sistemas de *e-mail*, a banco de dados, entre outras características. Graças a estas características, o JEE foi desenvolvido para suportar uma carga gigantesca e de usuários simultâneos.

Em relação a esta plataforma, três tecnologias, em especial, merecem destaque:

a) Servlet são classes pré-compiladas que residem em um servidor e processam requisições e devolvem respostas, baseado no paradigma cliente/servidor. Esta tecnologia é bastante utilizada na criação de conteúdos dinâmicos e em controles de aplicações JSP (GONÇALVES, 2007);
b) JSP, tratam-se de páginas *HyperText Markup Language* (HTML) dinâmicas. É uma linguagem de *scripts* que permite ao desenvolvedor incorporar elementos dinâmicos em páginas da *web*, utilizando a linguagem Java embutida e algumas *tags* de marcação simples;
c) Enterprise JavaBeans (EJB): “compreende o componente que fica encapsulado no servidor, dentro da arquitetura EE. A tecnologia EJB proporciona rapidez e simplicidade no desenvolvimento distribuído, transacional e seguro de aplicações portáteis baseados na tecnologia Java” (SUN MICROSYSTEMS, 2006, tradução nossa).

JSP “é uma tecnologia baseada em Java que simplifica o processo de sites da web dinâmicos” (FIELDS; KOLB, 2000, p. 2). Ele permite à construção de sites dinâmicos, que utilizem uma infinidade de recursos disponíveis na linguagem Java e a acoplagem de novas

bibliotecas que adicionem recursos a aplicação. Também facilita a *designers* de páginas web e programadores, no processo de codificação e manutenção da aplicação em virtude da utilização de *tags* de marcação específica. Por padrão JSP fornece quatro categorias de tags para utilização. Estas categorias compreendem funções de diretivas (contém informações da maneira como as páginas devem ser processadas e definem propriedades globais da página JSP), de elementos (contém elementos para criação de *scripts* e para gerar conteúdo dinâmico na exibição do resultado final na página), de comentários (documentações de código em geral) e de ações ("podem transferir controle entre páginas, especificar *applets* e interagir com componentes JavaBeans no servidor") (FIELDS; KOLB, 2000, p. 52).

Utilizando como base para desenvolvimento, as quatro categorias de *tags*, padrões da JSP, o programador utiliza as *tags* de elementos e ações para criação de scripts e dinamização da página JSP. Porém a codificação da implementação não fica separada da visualização e isso prejudica a manutenção e a legibilidade do código JSP.

Porém, JSP permite a extensão das *tags* já existentes para a criação de novas *tags* denominadas, *tags* customizadas.

As *tags* customizadas têm por função auxiliar na separação de conteúdo entre as camadas de negócio e apresentação. "As tags personalizadas são explicitamente projetadas para adicionar funcionalidade às páginas JSP, incluindo a geração dinâmica de conteúdo de página, como HTML" (FIELDS; KOLB, 2000, p. 406).

Com a lógica embutida nas *tags*, cria-se a possibilidade de tornar mais fácil a customização dos *layouts* das páginas sem interferir na sua lógica, e modificar questões de lógica sem precisar efetuar alterações na camada de apresentação.

Seu funcionamento é baseado em dois componentes: um arquivo de extensão JAR que contenha a classe que implementa a biblioteca e um arquivo de formato TLD. "[...] o arquivo TLD é um documento XML que descreve as *tags* personalizadas que compõe a biblioteca personalizada" Todd e Szolkowski (2003, p. 98). Sua função é identificar e descrever as *tags* customizadas fazendo o mapeamento de informações como: nome da classe que será utilizada para executar uma determinada *tag*, os atributos que a mesma utilizará para a conversação com a página JSP e o nome que será utilizado pela página JSP para que a mesma seja utilizada.

2.4 TRABALHOS CORRELATOS

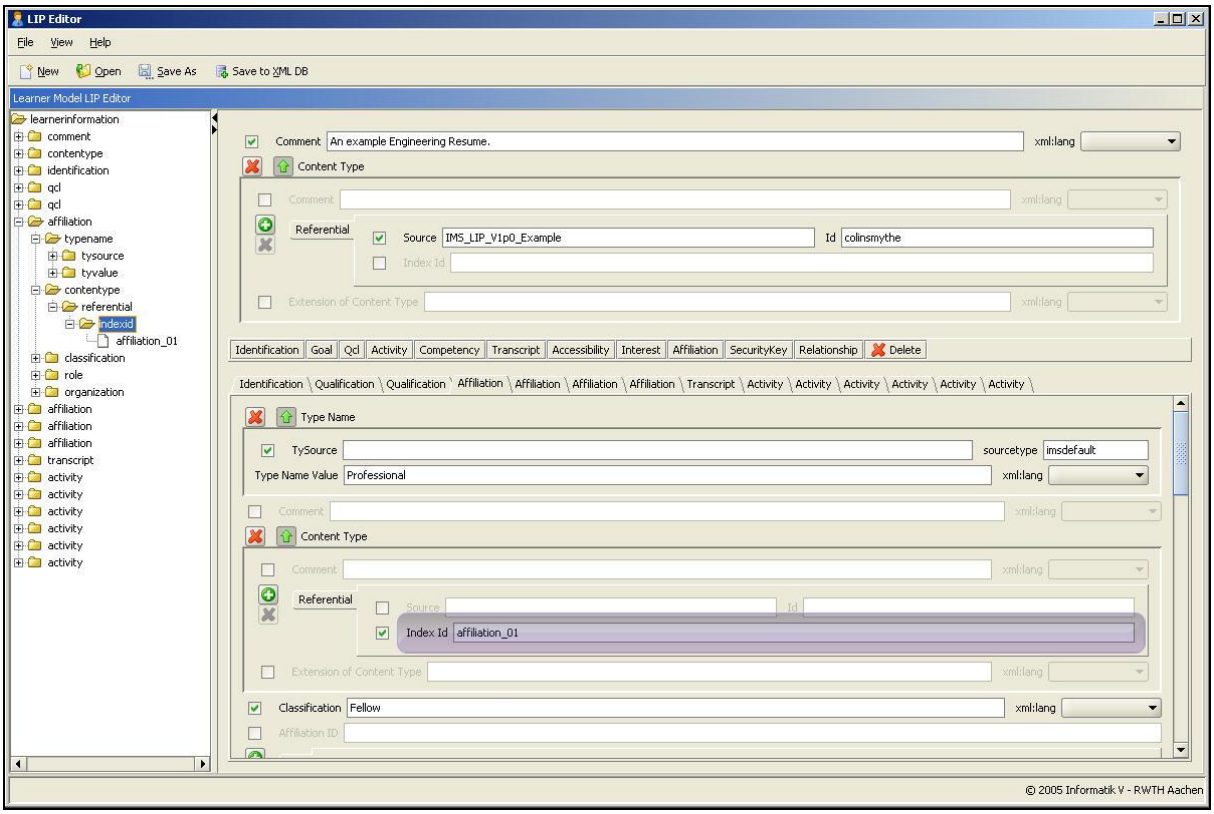
Tendo como objetivo a base à metodologia, os trabalhos descritos a seguir possuem características em comum com o proposto neste momento, dentre os quais foram selecionados o LIP Editor, o Representação de Ambientes Educacionais de Hipermídia Adaptativa através de Ontologias e o OntoLearner.

2.4.1 LIP Editor

"LIP Editor é o primeiro editor de modelagem de alunos que cumpre a especificação IMS LIP" (RWTH AACHEN UNIVERSITY, 2005, tradução nossa). A ferramenta citada, trata-se de um editor, que disponibiliza das funcionalidades relacionadas a criação de arquivos LIP. Dentre elas estão a criação de um novo arquivo, edição de arquivos no formato LIP, importar arquivos gerados por outras ferramentas (desde que o mesmo atenda a especificação), salvar como arquivo XML e visualizar a estrutura do arquivo em texto ou em *Document Object Model* (DOM). Na figura 3 é apresentada à criação de um novo arquivo LIP.

O LIP Editor permite trabalhar com todas as categorias de dados previstas na especificação IMS LIP. Estes dados podem ser criados de maneira hierárquica, sendo apresentados no formato de árvore de diretórios, facilitando a sua navegabilidade. Permite também que uma mesma categoria possua vários níveis de informação. A ferramenta ainda consiste os dados de entrada para os campos e permite a interligação entre duas categorias como previsto na especificação do IMS LIP.

Em geral sua interface é de fácil manipulação permitindo a construção de um XML, com todas as possibilidades previstas pelo modelo IMS LIP. Na figura 3 é apresentada a interface.



Fonte: RWTH Aachen University (2005).

Figura 3 – Ambiente do LIP Editor

2.4.2 Representando Ambientes Educacionais de Hipermídia Adaptativa através de Ontologias

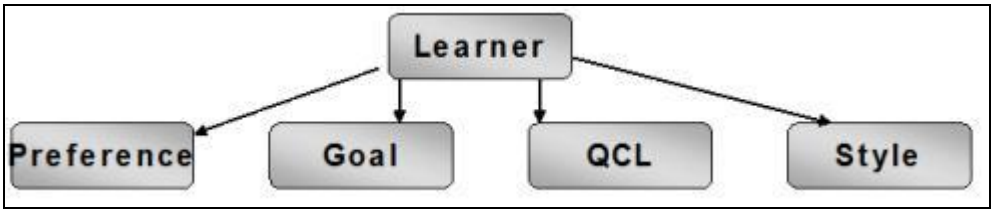
A proposta de um ambiente baseado em ontologias de Hipermídias Adaptativas baseia-se no argumento de que a aprendizagem torna-se mais objetiva e eficiente quando o ensino é feito de modo particular, e essa particularidade pode ser resolvida através da Hipermidia Adaptativa.

> Ambientes educacionais geralmente possuem um grande número de usuários, e cada um deles tem seus próprios objetivos, interesses e estilo de aprendizado. Porém, mesmo diante de tamanha diversidade, os tradicionais ambientes provêem a todos os estudantes os mesmos materiais de ensino estático. (VÉRAS et al., 2008, p. 1).

O Ambiente está subdividido em três modelos que compreendem: o modelo de domínio (lida com o controle dos domínios de ensino e seus conteúdos), o modelo de usuários (que baseiam a representação das suas informações no reuso do padrão IMS-LIP) e o modelo de adaptação que possui técnicas “para adaptar o conteúdo e a interface da aplicação de acordo com cada usuário individualmente”(VÉRAS et al., 2008, p. 3).

2.4.3 OntoLearner

Trata-se de uma proposta de ontologia que utiliza a combinação dos padrões PAPI e IMS LIP para a formatação de um modelo universal para gerenciamento de informações de alunos. Sua ontologia é definida por cinco categorias de informação, provenientes de fontes diferentes. Do modelo PAPI, foi selecionada para utilização a categoria *preferences*, que contém uma listagem das preferências do aluno como: acessibilidade, área de interesse e autores favoritos. Da especificação IMS LIP, incorporaram-se os recursos das categorias QCL e *goal*, que armazenam qualificações, certificações e licenças, e objetivos, respectivamente. Ainda necessitou-se criar uma nova categoria denominada *style*, para armazenamento de informações referente às características cognitivas dos usuários, pois nenhuma das especificações existentes prevê o gerenciamento deste tipo de informação. "Nem o PAPI e nem o LIP incluem a definição de estilos cognitivos e de aprendizagem, que são extremamente importantes para o processo de adaptabilidade de sistemas de acordo com as características do usuário." (MUSA; OLIVEIRA, 2007). Na Figura 4 é apresentado um esquema geral das categorias previstas na ontologia proposta.



Fonte: Musa e Oliveira, (2007).

Figura 4 – Esquema geral das categorias do OrtoLearner

O OntoLearner ainda prevê que os dados mantidos pelo seu modelo, possam ter seu armazenamento de maneira distribuída, onde diferentes aplicações possam se comunicar e realizar troca de informações de alunos e que estas informações possam ser aplicadas e utilizadas para que os sistemas acoplem a eles recursos de hipermídia adaptativa.

3 DESENVOLVIMENTO

Neste capítulo são apresentados os requisitos, a especificação e a implementação do trabalho. Também são apresentados os cenários permitidos para utilização do *framework*, diagramas de classe do *kernel*, do *framework* e das *tags* e o diagrama de seqüência do objeto de busca de dados. Por fim é apresentada a integração do mesmo com o BRUCE[3] e a sua operacionalização.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos em relação à API são:

a) permitir que sejam armazenados os dados do modelo IMS LIP em documentos XML e BDR (Requisito Funcional - RF);
b) permitir que se criem as áreas de dados (arquivos XML ou esquemas e tabelas de BDR), que permitam gravar, ler, excluir e listar todos os registros (RF);
c) permitir que o desenvolvedor defina formas de acesso aos dados, criando uma nova classe descrita nos padrões definidos nos itens a e b (RF);
d) permitir que se acessem os dados no modelo IMS LIP, inclusive permitindo o agrupamento dos usuários, baseado nas onze categorias descritas pelo modelo (RF);
e) utilizar o *framework* XStream para gerenciamento em XML (Requisito Não-Funcional – RNF);
f) ser compatível com os bancos de dados MySQL e Firebird (RNF).

Os requisitos em relação às *tags* customizadas são:

a) fornecer *tags* que tornem possível a listagem de usuários por categoria (RF);
b) montar um formulário de pesquisa por categoria (RF);
c) montar um formulário que permita o cadastro de usuários (RF);
d) permitir a exclusão de usuários (RF);

[3] BRUCE é um ambiente web, criado com o propósito de realizar a validação do componente CELINE

e) tornar o uso destas *tags* compatíveis com qualquer navegador (RNF).

Os requisitos em relação a integração com o CELINE são:

a) implementar uma classe *Data Acess Object* (DAO) que compatibilize os recursos do CELINE com o *framework* (RF);
b) permitir configurar o acesso aos dados no arquivo de configuração do CELINE (`celine-config.xml`) para direcioná-los aos *framework* (RF).

3.2 ESPECIFICAÇÃO

Com o intuito de realizar a especificação do *framework*, foram construídos diagramas existentes na *Unified Markup Language* (UML) e do Modelo de Entidade Relacionamento (MER). Para isto foi utilizado o *Enterprise Architect* no processo de confecção dos artefatos relacionados a UML e o *DBDesigner* para a criação do MER.

3.2.1 *Templates*

Os *templates* são documentos que objetivam estabelecer um modelo que outras páginas web possam acoplar ao seu código e compartilhar características como *layouts* e comportamentos.

Para permitir uma maior flexibilidade na aplicação, o *framework* permite que no seu arquivo de configuração seja definido o arquivo que contenha os arquivos de *template* para o cadastro de usuários.

Este *template* deve conter um HTML onde sejam definidos campos do tipo texto para persistir as informações de usuários que estejam previstas no modelo IMS LIP. Os campos de texto devem ter seus nomes compatíveis aos nomes dos objetos que a aplicação utiliza. Para expressar a hierarquia entre os objetos, os nomes dos campos são baseados em caminhos onde seus vários sub-caminhos são separados por ponto e possuem seus índices delimitados por colchetes. Também existe a possibilidade da utilização de um identificador especial `##id##` que deve ser utilizado em conjunto a um campo do formulário de nome `id`, servindo como um controle da aplicação durante a edição de um usuário. Ainda é possível a utilização de campos

do tipo caixa de checagem. Na utilização deste campo, sua propriedade `checked` deverá conter o caminho para acesso ao valor do objeto, assim como fora feito com os campos de texto. Quando da utilização deste tipo de campo é necessário utilizá-lo combinado a um campo de tipo `hidden` para uso na rotina de salvamento das informações, utilizando de algum recurso como um *script* para realizar a sincronização dos valores.

A *tag* de cadastro de usuários carrega o *template* especificado no arquivo de configuração e o utiliza na criação da página de cadastro.

3.2.2 Diagrama de casos de uso

A figura 5 apresenta o Diagrama de Casos de Uso do *framework.*

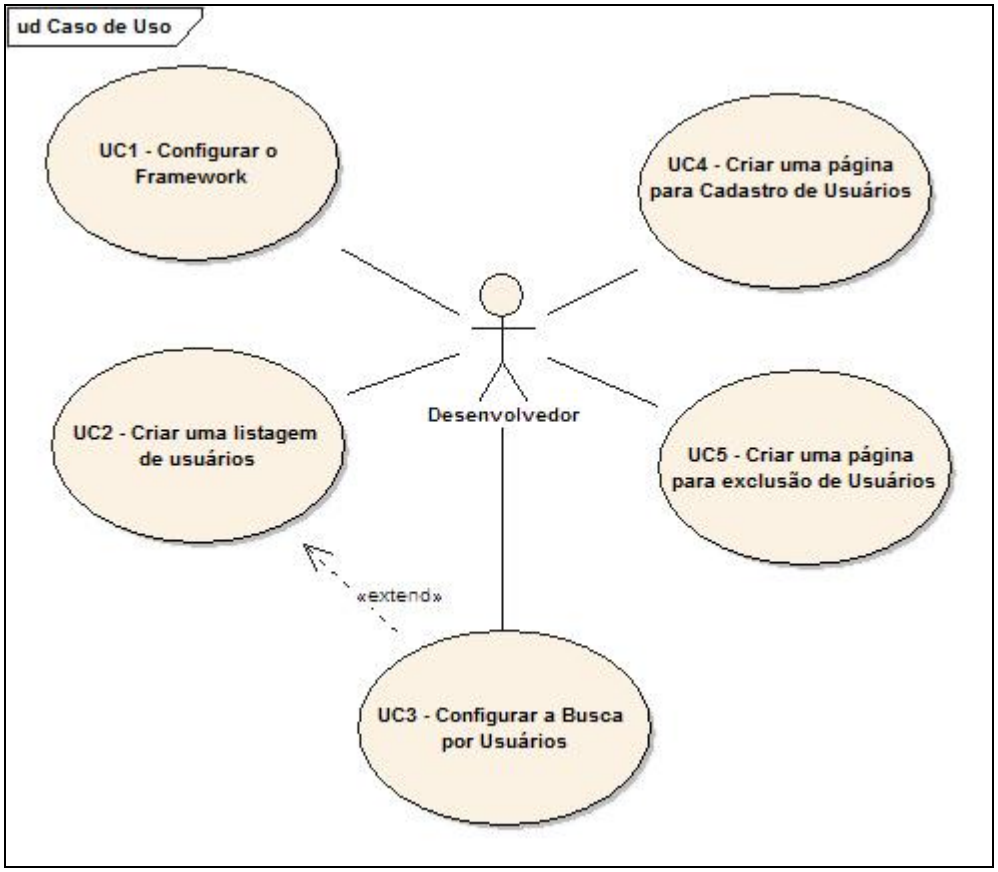


Figura 5 – Diagrama de casos de uso do *framework*

No Quadro 2 é apresentado o cenário para o caso de uso configurar o template das páginas.

