UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

Marcelo Scopel

WSMEL: uma arquitetura para integração de serviços educacionais usando dispositivos móveis na formação de comunidades virtuais espontâneas

São Leopoldo
2006

UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

Marcelo Scopel

WSMEL: uma arquitetura para integração de serviços educacionais usando dispositivos móveis na formação de comunidades virtuais espontâneas

Dissertação a ser avaliada como requisito parcial para a obtenção do título de Mestre em Computação Aplicada

Orientador:
Prof. Dr. Sérgio Crespo Coelho da Silva Pinto

São Leopoldo
2006

Ficha catalográfica elaborada pela Biblioteca da
Universidade do Vale do Rio dos Sinos

S422w Scopel, Marcelo
WSMEL: uma arquitetura para integração de serviços educacionais usando dispositivos móveis na formação de comunidades virtuais espontâneas / por Marcelo Scopel. - 2005
109f. : il. ; 29cm.

Dissertação (mestrado) - Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2005. "Orientação: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto, Ciências Exatas e Tecnológicas".

1.Engenharia de Software 2. Arquitetura de Software 3.Ontologia 4.Ensino à Distância I.Título.

CDU 004.41:37.018.43

Catalogação na Publicação:
Bibliotecária Eliete Mari Doncato Brasil - CRB 10/1184

"É uma lei imutável nos negócios que as palavras são palavras, explicações são explicações, promessas são promessas - mas somente o desempenho é realidade."
(Harold S. Green)

Dedicatória

Dedico este trabalho a algumas das pessoas
mais importantes da minha vida: meus pais,
Elaine e Wilmar, pelo exemplo de vida;
minhas irmãs e amigos, pelo afeto e companheirismo.

Agradecimentos

Quero agradecer a algumas pessoas que me auxiliaram ao longo destes dois anos:

- A Deus, pelo dom da vida e por me dar forças para lutar sempre.
- Aos meus pais, pelo apoio e amor incondicional.
- Aos meus amigos: Gilberto, Glauco, Leonardo e Ricardo, pelas trocas de idéia, ajuda e amizade.
- Aos bolsistas Robson e Fabrício, pelo apoio durante a realização deste trabalho.
- Ao amigo e professor Dr. Sérgio Crespo Coelho da Silva Pinto, pela orientação e auxílio durante o desenvolvimento deste trabalho.
- Ao grupo do laboratório de computação móvel da Unisinos (Mobilab), pelos equipamentos emprestados.
- A minha namorada e companheira Viviane Regadas, pela pessoa especial que é, por todo o seu amor, carinho, apoio e compreensão.
- Ao grande amigo Fábio Oliveira Silva, pelas conversas, conselhos e apoio.
- A CAPES e MEC, pelo incentivo financeiro.
- A todos que, de alguma forma, colaboraram para a realização deste trabalho.

Resumo

A Internet e o ensino à distância (EAD) possibilitam a configuração de inúmeros cenários de aprendizagem em um mundo globalizado. A união com dispositivos móveis e a grande variedade de *softwares* educacionais modificam a postura das instituições que implantam ambientes de EAD. O problema deixa de ser "como produzir o *software* educacional que preciso", para ser "onde encontrar, como compor e como usar o *software* que necessito". Entretanto, os *softwares* desenvolvidos para estes ambientes de ensino à distância, apesar de não estarem isolados uns dos outros, ainda são criados com um foco interno. A metodologia de desenvolvimento prevalecente consiste na criação de interfaces diretas, o que obriga os projetistas a especificar, projetar, codificar e depurar programas personalizados para acessarem os dados de um aplicativo, e, então, mapear e converter as respectivas estruturas de dados conforme o necessário, para introduzi-los em outro. Desta forma, cada instância da integração é especializada, a implementação é codificada de maneira monolítica, não proporcionando modularização e tão pouco reuso. Esta pesquisa propõe e implementa uma arquitetura denominada *Web Service Mobile Learning* que, contrária ao desenvolvimento de um ambiente de ensino de maneira monolítica, possibilita a composição de ambientes educacionais e a interoperabilidade de dados entre aplicações independentes da linguagem de implementação. Estes ambientes são personalizados de acordo com as preferências de cada usuário por um determinado tipo de aplicação, e suas comunidades virtuais estabelecidas com base nestas escolhas. A solução utiliza agentes de *software* baseados em *Web services* que, através de consultas a ontologias, realizam a integração entre aplicações. Como forma de testar e validar os serviços desenvolvidos foi realizado um estudo de caso, onde foi possível avaliar e comprovar que a arquitetura é capaz de realizar o objetivo proposto.

Palavras-chave: agente de *software*, integração, interoperabilidade, ontologia, *Web service*.

Abstract

Internet and e-learning allow us to configure several learning environments in a globalized world. The connection with mobile devices and the great variety of educational softwares have modified the way institutions implement e-learning environments. The question that then arises is no longer "how to produce the software I need", but "where to find it and how to integrate it with the software already in use devices to construct a customized educational environment". However, softwares developed to this e-learning environments, despite of not being isolated from each other, are still created with a inner focus. The development methodology most used consist in the creation of direct interfaces, what makes designers to specify, design, codify and purify customized programs to access the softwares data and, then, assign and convert the correspondent data structures as needed, introducing them into another. So, its possible to affirm that each integration instance is specialized, the setup is codified in a monolithic way, what does not permit modulating and reusing as well. This research proposes and implements a so called architecture Web Service Mobile Learning what, against the developing of a monolithic teaching environment, makes possible the educational environments composing and the data interoperability among applications, no matter which implementation language is used. Those environments are customized according to each and every user's preferences, and his virtual communities are established through those choices. The solution uses software agents based in Web Services that, through ontology searches, make the integration among applications. As a way to test and validate the developed services was made a case study, when was possible to evaluate and prove that the architecture is able to do the proposed objective.

Keywords: interoperability, integration, ontology, software agent, web Service.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas

API – *Application Program Interface*
AI – *Administrator In-Classroom*
B2B – *Business to Business*
CIM – Classe, Interface e Metadados
DAML – *The DARPA Agent Markup Language*
DOM – *Document Object Model*
DTD – *Document Type Definition*
EAD – Educação à Distância
EIA – *Enterprize Application Integration*
EUME – *Ubiquitous and Multimedia Environment for Education*
FTP – *File Transfer Protocol*
GIC – *Group In-Classroom*
HTML – *Hyper Text Markup Language*
HTTP – *HyperText Transfer Protocol*
IEEE – *Institute of Electrical and Electronics Engineers*
IIS – *Internet Information Services*
IMS – *Instructional Management Systems*
LOM – *Learning Object Metadata*
LTSC – *Learning Technology Standards Committee*
MAS – *Multi Agent System*
MIME – *Multipurpose Internet Mail Extension*
MVC – *Mode View Controller*
PDA – *Personal Digital Assistant*
POC/I – *Professor In-Classroom Interface*
RDF – *Resource Description Framework*
RMI – *Remote Method Invocation*
RPC – *Remote Procedure Call*
SAX – *Simple API for XML*
SGML – *Standard General Markup Language*
SIC – *Student In-Classroom*
SMTP – *Simple Mail Transfer Protocol*
SOAP – *Simple Object Access Protocol*
TI – Tecnologia da Informação
UDDI – *Universal Distribution Discovery and Integration*
UNISINOS – Universidade do Vale do Rio dos Sinos
UML – *Unified Modeling Language*
UNSPC – *United Nations Development Program*
URI – *Uniform Resource Indicator*
URL – *Uniform Resource Locator*
W3C – *World Wide Web Consortium*
Wi-Fi – *Wireless Fidelity*

WSDL – *Web Services Description Language*
WSMEL – *Web Services Mobile Environment Learning*
WWW – *World Wide Web*
XML – *eXtensible Markup Language*
XSD – *XML Schema Definition*
XSLT – *eXtensible Stylesheet Language Transformations*

Sumário

CAPÍTULO 1

Introdução

O uso da computação móvel e a sua difusão permitiu o desenvolvimento de aplicações móveis também na área da educação, implicando em uma nova modalidade de ensino à distância, chamada de *mobile learning* ou, simplesmente, *M-Learning* (Pagani, 2005).

Assim como em outras áreas, no ensino à distância, nenhuma aplicação está isolada das outras, e apesar de muitas terem sido criadas com um foco interno, unir aplicações tem se tornado comum (Scopel, 2005). No entanto, conectar *softwares* é mais do que trocar *bytes*, é, portanto, necessária a criação de processos que unam aplicações separadas de forma coerente (Chappell, 2004).

Na maioria das vezes, a metodologia de desenvolvimento prevalecente utilizada para as interfaces de aplicativo consiste na criação de interfaces diretas, de ponto a ponto. Assim, os desenvolvedores que têm conhecimento das *Application Program Interfaces* (APIs) respectivas dos aplicativos de interface especificam, projetam, codificam e depuram programas personalizados para acessarem os dados do aplicativo fonte, e, então, mapear e converter as respectivas estruturas de dados conforme o necessário, para introduzi-los no aplicativo alvo. Isso produz um conjunto de funções altamente acoplado e específico que existe e é executado na forma de um código de procedimento, assim como os próprios aplicativos. Portanto, uma vez que cada instância de integração é especializada, a implementação é codificada de maneira monolítica e não é modular nem reutilizável (Chappell, 2004).

O WSMEL, um acrônimo para *Web Service Mobile Environment Learning*, por sua vez, permite ao usuário de um ambiente de *M-Learning* conectar diversas aplicações, independente das linguagens às quais foram desenvolvidas; e, então, criar um ambiente educacional personalizado, integrado e modular, de acordo com as preferências no uso de determinados *softwares*. As aplicações desenvolvidas com base nesta arquitetura possibilitam a comunicação com um bom número de outros *softwares*, permitindo a troca de informações entre aplicativos de maneira bidirecional. Por esta razão, a integração não pode estar acoplada ao código fonte de cada aplicação, pois teríamos um problema sempre que um novo *software* fosse acoplado ao ambiente. Sendo assim, a arquitetura foi projetada com base em quatro camadas: a camada de agentes, a camada de interoperabilidade, a camada de clientes e a camada de persistência, sendo que a interoperabilidade entre aplicativos está fundamentada principalmente na interação entre duas camadas: a de agentes (*Web services*) e a de interoperabilidade.

O agente baseado em informações provenientes da camada de interoperabilidade realiza o mapeamento do formato e as estrutura das informações geradas pelo aplicativo de origem para o formato e estrutura exigidos pelo aplicativo de destino, de maneira que uma aplicação não precisa conhecer detalhes de implementação da outra, uma vez que tais detalhes estão presentes na ontologia definida pela camada de interoperabilidade.

Sendo assim, o primeiro passo na construção da interface de integração de aplicativos consiste em especificar as propriedades e interfaces de cada aplicativo, sendo que alguns dos aplicativos mais utilizados como mecanismos de comunicação em ambientes de ensino à distância já foram definidos nesta camada. Entretanto, prevendo a inserção de novos aplicativos, desenvolveu-se um editor de *schemas* (*InteropEdit*), o qual permite definir a estrutura e os metadados semânticos que "declaram" o significado, as funções e os requisitos de processamento de cada *software*. Para isso, o editor de *schemas* cria um documento *XML Schema DataType*(XSD) que atende ao *World Wide Web Consortium* (W3C).

Segundo a Microsoft (DevBtSol, 2004), um *schema* provê consistência semântica e interoperabilidade, pois se trata de uma especificação que define formalmente uma ampla série de primitivas de tipo de dados e componentes estruturais para criar documentos XML, ou seja, ele atua como um dicionário de elementos abstratos, entidades de atributos e regras organizacionais. Conseqëntemente, criar documentos XML que correspondam ao dicionário de um *schema* tem uma vantagem significativa: o sentido, a função e o uso do conteúdo de um documento são compreensíveis para, e operável por, qualquer aplicativo habilitado pela XML, possibilitando ainda que *schemas* sirvam inclusive de base para protocolos de serviços da *Web*.

Os protocolos de serviços da *web*, o Protocolo de Acesso a Objeto Simples (*Simple Object Access Protocol* - SOAP), e a Linguagem de Definição dos Serviços da Web (*Web Services Definition Language* - WSDL) provêem os recursos e as facilidades para geração de mensagens necessárias para vincular e executar uma funcionalidade programática em qualquer parte, ou em qualquer plataforma, sem a necessidade de um código personalizado (DevBtSol 2004). Por sua vez, cada classe de aplicativo móvel possui complexidades únicas e especializadas, no entanto, o fato de desenvolver o agente como um serviço da *web*, possibilita o intercâmbio de dados entre aplicações independente do nível de abstração e encapsulamento usados por cada uma, pois a claridade e a consistência para esta integração estão presentes na ontologia especificada pela camada de interoperabilidade.

Além da integração de aplicativos em ambientes de execução móvel, este trabalho buscou simular a formação de comunidades sociais em um ambiente virtual. Segundo Nader (Nader, 2002), a constituição física do ser humano revela que ele foi programado para conviver e se completar com outro ser de sua espécie. Assim, o surgimento do pequeno grupo, formado não apenas pelo interesse material, mas também pelos sentimentos de afeto, tende a propagar-se em cadeia, com a formação de outros pequenos núcleos, até se chegar à constituição de um grande grupo social. Ou seja, a comunidade significa algo que excede a mera comunidade local, abraçando todas as formas de relação caracterizadas por um alto grau de intimidade pessoal, profundidade emocional, compromisso moral, coesão social e continuidade no tempo (Batista, 1976).

Desta maneira, as comunidades sociais são formadas por núcleos ou grupos de indivíduos que dividem interesses em comum como religião, música, educação, tecnologia entre outras. Essas

comunidades funcionam conforme as características de seus membros e participantes, os quais são: o local de encontro, a qualidade do atendimento, os gostos ou preferências em comuns (Nyíri, 2002).

A formação de comunidades educacionais se dá, por sua vez, pelo interesse de seus membros em discutir e compartilhar conhecimentos acadêmicos com seus colegas de profissão e colaboradores (Nader, 2002), e como a Internet dispõe de várias ferramentas clássicas e ambientes de ensino virtual que permitem a comunicação e colaboração entre usuários -sendo estes já utilizados em larga escala - a formação das comunidades virtuais de ensino é definida em sua maioria pela oferta de um curso ou oficina, ou projetos e desafios, sendo este o ponto de entrada para a participação de uma comunidade. Além disso, uma comunidade virtual de ensino é fortemente baseada em uma estrutura rígida e não leva em consideração as preferências e recursos de cada indivíduo, contrariando a forma como as comunidades sociais são desenvolvidas (Scopel, 2004).

No entanto, o que geralmente acontece é que, ao ingressar em uma comunidade de EAD de um determinado dispositivo educacional, o indivíduo participante desta comunidade obriga-se a utilizar os recursos oferecidos pelo ambiente, e, portanto, a formação de uma comunidade não é estabelecida de forma espontânea. Ou seja, como o conteúdo ou um curso são apenas um dos itens que despertam interesse nos alunos, a forma como a comunicação e interação são feitas deve também levar em consideração a experiência, a preferência por certo dispositivo de *software*, e a capacidade do equipamento disponível. Pois no momento em que pessoas se unem pelo uso de determinados dispositivos de *software*, estabelece-se um tipo diferente de grupo ou comunidade, o que pode ser facilmente observado em dispositivos como ICQ, MSN Messenger, Orkut, dentre outros (Scopel, 2004).

1.1 Motivação

Kist (Kist, 2002) salienta que as pesquisas e os trabalhos realizados na área da educação à distância ploriferam em grande velocidade, partindo de simples ferramentas de comunicação, avaliação e suporte para ambientes integradores, que disponibilizam a partir de um único lugar, diversos recursos e ferramentas para atender as principais necessidades decorrentes das atividades de professores e alunos. Para Santanchè (Santanchè, 2002), a integração proporcionada pela Internet causa uma mudança significativa na postura dos usuários frente ao *software* educacional, e o problema deixa de ser "como produzir o *software* que preciso", para ser "onde encontrar e como usar o *software* que necessito".

Embora sejam inúmeros os *softwares* educacionais existentes na Internet, a maioria aborda diversos aspectos do mesmo problema, ou o mesmo problema sobre perspectivas diferentes. Assim, segundo Santanchè (Santanchè, 2002), ao buscar um produto que atenda à sua necessidade, o usuário se defronta freqüentemente com uma coleção de peças de *software*, que são incompatíveis e não dispõem de mecanismos de integração.

Para Downes (Downes, 1998), o modelo predominante para o projeto de cursos irá se assemelhar à arquitetura dos computadores modernos. Haverá um *backbone* (espinha dorsal), análogo à placa mãe do computador, que estabelece a estrutura básica do curso. No *backbone* serão conectados vários módulos de estudo, ferramentas de comunicação, e sistemas de informação do estudante. Segundo Downes (Santanchè, 2002), ao contrário da estrutura predominante dos sistemas para cursos *on-line*, no futuro haverá fabricantes especializados na construção de módulos com tarefas

específicas, tal como *e-mail*, *chat*, etc. Tais módulos devem ser passíveis de integração, beneficiando o usuário com uma combinação das melhores soluções em cada área, permitindo uma liberdade de escolha ao módulo que mais lhe aprouver, para o desenvolvimento de uma tarefa.

Freitas (Freitas, 2003) relata que a combinação de um ambiente educacional com a mobilidade oferecida por *Personal Digital Assistants* (PDAs), *Smarthphones*, e equipamentos do gênero, formando um ambiente de aprendizado móvel, aumenta o processo de cooperação e interação entre os seus usuários. Sendo assim, o potencial da Internet móvel permite criar facilidades de acesso aos seus recursos, de qualquer lugar e a qualquer momento.

Os itens apresentados acima motivam o desenvolvimento deste trabalho, o qual pretende utilizar técnicas computacionais e de engenharia de *software*, tais como *Web services, Design Patterns,* ontologias e tecnologias de computação móvel; a fim de criar uma arquitetura que possibilite a integração de diversos recursos educacionais, permitindo ao usuário definir de forma modular e integrada os recursos que lhe são convenientes. Tal arquitetura deve proporcionar um ambiente educacional móvel, o qual poderá ser acessado através de PDAs, em qualquer lugar que possua uma conexão à Internet.

1.2 Problema

A cada ano, cresce em todo o mundo a legião de *softwares* educacionais. Alguns grupos trabalham em versões para acesso a partir de dispositivos móveis, entretanto, tais ambientes sofrem com a falta de mecanismos de interação entre seus recursos. Normalmente este processo exige a modificação das aplicações de forma a prover esta interligação. Esta modificação, quando possível, implica a alteração do código fonte das aplicações, originando problemas sempre que uma interação com outras aplicações for necessária (Silva, 2005).

Somado ao fato de não proverem integração entre aplicações, estes ambientes de ensino à distância baseiam-se na formação de comunidades de ensino apenas pelo interesse do aluno em realizar ou consumir algum material educacional, ignorando suas preferências pessoais, bem como as limitações computacionais do equipamento de *hardware*. Situação esta que causa muitas vezes o desinteresse em permanecer no ambiente de ensino, uma vez que o mesmo é forçado a utilizar apenas os recursos que este oferece.

Sendo assim, o ponto crítico no desenvolvimento deste trabalho está relacionado ao desenvolvimento de uma camada de interoperabilidade que possibilite a integração de *softwares* em um ambiente móvel para ensino à distância por meio de um agente de *software*.

1.3 Questão de pesquisa

A partir do problema acima descrito, a questão central deste trabalho é: como implementar um agente de *software* e uma camada de interoperabilidade que forneça os recursos necessários para integração de aplicações no domínio do ensino à distância, desenvolvidos nas mais diversas linguagens de programação e suportadas pelos sistemas operacionais instalados em dispositivos móveis, como PDAs e *handhelds*.

1.4 Objetivos

O objetivo geral deste trabalho foi modelar e desenvolver uma arquitetura de *software* que permitisse o desenvolvimento de um ambiente de ensino à distância para dispositivos móveis, e que possibilitasse a integração das mais diversas ferramentas de forma independente da plataforma de desenvolvimento.

Visando alcançar este objetivo, definiram-se os seguintes passos:

- realizar um estudo sobre as tecnologias necessárias para o desenvolvimento de *Web services* e sobre *Design Patterns*;
- realizar um estudo sobre ontologias e agentes de *software*;
- modelar os recursos a serem desenvolvidos;
- implementar a camada de interoperabilidade entre os dispositivos de *software*, utilizando tecnologias baseadas em XML e ontologias;
- formalizar a arquitetura e interoperabilidade dos serviços através do desenvolvimento de dois dispositivos de *software* que possam fazer parte de um ambiente educacional à distância;
- tornar os serviços públicos e disponíveis para quaisquer outras aplicações.

1.5 Organização do volume

Este volume está organizado em 8 capítulos, sendo que o capítulo 2 apresenta o contexto do trabalho, no qual são revisados conceitos sobre mobilidade, componentes e integração, agentes de *software* e ontologias. No capítulo 3, é apresentada a revisão bibliográfica realizada sobre as tecnologias *web services* e *Design Patterns*.

O capítulo 4 apresenta os trabalhos relacionados que, de alguma maneira, contribuíram para a realização desta dissertação de mestrado. No capítulo 5, é apresentado o protótipo da arquitetura que possibilita a integração entre aplicações desenvolvidas para ambientes de *M-Learning*, a WSMEL. No capítulo 6, é apresentado um estudo de caso para validação e verificação da arquitetura, onde foram executados testes funcionais e de desempenho. Finalmente, no capítulo 7, são apresentadas as conclusões e trabalhos futuros referentes a esta dissertação, e no capítulo 8, as referências bibliográficas.

CAPÍTULO 2

Conceitos fundamentais

A pesquisa desenvolvida para esta dissertação de mestrado engloba uma série de áreas ligadas diretamente à Ciência da Computação, obrigando um estudo sobre os assuntos que possuem uma relação intrínseca com o desenvolvimento do protótipo proposto. O objetivo deste capítulo é dar ao leitor uma visão geral acerca dos conceitos utilizados no desenvolvimento desta pesquisa. São abordados itens relacionados a mobilidade, componentes, agentes de software e ontologias.

2.1 Mobilidade

Mobilidade é o termo utilizado para identificar dispositivos que podem ser operados à distancia ou sem fio, os quais podem variar desde um simples *pager*, até os mais modernos *handhelds* (Netto, 2005). O termo "Computação móvel" consiste em um paradigma computacional que tem como objetivo prover ao usuário acesso permanente a uma rede fixa ou móvel independente de sua posição física, ou seja, é a capacidade de acessar informações em qualquer lugar e a qualquer momento (Marques, 2004).

A computação móvel está se tornando uma área madura e parece destinada a se tornar o paradigma computacional dominante no futuro. Dispositivos móveis, também chamados genéricamente de *handhelds*, estão aparecendo de diversas formas, como por exemplo: PDAs, telefones celulares, *smarthphones* e vários outros tipos de dispositivos. O mercado desses dispositivos está crescendo e eles estão sendo usados em aplicações que envolvem negócios, indústria, escolas, hospitais, lazer, etc (Marques, 2004).

Entretanto, tais dispositivos ainda sofrem com algumas limitações que devem ser consideradas durante o desenvolvimento de aplicações (Freitas, 2003):

- tamanho pequeno da tela, que pode ser bastante limitada, necessitando inovações no projeto da interface gráfica do usuário;
- desempenho limitado, em termos de processador, memória, espaço de armazenamento e tempo de vida da bateria;
- conectividade lenta;
- variedade de sistemas operacionais.

2.2 Componentes de *software*

O desenvolvimento de *software* baseado em componentes emergiu no final da década de 90 como uma abordagem baseada no reuso para desenvolvimento de sistemas, cuja a motivação foi a frustração de que o desenvolvimento orientado a objetos não tinha conduzido a um extensivo reuso, como originalmente foi sugerido. Isso porque as classes de objetos individuais eram muito detalhadas e específicas e tinham que estar associadas a uma aplicação em tempo de compilação ou quando o sistema estivesse conectado, e ainda porque o conhecimento detalhado das classes era necessário para sua utilização, e isso, geralmente, significava que o código fonte precisava estar disponível, apresentando problemas difíceis para a comercialização de componentes (Sommerville, 2004).

No entanto, os componentes são mais abstratos do que as classes de objetos e podem ser considerados provedores de serviços *stand-alone*, e, portanto, quando um sistema precisa de algum serviço, ele chama um componente para fornece-lo, sem se preocupar a respeito de onde esse componente está sendo executado, ou da linguagem de programação utilizada para desenvolve-lo. Visualizar um componente como um provedor de serviços enfatiza duas importantes características de um componente reutilizável (Sommerville, 2004):

- o componente é uma entidade executável independente. O código-fonte não está disponível, de modo que o componente não é compilado com outros componentes do sistema;
- os componentes publicam sua interface e todas as interações são feitas por meio dessa interface. A interface do componente é expressa em termos de operações parametrizadas e seu estado interno nunca é exposto.

Uma interface denominada *provides* define os serviços oferecidos pelo componente, enquanto que uma interface denominada *requires* especifica quais serviços devem estar disponíveis a partir do sistema que está utilizando o componente. Conseqüentemente, se estas interfaces não forem fornecidas, o componente não funcionará.

Para Santanchè (Santanchè, 2002), no desenvolvimento de *software* educacional, observa-se que houve uma evolução nas ferramentas genéricas para programação, que combinam a tecnologia de componentes com interfaces visuais de construção. Por outro lado, no domínio específico da educação, a prática na construção de componentes educacionais ainda é inexpressiva.

2.3 Integração de dispositivos de *software*

O termo integração para as Ciências da Computação consiste em um processo que combina componentes de *software* e de *hardware* em um sistema único. A integração de *software* consiste na união de uma série de componentes de *software* usando uma infra-estrutura que provê uma forma de desenvolver um sistema sem modificar os componentes originais (Arhippainem, 2003). Segundo Santanchè (Santanchè, 2002), a integração de *software* pode acontecer em diversos níveis, de formas diferentes e com propósitos diferentes, as quais neste trabalho dizem respeito ao intercâmbio de dados e à integração de procedimentos.

O intercâmbio de dados pode ocorrer tanto de maneira assíncrona quanto síncrona, sendo que, intercambiar os dados de maneira assíncrona consiste na troca de arquivos, cujo formato pode ser

compreendido por dois programas de computador diferentes. É muito comum, por exemplo, que dois processadores de textos diferentes dêem suporte de leitura e edição do mesmo formato de arquivo. Já no intercâmbio de dados de maneira síncrona, não apenas dois programas de computador devem ser capazes de compreender um formato comum para representação de dados, como precisam de um protocolo para realizar a troca de dados dinamicamente.

O ato de integrar procedimentos de *software* consiste não apenas em trocar dados com outro módulo de *software*, mas em utilizar serviços como parte de sua tarefa, ou então trabalhar em colaboração em uma tarefa que é resultante da interação de ambos. Isto significa que os dois módulos precisam trabalhar em conjunto, como se pertencessem ao mesmo programa. A integração de procedimentos pode ser fruto de uma necessidade coletiva mais ampla de construir produtos em colaboração, onde integrar é a condição necessária para a soma dos esforços e a combinação de resultados. Ela exige um deslocamento de uma ótica de construção de aplicações monolíticas, para um enfoque baseado na produção de componentes de *software*, passíveis de combinação em diferentes configurações e, por este motivo, ideais para colaboração.

A Figura 2.1 demonstra um produto de *software* constituído de diversos outros *softwares* / componentes, onde independente do tipo de plataforma em que foram desenvolvidos, os mesmo podem atuar de maneira integrada, seja essa, uma integração de dados ou de procedimentos.

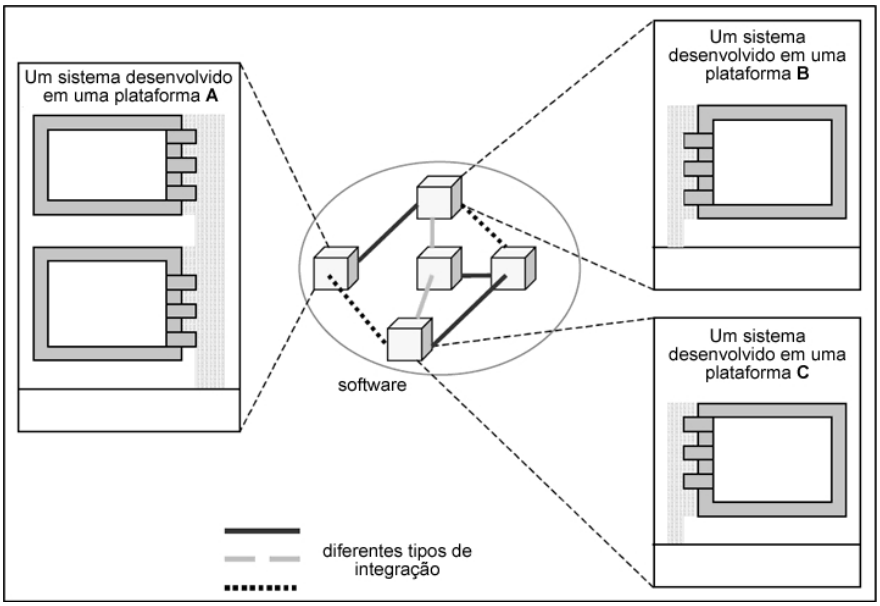


Figura 2.1: Integração de *software* baseada em componentes. Adaptada de (Arhippainem, 2003)

2.4 Agentes de *software*

A idéia de agente teve origem com John McCarthy na metade da década de 50, e o termo foi criado por Oliver G. Selfridge poucos anos antes, quando ambos estavam no *Massachusetts Institute of Technology* (MIT). Eles tinham em mente um sistema que, quando definido um objetivo, pudesse resolver os detalhes das operações computacionais apropriadas e pudesse pedir e receber conselhos, oferecidos em termos humanos, quando estivesse em dificuldade. Portanto, um agente poderia ser

um "*softrobot*" vivendo e realizando suas tarefas dentro do mundo dos computadores (Bradshaw, 1997).

