UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

ALEX FERNANDO ORLANDO

UMA INFRA-ESTRUTURA COMPUTACIONAL PARA O GERENCIAMENTO DE PROGRAMAS DE ENSINO INDIVIDUALIZADOS

SÃO CARLOS
2009

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

ALEX FERNANDO ORLANDO

UMA INFRA-ESTRUTURA COMPUTACIONAL PARA O GERENCIAMENTO DE PROGRAMAS DE ENSINO INDIVIDUALIZADOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos para obtenção do título de Mestre em Ciência da Computação.

Orientação: ***Cesar Augusto Camillo Teixeira***

Co-orientação: ***Antonio Francisco do Prado***

SÃO CARLOS
2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

O71ie

Orlando, Alex Fernando.
Uma infra-estrutura computacional para o gerenciamento de programas de ensino individualizados / Alex Fernando Orlando. -- São Carlos : UFSCar, 2010.
179 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Engenharia de software. 2. Ambiente em educação à distância. 3. Arquitetura de software. 4. Arquitetura orientada a serviços. 5. Interoperabilidade. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma Infra-Estrutura Computacional para o Gerenciamento de Programas de Ensino Individualizados"

ALEX FERNANDO ORLANDO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. César Augusto Camilo Teixeira
(Orientador - DC/UFSCar)

Prof. Dr. Antonio Francisco do Prado
(Co-orientador - DC/UFSCar)

Profa. Dra. Camila Domeniconi
(DPsi/ UFSCar)

Prof. Dr. Wilson Vicente Ruggiero
(POLI/USP)

São Carlos
Outubro/2009

AGRADECIMENTOS

À minha família, que é a base firme de tudo o que sou: Minha mãe Sônia, meu pai João, e meus irmãos Fernanda e Rafael. Meu amor e gratidão por vocês são eternos.

À minha namorada Isabela, pelo amor, carinho, compreensão e ajuda para resolver vários problemas. Você é muito importante para mim, meu anjo, e sem você eu não teria conseguido.

Aos meus sogros, Eduardo e Sueli, por terem me acolhido como parte da família.

Ao meu orientador, Prof. Dr. Cesar Augusto Camillo Teixeira, pelos valiosos ensinamentos, confiança no meu trabalho, paciência e, principalmente, companheirismo.

À Profª. Drª Deisy das Graças de Souza, por ter acreditado no projeto e por ter oferecido uma fundação sólida para sua realização.

Ao Rodrigo Estevan Bela por ter sido meu parceiro no desenvolvimento do GEIC e por ter me acompanhado durante todo o mestrado.

Aos camaradas dos laboratórios LECH e LINCE: Jussara, Mainá, Leonardo, Cris, Tiago, Daniel, Erick, Raphael Jesus, Raphael Trancinha e muitos outros.

A todos os meus amigos. Tudo o que faço é com uma pequena ajuda de vocês.

A FAPESP pelo apoio financeiro.

RESUMO

No processo educacional dificuldades com a alfabetização podem acarretar sérios prejuízos à evolução acadêmica do aluno. Pesquisas com métodos de ensino para aplicação individualizada, baseados no paradigma de equivalência de estímulos, mostram que é possível melhorar significativamente a aprendizagem de leitura com compreensão (e de outras habilidades). Embora apresentem bons resultados, esses métodos têm sua abrangência limitada, pois a participação presencial de um tutor nem sempre é possível ou economicamente viável. Além disso, a autoria e gerenciamento de programas de ensino baseados nesses métodos são complexos e exigem o treinamento de tutores e especialistas de domínio. Alguns resultados já foram alcançados no sentido de informatizar esse processo, porém as soluções apresentadas oferecem pouca flexibilidade, pois são voltadas para o ensino de um assunto específico ou são muito genéricas, exigindo esforço complementar para sua efetiva utilização. Nesta dissertação são exploradas e avaliadas técnicas de engenharia de *software* e computação distribuída na busca de uma solução que possa proporcionar a redução dos custos envolvidos na criação, aplicação e gerenciamento de programas de ensino individualizados, além de permitir a difusão em larga escala do ensino através desses métodos. É proposta uma infra-estrutura de *software* que viabiliza a realização desses objetivos, composto de uma arquitetura, módulos para realização de tarefas específicas, e recomendações de padrões de arquitetura, projeto e interface para auxiliar no desenvolvimento ou adaptação de novos módulos. Com o objetivo de maximizar a base instalada dos módulos desenvolvidos e também de permitir que um maior número de alunos seja beneficiado, a solução promove a interoperabilidade de aplicações de ambiente *Web*, dispositivos móveis e de TV Digital Interativa. A fim de validar a infra-estrutura proposta e também de oferecer uma codificação operacional ao problema do ensino individualizado por computador, foi desenvolvido um sistema modular chamado GEIC, já implantado e em uso por pesquisadores de várias universidades. Apesar da motivação inicial deste trabalho ter sido oferecer uma infra-estrutura para a criação de sistemas para o ensino de escrita e leitura, a solução proposta é bastante abrangente e suporta outros domínios e paradigmas de ensino.

Palavras-chave: Programas de ensino individualizados. Infra-estrutura de *software*. Arquitetura orientada a serviços. Interoperabilidade.

ABSTRACT

Difficulties in learning how to read and write might bring serious jeopardy to students' progression in school. Researches focusing teaching methods for individualized application, based on the paradigm of stimulus equivalence, show that it is possible to significantly improve learning of reading and other skills Although they present good results, these methods have limited range, because the presence of a tutor is not always possible or economically viable. Besides, authoring and management of learning programs based on these methods are complex and demand training of tutors and specialists. Some results have already been reached concerning to informatization, though the solutions presented offer little flexibility, since they focus on teaching topics that are either too specific or too broad, which requires complementary effort for its effective usage. This dissertation explores and evaluates software engineering and distributed computing techniques in an attempt to find a solution that might diminish costs involved in the creation, application and management of individualized programs, besides allowing large scale diffusion of teaching through these methods. A software infrastructure that allows these objectives is proposed, composed by an architecture, modules to perform specific tasks, and recommendations on some patterns of architecture, project and interface in order to help the development or adaptation of new modules. In order to maximize the installed base of the developed modules and also to benefit a larger number of students, the solution promotes interoperability of Web applications, mobile devices and interactive digital television. Aiming the validation of the proposed infrastructure and also to offer an operational codification to the problem of computerized individualized learning, a modular system named GEIC was developed and deployed, being used by researchers of several universities. Despite the initial motivation of this work to offer an infrastructure for the creation of systems specific for teaching reading and writing, the solution proposed here is very broad in scope and supports other domains and learning paradigms.

Keywords: Individualized learning programs. Software infrastructure. Service-oriented architecture. Interoperability.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BDD	*Behavior-Driven Development*
CAPSI	*Computer-Aided Personalized System of Instruction*
CLDC	*Connected Limited Device Configuration*
CORBA	*Common Object Request Broker Architecture*
CR	*Constructed Response*
CSS	*Cascading Style Sheet*
DCOM	*Distributed Component Object Model*
(D)TO	*(Data) Transfer Object*
EJB	*Enterprise JavaBeans*
FIFO	*First-In First-Out*
GEIC	Gerenciador de Ensino Individualizado por Computador
GFDL	*GNU Free Documentation Language*
GPL	*GNU Public License*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
Java EE	*Java Platform, Enterprise Edition*
Java ME	*Java Platform, Mobile Edition*
Java SE	*Java Platform, Standard Edition*
JMS	*Java Message System*
JPA	*Java Persistence API*
JSF	*JavaServer Faces*
JSP	*JavaServer Pages*
LCMS	*Learning Content Management System*
LECH	Laboratório de Estudos do Comportamento Humano
LINCE	Laboratório de Inovação em Ciência e Engenharia
LMS	*Learning Management System*
LO	*Learning Object*
LOM	*Learning Object Metadata*
MEF	Máquina de Estados Finita

MIDP	*Mobile Information Device Profile*
MVC	*Model-View-Controller*
MTS	*Matching-To-Sample*
NCL	*Nested Context Language*
OASIS	*Organization for the Advancement of Structured Information Standards*
ORM	*Object/Relational Mapping*
PEI	Programa de Ensino Individualizado
PSI	*Personalized System of Instruction*
QTI	*Question and Test Interoperability*
REST	*Representational State Transfer*
SBTVD	Sistema Brasileiro de Televisão Digital
STB	*Set-Top-Box*
SCORM	*Shareable Content Object Reference Model*
SGBD	Sistema Gerenciador de Banco de Dados
SOA	*Service-Oriented Architecture*
SOAP	*Service-Oriented Access Protocol*
SQL	*Structured Query Language*
TVDI	Televisão Digital Interativa
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
XML	*eXtended Markup Language*
XP	*eXtreme Programming*
WSDL	*Web Service Description Language*

SUMÁRIO

1 INTRODUÇÃO

A aprendizagem de leitura e escrita consiste em um processo complexo, uma vez que requer a aprendizagem de relações arbitrárias que caracterizam o comportamento simbólico (SIDMAN, 1994). No processo educacional dificuldades com a alfabetização podem acarretar sérios prejuízos à evolução acadêmica do aluno. Apesar do fracasso não significar necessariamente que os alunos apresentem deficiências intelectuais, há evidencias de que eles demandem programas especiais, de preferência individualizados (KELLER, 1967a). Pesquisas com métodos de ensino para aplicação individualizada, baseados no paradigma de equivalência de estímulos, mostram que é possível melhorar significativamente não só a aprendizagem de leitura com compreensão de alunos em período escolar, mas também com outras populações de estudantes: pré-escolares, estudantes de classe especial e adultos (SOUZA; DE ROSE, 2006).

Baseado nos princípios dos programas de ensino individualizados (PEIs) foi criado na década de 1960 o Sistema Personalizado de Instrução (*Personalized System of Instruction* – PSI) (KELLER, 1967a; KELLER, 1967b), também conhecido como "Método Keller" ou "Plano Keller". Em um curso baseado no PSI, o material a ser ensinado é dividido em unidades menores de estudo, tais como um capítulo ou um grupo de palavras. Outras características essenciais desse sistema que mais diferem dos procedimentos tradicionais de ensino são o ritmo individualizado, exigência do domínio do material por parte do aluno, uso de professores e aulas apenas como veículos para motivação, ênfase dada à palavra escrita na comunicação professor-aluno, e o uso de tutores durante as sessões de ensino.

Embora apresentem bons resultados (TAVEGGIA, 1976; KULIK; KULIK; COHEN, 1979; KULIK; KULIK; BANGERT-DROWNS, 1990), esses métodos tem sua abrangência limitada, pois a viabilidade de sua adoção em larga escala é dificultada por alguns obstáculos inerentes às suas principais vantagens. O desenvolvimento de PEIs é um trabalho demorado e minucioso, e o recrutamento de tutores é uma tarefa custosa, uma vez que esses precisam ter domínio sobre o conteúdo ensinado e estarem dispostos a aplicar vários testes e a prover *feedback* individual (SHERMAN, 1977; SHERMAN, 1992). A análise dos dados também é custosa, visto que para cada tentativa executada durante uma sessão de ensino são registradas muitas variáveis, dentre elas as respostas emitidas. Todos esses obstáculos coloca-

ram o ensino baseado em PEIs em desvantagem frente a métodos mais tradicionais (AINSWORTH, 1979).

Vale notar que no início do desenvolvimento de PEIs baseados nesse sistema os testes eram impressos em papel e organizados seqüencialmente em pastas tipo fichário (DE ROSE, 1989). Com sua popularização, os computadores pessoais passaram a ser usados para apresentação de PEIs e registro do desempenho de cada aluno (SOUZA et al., 2009). A eficiência e flexibilidade dos cursos aumentaram com a adoção de redes de comunicação como a Internet (PEAR; CRONE-TODD, 1999). Mais recentemente, com o surgimento de novas plataformas de *hardware*, a consolidação de sistemas operacionais livres, o início da implantação da TV Digital Interativa (TVDI) e a expansão das redes de telefonia móvel, tornou-se oportuno repensar novamente as formas e meios utilizados para a aplicação de PEIs.

Nesta dissertação são exploradas e avaliadas técnicas de engenharia de *software* e computação distribuída na busca de uma solução que possa proporcionar a redução dos custos envolvidos na criação, aplicação e gerenciamento de PEIs, além de permitir a difusão em larga escala do ensino através desses métodos. É proposta uma infra-estrutura de *software* que viabiliza a realização desses objetivos, composta de uma arquitetura, módulos para realização de tarefas específicas, e recomendações de padrões de arquitetura, projeto e interface para auxiliar no desenvolvimento ou adaptação de novos módulos. Com o objetivo de maximizar a base instalada dos módulos e permitir que um maior número de alunos seja beneficiado, a solução promove a interoperabilidade de aplicações de ambientes *Web*, dispositivos móveis e TVDI.

A fim de validar a infra-estrutura proposta e avaliar as técnicas de Engenharia de *Software* e Computação Distribuída adotadas, e também de oferecer uma solução operacional ao problema do ensino individualizado por computador, foi desenvolvido um sistema chamado GEIC baseado no paradigma de equivalência de estímulos.

Espera-se que a criação da infra-estrutura proposta traga contribuições tanto para desenvolvedores e engenheiros de *software* quanto para especialistas de domínio de várias áreas (psicologia, educação especial, pedagogia etc.) e alunos com dificuldades de aprendizado. A solução desenvolvida é genérica e permite que outros paradigmas de ensino além da equivalência de estímulos possam se beneficiar. Além disso, a partir da infra-estrutura podem ser criadas soluções voltadas não apenas para pesquisa e ensino, mas também para avaliações e outras situações que envolvam resolução de problemas.

O desenvolvimento do assunto proposto está assim organizado:

Capítulo 1: A fim de facilitar o entendimento do problema exposto neste trabalho, são apresentados conceitos básicos sobre aprendizagem de leitura e escrita, programas de ensino individualizados, TVDI e dispositivos móveis. Em seguida, são identificados os obstáculos que dificultam a adoção em larga desse tipo de programa de ensino.

Capítulo 2: É exposto o objetivo do trabalho, que é explorar e avaliar técnicas de arquitetura de *software* e computação distribuída na busca de uma solução que proporcione, entre outras coisas, a redução dos custos envolvidos na autoria, aplicação e gerenciamento de PEIs. São também enumeradas as contribuições esperadas com a realização deste trabalho.

Capítulo 3: É descrita a metodologia utilizada para o desenvolvimento, teste e avaliação desta proposta, incluindo a escolha de um ciclo de vida de *software*, duração dos ciclos de desenvolvimento, questionários utilizados, formato da documentação etc.

Capítulo 4: É apresentada, na forma de uma infra-estrutura computacional, uma solução para o problema, envolvendo a descrição das etapas de análise de requisitos, seleção de disciplinas de suporte (gerenciamento de projeto), arquitetura de *software* e projeto.

Capítulo 5: É descrito um estudo de caso, realizado tanto para validar a proposta quanto para prover um produto completo baseado nas idéias discutidas. O sistema desenvolvido encontra-se em funcionamento em ambiente de produção.

Capítulo 6: A fim de reforçar a validação do trabalho, são descritos três questionários e experimentos realizados durante o decorrer do projeto, bem como uma avaliação subjetiva a respeito das técnicas utilizadas no decorrer do projeto.

Capítulo 7: São apresentados trabalhos correlatos e as vantagens e desvantagens de cada um deles, e porque a solução desde trabalho se mostrou a mais adequada.

Capítulo 8: São feitas considerações finais em relação ao trabalho. São discutidos os resultados alcançados, assim como são oferecidas diretivas para que os assuntos aqui abordados possam ser retomados e expandidos em projetos futuros.