UC1 – Configurar o *Framework*

Sumário: Desenvolvedor define o Template das Páginas de cadastro.
Ator primário: Desenvolvedor.
Precondições: Nenhuma.

Fluxo Principal:

1. O desenvolvedor abre o arquivo de configuração celinelip-config.xml.
2. O desenvolvedor define na T*ag* referente ao tpl o diretório que contenha os arquivos de *template*.
3. O desenvolvedor define XML como o tipo de persistência de dados para o sistema.
4. O desenvolvedor define a pasta onde estarão localizados os arquivos de dados.

Fluxo alternativo (1):

1. No passo 3, caso o tipo de persistência seja Relational DataBase (RDB), o desenvolvedor define a url de conexão, usuário e senha.

Fluxo alternativo (2):

1. No passo 3, caso o tipo de persistência seja um Data Acces Object (DAO) customizável, o desenvolvedor define o nome da classe DAO, o nome do seu método de inicialização e os nomes e valores para os seus atributos.

Pós-condições: *Framework* configurado.

Quadro 2 – Cenário do caso de uso de configurar o *framework*

No Quadro 3 é apresentado o cenário para o caso de uso criar uma listagem de usuários.

UC2 – Criar uma Listagem de Usuários

Sumário: Desenvolvedor define\cria um JSP que apresentará a listagem das informações do Usuário, para manutenção desse cadastro.
Ator primário: Desenvolvedor.
Precondições: Nenhuma.

Fluxo Principal:
1. O desenvolvedor cria um arquivo JSP e adiciona a *tag* customizada para listagem de usuários.
2. Dentro da *tag* de listagem de usuários, o desenvolvedor monta uma tabela HTML com as colunas de informações do usuário.
3. O desenvolvedor monta e disponibiliza um link que conterá uma identificação do usuário para que o mesmo possa remeter a uma página de alteração.
4. O desenvolvedor também disponibiliza um link que contenha o identificador do usuário e que remeta a uma página de exclusão.
5. O desenvolvedor adiciona um botão que remeterá a inclusão de um novo usuário passando uma identificação nova.

Pós-condições : uma listagem de usuários com links inclusão, alteração e exclusão é apresentada.

Quadro 3 – Cenário do caso de uso de criar uma listagem de usuários

No Quadro 4 é apresentado o cenário para o caso de uso configurar a busca por usuários.

UC3 – Configurar a busca por Usuários

Sumário: Desenvolvedor define um formulário para filtro dos usuários exibidos na listagem.
Ator primário: Desenvolvedor.
Precondições: Um JSP para listagem de usuários deve ter sido definido.

Fluxo Principal:
1. O desenvolvedor cria um formulário que deve ser adicionado dentro do mesmo JSP de listagem de usuários.
2. O desenvolvedor cria campos que se encaixem nos padrões do *Framework*, para receberem os valores da busca.
3. O desenvolvedor define um campo invisível que conterá a regra para a busca.

Pós-condições: um formulário de busca é apresentado.

Quadro 4 – Cenário do caso de uso de configurar a busca por usuários

No Quadro 5 é apresentado o cenário para o caso de uso criar uma página para o

cadastro de usuários.

UC4 – Criar uma página para o Cadastro de usuários
Sumário: Desenvolvedor define\cria um JSP que conterá o cadastro de usuários. **Ator primário:** Desenvolvedor. **Precondições:** Um identificador ser recebido como parâmetro. **Fluxo Principal:** **1.** O desenvolvedor cria um arquivo JSP e adiciona a *tag* customizada do *framework* para cadastro de usuários. **2.** O desenvolvedor utiliza a variável de *template* que contem o HTML do cadastro. **Pós-condições:** um cadastro com as informações do usuário é apresentado.

Quadro 5 – Cenário do caso de uso de criar uma página para o cadastro de usuários

No Quadro 6 é apresentado o cenário para o caso de uso criar uma página para exclusão de usuários.

UC5 – Criar uma página para exclusão de usuários
Sumário: Desenvolvedor define\cria um JSP que conterá a exclusão de usuários. **Ator primário:** Desenvolvedor. **Precondições:** Um identificador ser recebido como parâmetro. **Fluxo Principal:** **1.** O desenvolvedor cria um arquivo JSP e adicina a *Tag* customizada para a exclusão de usuários. **Pós-condições:** um usuário foi removido.

Quadro 6 – Cenário do caso de uso de criar uma página para exclusão de usuários

3.2.3 Diagrama de Classes

Na figura 6 são apresentadas as classes da camada de controle do *framework* que consistem na estrutura principal e na parte de armazenamento dos dados, e no Quadro 7 elas são explicadas.

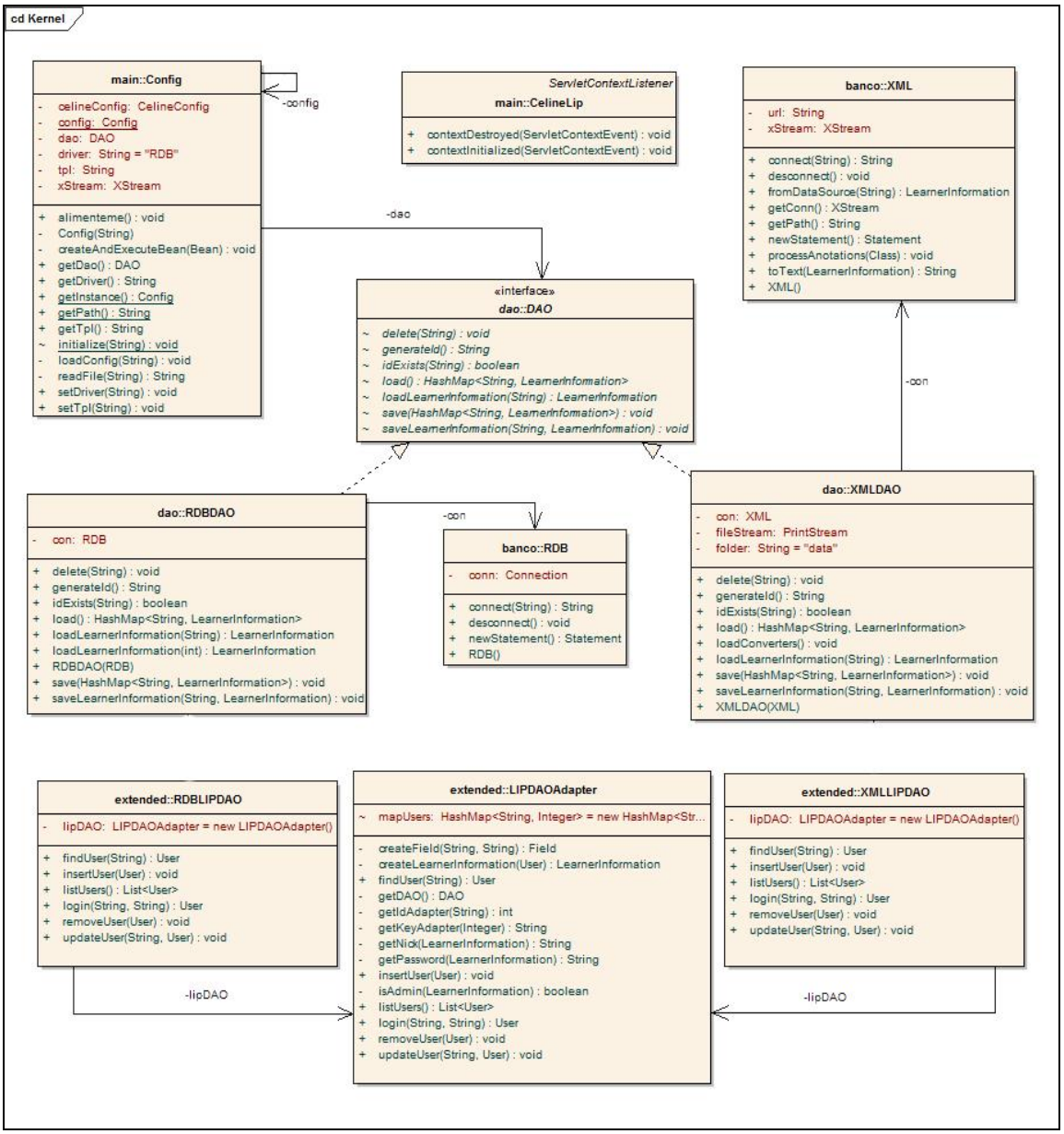
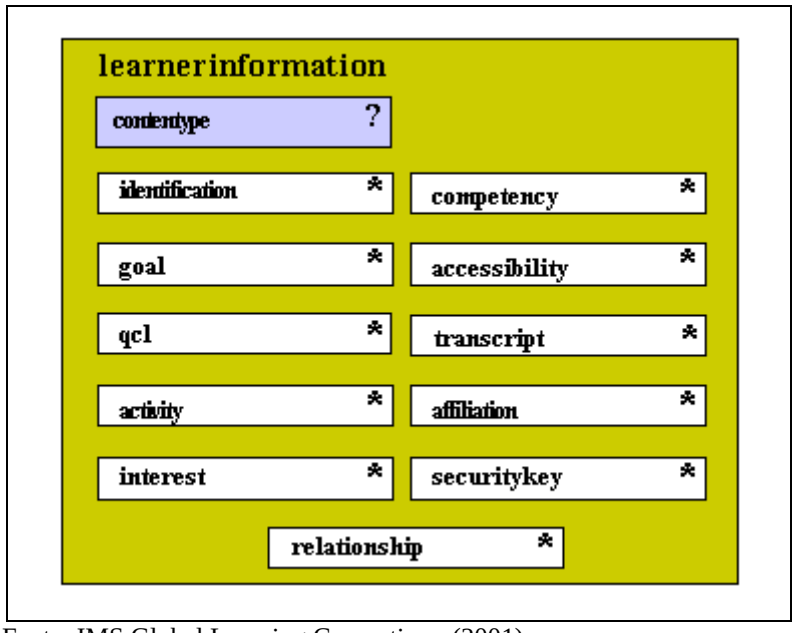


Figura 6 – Diagrama de classes do *framework*

Classe	Descrição
CelineLip	Classe necessária pela Inicialização do *framework* e a chamada do arquivo de configurações.
Config	Classe responsável por manter e decidir a partir do arquivo de configuração, qual a forma de conexão e acesso aos dados será utilizada e manter o caminho do template utilizado na *tag* de cadastro.
RDB	Classe que realiza a conexão a banco de dados do Gênero RDB.
XML	Classe que realiza a conexão a bancos de dados que sejam gravados em arquivos no formato XML.
DAO	Interface que define os métodos de Acesso aos dados dos usuários.
RDBDAO	Classe que realiza DAO e faz o acesso aos dados dos usuários quando o tipo de persistência for RDB.
XMLDAO	Classe que realiza DAO e faz acesso aos dados dos usuários quando o tipo de persistência for XML.
LIPDAOAdapter	Classe que realiza a adaptação e integração entre os objetos de acesso a dados do CELINE e CELINELIP.
RDBLIPDAO	Classe que estende RDBDAO do componente CELINE e utiliza LIPDAOAdapter para se comunicar com o CELINE LIP.
XMLLIPDAO	Classe que estende XMLDAO do componente CELINE e utiliza LIPDAOAdapter para se comunicar com o CELINE LIP.

Quadro 7 – Descrição das classes do *kernel* do *framework*

O modelo IMS LIP tem por finalidade manter informações de alunos em cursos. Estas informações são divididas em categorias conforme exibido na figura 7.



Fonte: IMS Global Learning Consortium, (2001).

Figura 7 – Categorias do modelo IMS LIP

Na figura 8 são apresentadas as classes relacionadas ao modelo IMS LIP e suas 11 categorias, e no Quadro 8 elas são explicadas.

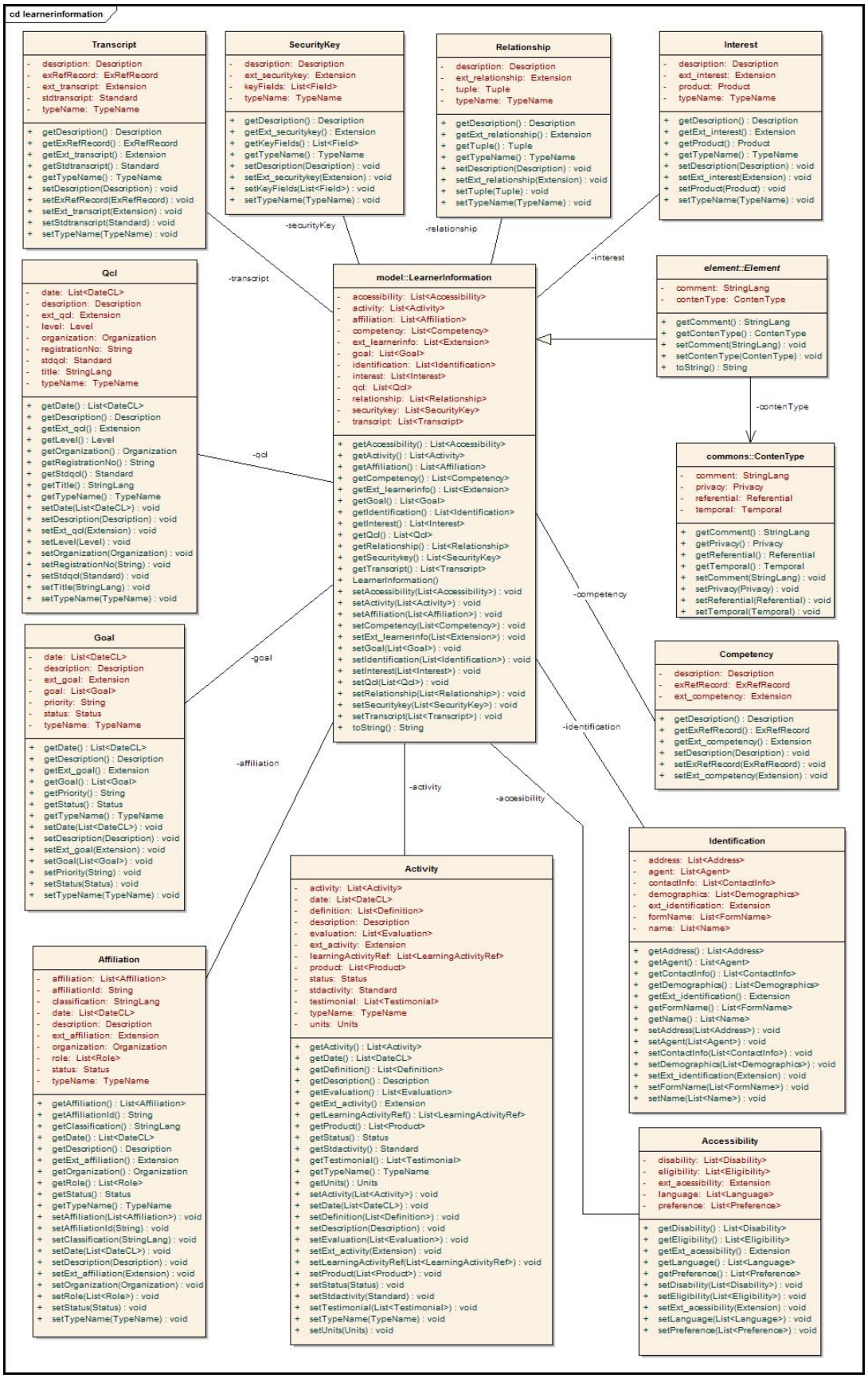


Figura 8 – Classes de modelo do IMS LIP

Classe	Descrição
LearnerInformation	Classe principal do modelo que mantém os dados dos usuários nas onze categorias.
Element	Classe base para as demais classes do modelo, que possui um comentário e campos para definir os tipos de conteúdo que um objeto irá conter.
ContenType	Classe que mantém os tipos de conteúdos dos objetos.
Accessibility	Classe que mantém informações de acessibilidade do usuário.
Activity	Classe que mantém informações de atividades do usuário.
Affiliation	Classe que mantém informações de filiação do usuário.
Competency	Classe que mantém informações de competencia do usuário.
Goal	Classe que mantém informações de objetivos e metas do usuário.
Identification	Classe que mantém informações de identificação do usuário.
Interest	Classe que mantém informações de interesses do usuário.
Qcl	Classe que mantém informações de qualificações, certificações e licensas do usuário.
Relationship	Classe que mantém informações de relacionamento do usuário.
SecurityKey	Classe que mantém informações de informações de acesso e segurança do usuário.
Transcription	Classe que mantém informações de desempenho do usuário.

Quadro 8 – Descrição das classes de modelo do *framework*

Na figura 9 são apresentadas as classes relacionadas às *tags* customizadas e suas dependências e no Quadro 9 elas são explicadas.