Para Daniel Serain (Serain, 2002) o conceito de agentes de *software* propõe uma extensão do modelo orientado a objeto. Ou seja, agentes são sistemas integrados que (1) internamente incorporam um maior número de capacidades de diferentes áreas (inteligência artificial, banco de dados, linguagens de programação, teoria da computação) e (2) externamente estabelecem comunicação com outras aplicações através de uma linguagem independente de características específicas de estrutura de cada aplicação.

Lucena (Lucena, 2002) os considera como sendo programas para os quais é possível delegar aspectos de uma tarefa, diferenciando-se do *software* "tradicional" por serem personalizados, executarem continuamente e serem semi-autônomos.

Embora este termo venha sendo amplamente usado tanto na indústria como no meio acadêmico, ainda não existe consenso sobre a sua definição. Entretanto, se assume que as características que todo agente de *software* deve ter são (Miller, 1998):

- **reatividade:** agentes percebem seu ambiente (que pode ser o mundo físico, um usuário via uma interface gráfica, uma coleção de outros Agentes, a Internet, ou talvez todos esses ambientes combinados) e respondem de maneira adequada às mudanças que nele ocorrem;
- **autonomia:** agentes operam sem a intervenção direta de humanos ou outros, e têm algum tipo de controle sobre suas ações e estado interno;
- **habilidade social:** agentes interagem com outros Agentes (e possivelmente humanos) através de algum tipo de linguagem para comunicação;
- **pró-atividade:** agentes não agem simplesmente em resposta ao seu ambiente, eles são capazes de exibir um comportamento direcionado a objetivos, de tomar a iniciativa.

2.4.1 Por que agentes de *software*?

Dentre as diversas funcionalidades que agentes de *software* proporcionam podemos destacar a busca pela diminuição da complexidade da computação distribuída e superação da limitação das interfaces dos usuários (Bradshaw, 1997).

Assim, ao mesmo tempo em que os sistemas distribuídos deixam de ser o futuro da computação para tornar-se o presente, é necessário que estes estabeleçam certo nível de interoperabilidade. Esta interoperabilidade requer conhecimento das capacidades de cada sistema, alocação de recursos, execução, monitoramento e possível intervenção entre eles (Serain, 2002). A Figura 2.2 expressa um agente único como sendo um gerenciador global de recursos, sem este realizar qualquer interação com outros agentes, representando a comunicação do tipo agente-aplicação.

Por sua vez, a Figura 2.3 demonstra um sistema em que diversos agentes formam uma sociedade em um ambiente específico, cooperando entre si. Tais sistemas são conhecidos como *Multiagent Systems* (MAS) e são compostos por agentes que se relacionam com os demais de forma coordenada.

Serain (Serain, 2002) destaca que agentes de *software* são particularmente vantajosos no desenvolvimento de sistemas distribuídos, visto que oferecem flexibilidade em projetos de integração entre aplicações e a reusabilidade de componentes, pois podem ser implementados em diferentes

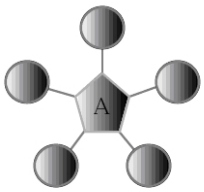


Figura 2.2: Cooperação entre sistemas através de um único agente (Bradshaw, 1997)

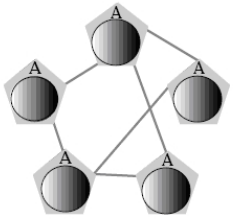


Figura 2.3: Cooperação entre sistemas através de agentes distribuídos (Bradshaw, 1997)

linguagens de programação e serem executados em diferentes plataformas. A interoperabilidade entre aplicações só é possível porque os agentes compartilham uma linguagem e um vocabulário comum, que contém informações apropriadas de aplicações a uma área comum, o que significa que estas são descritas através de uma ontologia compartilhada.

2.5 Ontologias

Ontologia é um termo inicialmente utilizado na filosofia, o qual está intimamente ligado ao estudo dos seres, das coisas enquanto seres, dos objetos enquanto coisas, denominados - os seres e as coisas -, genericamente, como entidades. Na Ciência da Computação, ontologia é um termo utilizado desde o início da década de 90 para a representação computacional de conhecimento em áreas como engenharia de conhecimento e processamento de linguagem natural. Em resumo, uma ontologia é definida como "a especificação explícita de uma conceitualização", sendo que uma ontologia comum é uma formalização compartilhada de certo domínio de aplicação. Assim, por exemplo, uma formalização de conceitos sobre informações em ambientes de ensino à distância poderia permitir que diversos aplicativos deste domínio compartilhassem um vocabulário comum sobre o assunto (Chandrasekaran, 1999).

2.5.1 Por que desenvolver uma ontologia?

Há alguns anos, o desenvolvimento de uma ontologia tem focado o seu uso na *web*,e, em conseqüência, o W3C desenvolveu linguagens para codificação de conhecimento de maneira a torná-lo compreensível por agentes de *software* na busca de informação (Novello, 2005).

Muitas áreas têm desenvolvido ontologias padronizadas de domínios específicos objetivando o compartilhamento de informação. A medicina, por exemplo, produziu um vocabulário padronizado e estruturado, e uma rede semântica de sistemas de uma linguagem médica unificada. Ontologias têm

sido desenvolvidas com inúmeras finalidades, um exemplo, é a *United Nations Development Program* e da *Pun e Bradstreet*, que combinaram seus esforços para desenvolver a ontologia denominada UNSPC, que provê uma terminologia para produtos e serviços (Noy, 2004).

Algumas razões para desenvolver-se uma ontologia incluem:

- o compartilhamento e o entendimento de uma estrutura de informação por pessoas e agentes de *software*. Por exemplo, suponha diversos *web sites* médicos distintos, contendo informações médicas ou provendo alguns serviços. Se estes *sites* compartilharem e publicarem sobre a mesma ontologia de termos, estes agentes poderão extrair e agregar informação de um *site* para outro;
- habilitar o reuso de um domínio de conhecimento.

2.5.2 Detalhando uma ontologia

Noy (Noy, 2004) define uma ontologia como uma descrição formal de conceitos em um domínio de discurso, sendo composta por:

- **classes** (geralmente chamadas de conceitos): descrevem conceitos de um domínio;
- ***slots*** (geralmente chamados de regras ou propriedades): descrevem as propriedades das classes e suas instâncias;
- **restrições:** definem propriedades específicas de um *slot*.

Na prática, o desenvolvimento de uma ontologia inclui:

- definir as classes da ontologia;
- organizar as classes em uma taxonomia hierárquica (subclasse - superclasse);
- definir os *slots* e descrever quais os valores reservados para estes;
- definir valores para os *slots* das instancias.

As ontologias, geradas por um sistema, são resultados da integração de processos representados através de uma árvore, em que cada nó representa um conceito da ontologia, e cada relacionamento entre os nós pode ser um relacionamento taxonômico (Novello, 2005).

Segundo Novello (Novello, 2005), taxonomias representam a maneira como se organizam classes e subclasses dentro de uma ontologia. Uma taxonomia é um sistema de classificação que agrupa e organiza o conhecimento num domínio, usando relações de generalização e especialização através de herança simples ou múltipla.

2.5.3 Metodologia de desenvolvimento

Uma ontologia consiste no modelo da realidade de um domínio, portanto os conceitos nesta ontologia precisam refletir esta realidade. Noy (Noy, 2004) enumera alguns passos que devem ser seguidos no desenvolvimento de uma ontologia:

1. **Determinar o escopo:** esta etapa consiste em responder as seguintes questões: Qual o domínio que esta ontologia irá cobrir? Para que ela será utilizada? A que tipos de perguntas as informações contidas nesta ontologia irão prover respostas? Quem usará e quem manterá esta ontologia? As respostas para estas questões podem mudar durante o processo de desenvolvimento da ontologia, mas, com o tempo, auxiliam na definição do escopo do modelo.

2. **Considerar o reuso de ontologias existentes:** reusar uma ontologia existente pode ser um requisito se o sistema precisa interagir com outras aplicações que já possuem uma ontologia ou um vocabulário controlado. Para isso, muitas ontologias estão disponíveis no formato eletrônico e podem ser importadas para o ambiente da ontologia que está sendo desenvolvido, sendo que já existem algumas bibliotecas de ontologias reutilizáveis na *Web* e na literatura, como exemplo, pode-se citar a biblioteca de ontologias Ontolingua [1] ou a *DARPA Agent Markup Language* (DAML).

3. **Enumerar os termos importantes desta ontologia:** inicialmente, cria-se uma lista de termos sem se preocupar com a sobreposição de conceitos, relações entre seus termos, ou propriedades que estes conceitos podem ter. Após este levantamento, torna-se necessário desenvolver uma hierarquia de classes e definir as propriedades dos conceitos (*slots*). Essas etapas são mais difíceis que a primeira e consistem na fase mais importante no processo de desenvolvimento de uma ontologia.

4. **Definir as classe e sua hierarquia:** para esta etapa, combinam-se os processos de desenvolvimento *top-down* (inicia com a definição dos conceitos mais importantes no domínio e subseqüente especialização) e *bottom-up* (definem-se classes mais específicas e depois se agrupam em conceitos gerais).

5. **Definir as propriedades das classes (*slots*):** as classes sozinhas não provêem nenhuma informação que forneça respostas levantadas durante o primeiro passo. Uma vez definida algumas classes, é preciso definir a estrutura interna dos conceitos.

6. **Definir as restrições dos *slots*:** um *slot* pode ter diferentes restrições descrevendo o tipo de valor, o número de valores (cardinalidade), e outras características que os *slots* podem ter. Por exemplo, o valor aceito para um *slot* "nome" é uma *string*. As restrições mais comuns descrevem:

   (a) cardinalidade: define quantos valores um *slot* pode ter;

   (b) tipo de valor: define que tipo de valor o *slot* aceita, podendo ser: *string, number, boolean, enumerated* ou *instance*.

7. **Criar instâncias:** o último passo consiste na criação de instâncias individuais das classes na hierarquia. Definir uma instância individual de uma classe requer:

   (a) escolher a classe;

   (b) definir uma instância individual desta classe;

[1] (www.ksl.stanford.edu/software/ontolingua)

(c) definir os tipos de valores para os *slots*.

CAPÍTULO 3

Tecnologias de engenharia de *software*

O reuso de sistemas de aplicações é um dos objetivos principais de grande parte de desenvolvedores de software, visto que, custos gerais de desenvolvimento podem ser reduzidos, menos componentes de software têm que ser especificados, projetados, implementados e validados. O objetivo deste capítulo é apresentar ao leitor tecnologias ligadas a Engenharia de Software que buscam a reutilização de componentes e a interoperabilidade entre sistemas .

3.1 Design Patterns

Ao questionar-se por que a tecnologia de orientação a objetos é mais recomendada que as suas concorrentes, grande parte dos engenheiros de *software* responderiam "reuso", pois na realidade, a reutilização de componentes de *software* está diretamente relacionada a duas questões que afetam os projetistas e os usuários dos produtos: qualidade e produtividade (Metsker 2004).

No entanto é difícil construir um produto totalmente reutilizável logo na primeira tentativa. Isso significa que a experiência dos projetistas é fundamental para desenvolver bons componentes de *software*, e, além disso, profissionais experientes costumam reutilizar soluções que já funcionaram no passado, ao invés de resolver cada problema partindo do zero (Gamma, 1995).

Design Patterns (Padrões de Projeto) são soluções genéricas para problemas recorrentes em Engenharia de *Software* (Rheihemer 2002). Conforme Christopher Alexander (Gamma 1995), "*cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução inúmeras vezes, sem nunca fazê-la novamente*". Ou seja, cada padrão identifica classes e instâncias participantes com seus papéis, colaborações e a distribuição de responsabilidades, sendo que estes elementos podem ser customizados para resolver um problema num contexto particular (Gamma, 1995).

Os *Design Patterns* possuem quatro elementos essenciais:

- **nome**: identifica o problema e a solução adotada em uma ou duas palavras;
- **problema**: informa em que ocasiões o padrão pode ser utilizado;
- **solução**: descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaboração. A solução não descreve uma implementação articular, mas, ao invés

disso, fornece uma descrição abstrata de um problema e como uma combinação de classes e objetos o resolve;

- **conseqüências**:informam os resultados e desafios na utilização do padrão a fim de esclarecer os custos e benefícios de sua aplicação.

Devido à granularidade e ao seu nível de abstração, padrões de projeto são agrupados em famílias. Tal agrupamento é feito a partir de dois critérios, o primeiro diz respeito à finalidade (que reflete o que o padrão faz), que podem ser de criação, estrutural ou comportamental, e o segundo refere-se ao escopo, e especifica se o padrão se aplica primeiramente a classes ou a objeto. Desta forma, os padrões podem ser tabelados conforme exibido na Figura 3.1 e descritos posteriormente (Gamma 1995).

		Propósito		
		1 - Criação	2 -Estrutura	3 - Comportamento
Escopo	Classe	*Factory Method*	*Class Adapter*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Object Adapter* *Bridge* *Composite* *Decoretor* *Façade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

Figura 3.1: *Design Patterns* (Gamma, 1995)

Os Padrões de criação abstraem o processo de instanciação, e ajudam a tornar o sistema independente de como os objetos são criados, compostos e representados. Um padrão de criação de classes usa a herança para variar a classe que é instanciada, enquanto um padrão de criação de objetos delega a instanciação para outro objeto. Os padrões de criação dão muita flexibilidade quanto ao que é criado, quem cria, como e quando é criado. Por sua vez, os padrões estruturais preocupam-se com a forma com que as classes e objetos são compostos para formar estruturas maiores, e os Padrões comportamentais preocupam-se com algoritmos e a atribuição de responsabilidades entre objetos, sendo que eles não descrevem apenas padrões de objetos ou classes, mas também os padrões de comunicação entre eles. Estes padrões caracterizam fluxos de controle difíceis de seguir em tempo de execução; eles afastam o foco do fluxo de controle, para que seja possível concentrar-se somente na maneira como os objetos são interconectados.

Os padrões citados acima estão disponíveis para consulta em livros e na Internet, em fontes de pesquisa como (Gamma, 1995) e (Duell, 1997). O catálogo de padrões de projetos em (Gamma, 1995) possui 23 padrões, os quais são apresentados a seguir:

1. **Padrões de criação**
   Classe:

   - ***Factory Method:*** define uma interface para criar um objeto, mas deixa as subclasses decidirem a classe a ser instanciada.

   Objeto:

   - ***Abstract Factory*:** fornece uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas;
   - ***Builder*:** separa a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar diferentes representações;
   - ***Prototype*:** especifica os tipos de objeto a serem criados usando uma instância prototípica e permite criar novos objetos copiando este protótipo;
   - ***Singleton*:** garante que uma classe tenha uma única instância e provê um ponto global de acesso a ela.

2. **Padrões estruturais**
   Classe:

   - ***Adapter:***converte a interface de uma classe em outra interface esperada.

   Objeto:

   - ***Adapter (object)*:** converte a interface de um objeto em outra interface esperada;
   - ***Bridge*:** separa uma abstração da sua implementação de modo que as duas possam variar independentemente;
   - ***Composite*:** compõe objetos em estrutura de árvore para representar hierarquias do tipo parte-todo;
   - ***Decorator*:** atribui responsabilidades adicionais a um objeto dinamicamente;
   - ***Façade*:** fornece uma interface unificada para um conjunto de interfaces;
   - ***Flyweight*:** usa compartilhamento para suportar grande quantidade de objetos;
   - ***Proxy*:** provê um objeto procurador, ou um marcador de outro objeto, para controlar o acesso a ele.

3. **Padrões de comportamento**
   Classe:

   - ***Interpreter:*** dada uma linguagem, define uma representação para a sua gramática com um interpretador que usa a representação para interpretar sentenças nesta linguagem;
   - ***Template:*** define o esqueleto de um algoritmo numa operação, deixando que subclasses completem algumas das etapas. Esse padrão permite que subclasses redefinam determinadas etapas de um algoritmo sem alterar a estrutura do mesmo.

Objeto:

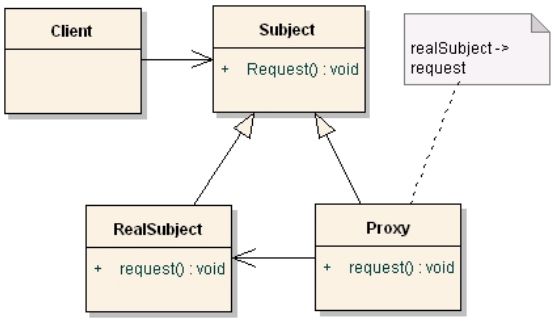
- ***Chain of Responsibility*:** evita o acoplamento do remetente de uma solicitação ao seu destinatário, dando a mais de um objeto a chance de tratar a solicitação;
- ***Command*:** encapsula uma solicitação como um objeto agregado sem expor sua representação subjacente;
- ***Iterator*:** fornece uma maneira de acessar seqüencialmente os elementos de um objeto agregado sem expor sua representação subjacente;
- ***Mediator*:** define um objeto que encapsule a forma com a qual um conjunto de objetos interage. Promove o acoplamento fraco evitando que objetos referenciem uns aos outros explicitamente;
- ***Memento*:** sem violar o princípio do encapsulamento, captura e externaliza o estado interno de um objeto de forma a poder restaurar o objeto mais tarde.
- ***Observer*:** define uma dependência um-para-muitos entre objetos de forma a avisar e atualizar vários objetos quando o estado de um objeto muda;
- ***State*:** permite que um objeto altere o seu comportamento quando o seu estado interno muda. O objeto estará aparentemente mudando de classe com a mudança de estado;
- ***Strategy*:** define uma família de algoritmos, encapsula cada um, e deixa-os intercambiáveis;
- ***Visitor*:** representa uma operação a ser realizada nos elementos de uma estrutura de objetos. O padrão *Visitor* permite que se defina uma nova operação sem alterar as classes dos elementos nos quais a operação age.

3.1.1 Descrição

Souza (Souza, 2004) relata que as notações gráficas existentes para modelagem de projetos, embora importantes e úteis, não são suficientes para descrever os *design patterns*, pois apenas capturam o produto final do projeto como relacionamentos entre classes e objetos. Para reutilizar o projeto se faz necessário registrar também as decisões, alternativas e análises de custos e benefícios que resultaram do mesmo. Além disso, exemplos concretos também são importantes, pois auxiliam a visualizar o padrão sendo utilizado efetivamente. Por isso, sugere-se um formato coerente para a descrição dos padrões em que cada padrão é descrito segundo um modelo, o que resulta em uma uniformidade na estrutura das informações tornando-os mais fáceis de aprender, comparar e utilizar.

Para exemplificar um modelo de descrição de padrões, a tabela 3.1 apresenta um uma descrição parcial do padrão estrutural de objetos *Proxy*, conforme modelo proposto por (Gamma, 1995).

Tabela 3.1: Exemplo da descrição parcial do padrão *Proxy* (Gamma, 1995)

Intenção: Fornece um substituto ou marcador da localização de outro objeto para controlar o acesso ao mesmo.
Também conhecido como: *Surrogate*
Motivação: Uma razão para controlar o acesso a um objeto é adiar o custo integral de sua criação e inicialização até o momento em que realmente necessita-se usá-lo.
Aplicabilidade: o padrão *proxy* é aplicável sempre que há necessidade de uma referência mais versátil, ou sofisticada, do que um simples apontador para um objeto. Algumas situações comuns para o uso incluem: • um *remote Proxy*, que fornece um representante local para um objeto num espaço de endereçamento diferente; • um *virtual Proxy*, que cria objetos sob demanda; • um *protection Proxy*, que controla o acesso para um objeto original; • um *smart reference*, um substituto para um *pointer* que executa ações adicionais quando um objeto é acessado.
Estrutura: 
Participantes: • ***Proxy (Image Proxy)***: – mantém uma referência que permite ao *Proxy* acessar o objeto real (*real subject*). O *Proxy* pode referenciar um *Subject* se as interfaces de *realSubject* e *Subject* forem as mesmas; – Fornece uma interface idêntica ao *subject*, de modo que o *proxy* possa substituir o objeto real (*real subject*). – Controla o acesso ao objeto real e pode ser responsável pela sua criação e exclusão.

- ***Subject***: define uma interface comum para o *RealSubject* e *Proxy*, de maneira que um *Proxy* possa ser usado em qualquer lugar em que um *RealSubject* é esperado.
- ***RealSubject:*** Define o objeto real que o *Proxy* representa.

Colaborações: Dependendo do seu tipo, *Proxy* repassa solicitações para *realSubject* quando apropriado.

Conseqüências: O padrão *Proxy* introduz um nível de referência indireta no acesso a um objeto. A referência indireta adicional tem muitos usos, dependendo do tipo de *Proxy*:

- um *Proxy* remoto pode ocultar o fato de que um objeto reside num espaço de endereçamento diferente;
- um *Proxy* virtual pode executar otimizações, tais como a criação de um objeto sob demanda;
- tanto *proxies* de proteção como *smart references* permitem tarefas adicionais de organização quando um objeto é acessado.

3.2 Web services

Um *Web service* é um componente de *software*, ou uma unidade lógica de aplicação, que se comunica através de tecnologias e padrões de Internet. Esse componente provê dados e serviços para outras aplicações. Esta tecnologia combina os melhores aspectos do desenvolvimento baseado em componentes e a *Web*. Como componentes, representam uma funcionalidade implementada em uma "caixa-preta", que pode ser reutilizada sem a preocupação de como o serviço foi implementado. As aplicações acessam os *Web services* através de protocolos e formatos de dados padrões, como *Hypertext Transference Protocol* (HTTP), XML e SOAP (Dextra, 2003).

Diferentemente dos *web sites* tradicionais, que são projetados para as pessoas interagirem com informações, os *Web services* conectam aplicações diretamente com as outras aplicações, e têm como idéia básica que essa conexão se dê sem que seja necessário efetuar grandes customizações nas próprias aplicações. Além disso, uma das premissas fundamentais é que o padrão usado pelas conexões seja aberto e independente de plataforma tecnológica ou linguagens de programação.

3.2.1 Arquitetura

Oellerman (Oellermann, 2001) compara uma aplicação que utiliza *Web services* como uma corrente, em que cada elo representa uma funcionalidade na aplicação. Um *link* entre elos diferentes representa processos compartilhados via *Web services*, onde ocorre a conexão entre um serviço e um consumidor deste serviço, o que significa que um *Web service* pode ser o consumidor de outro *Web service*. Assim, o primeiro elo da corrente representa a interface da aplicação, de onde requisições a *Web*

services poderão ser realizadas. A representação utilizada por Oellerman não implica em um forte acoplamento entre os serviços, os quais podem ser utilizados em diferentes aplicações.

A arquitetura de comunicação de um *Web service* está definida nas interações de três papéis: provedor, solicitante e um servidor de registro (Oellermann, 2001 - Ferris, 2003 - Newcomer 2002), ilustrados na Figura 3.2.

Figura 3.2: Arquitetura de *Web services*. (Ferris, 2003)

Ferris (Souza, 2004) descreve o provedor de serviços como o responsável por disponibilizar os serviços e armazenar sua descrição, através da WSDL, a qual contém detalhes de interface, operações dos serviços e mensagens de entrada e saída para as operações. Sendo assim, após o recebimento da descrição de um serviço, o provedor publica a descrição do mesmo, em WSDL, em um registro de serviços, ou seja, o solicitante do serviço é uma aplicação que invoca uma interação com o serviço, a qual pode ser um navegador *Web* ou outra aplicação qualquer como um outro *Web service*. Já o servidor de registro é o local onde os provedores publicam os seus serviços, os quais são procurados pelos solicitantes.

3.2.2 Tecnologia

As tecnologias utilizadas em *Web services* permitem que os serviços possam ser disponibilizados na *Web* de forma padronizada (Souza, 2004), as quais, baseadas em XML, são utilizadas para transportar e transformar dados entre aplicações. Desta forma, as tecnologias utilizadas no desenvolvimento de um *Web service* podem ser representadas conforme a figura 3.3.

Assim, XML é a base para desenvolvimento de um *Web service*, pois provê uma linguagem para definição e processamento de dados. Os serviços são invocados e fornecem resultados através da troca de mensagens, empacotadas através do protocolo SOAP, o qual provê um formato de serialização, utilizado para transmitir documentos em uma rede de comunicação e uma convenção para representar interações entre *Remote Procedure Calls* (RPCs) (Newcomer, 2002). Sendo assim, o uso de XML é fundamental para que se garanta a interoperabilidade.

Uma vez que, para promover interoperabilidade entre sistemas heterogêneos é necessário um mecanismo que permita que a estrutura e o tipo de dados possam ser compreendidos pelos *Web services*, a WSDL é utilizada com este objetivo, pois permite que mensagens com a descrição precisa dos serviços possam ser trocadas (Booth, 2003). Já o registro *Universal Distribution Discovery and*

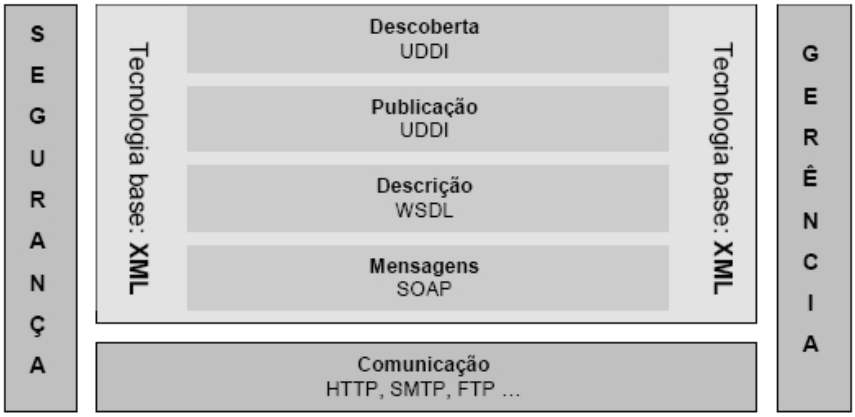


Figura 3.3: Tecnologias no desenvolvimento de um *Web service*. (Souza, 2004)

Integration (UDDI) é utilizado para publicação e descoberta de informações sobre *Web services* (Newcomer, 2002).

A Figura 3.4 demonstra um simples *Web service*, em que o WSDL é obtido do registro UDDI ou outra localização, uma mensagem SOAP é gerada para transmissão a um site remoto, e, depois disso, uma aplicação submete um documento (um arquivo em XML) para um URI[1] correspondente a um *Web service* usando um *schema* em XML, tal como WSDL, que também será utilizado para gerar a saída para o método invocado. Finalmente o computador que envia a requisição usa o SOAP para transformar os dados do formato nativo para um formato de tipo de dados predefinido em um *schema* XML existente em um arquivo WSDL.

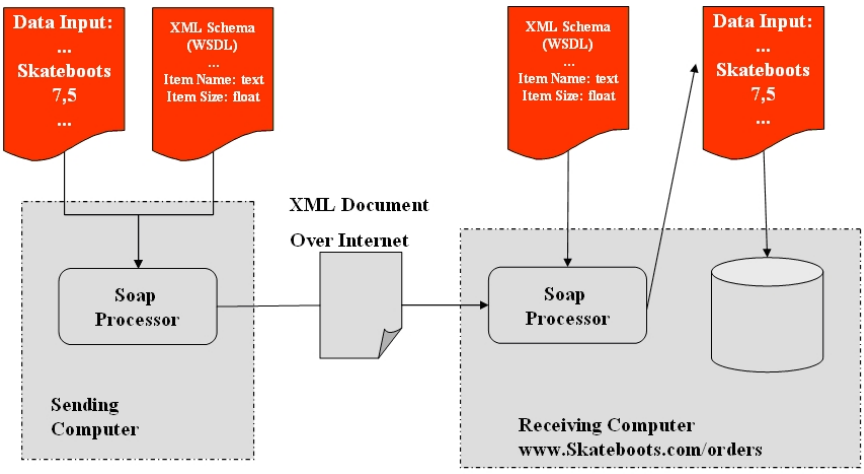


Figura 3.4: Invocação de um *Web service*. Adaptada de (Newcomer, 2002)

As seções a seguir detalham as tecnologias apresentadas neste tópico, não abordando, no entanto, os itens referentes à segurança e à gerência.

[1] O termo URI frequentemente aparece em especificações *Web services* no lugar de URL. Um URI é um termo que inclui URL e URN. Um URN é um nome que não referencia um recurso físico (Newcomer, 2002).

3.2.2.1 *eXtensible Markup Language*

No contexto dos *Web services*, a XML não é apenas utilizada como um formato para a troca de mensagens, mas também como a forma através da qual os serviços são definidos. Como conseqüência, é importante conhecer-lá na forma como ela é utilizada para definir e implementar os *Web services* (Newcomer, 2002). Usando a XML, podem-se definir qualquer número de elementos (*tags*), representados entre os sinais de "<" e "/>", que associam significado as informações. Considere a expressão ilustrada na Figura 3.5

```
<user>
      <userID> marcscopel</userID>
      <userName> Marcelo Scopel </userName>
      <userAdress>Porto Alegre</userAdress>
      <userPhone>99649363</userPhone>
</user>
```

Figura 3.5: Exemplo de arquivo XML

Neste exemplo, a XML não apenas demonstra elementos que descrevem dados, como também uma estrutura que relaciona um grupo de dados, o que torna fácil imaginar a busca por um elemento que satisfaça a certos critérios, tal como endereço (<userAddress>) para um determinado indivíduo. Além disso, esquemas associados a um documento em XML validam os dados separadamente e descrevem outros atributos relacionados a estes dados.