2 IDENTIFICAÇÃO DO PROBLEMA

2.1 APRENDIZAGEM DE LEITURA E ESCRITA

A aprendizagem de leitura e escrita consiste em um processo complexo, uma vez que re-quer a aprendizagem de relações arbitrárias que caracterizam o comportamento simbólico (SIDMAN, 1971; SIDMAN, 1994). Em virtude dessa complexidade, um grande contingente de alunos fracassa nessa tarefa, em especial quando os métodos de ensino carecem de algumas propriedades fundamentais para promover a aprendizagem relacional.

As habilidades básicas necessárias para se aprender a relacionar elementos da fala, do texto e do mundo são discriminações simples e condicionais. Do mesmo modo que o aluno aprende a distinguir entre uma televisão e um telefone, deve aprender a discriminar entre as palavras faladas “televisão” e “telefone” e entre as palavras impressas “televisão” e “telefone” (discriminações simples). Além disso, deve aprender discriminações em que a condição é importante (discriminação condicional), por exemplo, escrever a palavra “televisão” será uma resposta correta se o professor houver ditado a palavra “televisão”, mas será incorreta se ele houver ditado “telefone”.

Para facilitar a aprendizagem destas discriminações, programas de ensino (PEs) baseados no paradigma da equivalência de estímulos são compostos por tentativas (testes) consecutivas. Em cada tentativa, certa configuração de elementos (estímulos) que podem ser visuais (figuras e palavras impressas), sonoros (como palavras ditadas) e até mesmo táteis (como os empregados em Braille) é apresentada e o aluno deve apresentar uma resposta. Um PE usualmente inclui uma conseqüência ou *feedback* para o comportamento do aluno, de modo que ele tenha uma indicação precisa de acertos ou erros, e pode envolver uma nova oportunidade de resposta sempre que ele hesitar ou apresentar uma resposta incorreta.

Ao longo de uma seqüência de tentativas, o aluno tem múltiplas oportunidades para cada discriminação a ser aprendida, de modo a fortalecer sua aprendizagem e garantir sua manutenção. Um programa especialmente eficaz de tentativas discretas para o ensino de discriminações condicionais é o de emparelhamento com o modelo. Nele, em cada tentativa é apresentado um estímulo modelo e um conjunto de estímulos de comparação, entre os quais o aluno deve escolher um – essa tarefa favorece a aprendizagem da relação entre o estímulo

modelo e o estímulo de comparação correto. Um exemplo de tentativa de emparelhamento com modelo pode ser visto na Figura 1.

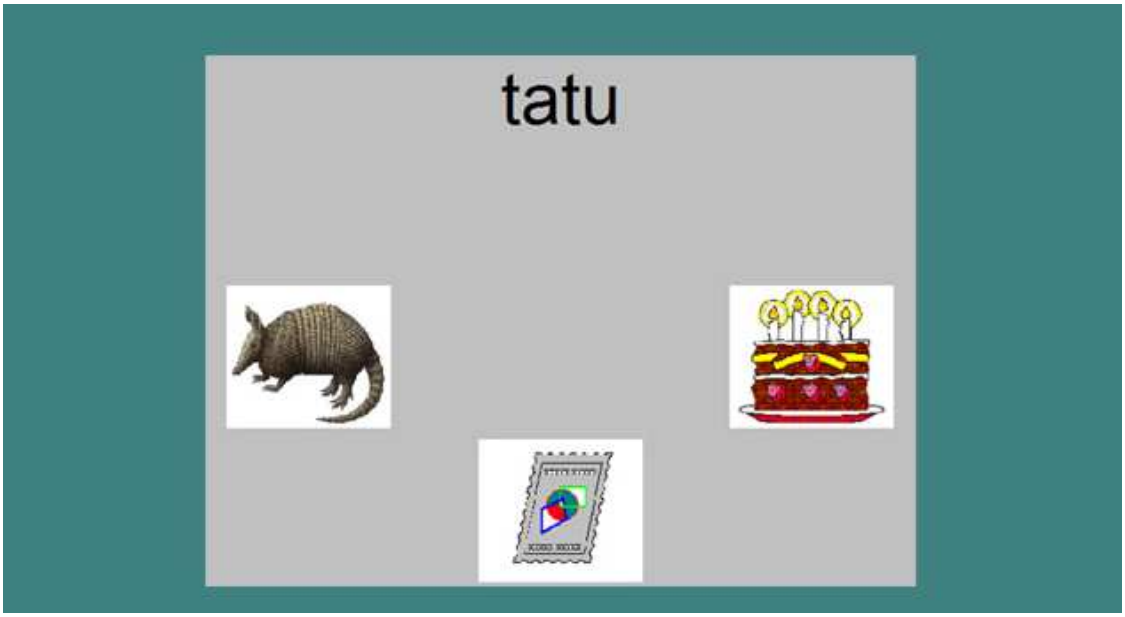


Figura 1 - Exemplo de tentativa de emparelhamento com modelo

PEs baseados no procedimento de emparelhamento com o modelo tem sido bem sucedidos em ensinar leitura a pessoas com atraso de desenvolvimento, tais como retardo mental e autismo e também a alunos que, embora não apresentem atrasos acentuados, mostram dificuldade na aprendizagem em sala de aula (SOUZA; DE ROSE, 2006). No entanto, na década de 70 Murray Sidman observou, ao realizar pesquisas com ensino de leitura, que esses procedimentos podem gerar um resultado ainda mais importante: eles podem favorecer a emergência de comportamentos que não foram diretamente ensinados e, com isso, aumentar muito o potencial de aprendizagem do aluno (SIDMAN, 1971; SIDMAN, 1994; STROMMER; MAKAY; STODDARD, 1992).

A emergência de novas relações a partir de outras diretamente ensinadas mostrou que os elementos das discriminações condicionais tornavam-se membros de uma mesma classe de estímulos por equivalência (SIDMAN; TAILBY, 1982). A partir disso concluiu-se que palavras ditadas, figuras e palavras impressas haviam se tornado estímulos equivalentes. Há três propriedades da definição matemática de equivalência: reflexividade, simetria e transitividade. Reflexividade corresponde à relação condicional entre dois estímulos idênticos (A=A). Já simetria corresponde à relação condicional inversa à treinada entre dois estímulos diferentes (se A=B então B=A). Finalmente transitividade é a relação condicional entre dois estímulos não relacionados anteriormente (se A=B e B=C, então A=C).

Apesar do fracasso durante a aprendizagem não significar necessariamente que os alunos apresentem deficiências intelectuais, a pesquisa tem evidenciado que eles demandam PEs especiais, de preferência individualizados (KELLER, 1967a). De início esses PEs devem ensinar certos componentes mais básicos das relações simbólicas, que são requisitos para aprendizagens mais complexas, com aumento gradual no grau de dificuldade e respeitando o ritmo de cada aluno.

Baseado nos princípios da análise do comportamento e dos PEs individualizados (PEIs), Keller criou na década de 1960 o Sistema Personalizado de Instrução (*Personalized System of Instruction* – PSI) (KELLER, 1967b; KELLER, 1968). Esse sistema, também conhecido como "Método Keller" ou "Plano Keller", surgiu a partir da Instrução Programada de Skinner (SKINNER, 1958).

Em um curso baseado nos princípios do PSI, o material a ser ensinado é dividido em unidades menores de estudo, tais como um capítulo ou um grupo de palavras. Outras características essenciais desse sistema que mais diferem dos procedimentos tradicionais de ensino são:

- Ritmo individualizado: permite a um aluno avançar em um curso em seu próprio ritmo de acordo com suas habilidades, gastando assim menos tempo em material já dominado e focando esforços em outros materiais não dominados;
- Domínio do material: requer que um aluno aprenda uma unidade menor de estudo e passe em um teste a respeito desse conteúdo, só assim avançando para uma nova unidade. Se o critério de aprovação não for alcançado, requer que o aluno estude novamente o conteúdo e refaça o teste quantas vezes forem necessárias;
- Uso de professores, aulas e demonstrações apenas como veículos para motivação, ao invés de fontes de informação crítica: o instrutor ou professor é visto apenas como um facilitador do ensino, sendo que a sala de aula é usada tipicamente para esclarecer dúvidas e motivar alunos. Geralmente o conteúdo do curso deve ser lido pelo aluno fora da sala de aula;
- Ênfase dada à palavra escrita na comunicação professor-aluno;
- Uso de tutores: também conhecido como mentores, são pessoas que já dominaram o material ensinado, podendo ser inclusive alunos de turmas anteriores ou que estão na mesma turma, mas que estão em passos mais avançados. Esses tutores provêem *feedback* individualizado sobre o desempenho dos alunos nos testes, e geralmente oferecem ajuda em áreas onde o aluno possui dificuldades.

Vários trabalhos encontrados na literatura atestam a eficácia de PSI em relação a métodos tradicionais de ensino (TAVEGGIA, 1976; KULIK; KULIK; COHEN, 1979; KULIK; KULIK; BANGERT-DROWNS, 1990). Do mesmo modo, PSI tem se mostrado um sistema ideal para a realização de aulas à distância (GRANT; SPENCER, 2003). Entretanto, a viabilidade de sua adoção em larga escala é dificultada por alguns obstáculos inerentes às suas principais vantagens.

O desenvolvimento de PEIs é um trabalho demorado e minucioso, visto que cada material precisa ser dividido em tentativas que devem ser criadas e aplicadas de acordo com o desempenho do aluno. Além disso, várias tentativas costumam ser pequenas variações de uma tentativa comum, o que geralmente dificulta a tarefa de gerenciá-las.

O recrutamento de tutores também é uma tarefa custosa, uma vez que esses precisam ter domínio sobre o conteúdo ensinado e estarem dispostos a aplicar vários testes e prover *feedback* individual (SHERMAN, 1977; SHERMAN, 1992). Além do mais, o treinamento e supervisão desses tutores exigem um grande comprometimento por parte da instituição de ensino, tanto em termos financeiros quanto estruturais. A participação presencial de um tutor em uma sessão de ensino passa também a ser problemática, principalmente a partir do momento em que a quantidade de alunos supera em muito a quantidade de tutores disponíveis. Não obstante, em casos onde alunos e tutores se encontram geograficamente distantes, os custos relacionados ao transporte podem ser proibitivos.

A análise dos dados também costuma ser um trabalho custoso, visto que para cada tentativa executada durante uma sessão de ensino são registradas muitas variáveis, dentre elas as respostas emitidas e suas latências, a configuração de cada tentativa no momento das respostas etc. Além do mais, o número de tentativas em cada PEI geralmente ultrapassa as centenas.

Todos esses obstáculos relacionados ao desenvolvimento, gerenciamento e aplicação de unidades de ensino, além da questão de alocação de recursos humanos, fizeram com que o ensino baseado em PEIs fosse colocado em desvantagem frente a métodos mais tradicionais (AINSWORTH, 1979). Outros obstáculos menos fundamentados surgiram da crença de que o conteúdo não poderia ser ensinado adequadamente sem que houvesse um professor lecionando à frente da sala de aula (SHERMAN, 1992).

Apesar dos debates a respeito da viabilidade de PSI, vale notar que, no início do desenvolvimento de PEs baseados nesse sistema, as tentativas eram impressas em papel e organizadas seqüencialmente em pastas tipo fichário (DE ROSE, 1989). Porém, com o surgi-

mento de computadores pessoais, esses passaram a ser usados para apresentação das tentativas, com a vantagem de que o computador viabiliza não apenas a apresentação do material instrucional, mas também o registro confiável do desempenho do aluno (SOUZA et al., 2009).

PSIs têm sido desenvolvidos e aplicados com o apoio de computadores há décadas (CROWELL; QUINTANAR; GRANT, 1981), sendo que a eficiência e flexibilidade desses cursos aumentaram com a difusão de redes de comunicação como a Internet (PEAR; CRONE-TODD, 1999), criando novas possibilidades como educação a distância. Recentemente, com o surgimento de novas plataformas de *hardware*, a consolidação de sistemas operacionais livres, o início da implantação da TV Digital Interativa e a expansão das redes de telefonia móvel, tornou-se possível e oportuno repensar as formas e meios utilizados para a aplicação de PEIs. Porém, a realização da interoperabilidade entre essas novas plataformas e tecnologias é uma tarefa bastante complexa, já que as particularidades presentes em cada uma delas podem inviabilizar o processo.

2.2 TELEVISÃO DIGITAL INTERATIVA

A televisão é um dos mais importantes meios para difusão de informações entre as pessoas por todo o mundo, democratizando assim o acesso a notícias, entretenimento, serviços e educação. O alcance do serviço de televisão beira a totalidade dos territórios nacionais em países tão distintos como o Brasil e os da União Européia (IBGE, 2009; TELECO, 2009; CPQD, 2009a). Em locais com grandes níveis de desigualdade social o acesso à televisão funciona como elemento de integração, fazendo com que tanto vilarejos remotos e subdesenvolvidos quanto metrópoles altamente desenvolvidas obtenham sinais de televisão aberta (também conhecidos como sinais terrestres).

A TV Digital Terrestre brasileira, também conhecida como Sistema Brasileiro de TV Digital (SBTVD), garante ganhos significativos em termos de qualidade de som e imagem em relação à televisão tradicional (transmissão analógica). Além do mais, aumenta a oferta de canais, assim como oferece a possibilidade de interação direta do telespectador com as emissoras ou com qualquer outro dispositivo conectado a Internet. Através desses recursos vários benefícios poderão ser estendidos a uma vasta camada da população que atualmente

tem acesso ao entretenimento audiovisual de forma quase exclusivamente passiva (CPQD, 2009a).

Tradicionalmente, usuários e emissoras de televisão estabelecem uma relação unidirecional onde o último distribui conteúdo e o primeiro o consome. Para que haja efetivamente interação entre esses dois papéis, é necessário que exista um meio de comunicação propício ligando usuários a emissoras e fazendo com que informação trafegue também em sentido inverso. Para satisfazer essa necessidade foi especificado para o SBTVD o canal de interatividade, sistema que permite a interação de usuários com provedores de serviços e com outras aplicações (ITU-T, 1997). Ele é composto por dois canais:

- **Canal de retorno**: também conhecido como canal de subida, estabelece a comunicação no sentido do usuário para provedores de serviços e emissoras;
- **Canal direto**: também conhecido como canal interativo direto ou de descida, estabelece a comunicação no sentido de provedores de serviço para usuários.

Como ilustrado na Figura 2, para que esse canal de interatividade funcione na prática, é essencial que as redes de televisão integrem-se com as redes de telecomunicações (MANHÃES; SHIEH, 2005).

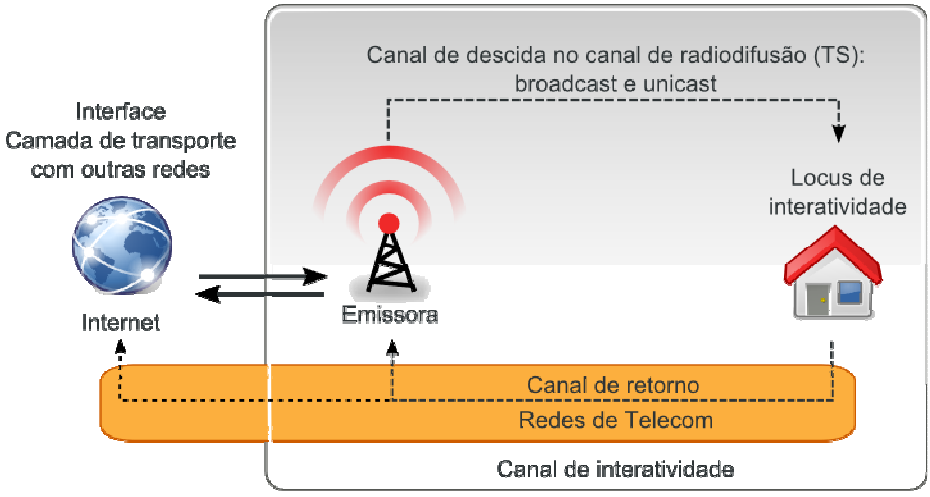


Figura 2 - Diagrama simplificado do canal de interatividade

Em sociedades modernas o valor de se assistir TV tem sido extensivamente criticado, com alguns conteúdos exibidos sendo considerados nocivos especialmente para crianças. Porém, um dos primeiros usos não-comerciais de sucesso na TV dos anos 1960 foi a difusão de conteúdo educacional para lares e escolas (REVELLE, 2003). Apesar das críticas relativas ao conteúdo da TV, a pesquisa releva vários aspectos positivos, tais como a oportunidade de ensino. As descobertas sugerem que aplicações de TV Digital interativa (TVDI) provê-

em suporte à educação e entretenimento para crianças e pessoas mais jovens, assim como educação contínua para todos.