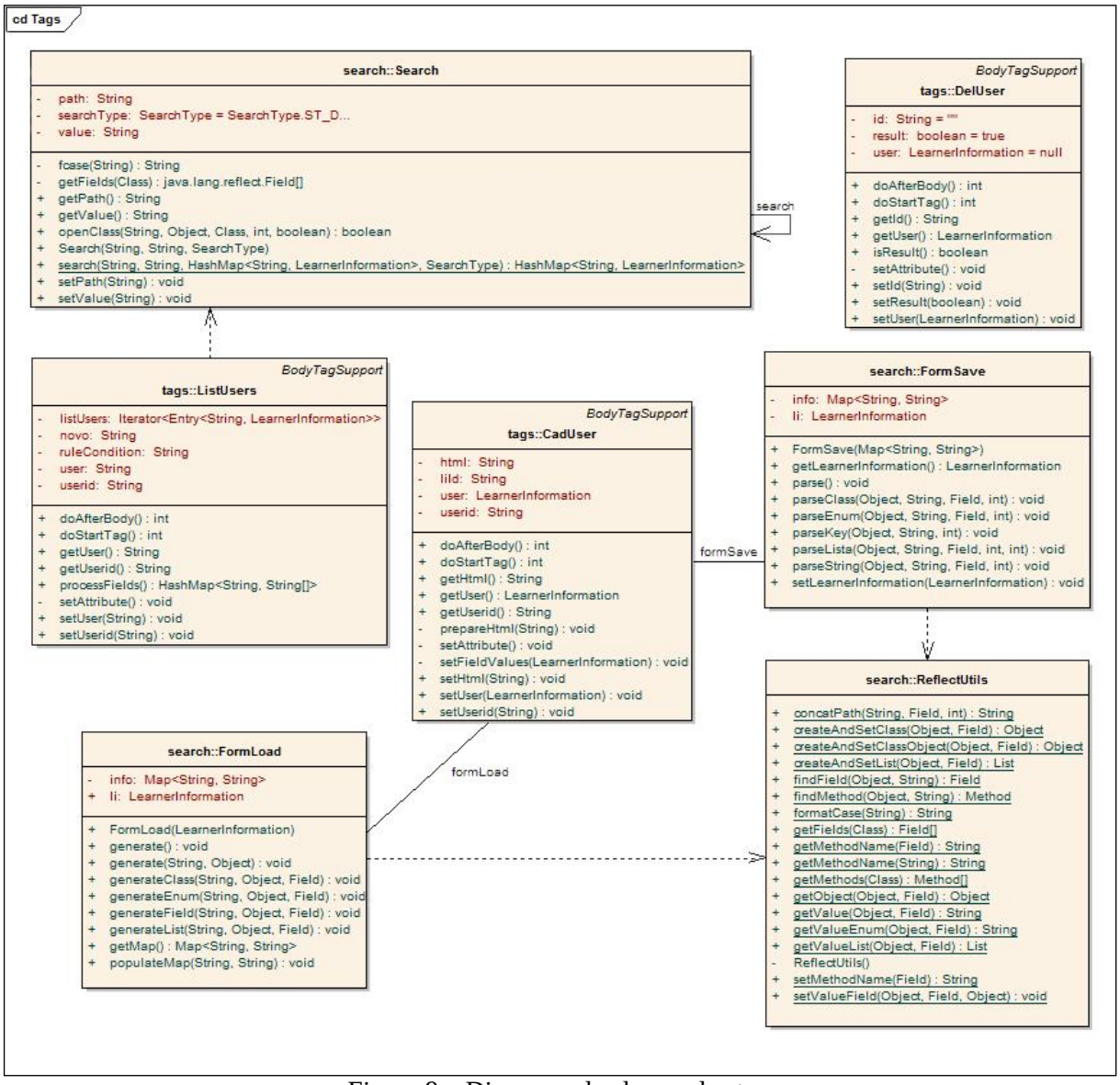


Figura 9 – Diagrama de classes das *tags*

Classe	Descrição
ListUsers	Classe que implementa a *tag* customizada para listagem de usuários e é responsável pelo carregamento dos dados dos usuários, controle dos métodos de busca e apresentação das informações.
Search	Classe que faz leitura nos objetos de modelo por reflexão e busca pelos valores definidos na busca.
DelUser	Classe que implementa a *tag* customizada para exclusão de usuários e faz a chamada no DAO, a partir de um identificador, do método que realiza a exclusão do usuário.
CadUser	Classe que monta o formulário de cadastro dos usuários, carregando informações necessárias e também realiza o controle de persistência dos dados.
FormLoad	Classe que percorre os objetos por reflexão e monta um mapa de valores a serem utilizados no formulário de cadastro de usuários, na ação de edição de um usuário.
FormSave	Classe que percorre um mapa de dados vindos de um formulário e define os valores para os objetos relativos do modelo de dados.
ReflectUtils	Classe que agrupa uma série de funções comuns as classes que utilizam reflexão para obter e definir os valores dos objetos do modelo.

Quadro 9 – Descrição das classes de *tags* customizadas

3.2.4 Arquivo TLD

O arquivo TLD, consiste em um descritor para uma biblioteca de *tags*, onde nele são mapeadas as formas de acesso a biblioteca e seus atributos (SZOLKOWSKI ; TODD , 2003).

Desta maneira dentro do diretório `META-INF` foi definido o arquivo `celinelip.tld`, resolvido pelo endereço `http://www.furb.br/celinelip/tags`, contendo o descritor para as tags: `CadUser`, `ListUsers` e `DelUser`.

No quadro 10 é apresentada a definição do endereço de acesso a biblioteca de *tags* na linha 3 e os descritores das *tags* `listUsers`, `cadUser` e `delUser`.

```
 1.<?xml version="1.0" encoding="UTF-8"?>
 2.<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee [3]"
 3.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4.   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 5.      web-jsptaglibrary_2_0.xsd">
 6.
 7.   <tlib-version>1.0</tlib-version>
 8.   <short-name>CELINELIP</short-name>
 9.   <uri>http://www.furb.br/celinelip/tags</uri>
10.   <jsp-version>1.2</jsp-version>
11.
12.   <tag>
13.      <name>listUsers</name>
14.         <tag-class>br.furb.celinelip.tags.ListUsers</tag-class>
15.            <body-content>JSP</body-content>
16.
17.            <attribute>
18.               <name>user</name>
19.            <required>false</required>
20.         </attribute>
21.
22.         <attribute>
23.            <name>userid</name>
24.            <required>true</required>
25.         </attribute>
26.
27.         <attribute>
28.            <name>novo</name>
29.            <required>false</required>
30.         </attribute>
31.   </tag>
32.   <tag>
33.   <name>delUser</name>
34.      <tag-class>br.furb.celinelip.tags.DelUser</tag-class>
35.      <body-content>empty</body-content>
36.
37.      <attribute>
38.            <name>nextURL</name>
39.            <required>true</required>
40.            <rtexprvalue>true</rtexprvalue>
41.         </attribute>
42.   </tag>
43.
44.   <tag>
45.   <name>cadUser</name>
46.            <tag-class>br.furb.celinelip.tags.CadUser</tag-class>
47.      <body-content>empty</body-content>
48.
49.      <attribute>
50.         <name>var</name>
51.         <required>false</required>
52.      </attribute>
53.
54.      <attribute>
55.            <name>nextURL</name>
56.            <required>true</required>
57.            <rtexprvalue>true</rtexprvalue>
58.         </attribute>
59.    </tag>
60.</taglib>
```

Quadro 10 – Arquivo TLD que contêm as *tags*

A *tag* `listUser` responsável pela listagem dos usuários é uma *tag* que possui um corpo no formato JSP e possui 3 atributos: `userId` (armazena uma lista de identificadores dos usuários), `user` (armazena uma lista de `LearnerInformation`) e `novo` (guarda um identificador para criação de um novo usuário). As *tags* `delUser` e `cadUser` não dispõe de um corpo em sua aplicação e sua propriedade `<body-content>` esta definida como *empty*. Ambas disponibilizam de um atributo `nextURL` que durante a chamada da *tag* obriga a definição de uma página de retorno, para qual a aplicação será remetida após a conclusão das suas operações. A `cadUser` ainda dispõe do atributo `var` que contém o HTML do formulário de cadastro.

3.2.5 Diagrama de Seqüência

A figura 10 demonstra a seqüência de operações executadas na realização de uma busca por um atributo do usuário.

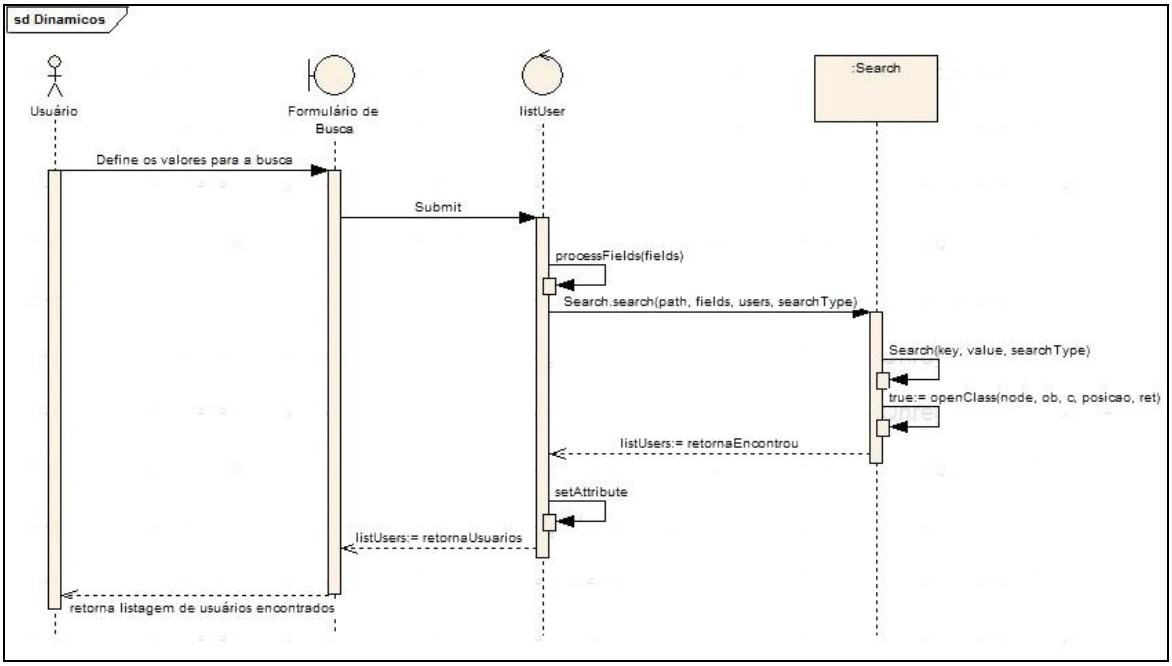


Figura 10 – Diagrama de seqüência do processo de busca

A busca é iniciada através do preenchimento por parte do usuário de um formulário com os dados para busca. Independentemente de qual atributo o usuário esteja procurando, o sistema sempre realizará o mesmo procedimento. O usuário submete o formulário para a classe `ListUsers` que realiza o seu acesso através de um mapa de dados que recebe através

do objeto de *request*. Este mapa é então processado e apenas os itens do mapa que estiverem definidos dentro do padrão do *framework* são utilizados.

`ListUsers` invoca o método estático `search`, que recebe a classe `LearnerInformation` como o endereço inicial para a busca, a lista dos campos de dados, o universo de usuários que deverá ser pesquisado e o tipo de busca que será aplicada. O método `search` cria um objeto da classe `Search` definindo a chave e o valor de busca. Ainda na função `search` é realizada a chamada do método `openClass` para cada usuário dentro do universo de busca definido anteriormente. A chamada de `openClass` abre por reflexão, nodo a nodo, recursivamente, as classes definidas no pacote de modelo, procurando por algum atributo que possua o valor informado inicialmente na pesquisa. Caso encontre uma sentença que satisfaça a condição apresentada inicialmente, o método `openClass` retorna verdadeiro para o método `search` e o mesmo já encarrega-se de adicionar o usuário encontrado a lista de usuários, que posteriormente é retornada a `ListUsers` que através do método `setAttribute` define no atributo `user` da *tag* customizável de cadastro de usuários a lista dos usuários encontrados.

3.2.6 Modelo entidade relacionamento

A figura 11 apresenta o modelo entidade relacionamento que demonstra a estrutura criada para armazenar os dados das 11 categorias do padrão IMS LIP.

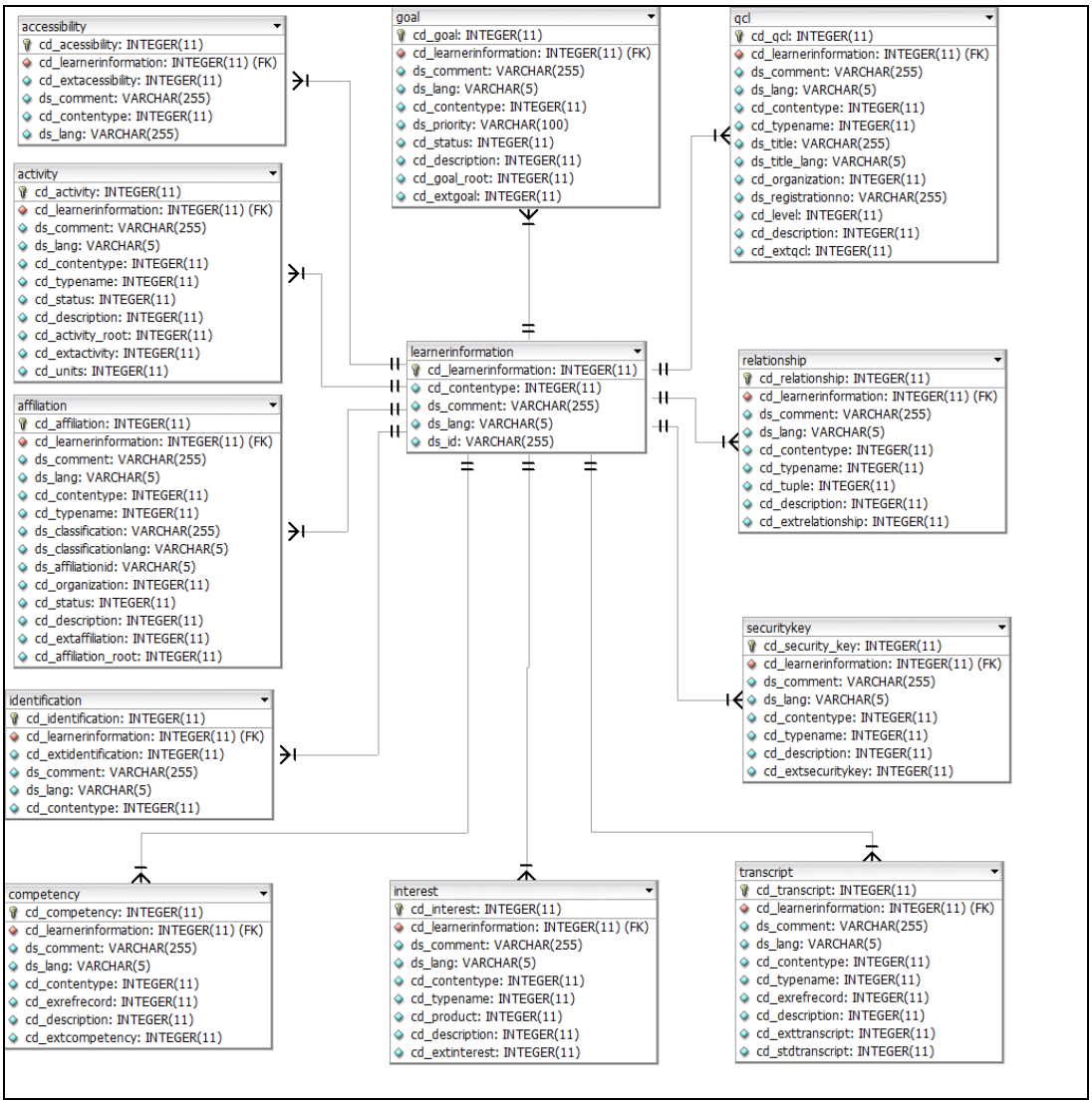


Figura 11 – Modelo entidade relacionamento das 11 categorias do IMS LIP

3.3 IMPLEMENTAÇÃO

Para o desenvolvimento do *framework* foi utilizada a linguagem de programação Java JDK 1.6 através da plataforma de desenvolvimento Eclipse, com o *plugin Web Tools Plataform* (WTP). Para o servidor web foi utilizado o Apache Tomcat 6.0 e o banco de dados MySQL 5.0.

3.3.1 Mapeamento do modelo IMSLIP

Para realizar o mapeamento dos dados especificado pelo padrão IMSLIP, foram criadas classes de modelo para realizar o armazenamento dos dados de uma maneira hierárquica, buscando aproximar-se ao máximo da estrutura oficial que foi apresentada em XML.

Na seqüência identificaram-se quais são as operações necessárias para realizar a manipulação das classes de modelo e seus valores. Utilizou-se o padrão *Data Acces Object* (DAO[4]), para criar uma interface de mesmo nome que fornecesse acesso aos dados do *framework*. Desta maneira foram implementadas, a partir desta interface, duas formas padrões de acesso aos dados: *Relational DataBase* (RDB) e *eXtensible Markup Language* (XML).

No Quadro 11 é apresentada a descrição da interface de acesso dos dados.

```
1.  public interface DAO {
2.
3.      // Salva todos os usuários
4.      void save(HashMap<String, LearnerInformation> hmLearnerInformation)
5.      throws Exception;
6.
7.      // Salva um usuário específico
8.      void saveLearnerInformation(String key, LearnerInformation li)
9.      throws Exception;
10.
11.     // Carrega todos os usuários
12.     HashMap<String, LearnerInformation> load() throws Exception;
13.
14.     // Carrega todos os usuários
15.     LearnerInformation loadLearnerInformation(String id) throws Exception;
16.
17.     // Apaga um usuário específico
18.     void delete(String id) throws Exception;
19.
20.     // Gera uma nova chave para o usuário
21.     String generateId() throws Exception;
22.
23.     // Verifica se usuário existe
24.     boolean idExists(String id) throws Exception;
25. }
```

Quadro 11 – *Interface* DAO

No caso do acesso através de RDB, criou-se uma classe `RDBDao` que baseia sua conexão ao banco de dados, através da *Application Programming Interface (API) Java DataBase Connectivit* (JDBC). Desta maneira seus métodos enviam ao banco de dados instruções no formato *Structured Query Language* (SQL), que realizam operações de

[4] "DAO trata-se de um objeto auxiliar usado para encapsular o acesso a base de dados" (FOWLLER, 2003, p. 153).

consulta, alteração, inclusão e exclusão dos dados armazenados nos objetos, correspondentes no banco de dados.