Desta forma, duas partes que troquem dados em XML poderão entender e interpretar os elementos da mesma forma somente se elas compartilharem da mesma definição, ou seja, se as duas partes que compartilharem o mesmo XML também compartilharem do mesmo esquema, elas poderão entender o significado dos elementos entre as *tags* igualmente, que é a forma exata como os *Web services* trabalham (Newcomer, 2002).

A sintaxe de XML usada em *Web services* especifica como os dados são representados, define como os dados são transmitidos, e detalhes de como os serviços são publicados e descobertos. Considere o arquivo em XML da Figura 3.5, em que para validação deste documento, torna-se necessário a utilização de um esquema XML (figura 3.6).

```
01    <xsd:schema xmlus:xsd=http://www.w3.org/2004/xmlSchema>
02          <xsd: element name = "user" type="userType"/>
03          <xsd: complexType name = "userType">
04          <xsd:sequence>
05                <xsd:element name = "userId" type = "xsd:string"/>
06                <xsd:element name = "userName" type = "xsd:string"/>
07                <xsd:element name = "userAdress" type = "xsd:string"/>
08                <xsd:element name = "userPhone" type = "xsd:string"/>
09          </xsd:sequence>
10          </xsd: complexType
11    </xsd:schema>
```

Figura 3.6: *Schema* XML

O exemplo ilustra um esquema que define a validação do arquivo em XML, os tipos de dados correspondentes a cada elemento, e a estrutura do documento para cada registro de usuário. A primeira declaração do esquema (linha 01) referencia o esquema em XML que está em uso através

do <*namespace* www.w3.org/2004/xmlSchema>, mecanismo este que é utilizado constantemente pelo W3C para identificar versões aplicáveis. O esquema também contém os tipos de dados (linhas 05, 06, 07, 08) e a seqüência que estabelece a ordem dos elementos, desta forma sabemos que o elemento ***userID*** precisa preceder o elemento ***userName***. O *parser* XML valida instâncias de documentos de acordo com os nomes de elementos declarados no respectivo esquema, e elementos que não foram declarados podem ser rejeitados.

3.2.2.2 XML *Schemas*

Um *schema* (esquema) é uma especificação formal da gramática utilizada por um documento XML específico. Ou seja, são utilizados para validar os documentos XML, determinando quando estes estão de acordo com a gramática expressa pelo *schema*, permitindo assim, a troca de dados entre aplicações (Coyle, 2002).

Em XML, existem dois tipos principais de esquemas: *Data Type Definitions*(DTD) e *Schemas*, conforme pode ser observado na Figura 3.7, e os quais são descritos a seguir.

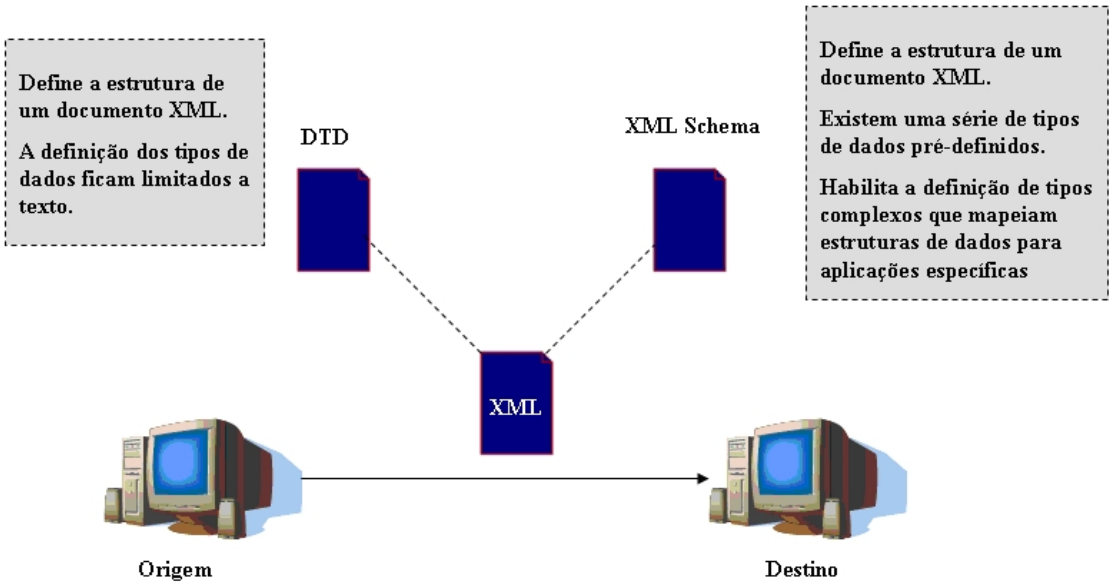


Figura 3.7: *Schemas* que podem ser utilizados na definição de um XML. Adaptada de (Coyle, 2002)

DTDs possuem o seu foco voltado primeiramente à estrutura, permitindo a construção de um projeto de vocabulário através da especificação de elementos e atributos que são apropriados para um conjunto de instâncias de documentos XML, descrevendo os tipos de dados associados a um documento XML. Uma DTD pode ser publicada para consulta e utilizada para a criação de documentos XML. As principais características deste esquema XML são (Coyle, 2002):

- uso de uma sintaxe diferente da XML, isto porque foram primeiramente desenvolvidas no universo do *Standard General Markup Language* (SGML), anterior a XML;
- definição dos elementos e atributos que podem validar o esquema;

- inviabilidade de realizar distinção entre tipos de dados. A DTD é limitada à declaração de que um elemento precisa conter texto, mas ela não consegue controlar qual o tipo de texto, por não conseguir, por exemplo, distinguir entre números e caracteres do alfabeto.

Schemas fornecem mais detalhes sobre os tipos de dados que podem aparecer como parte de um documento XML. Um *schema* define um vocabulário e regras para gerenciamento do conteúdo dos elementos e atributos de um documento XML, isso porque suportam uma grande faixa de tipos de dados e criação de tipos complexos (Coyle, 2002). A Figura 3.8 ilustra os tipos de dados existentes em um XML *Schema* e sua hierarquia.

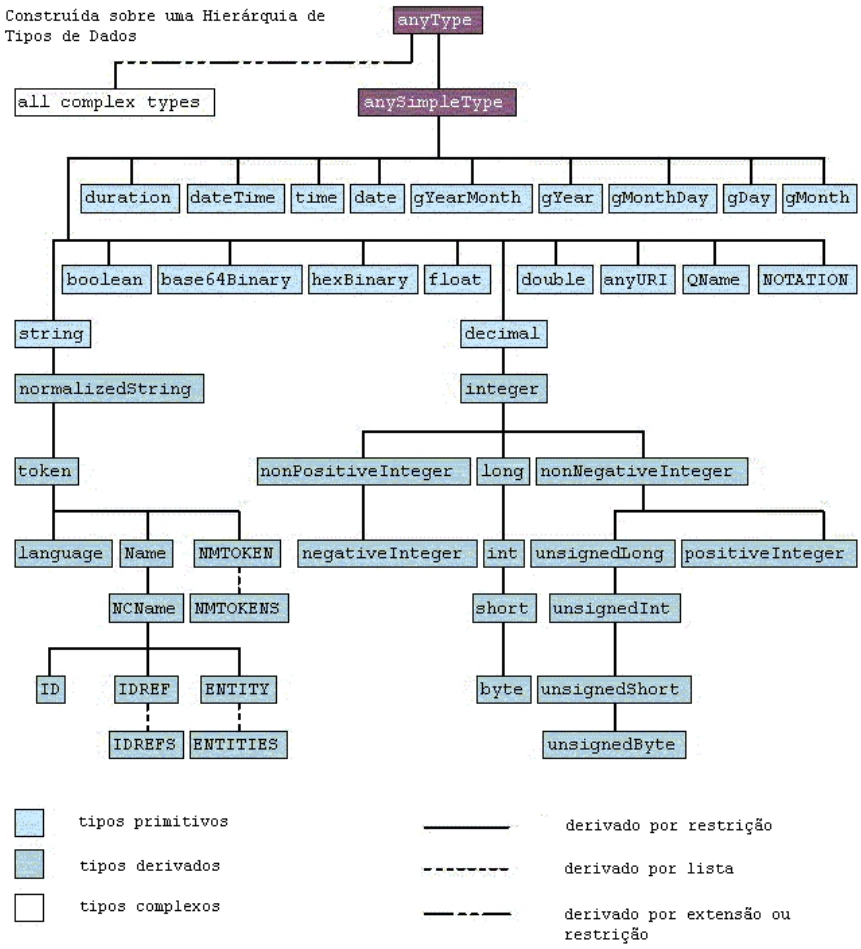


Figura 3.8: Hierarquia de *datatypes* de um XML *Schema*. Adaptada de (W3C, 2005)

Na Figura 3.8, todos os tipos de dados derivam de *anyType*, a partir do qual, a hierarquia pode ser dividida em dois grupos: tipos simples e complexos. Movendo-se na estrutura da árvore, os tipos derivam de seus pais, modificando algumas de suas propriedades e adaptando-se a quatro possíveis regras de validação: restrição, enumeração, lista e união. Como exemplo de restrição temos: *positiveInteger* herda de *nonNegativeInteger* pela restrição do escopo de valores para números positivos. Podemos ainda definir (enumerar) valores legais para um tipo de dado subjacente, como por exemplo, um *simpleType* chamado "feriado" e criar uma lista de possíveis valores que podem ser validados pelo mesmo. A regra de validação "união" permite a mistura de tipos de dados, tal como

positiveInteger e *String*, como exemplo cita-se uma companhia que usa o número 9 ou a *string* "vice presidente" como um elemento em seu XML (W3C, 2005).

Para facilitar o processo de validação, o programador da aplicação pode utilizar *parsers*, dentre os principais encontram-se o *Document Object Model* (DOM) e o *Simple API for XML* (SAX) (Coyle, 2002).

3.2.2.3 *Web service Description Language*(WSDL)

A WSDL é uma linguagem padrão XML que foi criada para descrever e publicar os formatos e protocolos de um *Web service*, e cujos elementos contêm a descrição dos dados, das operações que podem ser realizadas com estes dados e informações sobre o protocolo de transporte que será utilizado (Newcomer, 2002).

Projetada para ser analisada como qualquer outro documento XML, a WSDL é altamente flexível e extensível. Assim, se o remetente e o destinatário da mensagem puderem compartilhar e entender arquivos WSDL da mesma forma, então a interoperabilidade pode ser assegurada (Souza, 2004).

Segundo Newcomer, a WSDL é dividida em três elementos principais: definições de tipo de dados, operações abstratas e protocolos de ligação. Cada um desses elementos pode ser especificado em documentos XML diferentes e importado em diferentes combinações para criar a descrição final de um *Web service*, ou definidos juntos em um único arquivo XML. A definição de tipo de dados determina a estrutura e o conteúdo das mensagens, sendo que as operações abstratas determinam as operações possíveis, e o protocolo de ligação determina a forma de transmissão das mensagens pela rede até os destinatários.

O uso da WSDL permite a divisão da descrição dos serviços básicos em duas partes (interface e implementação do serviço) como mostrado na Figura 3.9, o que permite que estas partes possam ser utilizadas separadamente (Souza, 2004).

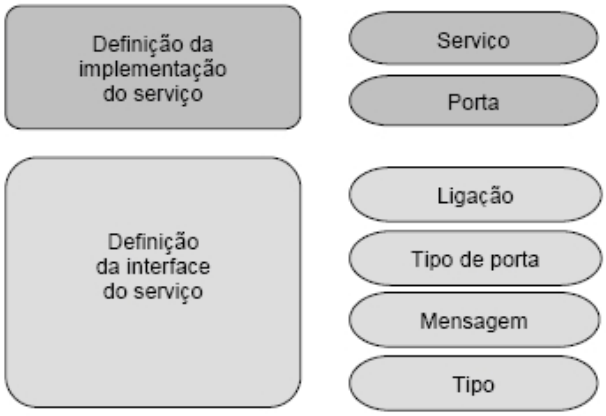


Figura 3.9: Camada de descrição de serviços. (Souza, 2004)

A implementação do serviço, definida através de um documento WSDL, descreve como uma interface é implementada por um provedor. Para isso, um arquivo de implementação descreve onde o *Web service* está instalado e como este é acessado. Além das definições de interface e

implementação, a WSDL especifica extensões para ligação com protocolos e formatos de mensagem, como SOAP, HTTP e *Multipurpose Internet Mail Protocol* (MIME) (Hansen, 2003).

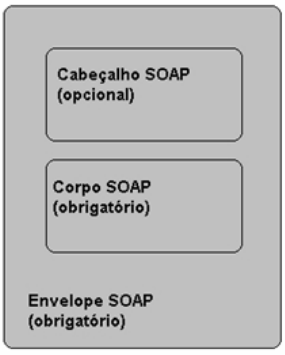
Uma interface do serviço contém a definição WSDL, o que possibilita que esta seja utilizada, instanciada e referenciada por múltiplas definições de implementação de serviços. A ligação descreve o protocolo, o formato dos dados, a segurança, e outros atributos para a interface de um serviço particular; no tipo de porta, os elementos das operações do *Web service* são definidos; a mensagem é utilizada para definir os parâmetros de entrada e saída de dados de uma operação; e o tipo define o uso de tipos de dados complexos dentro de uma mensagem (Hansen, 2003).

Newcomer (Newcomer, 2002) considera um arquivo WSDL difícil de ler e também entender, entretanto, cita que grande parte das ferramentas de desenvolvimento de *Web services* gera e consome estes arquivos automaticamente.

3.2.2.4 *Simple Object Access Protocol* (SOAP)

O SOAP é um protocolo utilizado na troca de informações em um ambiente descentralizado e distribuído, permitindo que isso seja feito entre diversas aplicações independente de sistema operacional, linguagem de programação ou plataforma (Newcomer 2002). Isso porque a comunicação é feita através de trocas de mensagens, transmitidas em formato XML, incluindo parâmetros usados na chamada, bem como os dados de resultados, o que significa que as mensagens podem ser entendidas por quase todas as plataformas de *hardware*, sistemas operacionais, linguagens de programação ou *hardware* de rede. Pode ser utilizado também para invocar, publicar e localizar *Web services* no registro UDDI (Hansen, 2003).

O pacote SOAP constitui-se de três partes, que podem ser verificadas na Figura 3.10(Seely 2002):




```
<env:Envelope xmlns:env="">
<env:Body>
<m:getLastTradePrice env:encodingStyle =
"http://www.w3.org/2001/06/soap-enconding"
xmlns:m="http://www.w3.org/2001/06/quotes"
<symbol>DIS</symbol>
</m:getLastTradePrice>
</env:Body>
</env:Envelope>
```

Figura 3.10: Partes de uma mensagem SOAP

- envelope SOAP: define o início e fim da mensagem, quem pode processá-la e se o tratamento é obrigatório ou opcional;
- codificação SOAP: define os mecanismos de serialização que podem ser usados para a troca de instâncias ou tipos de dados por uma aplicação;
- RPC SOAP: especifica como o modelo RPC interage com o SOAP, com o objetivo de invocar procedimentos em um sistema remoto.

Além dos itens citados anteriormente, outros conceitos são importantes (Hansen, 2003), tais como:

- cliente SOAP: é o programa que cria um documento XML contendo as informações necessárias para invocar remotamente um método de um sistema distribuído;
- servidor SOAP: é o responsável por executar uma mensagem SOAP e atuar como um interpretador e distribuidor de documentos;
- mensagem SOAP: é a forma de comunicação básica entre nodos SOAP.

A invocação do serviço usando o SOAP ocorre como demonstrado na Figura 3.11

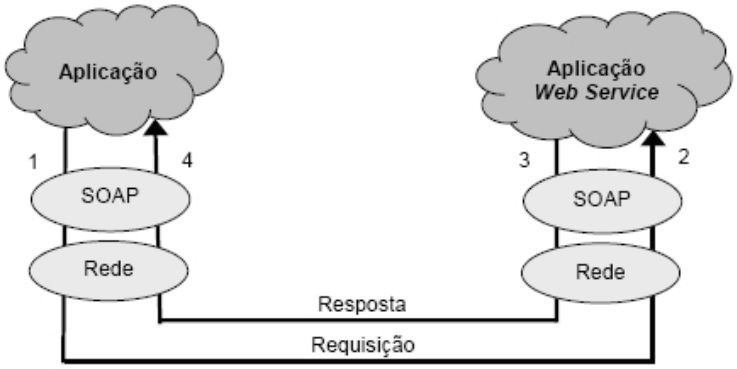


Figura 3.11: Invocação de um serviço através do protocolo SOAP. (Souza, 2004)

Primeiramente, a aplicação (1) requisita uma mensagem SOAP e invoca a operação do serviço através de um provedor de *Web services*. Logo após, o solicitante de serviço apresenta a mensagem junto com o endereço de rede do provedor de *Web service*, e a infra-estrutura de rede (2) entrega a mensagem para um servidor SOAP. Então, o servidor SOAP redireciona a mensagem requisitada para o provedor de serviço *Web service*. O servidor *Web* (3) é responsável por processar uma mensagem de requisição e formular resposta, e quando a mensagem XML chega no nodo requisitante, é convertida para uma linguagem de programação, sendo então entregue para a aplicação (4) (Hansen, 2003).

3.2.2.5 *Universal Description, Discovery And Integration*(UDDI)

O UDDI consiste em uma especificação técnica para descrever, descobrir e integrar *Web services*, e é constituído de duas partes: uma especificação técnica para construir e distribuir *Web services*, através das quais as informações são armazenadas em um formato XML específico; e, o UDDI *Business Registry*, que é uma implementação operacional completa da especificação UDDI (Newcomer, 2002).

Os dados UDDI são divididos em três categorias principais (Cerami, 2002):

- páginas brancas: inclui informações gerais sobre uma empresa específica, tais como nome, descrição, contato, endereço e números de telefone;

- páginas amarelas: nesta categoria são incluídos dados gerais de classificação da empresa ou serviço oferecido;

- páginas verdes: contém informações técnicas sobre *Web services*, e possui geralmente, um ponteiro para uma especificação externa e um endereço para invocar o *Web service*.

As entidades UDDI provêem suporte para descrever a informação sobre negócios e serviços, e como a informação presente no WSDL complementa a informação presente no UDDI. O processo de registro das informações ocorre como demonstrado na Figura 3.12 (Newcomer, 2002).

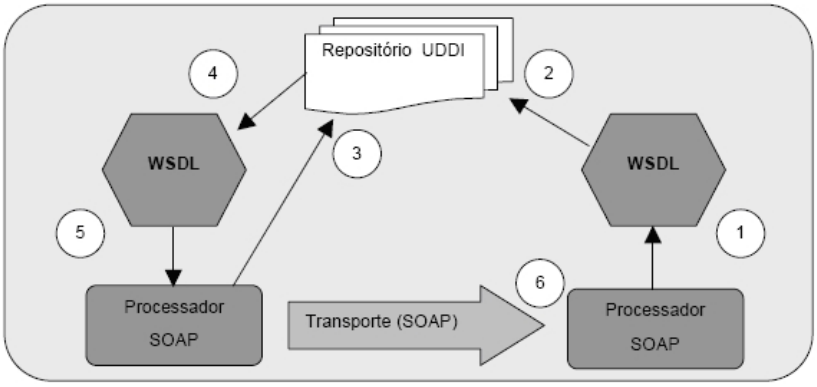


Figura 3.12: UDDI utilizado para descobrir um *Web service.* Adaptada de (Newcomer, 2002)

Sendo assim, gera-se primeiramente o arquivo WSDL para descrever o *Web service* com suporte do processador SOAP (1) e utiliza-se a API UDDI para registrar as informações no repositório (2). Depois disso os dados são transmitidos junto com informações sobre contato, e o registro possui uma entrada com uma URL que aponta para o Servidor SOAP com a localização do WSDL ou outro documento com a descrição do *Web service*. Então, um outro processador SOAP requisita o registro (3) para obter o WSDL (4) e, finalmente o cliente gera a mensagem apropriada (5) para enviar a uma operação específica através de determinado protocolo (6). Cabe salientar que o cliente e o servidor devem estabelecer o mesmo protocolo (nesse exemplo, SOAP sobre HTTP) e compartilhar a mesma semântica para a definição do serviço, a qual, neste exemplo, é definida através da WSDL (Newcomer, 2002).

A informação que conclui o registro de um serviço tem quatro estruturas de dados. Esta divisão na forma de tipo de informação oferece uma divisão simples que ajuda em uma busca rápida e na compreensão dos diferentes dados que compõem o registro (Figura 3.13) (Souza, 2004).

- ***businessEntity:*** representa a informação sobre um negócio específico ou informações descritivas sobre uma entidade, bem como os serviços que ele fornece. Do ponto de vista de um arquivo em XML, o *businessEntity* é uma estrutura de dados *top-level* que considera a descrição da informação sobre um negócio ou entidade;

- ***businessService:*** define descrições técnicas e de negócios (nome, descrição e uma lista opcional de *bindingTemplates*) para um *Web service*, ou para um grupo de *Web services* relacionados, e representa uma classificação lógica do serviço. O nome do elemento inclui

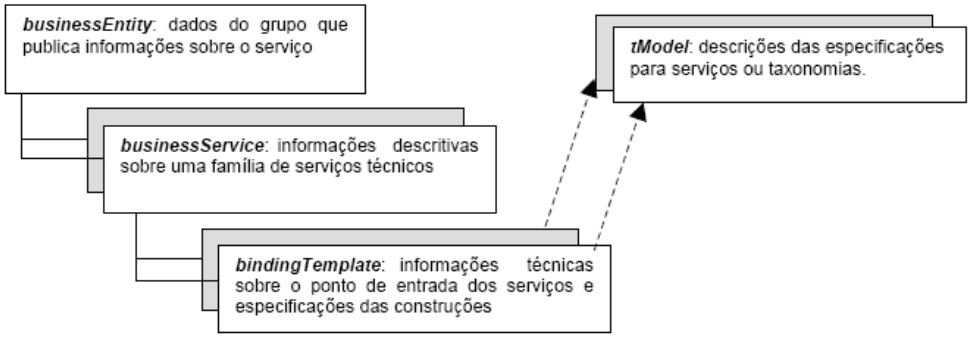


Figura 3.13: Estrutura UDDI. (Souza, 2004)

o termo "*business*" numa tentativa de descrever o propósito deste nível na hierarquia de descrição de serviços. Assim, cada estrutura *bussinessService* é filho lógico de uma única estrutura *businessEntity*;

- ***bindingTemplate:*** são descrições técnicas de como e onde acessar um *Web service* específico, além de serem as estruturas que fornecem suporte para acessar remotamente os serviços. Através delas, também são definidos o suporte, às tecnologias, os parâmetros específicos da aplicação, e os arquivos de configuração. Não é utilizado apenas com serviços baseados em HTTP, mas também em *e-mail*, fax, FTP, dentre outros;

- ***tModel:*** é representada por meio de metadados (dados sobre dados). O propósito de um *tModel* dentro de um registro UDDI é oferecer um sistema de referência para os documentos WSDL.

CAPÍTULO 4

Trabalhos relacionados

Neste capítulo, serão apresentados alguns trabalhos que, de alguma forma, relacionam-se ao que foi desenvolvido nesta pesquisa. Primeiramente, é feita uma análise sobre a estratégia utilizada no projeto Anima para a integração de softwares em um ambiente de e-learning. Logo após, é apresentado uma solução de um ambiente educacional interativo acessado através de PDAs e equipamento do gênero. Então, é analisado um framework para integração de aplicações B2B (Business to Business), o Microsoft BizTalk. Por fim, são apresentadas as considerações finais em relação aos trabalhos apresentados.

4.1 Projeto Anima

O Anima consiste em um projeto de mestrado desenvolvido por André Santanchè no ano de 2002, para qual uma das suas principais metas é a integração de *software* independente de plataforma ou implementação, e cujas origens partem de um ambiente destinado à construção de aplicações educacionais, denominado "Casa Mágica". Um projeto desenvolvido a partir deste ambiente é organizado na forma de componentes de *software* intercomunicantes e a tarefa principal do Anima foi estabelecer um modelo que permitisse a integração destes componentes com quaisquer outros, independente de sua origem, linguagem ou implementação(Santanchè, 2002).

Para isso, o modelo proposto foi organizado sob a forma de objetos intercomunicantes, onde os componentes de *software* são tomados como um tipo particular de objeto de forma a se obter uma representação homogênea para o sistema completo, e os dados deslocados para uma camada neutra, independente de plataforma e implementação, a qual utiliza XML como base para representação (Figura 4.1).

Segundo Santanchè (Santanchè, 2002), torna-se necessário identificar quais dados são candidatos ao deslocamento. Além disso, em sistemas orientados a objetos, que são o alvo do modelo Anima, o módulo de *software* é codificado como um conjunto de objetos que pertencem a classes, as quais representam aspectos estáticos. Por sua vez objetos da mesma classe se diferenciam pela sua identidade e pelo seu estado, representados através de valores de seus atributos. A separação entre classes e objetos permite representar objetos através dos dados que os distinguem na camada

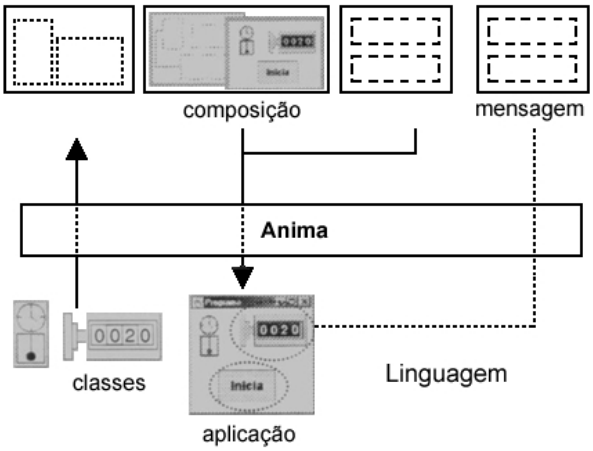


Figura 4.1: Deslocamento de representações para uma camada neutra.(Santachè, 2002)

neutra em XML, mantendo a implementação de sua classe codificada na linguagem de programação original.

Uma aplicação é representada no modelo Anima como a captura de uma configuração particular de um conjunto de objetos e o modo como eles estão interligados. O recurso denominado "composição", representado na figura 4.1, contém a representação XML tanto da configuração dos objetos, quanto de sua interligação, funcionando como um "espelho"que reflete a constituição de uma aplicação em uma superfície neutra. Tais espelhos, como consistem em representações XML, podem circular livremente pela Internet e assumem a forma de aplicação apenas quando chegam ao seu destino. Este processo é baseado em mecanismos de conversão que utilizam a linguagem *XSL Transformation* (XSLT) para realizar a transformação do documento XML em uma saída diferente (Santanchè, 2002).

A integração de componentes de *software* exige um registro na camada neutra de informações associadas à sua classe (tais como atributos que os caracterizam ou mensagens a que objetos respondem), interfaces de que se dispõem para a integração, e metadados (especialmente caracterizados com objetos educacionais) tal como foi definido pelo *Learning Object Model* (LOM). Para fazer este registro, Santanchè utilizou um recurso conhecido como "Classe Interface e Metadados"(CIM), o qual utiliza o *Resource Description Framework* (RDF). A representação CIM abrange aspectos como descrição de características das classes relevantes para o projeto de composições, entre elas identificação, herança, atributos e interfaces, e declaração dos tipos de mensagens que as classes podem receber ou enviar metadados (Santanchè, 2002).

As mensagens são transferidas para a camada neutra em XML com o intuito de viabilizar a comunicação entre objetos, independente da linguagem ou sistema. Por sua vez, cada classe de objetos, que irá se comunicar utilizando as mensagens em XML, adiciona uma interface de conversão, cuja função é transformar as mensagens do formato interno para o externo e vice-versa (Santanchè, 2002).

Sendo assim, objetos da mesma aplicação são agrupados em um espaço de trabalho, e para cada espaço de trabalho existe um módulo denominado "mediador", responsável pelo trânsito de mensagem entre os objetos. Como objetos de diferentes linguagens ou sistemas utilizam espaços

distintos, os mediadores destes espaços estão habilitados a se comunicar e atuam como *proxy*, transferindo mensagens entre objetos de espaços distintos (Santanchè, 2002).

Segundo Santanchè, deslocar apenas uma parte da aplicação para uma camada neutra parece ser uma solução incompleta. No entanto, através da utilização de linguagens de código móvel (*Mobile Code Language*) os componentes de *software* conseguem trafegar livremente pela rede, podendo ser executados em diferentes plataformas e sistemas.

Portanto, a combinação de classes produzidas em linguagens de código móvel e representações CIM resultam em pacotes altamente autônomos e portáveis, não apenas no que diz respeito a sua execução, como também em relação às informações necessárias para a sua compreensão e uso. Do ponto de vista da integração de resultados, os objetos que as compõem poderão vir acompanhados de documentação, através da representação CIM da classe, o que permite o entendimento das peças de uma composição independente do ambiente que as produziu (Santanchè, 2002).

4.2 Projeto EUME

O projeto EUME, o acrônimo inglês para *Ubiquitous and Multimedia Environment for Education*, consiste em um sistema educacional suportado por computador, e pode ser definido como uma plataforma que disponibiliza serviços utilizados para a atividade de ensino, ferramentas de autoria e gerenciamento de *hardware* em ambientes de ensino à distância através da interface de um PDA.