Em particular, a TVDI é especialmente adequada para ensino informal e também para motivação de sua audiência (CHORIANOPOULOS; LEKAKOS, 2006), e pode ampliar o acesso à educação através de uma de suas grandes vantagens, que é a acessibilidade, o que suporta também a idéia de ensino por toda a vida. Além disso, a TVDI permite um ensino mais formal junto ao dia-a-dia (AARRENIEMI-JOKIPELTO, 2005).

O termo "*e-learning*" se refere ao emprego de novas tecnologias para fins de ensino, assim como ao foco no desenvolvimento de mecanismos flexíveis para a difusão de conteúdo de ensino (HENRY, 2001). A convergência de *e-learning* e TV, agregada a avanços em qualidade de som e imagem, personalização e interatividade, recebe o nome de "*t-learning"* (LYTRAS; CHOZOS; POULOUDI, 2002). *T-learning* é principalmente um fenômeno social, já que o ensino é o ponto principal, e não a tecnologia. As habilidades de busca de conhecimento e de saber como usá-lo são cada vez mais importante comparados com apenas conteúdo, já que esse é menos importante por estar temporalmente e socialmente situado (AARRENIEMI-JOKIPELTO, 2005).

O acesso à Internet para a realização da TVDI é um dos principais requisitos a serem tratados durante a etapa de projeto do SBTVD (BRASIL, 2003). Isso representa um problema, já que a partir de um ponto de vista técnico os padrões e tecnologias existentes são suficientes, porém de um ponto de vista econômico é muito improvável que parte da população brasileira obtenha esse acesso ao canal de retorno a um custo aceitável. Como resultado, é essencial que esse canal seja suficientemente flexível e que possa ser acessado a partir de um conjunto personalizado de tecnologias, adequadas ao poder aquisitivo e à localização física de cada usuário.

2.3 DISPOSITIVOS MÓVEIS

Um dispositivo móvel é um dispositivo computacional pequeno e leve o suficiente para ser operado enquanto o usuário o carrega em suas mãos ou em seu bolso. Sua principal diferença em relação a um computador portátil é que o último, apesar de poder ser transportado de um local para outro, só se torna de uso prático quando posicionado em algum lugar

estático. Exemplos de dispositivos móveis incluem telefones celulares e *smart phones*, PDAs (*Personal Digital Assistants*) e *notebooks* ultra-portáteis (*netbooks).* Esses vários tipos de dispositivos diferem significativamente entre si em itens como tamanho e qualidade do visor, dispositivos de entrada (botões numéricos tradicionais, teclados alfa-numéricos e tela sensível ao toque), presença de câmeras fotográficas e filmadoras etc.

A popularidade dos dispositivos móveis se deve principalmente aos aparelhos celulares e o uso da rede de telefonia móvel. Porém, essa classe de dispositivos é adequada também para usuários que necessitam de algumas das comodidades oferecidas por um computador pessoal, mas que não possuem condições financeiras para se ter um. Aliado ao fato da implantação de redes de telefonia móvel ser relativamente fácil e exigir baixo investimento e pouca infra-estrutura, pode-se explicar a alta taxa de adoção de tecnologias móveis em países emergentes. Outra ocasião onde o uso de dispositivos móveis é mais conveniente é em ambientes onde se necessita de mobilidade, mas não é prático transportar um computador pessoal. Exemplos desses ambientes são locais de difícil acesso (desertos e selvas), fábricas e linhas de montagem, campos de pesquisa etc.

A utilização de dispositivos móveis para o ensino pode aumentar as possibilidades de acesso a conteúdo educacional. *M-learning* é uma forma de *e-learning* para dispositivos móveis. Na literatura há diferentes definições para *m-learning*, sendo que algumas delas consideram-na apenas como educação a distância com o uso de redes sem fio (GEORGIEV; GEORGIEVA; SMRIKAROV, 2004). Porém, neste trabalho a definição de *m-learning* inclui a possibilidade de se ter acesso a materiais educativos a partir de qualquer lugar e a qualquer momento, sem conexão física permanente com redes de computadores via cabo.

Considerando mobilidade do ponto de vista do aprendiz ao invés do da tecnologia, pode-se dizer que *m-learning* está em praticamente todo lugar: desde alunos fazendo revisões para uma prova até estudantes de línguas melhorando seu vocabulário enquanto estão viajando a outros países. Todos esses exemplos de ensino formal e informal ocorrem enquanto as pessoas estão em movimento (O'MALLEY, 2004).

M-learning é um paradigma emergente e em estado de intenso desenvolvimento. Na pesquisa de Wains e Mahmood (2008) foi desenvolvida uma solução que integrasse *m-learning* e *e-learning* de modo a ampliar a distribuição de material de ensino para dispositivos leves e baratos, tais como telefones celulares. Dessa forma, objetivou-se aumentar a liberdade dos estudantes permitindo a eles estudarem a qualquer momento e em qualquer lugar, seguindo seus próprios ritmos e de acordo com necessidades pessoais.

A adoção de dispositivos móveis como ferramenta de auxílio ao ensino oferece alguns desafios, tais como apresentar conteúdo e respostas de maneira homogênea, independentemente do dispositivo utilizado (WAINS; MAHMOOD, 2008). Essas limitações implicam que talvez *m-learning* seja mais adequado para conteúdos específicos, tais como ensino de línguas ou outros conteúdos de natureza modular. Há várias outras limitações inerentes à natureza diminuta dos dispositivos móveis, entre elas: baixa disponibilidade de energia elétrica (a bateria deve resistir o máximo de tempo sem necessitar de recarga), baixo poder de processamento (devido, entre outras coisas, às limitações de energia elétrica), pouca memória, pouco espaço de armazenamento e ambiente de desenvolvimento simplificado.

3 OBJETIVOS

O objetivo deste trabalho é explorar e avaliar técnicas de engenharia de *software* e computação distribuída na busca de uma solução interdisciplinar que proporcione a redução dos custos envolvidos no gerenciamento de PEIs. É proposta uma infra-estrutura computacional que viabiliza o desenvolvimento de módulos de sistema voltados para a autoria, aplicação e gerenciamento de PEIs. Apesar de ter sido baseada no paradigma de equivalência de estímulos e no ensino de leitura e escrita, essa infra-estrutura deve ser genérica e suportar outros paradigmas de ensino e domínios.

Cada módulo dessa infra-estrutura deve possuir uma tarefa específica que auxilie especialistas a elaborarem PEIs, aplicar esses programas em alunos, gerenciar os membros da equipe de especialistas e também os alunos cadastrados no sistema. Adicionalmente, a aplicação de PEIs deverá ser feita por tutores tanto de maneira presencial quanto de maneira remota, permitindo assim que se reduzam drasticamente os gastos com o transporte tanto de tutores quanto de alunos.

Deve-se ampliar a base instalada das aplicações desenvolvidas e também permitir que um maior número de alunos seja beneficiado, principalmente aqueles que não possuem meios de se locomover até as escolas ou centros de pesquisa. Para isso, o ambiente deverá facilitar a criação de aplicações em diversas plataformas de *hardware*, tais como *Web*, TVDI e dispositivos móveis.

Espera-se que a criação da infra-estrutura proposta traga contribuições tanto para desenvolvedores e engenheiros de *software* quanto para especialistas de domínio de várias áreas (psicologia, educação especial, pedagogia etc.) e alunos com dificuldades de aprendizado.

Desenvolvedores e engenheiros de *software*, tanto aqueles já envolvidos com o projeto quanto futuros membros, passarão a dispor de uma infra-estrutura integrada composta de uma arquitetura de *software* bem especificada e de módulos já existentes funcionando nessa arquitetura, o que promoverá a reutilização de código-fonte. Em situações onde novos módulos forem necessários, os desenvolvedores contarão com ampla documentação, padrões de projeto e de uso da rede de comunicação, componentes de *software* e infra-estrutura de servidores, dessa maneira reduzindo os custos relacionados à programação, testes e implantação de novas funcionalidades.

Especialistas de domínio contarão com uma plataforma para a realização de pesquisas na área educacional, não mais necessitando de soluções parciais de *software* ou que não atendam às minúcias de suas áreas de interesse. Além disso, devido à flexibilidade do ambiente, no futuro poderão ser realizadas pesquisas e cursos sobre outros campos, tais como ensino de matemática, notação musical e sinais de trânsito. A análise dos dados obtidos através das pesquisas será facilitada, já que módulos específicos para essa finalidade auxiliarão na organização dos resultados e geração de relatórios.

Tutores terão a opção de acompanhar à distância a execução de um PEI, não mais tendo que se locomover até o local onde estão os alunos, gerando assim uma economia tanto em termos de tempo quanto de tutores necessários para uma determinada quantidade de alunos. Por se tornar uma tarefa menos custosa em termos de tempo, espera-se que a tarefa de monitorar alunos se torne também mais atrativa, fazendo assim com que mais tutores estejam disponíveis e mais alunos possam ser atendidos.

Alunos não precisarão mais se locomover até as escolas ou centros de pesquisa para poderem executar PEIs, o que é um benefício ainda maior para pessoas carentes ou que residem em áreas de difícil acesso ou geograficamente distantes. Em locais onde não há computadores disponíveis, o acesso aos PEIs poderá ser feito através de telefones celulares e TV-DI. Espera-se também que a criação de uma solução multiplataforma, disponível tanto através da *Web* quanto da TVDI e dispositivos móveis fomente o fortalecimento da ubiqüidade, eliminando assim barreiras entre as pessoas e as tecnologias através de soluções com foco em usabilidade.

O uso dessas novas tecnologias deverá tornar a solução atrativa também para pesquisadores das áreas de computação e engenharia, já que esses terão um ambiente real para efetuarem suas pesquisas. Como resultado do uso de um grande número de metodologias, ferramentas de *software*, e tecnologias, espera-se contribuir para a pesquisa em computação através da avaliação desses itens baseada em seus usos num ambiente de produção (e não apenas no âmbito acadêmico). A partir dessa avaliação será possível localizar as deficiências de cada solução e sugerir melhorias para suportar melhor tanto a infra-estrutura proposta quanto outras soluções de grande escala.

Finalmente, espera-se que a solução firme as bases para a criação de uma comunidade virtual (rede social) interdisciplinar composta de especialistas de domínio e tutores, cujo objetivo comum é promover o ensino de qualidade. A colaboração entre esses indivíduos facilitará sua comunicação, troca de informações e um fluxo cada vez mais intenso de experi-

ências. A união de metodologias de ensino consolidadas, conceitos de engenharia de *software* e computação distribuída, boas práticas de gerenciamento de projetos e tecnologias modernas, todos unidos sob a disciplina e a sistemática da engenharia de *software*, contribuirão de maneira substancial para um fim social tão importante quando a educação.

Para auxiliar no entendimento da proposta é conveniente separar os objetivos em dois grandes conjuntos: aqueles relacionados à infra-estrutura para ao desenvolvimento de sistemas (ambiente de desenvolvimento) e aqueles relacionados ao uso dos sistemas (ambiente de produção). Cada um desses ambientes possui um conjunto de requisitos bem distinto, além de requisitos relacionados à integração entre eles.

4 METODOLOGIA

Para guiar o desenvolvimento da solução como um todo foi usado o processo de desenvolvimento de *software* conhecido como modelo espiral (BOEHM, 1998), onde em cada ciclo de evolução há fases de identificação e especificação de requisitos, análise, identificação de riscos, projeto, codificação e testes. Esse modelo, que combina as características dos modelos “protótipo” e “cascata”, é ideal para projetos complexos.

Nesse modelo, os desenvolvedores iniciam o projeto e a codificação das partes essenciais do sistema em um protótipo e refinam o mesmo, adicionando novas funcionalidades e melhorias até ele se tornar o produto de *software* final que eventualmente se tornará o produto a ser entregue. A principal diferença deste modelo em relação à prototipagem tradicional é que, naquele, faz-se primeiramente um protótipo para auxiliar na elaboração dos requisitos finais do sistema, o qual depois é descartado. Já neste, o próprio protótipo evolui até se tornar o produto final.

O modelo espiral é adequado à resolução do problema apresentado, pois baseado nele é possível liberar versões funcionais da solução em ciclos relativamente curtos, e com isso receber *feedback* constante de desenvolvedores de *software* e especialistas de domínio.

No início da elaboração deste trabalho foram feitas as primeiras reuniões com especialistas de domínio das áreas de análise do comportamento e educação especial. O principal objetivo dessas primeiras reuniões foi a familiarização com conceitos e técnicas da área de psicologia, assim como o levantamento de uma bibliografia de referência que facilitaria o desenvolvimento de uma solução computacional para um problema interdisciplinar. Após a etapa foram realizadas reuniões no formato *brainstorming* (técnica de criatividade de grupo projetada para gerar um grande número de idéias para a solução de um problema).

A partir dessas primeiras reuniões foram especificados os requisitos necessários para o desenvolvimento do primeiro protótipo da infra-estrutura proposta. Com esse protótipo construído e avaliado, que incluiu os primeiros esboços da arquitetura de *software*, gerenciamento de projeto e módulos de *software*, o restante do desenvolvimento desta proposta foi realizado em ciclos de aproximadamente 30 a 60 dias de duração. No início de cada ciclo foram feitas reuniões com especialistas de domínio e desenvolvedores de *software* a fim de levantar os principais requisitos a serem realizados. Ao final de cada ciclo todos os membros

da equipe eram convocados a comparecer em uma sessão de testes coletiva, onde o resultado era então avaliado e sugestões e críticas eram registradas.

A fim de validar continuamente a infra-estrutura proposta e também de oferecer uma solução operacional ao problema do ensino individualizado por computador, foi desenvolvido junto a este trabalho um sistema de *software* modular. Esse sistema permitiu que tanto especialistas de domínio quanto desenvolvedores de *software* pudessem ter meios mais concretos para acompanhar a evolução do trabalho.

Para se obter uma avaliação mais precisa a respeito da qualidade da infra-estrutura e do sistema desenvolvido, foram realizados, além do estudo de caso, alguns estudos experimentais voltados tanto para especialistas de domínio quanto para engenheiros de *software*. Esses experimentos seguem uma forma simplificada do estudo experimental realizado por Travassos (2002).

Foi elaborado um questionário voltado aos especialistas de domínio envolvidos durante o desenvolvimento da solução. O principal objetivo desse questionário foi levantar os pontos fortes e fracos do sistema, coletar sugestões para futuras melhorias e descobrir se o processo adotado para o desenvolvimento da solução evoluiu adequadamente conforme as necessidades dos usuários.

Além de avaliar os usuários já familiarizados com o sistema, foi realizado um experimento com especialistas do domínio que nunca haviam tido contato com o sistema desenvolvido. A esses participantes foi ministrada uma aula introdutória sobre o domínio e o sistema, e em seguida foram elaboradas algumas atividades individuais, tais como criar tentativas, construir PEIs e aplicá-los em alguns alunos-teste.