Para realizar o acesso aos dados através da leitura de arquivos XML, utilizou-se o *framework* XStream[5]. Ele realiza de maneira fácil e ágil a integração entre arquivo XML e classe de modelo. Porém, o *framewok* possui algumas limitações, como a dificuldade para se manipular um nodo que possua ao mesmo tempo um valor e um atributo. No quadro 12 é apresentado um exemplo de uma classe que apresentou o caso citado acima.

```
 1.public class StringLang {
 2.
 3.   private String lang;
 4.   private String content;
 5.
 6.   public StringLang(){
 7.   }
 8.
 9.   public StringLang(String content){
10.         this.setContent(content);
11.   }
12.
13.   public String getLang() {
14.         return lang;
15.   }
16.
17.   public void setLang(String lang) {
18.         this.lang = lang;
19.   }
20.
21.   public String getContent() {
22.         return content;
23.   }
24.
25.   public void setContent(String content) {
26.         this.content = content;
27.   }
28.
29.   public String toString(){
30.         return content;
31.   }
32.}
```

Quadro 12 – Classe `StringLang`

Esta classe é utilizada para armazenar os dados do XML que é apresentado no Quadro 13.

```
1.<qcl>
2.   <comment lang="en">
3.      Military Profile
4.   </comment>
5.</qcl>
```

Quadro 13 – Exemplo de uso de utilização da `StringLang`

[5] "XStream é uma simples biblioteca para serializar objetos para XML e vice – versa" (XSTREAM, 2008, tradução nossa).

Observa-se que o nodo do XML `comment` possui uma propriedade `lang` e também possui um valor e este tipo de situação não pode ser resolvido com o padrão de funcionamento da XStream. Para contornar esta situação necessitou que para cada classe que possuisse características semelhantes a esta, fosse criado um conversor. Os conversores são classes que são registradas no objeto do tipo XStream e que são responsáveis por "prover uma estratégia para conversão particular de tipos de objetos encontrados no grafo de objetos do XML" (XSTREAM, 2008, tradução nossa). Estas classes conversoras deverão implementar a interface `Converter` que está localizada dentro do pacote `com.thoughtworks.xstream.converters`. Esta interface obriga que as classes que a implementarem, realizem os métodos: `marshal` (escreve no XML a partir dos valores de um Objeto), `unmarshal` (faz a leitura do XML e grava os valores encontrados, nas estruturas correspondentes no XML) e `canConvert` (autoriza a utilização do conversor no XML).

No quadro 14, observa-se a implementação do conversor `StringLangConverter`, que é autorizado a converter objetos do tipo `StringLang`.

```
 1.public class StringLangConverter implements Converter{
 2.
 3.  public void marshal(Object value, HierarchicalStreamWriter writer,
 4.     MarshallingContext context) {
 5.     StringLang s = (StringLang)value;
 6.     if (s.getLang() != null){
 7.        writer.addAttribute("lang", s.getLang());
 8.     }
 9.     if (s.getContent() != null){
10.         writer.setValue(s.getContent());
11.      }
12.   }
13.
14.   public Object unmarshal(HierarchicalStreamReader reader,
15.      UnmarshallingContext context) {
16.
17.      String lang = reader.getAttribute("lang");
18.
19.         if (lang==null && reader.getValue() != null){
20.            return new StringLang(reader.getValue());
21.         }else if (lang!=null && reader.getValue() == null){
22.            return new StringLang(null, reader.getAttribute("lang"));
23.         } else {
24.            return new
25.             StringLang(reader.getValue(),reader.getAttribute("lang"));
26.         }
27.   }
28.
29.   public boolean canConvert(Class clazz) {
30.      return clazz.equals(StringLang.class);
31.   }
32.}
```

Quadro 14 – Classe `StringLangConverter`

Para facilitar a integração do *framework* com o XML, utilizou-se de uma funcionalidade diponibilizada pelo XStream denominada Anotações. “Anotações são como meta-tags que você pode adicionar ao seu código e aplicá-los às declarações do pacote, as declarações de tipo, construtores, métodos, campos, parâmetros e variáveis” (DEVELOPER.COM, 2005, tradução nossa). Estas anotações são realizadas dentro das classes de modelo e são feitas na linha acima da declaração da classe e dos atributos, desta maneira o XStream irá processá-las e identificar a qual elemento do código a mesma esta relacionada. Sua utilização tornou-se necessária devido a não total compatibilidade entre os nomes de classes e atributos, entre o modelo de classes e o padrão do XML resultante. Pode-se observar a utilização deste recurso no Quadro 15.

```
1.public class Standard {
2.
3.  @XStreamAsAttribute
4.   private String mode;
5.
6.   ...
7.}
```

Quadro 15 – Exemplo de utilização de atributos

No modelo, `mode` é um atributo da classe Standard, enquanto isso no XML ele é definido como uma propriedade de um nodo `standard`. Caso não fosse definida a anotação `@XStreamAsAttribute`, `mode` seria um novo nodo existente dentro de um nodo principal `standard`, tornando inválido o XML gerado pelo *framework*.

3.3.2 *Tag* `ListUsers`

A *tag* `LisUsers` é a funcionalidade do *framework* que fica responsável por realizar todo o controle sobre a listagens das informações do cadastro de usuários que o desenvolvedor irá apresentar em sua aplicação, permitindo inclusive, aplicar filtros para consulta. Compreende uma classe que estende os recursos existentes na classe `BodyTagSupport`, que pertence ao pacote `javax.servlet.jsp.tagext` e disponibiliza uma série de funcionalidades que auxiliem *tags* que trabalhem com o corpo da página necessitam. Desta classe necessitou ser reimplementado o médoto `doStartTag`, que prevê a realização dos processos que deverão acontecer assim que a *tag* `ListUsers` for iniciada. A primeira operação que o método prevê é o carregamento dos usuários existentes na fonte de dados.

Na seqüência da execução da *tag*, são processados os campos do formulário de busca e montado uma espécie de plano de consulta, para a validação dos registros a serem exibidos.

No Quadro 16 é exibida a chamada da função de busca.

```
 1.private HashMap<String, LearnerInformation>
 2.  processSearch(HashMap<String, LearnerInformation> hashusers,
 3.  HashMap<String, String[]> mapFields){
 4.  // Inicializa hashusers
 5.  HashMap<String, LearnerInformation> users = hashusers;
 6.
 7.  // operador padrao inicia como AND
 8.  boolean opeOR = true;
 9.
10.   // Tokens de indice par devem conter os operadores
11.   int i = 1;
12.
13.   SearchType searchType;
14.   try{
15.      searchType = SearchType.valueOf(this.searchType);
16.   }catch (Exception e) {
17.   searchType = SearchType.ST_DEFAULT;
18.   }
19.
20.   // quebra a String por espaços
21.   StringTokenizer st = new StringTokenizer(this.ruleCondition);
22.
23.   // adiciona os tokes ao mapa de execução
24.   List<String> exeMap = new ArrayList<String>();
25.   while (st.hasMoreElements()){
26.   exeMap.add(st.nextToken());
27.   }
28.
29.   // Percorre toda as ruleCondition
30.   for (String item : exeMap) {
31.   if (i%2 == 0){
32.            if (item.equalsIgnoreCase("AND")){
33.         opeOR = false;
34.      } else if (item.equalsIgnoreCase("OR")){
35.         opeOR = true;
36.      }
37.   }else{
38.      // pega o path
39.      String path = item;
40.      // A lista original deve ser enviada a busca
41.      if (opeOR){
42.         users.putAll(Search.search(path, mapFields.get(path)[0],
43.         hashusers, searchType));
44.      }
45.      // A mesma lista deve ser enviada a busca
46.      else{
47.         users = Search.search(path, mapFields.get(path)[0], users,
48.            searchType);
49.      }
50.      }
51.   i++;
52.   }
53.   return users;
54.}
```

Quadro 16 – A classe `ListUsers` no momento em que realiza a busca

Primeiramente, `ruleCondition` é um campo carregado via `request`, onde nele é definida uma expressão de busca no formato *Backus-Naur Form* (BNF) apresentada no Quadro 17.

```
<ruleCondition>::= <atributo>
<atributo>::= identificador <atributo2>
<atributo2>::= <operador> identificador <atributo2> | ε
<operador>::= AND | OR
```

Quadro 17 – BNF da expressão de busca

A implementação quebra esta chave de busca em *tokens* e entende que as partes da expressão de índice ímpar, se referem aos caminhos que o método de busca deverá percorrer na tentativa de encontrar os valores desejados e as partes de índice par, são os operadores lógicos que serão utilizados para montar o resultado. Para cada `path` de atributo utilizado no formulário, a busca realiza uma chamada para o método `search`. São passados ao método os parâmetros que definem o caminho onde deve ser executada a busca, o valor a ser encontrado, o universo de busca e o tipo de comparação a ser realizada.

Possuindo a listagem dos usuários a ser exibida, é finalizada a execução da *tag* com a atribuição dos valores encontrados para o atributo `pageContext`. Este atributo permite a utilização destes valores dentro do escopo da *tag* que fora descrita durante este capítulo.

3.3.3 A Classe `Search`

`Search` é uma classe utilitária, pertencente ao pacote `br.furb.celinelip.util.search`, que provê uma ferramenta de busca hierárquica em classes e dependências, a valores de atributos de classes do pacote de modelo do *framework* CelineLip. Sua chamada ocorre através da execução da função estática `search` que executa a consulta em um mapa de `LearnerInformation` e retorna um novo mapa, apenas com os objetos `LearnerInformation` que satisfizeram a condição.

Para realizar esta consulta ao mapa, é criada uma instância de `Search`, recebendo como parâmetros o caminho que deverá buscar, o valor a ser encontrado para o caminho especificado e o tipo de comparação na busca. No Quadro 18 é apresentada a chamada a função de busca.

```
 1.public static HashMap<String,LearnerInformation> search (String
 2.   chave, String valor, HashMap<String,LearnerInformation> itens,
 3.   SearchType searchType) {
 4.
 5.  HashMap<String, LearnerInformation> res = new
 7.  HashMap<String,LearnerInformation>();
 8.
 9.  Set<String> keys = itens.keySet();
10.   Search search = new Search(chave, valor, searchType);
11.
12.   for (String key : keys) {
13.      LearnerInformation li = itens.get(key);
14.
15.      boolean ret = search.openClass("learningInformation",
16.             li,   LearnerInformation.class, 0, false);
17.      if (ret){
18.            res.put(key, li);
19.      }
20.   }
21.   return res;
22.}
```

Quadro 18 – Chamada da função de busca

Após isso é invocado o método `openClass`, conforme apresentado na linha 14 do Quadro 18 que abrirá recursivamente, classe a classe, desde que elas pertençam ao pacote `br.furb.celinelip.model`, a partir de `LearnerInformation`, abrindo todas as suas dependências hierarquicamente.

Este método também realiza a comparação entre os valores, assim que consegue alcançar o caminho informado na busca. Esta comparação pode ser realizada de três modos: `ST_CASE_SENSITIVITY` (faz distinção entre maiúsculas e minúsculas), `ST_PARTIAL` (verifica se o valor do campo contém a palavra desejada), `ST_DEFAULT` (compara os valores, sem distinção de maiúsculas e minúsculas).

O Quadro 19 apresenta o método *`openClass`*.

```
 1. public boolean openClass(String node, Object ob, Class c, int posicao,
 2.   boolean ret) {
 3.   if (ret) return ret; if (ob == null) return false;
 4.   String content;
 5.   if (c.isEnum()){
 6.      content = String.valueOf(ob);
 7.   } else{
 8.      try{
 9.         content = ob.toString();
10.      }catch (Exception e) {
11.        content = null;
12.      }
13.   }
14.   if (node.toLowerCase().contains(path)){
15.      if(content != null){
16.         switch (this.searchType) {
17.         case ST_CASE_SENSITIVITY:
18.            if (content.equals(this.value)){
19.               return true;
20.            }
21.            break;
22.         case ST_PARTIAL:
```

```
23.            if (content.toLowerCase().contains(this.value.toLowerCase())){
24.               return true;
25.            }
26.            break;
27.
28.         default:
29.            if (content.equalsIgnoreCase(this.value)){
30.               return true;
31.            }
32.            break;
33.         }
34.      }
35.   }
36.   if (!c.getName().startsWith("br.furb.celinelip.model"))
37.      return false;
38.   try {
39.      posicao++;
40.      java.lang.reflect.Field[] fieldlist = getFields(c);
41.      if (fieldlist.length == 0) {
42.         return false;
43.      }
44.      for (int i = 0; i < fieldlist.length; i++) {
45.         java.lang.reflect.Field fld = fieldlist[i];
46.         if (java.util.List.class.isAssignableFrom(fld.getType())) {
47.            String namemethod = "get" + fcase(fld.getName());
48.            Method met = c.getMethod(namemethod);
49.            Object retob = met.invoke(ob);
50.            List retva = (List) retob;
51.            if (retva != null) {
52.               for (Object o : retva) {
53.                  if(!c.isEnum()){
54.                     if(this.openClass(node+"."+fld.getName(),o,
55.                       o.getClass(), posicao, false)) return true;
56.                  }
57.               }
58.            }
59.         } else {
60.            if (ob != null) {
61.               try {
62.                  fld.setAccessible(true);
63.                  if(!c.isEnum()){
64.                     if(this.openClass(node+"."+fld.getName(),
65.                        fld.get(ob),fld.getType(),
66.                        posicao, false))
67.                           return true;
68.                  }
69.               } catch (Throwable e) {
70.                  e.printStackTrace();
71.               }
72.            }
73.         }
74.      }
75.   } catch (Throwable e) {
76.      e.printStackTrace();
77.   }
78.   return false;
79. }
```

Quadro 19 – Rotina de comparação de valores

Para as classes onde não é realizada a comparação de valores, o método pega a lista de campos do objeto atual, tanto atributos da própria classe, como também das suas ancestrais. A rotina trata duas situações: uma onde o campo é uma lista de objetos e a outra, onde o campo já é o objeto que deve ser aberto por uma nova chamada de `openClass`. Quando a rotina estiver trabalhando com a lista, utiliza-se uma iteração convencional para percorrer cada

elemento da lista e abrir seus objetos, para que também sejam explorados, na busca dos valores conforme apresentado na linha 54 do Quadro 19.

Pode-se observar no Quadro 19 na linha 60, que caso o objeto atual seja nulo, o método não continua sua execução. Também fica claro na linha 63 do Quadro 19 que objetos do tipo enumeração não precisam ser investigados e que assim que um nodo da recursividade recebe uma resposta verdadeira, ele fecha todas as chamadas que estejam em aberto, pois um item localizado já torna toda a consulta válida.

3.3.4 Classe de formulário de dados

Para facilitar a manipulação dos dados que são utilizados pelo formulário, criou-se uma solução para realizar a comunicação entre as classes de salvamento e carregamento dos dados necessários para os formulários de cadastro. Esta solução baseia-se na utilização de caminhos, isto é, os campos *input* do tipo *text* do HTML no formulário de cadastro, devem possuir a sua propriedade *name*, como um caminho relativo para os objetos que armazenam os dados no pacote de modelo, podendo assim ser manipulados por classes utilitárias, que compreendam estes caminhos e os acessem para realizar as operações desejadas.

No quadro 20 é apresentado um código HTML onde existem alguns exemplos de nomeação de campos, que atendem ao padrão compreendido pelas classes utilitárias.

```
1.<tr>
2.   <td class="label"><strong>1.Comment: </strong></td>
3.   <td colspan="7">
4.      <input type="text" name="learnerinformation.comment.content"/>
5.      <input type="text" name="learnerinformation.comment.lang" />
6.   </td>
7.</tr>
```

Quadro 20 – Exemplos de nomeação de campos no HTML

Desta maneira foram criadas duas classes utilitárias principais: `FormLoad` (Responsável por criar um mapa de valores, onde as chaves estejam dentro do padrão utilizado no `form`, baseado em um objeto de informações de usuário) e `FormSave` (recebe um mapa de campos e atribui no atributo de classe relativo no pacote de modelo).

3.3.4.1 A classe `FormLoad`

`FormLoad` é uma classe utilitária, contida dentro do pacote `br.furb.celinelip.util.search`, que prevê uma ferramenta de criação de um mapa de dados baseando sua busca de valores em funções recursivas, que realizem uma busca em profundidade nos objetos do pacote de modelo, utilizando recursos de reflexão e obtenham os valores dos mesmos invocando os seus métodos *getters*.

Inicialmente, é invocado o seu método construtor e o mesmo recebe como parâmetro, o objeto de tipo `LearnerInformation` do qual ele deverá montar um mapa de dados no formato `HashMap` que irá conter em sua chave o caminho do objeto no modelo e o valor para o objeto relativo ao caminho. Na seqüência é necessária a execução do método `generate`, que recebe o nodo inicial de abertura como sendo `learnerinformation`, realizando o início da construção do mapa de valores. Durante a execução de uma chamada a `generate`, primeiro captura-se uma lista de campos, itera-se esta lista, e identifica-se se o elemento atual da iteração trata-se de um objeto `List` ou apenas um objeto. Para cada caso, executa-se um comportamento distinto e métodos especialistas em cada situação são disparados.

No quadro 21 podemos visualizar a lógica de funcionamento de `generate`.

```
 1.public void generate(String path, Object o) {
 2.  Field[] fields = ReflectUtils.getFields(o.getClass());
 3.
 4.  for (int i = 0; i < fields.length; i++) {
 5.      Field f = fields[i];
 6.   if (List.class.isAssignableFrom(f.getType())) {
 7.      generateList(path, o, f);
 8.   } else {
 9.      generateField(path, o, f);
10.      }
11.   }
12.}
```

Quadro 21 – Método `generate`

Observa-se que o método utilizado para listas é `generateList` e para atributos os demais atributos de classe é o `generateField`. O método especializado em listas itera a lista e define um identificador numérico para os itens da lista iniciando em zero, depois concatena o *path* atual com o nome do atributo e o índice entre colchetes e abre novamente o método `generate`, para explorar os seus sub-nodos. Em `generateField` caso o campo seja do tipo `String`, o sistema já popula o mapa de valores do contrário ele trata com métodos distintos atributos de classes do tipo `enum` e os demais tipos de objetos.