O ambiente EUME é constituído por dispositivos móveis (PDAs), um vídeo projetor, um *whiteboard* interativo e um servidor. O protótipo provê diferentes serviços de gerenciamento de recursos, como um controle remoto do projetor, gerenciamento remoto de arquivos, navegação via *web*, assim como também a possibilidade de o professor gravar e acessar anotações no *whiteboard* (Vila, 2003 - Riera, 2004).

4.2.1 Arquitetura

A arquitetura baseia-se em um sistema híbrido composto por quatro camadas, o qual é implementado através da tecnologia Java e a API *Remote Method Invocation* (RMI), que estabelece uma comunicação entre as diferentes camadas, representadas através da Figura 4.2 e descritas a seguir.

- *Resource Tier* (camada de recursos): contém os componentes que manuseiam diretamente a aplicação. Esta camada oferece um serviço de nomes (*Name Service*), o qual pode ser consultado por serviços externos para verificar os recursos disponíveis e sua localização [1].

- *Service Tier* (camada de serviços): representa a camada constituída por componentes de *software*, os quais provêem um gerenciamento em alto-nível dos serviços de aprendizado, que por sua vez são agrupados de acordo com opções pré-definidas pelos usuários, e são chamados de *Resource Management Systems* (RMS). Um exemplo de serviço nesta camada é um gerador de perguntas e um analisador de resultados.

- *Client Tier* (camada de cliente): é necessária para que o usuário possa interagir com o sistema. O *Professor Out-of-Classroom interface* (POC/I) permite ao professor obter informações do

[1] (Riera, 2000) utiliza o termo recurso para referir-se a *web browsers*, aplicações *desktops*, projetores digitais e um quadro branco interativo.

curso para planejar suas atividades educacionais, projetar recursos e orientar atividades; o *Professor In-Classroom Interface* (PIC/I) auxilia o professor na requisição de recursos e a estabelecer a comunicação com estudantes; e ainda AI, SIC/I e GIC/I (*Administrator, Student In Classroom e Group In-Classroom*), que são interfaces para administrador, estudante e grupo, respectivamente.

- *Mediator Tier* (camada mediadora): esta camada atua como um *bridge* entre clientes e serviços.

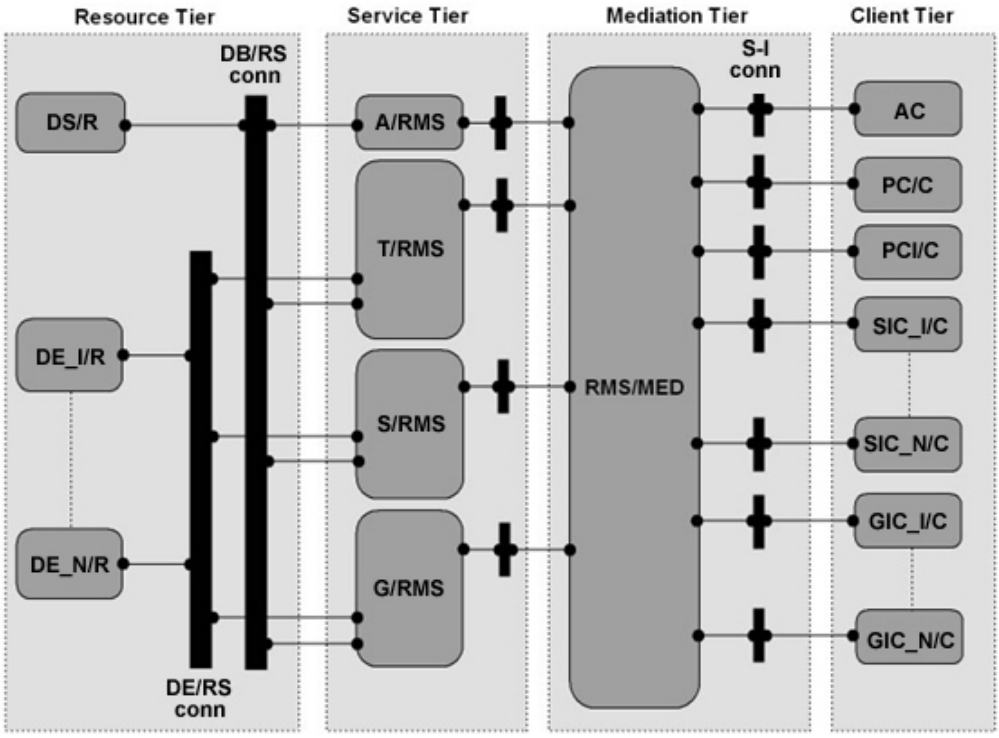


Figura 4.2: Arquitetura do Projeto EUME

4.2.2 EUME Onto

A EUME Onto é uma ontologia educacional que descreve termos, índices e recursos do projeto de aprendizagem, composta por três ontologias relacionadas:

1. *Learning Design*: baseada na especificação do Instructional Management Systems (IMS), o objetivo desta ontologia é fornecer uma base de conhecimento para que os professores possam especificar suas atividades, pré-requisitos, métodos e recursos que serão utilizados em um curso (Figura 4.3).

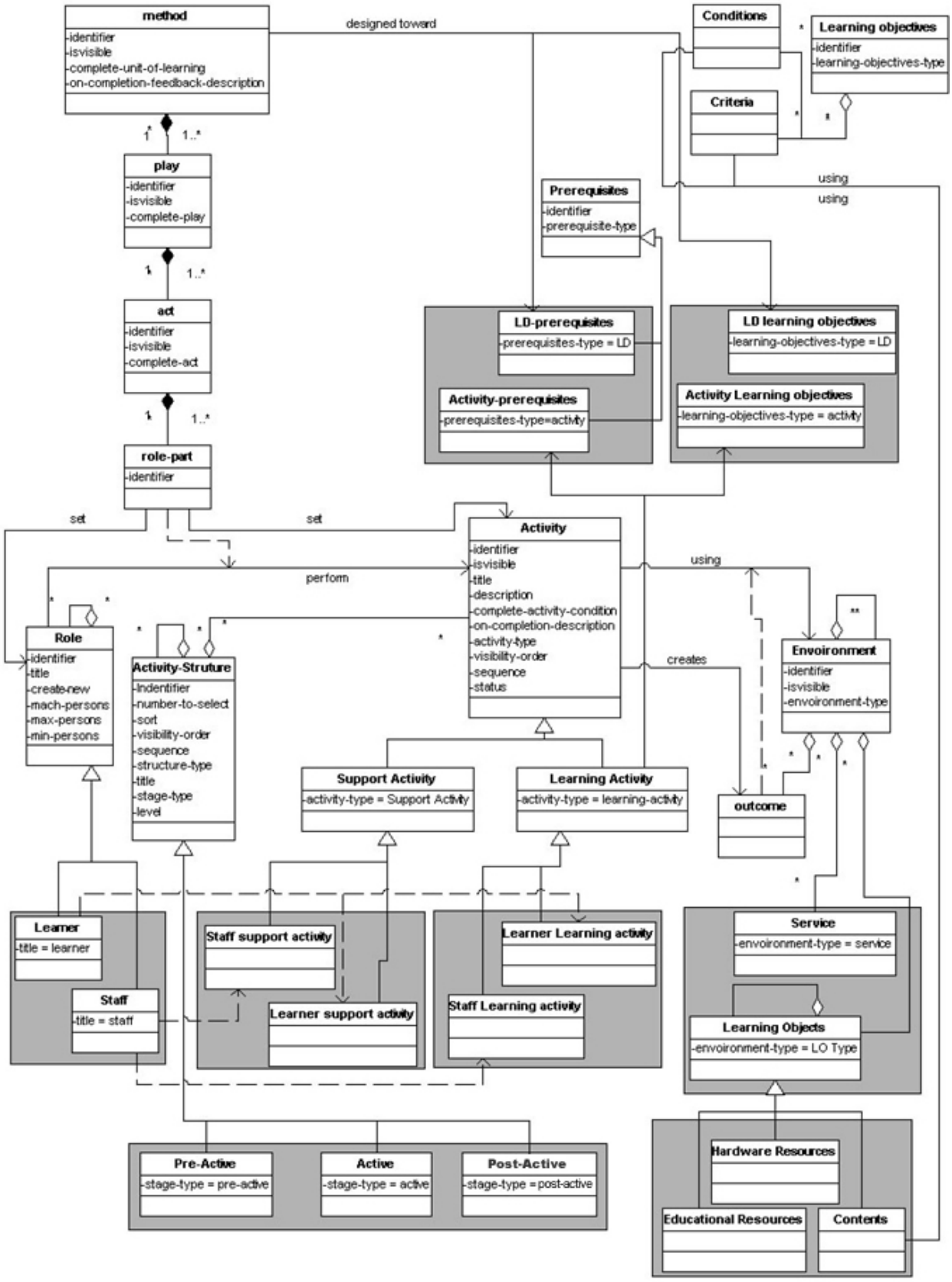


Figura 4.3: Ontologia de *Design* de Aprendizado

Para ilustrar esta ontologia, o diagrama da Figura 4.4 demonstra um exemplo de atividade, onde, baseado em conceitos pré-definidos sobre comunicação, e a partir de uma situação na qual diversos formulários de comunicação são postos, os estudantes podem ser questionados sobre dez categorias de comunicação mostradas em um filme e em uma exposição oral, devendo descrever, no mínimo, sete delas.

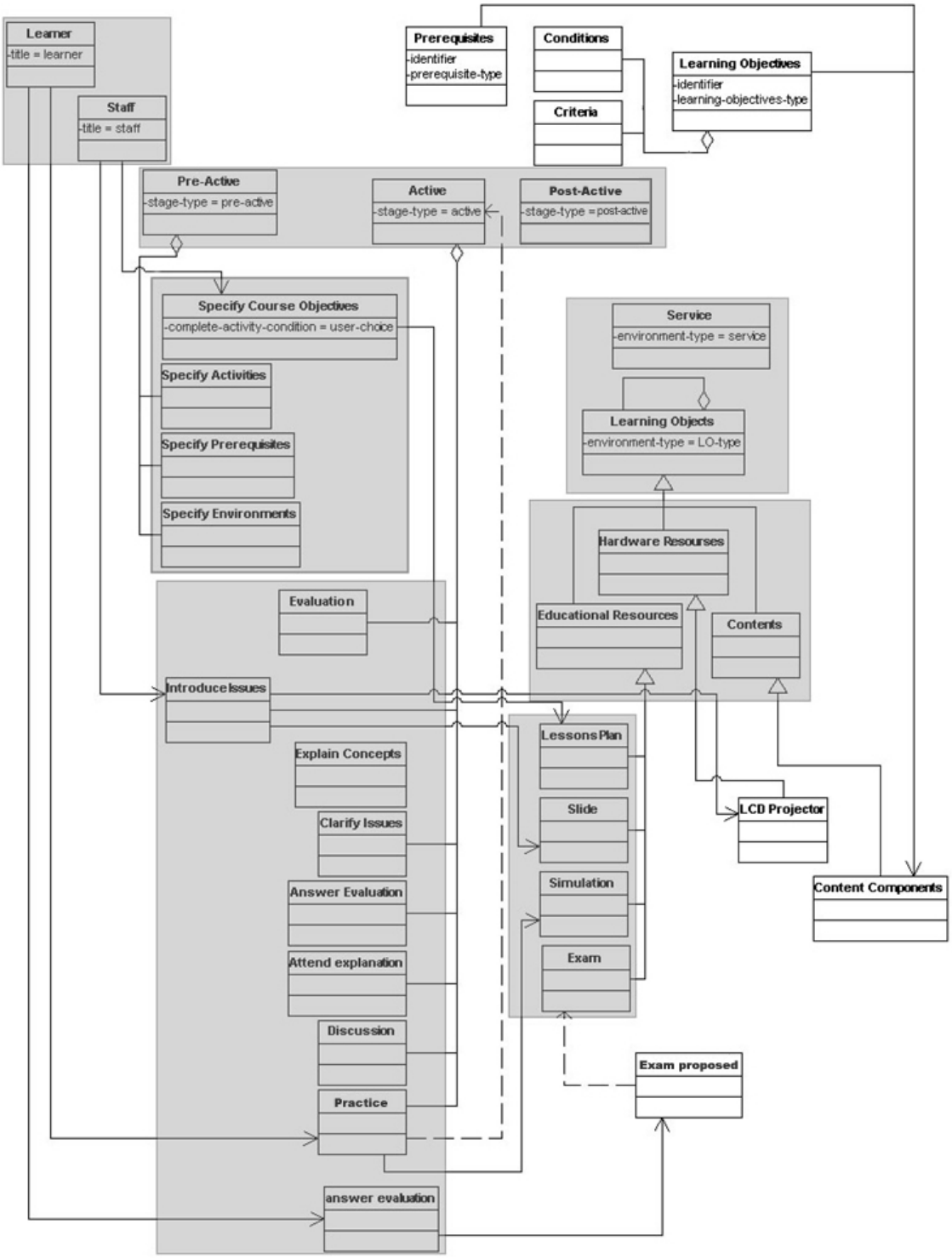


Figura 4.4: Aplicação da ontologia de *Design* de Aprendizado

Os termos especificados dentro da ontologia (Figura4.4) compreendem:

- *Learning objetive:* permite verificar a compreensão do estudante sobre conceitos de comunicação;
- *Prerequisites:* conceitos sobre categorias de comunicação;
- *Roles:* assinatura do professor como funcionário e do estudante como aprendiz;
- *Activities:* explanação de conceitos, demonstração de exemplos e esclarecimentos podem ser consideradas como atividades completas de aprendizagem. As atividades *Evaluation* e *Answer Evaluation* são consideradas atividades de suporte para a atividade corrente;
- *Environment:* são considerados o filme, livros, artigos das referências bibliográficas, o quadro branco e o PDA.

2. *Learning content and educational/didatical resource:* os conceitos utilizados nesta ontologia especificam os componentes (blocos) de índices e estruturas necessárias para a criação de objetos de aprendizagem. Exemplos de componentes são: parágrafos, tabelas, figuras, *slides* e animações, os quais podem ser utilizados para criar outras estruturas, mais gerais, como por exemplo, um capítulo, ou um livro.

3. *Hardware resource ontology*: descreve os recursos eletrônicos necessários para suportar o processo de aprendizado. Esta ontologia é necessária para que os agentes possam acessar os recursos adicionais, como os projetores, quadro branco eletrônico, PDAs, vídeo câmeras, recursos de audio, etc.

4.3 Microsoft BizTalk

O Microsoft BizTalk, segundo Chris Peltz (Peltz, 2003), consiste de uma solução para projeto, criação e execução de integrações entre aplicações B2B, com base na utilização de *Web services*. O BizTalk *Server* permite aos desenvolvedores e arquitetos de sistema a utilização da mesma metodologia de desenvolvimento e componentes para a criação de processos de fluxo de trabalho, interfaces de integração de aplicativos e interações de parceiros de negócio (DevBtSol, 2004). Esta metodologia foi chamada pela Microsoft de SOA (*Service Oriented Architecture*) e está fundamentada em tecnologias como a XML e *Web services*.

Fazem parte deste produto:

- uma ferramenta de edição XML para a definição da semântica (esquemas XML) dos documentos;
- uma ferramenta de mapeamento baseado em XSLT para transformar documentos em diferentes formatos de maneira dinâmica;
- uma infra-estrutura de mensagens "Publicar e Assinar"que fornece as facilidades do processamento lógico, onde intercâmbios entre documentos podem ser validados, autenticados, criptografados, transformados e roteados. A correlação e persistência das mensagens e transações também estão previstos;

- um modelo gráfico de orquestração para a criação de processos sofisticados usando uma metodologia de construção no formato "arrastar e colar".

Para Chris Peltz (Peltz, 2003), um dos mais importantes e poderosos recursos do BizTalk Server é a adoção do padrão de esquema XML na definição interna dos seus documentos. Este esquema é utilizado para criar um modelo semântico (uma definição de documento) e estrutural interno das informações sobre formatos proprietários, que serão recebidos ou enviados para aplicações externas (Figura 4.5).

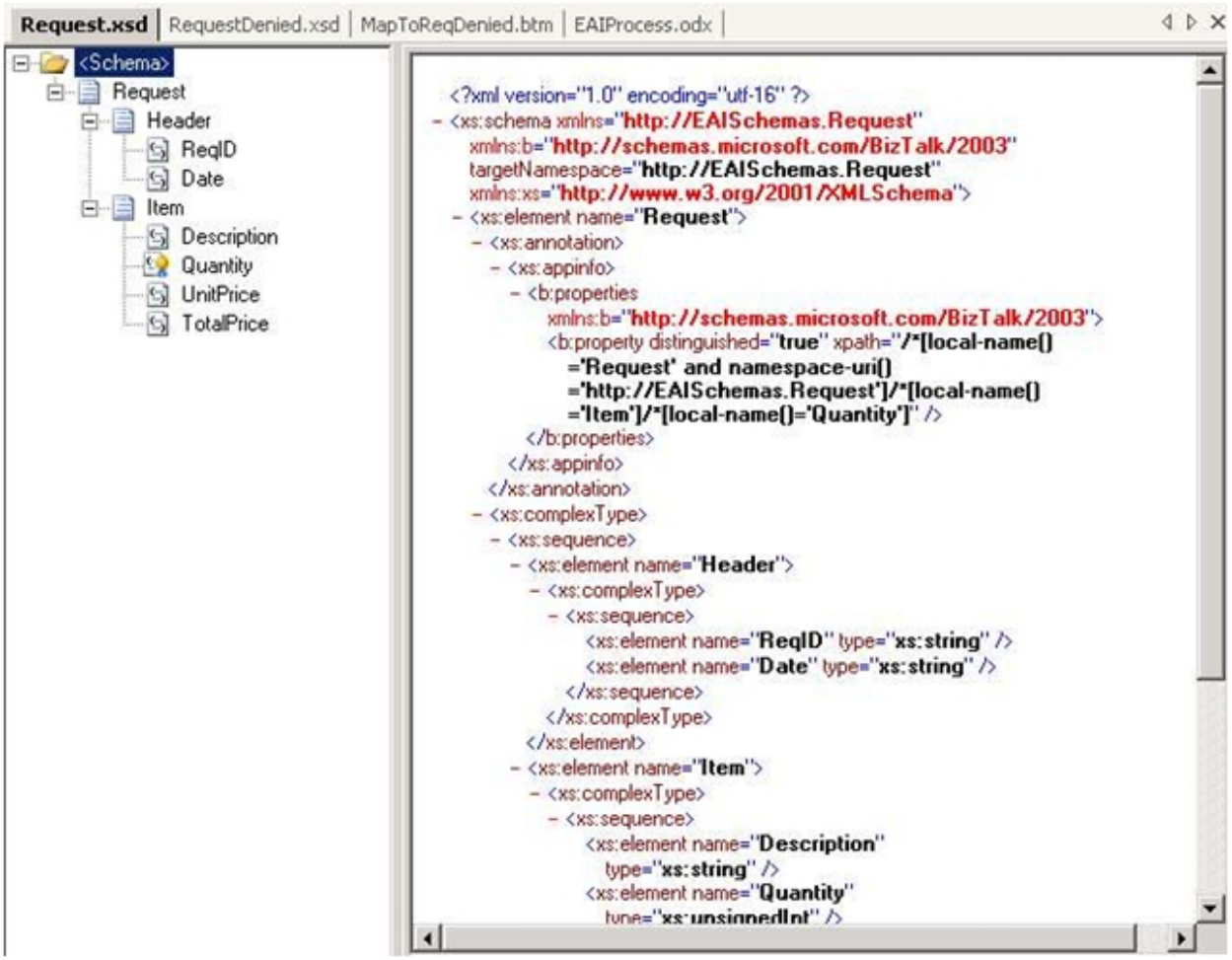


Figura 4.5: BizTalk *Schema Editor*

Uma ferramenta de mapeamento (Figura 4.6) é usada para a conversão da informação sobre o formato de uma aplicação para qualquer outro formato, e, posteriormente os mapas de transformação também são armazenados e publicados em um repositório.

Um intercâmbio é iniciado quando o BizTalk Server recebe informações de uma aplicação que são identificadas como uma entrada para outra aplicação, ele então executa a conversão do formato através do mapeamento, facilitando e fornecendo informações para a aplicação. A flexibilidade e eficiência deste concentrador de informações ficam mais evidentes quando ele é aplicado em intercâmbios “um-para-muitos” ou “muitos-para-muitos” (Chappell, 2004 - Peltz, 2003).

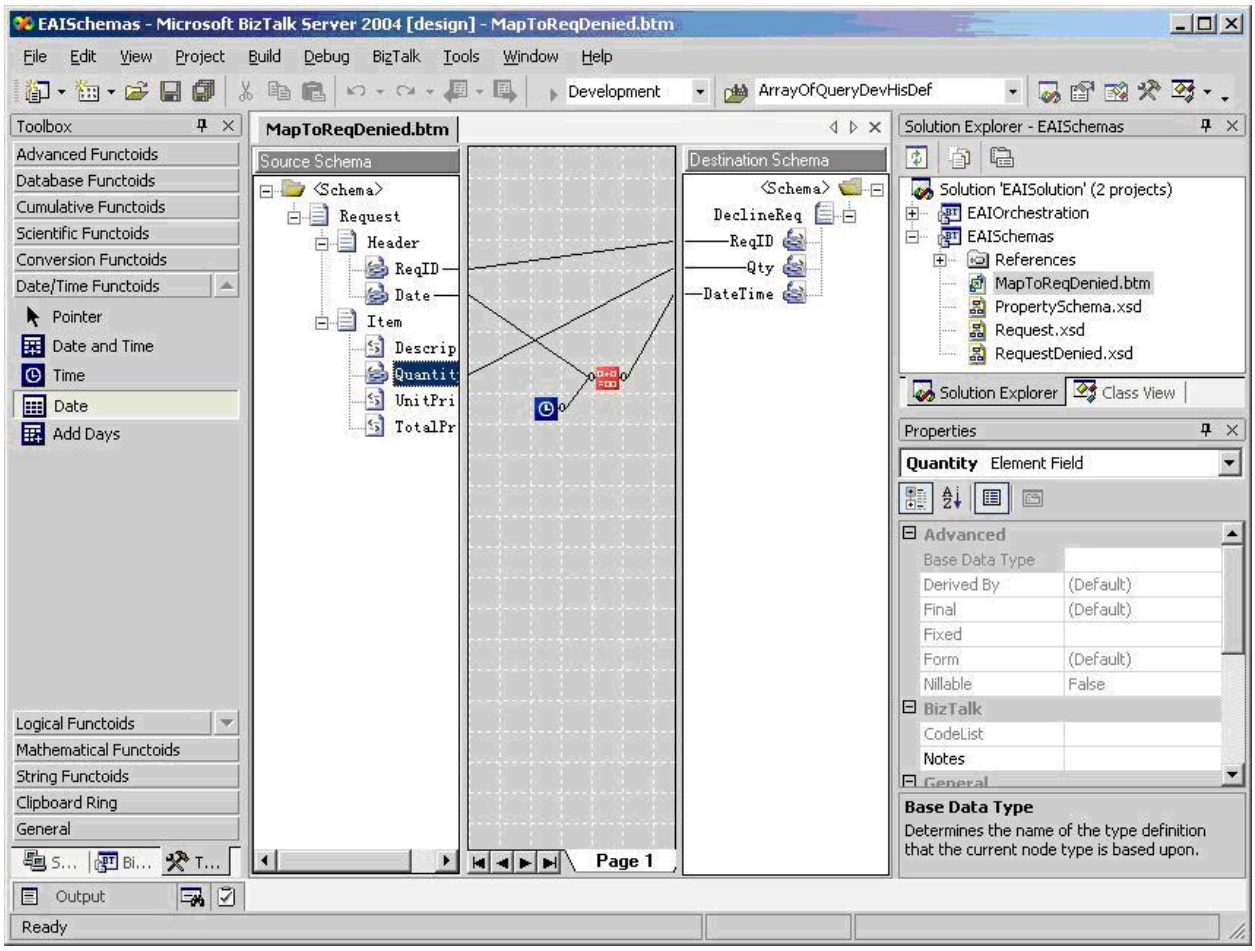


Figura 4.6: *BizTalk Server Mapper*

4.4 Considerações sobre trabalhos relacionados

Os estudos feitos por Santanchè no projeto Anima mostram a importância da reutilização e integração de *softwares* em ambientes eletrônicos de EAD. O trabalho realizado mostra, ainda, que características como reusabilidade, facilidade de distribuição, extensibilidade, facilidade de configuração e combinação são requisitos importantes no desenvolvimento de aplicações para execução nestes ambientes. Portanto, através da utilização de tecnologias como a XML e a XSLT, Santanchè desenvolveu um protótipo de arquitetura que possibilita a composição de aplicações e o intercâmbio de dados no ambiente "Casa Mágica".

O Microsoft BizTalk apresenta um conceito para a integração de aplicações B2B, elevando este processo da camada de programas para a camada de informação (documentos) e transporte (mensagens). Separando, desta forma, as informações das aplicações que operam sobre ela, expondo-as como texto puro e usando metadados XML auto-descritíveis para fornecer significado e estrutura, o que possibilita que a informação possa ser processada por qualquer aplicação capaz de processar e interpretar o metadado XML. Este produto da Microsoft permite até mesmo que as funções operacionais e os métodos de chamada das próprias aplicações possam ser descritos e expostos usando XML, permitindo que sejam executados sem preocupações com o lugar onde eles residem, como foram originalmente desenvolvidos, ou em que plataformas são executadas.

A arquitetura de *software* proposta pelo ambiente EUME possibilita o projeto de um ambiente de ensino cooperativo, em que estudantes e professores usam PDAs para acessar recursos educacionais, tais como dispositivos de *hardware* (projetores, *digital whiteboards*) e objetos educacionais. Com a finalidade de enfatizar escabilidade e extensibilidade, a arquitetura consiste em um esquema de "n-camadas", o qual é baseado na interação de agentes e no uso de ontologias para estabelecimento de comunicação entre componentes de *softwares* escritos na linguagem Java.

Os estudos realizados permitiram a definição da arquitetura proposta e das tecnologias para desenvolvimento do protótipo apresentado nesta dissertação de mestrado, pois assim como o projeto Anima e o Microsoft BizTalk, o WSMEL propõe uma arquitetura para a integração de aplicações. No entanto, a diferença é que o WSMEL integra *softwares* utilizados como mecanismos de comunicação em ambientes móveis para o ensino à distância, utilizando equipamentos como *handhelds* e PDAS, permitindo, desta forma, um ambiente similar ao proposto pelo projeto EUME, acrescentado à possibilidade de composição de um ambiente de EAD integrado formado por aplicações desenvolvidas em diferentes linguagens. Além disso, caracteriza-se pela utilização de *Web services* e ontologias, o que possibilita que a integração não dependa de alterações no código fonte de cada aplicação incorporada ao ambiente.

Para melhor compreensão da relação dos trabalhos descritos neste capítulo com o WSMEL, um comparativo entre estas arquiteturas pode ser conferido na tabela 4.1 a fim de justificar o desenvolvimento deste trabalho, conforme descrito acima.

Tabela 4.1: Quadro comparativo entre trabalhos relacionados

	Projeto Anima	Projeto Eume	Microsoft Biztalk	WSMEL
Integração entre *softwares*	X		X	X
Independência de Plataforma de desenvolvimento	X		X	X
Utilização de *Web services*			X	X
Execução distribuída		X	X	X
Interoperabilidade baseda em ontologias	X	X	X	X
Ambiente de execução móvel		X		X

CAPÍTULO 5

WSMEL: Integrando *softwares* educacionais em plataformas móveis

Nos capítulos anteriores, foram apresentados aspectos relacionados a agentes de software, ontologias, padrões de projeto e Web services. Neste capítulo, será descrita a proposta para promover a integração de aplicações no domínio da educação à distância e execução em ambientes móveis, cuja solução envolveu o desenvolvimento de uma ontologia preliminar e um agente de software para a manipulação de dados entre diferentes aplicações. Também propõe-se que os usuários possuam versões personalizadas de seus ambientes de ensino, onde as comunidades sejam formadas de maneira espontânea, levando em consideração os interesses e preferências de cada um. De forma resumida, esta arquitetura permitirá aos desenvolvedores de software projetar aplicações sob a forma de componentes e estas realizarem o intercâmbio de dados sem a preocupação de como isso será feito.

Grande parte dos ambientes de ensino à distancia são formados por vários módulos de programas que possuem procedimentos operacionais rígidos; uma vez que, para promover a integração entre aplicações, desenvolver capacidades programáticas estendidas ou tentar tornar a informação de maneira diferente da definida nestes ambientes leva um tempo considerável, além de despender grande capital e recursos, isto porque o trabalho envolvido no desenvolvimento da integração é formado por muitas tarefas de programação em baixo nível e seqüenciais (Sing, 2004).

Assim, este tipo de esforço em desenvolvimento é tipicamente medido em "homem-hora", e o processo é altamente linear, no qual cada etapa é dependente (até sua conclusão) de uma etapa anterior, o que implica que não pode ser facilmente quebrada, tanto completamente como em tarefas independentes executadas por recursos distribuídos. A conseqüência de aumentar a estrutura necessária para atender a carga de trabalho dos projetos de integração significa apenas adicionar mais e mais recursos de programação, pois como cada instância de integração é especializada e manifestada em um código monolítico não reutilizável, a eficiência geral de programação não é aumentada pela proliferação de recursos de programação (DevBTSol, 2004).