Com o intuito de avaliar a infra-estrutura proposta do ponto de vista da engenharia de *software* e da computação distribuída, foi elaborado um experimento voltado a engenheiros e desenvolvedores de *software*. O experimento consistiu em reunir um grupo de alunos de graduação e pós-graduação da área de ciência da computação, ministrar a eles uma breve aula expositiva sobre os conceitos básicos a serem aplicado, e pedir a eles que executassem algumas tarefas simples, tais como desenvolver um módulo de *software*. Após isso, todos os alunos responderam um questionário cujo principal objetivo foi avaliar com que facilidade as tarefas foram executadas com o auxílio da infra-estrutura proposta.

Finalmente, foi feita uma avaliação subjetiva a respeito dos conceitos e técnicas utilizados durante o desenvolvimento deste trabalho. Nessa avaliação foram abordados os padrões arquiteturais adotados, padrões de projeto, plataformas de desenvolvimento, bibliotecas

de *software*, ferramentas para gerenciamento de projeto e outras técnicas e métodos da engenharia de *software* e computação distribuída.

Uma avaliação do desempenho de alunos que utilizaram o sistema desenvolvido está fora do escopo deste trabalho. Essa tarefa será iniciada por pesquisadores da área de psicologia, logo após a conclusão do sistema desenvolvido.

Em todas as etapas do processo de desenvolvimento da solução foi utilizado UML (*Unified Modeling Language*) (OMG, 2009), uma linguagem de modelagem padronizada e de uso geral no campo da engenharia de *software*. UML inclui um conjunto de técnicas de notação gráfica para a criação de modelos abstratos para sistemas específicos. Os principais diagramas utilizados foram o de casos de uso e de classes. Porém, em casos onde um diagrama UML dificultava a compreensão de um conceito, ilustrações menos formais e mais amigáveis foram utilizadas.

Com o objetivo de obter um retorno da comunidade acadêmica, foram feitas exibições a pesquisadores de outras universidades, tanto de maneira informal quanto através de escrita de artigos e apresentações em simpósios e congressos. Isso permitiu a generalização da infra-estrutura proposta, assim como gerou melhorias no sistema desenvolvido, tornando-o adequado para uso em domínios a princípio desconhecidos. Foram realizados também treinamentos para a comunidade acadêmica local a fim de fomentar a criação de uma comunidade de especialistas de domínio.

5 INFRA-ESTRUTURA PROPOSTA

Com base na metodologia descrita no capítulo anterior, foi concebida uma infra-estrutura voltada para a construção de sistemas de *software* para o gerenciamento de PEIs. Na literatura há pouco consenso sobre o que caracteriza um sistema ou uma infra-estrutura. No contexto deste trabalho, um sistema é um conjunto ou arranjo de elementos e processos relacionados cujo comportamento satisfaz as necessidades (operacionais ou dos clientes) e provê sustentação para o ciclo de vida dos produtos (IEEE, 1998). Já uma infra-estrutura representa, de maneira geral, a estrutura básica ou características de um sistema ou organização (MILLER, 1993). Especificamente no domínio da computação, uma infra-estrutura de software se relaciona a sistemas de software, o que inclui tópicos como sistemas operacionais, *middlewares* e quaisquer programas que ofereçam suporte a aplicações de *software* (IET, 2010).

A seguir serão detalhados todos os passos, assim como conceitos relacionados, que permitiram o desenvolvimento da solução tanto em relação ao ambiente de desenvolvimento quanto ao de produção. Partindo da etapa de análise de requisitos, são discutidas as decisões relacionadas ao gerenciamento de projeto de *software*, escolha da arquitetura e projeto.

5.1 ANÁLISE DE REQUISITOS

Para que um problema possa ser resolvido com sucesso, as necessidades e condições envolvidas, também chamadas requisitos, devem ser previamente determinadas. A análise de caso de uso é uma técnica usada para identificação de requisitos de um sistema e leva em conta principalmente as diferentes classes de usuários (atores) e seus objetivos ou tarefas (casos de uso) (TAYLOR, 2002).

A etapa de levantamento de requisitos foi parte importante do desenvolvimento deste trabalho, pois além de elucidar alguns conceitos específicos do domínio de aprendizagem, contribuiu para que se obtivesse uma visão bastante geral tanto da infra-estrutura necessária quanto do sistema a ser desenvolvido a partir dela. Porém, a quantidade de documentação necessária para se registrar todos os requisitos funcionais e não funcionais do domínio pode rapidamente ultrapassar a casa das dezenas de páginas, o que de certa forma prejudicaria

o entendimento da proposta apresentada. Logo, serão descritos apenas os atores e requisitos mais importantes e que são indispensáveis para o entendimento do trabalho.

Para facilitar o entendimento e visualização de todos os elementos envolvidos nesta etapa, esses serão organizados em dois grupos: 1) Ambiente de produção, relacionado ao uso do sistema desenvolvido a partir da infra-estrutura proposta e 2) Ambiente de desenvolvimento, relacionado ao processo adotado e ao projeto e desenvolvimento dos módulos do sistema.

A interoperabilidade com outros sistemas de ensino eletrônico através de padrões como SCORM, LOM e QTI (SUN, 2002), apesar de bastante desejável e necessária para se obter o máximo de reúso das unidades de ensino, não será abordada neste trabalho. Além de o domínio requerer particularidades que dificultariam esse processo, a complexidade inerente a esses padrões dificultaria a disponibilização em tempo hábil de uma prova de conceito. Assim, o assunto será eventualmente retomado em trabalhos futuros.

5.1.1 ATORES

Um ator modela o tipo de papel de uma entidade que interage com o sujeito (por exemplo, trocando sinais e dados), mas que é externo ao sujeito. Atores podem representar papéis executados por usuários humanos, *hardware* externo ou outros sujeitos, e não necessariamente representam uma entidade física específica, mas meramente uma faceta (papel) de alguma entidade que é relevante para a especificação dos casos de uso associados (OMG, 2009).

No domínio deste trabalho, foram identificados os seguintes atores:

- **Engenheiro de *Software*:** Aplica os princípios de engenharia de *software* ao projeto, desenvolvimento, teste, avaliação e manutenção de sistemas de *software*. Supervisiona todo o processo que circunda o ciclo de vida de um sistema, realizando o intermédio entre especialistas de domínio e desenvolvedores de *software*, gerenciando projetos, validando e disponibilizando novas versões de um sistema etc.
- **Programador:** Codifica novos módulos de *software* ou combina/reutiliza módulos já existentes para criar novas funcionalidades. Além disso, procura garantir, a-

através de testes, que o código-fonte gerado satisfaça os critérios de qualidade estabelecidos. Parte do ambiente de desenvolvimento.

- **Pessoa:** É a base para os outros dois atores do ambiente de produção (membro da equipe e aluno) e possui atributos como nome de usuário, senha etc.
- **Membro da equipe:** Executa as tarefas relacionadas à elaboração de conteúdo de ensino, administração de recursos humanos etc. Cada membro possui um conjunto de permissões que o caracteriza, basicamente, como um tutor, especialista de domínio ou administrador de recursos humanos (esses termos mais especializados serão usados até o final deste trabalho):
    - **Tutor:** Monitora os alunos enquanto esses executam PEIs. Além dessa tarefa de acompanhamento, um tutor realiza a avaliação subjetiva do desempenho de alunos em tarefas que envolvam resposta oral. Podem ser tutores especialistas de domínio, voluntários com conhecimento do domínio e até mesmo outros alunos que já dominam o material sendo ensinado.
    - **Especialista de Domínio:** O conhecedor do conteúdo a ser ensino, podendo ser representado tanto por psicólogos quanto por pedagogos ou programadores de ensino. Sua principal tarefa está relacionada à criação e teste de PEIs, assim como à consulta dos dados obtidos a partir da execução de PEIs por estudantes. Devido ao seu conhecimento do domínio, pode atuar como tutor.
    - **Administrador de Recursos Humanos:** Administra recursos humanos (alunos, tutores etc.), mantendo cadastros, regulando o acesso a recursos protegidos e atribuindo PEIs a alunos. Pode delegar suas tarefas a outros administradores. Parte do ambiente de produção.
- **Aluno:** É exposto ao conteúdo a ser ensinado, executando os PEIs criados pelo especialista de domínio. A execução de PEIs pode ser feita tanto de maneira presencial (acompanhado de um tutor) quanto remota (sem a presença de um tutor). Dependendo do seu conhecimento do assunto sendo ensinado, pode realizar o papel de tutor para alunos menos avançados. Parte do ambiente de produção.

Uma visão geral dos atores, assim como eles se relacionam aos ambientes de desenvolvimento e produção, está ilustrada na Figura 3.

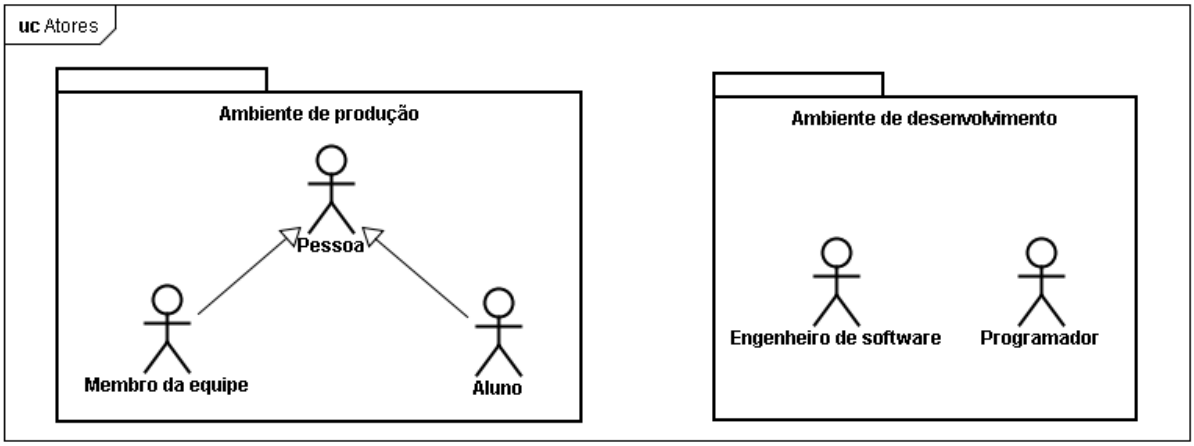


Figura 3 - Visão geral dos atores do domínio

5.1.2 REQUISITOS FUNCIONAIS

Um requisito funcional define uma função de um sistema, e é descrito como um conjunto de entradas, comportamentos e saídas (SOMMERVILLE, 2006). Pode representar cálculos, detalhes técnicos, manipulação de dados e outras funcionalidades específicas que definem o que o sistema deve realizar, e são expressos em casos de uso. Um caso de uso é a descrição do comportamento de um sistema em relação a como ele responde a uma requisição de fora desse sistema, ou seja, descreve “quem” pode fazer “o que”.

5.1.2.1 AMBIENTE DE PRODUÇÃO

Uma visão geral do ambiente de produção pode ser vista na Figura 4. Os principais casos de uso são:

- **Acessar *site*:** realizado pelos atores do domínio (Aluno, Tutor, Especialista de Domínio e Administrador de RH), consiste em acessar um *site* através de um navegador *Web*. A partir desse *site* podem-se ler notícias a respeito do sistema, conferir a situação do projeto (quantidade total de alunos atendidos, número de PEIs disponíveis etc.) e, principalmente, acessar os módulos do sistema;

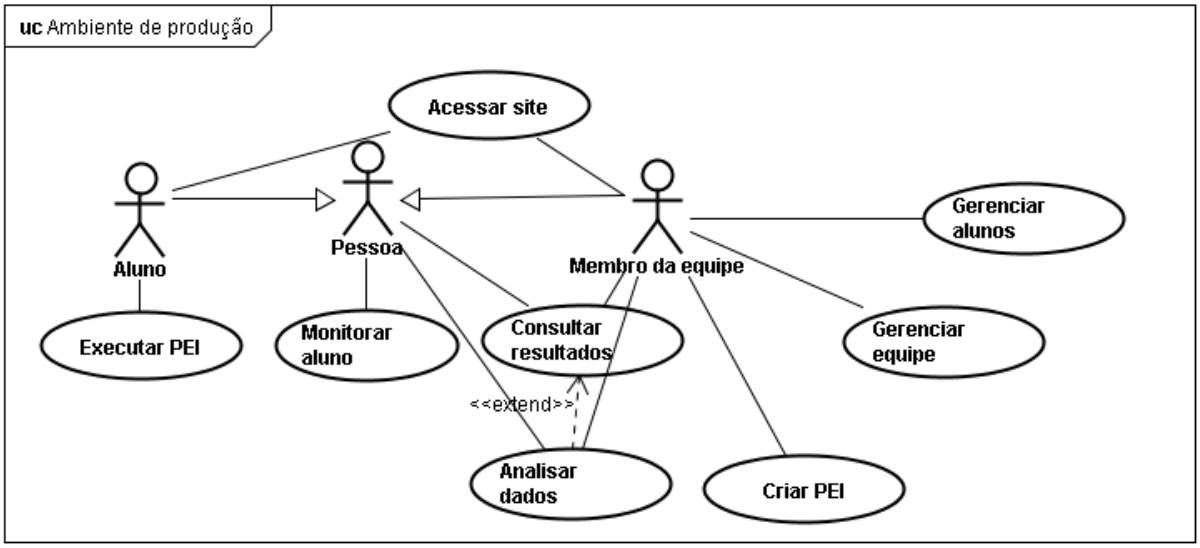


Figura 4 - Principais casos de uso do ambiente de produção

- **Criar PEI:** realizada pelo Especialista de domínio, consiste em organizar um conjunto de tentativas em que são apresentados os estímulos, e frente ao qual um Aluno deve apresentar uma resposta ativa. Para permitir um melhor agrupamento das tentativas, já que essas podem chegar às centenas dentro de um PEI, uma estrutura baseada no conceito de máquinas de estado finitas (MEF) deve ser adotada como modelo. Nesse modelo, as unidades de ensino (estados) são interligadas por transições que são realizadas mediante a satisfação de critérios (baseado, por exemplo, na quantidade de erros cometidos, tempo gasto para se completar tarefas etc.). As unidades de ensino são PEIs (conjunto de passos), passos (conjunto de blocos), bloco (conjunto de tentativas) e tentativa (configuração de estímulos visuais, textuais e/ou sonoros);
- **Executar PEI:** realizada pelo aluno de maneira presencial (sob supervisão de um tutor) ou remota, deve seguir as diretivas definidas pelos especialistas de domínio que criaram o PEI a ser executado, ou seja, executar as tentativas na ordem em que elas foram especificadas. A conseqüência à resposta do aluno deve ocorrer de forma imediata, ou seja, respostas incorretas devem receber mensagens de correção (geralmente seguidas de uma nova oportunidade) e respostas corretas devem receber mensagens de confirmação. O momento em que o aluno emite respostas deve ser registrado com grande precisão, já que seu comportamento deve poder ser analisado em detalhes e, eventualmente, ser observado em tempo-real por um tutor. Nenhuma interrupção ou atraso deve ocorrer enquanto o PEI estiver sendo

executado, já que isso pode desviar a atenção do aluno e prejudicar seu aprendizado (SKINNER, 1953). Exemplos de interrupções que não devem ocorrer são telas de carregamento (*loading*), alta latência de rede etc.;

- **Monitorar aluno:** realizado por tutores de maneira presencial ou remota, consiste em preparar o ambiente informatizado para que o aluno possa prontamente iniciar uma sessão, assim como supervisionar o processo de execução de PEIs. Inclui também tarefas mais ativas, tais como interromper uma sessão em caso de imprevistos ou avaliar subjetivamente o aluno em tentativas que envolvam respostas orais, tais como ditados e leituras. Durante a monitoria deve ser possível realizar anotações através do teclado a respeito do desempenho do aluno ou de eventuais dificuldades. Dependendo do conteúdo a ser ensinado, as avaliações subjetivas podem ser feitas tanto sincronamente (enquanto o aluno executa uma sessão) quanto assincronamente (após o aluno ter executado uma sessão). Além disso, vários tutores podem monitorar um aluno simultaneamente, assim como um único tutor pode monitorar vários alunos;
- **Consultar resultados:** realizada por tutores e especialistas de domínio, deve permitir a exibição dos dados coletados durante a execução dos PEIs, que incluem tanto a latência das respostas quanto o registro das próprias respostas. Esses dados podem servir tanto para fins de análise quanto para auxiliar na tomada de decisões a respeito do prosseguimento de um aluno em um PEI. Para que as consultas sejam relevantes é importante dispor de informações precisas sobre quais passos e blocos foram executados, quantos erros foram registrados, quais tentativas foram apresentadas e em que seqüência, quais foram respondidas de maneira adequada etc. Além disso, devem ser oferecidas diferentes visões sobre os dados coletados, tais como tabelas e planilhas, gráficos e relatórios;
- **Analisar dados:** realizada pelo especialista de domínio após a consulta dos resultados, consiste em extrair conhecimento a partir da observação dos dados. Essa análise deve poder ser feita com o auxílio de geração automática de estatísticas, provendo assim ao especialista de domínio um conjunto bastante amplo de recursos para orientar o prosseguimento de seu curso ou pesquisa;
- **Gerenciar equipe:** realizado pelo administrador de recursos humanos, envolve o cadastro de tutores e especialistas de domínio, assim como o controle das permissões que cada um deles terá dentro do sistema. Inclui também atribuir responsabi-

lidade sobre alunos a membros específicos, o que permite aos últimos tanto gerenciar alunos (cadastro e remoção) quanto à possibilidade de monitorá-los durante a aplicação de um PEI;

- **Gerenciar alunos:** também realizado pelo administrador de recursos humanos, consiste em manter os cadastros de alunos e de suas escolas e cidades. Consiste também em associar PEIs a alunos, visto que cada um desses pode estar associado simultaneamente a vários PEIs. Em casos excepcionais o administrador pode ainda, mediante justificativa, adiantar o passo em que o aluno se encontra em um PEI.