No Quadro 22, segue a apresentação do método `generateField`.

```
 1.public void generateField(String path, Object o, Field f) {
 2.  if (o == null) return;
 3.  if (o instanceof String) {
 4.     String newPath = ReflectUtils.concatPath(path, f, -1);
 5.   populateMap(newPath, (String)o);
 6.  } else if (f.getType().isEnum()) {
 7.     generateEnum(path, o, f);
 8.  } else {
 9.   generateClass(path, o, f);
10.   }
11.}
```

Quadro 22 – Método `generateField`

Em `generateClass`, ele novamente verifica se o atributo é do tipo String para popular o mapa de valores e caso não seja, abre novamente a recursão para continuar a varredura dos objetos.

Após toda a execução do processo de montagem do mapa, utiliza-se o método `gethMap` para retornar para a aplicação um mapa que contenha o caminho do campo e o seu valor.

3.3.4.2 A classe `FormSave`

`FormSave` é uma classe utilitária, existente dentro do pacote `br.furb.celinelip.util.search`, que prevê um utilitário de atribuição de dados contidos em um mapa, para o seu objeto relativo no objeto de modelo, baseando sua busca de valores em funções recursivas, que realizem uma busca em profundidade nos objetos do pacote de modelo, utilizando recursos de reflexão e obtenham os valores dos mesmos invocando os seus métodos *setters*.

A classe é instanciada recebendo o mapa de valores que deverão ser atribuídos aos objetos. Na seqüência é necessário passar através do método `setLearnerInformation`, qual objeto `LearnerInformation` será modificado e em seguida executa-se o método parse, que realizará a busca aos objetos, a partir do caminho informado no mapa e a atribuição dos valores aos mesmos. No método `parse` é iterado o mapa de caminhos e valores e executado o método `parsekey` para cada item do mapa.

No Quadro 23, é apresentado o código executado pelo método.

```
 1.public void parseKey(Object o, String key, int pos) {
 2. StringTokenizer st = new StringTokenizer(key, ".");
 3.  int i = 0;
 4.  while (st.hasMoreElements()) {
 5.     String el = st.nextToken();
 6.   if (i++ < pos) {
 7.      continue;
 8.   }
 9.
10.   int index = -1;
11.   if (el.contains("[") && el.contains("]")) {
12.      // trata como lista
13.      String idx = el.substring(el.lastIndexOf('[') + 1,
14.          el.lastIndexOf(']'));
15.            index = Integer.parseInt(idx);
16.      el = el.substring(0, el.lastIndexOf('['));
17.   }
18.   Field f = ReflectUtils.findField(o, el);
19.   if (f == null) continue;
20.   if (index >= 0) {
21.            parseLista(o, key, f, pos, index);
22.   } else if (f.getType().isEnum()) {
23.      parseEnum(o, key, f, pos);
24.   } else if (f.getType().getName().equals(String.class.getName())){
25.      parseString(o, key, f, pos);
26.   } else {
27.      parseClass(o, key, f, pos);
28.   }
29.   }
30.}
```

Quadro 23 – Método `parseKey`

Primeiramente o caminho é quebrado em *tokens* por pontos. Na seqüência, para cada *token* é executada uma nova chamada de `parse`, específica para cada tipo. No caso de `parseLista` é necessário também passar o índice proveniente do `path`. Dentro da chamada de cada método de `parse` é executado o método `createAndSetClassObject` que realiza a criação, e chama o *setter* para o atributo em questão. No Quadro 24, é apresentado a estrutura do método `createAndSetClassObject` .

```
 1.public static Object createAndSetClassObject(Object o, Field f) {
 2.  String el = f.getType().getName();
 3.  Class clz;
 4.  Object obj = null;
 5.  try {
 6.     clz = Class.forName(el);
 7.     obj = clz.newInstance();
 8.   Method m = ReflectUtils.findMethod(o, setMethodName(f));
 9.   m.invoke(o, obj);
10.   ...
11.   return obj;
12.}
```

Quadro 24 – Método `createAndSetClassObject`

Pode-se observar o uso de reflexão para buscar o nome do método a ser executado para realizar a atribuição do valor ao objeto. A função `findMethod`, percorre toda a lista de métodos e realiza a comparação entre os nomes, ignorando maiúsculas e minúsculas.

Por fim, para resgatar o objeto `LearnerInformation` com os novos valores atribuídos pode ser resgatado através da chamada da função `getLearnerInformation`.

3.4 OPERACIONALIDADE

Nesta seção é apresentada a operacionalização do *framework* em uma aplicação WEB. Os exemplos serão exibidos utilizando a ferramenta de desenvolvimento Eclipse Java EE IDE e o banco de dados adotado para este exemplo será o MySQL 5.0. Na seção 3.4.1 serão demonstradas as configurações iniciais para que o *framework* possa entrar em funcionamento. Na seção 3.4.2 será montada a página inicial de listagem de usuários e apresentados os parâmetros e cuidados que devem ser tomados na criação do formulário de busca. Na seção 3.4.3 será demonstrada a criação dos arquivos JSP de cadastro e de exclusão de usuários.

3.4.1 Visão geral do BRUCE

Segundo Vahldick (2008), o BRUCE consiste em um ambiente web que serve como uma aplicação de exemplo e documentação para os desenvolvedores que desejam utilizar o CELINE.

Seu desenvolvimento foi realizado utilizando JSP e as *tags* customizadas do *framework* CELINE, que realizam funções de gerenciamento de cursos, conteúdos de cursos, gerenciamento de usuários e de autenticação no ambiente, de acordo com suas informações de acesso.

Para implementar a questão de segurança o componente trabalha com URLs de extensão "`.do`" que em tempo de execução realizam a substituição para ".jsp". Os arquivos ficam localizados dentro do diretório `WEB-INF\views`, impossibilitando seu acesso diretamente pelo *browser*. "[...] somente a aplicação fornece essas páginas através do repasse de requisição." (VAHLDICK, 2008, p. 93).

A aplicação utiliza do arquivo de configuração `celine-config.xml` no formato XML, que fica localizado no diretório `WEB-INF`. Nele ficam definidas informações como o diretório de armazenamento dos cursos, a página de apresentação de erros e o tipo de persistência utilizada, com suas configurações necessárias, podendo adotar três tipos sendo eles XML, RDB, ou uma classe definida como *Bean* que possa ser criada pelo desenvolvedor. No Quadro 25 é apresentado um exemplo de configuração para o arquivo `celine-config.xml`.

```
 1.<config>
 2.   <courses-folder>file:/C:/temp/courses/</courses-folder>
 3.
 4.   <error-page>error.do</error-page>
 5.
 6.   <database-source>
 7.         <rdb>
 8.               <driver>com.mysql.jdbc.Driver</driver>
 9.               <url>jdbc:mysql://localhost:3306/celine</url>
10.               <user>root</user>
11.               <password></password>
12.         </rdb>
13.   </database-source>
14.</config>
```

Quadro 25 – Apresentação do `celine-config.xml`

Dois perfis de usuários são utilizados na aplicação: administradores e usuários comuns. Os administradores realizam processos como criação e manutenção dos cursos, criação de usuários, e gerência de conteúdos dos cursos. Os usuários comuns que também podem ser definidos como alunos, poderão registrar-se nos cursos, listar os cursos nos quais estão escritos e acompanhar os conteúdos disponibilizados pelo administrador.

3.4.2 Configurações iniciais do *framework*

Após criar a estrutura de uma aplicação web, deve-se copiar os arquivos `celinelip.jar`, `celinelms.jar`, `celinescorm.jar`, `mysql-connector-java-5.1.7-bin.jar`, `xpp3_min-1.1.4c.jar`,`xstream-1.3.1.jar` para a pasta `WEB-INF\lib`.

Na seqüência o desenvolvedor deve adicionar ao seu arquivo `web.xml`, um listener que realiza a inicialização de todo o *framework*. Na *tag* `listener-class` deve ser definida a classe `br.furb.celinelip.main.CelineLip`, como ilustrado no Quadro 26.

```
1.<listener>
2.   <listener-class>br.furb.celinelip.main.CelineLip</listener-class>
3.</listener>
```

Quadro 26 – Criação de um *listener*

O *framework* necessita de um arquivo para configurar o tipo de persistência de dados utilizada e seus parâmetros, e definir o caminho para o arquivo de *template*. Esse arquivo de configuração deve ficar localizado dentro do diretório `WEB-INF` e deve ter o nome `celinelip-config.xml`.

Caso utilize conexão com algum banco de dados relacional, neste exemplo usando o MySQL, deve ser especificado na tag `<rdb>` o *driver* de conexão utilizado, a *string* de conexão do *driver*, e as informações de acesso (usuário e senha do banco de dados), conforme ilustrado no Quadro 27. O esquema de criação das tabelas para o *framework* encontra-se no Apêndice A.

```
 1.<config>
 2.   <database-source>
 3.     <rdb>
 4.        <driver>com.mysql.jdbc.Driver</driver>
 5.        <url>jdbc:mysql://localhost:3306/celinelip</url>
 6.        <user>root</user>
 7.        <password/>
 8.     </rdb>
 9.  </database-source>
10.</config>
```

Quadro 27 – Configuração RDB

No entanto, caso o usuário opte pela utilização de arquivos em disco no formato XML, dentro da *tag* `<xml>` deverá ser informado o diretório onde serão armazenados os arquivos de dados conforme apresentado no Quadro 28.

```
1.<config>
2.   <database-source>
3.      <xml>D:\\Programacao\\workspaces\\usarlip\\data\\n</xml>
4.   </database-source>
5.</config>
```

Quadro 28 – Configuração XML

O *framework* CELINE LIP dispõe de duas alternativas de classes para realizar a integração com o componente CELINE: `RDBLIPDAO` e `XMLLIPDAO`, que encontram-se dentro do pacote `br.furb.celinelip.util.dao.extended`. É necessário referenciar a DAO a ser utilizada *tag* `<database-source>` no seu arquivo de configuração `celine-config.xml`, que assim como o arquivo de configuração do CELINE LIP, também deve estar dentro do diretório `WEB-INF`. Utiliza-se o mecanismo *BEAN* para utilização destas classes.

Caso a persistência do CELINE esteja baseada, no tipo XML do CELINE LIP, o arquivo de configuração deverá estar conforme apresentado no Quadro 29.

```
 1.<config>
 2.  <courses-folder>file:/C:/temp/courses/</courses-folder>
 3.  <error-page>error.do</error-page>
 4.  <database-source>
 5.     <!--
 6.     No XML do CELINELIP
 7.      -->
 8.     <bean class="br.furb.celinelip.util.dao.extended.XMLLIPDAO">
 9.        <bean-attribute name="fileName">WEB-INF/celine.xml</bean-attribute>
10.      </bean>
11.
12.   </database-source>
13.</config>
```

Quadro 29 – Integração CELINE por XML

Conforme apresentado, no Quadro 29, é necessária a definição do atributo `filename` da DAO de XML do CELINE. Desta maneira os usuários serão buscados no tipo de persistência em arquivos XML do CELINE LIP e são repassados à correspondente em XML do CELINE.

Caso a persistência do CELINE esteja baseada, no tipo RDB do CELINE LIP o arquivo de configuração deverá estar conforme apresentado no Quadro 30.

```
 1.<config>
 2.  <courses-folder>file:/C:/temp/courses/</courses-folder>
 3.  <error-page>error.do</error-page>
 4.  <database-source>
 5.     <!--
 6.     No RDB do CELINELIP
 7.      -->
 8.
 9.     <bean class="br.furb.celinelip.util.dao.extended.RDBLIPDAO">
10.
11.        <bean-attribute name="driver">
12.           com.mysql.jdbc.Driver
13.        </bean-attribute>
14.
15.         <bean-attribute name="url">
16.            jdbc:mysql://localhost:3306/celine
17.         </bean-attribute>
18.
19.         <bean-attribute name="user">
20.            root
21.        </bean-attribute>
22.
23.        <bean-attribute name="password"/>
24.     </bean>
25.   </database-source>
26.</config>
```

Quadro 30 – Integração CELINE por RDB

Conforme apresentado, no Quadro 30, é necessária a definição dos atributos `driver`, `url`, `user` e `password` da DAO de RDB do CELINE. Desta maneira os usuários serão buscados no tipo de persistência em RDB do CELINE LIP e repassada à RDB do CELINE.

Pode-se ainda utilizar uma forma de conexão personalizada, onde seja definida a classe DAO implementada para realizar a persistência, o seu método de inicialização e os atributos e valores que a serem utilizadas pela DAO. No Quadro 31 é apresentada a configuração de um

DAO personalizável, definido dentro da *tag* `<bean>`.

```
 1.<bean>
 2.    <class>br.furb.celinelip.util.dao.TestDAO</class>
 3.   <initialize-method>connect</initialize-method>
 4.   <bean-attribute>
 5.      <name>adminUser</name>
 6.      <value>adilson</value>
 7.   </bean-attribute>
 8.   <bean-attribute>
 9.      <name>password</name>
10.      <value>123</value>
11.    </bean-attribute>
12.</bean>
```

Quadro 31 – Definição de DAO personalizável

Finalizando a parte de configuração, ainda é necessário definir o arquivo de *template*. A definição deste arquivo é realizada na *tag* `<tpl>`. No Quadro 32 é realizada a definição do *template* de cadastro.

```
1.<tags>
2.    <tpl>D:\\Programacao\\workspaces\\bruce\VISUAL\\cadastro.html</tpl>
3.</tags>
```

Quadro 32 – Definição do *template*

O *template* deverá conter um formulários simples. Os campos do *template* devem ser apenas do tipo *input* do HTML, com a sua nomeação construída dentro de um padrão que atenda a especificação. Para isso sua propriedade *name* deve seguir o modelo de especificação do *framework* que foi baseado na documentação oficial do IMS LIP e a nomenclatura de seus campos devem ser em caracteres minúsculos. No Quadro 33 é apresentado um *template* básico para cadastrar informações de acesso de usuários.

```
 1.<form>
 2.    <input type="hidden" name="id" value="##id##"/>
 3.
 4.   <input name="learnerinformation.securitykey[0].keyfields[0].fieldlabel.typename.tyvalue"
 5.     type="hidden" value="lmsnick" />
 6.   <input name="learnerinformation.securitykey[0].keyfields[1].fieldlabel.typename.tyvalue"
 7.     type="hidden" value="lmspassword" />
 8.   <input name="learnerinformation.securitykey[0].keyfields[2].fieldlabel.typename.tyvalue"
 9.     type="hidden" value="lmsadmin" />
10.
11.    Nick: <input name="learnerinformation.securitykey[0].keyfields[0].fielddata"
12.              type="text"/> <br />
13.    Senha: <input name="learnerinformation.securitykey[0].keyfields[1].fielddata"
14.               type="text" /> <br />
15.    <input name="learnerinformation.securitykey[0].keyfields[2].fielddata"
16.      id="learnerinformation.securitykey[0].keyfields[2].fielddata" type="hidden"/>
17.
18.    Administrador:
19.    <input type="checkbox" value="true" name="usr"
20.       checked="learnerinformation.securitykey[0].keyfields[2].fielddata"
21.       OnClick= 'document.getElementById(
22.       "learnerinformation.securitykey[0].keyfields[2].fielddata").value = this.checked'
23.     /><br/>
24.
25.    <input type="submit" value="Salvar"/>
26.    <a href="manageuserslip.do">Sair</a>
27.
28.</form>
```

Quadro 33 – *Template* básico de cadastro

Na linha 2 do Quadro 33 é apresentada a inclusão do identificador `##id##` que deve ser colocado em um campo *input* de nome `id` dentro do formulário HTML para que ele seja o controle da *tag* na relação com o usuário em cadastro.

Para campos que trabalham com mais tipos de valores é necessário que seja utilizado o número do índice dentro de colchetes para referenciar com qual item se está trabalhando no momento, como pode ser visualizado na linha 4 do Quadro 33.

Ainda para *input* cujo tipo seja *checkbox* é possível utilizá-lo, definindo na sua propriedade `checked` a mesma nomenclatura utilizada para os campos e utilizando juntamente a ele um *input* de tipo *hidden*, que mantenha o valor que posteriormente será utilizado pela persistência dos dados, seu nome deve ser criado em conformidade com o padrão de campos que vem sendo utilizado. A utilização de um *checkbox* é apresentada entre as linhas 15 e 23 do Quadro 33.

Baseado na especificação do IMS LIP chega-se a conclusão de que o usuário pode ter mais de uma informação de acessibilidade e que estamos trabalhando com a primeira delas.

3.4.3 Criação de listagem

Para criar a página de listagem o desenvolvedor deverá criar um arquivo JSP que contenha em sua primeira linha a seguinte instrução de código, conforme apresentado no Quadro 34.

```
 1.<%@taglib prefix="lip" uri="http://www.furb.br/celinelip/tags" %>
 2.<table>
 3.
 4.<lip:listUsers user="li" userid="meuid">
 5.   <tr>
 6.     <td>
 7.     <a href="editar.do?id=${meuid}">${li.securitykey[0].keyFields[0].fieldData}
 8.     </a></td>
 9.     <td>${li.securitykey[0].keyFields[1].fieldData}</td>
10.     <td>${li.securitykey[0].keyFields[2].fieldData}</td>
11.     <td width="5%"><a href="excluir.do?id=${meuid}">remove</a></td>
12.   </tr>
13.</lip:listUsers>
14.
15.</table>
16.<a href="editar.do?id=${novo}">Novo</a>
```

Quadro 34 – Inclusão da biblioteca de *tags* para listagem

Incluída esta linha, torna-se possível realizar o acesso as *tags* disponibilizadas pelo framework pela propriedade `prefix` definida no Quadro 34. A *tag* `listUsers` é responsável por realizar a listagem dos usuários. Contém atributos `user` e `userid`, que conforme exemplo no Quadro 34 foram nomeados de `li` e `meuid` respectivamente. Estes atributos podem ser

manipulados livremente dentro do escopo da *tag* quando utilizada a notação `${nome_do_atributo}`. Assim como no desenvolvimento do *template* o acesso aos valores das propriedades se realiza por meio do caminho para chegar até o atributo que é baseado na estrutura definida na especificação das classes do modelo e para os casos com mais de um valor, onde o usuário pode ter várias identificações, também utilizam-se os colchetes para acessar índices diferentes de valores. Ambas situações podem ser visualizadas na linha 8 do Quadro 34.