Os *Web services* surgem, então, como uma forma de reduzir significativamente as ineficiências de integração descritas no parágrafo anterior (Sadiq, 2003). Entretanto, quando isolados, possuem

uma relevante limitação funcional, portanto, desta forma, durante o decorrer desta pesquisa pode-se observar que o poder de integração dos *Web services* foi combinado com o uso de ontologias para determinar mecanismos de integração entre aplicações.

5.1 Arquitetura do *Web service Mobile Environment Learning*

A arquitetura WSMEL, ilustrada na Figura 5.1, foi criada para possibilitar ao usuário de *M-Learning* a construção de um ambiente com as aplicações de sua preferência, de maneira integrada, independente das linguagens às quais foram implementadas, em um ambiente de execução distribuído, no qual:

- ***Mobile Gateway Services***: é um servidor que hospeda os serviços da *web*, responsáveis por executar os métodos apropriados para prover a integração entre aplicações presentes na camada de interoperabilidade; camada esta que define a estrutura e os metadados semânticos dos aplicativos remotos. Além da camada de interoperabilidade, estão presentes neste servidor as camadas de agente, e persistência, que serão descritas nos próximos capítulos.
- ***Client Layer:*** consiste nas aplicações que estão sendo executadas nos equipamentos móveis, PDAs ou *handhelds*. Estes aplicativos têm funcionalidades individuais, como, por exemplo, enviar mensagens eletrônicas através de um cliente de *e-mail* ou cadastrar reuniões em uma agenda pessoal, entretanto, podem integrar informações através da invocação de serviços presentes no *mobile gateway services*.

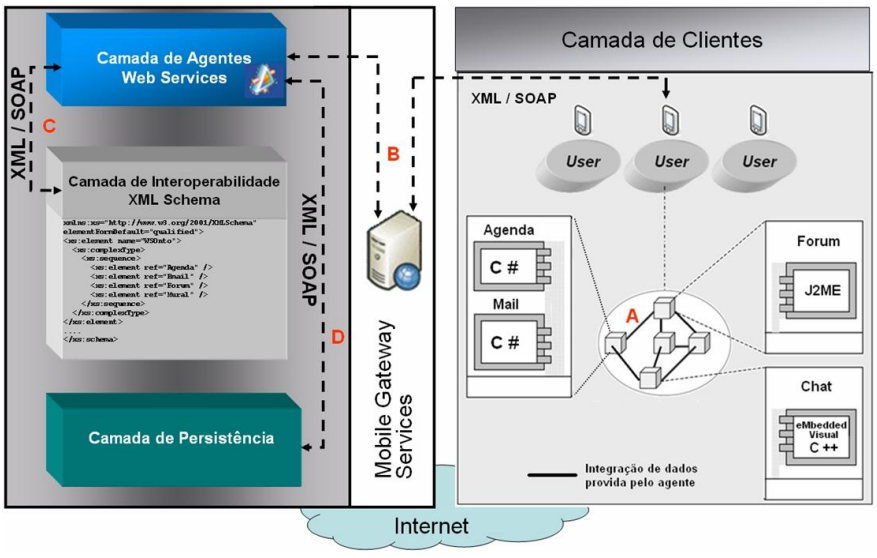


Figura 5.1: Arquitetura WSMEL

Através da ilustração exibida na Figura 5.1, podemos verificar em (a) um ambiente de *M-Learning* personalizado, formado de acordo com as preferências do usuário, sendo que aplicações da camada de cliente são que dão suporte ao processo de aprendizagem (*E-mail*, *Chat*, Fórum, Agenda, etc.), implementadas em diferentes linguagens de programação e acopladas ao ambiente, tais como componentes de *hardware* são acoplados na placa mãe de um computador. Assim como os componentes de *hardware* estabelecem uma comunicação e operam de maneira integrada objetivando

um resultado, as aplicações desenvolvidas com base nesta arquitetura podem operar de maneira integrada, realizando o intercâmbio de dados sempre que for possível. Tais aplicações, primeiro precisam ser suportadas pelo sistema operacional, e, segundo, têm que seguir a ontologia especificada na camada de interoperabilidade. Já, o processo de intercâmbio de dados requer a invocação de um agente (b), que procura relação entre dados de uma aplicação com dados da outra, através de pesquisas a camada de interoperabilidade (c) e do armazenamento destas informações na camada de persistência (d), atuando desta forma como um *middleware* de integração entre aplicações (Figura 5.2).

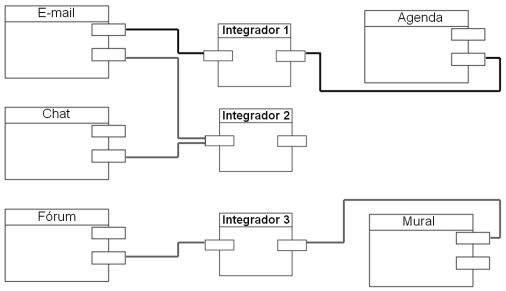


Figura 5.2: O agente *Web service* atuando como um *middleware* de aplicações

Uma das vantagens proporcionadas por esta arquitetura é que mesmo que o número de aplicações acopladas ao ambiente aumente, elas além de não interferirem no seu funcionamento, ainda podem realizar a integração com as demais sem qualquer alteração no seu código fonte, ou seja, aplicações que estão interessadas no intercâmbio de dados podem ampliar o sistema pela adição de agentes de *software*, que operam remotamente, sem sobrecarregar o sistema e sem exigir que uma conheça detalhes de implementação da outra.

As funcionalidades de cada camada são descritas de maneira resumida na Figura 5.3, e são abordadas de forma detalhada nos próximos capítulos.

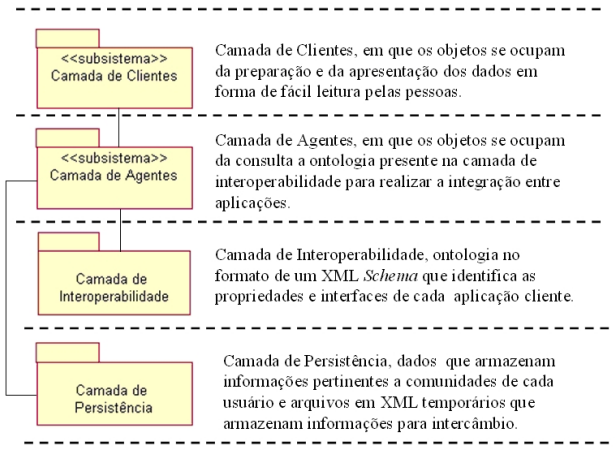


Figura 5.3: Funcionalidades resumidas das camadas WSMEL

5.1.1 Camada de agentes

A camada de agentes foi desenvolvida com o objetivo de garantir a flexibilidade e a integração entre as aplicações, sendo que, virtualmente, todas as formas de integração requerem que seja estabelecido um tipo de comunicação entre os fornecedores e os consumidores de dados (Sadiq, 2003). Portanto, como forma de permitir esta comunicação, o agente foi desenvolvido sobre a ótica de um *Web service*, padrão de tecnologia usado para integração entre aplicações e sistemas abertos (Peltz, 2003).

Assim, desenvolvê-lo como um serviço possibilita a sua agregação a diferentes aplicações, independentemente da linguagem as quais foram implementadas e da plataforma a qual estão sendo executadas. De maneira resumida, o agente tem a função de realizar um processo de mapeamento entre as aplicações com base em um vocabulário definido na camada de interoperabilidade (Figura 5.4).

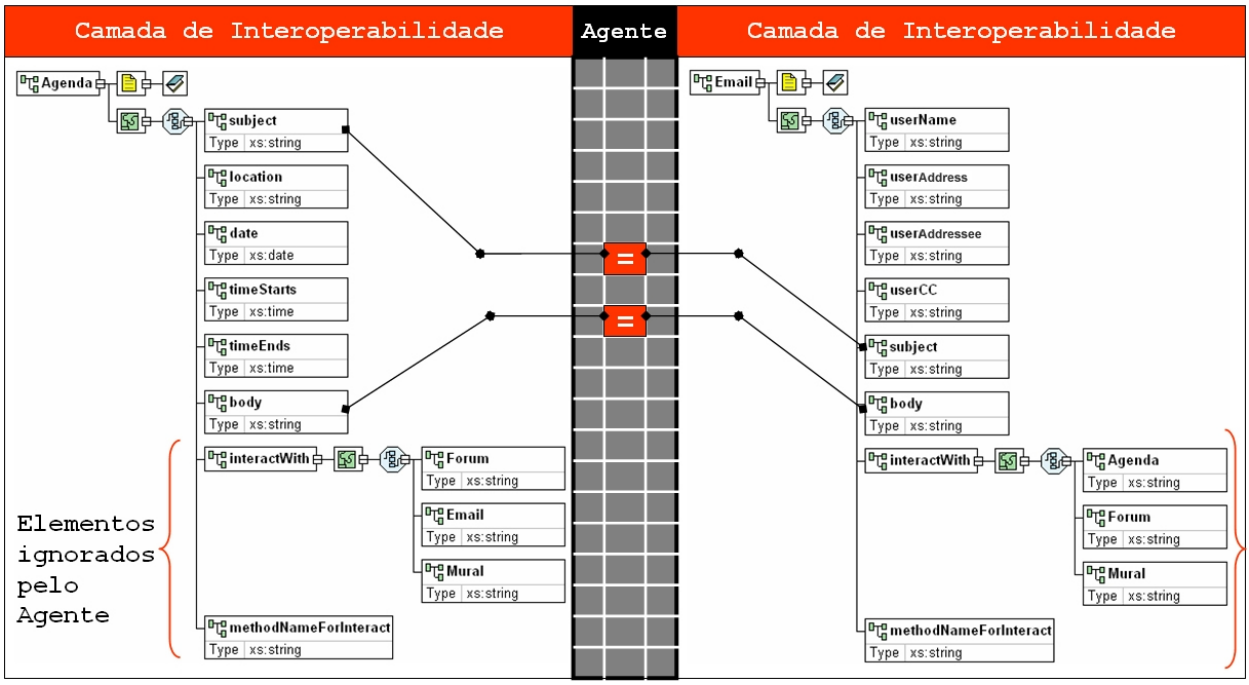


Figura 5.4: Simulação de um processo de mapeamento

O exemplo ilustrado na Figura 5.4 simula uma pesquisa para realizar a integração entre dois *softwares* pertencentes a um ambiente específico de um usuário aleatório, em que o agente identifica o vocabulário utilizado por cada uma das aplicações e os termos comuns entre elas, gerando informações na camada de persistência, no formato XML. Deste modo, três arquivos são criados na camada de persistência, sendo que os dois primeiros contém os elementos e respectivos tipos de dados relativos às aplicações diretamente ligadas ao intercâmbio de informações, e o terceiro contém quais informações podem ser integradas sem o auxílio do usuário final (Figura 5.5).

Origem

```
- <dsElemsDevice>
  - <elementsOfDevice>
      <element>user</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>subject</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>location</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>date</element>
      <dataType>date</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>timeStarts</element>
      <dataType>time</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>timeEnds</element>
      <dataType>time</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>body</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  </dsElemsDevice>
```

Destino

```
- <dsElemsDevice>
  - <elementsOfDevice>
      <element>userName</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>userAddress</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>userAddressee</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>userCC</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>subject</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  - <elementsOfDevice>
      <element>body</element>
      <dataType>string</dataType>
    </elementsOfDevice>
  </dsElemsDevice>
```

Atributos equivalentes e respectivas informações

```
- <dsElemsDevice>
  - <elementsEquals>
      <elementOne>subject</elementOne>
      <elementTwo>subject</elementTwo>
      <information>Reunião do grupo de Engenharia de Software</information>
    </elementsEquals>
  - <elementsEquals>
      <elementOne>body</elementOne>
      <elementTwo>body</elementTwo>
      <information>Definição de responsabilidades na escrita de artigo para o NWeSP 06.</information
    </elementsEquals>
  </dsElemsDevice>
```

Figura 5.5: Vocabulário utilizado pelas aplicações

É importante salientar ainda que o agente só identifica as propriedades totalmente idênticas, entretanto possibilita, via interface de mapeamento (Figura 5.6.A), a realização de outras associações, resultando na alteração do XML exibido na Figura 5.5. Também deve se notar que, para que a associação seja possível, os elementos devem possuir o mesmo *dataType*, caso contrário, uma mensagem de incompatibilidade é exibida ao usuário (Figura 5.6.B).

(A) Mapeamento executado pelo usuário (B) Tipos incompatíveis

Figura 5.6: Interface gráfica de mapeamento

O processo de interação entre as aplicações, o agente e a interface gráfica de mapeamento podem ser observados no próximo capítulo.

5.1.1.1 Modelo de Interações

O modelo apresentado na Figura 5.7 mostra o funcionamento da solução de forma seqüencial, inserido no contexto do problema de integração entre aplicações heterogêneas.

Uma das primeiras ações executadas pela aplicação que o usuário acabou de instanciar, antes mesmo de abrir o formulário principal, é solicitar ao agente que verifique na camada de interoperabilidade as aplicações com as quais pode interagir. Conseqüentemente o retorno desta informação habilita a aplicação instanciada a realizar a integração de dados e o usuário pode, a partir deste momento, escolher o destino das informações.

Após a determinação de qual aplicação será o destino dos dados, o agente através dos métodos apropriados volta a pesquisar a camada de interoperabilidade em busca de elementos equivalentes entre as duas aplicações. Então, terminada esta pesquisa, o agente gera informações na camada de persistência e comunica à aplicação que o invocou, que, por sua vez, invoca a interface de mapeamento responsável pela leitura dos dados na camada de persistência e exibição no formulário apropriado, no qual o usuário consegue interagir e fazer associações que o agente não foi capaz de realizar, refletindo em alterações na camada de persistência.

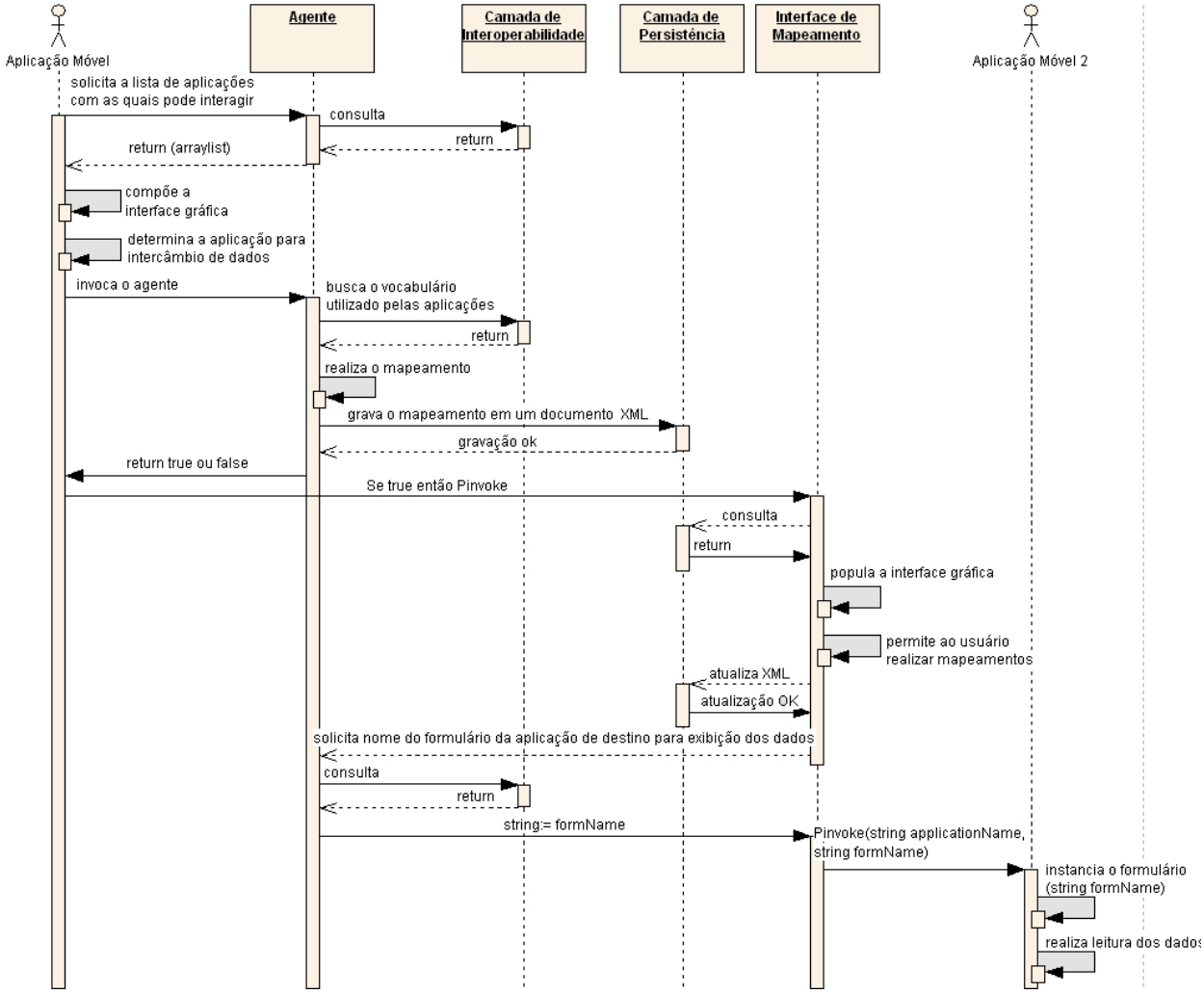


Figura 5.7: Diagrama de interação de atuação do agente

Finalmente, confirmado o processo de integração, a aplicação de destino é invocada e recebe como parâmetro o nome do formulário para exibição dos dados, o qual tem a responsabilidade de realizar a leitura da informação.

5.1.1.2 Modelo de caso de uso

O modelo exibido através da Figura 5.8 especifica as operações realizadas pelo agente *Web service* no contexto do problema da integração entre aplicações móveis utilizadas no domínio de ambientes de *M-Learning*. Os métodos implementados por este serviço possibilitam: uma pesquisa a ontologia armazenada na camada de interoperabilidade com a finalidade de obter informações sobre as aplicações que farão o intercâmbio de dados (UC1); o armazenamento destas informações na camada de persistência e notificação da aplicação de origem dos dados (UC2); a leitura dos dados armazenados na camada de persistência pela interface de mapeamento (UC3); a modificação das propriedades identificadas pelo agente no UC1 (UC4); e a identificação do método que deverá ser instanciado pela aplicação de destino dos dados (UC5).

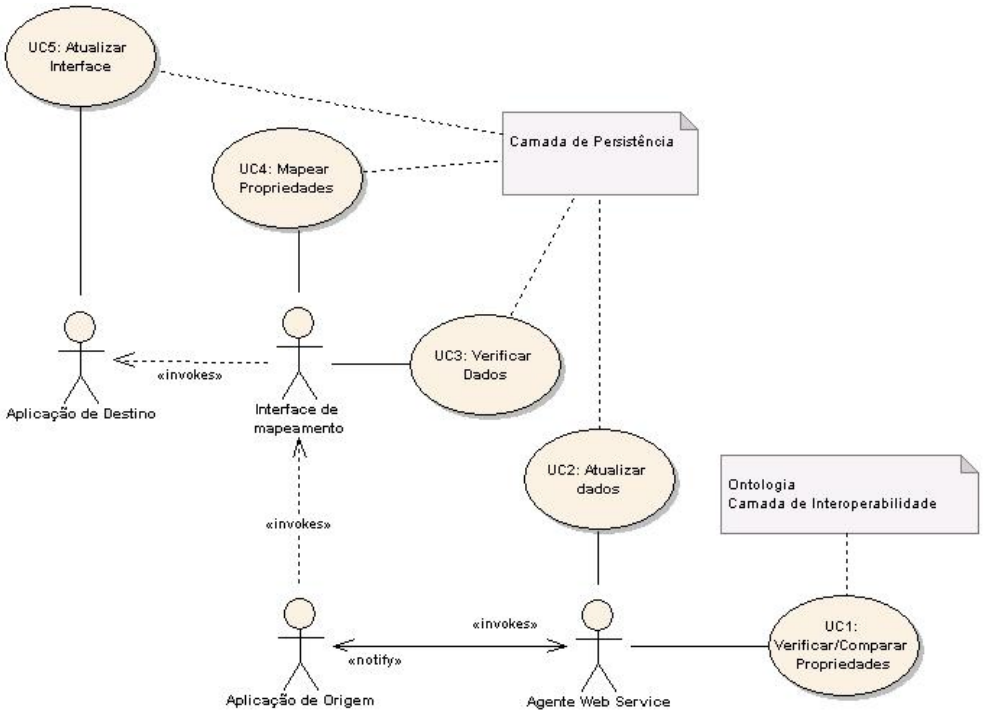


Figura 5.8: Diagrama de casos de uso do agente *Web service*

5.1.1.3 Modelo de classes

A Figura 5.9 mostra o modelo de classes do agente *Web service*, que foi modelado através do padrão de projeto *Proxy*.

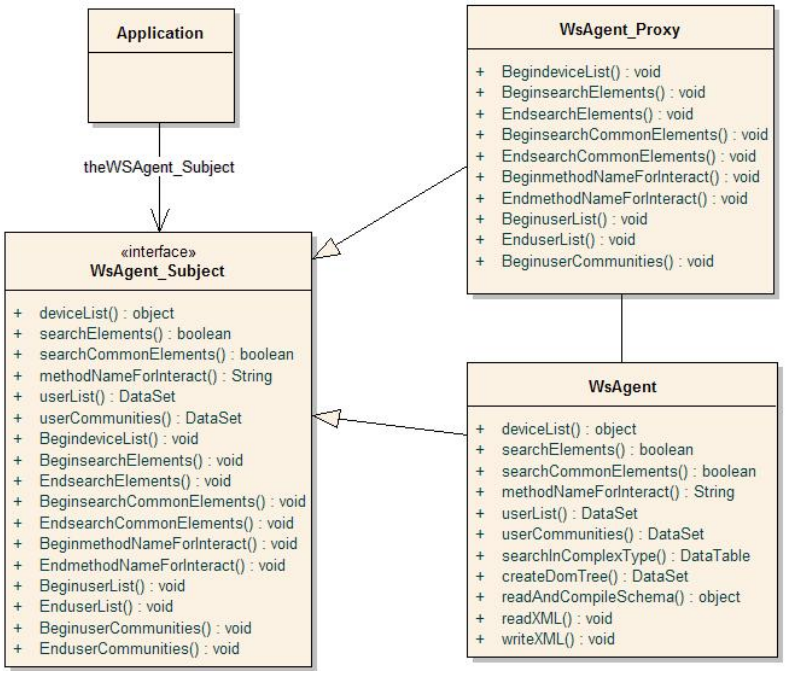


Figura 5.9: Diagrama de classes do agente *Web service*

É importante salientar que a classe "WSAgent" corresponde ao *Web service*, e contém os métodos que são invocados pelas aplicações móveis (na plataforma .NET chamados de *Web Methods*) e métodos utilizados internamente. Por sua vez, a classe "WsAgent_Subject" consiste em uma classe abstrata e define as interfaces do agente. Por último, a classe "WsAgent_Proxy" contém os ponteiros para invocação do *Web service*. Os métodos pertinentes a este serviço serão descritos na seção a seguir.

5.1.1.4 Métodos *da Web*

Foram desenvolvidos seis métodos da *Web* para o serviço "WsAgent": "deviceList", para obtenção da lista de aplicações passíveis de integração; "searchElements", para consulta a camada de interoperabilidade em busca das propriedades de cada aplicação; "searchCommonElements", para identificação de similaridades entre as aplicações; "methodNameForInteract", para identificação do formulário que deverá exibir os dados de intercâmbio; "userList" para identificação dos usuários de uma comunidade específica; e "userCommunities" para identificação das comunidades às quais o usuário pertence. Os tópicos a seguir descrevem mais detalhadamente cada um destes métodos:

1. **deviceList**: este método busca, através de uma consulta à camada de interoperabilidade, a lista de aplicativos com os quais a aplicação que o invocou consegue interagir, sendo que os parâmetros de entrada e saída do mesmo, entre outras informações, são apresentadas na Figura 5.10.

WsAgent - deviceList

SOAP

A seguir é apresentada uma amostra de solicitação e de resposta SOAP. Os **espaços reservados** mostrados são substituídos por valores reais.

```
POST /WsAgent/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/deviceList"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <deviceList xmlns="http://tempuri.org/">
      <device>string</device>
    </deviceList>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <deviceListResponse xmlns="http://tempuri.org/">
      <deviceListResult>
        <anyType />
        <anyType />
      </deviceListResult>
    </deviceListResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 5.10: Descrição do *Web method deviceList*

As informações principais para uso deste método são:

- Nome: deviceList
- URL do serviço: http://10.16.165.95/wsagent/wsagent.asmx
- Parâmetro de entrada: *String*
- Parâmetro de saída: *arraylist* contendo a "lista de aplicações"

Por sua vez, a Figura 5.11 apresenta o código fonte (em .NET C#) com a definição deste serviço A implementação realiza uma pesquisa a camada de interoperabilidade para identificação das aplicações que podem ser integradas com a aplicação no parâmetro de entrada.

```
public ArrayList deviceList(string device)
{
    XmlSchema schemaOntologia = ReadAndCompileSchema("OntoInterop2.xsd");
    ArrayList listaOp = new System.Collections.ArrayList();
    foreach (XmlSchemaObject xso in schemaOntologia.Items)
    {
        if (xso is XmlSchemaElement)
        {
            XmlSchemaElement elem = (XmlSchemaElement)xso;
            if (elem.QualifiedName.Name.Equals(device))
            {
                XmlSchemaComplexType custType = (XmlSchemaComplexType)elem.ElementType;
                XmlSchemaSequence children = (XmlSchemaSequence)custType.Particle;
                foreach(XmlSchemaParticle p1 in children.Items)
                {
                if (p1 is XmlSchemaElement)
                    {
                        XmlSchemaElement childElem = (XmlSchemaElement)p1;
                        if (childElem.Name.Equals("interactWith"))
                        {
                            XmlSchemaComplexType lista = (XmlSchemaComplexType)childElem.ElementType
                            XmlSchemaSequence listaChildren = (XmlSchemaSequence)lista.Particle;
                            foreach (XmlSchemaParticle p2 in listaChildren.Items)
                            {
                                if (p2 is XmlSchemaElement)
                                {
                                    XmlSchemaElement aplicacoes = (XmlSchemaElement)p2;
                                    listaOp.Add(aplicacoes.Name.ToString());
                                }
                            }
                            break;
                        }
                    }
                }
            }
        }
    }
    return listaOp;
}
```

Figura 5.11: Fonte do *Web method deviceList*

2. **searchElements**: este método busca, através de uma consulta à camada de interoperabilidade, a relação dos atributos e respectivos *dataTypes* de uma aplicação específica instanciada pelo usuário. A Figura 5.12 demonstra os parâmetros de entrada e saída do mesmo entre outras informações.

WsAgent - searchElements

SOAP

A seguir é apresentada uma amostra de solicitação e de resposta SOAP. Os **espaços reservados** mostrados são substituídos por valores reais.

```
POST /WsAgent/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/searchElements"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchElements xmlns="http://tempuri.org/">
      <device>string</device>
    </searchElements>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchElementsResponse xmlns="http://tempuri.org/">
      <searchElementsResult>boolean</searchElementsResult>
    </searchElementsResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 5.12: Descrição do *Web method searchElements*

As informações principais para uso deste método são:

- Nome: searchElements
- URL do serviço: http://10.16.165.95/wsagent/wsagent.asmx
- Parâmetro de entrada: *String* "nome da aplicação"
- Parâmetro de saída: *boolean*

O código fonte (em .NET C#) com a definição do serviço é apresentado na Figura 5.13, pode-se perceber que a implementação realiza uma pesquisa à camada de interoperabilidade para identificação dos atributos e respectivos *dataTypes*. O fato de ser uma estrutura em árvore exige a implementação de um método recursivo (**searchInComplexType**), de forma que todos os elementos diretamente ou indiretamente ligados ao nodo recebido como parâmetro sejam incluídos na pesquisa, sendo que estas informações são posteriormente armazenadas em um XML na camada de persistência.

```
public bool searchElements(string device)
{try
    {
        DataTable tbElemsDevice = createDataTable("elementsOfDevice");
        DataRow myDataRow;
        DataSet ds = new DataSet("dsElemsDevice");
        XmlSchema schemaOntologia = ReadAndCompileSchema("OntoInterop4.xsd");
        foreach (XmlSchemaObject xso in schemaOntologia.Items)
        {if (xso is XmlSchemaElement)
            {
                XmlSchemaElement element = (XmlSchemaElement)xso;
                if (element.Name.Equals(device))
                {
                    XmlSchemaComplexType ct = (XmlSchemaComplexType) element.ElementType;
                    XmlSchemaSequence sequence = (XmlSchemaSequence)ct.Particle;
                    foreach(XmlSchemaParticle p1 in sequence.Items)
                    {if (p1 is XmlSchemaElement)
                        {
                            XmlSchemaElement childElement = (XmlSchemaElement)p1;
                            if (childElement.ElementType is XmlSchemaComplexType)
                            {tbElemsDevice = searchInComplexTtype(childElement,tbElemsDevice);}
                            else if (!childElement.Name.Equals("methodNameForInteract"))
                            {
                                myDataRow = tbElemsDevice.NewRow();
                                myDataRow["element"] = childElement.Name;
                                myDataRow["dataType"] = childElement.SchemaTypeName.Name;
                                tbElemsDevice.Rows.Add(myDataRow);
                            }
                        }
                    }
                    ds.Tables.Add(tbElemsDevice);
                }
            }
        }
        writeXML(ds, device);
    }catch(Exception)
    {return false;}
    return true;
}
```

Figura 5.13: Fonte do *Web method searchElements*

3. **searchCommonElements**: Este método busca, através de uma consulta à camada de interoperabilidade, similaridades entre os atributos da aplicação de origem dos dados e a aplicação de destino, e tem como formato da resposta uma variável *booleana* que indica se o agente foi capaz ou não de identificar estas equivalencias. As informações principais para uso deste método são:

    - Nome: searchCommonElements
    - URL do serviço: http://10.16.165.95/wsagent/wsagent.asmx
    - Parâmetros de entradas: *String* "nome da aplicação de origem", *String* "nome da aplicação de destino"
    - Parâmetro de saída: *boolean*

    Tais parâmetros, entre outras informações, são apresentados na Figura 5.14.