5.1.2.2 AMBIENTE DE DESENVOLVIMENTO

Uma visão geral dos casos de uso do ambiente de desenvolvimento pode ser vista na Figura 5.

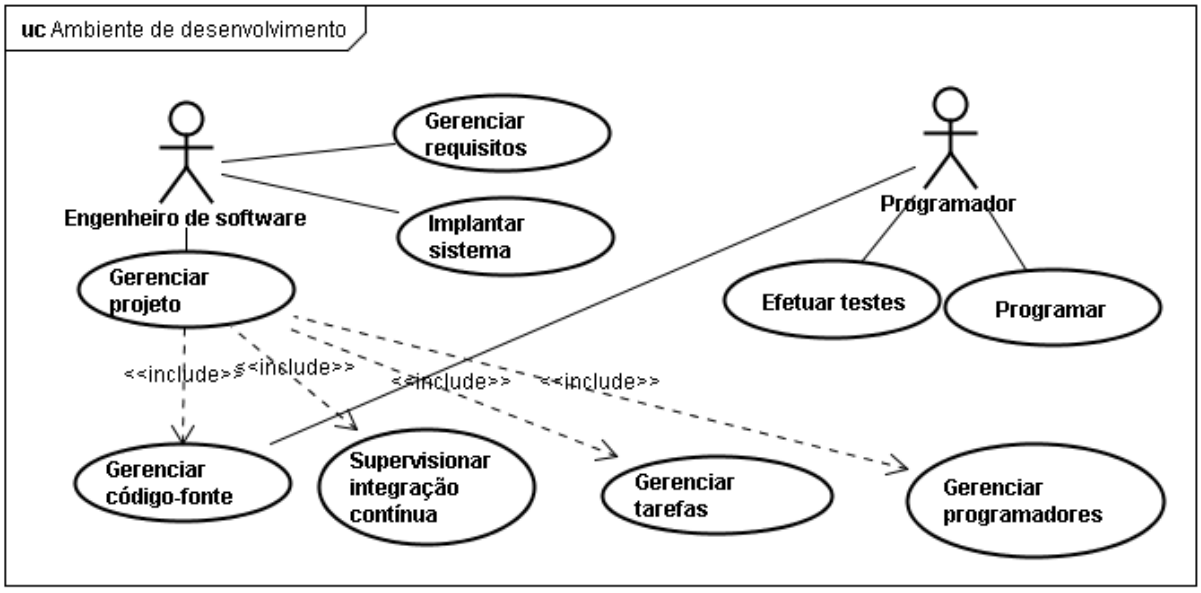


Figura 5 - Principais casos de uso do ambiente de desenvolvimento

Os principais casos de uso são:

- **Gerenciar código-fonte:** também conhecido como “controle de versão” e “controle de revisão”, é realizado pelo engenheiro de *software* e consiste em gerenciar mudanças não só no código-fonte, mas também em documentos, programas e ou-

tras informações armazenadas em computadores. Ferramentas de *software* para gerenciamento de código-fonte são essenciais na organização de projetos que envolvem muitos programadores.

- **Supervisionar integração contínua:** realizado pelo engenheiro de *software*, consiste em executar um conjunto de práticas que visam acelerar a entrega de *software* através da redução dos tempos de integração. Consiste também em automatizar essa prática e fazer com que ela se torne transparente aos programadores.
- **Gerenciar tarefas:** realizado pelo engenheiro de *software*, consiste em manter uma lista de *tickets* (tarefas, defeitos e melhorias) relacionados ao sistema, e a partir deles criar uma base de conhecimento contendo informações sobre problemas, resoluções etc.
- **Gerenciar programadores:** realizado pelo engenheiro de *software*, é a tarefa de cadastrar programadores e controlar o acesso desses aos recursos presentes no ambiente de desenvolvimento. Além do acesso, envolve o controle das horas gastas pelos programadores em cada tarefa, além de outras informações, tais como quantidade de horas gastas no projeto, quantidade de erros solucionados etc.
- **Gerenciar projeto:** realizado pelo engenheiro de *software*, inclui as tarefas já descritas de gerenciar o código-fonte, supervisionar a integração contínua, e gerenciar *tickets* e programadores. Seu objetivo é a finalização dentro do prazo e com sucesso dos objetivos do projeto.
- **Efetuar testes:** realizado pelo programador e também de maneira automatizada, consiste em elaborar e aplicar testes unitários, de sistema e de integração em todo o código-fonte do sistema e também na infra-estrutura de servidores. Para garantir ainda mais qualidade ao código-fonte, devem ser aplicados também analisadores estáticos e dinâmicos, cobertura de testes etc.
- **Programar:** realizado pelo programador, consiste em codificar soluções a partir dos requisitos, seguindo para isso as diretivas de arquitetura e de projeto estabelecidas pelo engenheiro de *software*;
- **Gerenciar requisitos:** realizado pelo engenheiro de *software*, representa a coleta e o refinamento dos requisitos originados nas reuniões com especialistas e tutores. Representa também a execução de sessões de teste e de experimentos a fim de observar padrões de uso do sistema;

- **Implantar sistema:** também realizado pelo engenheiro de *software*, é a implantação no ambiente de produção dos artefatos desenvolvidos no ambiente de desenvolvimento. Deve ser realizado apenas após extensivas sessões de teste, consultas a especialistas e execuções de verificações automatizadas (testes unitários, cobertura de testes, análise estática etc.).

5.1.3 REQUISITOS NÃO-FUNCIONAIS

Um requisito não-funcional especifica critérios que podem ser usados para se qualificar o sistema e suas funções (SOMMERVILLE, 2006), e não comportamentos específicos como os requisitos funcionais, sendo por isso conhecido também como “requisito de qualidade”. No âmbito deste projeto há vários requisitos não-funcionais muito importantes que, se suficientemente satisfeitos, contribuirão para a criação de sistemas de baixo custo, interoperáveis, escaláveis (dentre várias características desejáveis). Esses requisitos são:

- **Baixo custo:** Um dos itens mais importantes, o custo envolvido na criação, aplicação e gerenciamento de PEIs deve ser bastante baixo e permitir que essas tarefas sejam desempenhadas com o mínimo de investimento financeiro, tornando assim o projeto acessível a pequenas escolas e laboratórios.
- **Interoperabilidade:** Habilidade de dois ou mais sistemas ou componentes para trocarem informações e fazerem uso dessas informações. Como um dos objetivos deste projeto é aumentar a difusão de PEIs, não apenas PCs, mas também plataformas de *hardware* e *software* como TVDI e dispositivos móveis, devem ser suportadas e a comunicação entre elas possível. Para isso é indispensável tanto o uso de protocolos de comunicação livres e padronizados quanto, na impossibilidade do uso desses, acessórios que façam as devidas adaptações e traduções.
- **Disponibilidade:** Proporção de tempo em que um sistema está em condições de funcionamento, ou seja, a probabilidade dos usuários acessarem com sucesso um sistema num dado momento. Por se tratar de uma infra-estrutura voltada para *Web* e que centraliza o sistema em servidores, devem ser tomados cuidados adicionais para que problemas com quedas de energia e indisponibilidade de rede não afetem sessões de ensino em andamento nem as impeçam de serem iniciadas.

- **Escalabilidade:** Propriedade de um sistema ou de sua rede de comunicação de comportar cargas cada vez maiores de trabalho de maneira contínua e uniforme. Além disso, a estrutura do sistema deve poder ser facilmente expandida através da substituição ou adição de *hardware*. Escalabilidade deve ser um ponto crucial da infra-estrutura proposta, uma vez que ela é a base para dezenas de módulos, e que cada um desses módulos será usado por dezenas ou centenas de usuários simultaneamente. Assim, as escolhas de *hardware* e *software* no início do projeto devem levar em conta como eles se comportarão em ambientes de uso intensivo, e atenção especial deve ser atribuída a características como balanceamento de carga, redundância, tolerância a falhas etc.
- **Latência de rede:** Também conhecida como tempo de resposta de rede, é o tempo que um sistema leva para reagir a uma entrada através de uma rede de computadores. No contexto deste projeto, latência é o intervalo entre o instante em que um usuário requisita uma resposta do sistema e o instante em que essa resposta começa a ser recebida. Durante a execução de uma sessão de ensino é indispensável que as conseqüências às suas respostas sejam recebidas imediatamente, ou seja, com uma latência pequena o suficientemente de tal modo que o aluno não seja interrompido e não tenha sua atenção desviada.
- **Freqüência de acesso à rede:** é a quantidade de vezes em que o sistema acessa a rede de comunicação em um determinado período de tempo. Esse requisito torna-se mais importante à medida que as redes de comunicação passam a ser mais caras e com limite de transmissão de dados, tais como redes de telefonia celular. Logo, o acesso às redes de comunicação deve ser minimizado, valendo-se para isso de técnicas como envio de dados em lote, *caches* locais etc.
- **Confiabilidade / *Backup*:** realização de cópias dos dados afim de que estes sejam usados para recuperar o conteúdo original caso esse seja perdido ou danificado. Os principais tipos de dados a serem armazenados são as descrições dos programas de ensino, os desempenhos dos alunos e o código-fonte do sistema.
- **Usabilidade:** a facilidade com que uma ferramenta pode ser usada para se atingir um objetivo específico. Esse requisito é especialmente importante, pois a infra-estrutura proposta será usada por pessoas com conhecimento em informática (programadores e engenheiros de *software*), especialistas de domínio, crianças, deficientes mentais, visuais e auditivos.

- **Documentação:** texto escrito que acompanha sistemas computacionais e que explica, entre outras coisas, como ele foi construído e como pode ser operado, ocupando assim uma posição de destaque na engenharia de *software*. Fundamental para que o projeto proposto mantenha-se em atividade em longo prazo, pois tanto programadores quanto especialistas de domínio poderão contar com instruções precisas a respeito da infra-estrutura.
- **Licença de *Software*/Documentação Livre:** *software*/documentação livre é aquele que pode ser usado, estudado e modificado sem restrições, e que pode ser copiado e redistribuído em sua forma modificada ou não modificada sem restrições, ou com restrições mínimas apenas para garantir que os próximos proprietários também possam ter essa liberdade. Essas condições promovem os direitos e a liberdade dos usuários de acessarem e modificarem *software* e documentação, o que é compatível com os objetivos deste projeto, que são entre outros difundir conhecimento.

5.2 ARQUITETURA DE *SOFTWARE*

Uma vez definido o conjunto de técnicas e ferramentas necessárias para se planejar metas e organizar e gerenciar os recursos da infra-estrutura descrita nesta dissertação, é pertinente especificar uma arquitetura de *software* capaz de dar suporte aos componentes de *software* e às relações entre eles. A arquitetura de um sistema é sua estrutura e inclui componentes de *software*, suas propriedades externas e visíveis e a relação entre elas (ECKERSON, 1995).

Há várias razões pelas quais um engenheiro de *software* deve adotar um uso mais sistemático de uma arquitetura de *software* (BASS; CLEMENTS; KAZMAN, 1998), dentre as quais se destacam:

- Representações da arquitetura de *software* são um ativador para a comunicação entre as partes interessadas no desenvolvimento de um sistema computacional.
- A arquitetura destaca decisões de projeto feitas no início que terão um impacto profundo em todo o trabalho de engenharia de *software* que se segue e, também importante, no sucesso definitivo do sistema como entidade operacional.

- A arquitetura constitui um modelo relativamente pequeno e inteligível sobre como o sistema é estruturado e como seus componentes trabalham juntos.

A arquitetura de *software*, também descrita como projeto estratégico, é uma atividade voltada para critérios globais, tais como paradigmas de programação, padrões arquiteturais, princípios de projeto etc. (EDEN; KAZMAN, 2003). O paradigma de programação adotado em todo este trabalho foi a orientação a objetos, visto que é o padrão *de facto* para o desenvolvimento de sistemas empresariais. Nas subseções seguintes serão descritos os outros critérios aplicados neste trabalho, tais como padrões arquiteturais.

5.2.1 PADRÕES ARQUITETURAIS

Arquiteturas de *software* geralmente são derivadas de sistemas que compartilham um conjunto similar de objetivos. Essa similaridade pode ser descrita como um estilo ou padrão arquitetural, e representa oportunidades para a reutilização das experiências envolvidas na realização das arquiteturas. Padrões de arquitetura de *software* estabelecem soluções para problemas arquiteturais em engenharia de *software* (BUSCHMANN, 1996), e proporcionam descrições de elementos, a relação entre eles e as restrições sobre seus usos.

Além da reutilização da experiência, a aplicação de padrões arquiteturais facilita em muito a tarefa de um arquiteto de *software*, já que um padrão é normalmente documentado em termo das motivações para seu uso e de sua estrutura e comportamento. Assim, neste trabalho foram utilizados padrões arquiteturais consolidados para formar uma solução que firmasse as bases para a realização dos requisitos do domínio. A arquitetura de *software* do ambiente proposto é baseada principalmente em três padrões arquiteturais: multicamadas, MVC e SOA.

5.2.1.1 MULTICAMADAS

A arquitetura multicamadas (ECKERSON, 1995), também conhecida como *n-tier*, é uma arquitetura cliente-servidor na qual as camadas de aplicações cliente, servidor de

aplicações e sistemas de informação são separadas logicamente. O uso mais comum dessa arquitetura é conhecido como 3-camadas (*3-tier*).