Neste exemplo já está construído o link para as páginas de alteração como na linha 7 e exclusão de acordo com a linha 11, onde a URL recebe apenas o identificador único do usuário, que pode ser capturado do atributo `userid` da *tag* `listUsers`.

Na linha 16 temos ainda a utilização do atributo `${novo}` da *tag*. O mesmo passa um novo identificador, único que foi gerado para a inclusão de um novo usuário.

Agora para que o sistema monte a listagem apenas dos itens desejados, é necessária a criação do formulário de busca. Este formulário deverá possuir sua ação remetendo para a mesma página e os campos de busca deverão possuir a propriedade `name` iniciada com lip. e o restante da propriedade deve conter o nome do atributo relativo a sua definição na especificação, pelo qual se deseja realizar a busca. O desenvolvedor pode colocar quantos campos considerar necessário. Também é necessário definir um campo HTML do tipo `hidden` que deve ser nomeado `lip.ruleCondition`. Seu valor deverá conter a condição de busca a ser realizada. No Quadro 35 é exemplificada a criação do formulário de busca, onde nas linhas 2 e 3 são criados os campos que serão buscados, na linha 4 é definida a expressão de busca e na 6 é definida o tipo de comparação a ser realizada na busca.

```
1.<form action="index.jsp">
2.    <input type="text" name="lip.fielddata"/>
3.    <input type="hidden" name="lip.tyvalue" value=”lmsnick”/>
4.    <input type="hidden" name="lip.ruleCondition"
5.         value="fielddata AND tyvalue"/>
6.      <input type="hidden" name="lip.searchType" value="ST_DEFAULT"/>
7.    <input type="submit" value="Buscar"/>
8.</form>
```

Quadro 35 – Formulário de busca

No campo `lip.rulecondition` podem ser utilizadas as expressões booleanas `AND` e `OR`, permitindo uma maior flexibilidade na busca. No exemplo apresentado no Quadro 35 na linha 4, a expressão de busca é definida por dois campos e é utilizado o operador booleano `AND`.

O campo `lip.searchType` podem ser definido de 3 maneiras: `ST_DEFAULT` (comparação simples de palavras), `ST_PARTIAL` (verifica por pedaços de palavras),

`ST_CASE_SENSITIVITY` (comparação de palavras com distinção entre maiúsculas e minúsculas).

O resultado final da página obtida é a tela de listagem com a possibilidade de busca por usuários cujo *nick* seja "gesi" na Figura 12.

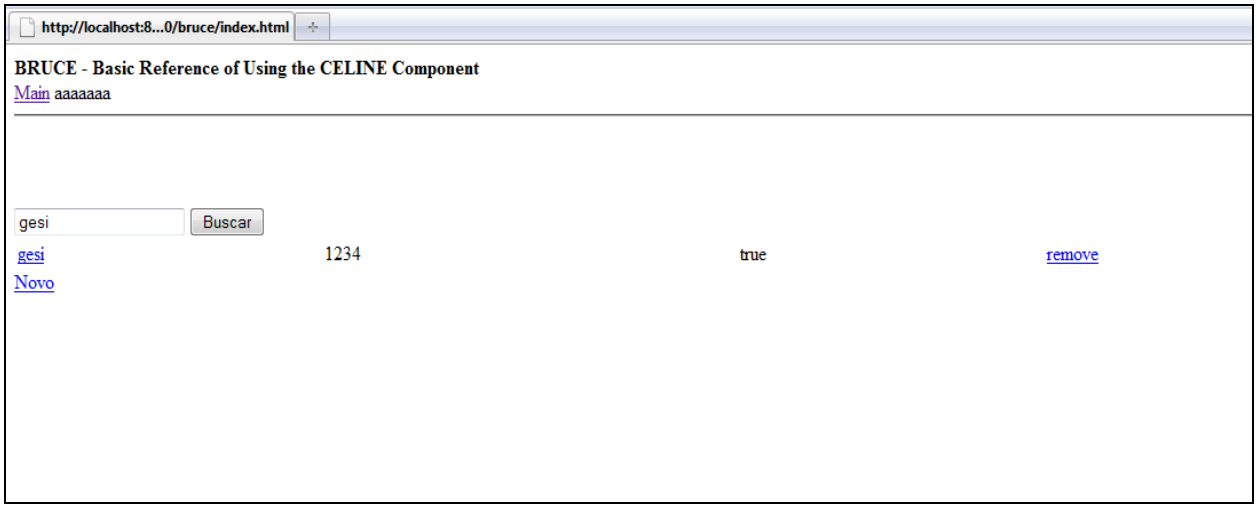


Figura 12 – Tela de listagem com busca

3.4.4 Criação das páginas de cadastro e de exclusão

Para que a aplicação utilize o recurso de cadastro de usuários será utilizada a *tag* `cadUser`, que realizará a inclusão de novos usuários e alteração de dados de usuários já existentes. Deve ser criado um novo arquivo JSP que realizará as operações relacionadas a *tag*, caso ele receba um parâmetro de nome `id` através do *request*, será realizada uma verificação se existe algum usuário com aquela identificação na sua base de dados, não existindo será apresentado o formulário para inclusão de um novo registro, do contrário o formulário será aberto com os dados de um usuário já existente. Para invocar os recursos disponibilizados pela *tag* `cadUser`, a página deverá conter as instruções apresentadas no Quadro 36.

```
1.<%@taglib prefix="lip" uri="http://www.furb.br/celinelip/tags" %>
2.
3.<lip:cadUser var="conteudo" nextURL="../../manageuserslip.do"/>
4.
5.${conteudo}
```

Quadro 36 – *Tag* de cadastro

A primeira linha assim como na página de listagem define que as *tags* definidas na *uri* na linha 1 poderão ser invocadas pelo prefixo `lip`. Desta maneira na linha abaixo é invocada a *tag* de cadastro e na linha seguinte é utilizada a variável `${conteudo}`, que contém o

formulário de cadastro, já carregado de acordo com a variável `tpl`, que foi definido inicialmente no arquivo de configurações. Na utilização da *tag* na linha 3 do Quadro 36, também é definida a página de retorno no atributo nextURL, que será redirecionada, assim que se encerrar a execução do cadastro. Desta maneira é apresentado como saída, o formulário que irá incluir e alterar usuários conforme ilustrado na Figura 13.

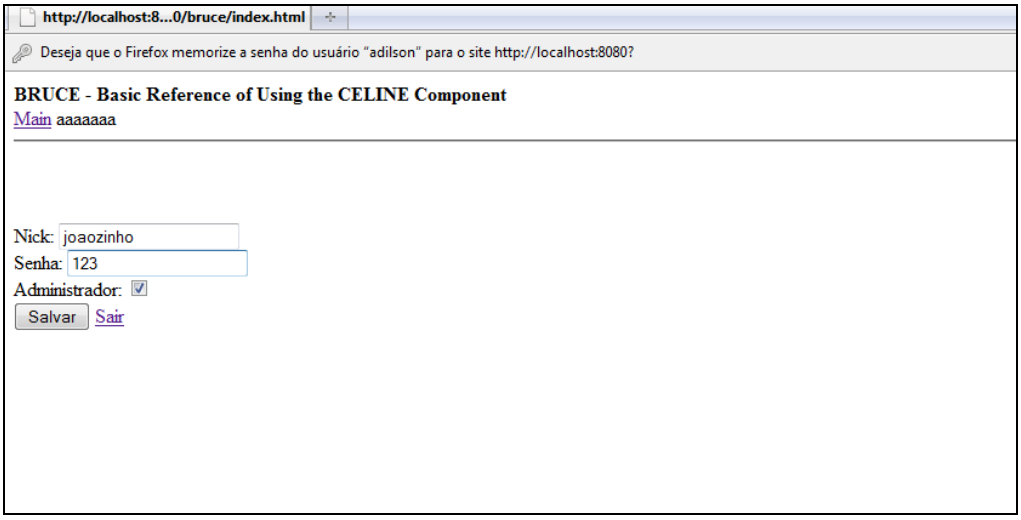


Figura 13 – Tela de listagem com busca

A Figura 13 apresenta o *template* padrão para cadastro baseado nas necessidades apresentadas pelo ambiente BRUCE que necessita apenas das informações de segurança do usuário. O desenvolvedor poderá criar o *template* de acordo com a necessidade de informações a serem armazenadas pela sua aplicação, ou customizar o padrão para atender necessidades específicas.

Por fim, resta apenas a criação da página de exclusão. Esta página deve utilizar a *tag* de exclusão `delUser`. Neste exemplo foi criada uma página JSP com conteúdo conforme Quadro 37.

```
1.<%@taglib prefix="lip" uri="http://www.furb.br/celinelip/tags" %>
2.
3.<lip:delUser nextURL="../../manageuserslip.do"/>
```

Quadro 37 – *Tag* de exclusão

A página utiliza de uma maneira bem simples a *tag* `delUser` que irá simplesmente efetuar a exclusão do registro e disponibilizar um link para retornar a página de listagem. Assim como a *tag* de cadastros é necessário a definição da página de retorno após a exclusão de um usuário. A definição da página de retorno é realizada no atributo `nextURL`.

3.5 RESULTADOS E DISCUSSÃO

Este trabalho alcançou o seu principal objetivo que foi desenvolver um *framework* que atendesse as especificações do padrão IMS LIP.

O *framework* desenvolvido permite tanto o gerenciamento dos dados de aprendizes em cursos, quanto a sua integração com outros sistemas de Ensino a Distância (EaD). Assim como acontecido na integração com o ambiente BRUCE, o *framework* mostrou-se de fácil integração e adaptação para programadores que necessitem utilizar seus recursos.

A utilização das diversas formas de persistência de dados que a ferramenta disponibiliza, funcionaram de uma maneira satisfatória, pois durante os testes demonstrou um bom desempenho e velocidade na leitura e gravação dos seus dados de acesso, mesmo quando o próprio desenvolvedor define uma persistência customizada que seja diferente dos padrões do *framework.*

O procedimento de busca de informações apresentou uma busca ágil, porém, não teve o comportamento desejado quanto ao agrupamento de informações na busca. Caso um aluno X tenha um nível de habilidade bom no idioma Inglês e ruim no idioma Português, e a expressão definida para a consulta for alunos que tenham nível Ruim e o idioma for Inglês, o aluno X será identificado como um resultado que atendeu a condição, pois o componente de busca procura dentro dos dados do aluno as duas situações em separado e depois aplica o operador lógico que no caso foi AND.

O IMS LIP prevê um espaço onde sejam armazenadas informações de extensão, que atendam a necessidades não atendidas pelas categorias existentes. Porém, o modelo não define o formato das estruturas de persistência para estes dados. Desta maneira o *framework* não suporta essa possibilidade em sua construção e informações provenientes destas estruturas não são gerenciadas.

As *tags* customizadas mostraram-se compatíveis aos diversos navegadores existentes no mercado, tendo-se realizado testes mais específicos nos navegadores: Internet Explorer, Mozilla Firefox, Opera e Google Chrome, durante o período de testes do *framework*.

A utilização dos *templates* para a tela de cadastros de usuários possibilita que sejam criadas telas de acordo com a necessidade de campos que a aplicação precisar, porém quando utilizada com todos os campos previstos pela especificação IMS LIP, devido ao grande volume de dados apresentados, a tela de cadastros tende a levar certo tempo até ser montada,

pois a implementação faz busca e substituição de valores, campo à campo da tela, com as informações do usuário já carregadas.

O *template* padrão disponibilizado pelo *framework* atende a todas as categorias previstas na especificação do IMS LIP, porém não permite que as categorias e suas dependências possuam mais de um valor, necessitando que o *template* padrão seja customizado de maneira a atender esta necessidade.

Como pode ser visto no Quadro 38, dos trabalhos apresentados o CelineLIP e o LIPEditor foram as ferramentas que atenderam a maior parte dos requisitos apresentados. Mas o LIPEditor é uma ferramenta com um propósito especifico de edição de arquivos XML, não apresentando recursos como integração com outros sistemas e possibilidade de utilização de outros tipos de persistência de dados.

O OntoLearner possui características que buscam atender realidades específicas, pois utilizam algumas categorias do IMS LIP apenas, que atendam um propósito específico. Em contrapartida, são mais voltados para a aplicação em sistemas de grande porte que necessitem de integração de informações e que utilizem bancos de dados distribuídos.

A proposta “Representação de Ambientes Educacionais de Hipermídia Adaptativa através de Ontologias”, apresenta uma solução que baseia o gerenciamento dos dados de usuários totalmente no IMS LIP, porém explora mais a utilização das informações de usuários, para criar um ambiente que utilize o recurso de Hipermídia Adaptativa e proporciona ao usuário facilidade no manuseio do ambiente.

O CelineLIP, procura atender aplicações de vários níveis e por se tratar de um *framework* oferece um alto grau de compatibilidade quando usado acoplado a outro sistema. Atende a grande parte dos requisitos apresentados pelos demais e nos casos onde ele não atende, possibilita a criação de customizações de maneira a atender as mais adversas necessidades.

Comparação				
	LIP Editor	Representação de Ambientes Educacionais de Hipermídia Adaptativa através de Ontologias	OntoLearner	CelineLIP
Baseado apenas no IMS LIP	X	X	-	X
Vários níveis para uma mesma categoria	X	-	-	-
Suporta informações de extensão	X	-	-	-
Dispõe de interface para cadastro	X	X	-	X
Possibilita integração entre sistemas	-	-	X	X
Permite utilizar vários tipos de persistência	-	-	-	X

Quadro 38 – Comparativo entre o *framework* e os seus trabalhos correlatos

4 CONCLUSÕES

O desenvolvimento do *framework* WEB foi voltado para auxiliar a construção de ambientes de EaD, que necessitem realizar o gerenciamento de informações de usuários, onde fosse preciso tratar um vasto leque de informações pertinentes ao aprendizado. Os ambientes que vierem a utilizar o CELINE LIP podem utilizar *tags* customizáveis no seu desenvolvimento para acessar as informações gerenciadas pelo *framework*. Esta facilidade de integração fica evidente mediante a integração do *framework* ao CELINE e em contrapartida ao ambiente BRUCE. O presente trabalho serve como peça chave na construção de ambientes de EaD, pois realiza a gerencia de informações de usuários, que tende a ser uma grande dificuldade na criação de sistemas, principalmente tratando-se de ambientes de educação a distância que necessitam de um maior controle dos dados de identificação do aprendiz.

Através do desenvolvimento deste trabalho foi possível adquirir conhecimentos sobre várias técnicas de programação, como a utilização de reflexão, que é uma poderosa ferramenta que permite a manipulação de objetos, baseada na análise da estrutura das classes ao qual o objeto pertence. Também proporcionou uma grande oportunidade de evolução nos conceitos de programação WEB baseada em JEE e o aprendizado de *tags* customizadas, que possibilitam a construção de aplicações que apliquem o padrão de desenvolvimento *Model View Controller* (MVC), e proporcionem uma boa legibilidade e mantenabilidade do código fonte.

O objetivo de oferecer uma API de pesquisa e agrupamento dos dados de acordo com as categorias do modelo IMS LIP, foi parcialmente atendido, pois a busca pela informação é executada da maneira correta, porém o agrupamento de informação poderia ser aprimorado de maneira a trazer resultados mais objetivos para as consultas. A maneira que a rotina de busca foi desenvolvida acabou criando a limitação na API. Os demais objetivos foram alcançados com êxito. Baseada na sua arquitetura e na facilidade encontrada no processo de integração com o ambiente BRUCE, o *framework* atigiu seu objetivo de permitir a construção de aplicações WEB que utilizem o padrão IMS LIP para fazer a gerência de seus usuários de uma maneira flexível e segura.

4.1 EXTENSÕES

Segue abaixo sugestões para trabalhos futuros:

a) definir e implementar um formato de extensão de informações para ser utilizado pelo modelo IMS LIP, para armazenar informações não definidas na especificação;
b) criação de um *template* que utilize a tecnologia *Asynchronous Javascript And XML* (AJAX), para facilitar o cadastro de usuários e permitir utilizar as categorias especificadas no modelo atribuindo vários valores para cada uma;
c) rever o processo de busca, de maneira a torná-lo mais objetivo e eficaz, baseado nos problemas apresentados na seção de resultados;
d) desenvolver uma ferramenta de criação de formulários de cadastro de usuários, que permitam a construção de formulários dinâmicos e customizáveis;
e) desenvolver uma ferramenta de relatórios baseada nas informações de usuários gerenciadas pelo *framework*;
f) adaptar o *framework* possibilitando gerenciamento das informações de aprendizes através de Web-services;
g) realizar testes com outros bancos de dados relacionais e
h) melhorar o tratamento de exceções para mostrar com mais clareza os erros de configurações no uso do *framework*.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLETON, Brad. **Patterns and software.** [S.l.], 1997. Disponível em: <http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>. Acesso em: 27 out. 2009.

ASSIS, Patrícia S. de. **Arquitetura para adaptação de sistemas hipermídia.** 2005. 130 f. Tese (Doutorado em Informática) - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0115608_05_pretextual.pdf>. Acesso em: 28 out. 2009.

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

DEVELOPER.COM. **An introduction to java annotations.** [S.l.], 2005. Disponível em: <http://www.developer.com/article.php/3556176>. Acesso em: 21 set. 2009.

FERREIRA, Aurélio B. de H. **Dicionário Aurélio básico da língua portuguesa**. São Paulo: Folha de S. Paulo; Rio de Janeiro: Nova Fronteira, 1995. 687 p.

FIELDS, Duane K.; KOLB, Mark A. **Desenvolvendo na Web com JavaServer Pages.** Tradução Rejane Freitas. Rio de Janeiro: Ciência Moderna, 2000.