WsAgent - searchCommonElements

SOAP

A seguir é apresentada uma amostra de solicitação e de resposta SOAP. Os **espaços reservados** mostrados são substituídos por valores reais.

```
POST /WsAgent/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/searchCommonElements"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchCommonElements xmlns="http://tempuri.org/">
      <device_1>string</device_1>
      <device_2>string</device_2>
      <xd>xml</xd>
    </searchCommonElements>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchCommonElementsResponse xmlns="http://tempuri.org/">
      <searchCommonElementsResult>boolean</searchCommonElementsResult>
    </searchCommonElementsResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 5.14: Descrição do *Web method searchCommonElements*

A seguir, na Figura 5.15, é apresentado o código fonte (em .NET C#) com a definição do serviço **searchCommonElements**, cuja implementação realiza uma pesquisa à camada de interoperabilidade para identificação dos atributos relacionados á aplicação que o invocou e as armazena em uma estrutura em memória; posteriormente, ele volta a pesquisar para determinar quais os elementos da aplicação de destino são equivalentes a esta estrutura, e finalmente, o resultado desta pesquisa é armazenada em arquivos com extensão XML na camada de persistência.

```
public bool searchCommonElements(string device_1, string device_2, XmlDocument xd)
{
    #region
    foreach (XmlSchemaObject xso in schemaOntologia.Items)
    {
        if (xso is XmlSchemaElement)
        {
            variáveis locais
            if (elem.QualifiedName.Name.Equals(device_2))
            {
                DataRow[] adr = dados.Tables["Dados"].Select();
                foreach(XmlSchemaParticle p2 in children2.Items)
                {
                    if (p2 is XmlSchemaElement)
                    {
                        childElem2 = (XmlSchemaElement)p2;
                        foreach (DataRow dr in adr)
                        {
                            if ((tabelaDisp1.ContainsKey(childElem2.Name))
                                &&(tabelaDisp1.ContainsValue(childElem2.ElementType))
                                &&(childElem2.Name.Equals(dr[0])))
                            {

                                myDataRow = myDataTable.NewRow();
                                myDataRow["elementOne"] = childElem2.Name;
                                myDataRow["elementTwo"] = childElem2.Name;
                                myDataRow["information"] = dr[1];
                                myDataTable.Rows.Add(myDataRow);

                            }
                        }
                    }
                }
                ds.Tables.Add(myDataTable);
            }
        }
    }
    writeXML(ds,device_1+"to"+device_2);
    return true;
```

Figura 5.15: Fonte do *Web method searchCommonElements*

4. **methodNameForInteract**: este método busca, através de uma consulta á camada de interoperabilidade, o nome do formulário que exibirá os dados provenientes da aplicação que invocou o agente. As informações principais para uso deste método são:

   - Nome: methodNameForInteract
   - URL do serviço: http://10.16.165.95/wsagent/wsagent.asmx
   - Parâmetro de entrada: *String* "nome da aplicação de destino"
   - Parâmetro de saída: *string*

   Os parâmetros relacionados, entre outras informações, são apresentados na Figura 5.16.

WsAgent - methodNameForInteract

SOAP

A seguir é apresentada uma amostra de solicitação e de resposta SOAP. Os **espaços reservados** mostrados são substituídos por valores reais.

```
POST /WsAgent/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/methodNameForInteract"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <methodNameForInteract xmlns="http://tempuri.org/">
      <device>string</device>
    </methodNameForInteract>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <methodNameForInteractResponse xmlns="http://tempuri.org/">

<methodNameForInteractResult>string</methodNameForInteractResult>
    </methodNameForInteractResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 5.16: Descrição do *Web method methodNameForInteract*

A seguir, é apresentado através da Figura 5.17 o código fonte (em .NET C#) com a definição do serviço **methodNameForInteract**, cuja a implementação realiza novamente uma pesquisa à camada de interoperabilidade, e ao localizar o elemento recebido como parâmetro de entrada, verifica o valor fixado á *tag* **methodNameForInteract** (ex.: <xs:element name = "methodNameForInteract" type = string fixed = "insertEvent">), sendo que tal valor indica o nome do formulário que deverá ser instanciado.

```
public string methodNameForInteract(string device)
{

      XmlSchema schemaOntologia = ReadAndCompileSchema("OntoInterop2.xsd");
      string methodNameForInteract="";

      foreach (XmlSchemaObject xso in schemaOntologia.Items)
      {
            if (xso is XmlSchemaElement)
            {
                  XmlSchemaElement elem = (XmlSchemaElement)xso;

                  if (elem.QualifiedName.Name.Equals(device))
                  {
                        XmlSchemaComplexType custType = (XmlSchemaComplexType)elem.ElementType;
                        XmlSchemaSequence children = (XmlSchemaSequence)custType.Particle;
                        foreach(XmlSchemaParticle p1 in children.Items)
                        {

                              if (p1 is XmlSchemaElement)
                              {
                                    XmlSchemaElement childElem = (XmlSchemaElement)p1;

                                    if (childElem.Name.Equals("methodNameForInteract"))

                                    {
                                          return childElem.FixedValue.ToString();

                                    }

                              }
                        }
                  }
            }
      }

      return methodNameForInteract;
}
```

Figura 5.17: Fonte do *Web Method methodNameForInteract*

5. **userList**: este método busca, através de uma consulta à base de dados armazenada na camada de persistência, o nome dos participantes de uma comunidade específica. As informações principais para uso deste método são:

   - Nome: userList
   - URL do serviço: http://10.16.165.95/wsagent/wsagent.asmx
   - Parâmetro de entrada: *String* "nome da aplicação que invocou o método"
   - Parâmetro de saída: *dataSet*

   Os parâmetros de entrada e saída do mesmo, entre outras informações, são apresentados na Figura 5.18.

WsAgent - userList

SOAP

A seguir é apresentada uma amostra de solicitação e de resposta SOAP. Os **espaços reservados** mostrados são substituídos por valores reais.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <userList xmlns="http://tempuri.org/">
      <device>string</device>
    </userList>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <userListResponse xmlns="http://tempuri.org/">
      <userListResult>
        <xsd:schema>schema</xsd:schema>xml</userListResult>
    </userListResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 5.18: Descrição do *Web method userList*

A Figura 5.19 abaixo apresenta o código fonte (em .NET C#) com a definição do serviço **userList**, cuja a implementação executa uma consulta á base de dados existente na camada de persistência com a finalidade de obter a lista de usuários de um determinado dispositivo cliente.

```
public DataSet userList(string device)
{
      SqlConnection conn = new SqlConnection("Data Source=10.16.165.206;Initial
                           Catalog=WSMEL_Pocket;User Id=sa;Password=;");
      string sql = "select usuario.nome, usuario.sobrenome from usuario,listaObjInstalados where
                   listaObjInstalados.idUsuario = usuario.idUsuario and listaObjInstalados.idAplicacao
                   =  (select idAplicacao from aplicacao where nomeAplicacao='"+device+"') order by
                   usuario.nome;";
      SqlDataAdapter da;
      DataSet ds;
      try
      {
            conn.Open();
            da = new SqlDataAdapter(sql,conn);
            ds = new DataSet();
            da.Fill(ds,"ComunityMember:-");
            return ds;
      }

      finally
      {
            conn.Close();
      }

}
```

Figura 5.19: Fonte do *Web method userList*

6. **userCommunities**: este método busca, através de uma consulta à base de dados armazenada na camada de persistência, a lista de comunidades ás quais um usuário específico pertence. As informações principais para uso deste método são:

   - Nome: userCommunities
   - URL do serviço: http://10.16.165.95/wsagent/wsagent.asmx
   - Parâmetro de entrada: *String* "nome", *String*"sobrenome"
   - Parâmetro de saída: *dataSet*

   Os parâmetros de entrada e saída do mesmo, entre outras informações, são apresentados na Figura 5.20.

WsAgent - userCommunities

SOAP

A seguir é apresentada uma amostra de solicitação e de resposta SOAP. Os **espaços reservados** mostrados são substituídos por valores reais.

```
POST /WsAgent/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/userCommunities"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <userCommunities xmlns="http://tempuri.org/">
      <userFirstName>string</userFirstName>
      <userLastName>string</userLastName>
    </userCommunities>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <userCommunitiesResponse xmlns="http://tempuri.org/">
      <userCommunitiesResult>
        <xsd:schema>schema</xsd:schema>xml</userCommunitiesResult>
    </userCommunitiesResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 5.20: Descrição do *Web method userCommunities*

A seguir, através da Figura 5.21, é apresentado o código fonte (em .NET C#) com a definição do serviço **userCommunities**, cuja a implementação realiza, basicamente, uma pesquisa á base de dados existente na camada de persistência com a finalidade de obter a lista de comunidades de um determinado usuário.

```
public DataSet userCommunities(string userFirstName, string userLastName)
{
      SqlConnection conn = new SqlConnection("Data Source=10.16.165.206;Initial
                           Catalog=WSMEL_Pocket;User Id=sa;Password=;");
      string sql = "select a.nomeAplicacao from Aplicacao a, listaObjinstalados b where
                    b.idAplicacao = a.idAplicacao and b.idUsuario =(Select idUsuario from
                    usuario where nome ='"+userFirstName+"' and sobrenome
                    ='"+userLastName+"')";
      SqlDataAdapter da;
      DataSet ds;
      try
      {
            conn.Open();
            da = new SqlDataAdapter(sql,conn);
            ds = new DataSet();
            da.Fill(ds,"UserCommuties:");
            return ds;
      }
      finally
      {
            conn.Close();
      }