Há certa confusão sobre o uso dos termos "*tier*" e "*layer*" com o significado de "camada". Porém, no escopo deste trabalho adota-se a visão de que um *layer* é um mecanismo de estruturação lógica para elementos que formam a solução de *software*, enquanto um *tier* é um mecanismo de estruturação física para infra-estruturas de sistemas (MICROSOFT, 2009). Esse padrão arquitetural é às vezes definido explicitamente como multicamadas distribuídas (JENDROCK, 2006), ressaltando que as camadas descritas podem estar localizadas em computadores separados que se comunicam através de uma rede.

O padrão 3-camadas estende o tradicional *cliente-servidor* ao incluir um servidor de aplicações entre as aplicações cliente e o servidor de armazenamento, como pode ser percebido a partir da Figura 6.

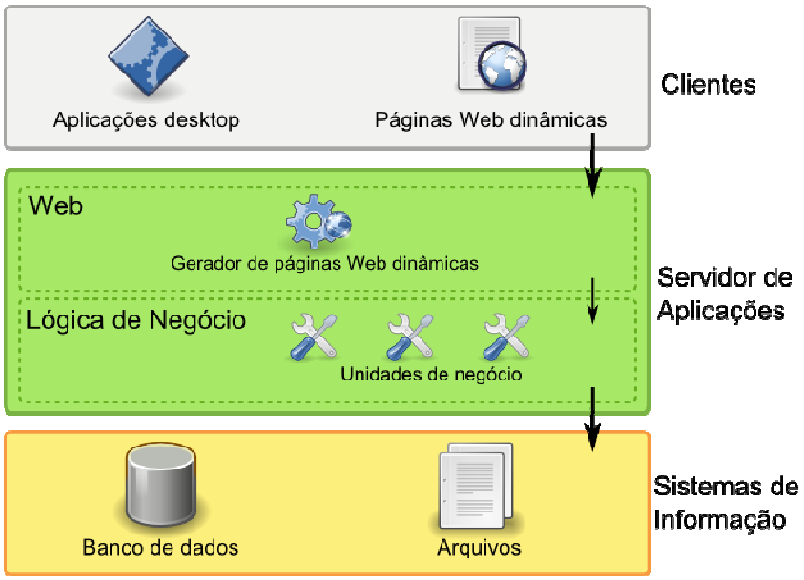


Figura 6 - Arquitetura multicamadas

As três camadas são as seguintes:

1. **Aplicações cliente**: o nível mais alto dentre as camadas. É onde estão as aplicações executadas nos computadores dos clientes, tais como aplicações *desktop* e páginas HTML dinâmicas;
2. **Servidor de aplicação**: é onde os conceitos do domínio da aplicação são representados, tais como entidades e lógica de negócio, e inclui a camada *Web*, que é onde as páginas dinâmicas são criadas;
3. **Sistemas de informação (acesso a dados)**: onde informações são mantidas em forma de arquivos, bancos de dados e outras formas de repositório.

Além das vantagens bem conhecidas de se projetar sistemas modulares com interfaces bem definidas, arquiteturas multicamadas permitem atualizações e substituições fáceis e independentes em cada uma de suas camadas, o que permite que novos requisitos sejam realizados mais facilmente e que se modernize toda a infra-estrutura de maneira gradual. Uma mudança no sistema operacional de um servidor, por exemplo, não afeta o código-fonte de uma aplicação cliente.

A arquitetura multicamadas foi usada extensivamente no escopo deste projeto e permitiu a cada camada (aplicações cliente, servidores de aplicação e acesso a dados) evoluir sem que isso influenciasse diretamente as outras camadas. Um exemplo prático desse tipo de benefício foi a substituição, pouco antes da metade do projeto, do SGBD (Sistema Gerenciador de Bancos de Dados) oficial, que foi concretizada em poucos minutos e sem que as aplicações cliente e o servidor de aplicações fossem alterados.

Algumas customizações foram necessárias na arquitetura multicamadas básica, como pode ser observado na Figura 7. Nessa nova arquitetura, foi incluído na camada de aplicações cliente suporte a aplicações de dispositivos móveis e de TV Digital. Além disso, foi adicionado suporte ao outro padrão arquitetural, a orientação a serviços (descrita adiante), que permitiu o uso de serviços *Web* tanto a partir das aplicações cliente quanto do próprio servidor de aplicações.

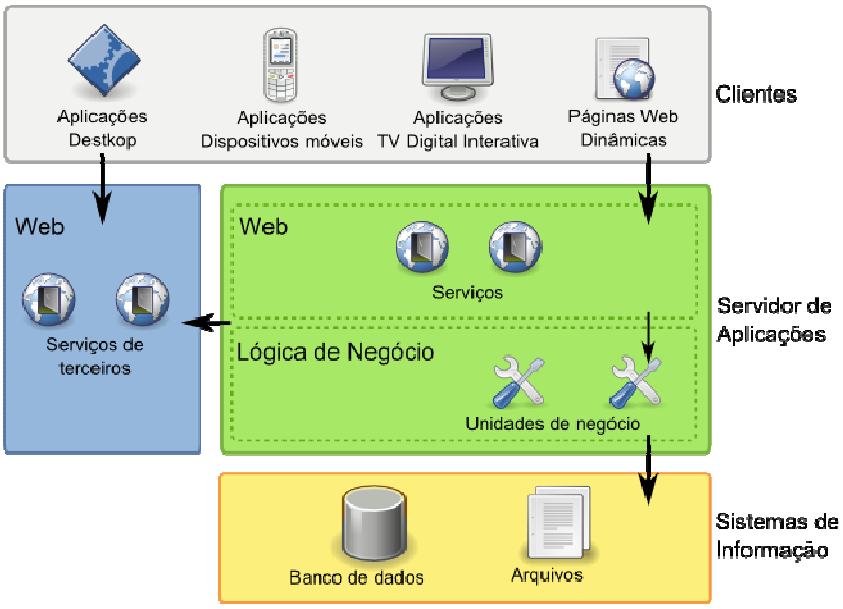


Figura 7 - Arquitetura multicamadas modificada

Mais detalhes sobre a aplicação desse padrão arquitetural, principalmente no que se refere ao servidor de aplicações e aos *proxies*, podem ser encontrados no capítulo 5.2.3 .

5.2.1.2 MVC

O padrão MVC (*Model-View-Controller*, ou *Modelo-Visão-Controlador*) é um padrão arquitetural cujo objetivo é isolar lógica de negócio, entrada e apresentação, permitindo assim independência no desenvolvimento, teste a manutenção de cada camada (REENSKAUG, 2009). Em uma aplicação MVC, cada modelo é associado a uma ou mais visões adequadas para apresentação. Quando um modelo tem seu estado mudado, ele notifica as visões associadas para que elas se atualizem. O controlador é responsável por iniciar requisições de mudanças e por prover dados necessários ao modelo.

Apesar de o MVC possuir variantes, o fluxo nesse padrão tem como objetivo realizar uma ponte entre o modelo mental do usuário (que tem a impressão de se estar manipulando diretamente o domínio da informação) e o modelo digital que existe no computador. Seu esquema padrão costuma seguir o que está representado na Figura 8:

1. O usuário interage com a interface de usuário ou ferramenta (por exemplo, pressionando o botão do *mouse*);
2. O controlador trata o evento de clique recebido da interface de usuário e notifica o modelo, eventualmente mudando o estado do modelo;
3. A visão obtém dados atualizados do modelo, exibe as novas informações e aguarda outras interações do usuário.

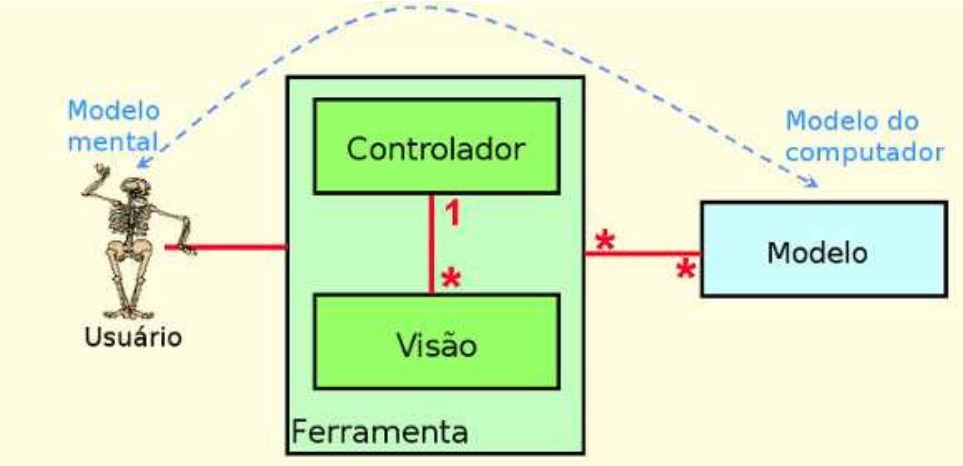


Figura 8 - Modelo MVC (adaptado do esboço original de (REENSKAUG, 2009))

A princípio as três camadas do MVC podem parecer semelhantes às do 3-camadas, porém uma regra fundamental no 3-camadas é que uma aplicação cliente nunca se comunica diretamente com a camada de dados, mas sim com a camada de aplicação. Logo, a 3-camadas pode ser considerada linear, ao contrário do MVC que pode ser considerada triangular (a visão envia atualizações para o controlador, o controlador atualiza o modelo, e a visão obtém dados atualizados do modelo).

O padrão MVC foi utilizado principalmente nas aplicações cliente desta proposta, e está presente em praticamente todos os *frameworks* para desenvolvimento de aplicações *desktop* e páginas *Web* dinâmicas. O uso de MVC proporcionou uma clara separação de interesses e possibilitou o desenvolvimento em paralelo das visões (tabelas, formulários etc.) e dos controladores (mecanismos de *callback* e *observer*s), aumentando assim a produtividade da equipe de desenvolvimento e também da manutenção.

5.2.1.3 SOA

Uma arquitetura orientada a serviços (*Service-Oriented Architecture* – SOA) é um padrão arquitetural para o desenvolvimento de sistemas de *software* que provê uma infraestrutura para troca de mensagens entre diferentes aplicações ou serviços (ERL, 2005). Para isso, são utilizadas unidades de negócio interoperáveis, empacotadas e com interfaces bem definidas conhecidas como serviços. Serviços podem se comunicar entre si, inclusive através de outros serviços em outros computadores, e independem da linguagem de programação utilizada para a codificação da lógica de negócio, assim como independem do sistema operacional ou plataforma de desenvolvimento em que estão hospedados.

Arquiteturas de *software* têm evoluído (BARESI; NITTO; GHEZZI, 2006), indo de estáticas, monolíticas e centralizadas para dinâmicas, modulares e distribuídas, essas últimas características atendidas por SOA. Não raramente SOA é discutida como sendo uma evolução em relação ao conceito de computação distribuída e programação modular, visto que é uma arquitetura flexível, largamente padronizada e que suporta compartilhamento de dados e conexão entre aplicações em diferentes sistemas.

Uma arquitetura orientada a serviços é divida em quatro partes principais (KRAFZIG; BANKE; SLAMA, 2004), e a relação entre eles é ilustrada na Figura 9:

- **Interface da aplicação**: aciona processos na camada de negócios
- **Serviço**: unidade de negócio composta por contratos, codificações e interfaces
- **Repositório de serviços**: local onde se encontram contratos de serviços
- **Barramento de serviços**: conexão da interface da aplicação com os serviços

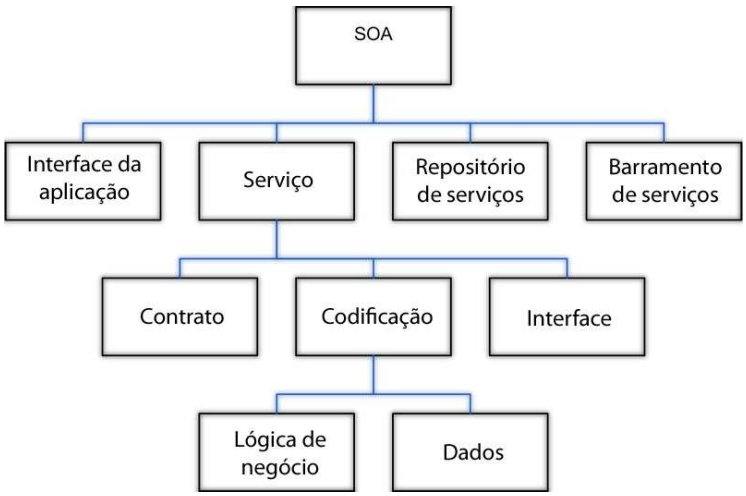


Figura 9 - Elementos de SOA (adaptado de (KRAFZIG, 2005))

O uso de uma SOA não está preso a um padrão ou tecnologia específica, podendo ser realizada através do uso de uma ou mais dentre várias tecnologias disponíveis no mercado, tais como DCOM (*Distributed Component Object Model*)[1], CORBA (*Common Object Request Broker Architecture*)[2] e *Web Services* (W3C, 2009a).

Um Serviço *Web* (*Web Service*) é um sistema de *software* que suporta a interação máquina-máquina sobre uma rede, utilizando para isso um protocolo de comunicação baseado, na maioria dos casos, em XML. Como Serviços *Web* são construídos sob vários padrões largamente adotados pela indústria de *software*, essa comunicação pode inclusive ser feita em uma variedade muito grande de plataformas, *frameworks* e linguagens de programação. Cada Serviço *Web* é identificado por um URI (W3C, 2009b), cujas interfaces e ligações são definidas e descritas. Sua definição pode ser descoberta por outros sistemas de *software*, e esses podem interagir com o serviço em uma maneira prescrita por essa definição.

Uma arquitetura básica de Serviços *Web* (W3C, 2009c), que é aquela que promove interações entre os agentes de *software* sem o uso de extensões, deve permitir:

- Troca de mensagens;

[1] http://www.microsoft.com/com/default.mspx

[2] http://www.omg.org/spec/CORBA/3.1

- Descrição de Serviços *Web*;
- Publicação e descoberta de descrições de *Serviços Web*.

Agentes de *software* nessa arquitetura básica podem assumir os seguintes papéis:

- **Requisitante de serviços**: aquele que requisita a execução de um serviço;
- **Provedor de serviços**: oferece e processa a requisição de um serviço;
- **Publicador ou agência de descoberta de serviços**: onde descrições de Serviços *Web* são publicadas e disponibilizadas para busca e acesso.