FOWLER, M. **Padrões de arquitetura de aplicações corporativas.** Tradução Pearson Education. São Paulo: Bookman, 2003.

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

GHELMAN, Raphael. **Extensão de um sistema de integração de repositórios de objetos de aprendizagem visando a personalização das consultas com enfoque em acessibilidade.** 2006. 128 f. Dissertação (Mestrado em Informática) - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.

GONÇALVES, Edson. **Desenvolvendo aplicações web com JSP e Servlets**. Rio de Janeiro: Ciência Moderna, 2007.

IMS GLOBAL LEARNING CONSORTIUM. **IMS learner information packaging information model specification.** Lake Mary**,** 2001. Disponível em: <http://www.imsglobal.org/profiles/lipinfo01.html>. Acesso em: 23 set. 2009.

MUSA, Daniela L.; OLIVEIRA, José P. M. OntoLearner: uma odontologia para perfis de alunos baseada em padrões**.** In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 18., 2007, São Paulo. Anais...São Paulo: SBC, 2007. P. 483-492. Disponível em: <http://200.169.53.89/download/CD%20congressos/2007/SBIE2007/fscommand/Full/34550.pdf>. Acesso em: 27 out. 2009.

NISKIER, Arnaldo. **Educação à distância:** a tecnologia da esperança. São Paulo: Loyola, 1999.

RWTH AACHEN UNIVERSITY. **IMS LIP editor.** [S.l.], 2005. Disponível em: <http://www-i5.informatik.rwth-aachen.de/i5new/staff/chatti/LIPEditor/index.html>. Acesso em: 24 set. 2009.

SUN MICROSYSTEMS. **Enterprise JavaBeans technology**. [S.l.], 2006. Disponível em: <http://java.sun.com/products/ejb/>. Acesso em: 23 set. 2009.

TODD, Nick.; SZOLKOWSKI, Mark. **Java server pages.** O guia do desenvolvedor. Rio de Janeiro: Campus, 2003.

VAHLDICK, Adilson. **Mecanismo de integração de objetos de aprendizagem com sistemas de apoio ao ensino e aprendizagem.** 2008. 72 f. Projeto de dissertação (Mestrado em Computação Aplicada) – Programa de Mestrado Acadêmico em Computação Aplicada, Universidade do Vale do Itajaí, São José.

VÉRAS, Douglas et al. Representando ambientes educacionais de hipermídia adaptativa através de ontologias. In: WORKSHOP SOBRE INFORMÁTICA NA ESCOLA, 28., 2008, Belém do Pará. **Anais...** Belém do Pará: [s.n.], 2008. p. 442–445. Disponível em: <http://www.prodepa.gov.br/sbc2008/anais/pdf/arq0014.pdf>. Acesso em: 27 out. 2009.

XSTREAM. **Converters.** [S.l.], 2008. Disponível em: <http://xstream.codehaus.org/converters.html>. Acesso em: 23 out. 2009.

W3SCHOOLS. **XML tree.** [S.l.], 2009. Disponível em: <http://www.w3schools.com/xml/xml_tree.asp>. Acesso em: 27 out. 2009.