}
```

Figura 5.21: Fonte do *Web method userCommunities*

5.1.2 Camada de interoperabilidade

No contexto desta pesquisa, a camada de interoperabilidade é responsável por duas funcionalidades: a primeira consiste em permitir que agentes de *software* possam obter informações a respeito das interfaces dos dispositivos e das estruturas de dados que os *softwares* utilizam, e a segunda é servir de base a projetistas que pretendem desenvolver ou melhorar uma aplicação que possa fazer parte de um ambiente de *M-Learning*.

A ontologia fornece a estrutura padrão utilizada por aplicações inseridas no domínio dos ambientes de *M-Learning* e permite definir os metadados semânticos que "declaram" o significado, as funções e os requisitos de processamento de cada *software*. Assim, o agente instanciado deve pesquisar a ontologia para descobrir o vocabulário utilizado por cada aplicativo. Cabe ressaltar que o conceito de ontologia foi adaptado para o uso na arquitetura WSMEL, e foi utilizado para criar os conceitos (classes) que indicarão ao agente quais os atributos (*slots*) que cada *software* possui e que tipo de restrições cada um mantém. Portanto, a ontologia consiste em um esquema preliminar, ou seja, um vocabulário representado através de um XML *schema*.

A Figura 5.22 exibe o vocabulário utilizado por dois dos aplicativos inseridos na camada, no caso, "*E-mail*" e "Forum". Estes *softwares* são compostos por elementos que especificam os atributos da aplicação, e por informações fundamentais para que aplicações consigam interagir (elementos *interactWith - methodNameForInteract*), sendo que a ontologia completa encontra-se em anexo a este trabalho.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="WSMEL-Ontology">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Agenda"/>
                <xs:element ref="Email"/>
                <xs:element ref="Forum"/>
                <xs:element ref="Mural"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Email">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="userName" type="xs:string" />
                <xs:element name="userAddress" type="xs:string" />
                <xs:element name="userAddressee" type="xs:string" />
                <xs:element name="userCC" type="xs:string" />
                <xs:element name="subject" type="xs:string" />
                <xs:element name="body" type="xs:string" />
                <xs:element name="interactWith">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Agenda" type="xs:string" />
                            <xs:element name="Forum" type="xs:string" />
                            <xs:element name="Mural" type="xs:string" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="methodNameForInteract" type="xs:string" fixed="sendMail" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
<xs:element name="Forum">
    <xs:annotation>
        <xs:documentation></xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="date" type="xs:date" />
            <xs:element name="time" type="xs:time" />
            <xs:element name="subject" type="xs:string" />
            <xs:element name="body" type="xs:string" />
            <xs:element name="status" type="xs:string" />
            <xs:element name="author" type="xs:string" />
            <xs:element name="reply" type="xs:string" />
            <xs:element name="authorReply" type="xs:string" />
            <xs:element name="interactWith">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Agenda" type="xs:string" />
                        <xs:element name="Email" type="xs:string" />
                        <xs:element name="Mural" type="xs:string" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="methodNameForInteract" type="xs:string" fixed="insertFAQ" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figura 5.22: XML *schema* pertinente à camada de interoperabilidade

Para auxiliar na definição desta ontologia, foi desenvolvido um editor que cria um XML *Schema*, no qual um aplicativo é representado a partir de um nodo raiz e os elementos anexados a este nodo configuram as propriedades e as restrições relacionadas ao aplicativo que está sendo inserido na camada de interoperabilidade. Assim, o editor permite consultar, editar ou inserir um novo conceito a esta camada, sendo que, para isso, o projetista utiliza as opções disponibilizadas pela barra de menus, ferramentas e janela de propriedades (Figura 5.23). A principal vantagem proporcionada por este editor é que o simples fato de declarar uma nova aplicação na camada de interoperabilidade já proporciona a possibilidade de integração desta com outras aplicações que o usuário utiliza.

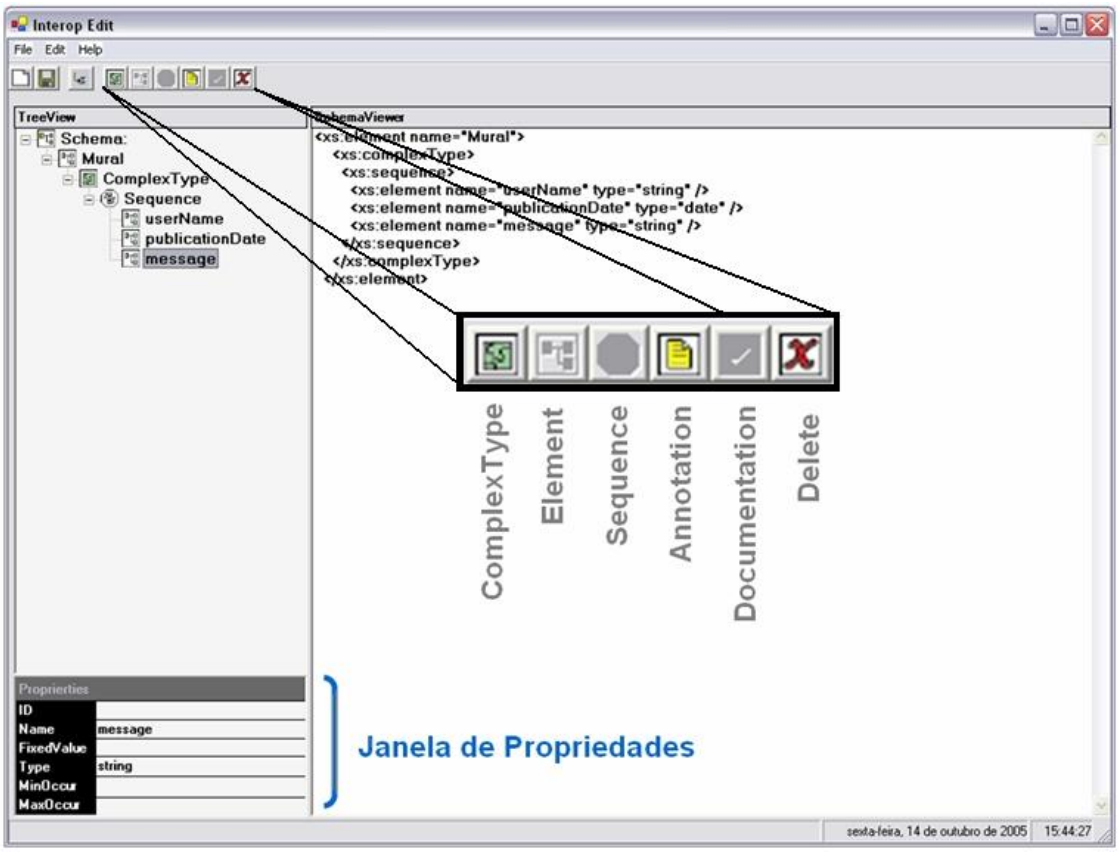


Figura 5.23: Funcionalidades do InteropEdit

Na figura 5.23, cada subjanela do editor apresenta uma única perspectiva da ontologia que está sendo editada, as quais são:

- ***TreeView***: mostra o XML *Schema* usando elementos gráficos para os nodos: elementos, atributos, tipos complexos e seqüências. "Nodos pai" podem ser expandidos para mostrar seus "elementos filhos", cujas propriedades são exibidas logo abaixo desta janela, na janela de propriedades;
- ***SchemaViewer***: mostra o texto XML utilizado para definir o *schema*;
- ***Properties***: utilizada para definir as restrições de cada elemento, tais como: ID, nome, valor fixo, tipo de dados, valor máximo e mínimo aceitos.

5.1.3 Camada de clientes

A camada de cliente engloba os *softwares* (aplicativos) que dão suporte ao processo de aprendizagem por parte do usuário, em que os dispositivos móveis são capazes de executar múltiplas aplicações, as quais apresentam ao usuário uma interface onde a informação é exibida.

Dentre as várias arquiteturas para o seu desenvolvimento, utilizou-se uma baseada no uso de *proxys*, separando as regras de negócio que governam cada aplicação em servidores remotos, e, assim, possibilitando o processamento distribuído e a diminuição de carga de processamento no dispositivo móvel (Figura 5.24). Diferentes *proxys* podem existir para diferentes aplicações, entretanto, nada impede que apontem para um único servidor, isso porque uma aplicação simples desenvolvida para incorporar esta camada deve possuir componentes que rodem localmente no dispositivo móvel, assim como componentes que rodem no servidor remoto, entre os quais o agente *Web service*.

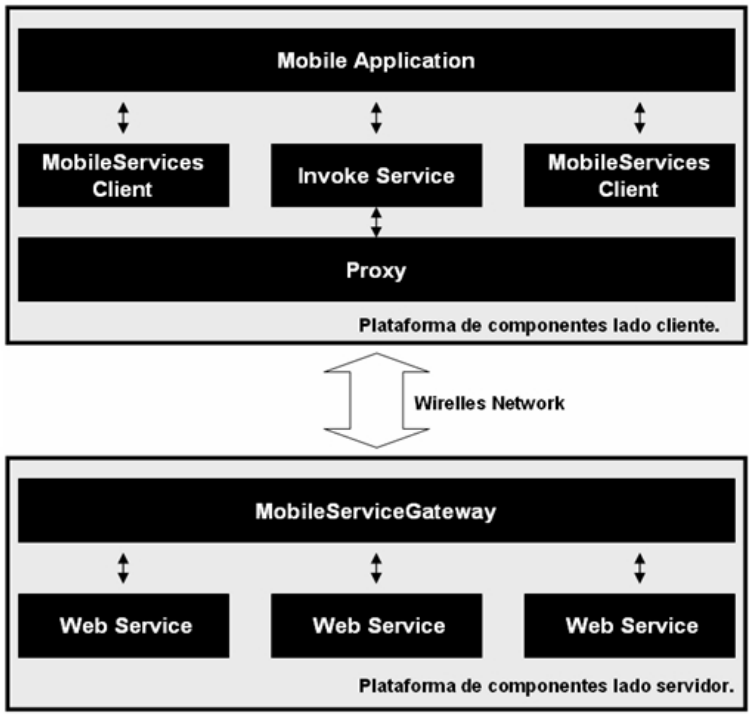


Figura 5.24: Arquitetura das aplicações cliente

A fim de validar a arquitetura WSMEL, desenvolveram-se duas aplicações classificadas como dispositivos clientes: uma "Agenda Pessoal"e uma ferramenta de "E-mail". Estas aplicações, além de acopladas ao ambiente do usuário (Figura 5.25.A), foram descritas na ontologia presente na camada de interoperabilidade, pois assim, o usuário aprendiz pode utilizá-las de maneira independente e integrá-las conforme sua necessidade. Além disso, o usuário pode acrescentar de maneira espontânea novas aplicações a este ambiente (Figura 5.25.B).

(A) Ambiente WSMEL

(B) Ambiente WSMEL acrescentado de novas aplicações

Figura 5.25: O ambiente WSMEL

5.1.3.1 Dispositivo cliente Agenda

A "Agenda" consiste em um espaço disponível para que o usuário aprendiz possa registrar eventos, tais como: reuniões, cursos, *Chat*, videoconferências, entre outros. Ao acessar a agenda, o usuário vai visualizar o calendário do mês atual e os itens agendados para o dia (Figura 5.26.A), além disso, através do calendário, eventos agendados para outros dias também podem ser verificados (Figura 5.26.B).

(A) Dispositivo Cliente Agenda

(B) Acesso ao Calendário

Figura 5.26: Dispositivo cliente Agenda

O modelo descrito abaixo especifica as operações que podem ser realizadas pelos usuários no contexto da utilização deste *software*.

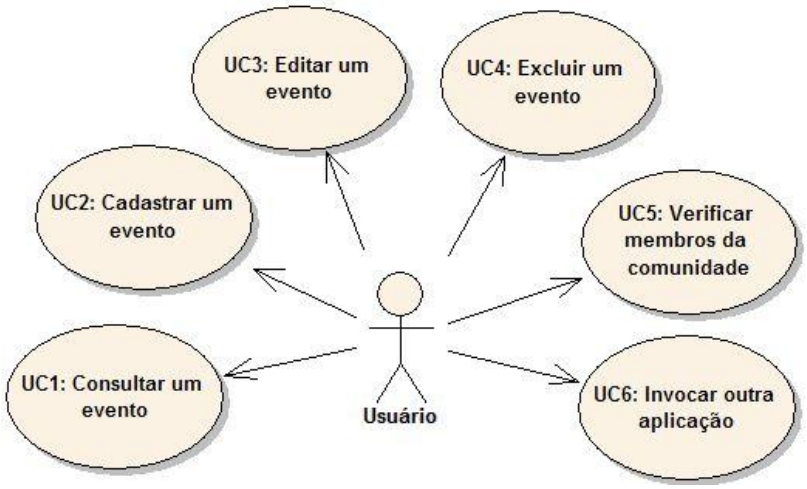


Figura 5.27: Diagrama *use/case* do *software* Agenda

Para descrição do diagrama de casos de uso demonstrado pela Figura 5.27 optou-se pelo modelo casual recomendado por (Cockburn, 2001), sendo que os quadros a seguir descrevem cada um dos seis casos de uso.

Tabela 5.1: UC1 Agenda - Consultar eventos

Descrição resumida: permite ao usuário do aplicativo consultar os eventos que possui cadastrado para uma data e hora específica.
Fluxo de eventos: o caso de uso começa quando o usuário seleciona uma data no calendário exibido pelo formulário principal.
Fluxo básico:
a. o usuário seleciona uma data;
b. o sistema exibe o formulário com os eventos cadastrados para a data selecionada;
c. o usuário pode editar, excluir o evento, ou integrar os dados do evento com outro aplicativo inserido no ambiente.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet e entrar no sistema.
Pós-condições: não se aplica

Tabela 5.2: UC2 Agenda - Cadastrar um evento

Descrição resumida: permite ao usuário do aplicativo inserir um novo evento a base de dados, sendo que o sistema verifica se já existem eventos cadastrados na data e horário determinados antes de efetivar o procedimento.
Fluxo de eventos: o caso de uso começa quando o usuário seleciona a opção "*Add Event*" do formulário principal.
Fluxo básico:
a. o usuário seleciona a opção "*Add Event*";
b. o sistema exibe o formulário com os campos para preenchimento;
c. o usuário preenche o formulário;
d. o sistema verifica se já existe um evento cadastrado para a data e horário informados;
e. o sistema insere o evento na base de dados.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão a rede sem fio com acesso a Internet e entrar no sistema.
Pós-condições: não se aplica.

Tabela 5.3: UC3 Agenda - Editar um evento

Descrição Resumida: permite ao usuário editar um evento cadastrado na base de dados.
Fluxo de Eventos: o caso de uso começa quando o usuário seleciona um dos eventos e clica na opção "*Edit Event*".
Fluxo Básico:
a. o usuário seleciona um evento a partir do formulário principal;
b. posteriormente seleciona a opção "*Edit Event*";
c. o sistema exibe o formulário com os campos preenchidos anteriormente;
d. o usuário pode, então, alterar estas informações;
e. o sistema verifica se já existe um evento cadastrado para a data e horário informados;
f. o sistema altera os atributos relacionados ao evento na base de dados.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet, entrar no sistema e possuir eventos cadastrados na base de dados.
Pós-condições: não se aplica.

Tabela 5.4: UC4 Agenda - Excluir um evento

Descrição resumida: permite a exclusão de um evento da base de dados.
Fluxo de eventos: o caso de uso começa quando o usuário seleciona um dos eventos e clica na opção "*Del Event*".
Fluxo básico:
a. o usuário seleciona um evento a partir do formulário principal;
b. posteriormente seleciona a opção "*Del Event*";
c. o sistema exibe o formulário para confirmação de exclusão;
d. o usuário confirmar a solicitação;
e. o sistema exclui o evento da base de dados.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet, entrar no sistema e possuir eventos inseridos na base de dados.
Pós-condições: não se aplica.

Tabela 5.5: UC5 Agenda - Verificar membros da comunidade

Descrição resumida: permite ao usuário do aplicativo conhecer os demais usuários que também usam este aplicativo.
Fluxo de eventos: o caso de uso começa quando o estudante seleciona a opção "*Community*" do formulário principal.
Fluxo básico:
a. o usuário seleciona a opção "*Community - Verify Members*";
b. o sistema exibe um formulário com a lista de usuários que também usam o aplicativo;
c. o usuário pode então consultar quem são estes usuários e que outros aplicativos os mesmos usam para comunicar-se dentro do ambiente WSMEL.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet e entrar no sistema.
Pós-condições: não se aplica.

Tabela 5.6: UC6 Agenda - Invocar outra aplicação.

Descrição resumida: permite o intercâmbio de dados locais com outra aplicação incorporada ao ambiente WSMEL.
Fluxo de eventos: o caso de uso começa quando o usuário seleciona um evento, e através do menu principal clica em "*Data - Send To*".
Fluxo básico:
a. o usuário seleciona um evento a partir do formulário principal;
b. o usuário seleciona a opção "*Data - Send To*";
c. o sistema exibe um formulário com os dados que podem ser integrados;
d. o usuário confirma esta informação ou realiza outras associações;
e. o sistema invoca a aplicação de destino dos dados;
f. a aplicação de destino exibe o formulário apropriado com os dados recebidos.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet e entrar no sistema.
Pós-condições: não se aplica.

O modelo de classes para o *software* Agenda pode ser verificado através da Figura 5.28. A aplicação é composta pelas classes "Agenda, ControlFaçade, AbstractWsAgenda, Abstract WSAgent e DateTimePicker", e foi modelada sob a ótica de três padrões de projeto, os quais são: MVC, *Façade* e *Proxy* (Metsker, 2004).

A classe "Agenda" corresponde á *view* no padrão MVC e é responsável pela renderização dos objetos, ou seja, ela representa a interface gráfica que é visualizada pelo usuário no momento em que este instancia a aplicação. Por sua vez, "ControlFaçade" representa um controlador e define a maneira como a interface do usuário reage à entrada do mesmo, e é responsável pelas chamadas aos *Web services*, os quais contém tanto as regras de negócio que governam a aplicação quanto o agente que realiza o intercâmbio de dados; trata-se, portanto, de uma fachada para invocação destes *Web services*. Já as classes "AbstractWsAgenda" e "AbstractWsAgent" consistem em interfaces de acesso a *Web services* e contém ponteiros para invocação dos mesmos.

"WsAgenda" contém as regras de negócio ligadas á aplicação, e possui métodos para inserção, alteração e exclusão de eventos na base de dados do usuário. Além disso, "WsAgent" contém as regras ligadas ao agente de *software* e possui métodos para o intercâmbio de dados entre aplicações, e para acesso à lista de usuários da comunidade do dispositivo cliente em questão.

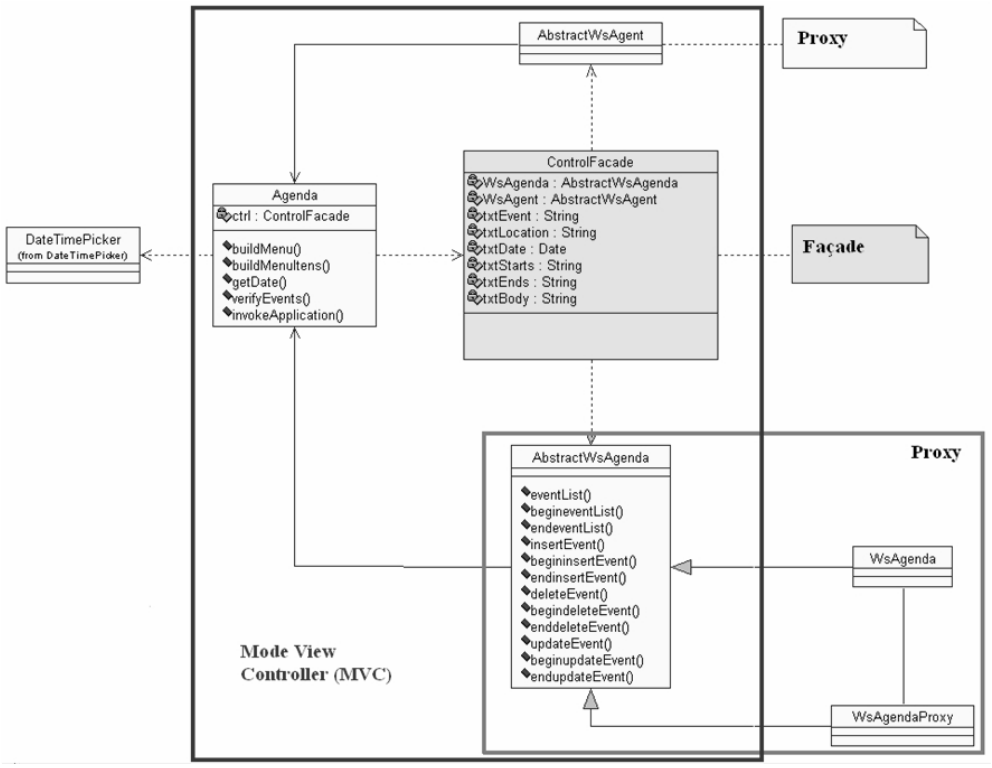


Figura 5.28: Modelo de Classes do *software* Agenda

5.1.3.2 Dispositivo cliente *E-mail*

O dispositivo cliente "*E-mail*" consite em um recurso que possibilita a troca de mensagens de forma rápida e versátil entre usuários do ambiente de *M-Learning*. Ao acessar esta aplicação, o usuário visualizará a sua caixa de entrada, permitindo a leitura de suas mensagens. O modelo de casos de uso abaixo especifica as operações que podem ser realizadas pelos usuários no contexto da utilização deste *software*.

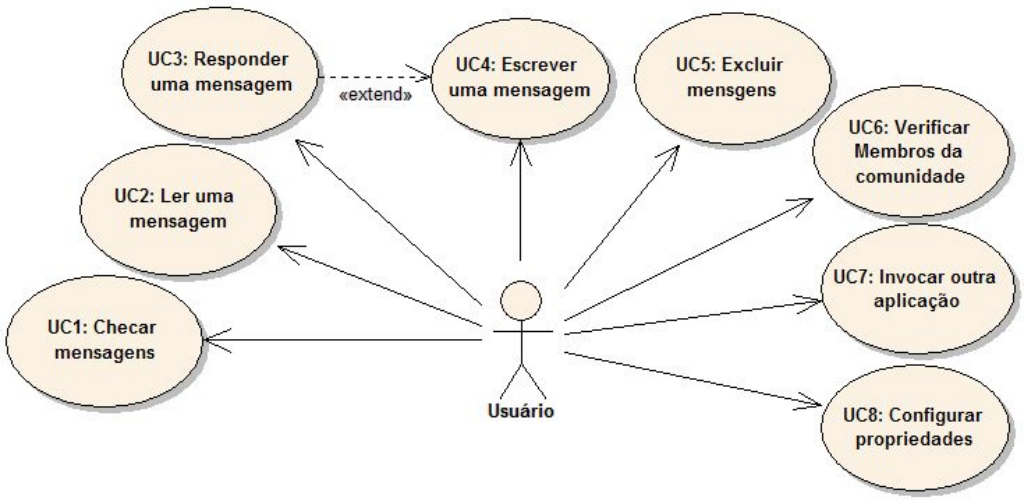


Figura 5.29: Diagrama use/case do *software E-mail*

Tabela 5.7: UC1 *E-mail* - Checar mensagens.

Descrição resumida: esse caso de uso é iniciado pelo sistema no mesmo momento em que o usuário instancia o aplicativo.
Fluxo básico: o sistema acessa o servidor de correio eletrônico, realiza o *download* das mensagens e exibe no formulário principal (Figura 5.30.A).
Requisitos especiais: o servidor de correio eletrônico deve suportar o protocolo POP3.
Pré-condições: possuir uma conexão a rede sem fio com acesso a Internet e entrar no sistema.
Pós-condições: não se aplica.

Tabela 5.8: UC2 *E-mail* - Ler uma mensagem.

Descrição resumida: permite a leitura de uma mensagem específica.
Fluxo básico:
a. o usuário seleciona uma mensagem a partir do formulário principal;
b. posteriormente clica no botão "*Read mail*";
c. o sistema exibe o conteúdo da mensagem através do formulário exibido na Figura 5.30.B. e permite ao usuário responder ao remetente.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão a rede sem fio com acesso à Internet, entrar no sistema e selecionar uma das mensagens da caixa de entrada.
Pós-condições: não se aplica.

Tabela 5.9: UC3 *E-mail* - Responder uma mensagem.

Descrição resumida: permite ao usuário responder ao remetente da mensagem.
Fluxo básico:
a. o usuário seleciona uma mensagem a partir do formulário principal;
b. posteriormente clica no botão"*Read mail*";
c. o sistema mostra o conteúdo da mensagem através do formulário exibido na Figura 5.30.B;
d. o usuário clica no botão "*Reply*";
e. o sistema faz o envio da mensagem ao servidor de correio eletrônico.
Requisitos especiais: comunicação via protocolo *Simple Mail Transfer Protocol* (SMTP).
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet, entrar no sistema e selecionar uma das mensagens da caixa de entrada.
Pós-condições: não se aplica.

Tabela 5.10: UC4 *E-mail* - Escrever uma mensagem.

Descrição resumida: permite a escrita e envio de uma mensagem de correio eletrônico a um ou mais destinatários. **Fluxo básico:** a. o usuário seleciona a *tab* "*New Message*" a partir do formulário principal; b. ao término clica em "*Send*"; c. o sistema envia a mensagem ao servidor de correio eletrônico. **Requisitos especiais:** suporte à SMTP. **Pré-condições:** possuir uma conexão à rede sem fio com acesso à Internet, entrar no sistema e selecionar uma das mensagens da caixa de entrada.

Tabela 5.11: UC4 *E-mail* - Excluir mensagens.

Descrição resumida: permite ao usuário excluir uma ou mais mensagens. **Fluxo básico:** a. o usuário seleciona a(s) mensagem(ens) que deseja excluir a partir do formulário principal, clica no botão "*Remove*"; b. o sistema exclui esta(s) mensagem(ens) do servidor de correio eletrônico. **Requisitos especiais:** não se aplica. **Pré-condições:** possuir uma conexão a rede sem fio com acesso a Internet, entrar no sistema.

Tabela 5.12: UC5 *E-mail* - Verificar membros da comunidade

Descrição resumida: permite ao usuário conhecer os demais usuários que também usam este aplicativo. **Fluxo de eventos:** o caso de uso começa quando o usuário seleciona a opção "*Community*" do formulário principal. **Fluxo Básico:** a. o usuário seleciona a opção "*Community - Verify Members*" do formulário principal; b. o sistema exibe um formulário com a lista de usuários que também usam o aplicativo; c. o usuário consulta quem são estes usuários e que outros aplicativos os mesmos usam para comunicar-se dentro do ambiente WSMEL. **Requisitos especiais:** não se aplica. **Pré-condições:** possuir uma conexão a rede sem fio com acesso a Internet, entrar no sistema.

(A) Caixa de Entrada (B) Interface gráfica para visualização de conteúdo

Figura 5.30: *Software E-mail*

Tabela 5.13: UC6 *E-mail* - Invocar outra aplicação.

Descrição resumida: permite o intercâmbio de dados locais com outra aplicação acoplada ao ambiente WSMEL.
Fluxo de eventos: o caso de uso começa quando o usuário seleciona um evento, e clica em "*Data-Send To*".
Fluxo Básico:
a. o usuário seleciona uma mensagem a partir do formulário principal;
b. posteriormente seleciona a opção "*Data-Send To*";
c. o sistema exibe um formulário com os dados que podem ser integrados;
d. o usuário confirma esta informação ou ainda realizar outras associações;
e. o sistema invoca a aplicação de destino dos dados;
f. a aplicação de destino exibe o formulário apropriado com os dados recebidos.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão a rede sem fio com acesso a Internet, entrar no sistema.
Pós-condições: não se aplica.

Tabela 5.14: UC8 *E-mail* - Configurar propriedades.

Descrição resumida: esse caso de uso permite ao usuário configurar as propriedades de acesso ao servidor de correio eletrônico.
Fluxo básico:
a. o usuário seleciona o *menu* "*config*" (Figura 5.31.B) e preenche o formulário de configuração com as informações apropriadas;
b. o sistema guarda estas informações em um arquivo XML.
Requisitos especiais: não se aplica.
Pré-condições: possuir uma conexão à rede sem fio com acesso à Internet, entrar no sistema.
Pós-condições: não se aplica.

(A) Interface gráfica para escrita de mensagens

(B) Interface gráfica para configuração de caixa postal

Figura 5.31: *Software E-mail* - Interfaces Gráficas

O modelo de classes para o *software E-mail* pode ser verificado na Figura 5.32. Esta aplicação, assim como o dispositivo cliente Agenda, também foi modelada através dos padrões de projeto *Proxy*, *Façade* e MVC. A classe **Email** renderiza as informações que são exibidas ao usuário, e o **Controller** consiste em uma fachada de acesso aos *Web services* que governam a aplicação e ao agente de *software*, ambos invocados através de um *Proxy*.

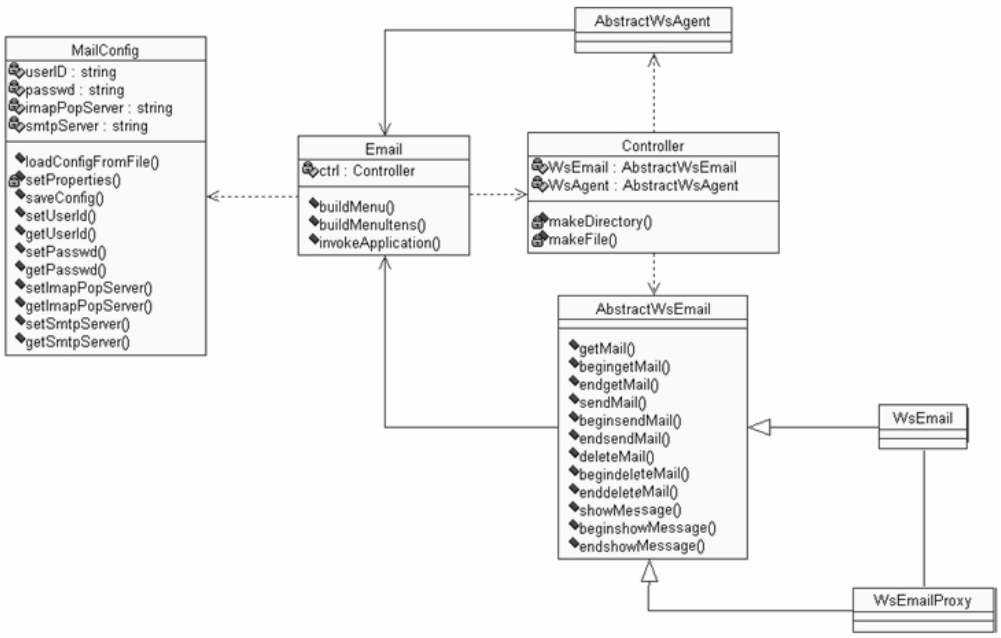


Figura 5.32: Modelo de classes do *software E-mail*

5.1.4 Camada de persistência

Na arquitetura WSMEL, a camada de persistência detém os dados manipulados pelas aplicações e as informações que o agente *Web service* identifica durante o processo de integração. É formada pela base de dados ilustrada através do diagrama "entidade - relacionamento" (ER), exibido na Figura 5.34 e por um repositório de arquivos XML, sendo que para armazenamento da base de dados foi utilizado o *Microsoft SQL Server Personal Edition*.

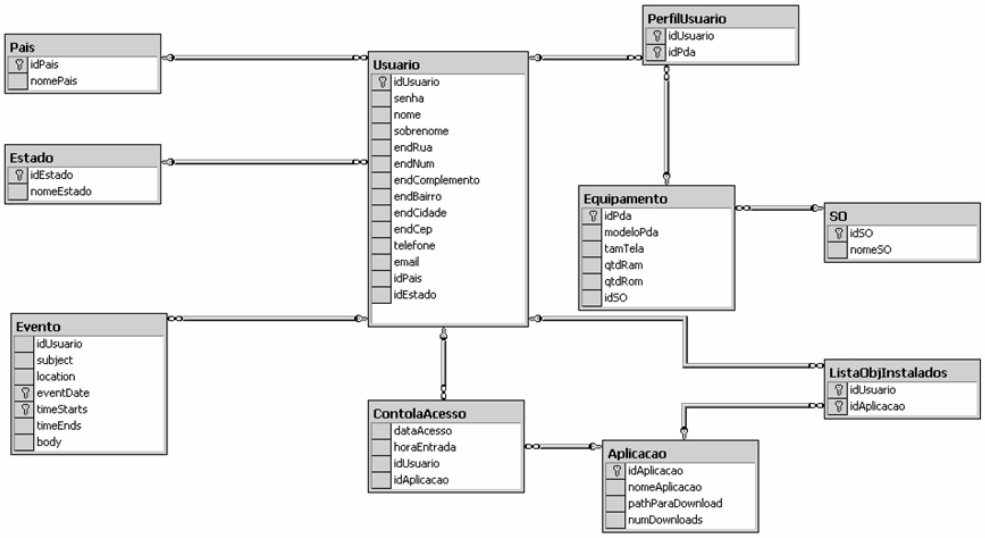


Figura 5.33: Diagrama ER da camada de persistência

A base de dados armazena informações sobre usuários, capturadas através do portal do projeto no momento em que o usuário faz o *download* de aplicações. As aplicações que cada usuário agregou a seu ambiente de ensino ficam armazenadas na tabela **listaObjInstalados**, que é consultada pelo agente para identificar as comunidades a qual o usuário pertence. A mesma base foi utilizada para guardar informações pertinentes ao *software* Agenda, protótipo desenvolvido como mecanismo de verificação da arquitetura.

5.2 Comunidades virtuais espontâneas

As comunidades virtuais em um ambiente de ensino à distância surgem de uma ação consciente, de uma infra-estrutura e de uma ação planejada que dá o impulso inicial à composição de um grupo. Portanto, essas comunidades vão se formando a partir de um agente articulador, que consegue reunir em torno de si, e converter em adeptos, um pequeno grupo de agentes (Silva, 2005), ou seja, na maioria dos casos, o professor cria uma comunidade, e seus alunos obrigam-se a utilizar os recursos que são oferecidos naquele ambiente de ensino. No entanto, o processo de formação da comunidade virtual, desta maneira, não ocorreu espontâneamente como seria a maneira ideal.

Por isso, a proposta do WSMEL é que as comunidades sejam formadas a partir dos interesses pessoais dos usuários por aplicativos e por material educacional, ou seja, que as comunidades sejam formadas de maneira "espontânea", sem que um articulador especifique em qual comunidade os usuários devem entrar e que recursos utilizar. Desta forma, as comunidades possuem relação direta com os recursos e objetos educacionais aos quais o usuário demonstrou interesse. Por exemplo, ao realizar o *download* do *software* Agenda, o usuário passa a fazer parte do grupo de pessoas que tem acesso a este *software*, ou a comunidade Agenda (Figura 5.34). Assim, o usuário não foi obrigado a utilizar este recurso, mas, de alguma forma, sente-se à vontade em fazê-lo.

Figura 5.34: Interface para consulta a membros de uma comunidade

Sendo assim, um dos métodos implementados no agente *Web service*, quando invocado, informa à aplicação os usuários que também estão fazendo uso de determinado dispositivo cliente, ou seja, quem são os membros da comunidade que usam aquele *software* ou objeto educacional. Além disso, o agente também informa à aplicação em quais outras comunidades o usuário selecionado está inserido, permitindo que o usuário da aplicação saiba quais as outras maneiras que pode interagir com aquele membro da comunidade em questão. Este processo está representado através da Figura 5.34, em que é possível conhecer os membros da comunidade **Agenda** e quais as outras maneiras de interagir com o usuário selecionado.

CAPÍTULO 6

Validação e verificação da arquitetura: um estudo de caso

O objetivo deste capítulo é descrever as etapas, o fluxo da informação e os testes envolvidos no processo de integração entre as duas aplicações desenvolvidas para a validação da arquitetura proposta. Para isso, foi verificado se os protótipos cumprem com suas especificações e o tempo de resposta em situações reais de uso.

As aplicações desenvolvidas com base na arquitetura WSMEL não exigem mais a necessidade de escrever um código de procedimentos para acessar e mapear os dados ligados à integração de aplicativos dentro do ambiente de *M-Learning*, isso porque, não existe a necessidade de compreender as APIs dos diversos aplicativos incorporados, pois as interfaces altamente acopladas e codificadas entre os aplicativos são eliminadas de forma eletiva, uma vez que, todas as informações necessárias para a integração estão inclusas na camada de interoperabilidade. Os passos e procedimentos envolvidos na criação de uma interface de integração baseada nesta arquitetura, onde dois aplicativos incorporados ao ambiente trocam informações entre si, serão descritos em um fluxo direcional de informações, embora a mesma metodologia acomode um intercâmbio bidirecional. O cenário utilizado para execução dos testes com a arquitetura desenvolvida é descrito a seguir.

6.1 Cenário de testes

Os testes foram executados com base na estrutura oferecida pelo laboratório de computação móvel vinculado a Unisinos (Universidade do Vale do Rio dos Sinos), conhecido como *Mobilab*, e compreendem:

- Dez HP iPAQ hx4700 Pocket PC (Figura 6.1), cujas especificações[1] podem ser verificadas através da tabela 6.1.

[1] http://h20285.www2.hp.com/estore/config.asp?cModel=LABR-FA282A%23AC4

Figura 6.1: HP iPAQ hx4700 Pocket PC

Tabela 6.1: Especificações do HP iPAQ hx4700 Pocket PC.

Sistema operacional	Microsoft ®Windows ®Mobile ™2003 *Second Edition*
Processador	Intel ®PXA270 Processor 624 MHz
Conectividade	Interface *wireless* padrão 802.11b e interface *bluetooth*
Memória	192MB de memória total (135MB acessíveis pelo usuário)
Tela	Tela TFT do tipo transflectivo 4"com luz auxiliar
Dimensões (L x W x H)	13,1 x 7.70 x 1.50 cm

- Um *Access Point* Cisco Aironet 1100 Series [2]: o qual oferece suporte aos padrões 802.11 g e 802.11 b, e conexão a diversos equipamentos Wi-FI, como pode ser verificado através da Figura 6.2.

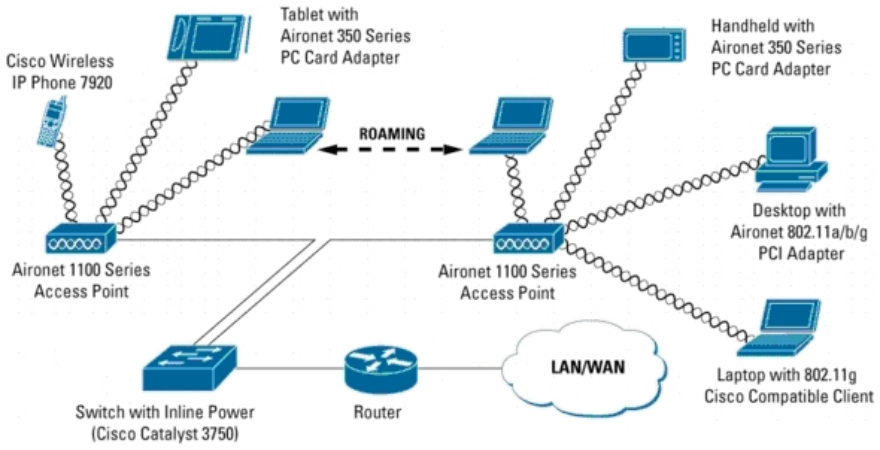


Figura 6.2: Estrutura suportada pelo *Access Point* Cisco Aironet 1100

[2] www.cisco.com

- Duas estações servidoras Pentium 4 2.2 GHz, 1 Gb de RAM executando *Internet Information Server*(IIS) e *Microsoft SQL Server 2000 Personal Edition.*

6.2 Inserção das aplicações na camada de interoperabilidade

O primeiro passo na construção da interface de integração entre aplicativos consiste na criação da ontologia preliminar, que define a estrutura e os metadados semânticos de cada aplicação que será oferecida aos usuários do ambiente de ensino. Tais informações são utilizadas para declarar o significado, as funções e os requisitos de processamento destes *softwares* e são utilizadas na composição da camada de interoperabilidade. Assim, o *InteropEdit* é utilizado para definir estas informações, além disso, as aplicações desenvolvidas neste projeto foram excluídas da camada de interoperabilidade para simular a sua não existência, e, posteriormente, foram novamente inseridas com a finalidade de testar o funcionamento do protótipo deste editor.

O "*InteropEdit*" cria, então, um documento XSD compatível com W3C, assim como um modelo de referência de nó de árvore visual do esquema e permite alterar as propriedades de cada elemento da árvore através da janela de propriedades. A ilustração da Figura 6.3 mostra o *InteropEdit*, o modelo de nó de árvore de documento à esquerda, e a representação XML do esquema no painel à direita para os dispositivos clientes **Agenda** e ***E-mail.***

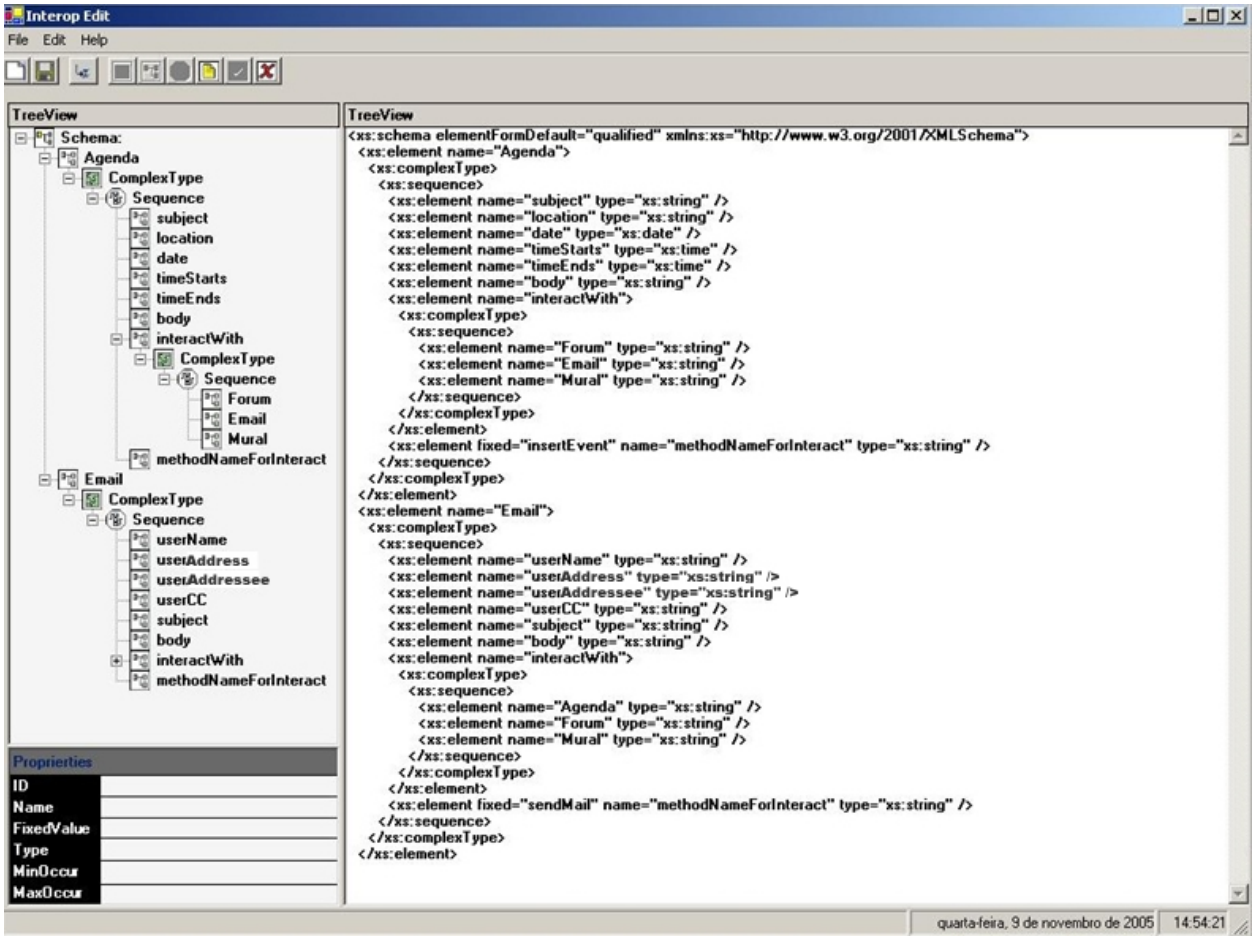


Figura 6.3: O editor de *schemas* InteropEdit

O esquema da Figura 6.3 descreve os metadados semânticos que governam as duas aplicações, e também mostra que, através do elemento *interactWhith*, foram especificadas outras aplicações com as quais ambas podem interagir. Pode-se notar ainda que o elemento *methodNameForInteract* especifica o nome do formulário que exibirá os dados quando a aplicação for o destino de dados provenientes de intercâmbio, e que após o término da inserção das informações acima, estas informações ficam inseridas na camada de interoperabilidade através do comando exibido na Figura 6.4.

Figura 6.4: Menu arquivo do InteropEdit

6.3 Cadastro de usuários e *download* das aplicações

A fim de proporcionar ao usuário o *download* das aplicações para composição de seu ambiente educacional móvel, desenvolveu-se o portal exibido através da Figura 6.5, onde primeiramente o usuário deverá realizar um cadastramento.

(A) Tela para cadastro de usuário (B) Tela para *download* de aplicativos

Figura 6.5: Portal WSMEL

Portanto, o cadastro consitui-se de algumas informações pessoais que servem de base para identificação das comunidades ao qual o usuário irá pertencer, tais informações são obtidas no

momento em que o usuário faz o *download* do *software*, sendo que as aplicações **Agenda** e ***E-mail*** foram obtidas a partir deste portal.

6.4 Iniciando o intercâmbio de dados

O terceiro passo consiste em instanciar uma das aplicações, neste caso a **Agenda** (Figura 6.6.a), e invocar o intercâmbio de dados com uma segunda aplicação, o ***E-mail***. O agente *Web service* neste caso irá mapear as informações equivalentes a partir de um ou mais nós em um esquema fonte para um ou mais nós no esquema de destino. Então, estas informações são exibidas ao usuário, através da interface de mapeamento, que pode modificar esse mapa conforme necessário (Figura 6.6.b). Finalmente, confirmando as alterações, a aplicação de destino exibe as informações no formulário apropriado (Figura 6.6.c).

(a) Evento cadastrado na agenda (b) Interface de mapeamento (c) Formulário alvo

Figura 6.6: Integração de dados entre as aplicações Agenda e *E-mail*

6.5 Taxonomia e execução dos testes

Em um primeiro momento foram estabelecidos testes funcionais, ou de caixa-preta, como geralmente são conhecidos, enfatizando as entradas, saídas e princípios funcionais dos dois protótipos de *softwares* desenvolvidos para incorporar o ambiente WSMEL, e cuja a responsabilidade foi atribuída a alunos de graduação não envolvidos na implementação da arquitetura. Para isso foi requisitado aos alunos para testarem os casos de uso citados nas seções 5.1.3.1 e 5.1.3.2 e intercambiar os dados entre aplicações independente de qual aplicação será a origem dos dados. Tais testes foram utilizados para verificar a confiabilidade da arquitetura proposta e podem ser observados através do gráfico exibido na Figura 6.7, a qual demonstra o resultado da simulação, onde nota-se que para cem simulações em um grupo de dez usuários, apenas três tiveram problemas de conexão com o serviço, devido à perda de sinal da rede sem fio. Nenhum erro funcional foi reportado.

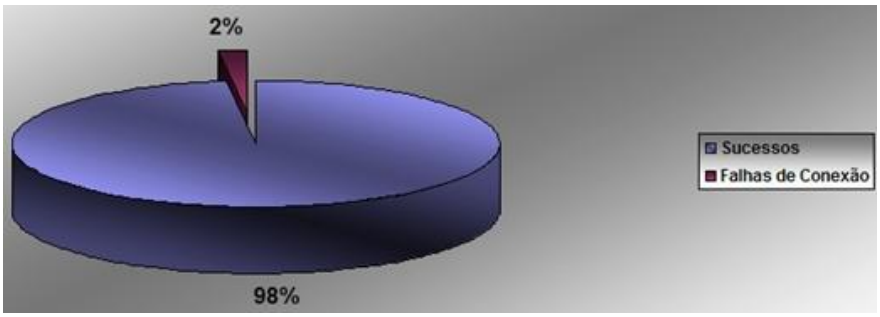


Figura 6.7: Porcentagem de sucesso dos testes funcionais

Em um segundo momento, testes de desempenho foram utilizados para quantificar o tempo de resposta das aplicações ao acessarem as regras de negócio implementadas no agente *Web service*. Com esse objetivo, a metodologia usada foi a de iniciar os testes com apenas um dispositivo e progredir até chegar ao limite de 10 dispositivos com acesso simultâneo, sendo que o resultado pode ser observado no gráfico exibido na Figura 6.8. Portanto, verifica-se com este teste que o tempo de resposta é proporcional ao volume de dados que está trafegando na rede, visto que em algumas situações, mesmo que o número de requisições simultâneas ao agente tenha aumentado, o tempo de resposta foi menor.

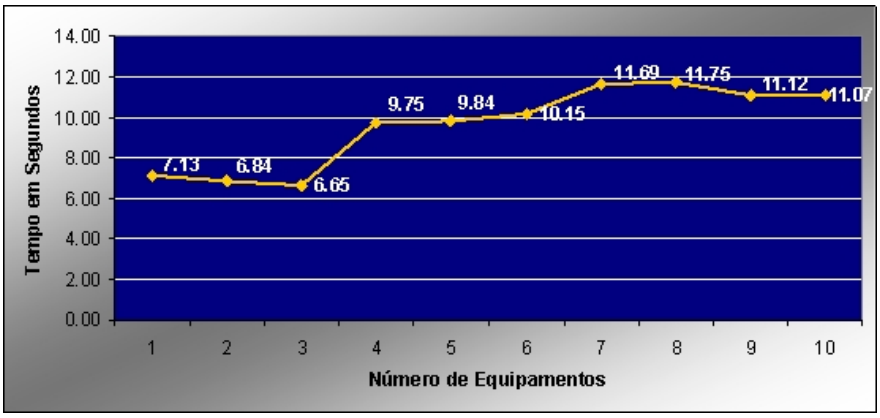


Figura 6.8: Tempo de respostas do agente *Web service*

CAPÍTULO 7

Considerações Finais

A pesquisa apresentada neste documento mostrou o WSMEL, uma arquitetura de software que fornece os recursos necessários para o desenvolvimento de um ambiente integrado de M-Learning, e cuja as raízes estão na integração de diversas aplicações. A seguir, serão apresentadas algumas conclusões, assim como também trabalhos relacionados que poderão ser desenvolvidos futuramente.

7.1 Conclusões

A realização deste trabalho permitiu elevar o processo de integração da camada de clientes (*softwares* aplicativos) para a camada de interoperabilidade e de agentes, separando a informação das aplicações que operam sobre ela, expondo-as como texto puro, e usando *schemas* XML auto descritíveis para que o agente conseguisse realizar o intercâmbio de dados.

O XML *Schema* é uma recomendação do W3C, uma versão estável, conveniente para uso na indústria de software (Deitel, 2003). O mesmo desempenha um papel importante na arquitetura, tendo em vista que é a tecnologia utilizada no desenvolvimento da ontologia incorporada pela camada de interoperabilidade. Sua função é atuar como um dicionário de elementos abstratos, entidades de atributos e regras organizacionais de cada aplicativo presente nesta camada, o que permite que o agente identifique termos idênticos entre os *softwares* que estão interagindo.

Para o desenvolvimento do trabalho, foram estudadas técnicas computacionais necessárias para implementação e disponibilização dos recursos. Assim, foram realizados estudos e experimentos sobre o desenvolvimento de ontologias, agentes de *software*, *Design Patterns* e *Web services*. Tendo como base estes estudos e experimentos, foi possível projetar a arquitetura e criar os diagramas de caso de uso, seqüência e classes.

Finalizada a modelagem, foram implementados os serviços oferecidos pelo agente *Web service*, visando prover o intercâmbio dos dados entre as aplicações. Durante esta fase implementou-se também o editor de *schemas* para acesso e manutenção da ontologia englobada pela camada de interoperabilidade. Após esta etapa, iniciou-se a implementação das aplicações que foram testadas, em um primeiro momento, através do emulador de Pocket PC e, em uma segunda instância, em equipamentos reais.

Como forma de testar e validar os serviços e protótipos de *softwares* foi realizado um estudo de caso para avaliar a arquitetura desenvolvida. Dentro deste estudo, testes funcionais e de desempenho foram efetuados com as aplicações acopladas ao ambiente WSMEL. Com base nos resultados obtidos, observou-se que todas as funcionalidades, tanto as dos dispositivos clientes, quanto do agente *Web Service* não apresentaram problemas, pois somente são causadas falhas quando ocorre a perda de sinal na rede *wireless*, ocasionando o não acesso ao serviço invocado.

A pesquisa ainda obteve as seguintes contribuições:

- um editor de XML *schemas* capaz de criar e editar ontologias para incorporação a camada de interoperabilidade;
- um agente *Web service* capaz de pesquisar propriedades equivalentes entre quaisquer aplicações inseridas na camada de interoperabilidade, garantido o intercâmbio de dados independente de plataforma e linguagem utilizada para o desenvolvimento de aplicações;
- um ambiente de *M-Learning* modular que pode incorporar aplicações de maneira integrada, desde que estas estejam inseridas no domínio expresso pela camada de interoperabilidade;
- um portal *web* para cadastro de usuários e *download* de aplicativos incorporados a camada de interoperabilidade;
- dois aplicativos que foram disponibilizados para *download* e uso no ambiente WSMEL.

Além de tais contribuições, a pesquisa proporcionou, ainda, duas publicações: uma internacional, submetida no ano de 2005, intitulada "*Using Web services to promote integration of mobile applications in distance learning domain*" (Scopel, 2005) no *IEEE International Conference on Next Generation Web Serices Pratices* (NWeSP'05), no qual foi possível mostrar a arquitetura WSMEL para a comunidade científica e ainda discutir a importância da interoperabilidade entre aplicações utilizadas em ambiente de *M-Learning*; e outra nacional, publicada no ano de 2004, intitulada "Pocket: Um ambiente de ensino a distância usando *handhelds* na formação de comunidades virtuais espontâneas"(Scopel, 2004) no Simpósio Brasileiro de Informática na Educação (SBIE 2004), onde apresentou-se uma visão inicial da arquitetura desenvolvida.

Desta forma, os objetivos deste trabalho foram alcançados, na medida que, dada a sua conclusão, a comunidade de desenvolvedores tem à disposição as funcionalidades necessárias para promover a integração de aplicações desenvolvidas a ambientes de *M-Learning* em que as comunidades são formadas com base nas preferências de cada indivíduo. Além disso, destaca-se a total disponibilidade e interoperabilidade que os recursos implementados possuem para serem reutilizados através da tecnologia de *Web services*.

7.2 Trabalhos futuros

A seguir, são apresentados alguns trabalhos que poderão ser realizados como continuidade dessa dissertação:

- manter e aprimorar os serviços desenvolvidos de forma que atenda o maior número de demandas possíveis com a garantia de flexibilidade, interoperabilidade e qualidade;

- aprimorar a ontologia desenvolvida como um esquema preliminar de consulta utilizado pelo agente *Web service*, de forma que o agente identifique atributos, não somente idênticos, mas também similares. Sugere-se a utilização de esquemas RDF;
- aprimorar o editor de ontologia: "InteropEdit", de forma que o mesmo contemple todas as propriedades possíveis para edição de um XML *Schema*.
- disponibilizar uma versão do "InteropEdit" para acesso via *browser*.
- inserir novas aplicações à camada de interoperabilidade e implementar novos protótipos que possam ser acoplados ao ambiente desenvolvido.

Referências Bibliográficas

(Arhippainem, 2003) Arhippainem, Leena. Use and integration of third-party components in software development. In ESPOO 2003. Finlândia.

(Batista, 1976) Batista, M. *Desenvolvimento de comunidades.* Cortez e Moraes. São Paulo. 1976

(Bradshaw, 1997) Bradshaw, J.M. Software Agents. Cambridge: MIT. 1997

(Britto, 2004) Britto, Jaime; Lopes, Heitor; Michalkiewicz, Edson L. Mobile computing on telemedicine and distance learning: Application on surgery pediatric oncology. In 2nd IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'04) - 2004.

(Cerami, 2002) Cerami, Ethan. Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly, First Edition February 2002.

(Chandrasekaran, 1999) Chandrasekaran, B.; Josephson, J. What are ontologies, and why do you need them? In IEEE Intelligent Systems. 1999

(Chappell, 2002) Chappell, D. Understanding .NET. Reading Addison-Wesley. 2002

(Coyle, 2002) Coyle, Frank P. *XML, Web services and the Data Revolution.* Reading Addison - Wesley Information Technology Series. 2002

(DevBtSol, 2004) Microsoft. Desenvolvimento de uma solução biztalk server. Reading Microsoft Document, 2004.

(Deitel, 2003) Deitel, H.M; Deitel, P.J.; Listfield, J.; Nieto, T. R.;Yager, C.; Zlatkina, M. *C# Como Programar.* Reading Addison - Pearson Education. 2003

(Duell, 1997) Duell, M. (1997). Catalog of non-software examples of design patterns. In Object Magazine, July - 1997.

(Felicíssimo, 2004) Felicíssimo, Carolina Haward; Breitman, Karin Koogann. *Uma Estratégia para o alinhamento taxonômico de ontologias.* Pontifícia Universidade Católica do Rio de Janeiro - PUC, Rio de Janeiro, 2004.

(Fensel, 2004) Fensel, Dieter. Ontologies: A silver bullet for knowledge management and Electronic Commerce. Reading Springer. 2000.

(Ferris, 2003) Ferris, Christopher; Farrell, Joel (2003). What Are Web services? In Communications of the ACM. Vol.46 n. 6. Junho. 2003.

(Fonseca, 2000) Fonseca, Frederico; Egenhofer, Max. Ontologias e interoperabilidade entre SIGS. In II Brazilian Symposium on GeoInformatics. 2000.

(Freitas, 2003) Freitas, S.;Levene, M. Evaluating the development of wearable devices, pdas and the use of other mobile devices in further and higher education institutions. In JISC, 2003.

(Hansen, 2003) Hansen, Roseli (2003). Glue script: Uma linguagem específica de domínio para composição de web services. Dissertação de Mestrado, Universidade do Vale do Rio dos Sinos - Unisinos. São Leopoldo, 2003.

(Hansen, 2002) Hansen, R.; Santos, C.; Pinto, S.C.S.;Lanius, G.; Massen, F. Web services: An architectural overview. In Proceedings of the First Seminar on Advanced Research in Electronic Business, Rio de Janeiro, 2002

(Hasebrook, 2004) Hasebrook, J. Learning support systems for organizational learning. Reading World Scientific, 2004.

(J.Hendler, 2001) J.Hendler. Agents and web semantic. In IEEE Intelligent Systems, 2001.

(Kaufmann, 1997) Kaufmann, M. Agents. Reading Addison-Wesley, 1997.

(Kist, 2002) Kist, T., Diverio, T., Lima, J., and Tollens, L. Benchmark de ambientes de gerenciamento de cursos a distância. In Simpósio Brasileiro de Informática na Educação - SBIE, São Leopoldo, 2002.

(Klein, 2001) Klein, M. XML, RDF, and relatives. In IEEE Intelligent Systems, 2001.

(Lucena, 2002) Lucena, C. Convergência entre a web dos usuários humanos e a web dos agentes de software: a engenharia e os impactos dos sistemas multi-agentes. In XXIX Seminário Integrado de Software e Hardware - SEMISH 2002. Florianópolis, SC.

(Marchal, 2000) Marchal, B. XML: Conceitos e aplicações. Berkley, 2000.

(Marques, 2004) Marques, M. C.; Loureiro, A. A. F. Adaptation in Mobile Computing. In: XXII Simpósio Brasileiro de Redes de Computadores, Gramado, RS. 2004. p.439 - 452

(McIlraith, 2001) McIlraith, S.; Son, T.; Zeng, H. Semantic web services. in ieee intelligent systems. In XXIX Seminário Integrado de Software e Hardware - SEMISH 2002. Florianópolis, SC.

(Metsker, 2004) Metsker, S. Padrões de Projeto em Java. Bookman, 2004.

(Miller, 1998) Miller, Eric. An introduction to the resource description framework. Reading D-Lib Magazine, maio de 1998.

(Nader, 2002) Nader, P. (2002). *Introdução ao estudo do direito.* Rio de Janeiro: Forence, 22 ed., 2002.

(Noy, 2004) Noy, Natalya F.; McCuinness, DeborahL. Ontology development 101: A guide to creating your first ontology. In Stanford University. Stanford, CA.

(Nyíri, 2002) Nyíri, Kristóf. Towards a philosophy of m-learning. In IEEE International Workshop on Wireless and Mobile Technologies in Education. 2002

(Oellermann, 2001) Oellermann, Willian L. Architecting Web services. Apress, 2001.

(Pacheco, 1994) Pacheco, R.; Kern, V. Uma ontologia comum para a integração de bases de informações e conhecimento sobre ciência e tecnologia. In Ci Inf., Brasília, v. 30, n.3, p. 56-63, set./dez. 2001.

(Pagani, 2005) Pagani, M. *Mobile and Wireless Systems Beyond 3g.* Reading Idea Group Inc (IGI), 2005.

(Perry, 2001) Perry, M.; O'Hara, K.; Sellen, A.; Brown, B.;Harper, R. Dealing with mobility: Understanding access anytime, anywhere. In ACM Transactions on Computer Human Interaction. Vol 8, n. 4, 2001.

(Rheinheimer, 2004) Rheinheimer, L. Wsagent: Um agente baseado em web services para promover interoperabilidade entre sistemas heterogêneos no domínio da saúde. Dissertação de Mestrado, Programa de Pós Graduação em Computação Aplicada - UNISINOS, São Leopoldo, 2004.

(Rheinheimer, 2003) Rheinheimer, L. and Pinto, S. Usando o framework Jlearningservices para instanciar serviços síncronos para ambientes de EAD. In Simpósio Brasileiro de Informática e Educação - SBIE, São Leopoldo, 2002.

(Sadiq, 2003) Sadiq, W. and Racca, F. Business Services Orchestration. Reading Cambridge University Press, 2003.

(Santanchè, 1999) Santanchè, A.; Teixeira, C. Explorando linguagens de markup extensíveis na construção de sistemas baseados na web. In XXVI SEMISH - Seminário Integrado de Software e Hardware - SBC'99 Rio de Janeiro, 1999.

(Santanchè, 2002) Santanchè, A.; Teixeira, C. Mais pontes e menos ilhas: estratégias para integração de software educacional. In Simpósio Brasileiro de Informática na Educação - SBIE. São Leopoldo, 2002.

(Scopel, 2004) Scopel, Marcelo; Kratz, Ricardo.; Pinto, Sérgio Crespo C. S. Pocket: Um ambiente de ensino a distância usando handhelds na formação de comunidades virtuais espontâneas. In Simpósio Brasileiro de Informática na Educação - SBIE, São Leopoldo, 2002.

(Scopel, 2005) Scopel, Marcelo; Kratz, Ricardo.; Pinto, Sérgio Crespo C. S. Using Web Services to promote integration of mobile applications in the distance learning domain. In IEEE International Conference on Next Generation Web Services Practices (NWsSP'05). Chung-Ang University, Seul, Korea, August, 22-26, 2005.

(Seely, 2002) Seely, Scot. SOAP: cross platform Web service development using XML. Prentice Hall, 2002.

(Serain, 2002) Serain, D. *Middleware and Enterprise Application Integration.* Springer, 2002.

(Silva, 2005) Silva, P.; Castro, J.; Roque, I. Information router: Plataforma web service de comunicação entre aplicações. In XML, Aplicações e Tecnologias Associadas, XATA 2005. Braga, Portugal.

(Sing, 2004) Sing, G.; Denoue, L.; Das, A. Collaborative note taking. In IEEE International Workshop on Wireless and Mobile Technologies in Education - WMTE. 2004.

(Sommerville, 2004) Sommerville, Ian. Engenharia de Software. Reading Addison Wesley, São Paulo, 2004.

(Souza, 2004) Souza, V.inícius. SWservice: uma biblioteca para a escrita da língua brasileira de sinais baseada em web services. Dissertação de Mestrado, Programa de Pós Graduação em Computação Aplicada - Unisinos, São Leopoldo, 2004.

(Staab, 2004) Staab, S. Knowledge representation with ontologies: The present and future. In IEEE Intelligent Systems, 2004.

(Vila, 2003) Vila, X.; Riera, A.; Sánchez, E.; Lama, M.; Moreno, D. A PDA based interface for a computer supported educational system. In IEEE International Conference on Advanced Learning Technologies - ICALT. 2003.

(Wache, 2001) Wache, H.; Vogele, U.; Visser, U.; Stuckenschmidt, G.; Shuster, H.; Neumann, S. Ontology-based integration of information - a survey of existing approaches. In IJCAI Workshop on Ontologies and Information Sharing. 2001.

Referências da *Web*

(Amorin, 2005) Amorin, R.; Sánchez, E.,; Vila, X.; Riera, A.; Lama, M.; Barro, S.. An educational ontology based on metadata standards. Disponível em <http://www.eume.net>. Acesso em: março de 2005.

(Balani, 2002) Balani, Naveen. Web Services architecture using MVC Style. Disponível em <http://www-106.ibm.com/developerworks/library/ws-mvc>. Acesso em: março de 2005.

(Booth, 2004) Booth, David. Web service architecture. W3C Working Draft. Disponível em <http://www.w3.org/TR/2003/WD-ws-arch-20030808>. Acesso em: dezembro de 2004.

(Bray, 2005) Bray, Tim. What is rdf? xml.com. Disponível em <http://www.xml.com/lpt/a/2001/01/24/rdf.html>. Acesso em: fevereiro de 2005.

(Champin, 2001) Champin, Pierre Antoine . RDF tutorial. Disponível em <http://www710.univ-lyon1.fr/~champin/rdf-tutorial>. Acesso em: março de 2005.

(Dextra, 2005) Dextra (2005). Boletim Dextra 02. Web services na integração de sistemas corporativos. Disponível em <http://www.dextra.com.br/empresa/boletim/0302-02/02tecnologia.htm>. Acesso em: abril de 2005.

(Downes, 2005) Downes, Stephen. The future of online learning. In Online journal of distance learning administration, volume I, número 3, Fall 1998. Disponível em <http://www.westga.edu/~distance/downes13.html>. Acesso em: janeiro de 2005.

(Holman, 2000) Holman, G. (2000). What is xslt? Disponível em <http://www.xml.com/pub/a/2000/08/holman/index.html>. Acesso em: março de 2005.

(Netto, 2005) Netto, Max Mosimann. Mobilidade e dispositivos móveis. Disponível em <http://www.projetando.net/Articles/ViewArticle.aspx?ArticleID=16&CategoryID=5>. Acesso em: fevereiro de 2005.

(Novello, 2005) Novello, Taísa Carla. Ontologias, sistemas baseados em conhecimento e modelo de banco de dados. Disponível em <www.inf.ufrgs.br/~clesio/cmp151/cmp15120021/artigo_taisa.pdf>. Acesso em: fevereiro de 2005.

(Orchard, 2005) Orchard, David. Extensibility, XML vocabularies, and XML schema. Disponível em <http://www.xml.com/pub/a/10/27/extend.html>. Acesso em: janeiro de 2005.

(Ropoli, 2005) Ropoli, E., Meneghel, L., Franco, A., Barcellos, M., Castilho, R., and Almeida, R. Orientações para o desenvolvimento de cursos mediados por computador. Disponível em <`http://www.ccuec.unicamp.br/ead.`>. Acesso em: abril de 2005.

(Sánchez, 2005) Sánchez, E.; Vila, X.; Riera, A.; Lama, M.; Barro, S.; and Amorin, A. The eume project: Modelling and design of an intelligent learning management system. Disponível em <`http://www.eume.net`>. Acesso em: março de 2005.

(Venu, 2005) Venu, V. A web services primer. Disponível em <`http://www.xml.com/pub/a/ws/2001/04/04/webservices/index.html`>. Acesso em: março de 2005.

(W3C, 2005) W3C (2005). XML schema part 2: Datatypes second edition. Disponível em <`http://www.w3.org/TR/xmlschema-2/`>. Acesso em: abril de 2005.

APÊNDICE A

Ontologia preliminar "ontoInterop.xsd"

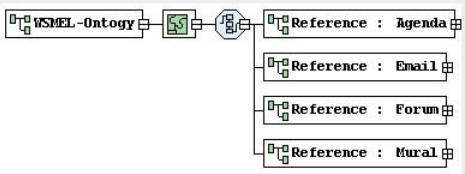
Este Apêndice traz informações referentes ao XML *Schema* correspondente aos aplicativos incluídos na camada de interoperabilidade.

Documentação

***Namespaces* declarados**

Prefixo	Namespace
xml	http://www.w3.org/XML/1998/namespace
xs	http://www.w3.org/2001/XMLSchema

WSMEL-Ontogy

Nome	WSMEL-Ontogy
Tipo	complex type
Diagrama	

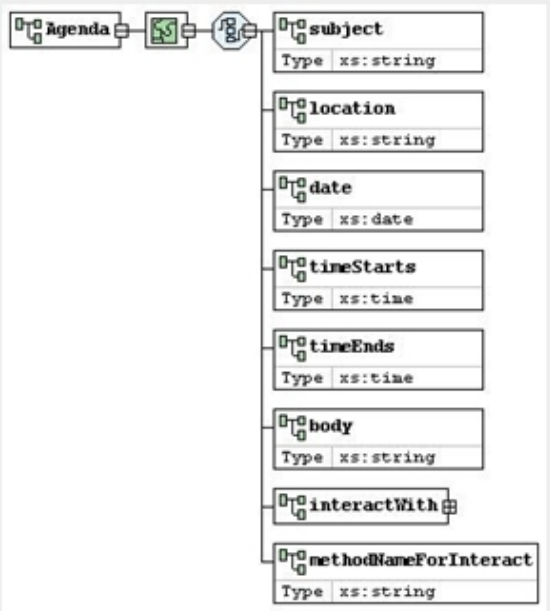
Representação de uma instância XML