A arquitetura básica está ilustrada na Figura 10:

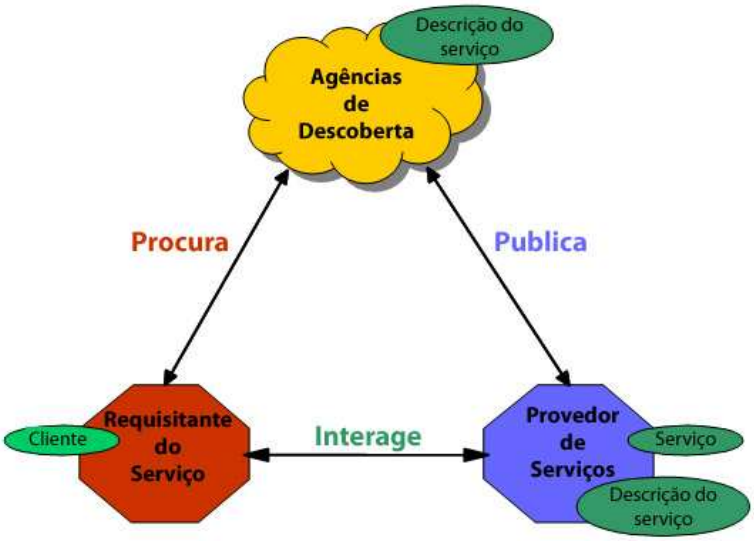


Figura 10 - Arquitetura básica de Serviços *Web*

Como citado anteriormente, para que uma arquitetura orientada a serviços utilizando serviços web seja realmente interoperável é necessário que sejam utilizados vários padrões consolidados pela indústria de desenvolvimento de *software*. Alguns desses principais padrões são:

- **HTTP** (*Hypertext Transfer Protocol*): é um protocolo de transporte de dados muito popular que funciona como uma das bases da *WorldWideWeb*. Seu principal atrativo é a simplicidade e eficiência (W3C, 2009d);
- **XML** (*eXtensible Markup Language*): é uma linguagem de marcação de dados que, como HTML, utiliza *tags*, permitindo assim a estruturação ou codificação de conteúdo (W3C 2009e);
- **SOAP** (*Simple Object Access Protocol* ou, mais recentemente, *Service Oriented Access Protocol*): define o formato das mensagens para comunicação entre as en-

tidades de uma arquitetura básica. Sua forma mais comum utiliza o envio de conteúdo XML sobre HTTP (W3C, 2009c);

- **WSDL** (*Web Service Description Language*): é utilizado para descrever um Serviço *Web* quanto às suas interfaces, formatos de dados, protocolo de transporte e localização na Internet (W3C, 2009f);
- **UDDI** (*Universal Description, Discovery and Integration*): oferece uma maneira comum para se publicar, descobrir e recuperar Serviços Web. Seu uso não é obrigatório em uma arquitetura em casos onde um requisitante está interessado em um serviço cujo endereço é conhecido com antecedência (OASIS, 2009).

A arquitetura de Serviços *Web* utilizada nesta infra-estrutura possui algumas diferenças em relação à arquitetura básica exibida na Figura 10. A primeira delas é que a princípio não será utilizada descoberta dinâmica de serviços, já que a flexibilidade adicional que esse recurso oferece atualmente não é necessária (os serviços adequados são conhecidos com antecedência). Além disso, em alguns casos a descoberta dinâmica pode se tornar prejudicial no domínio de PEIs, já que serviços de terceiros podem estar indisponíveis e inviabilizarem o início de uma sessão de ensino. Finalmente, deseja-se evitar a complexidade associada. A arquitetura modificada, sem a agência de descobertas e as ações de publicar e procurar serviços, pode ser vista na seção “a” da Figura 11.

A segunda alteração em relação à arquitetura básica é a adição de *proxies*, que são intermediários entre os requisitantes e os provedores de serviços. Esse intermédio se faz necessário em casos onde os clientes não suportam algum dos padrões utilizados em Serviços *Web* (SOAP, WSDL etc.), ou ainda o fazem a um custo computacional elevado. Intermediários também são desejáveis a fim de se adaptar e personalizar conteúdo, ou até mesmo aumentar o desempenho da comunicação através do uso de *caches*. A arquitetura modificada, com a inclusão de um *proxy* entre o requisitante e o provedor de serviço, pode ser vista na seção “b” da Figura 11.

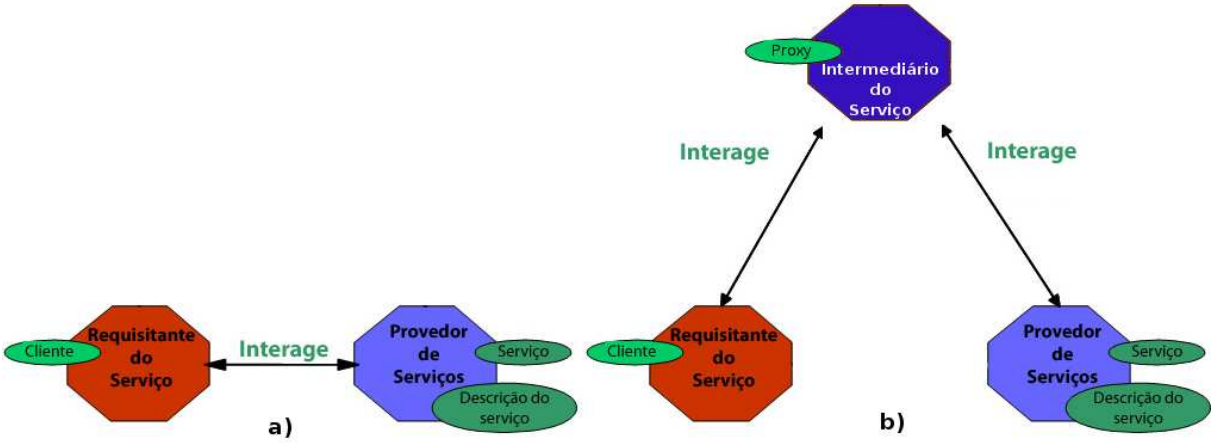


Figura 11 - Arquiteturas de Serviços *Web* modificadas

5.2.2 AMBIENTE DE PRODUÇÃO

Após definidos os padrões arquiteturais que serão utilizados no decorrer deste trabalho, é possível modelar com mais clareza a implantação física do ambiente de produção. O objetivo dessa modelagem é apresentar uma visão estática da infra-estrutura em tempo de execução, e detalhar os componentes sendo executados em cada nó do ambiente. Em outras palavras, serão descritos o *hardware* do sistema, o *software* instalado em cada um deles e os eventuais *middlewares* e *proxies* usados para conectar as máquinas umas às outras.

Uma visão geral do ambiente de produção pode ser visto na Figura 12, onde estão destacados os elementos (em sua maior parte servidores) da arquitetura: produção, mensagens, banco de dados, *backup*, *site*, *proxies* e aplicações cliente.

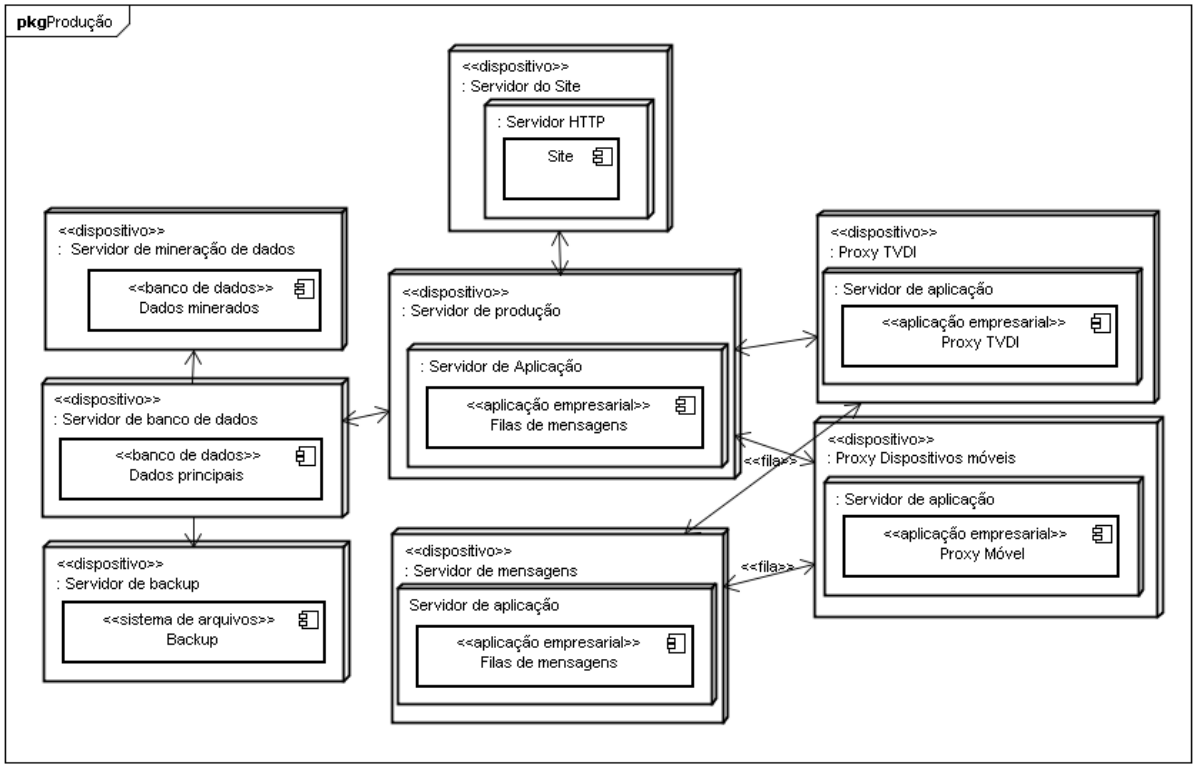


Figura 12 - Modelo de implantação do ambiente de produção

5.2.2.1 PRODUÇÃO

O servidor de produção (ou *servidor*), assim como foi representado no servidor de aplicações da Figura 7, é o núcleo do ambiente de produção e contém as subcamadas de negócio e *Web*. Uma visão detalhada desse servidor pode ser vista na Figura 13.

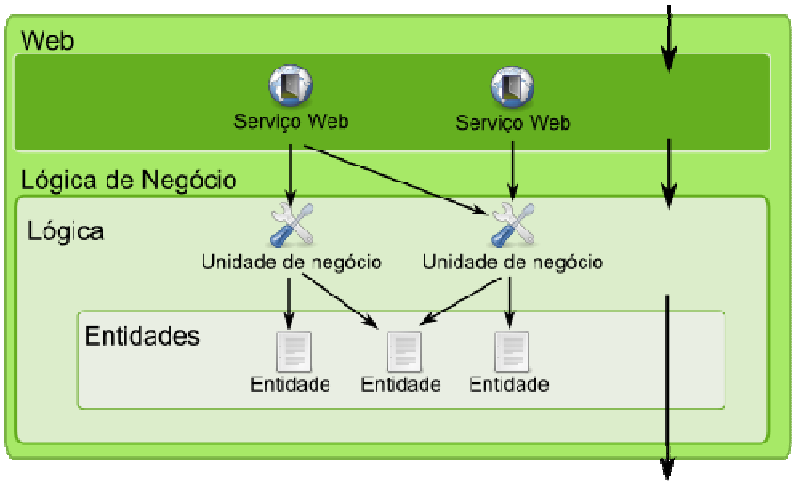


Figura 13 - Detalhamento do servidor de aplicações

A subcamada de lógica de negócio pode ser acessada apenas a partir da subcamada *Web* e possui mais duas subcamadas, a de entidades e a de lógica. A subcamada de entidades possui as definições dos tipos de interesse no domínio, tais como *Pessoa*, *Aluno*, *Tutor* etc. Já a subcamada de lógica é composta de componentes que, além de possuírem toda a lógica referente ao domínio, realizam a persistência e recuperação de dados a partir da camada de sistemas de informação.

Já a subcamada *Web* realiza a interface do servidor de aplicações com as aplicações cliente, seja através de Serviços *Web* ou de geradores de páginas *Web*. Isso faz com que a subcamada *Web* seja o único canal de acesso aos dados do sistema, facilitando assim eventuais redirecionamentos, validações de segurança etc. Usualmente cada aplicação cliente (módulo) utiliza um serviço, mas não é incomum que haja módulos que utilizem dois ou mais desses serviços.

Ambas as subcamadas podem, quando conveniente, acessar Serviços *Web* de terceiros, tais como validadores de dados, calendários etc.

5.2.2.2 SERVIDOR DE BANCO DE DADOS

O servidor de banco de dados, que representa um ou mais SGBDs (Sistema Gerenciador de Bancos de Dados) relacionais trabalhando em paralelo, é o responsável por armazenar todas as informações do ambiente de produção, sejam elas PEIs, registros de execuções de sessão, cadastros de alunos etc.

Seguindo à risca o modelo multicamadas, o SGBD pode ser acessado apenas pela subcamada de lógica de negócio do servidor de aplicações (mais especificamente na subcamada de lógica), o que garante uma grande flexibilidade à arquitetura e, ao mesmo tempo, uma clara separação de responsabilidades. Outro efeito positivo dessa decisão é que comandos SQL nunca serão diretamente efetuados no SGBD, já que essa tarefa será feita indiretamente pelo provedor de persistência de dados.

5.2.2.3 SERVIDOR DE MENSAGENS

O servidor de mensagens tem como única finalidade viabilizar a troca de dados entre tutores e alunos durante a execução de uma sessão. Uma mensagem é volátil (não é registrada em lugar algum) e é trocada sempre que um aluno ou tutor executa uma ação, por exemplo, “aluno X selecionou estímulo Y”. Essa troca de mensagens, que é transparente aos usuários, mantém as interfaces de usuário de alunos e tutores sincronizadas entre si.

Cada mensagem é enviada ao servidor de mensagens por uma aplicação cliente. O servidor, por sua vez, posiciona cada mensagem recebida em uma ou mais filas, e as entrega a outra aplicação cliente quando necessário. Uma ilustração dessa situação pode ser vista na Figura 14, onde se observa que a entrega de mensagens é assíncrona (ao contrário da comunicação síncrona entre cliente e servidor), o que é ideal para o tipo de situação observada durante uma sessão de ensino. Assim, durante uma sessão tanto o computador do aluno quanto o do tutor assumem o papel, simultaneamente, de cliente e servidor.

Figura 14 - Comunicação através de mensagens usando o servidor de mensagens

Devido ao fato das mensagens serem voláteis e não precisarem ser armazenadas no banco de dados, seria desnecessário utilizar o servidor de aplicações para essa finalidade, logo o servidor de mensagens serve tanto para isolar as responsabilidades quanto para aliviar a carga do sistema em geral.

5.2.2.4 SITE

O *site*, que de preferência deve estar em um servidor dedicado, é o ponto de partida para usuários (tutores e alunos) que desejam executar módulos do sistema. Isso significa que os módulos só poderão ser acessados através do *site* (mediante fornecimento de dados), que por sua vez deve se encarregar de exibir apenas os módulos relevantes ao usuário.

A partir do *site* pode-se também ler notícias a respeito do sistema, conferir a situação do projeto (quantidade total de alunos atendidos, número de PEIs disponíveis etc.) e acessar manuais e outros documentos.

5.2.2.5 BACKUP

O objetivo do servidor de *backup* é prover maior confiabilidade e segurança ao sistema, realizando cópias dos dados em intervalos regulares. As cópias dos dados do domínio são um recurso indispensável para permitir, em caso de perda ou danificação do SGBD, a recuperação dos dados originais. Os principais tipos de dados a serem armazenados são as descrições dos PEIs e os desempenhos dos alunos.

Porém, apenas manter um servidor de *backup* não é eficaz em eventuais acidentes onde todos os servidores possam ser danificados, tais como incêndios, enchentes ou vandalismo. Para isso, o servidor de *backup* realiza também cópias externas dos dados, tanto sob comando do usuário em mídias óticas (DVDs etc.) como também automaticamente em um servidor remoto.

5.2.2.6 PROXIES

Proxies realizam o intermédio entre os clientes e o servidor de aplicações, e usualmente são necessários apenas entre o servidor de aplicações e aplicações cliente executando em plataformas não-convencionais, tais como dispositivos móveis e TVDI. As principais

tarefas que um *proxy* pode executar são a conversão de protocolo, a manutenção de um *cache* de dados e adaptação de conteúdo. Um exemplo de *proxy* pode ser encontrado na Figura 15.