APÊNDICE A – Relação dos formatos das apresentações dos trabalhos

No Quadro 39 é apresentado o script de criação das tabelas necessárias na utilização do CELINELIP configurado com o tipo de persistência RDB. Para a geração do *script* foi utilizada a ferramenta HeidiSQL.

```
DROP DATABASE IF EXISTS `celinelip`;
CREATE DATABASE `celinelip` /*!40100 DEFAULT CHARACTER SET latin1 */;
USE `celinelip`;

# Table structure for table 'acessibility'
CREATE TABLE `acessibility` (
  `cd_acessibility` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_extacessibility` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `ds_lang` varchar(255) default NULL,
  PRIMARY KEY  (`cd_acessibility`),
  UNIQUE KEY `cd_acessibility` (`cd_acessibility`),
  KEY `cd_acessibility_2` (`cd_acessibility`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'activity'
CREATE TABLE `activity` (
  `cd_activity` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_status` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_activity_root` int(11) unsigned default NULL,
  `cd_extactivity` int(11) unsigned default NULL,
  `cd_units` int(11) unsigned default NULL,
  `cd_stdactivity` int(10) unsigned default NULL,
  PRIMARY KEY  (`cd_activity`),
  UNIQUE KEY `cd_activity` (`cd_activity`),
  KEY `cd_activity_2` (`cd_activity`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'activity_date'
CREATE TABLE `activity_date` (
  `cd_activity_date` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_activity_date`),
  UNIQUE KEY `cd_activity_date` (`cd_activity_date`),
  KEY `cd_activity_date_2` (`cd_activity_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'address'
CREATE TABLE `address` (
  `cd_address` int(11) unsigned NOT NULL auto_increment,
  `cd_identification` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_pobox` varchar(255) default NULL,
```

```
  `ds_pobox_lang` varchar(2555) default NULL,
  `cd_street` int(11) unsigned default NULL,
  `cd_geo` int(11) unsigned default NULL,
  `ds_locality` varchar(255) default NULL,
  `ds_locality_lang` varchar(255) default NULL,
  `ds_city` varchar(255) default NULL,
  `ds_city_lang` varchar(255) default NULL,
  `ds_statepr` varchar(255) default NULL,
  `ds_statepr_lang` varchar(255) default NULL,
  `ds_region` varchar(255) default NULL,
  `ds_region_lang` varchar(255) default NULL,
  `ds_country` varchar(255) default NULL,
  `ds_country_lang` varchar(2555) default NULL,
  `ds_postcode` varchar(255) default NULL,
  `ds_postcode_lang` varchar(255) default NULL,
  `ds_timezone` varchar(255) default NULL,
  `ds_timezone_lang` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stdaddress` int(11) unsigned default NULL,
  `ds_lat` varchar(255) default NULL,
  `ds_long` varchar(255) default NULL,
  PRIMARY KEY  (`cd_address`),
  UNIQUE KEY `cd_address` (`cd_address`),
  KEY `cd_address_2` (`cd_address`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;

# Table structure for table 'affiliation'
CREATE TABLE `affiliation` (
  `cd_affiliation` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_classification` varchar(255) default NULL,
  `ds_classificationlang` varchar(255) default NULL,
  `ds_affiliationid` varchar(255) default NULL,
  `cd_organization` int(11) unsigned default NULL,
  `cd_status` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extaffiliation` int(11) unsigned default NULL,
  `cd_affiliation_root` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_affiliation`),
  UNIQUE KEY `cd_affiliation` (`cd_affiliation`),
  KEY `cd_affiliation_2` (`cd_affiliation`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'affiliation_date'
CREATE TABLE `affiliation_date` (
  `cd_affiliation_date` int(11) unsigned NOT NULL auto_increment,
  `cd_affiliation` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_affiliation_date`),
  UNIQUE KEY `cd_affiliation_date` (`cd_affiliation_date`),
  KEY `cd_affiliation_date_2` (`cd_affiliation_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'agent'
CREATE TABLE `agent` (
  `cd_agent` int(11) unsigned NOT NULL auto_increment,
  `cd_identification` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_commentlang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_agentid` varchar(100) default NULL,
  `cd_agentdomain` int(11) unsigned default NULL,
```

```
  `cd_description` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stdagent` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_agent`),
  UNIQUE KEY `cd_agent` (`cd_agent`),
  KEY `cd_agent_2` (`cd_agent`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'agentdomain'
CREATE TABLE `agentdomain` (
  `cd_agentdomain` int(11) unsigned NOT NULL auto_increment,
  `cd_typename` int(11) unsigned default NULL,
  `cd_stdagentdomain` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_agentdomain`),
  UNIQUE KEY `cd_agent_domain` (`cd_agentdomain`),
  KEY `cd_agent_domain_2` (`cd_agentdomain`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'competency'
CREATE TABLE `competency` (
  `cd_competency` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_exrefrecord` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extcompetency` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_competency`),
  UNIQUE KEY `cd_competency` (`cd_competency`),
  KEY `cd_competency_2` (`cd_competency`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'contactinfo'
CREATE TABLE `contactinfo` (
  `cd_contactinfo` int(11) unsigned NOT NULL auto_increment,
  `cd_identification` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_telephone` int(11) unsigned default NULL,
  `cd_facsimile` int(11) unsigned default NULL,
  `cd_mobile` int(11) unsigned default NULL,
  `cd_pager` int(11) unsigned default NULL,
  `ds_email` varchar(100) default NULL,
  `ds_web` varchar(100) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_contactinfo`),
  UNIQUE KEY `cd_contact_info` (`cd_contactinfo`),
  KEY `cd_contact_info_2` (`cd_contactinfo`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;

# Table structure for table 'contentreftype'
CREATE TABLE `contentreftype` (
  `cd_contentreftype` tinyint(1) unsigned NOT NULL auto_increment,
  `ds_contentreftype` varchar(20) default NULL,
  PRIMARY KEY  (`cd_contentreftype`),
  UNIQUE KEY `cd_content_ref_type` (`cd_contentreftype`),
  KEY `cd_content_ref_type_2` (`cd_contentreftype`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'contentype'
CREATE TABLE `contentype` (
  `cd_contentype` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
```

```
  `ds_lang` varchar(255) default NULL,
  `cd_temporal` int(11) unsigned default NULL,
  `cd_referential` int(11) unsigned default NULL,
  `cd_privacy` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_contentype`),
  KEY `cd_conten_type` (`cd_contentype`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'date'
CREATE TABLE `date` (
  `cd_date` int(11) unsigned NOT NULL auto_increment,
  `ds_datetime` varchar(50) default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extdate` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stddate` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_date`),
  UNIQUE KEY `cd_date` (`cd_date`),
  KEY `cd_date_2` (`cd_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'definition'
CREATE TABLE `definition` (
  `cd_definition` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_definition_root` int(11) unsigned default NULL,
  `cd_extdefinition` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_definition`),
  UNIQUE KEY `cd_definition` (`cd_definition`),
  KEY `cd_definition_2` (`cd_definition`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'definition_field'
CREATE TABLE `definition_field` (
  `cd_definition_field` int(11) unsigned NOT NULL auto_increment,
  `cd_definition` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_definition_field`),
  UNIQUE KEY `cd_definition_field` (`cd_definition_field`),
  KEY `cd_definition_field_2` (`cd_definition_field`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'demographics'
CREATE TABLE `demographics` (
  `cd_demographics` int(11) unsigned NOT NULL auto_increment,
  `cd_identification` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_gender` tinyint(1) unsigned default NULL,
  `ds_placeofbirth` varchar(255) default NULL,
  `ds_placeofbirthlang` varchar(255) default NULL,
  `cd_date` int(11) unsigned default NULL,
  `ds_uid` varchar(100) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stddemographics` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_demographics`),
  UNIQUE KEY `cd_demographics` (`cd_demographics`),
  KEY `cd_demographics_2` (`cd_demographics`)
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'description'
CREATE TABLE `description` (
  `cd_description` int(11) unsigned NOT NULL auto_increment,
  `ds_shortdescription` varchar(255) default NULL,
  `ds_longdescription` mediumtext,
  `ds_longdescription_lang` varchar(255) default NULL,
  `ds_shortdescription_lang` varchar(255) default NULL,
  `cd_fulldescription` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_description`),
  UNIQUE KEY `cd_description` (`cd_description`),
  KEY `cd_description_2` (`cd_description`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'disability'
CREATE TABLE `disability` (
  `cd_disability` int(11) unsigned NOT NULL auto_increment,
  `cd_acessibility` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_extdisability` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stddisability` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_disability`),
  UNIQUE KEY `cd_disability` (`cd_disability`),
  KEY `cd_disability_2` (`cd_disability`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'duration'
CREATE TABLE `duration` (
  `cd_duration` int(11) unsigned NOT NULL auto_increment,
  `cd_evaluation` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_duration`),
  UNIQUE KEY `cd_evaluation_duration` (`cd_duration`),
  KEY `cd_evaluation_duration_2` (`cd_duration`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'eligibility'
CREATE TABLE `eligibility` (
  `cd_eligibility` int(11) unsigned NOT NULL auto_increment,
  `cd_acessibility` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_exteligibility` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stdeligibility` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_eligibility`),
  UNIQUE KEY `cd_eligibility` (`cd_eligibility`),
  KEY `cd_eligibility_2` (`cd_eligibility`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'evalmetadata'
CREATE TABLE `evalmetadata` (
  `cd_evalmetadata` int(11) unsigned NOT NULL auto_increment,
  `cd_typename` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_evalmetadata`),
  UNIQUE KEY `cd_eval_metadata` (`cd_evalmetadata`),
  KEY `cd_eval_metadata_2` (`cd_evalmetadata`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
# Table structure for table 'evalmetadata_field'
CREATE TABLE `evalmetadata_field` (
  `cd_evalmetadatafield` int(11) unsigned NOT NULL auto_increment,
  `cd_evalmetadata` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_evalmetadatafield`),
  UNIQUE KEY `cd_eval_metadata_field` (`cd_evalmetadatafield`),
  KEY `cd_eval_metadata_field_2` (`cd_evalmetadatafield`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'evaluation'
CREATE TABLE `evaluation` (
  `cd_evaluation` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_evaluationid` varchar(100) default NULL,
  `cd_evalmetadata` int(11) unsigned default NULL,
  `cd_status` int(11) unsigned default NULL,
  `ds_noofattempts` varchar(255) default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_evaluation_root` int(11) unsigned default NULL,
  `cd_extevaluation` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stdevaluation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_evaluation`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'evaluation_date'
CREATE TABLE `evaluation_date` (
  `cd_evaluation_date` int(11) unsigned NOT NULL auto_increment,
  `cd_evaluation` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_evaluation_date`),
  UNIQUE KEY `cd_evaluation_date` (`cd_evaluation_date`),
  KEY `cd_evaluation_date_2` (`cd_evaluation_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'exrefrecord'
CREATE TABLE `exrefrecord` (
  `cd_exrefrecord` int(11) unsigned NOT NULL auto_increment,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `ds_recformaturi` varchar(255) default NULL,
  `ds_recformatentityref` varchar(255) default NULL,
  `ds_recdataentityref` varchar(255) default NULL,
  `ds_recdatauri` varchar(255) default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extrefrecord` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_exrefrecord`),
  UNIQUE KEY `cd_ex_ref_record` (`cd_exrefrecord`),
  KEY `cd_ex_ref_record_2` (`cd_exrefrecord`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'exrefrecord_date'
CREATE TABLE `exrefrecord_date` (
  `cd_ex_ref_record_data` int(11) unsigned NOT NULL auto_increment,
  `cd_exrefrecord` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_ex_ref_record_data`),
  UNIQUE KEY `cd_ex_ref_record_data` (`cd_ex_ref_record_data`),
```

```
  KEY `cd_ex_ref_record_data_2` (`cd_ex_ref_record_data`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'extension'
CREATE TABLE `extension` (
  `cd_extension` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_extension`),
  UNIQUE KEY `cd_extension` (`cd_extension`),
  KEY `cd_extension_2` (`cd_extension`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'field'
CREATE TABLE `field` (
  `cd_field` int(11) unsigned NOT NULL auto_increment,
  `cd_fieldlabel` int(11) unsigned default NULL,
  `ds_fielddata` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_vvalue` varchar(255) default NULL,
  PRIMARY KEY  (`cd_field`),
  UNIQUE KEY `cd_field` (`cd_field`),
  KEY `cd_field_2` (`cd_field`)
) ENGINE=MyISAM AUTO_INCREMENT=286 DEFAULT CHARSET=latin1;

# Table structure for table 'formname'
CREATE TABLE `formname` (
  `cd_form_name` int(11) unsigned NOT NULL auto_increment,
  `cd_identification` int(11) default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_text` varchar(255) default NULL,
  `ds_text_lang` varchar(255) default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_form_name`),
  UNIQUE KEY `cd_form_name` (`cd_form_name`),
  KEY `cd_form_name_2` (`cd_form_name`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'fulldescription'
CREATE TABLE `fulldescription` (
  `cd_fulldescription` int(11) unsigned NOT NULL auto_increment,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_media` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_fulldescription`),
  UNIQUE KEY `cd_full_description` (`cd_fulldescription`),
  KEY `cd_full_description_2` (`cd_fulldescription`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'gender'
CREATE TABLE `gender` (
  `cd_gender` tinyint(1) unsigned NOT NULL auto_increment,
  `ds_gender` char(2) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_gender`),
  UNIQUE KEY `cd_gender` (`cd_gender`),
  KEY `cd_gender_2` (`cd_gender`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'geo'
CREATE TABLE `geo` (
  `cd_geo` int(11) unsigned NOT NULL auto_increment,
  `ds_lat` varchar(10) default NULL,
  `ds_lon` varchar(10) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
```

```
  PRIMARY KEY  (`cd_geo`),
  UNIQUE KEY `cd_geo` (`cd_geo`),
  KEY `cd_geo_2` (`cd_geo`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'goal'
CREATE TABLE `goal` (
  `cd_goal` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `ds_priority` varchar(100) default NULL,
  `cd_status` int(11) unsigned default NULL,
  `cd_description` int(11) default NULL,
  `cd_goal_root` int(11) unsigned default NULL,
  `cd_extgoal` int(11) unsigned default NULL,
  `cd_typename` int(10) unsigned default NULL,
  PRIMARY KEY  (`cd_goal`),
  UNIQUE KEY `cd_goal` (`cd_goal`),
  KEY `cd_goal_2` (`cd_goal`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'goal_date'
CREATE TABLE `goal_date` (
  `cd_goal_date` int(11) unsigned NOT NULL auto_increment,
  `cd_goal` int(11) unsigned NOT NULL,
  `cd_date` int(11) unsigned NOT NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_goal_date`),
  UNIQUE KEY `cd_goal_date` (`cd_goal_date`),
  KEY `cd_goal_date_2` (`cd_goal_date`,`cd_goal`,`cd_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'identification'
CREATE TABLE `identification` (
  `cd_identification` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_extidentification` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_identification`),
  UNIQUE KEY `cd_identification` (`cd_identification`),
  KEY `cd_identification_2` (`cd_identification`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;

# Table structure for table 'interest'
CREATE TABLE `interest` (
  `cd_interest` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_product` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extinterest` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_interest`),
  UNIQUE KEY `cd_interest` (`cd_interest`),
  KEY `cd_interest_2` (`cd_interest`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'language'
CREATE TABLE `language` (
  `cd_language` int(11) unsigned NOT NULL auto_increment,
  `cd_acessibility` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
```

```
  `ds_comment_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_extlanguage` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_language`),
  UNIQUE KEY `cd_language` (`cd_language`),
  KEY `cd_language_2` (`cd_language`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'learnerinformation'
CREATE TABLE `learnerinformation` (
  `cd_learnerinformation` int(11) unsigned NOT NULL auto_increment,
  `cd_contentype` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `ds_id` varchar(255) default NULL,
  PRIMARY KEY  (`cd_learnerinformation`),
  UNIQUE KEY `cd_learner_information` (`cd_learnerinformation`),
  UNIQUE KEY `idxDsId` (`ds_id`),
  KEY `cd_learner_information_2` (`cd_learnerinformation`,`cd_contentype`)
) ENGINE=MyISAM AUTO_INCREMENT=153 DEFAULT CHARSET=latin1;

# Table structure for table 'learningactivityref'
CREATE TABLE `learningactivityref` (
  `cd_learningactivityref` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `ds_source` varchar(255) default NULL,
  `ds_id` varchar(100) default NULL,
  `ds_text` varchar(255) default NULL,
  `ds_textlang` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_learningactivityref`),
  UNIQUE KEY `cd_learning_activity_ref` (`cd_learningactivityref`),
  KEY `cd_learning_activity_ref_2` (`cd_learningactivityref`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'level'
CREATE TABLE `level` (
  `cd_level` int(11) unsigned NOT NULL auto_increment,
  `ds_text` varchar(255) default NULL,
  `ds_text_lang` varchar(255) default NULL,
  `cd_level_root` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_level`),
  UNIQUE KEY `cd_level` (`cd_level`),
  KEY `cd_level_2` (`cd_level`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'media'
CREATE TABLE `media` (
  `cd_media` int(11) unsigned NOT NULL auto_increment,
  `ds_mediamode` varchar(100) default NULL,
  `ds_encoding` varchar(100) default NULL,
  `ds_source` varchar(100) default NULL,
  `ds_mymetype` varchar(100) default NULL,
  `cd_contentreftype` int(11) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_media`),
  UNIQUE KEY `cd_media` (`cd_media`),
  KEY `cd_media_2` (`cd_media`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'mobile'
CREATE TABLE `mobile` (
  `cd_mobile` int(11) unsigned NOT NULL auto_increment,
  `ds_country_code` varchar(100) default NULL,
  `ds_area_code` varchar(100) default NULL,
```

```
  `ds_ind_number` varchar(100) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_mobile`),
  UNIQUE KEY `cd_mobile` (`cd_mobile`),
  KEY `cd_mobile_2` (`cd_mobile`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'name'
CREATE TABLE `name` (
  `cd_name` int(11) unsigned NOT NULL auto_increment,
  `cd_identification` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_comment_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stdname` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_name`),
  UNIQUE KEY `cd_name` (`cd_name`),
  KEY `cd_name_2` (`cd_name`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;


# Table structure for table 'objectives'
CREATE TABLE `objectives` (
  `cd_objective` int(11) unsigned NOT NULL auto_increment,
  `cd_evaluation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_extobjective` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_objective`),
  UNIQUE KEY `cd_objective` (`cd_objective`),
  KEY `cd_objective_2` (`cd_objective`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'objectives_contentref'
CREATE TABLE `objectives_contentref` (
  `cd_objectives_content_ref` int(11) unsigned NOT NULL auto_increment,
  `cd_objective` int(11) unsigned default NULL,
  `ds_contentref` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_objectives_content_ref`),
  UNIQUE KEY `cd_objectives_content_ref` (`cd_objectives_content_ref`),
  KEY `cd_objectives_content_ref_2` (`cd_objectives_content_ref`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'objectives_media'
CREATE TABLE `objectives_media` (
  `cd_objectives_media` int(11) unsigned NOT NULL auto_increment,
  `cd_objective` int(11) unsigned default NULL,
  `cd_media` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_objectives_media`),
  UNIQUE KEY `cd_objectives_media` (`cd_objectives_media`),
  KEY `cd_objectives_media_2` (`cd_objectives_media`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'organization'
CREATE TABLE `organization` (
  `cd_organization` int(11) unsigned NOT NULL auto_increment,
  `cd_typename` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_organization`),
  UNIQUE KEY `cd_organization` (`cd_organization`),
  KEY `cd_organization_2` (`cd_organization`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
# Table structure for table 'partname'
CREATE TABLE `partname` (
  `cd_part_name` int(11) unsigned NOT NULL auto_increment,
  `cd_name` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_text` varchar(255) default NULL,
  `ds_textlang` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_stdpartname` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_part_name`),
  UNIQUE KEY `cd_part_name` (`cd_part_name`),
  KEY `cd_part_name_2` (`cd_part_name`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;

# Table structure for table 'preference'
CREATE TABLE `preference` (
  `cd_preference` int(11) unsigned NOT NULL auto_increment,
  `cd_acessibility` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_prefcode` varchar(100) default NULL,
  `ds_prefcode_lang` varchar(255) default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extpreference` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_preference`),
  UNIQUE KEY `cd_preference` (`cd_preference`),
  KEY `cd_preference_2` (`cd_preference`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'privacy'
CREATE TABLE `privacy` (
  `cd_privacy` int(11) unsigned NOT NULL auto_increment,
  `cd_privacyfield` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_privacy`),
  UNIQUE KEY `cd_privacy` (`cd_privacy`),
  KEY `cd_privacy_2` (`cd_privacy`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'product'
CREATE TABLE `product` (
  `cd_product` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_extproduct` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_product`),
  UNIQUE KEY `cd_product` (`cd_product`),
  KEY `cd_product_2` (`cd_product`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'proficiency'
CREATE TABLE `proficiency` (
  `cd_proficiency` int(11) unsigned NOT NULL auto_increment,
  `cd_language` int(11) unsigned default NULL,
  `ds_proficiency` varchar(100) default NULL,
  `ds_proficiency_lang` varchar(255) default NULL,
```

```
  `cd_profmode` int(1) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_proficiency`),
  UNIQUE KEY `cd_proficiency` (`cd_proficiency`),
  KEY `cd_proficiency_2` (`cd_proficiency`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'profmode'
CREATE TABLE `profmode` (
  `cd_profmode` int(11) unsigned NOT NULL auto_increment,
  `ds_profmode` varchar(50) default NULL,
  PRIMARY KEY  (`cd_profmode`),
  UNIQUE KEY `cd_prof_mode` (`cd_profmode`),
  KEY `cd_prof_mode_2` (`cd_profmode`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'qcl'
CREATE TABLE `qcl` (
  `cd_qcl` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `ds_title` varchar(255) default NULL,
  `ds_title_lang` varchar(255) default NULL,
  `cd_organization` int(11) unsigned default NULL,
  `ds_registrationno` varchar(255) default NULL,
  `cd_level` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extqcl` int(11) unsigned default NULL,
  `cd_stdqcl` int(10) unsigned default NULL,
  PRIMARY KEY  (`cd_qcl`),
  UNIQUE KEY `cd_qcl` (`cd_qcl`),
  KEY `cd_qcl_2` (`cd_qcl`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'qcl_date'
CREATE TABLE `qcl_date` (
  `cd_qcl_date` int(11) unsigned NOT NULL auto_increment,
  `cd_qcl` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_qcl_date`),
  UNIQUE KEY `cd_qcl_date` (`cd_qcl_date`),
  KEY `cd_qcl_date_2` (`cd_qcl_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'referential'
CREATE TABLE `referential` (
  `cd_referential` int(11) unsigned NOT NULL auto_increment,
  `ds_source` varchar(100) default NULL,
  `ds_id` varchar(100) default NULL,
  `ds_indexid` varchar(100) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_referential`),
  UNIQUE KEY `cd_referential` (`cd_referential`),
  KEY `cd_referential_2` (`cd_referential`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'relationship'
CREATE TABLE `relationship` (
  `cd_relationship` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
```

```
  `cd_tuple` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extrelationship` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_relationship`),
  UNIQUE KEY `cd_relationship` (`cd_relationship`),
  KEY `cd_relationship_2` (`cd_relationship`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'representation'
CREATE TABLE `representation` (
  `cd_representation` int(11) unsigned NOT NULL auto_increment,
  `cd_demographics` int(11) unsigned default NULL,
  `cd_typename` int(11) default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_representation`),
  UNIQUE KEY `cd_representation` (`cd_representation`),
  KEY `cd_representation_2` (`cd_representation`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'result'
CREATE TABLE `result` (
  `cd_result` int(11) unsigned NOT NULL auto_increment,
  `cd_evaluation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_result_root` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_result`),
  UNIQUE KEY `cd_result` (`cd_result`),
  KEY `cd_result_2` (`cd_result`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'resultinterpretscore'
CREATE TABLE `resultinterpretscore` (
  `cd_resultinterpretscore` int(11) unsigned NOT NULL auto_increment,
  `cd_result` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_resultinterpretscore`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'resultscore'
CREATE TABLE `resultscore` (
  `cd_resultscore` int(11) unsigned NOT NULL auto_increment,
  `cd_result` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_resultscore`),
  UNIQUE KEY `cd_result_score` (`cd_resultscore`),
  KEY `cd_result_score_2` (`cd_resultscore`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'role'
CREATE TABLE `role` (
  `cd_role` int(11) unsigned NOT NULL auto_increment,
  `cd_affiliation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_status` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extrole` int(11) unsigned default NULL,
  `cd_role_root` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_role`),
```

```
  UNIQUE KEY `cd_role` (`cd_role`),
  KEY `cd_role_2` (`cd_role`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'role_date'
CREATE TABLE `role_date` (
  `cd_role_date` int(11) unsigned NOT NULL auto_increment,
  `cd_role` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_role_date`),
  UNIQUE KEY `cd_role_date` (`cd_role_date`),
  KEY `cd_role_date_2` (`cd_role_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'securitykey'
CREATE TABLE `securitykey` (
  `cd_security_key` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_extsecuritykey` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_security_key`),
  UNIQUE KEY `cd_security_key` (`cd_security_key`),
  KEY `cd_security_key_2` (`cd_security_key`)
) ENGINE=MyISAM AUTO_INCREMENT=99 DEFAULT CHARSET=latin1;

# Table structure for table 'securitykey_fields'
CREATE TABLE `securitykey_fields` (
  `cd_security_key_fields` int(11) unsigned NOT NULL auto_increment,
  `cd_security_key` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_security_key_fields`),
  UNIQUE KEY `cd_security_key_fields` (`cd_security_key_fields`),
  KEY `cd_security_key_fields_2` (`cd_security_key_fields`)
) ENGINE=MyISAM AUTO_INCREMENT=286 DEFAULT CHARSET=latin1;

# Table structure for table 'sourcetype'
CREATE TABLE `sourcetype` (
  `cd_sourcetype` tinyint(1) unsigned NOT NULL auto_increment,
  `ds_sourcetype` varchar(15) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_sourcetype`),
  UNIQUE KEY `cd_source_type` (`cd_sourcetype`),
  KEY `cd_source_type_2` (`cd_sourcetype`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'standard'
CREATE TABLE `standard` (
  `cd_standard` int(11) unsigned NOT NULL auto_increment,
  `ds_mode` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_standard`),
  UNIQUE KEY `cd_standard` (`cd_standard`),
  KEY `cd_standard_2` (`cd_standard`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'status'
CREATE TABLE `status` (
  `cd_status` int(11) unsigned NOT NULL auto_increment,
  `cd_typename` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
```

```
  `cd_stdstatus` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_status`),
  UNIQUE KEY `cd_status` (`cd_status`),
  KEY `cd_status_2` (`cd_status`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'street'
CREATE TABLE `street` (
  `cd_street` int(11) unsigned NOT NULL auto_increment,
  `ds_non_fielded_street_address` varchar(255) default NULL,
  `ds_non_fielded_street_address_lang` varchar(255) default NULL,
  `ds_complex` varchar(255) default NULL,
  `ds_complex_lang` varchar(255) default NULL,
  `ds_street_number` varchar(255) default NULL,
  `ds_street_number_lang` varchar(255) default NULL,
  `ds_street_prefix` varchar(255) default NULL,
  `ds_street_prefix_lang` varchar(255) default NULL,
  `ds_street_name` varchar(255) default NULL,
  `ds_street_name_lang` varchar(255) default NULL,
  `ds_street_type` varchar(255) default NULL,
  `ds_street_type_lang` varchar(255) default NULL,
  `ds_street_suffix` varchar(255) default NULL,
  `ds_street_suffix_lang` varchar(255) default NULL,
  `ds_apt_type` varchar(255) default NULL,
  `ds_apt_type_lang` varchar(255) default NULL,
  `ds_apt_num_prefix` varchar(255) default NULL,
  `ds_apt_num_prefix_lang` varchar(255) default NULL,
  `ds_apt_number` varchar(255) default NULL,
  `ds_apt_number_lang` varchar(255) default NULL,
  `ds_apt_num_suffix` varchar(255) default NULL,
  `ds_apt_num_suffix_lang` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_street`),
  UNIQUE KEY `cd_street` (`cd_street`),
  KEY `cd_street_2` (`cd_street`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;

# Table structure for table 'telephone'
CREATE TABLE `telephone` (
  `cd_telephone` int(11) unsigned NOT NULL auto_increment,
  `ds_country_code` varchar(50) default NULL,
  `ds_area_code` varchar(50) default NULL,
  `ds_ind_number` varchar(50) default NULL,
  `ds_ext_number` varchar(50) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_telephone`),
  UNIQUE KEY `cd_telephone` (`cd_telephone`),
  KEY `cd_telephone_2` (`cd_telephone`)
) ENGINE=MyISAM AUTO_INCREMENT=39 DEFAULT CHARSET=latin1;

# Table structure for table 'temporal'
CREATE TABLE `temporal` (
  `cd_temporal` int(11) unsigned NOT NULL auto_increment,
  `cd_typename` int(11) unsigned default NULL,
  `cd_temporalfield` int(10) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_temporal`),
  UNIQUE KEY `cd_temporal` (`cd_temporal`),
  KEY `cd_temporal_2` (`cd_temporal`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'testimonial'
CREATE TABLE `testimonial` (
  `cd_testimonial` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
```

```
  `cd_typename` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_exttestimonial` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_testimonial`),
  UNIQUE KEY `cd_testimonial` (`cd_testimonial`),
  KEY `cd_testimonial_2` (`cd_testimonial`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'testimonial_date'
CREATE TABLE `testimonial_date` (
  `cd_testimonial_date` int(11) unsigned NOT NULL auto_increment,
  `cd_testimonial` int(11) unsigned default NULL,
  `cd_date` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_testimonial_date`),
  UNIQUE KEY `cd_testimonial_date` (`cd_testimonial_date`),
  KEY `cd_testimonial_date_2` (`cd_testimonial_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'transcript'
CREATE TABLE `transcript` (
  `cd_transcript` int(11) unsigned NOT NULL auto_increment,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_comment` varchar(255) default NULL,
  `ds_lang` varchar(255) default NULL,
  `cd_contentype` int(11) unsigned default NULL,
  `cd_typename` int(11) unsigned default NULL,
  `cd_exrefrecord` int(11) unsigned default NULL,
  `cd_description` int(11) unsigned default NULL,
  `cd_exttranscript` int(11) unsigned default NULL,
  `cd_stdtranscript` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_transcript`),
  UNIQUE KEY `cd_transcript` (`cd_transcript`),
  KEY `cd_transcript_2` (`cd_transcript`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'tuple'
CREATE TABLE `tuple` (
  `cd_tuple` int(11) unsigned NOT NULL auto_increment,
  `cd_tupleelement` int(11) unsigned default NULL,
  `cd_tuplerelation` int(11) unsigned default NULL,
  `cd_tuple_root` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_tuple`),
  UNIQUE KEY `cd_tuple` (`cd_tuple`),
  KEY `cd_tuple_2` (`cd_tuple`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'tupleelement'
CREATE TABLE `tupleelement` (
  `cd_tuple_element` int(11) unsigned NOT NULL auto_increment,
  `ds_source` varchar(255) default NULL,
  `ds_id` varchar(255) default NULL,
  `ds_indexid` varchar(255) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `cd_tuple` int(10) unsigned default NULL,
  PRIMARY KEY  (`cd_tuple_element`),
  UNIQUE KEY `cd_tuple_element` (`cd_tuple_element`),
  KEY `cd_tuple_element_2` (`cd_tuple_element`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'tuplerelation'
CREATE TABLE `tuplerelation` (
  `cd_tuple_relation` int(11) unsigned NOT NULL auto_increment,
  `cd_typename` int(11) unsigned default NULL,
  `ds_text` varchar(255) default NULL,
  `ds_text_lang` varchar(255) default NULL,
```

```
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_tuple_relation`),
  UNIQUE KEY `cd_tuple_relation` (`cd_tuple_relation`),
  KEY `cd_tuple_relation_2` (`cd_tuple_relation`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

# Table structure for table 'typename'
CREATE TABLE `typename` (
  `cd_typename` int(11) unsigned NOT NULL auto_increment,
  `ds_tysource` varchar(255) default NULL,
  `cd_sourcetype` tinyint(1) default NULL,
  `ds_tyname` varchar(250) default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  `ds_tyvalue` varchar(255) default NULL,
  PRIMARY KEY  (`cd_typename`)
) ENGINE=MyISAM AUTO_INCREMENT=286 DEFAULT CHARSET=latin1;

# Table structure for table 'units'
CREATE TABLE `units` (
  `cd_units` int(11) unsigned NOT NULL auto_increment,
  `cd_activity` int(11) unsigned default NULL,
  `cd_field` int(11) unsigned default NULL,
  `cd_learnerinformation` int(11) unsigned default NULL,
  PRIMARY KEY  (`cd_units`),
  UNIQUE KEY `cd_unit` (`cd_units`),
  KEY `cd_unit_2` (`cd_units`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Quadro 39 – *Script* de criação do Banco de Dados