```
<WSMEL-Ontogy>
   <Agenda> ... </Agenda> [1]
   <Email> ... </Email> [1]
   <Forum> ... </Forum> [1]
   <Mural> ... </Mural> [1]
</WSMEL-Ontogy>
```

Representação do XML *Schema*

```
<xs:element name="WSMEL-Ontogy">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Agenda"/>
         <xs:element ref="Email"/>
         <xs:element ref="Forum"/>
         <xs:element ref="Mural"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Agenda

Nome	Agenda
Tipo	complex type
Diagrama	

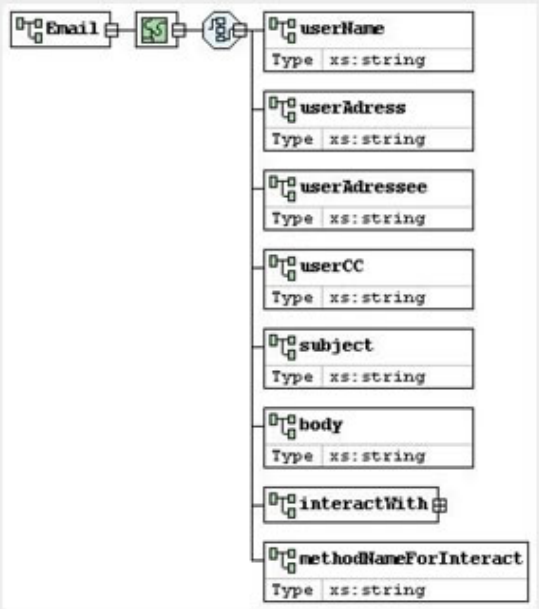
Representação de uma instância XML

```
<Agenda>
   <subject> xs:string </subject> [1]
   <location> xs:string </location> [1]
   <date> xs:date </date> [1]
   <timeStarts> xs:time </timeStarts> [1]
   <timeEnds> xs:time </timeEnds> [1]
   <body> xs:string </body> [1]
   <interactWith> [1]
      <Forum> xs:string </Forum> [1]
      <Email> xs:string </Email> [1]
      <Mural> xs:string </Mural> [1]
   </interactWith>
   <methodNameForInteract> insertEvent </methodNameForInteract> [1]
</Agenda>
```

Representação do XML *Schema*

```
<xs:element name="Agenda">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="subject" type="xs:string"/>
      <xs:element name="location" type="xs:string"/>
      <xs:element name="date" type="xs:date"/>
      <xs:element name="timeStarts" type="xs:time"/>
      <xs:element name="timeEnds" type="xs:time"/>
      <xs:element name="body" type="xs:string"/>
      <xs:element name="interactWith">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Forum" type="xs:string"/>
            <xs:element name="Email" type="xs:string"/>
            <xs:element name="Mural" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="methodNameForInteract" type="xs:string" fixed="insertEvent"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Email

Nome	Email
Tipo	complex type
Diagrama	

Representação de uma instância XML

```
<Email>
   <userName> xs:string </userName> [1]
   <userAddress> xs:string </userAddress> [1]
   <userAddressee> xs:string </userAddressee> [1]
   <userCC> xs:string </userCC> [1]
   <subject> xs:string </subject> [1]
   <body> xs:string </body> [1]
   <interactWith> [1]
      <Agenda> xs:string </Agenda> [1]
      <Forum> xs:string </Forum> [1]
      <Mural> xs:string </Mural> [1]
   </interactWith>
   <methodNameForInteract> sendMail </methodNameForInteract> [1]
</Email>
```

Representação do XML *Schema*

```
<xs:element name="Email">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="userName" type="xs:string"/>
         <xs:element name="userAddress" type="xs:string"/>
         <xs:element name="userAddressee" type="xs:string"/>
         <xs:element name="userCC" type="xs:string"/>
         <xs:element name="subject" type="xs:string"/>
         <xs:element name="body" type="xs:string"/>
         <xs:element name="interactWith">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Agenda" type="xs:string"/>
                  <xs:element name="Forum" type="xs:string"/>
                  <xs:element name="Mural" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="methodNameForInteract" type="xs:string" fixed="sendMail"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Forum

Nome	Forum
Tipo	complex type
Diagrama	

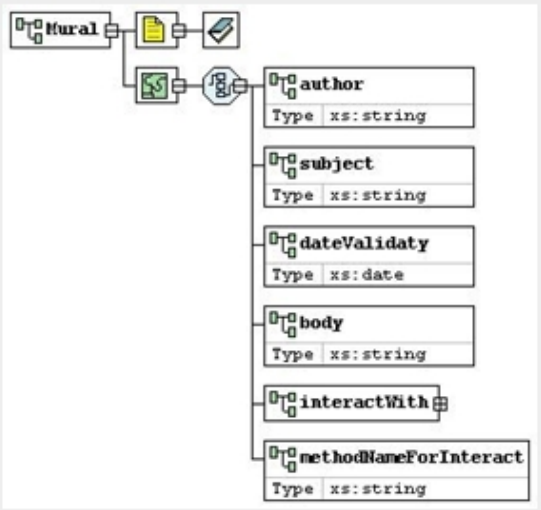
Representação de uma instância XML

```
<Forum>
   <date> xs:date </date> [1]
   <time> xs:time </time> [1]
   <subject> xs:string </subject> [1]
   <body> xs:string </body> [1]
   <status> xs:string </status> [1]
   <author> xs:string </author> [1]
   <reply> xs:string </reply> [1]
   <authorReply> xs:string </authorReply> [1]
   <interactWith> [1]
      <Agenda> xs:string </Agenda> [1]
      <Email> xs:string </Email> [1]
      <Mural> xs:string </Mural> [1]
   </interactWith>
   <methodNameForInteract> insertFAQ </methodNameForInteract> [1]
</Forum>
```

Representação do XML *Schema*

```
<xs:element name="Forum">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="date" type="xs:date"/>
         <xs:element name="time" type="xs:time"/>
         <xs:element name="subject" type="xs:string"/>
         <xs:element name="body" type="xs:string"/>
         <xs:element name="status" type="xs:string"/>
         <xs:element name="author" type="xs:string"/>
         <xs:element name="reply" type="xs:string"/>
         <xs:element name="authorReply" type="xs:string"/>
         <xs:element name="interactWith">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Agenda" type="xs:string"/>
                  <xs:element name="Email" type="xs:string"/>
                  <xs:element name="Mural" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="methodNameForInteract" type="xs:string" fixed="insertFAQ"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Mural

Nome	Mural
Tipo	complex type
Diagrama	

Representação de uma instância XML

```
<Mural>
   <author> xs:string </author> [1]
   <subject> xs:string </subject> [1]
   <dateValidaty> xs:date </dateValidaty> [1]
   <body> xs:string </body> [1]
   <interactWith> [1]
      <Agenda> xs:string </Agenda> [1]
      <Email> xs:string </Email> [1]
      <Forum> xs:string </Forum> [1]
   </interactWith>
   <methodNameForInteract> </methodNameForInteract> [1]
</Mural>
```

Representação do XML *Schema*

```
<xs:element name="Mural">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="author" type="xs:string"/>
         <xs:element name="subject" type="xs:string"/>
         <xs:element name="dateValidaty" type="xs:date"/>
         <xs:element name="body" type="xs:string"/>
         <xs:element name="interactWith">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Agenda" type="xs:string"/>
                  <xs:element name="Email" type="xs:string"/>
                  <xs:element name="Forum" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="methodNameForInteract" type="xs:string" fixed=""/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figura A.1: Ontologia preliminar "interopEdit.xsd".