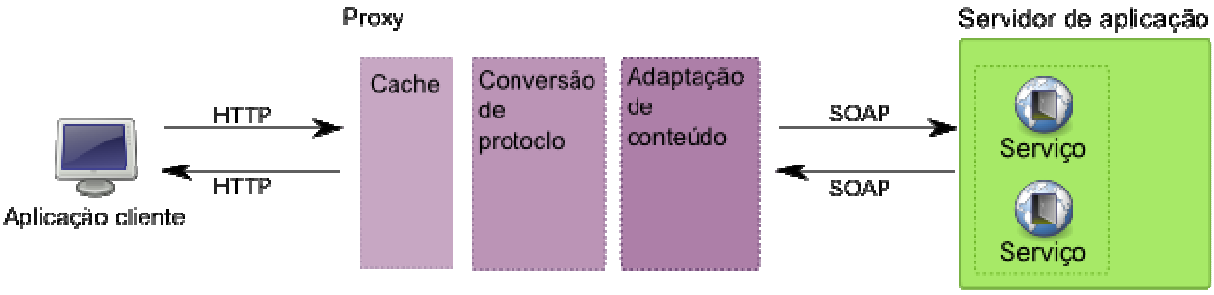


Figura 15 - Exemplo de *proxy* entre aplicação cliente e servidor de aplicação

A conversão de protocolos de comunicação de rede é indispensável para que plataformas de *hardware* como a TVDI e alguns dispositivos móveis possam acessar os serviços oferecidos. Essas plataformas, que em geral possuem menos recursos que um PC, contam apenas com recursos básicos para acesso à rede normalmente insuficientes para realizar transações com protocolos mais sofisticados, tais como SOAP.

Além da conversão de protocolos, os *proxies* podem ser utilizados como *cache* de dados. Esse recurso é de grande valor para armazenar dados que não mudam durante o decorrer do tempo, tais como PEIs já disponibilizados, e evitam que o servidor de aplicações realize trabalho desnecessário acessando o banco de dados e refazendo várias verificações. É útil também para auxiliar na conversão de protocolos, já que as formas convertidas dos dados podem ser armazenadas e reutilizadas.

Finalmente, o papel da adaptação de conteúdo é adaptar mídias de áudio e vídeo para os formatos mais adequados suportados pela TVDI e pelos dispositivos móveis. Aparelhos celulares, por exemplo, possuem telas minúsculas e com resolução reduzida, portanto seria desnecessário (e um desperdício de banda de rede) receber uma fotografia em alta definição para depois adaptá-la no próprio aparelho.

5.2.2.7 APLICAÇÕES CLIENTE

As aplicações cliente são representadas tanto por páginas *Web* dinâmicas quanto por aplicações para *desktop*, dispositivos móveis e TVDI, sendo que a maioria deles se co-

municam com o servidor de aplicações exclusivamente através de serviços *Web*. Em casos de monitoria essas aplicações acessam também o servidor de mensagens.

Apesar dos clientes serem representados como uma entidade homogênea, na realidade existe entre eles várias diferenças fundamentais que justificam a existência dos *proxies*. A plataforma PC é bastante entendida e conta com a maior quantidade de recursos (processamento, memória e bibliotecas de *software*) dentre todas as outras abordadas neste trabalho, logo não será mais especificada. Já as plataformas de TVDI e de dispositivos móveis serão discutidas em detalhes, visto que exibem diferenças significativas em relação às mais tradicionais (PC e *Web*). Vale observar que, na prática, o uso dessas novas plataformas é interessante apenas para fins de execução de sessões pelos alunos, já que é isso que caracteriza o aumento da abrangência dos PEIs. Para fins de criação de PEIs, consulta de resultados e tarefas administrativas, será adotada exclusivamente a plataforma *Web*.

5.2.2.8 TV DIGITAL INTERATIVA

A TV Digital Interativa possui um papel importante na difusão de programas de ensino, porém sua plataforma de *hardware* difere significativamente daquela mais tradicional encontrada nos PCs. Além de contar com poder de processamento e de armazenamento reduzidos, o recebimento de dados é delimitado pelos canais de descida e de retorno.

Uma das principais diferenças da TVDI em relação à TV analógica são as funcionalidades computacionais disponíveis nos aparelhos de TV. Aplicativos interativos podem ser executados paralelamente ou não à programação normal. A partir deste ponto o termo STB (*set-top-box*) é utilizado para indicar o recurso computacional do televisor digital, quer esteja Embutido no próprio televisor ou em um dispositivo externo.

A forma com que a TVDI se insere no ambiente de produção pode ser vista em mais detalhes na Figura 16. Nela pode-se observar a presença de um *proxy* entre a TV e o servidor de aplicações. As principais funções do *proxy* neste contexto são intermediar a comunicação com o servidor de aplicações e submeter os dados obtidos para a emissora.

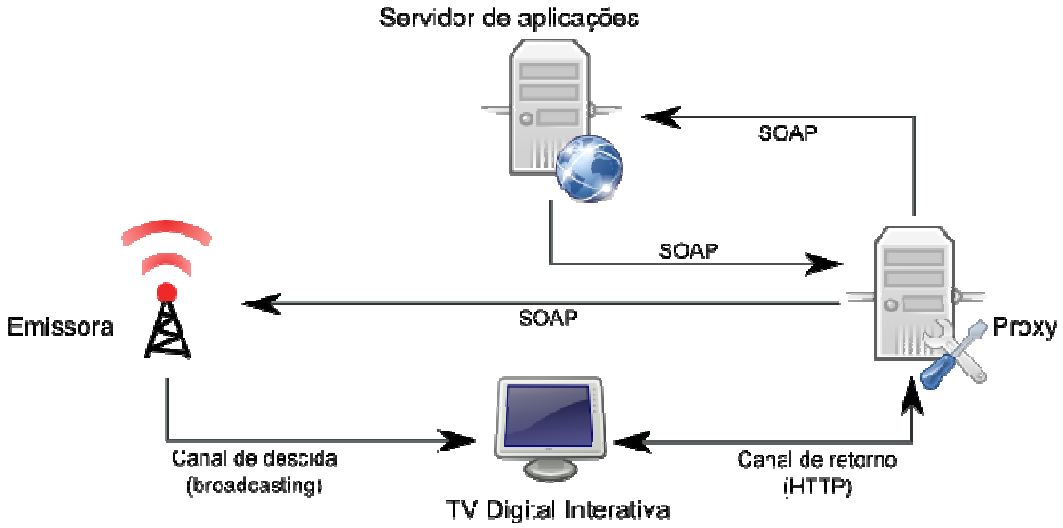


Figura 16 - Arquitetura de comunicação (TVDI)

O intermédio do STB com o servidor de aplicações é necessário visto que os Serviços *Web* não são acessíveis diretamente. Para isso, o *proxy* realiza a conversão de protocolos (de SOAP para HTTP, e vice-versa) e mantém um *cache* local com os dados obtidos a partir do servidor de aplicações (geralmente sessões de ensino).

Uma vez recebidas as informações do servidor de aplicações, a função do *proxy* é repassar esses dados para uma emissora, que por sua vez os transmitirá via canal de descida para os STBs. Esses dados devem ser identificados de modo que apenas o STB que requisitou uma sessão de ensino específica os receba, já que na transmissão via broadcasting os dados são difundidos a todos os receptores.

A utilização do canal de retorno tanto para o envio quanto para o recebimento de dados é possível, porém aproveitar o canal de *broadcast* para receber conteúdo de mídia rica pode ser mais econômico que exigir banda larga no canal de retorno.

5.2.2.9 DISPOSITIVOS MÓVEIS

Assim como a TVDI, dispositivos móveis são essenciais para uma maior difusão de programas de ensino. Entretanto, a maioria dos telefones celulares e *palmtops* possuem limitações ainda mais drásticas do que aquelas encontradas nos STBs, e logo requerem um tratamento especial nesta arquitetura.

As diferentes configurações de *hardware* e *software* levaram a indústria de desenvolvimento de *software* a criar soluções que facilitassem o processo de codificação de a-

plicações multiplataforma. Algumas plataformas que se destinam a esse fim são *Java ME* e *BREW* (*Binary Runtime Environment for Wireless*).

A forma com que dispositivos móveis se inserem no ambiente de produção podem ser vistas com mais detalhes na Figura 17. Nela pode-se observar a presença de um *proxy* entre apenas alguns tipos de dispositivos (básicos) e o servidor de aplicações, ao passo que outros tipos de dispositivos (avançados) realizam essa conexão diretamente.

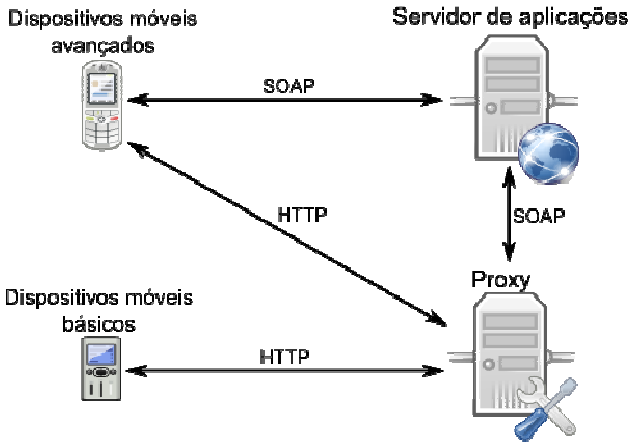


Figura 17 - Arquitetura de comunicação (dispositivos móveis)

Dispositivos móveis avançados, com recursos de *hardware* e *software* suficientes para o acesso direto a Serviços *Web*, representam ainda nos dias de hoje uma pequena fatia do total de aparelhos vendidos (TERGUJEFF, 2007). Porém, mesmo esses aparelhos podem impor restrições ao consumo de serviços, seja por não possuírem uma interface de programação suficientemente rica ou por disporem de recursos de processamento limitados.

Já os dispositivos móveis básicos requerem a presença de um *proxy* para a realização do intermédio com o servidor de aplicações. Para isso, o *proxy* realiza a conversão de protocolos (de SOAP para HTTP, e vice-versa) e mantém um *cache* local com os dados obtidos a partir do servidor de aplicações. Há outras opções para realizar a comunicação entre dispositivos móveis e Serviços *Web*, tais como o uso de bibliotecas de *software* como kSOAP[3], porém seu uso gera um aumento no número de dependências e também uma sobrecarga significativa no dispositivo. Uma vez recebidas as informações do servidor de aplicações, a função do *proxy* é repassar esses dados convertidos de volta para o dispositivo móvel.

[3] http://ksoap2.sourceforge.net

5.2.3 AMBIENTE DE DESENVOLVIMENTO

A infra-estrutura do ambiente de desenvolvimento é composta basicamente de servidores cujos objetivos são auxiliar engenheiros de *software* e programadores no desenvolvimento do sistema. Gerenciamento de projeto é uma disciplina de planejamento, organização e gerenciamento de recursos com o objetivo de se completar com sucesso os objetivos de um projeto (CLELAND; GAREIS, 2006).

Uma visão geral do ambiente de desenvolvimento pode ser visto na Figura 18, onde estão destacados os elementos (servidores) da arquitetura: gerenciamento de projeto, *backup*, repositório de código-fonte, *e-mail*, integração e testes.

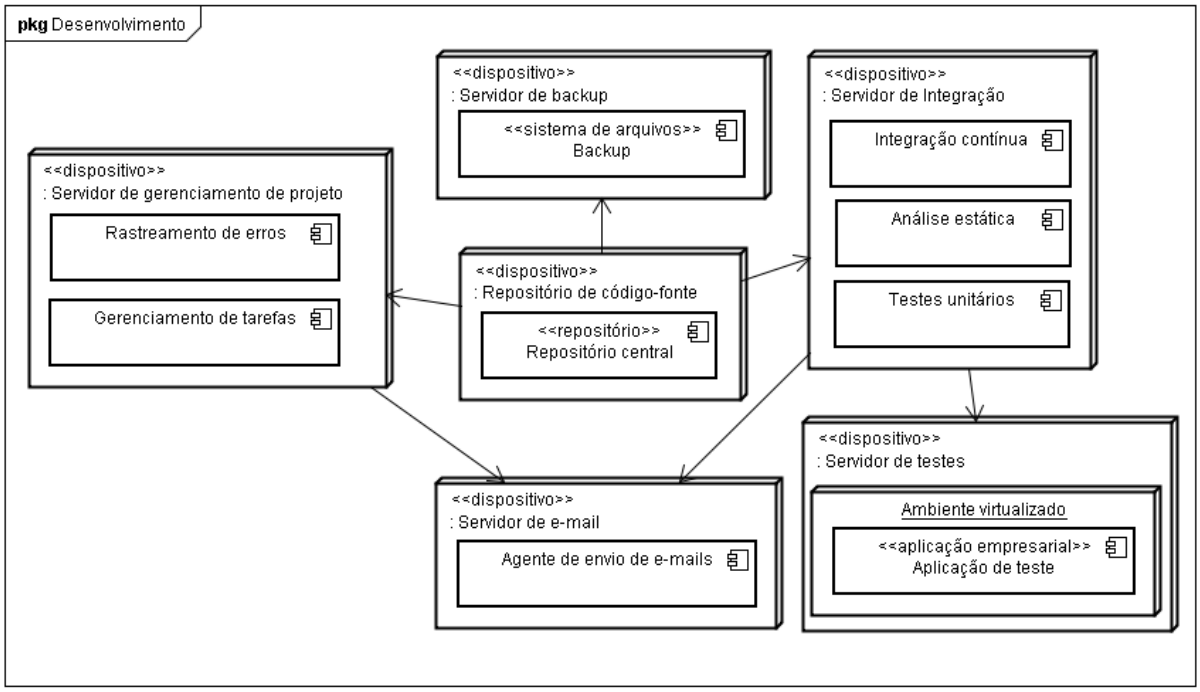


Figura 18 - Diagrama de implantação do ambiente de desenvolvimento

As atividades executadas a partir desses servidores, que persistem durante todo o processo de criação de *software* e o apóiam, são chamadas atividades guarda-chuva (Pressman 2001), e são definidas por um conjunto de tarefas adaptadas ao tipo de projeto e ao rigor com o qual a engenharia de *software* será aplicada. Para trabalhar em conjunto com a ferramenta de gerenciamento de projeto escolhida, foram adotadas algumas atividades guarda-chuva que serão descritas a seguir, e são oferecidas justificativas para a adoção de cada ferramenta.

5.2.3.1 CONTROLE DE REVISÕES

Também conhecido como controle de versões, é o gerenciamento de mudanças em documentos, programas, código-fonte e outras informações armazenadas como arquivos de computador. Mudanças são geralmente identificadas por um número de revisão (ou simplesmente "revisão") e são associadas ao momento em que foram feitas e quem as fez. Revisões podem ser comparadas, recuperadas, e em alguns tipos de arquivos (como código-fonte) fundidas.

Ferramentas de *software* para gerenciamento de código-fonte são essenciais na organização de projetos que envolvem muitos programadores, pois permitem que mais uma pessoa efetue alterações simultaneamente em um mesmo arquivo ou módulo, além de facilitar a integração dessas alterações em um único arquivo final. As ferramentas mais populares para essa finalidade incluem *Concurrent Versions System* (CVS)[4], *Subversion* (SVN)[5], *Git*[6], *Mercurial*[7] e *Perforce*[8]. *Subversion* se mostrou uma ótima solução para realizar o controle de revisões da infra-estrutura proposta, já que é *software* livre (licença Apache), atualmente é a mais popular dentre esse tipo de ferramenta e possui integração direta com a ferramenta de gerenciamento de projeto adotada (*Trac*).

O servidor de controle de revisões tem a função exclusiva de armazenar todo o código-fonte criado em todo o sistema, e pode ser acessado pelos programadores através da aplicação cliente do próprio *Subversion,* ou até mesmo por utilitários que integram o repositório ao gerenciador de janelas do computador do programador (tais como *TortoiseSVN*[9]). A Figura 19 mostra uma visão do repositório *Subversion* do sistema através da ferramenta *TortoiseSVN*.

---

[4] http://www.nongnu.org/cvs

[5] http://subversion.tigris.org

[6] http://git-scm.com

[7] http://mercurial.selenic.com/wiki

[8] http://www.perforce.com

[9] http://tortoisesvn.tigris.org

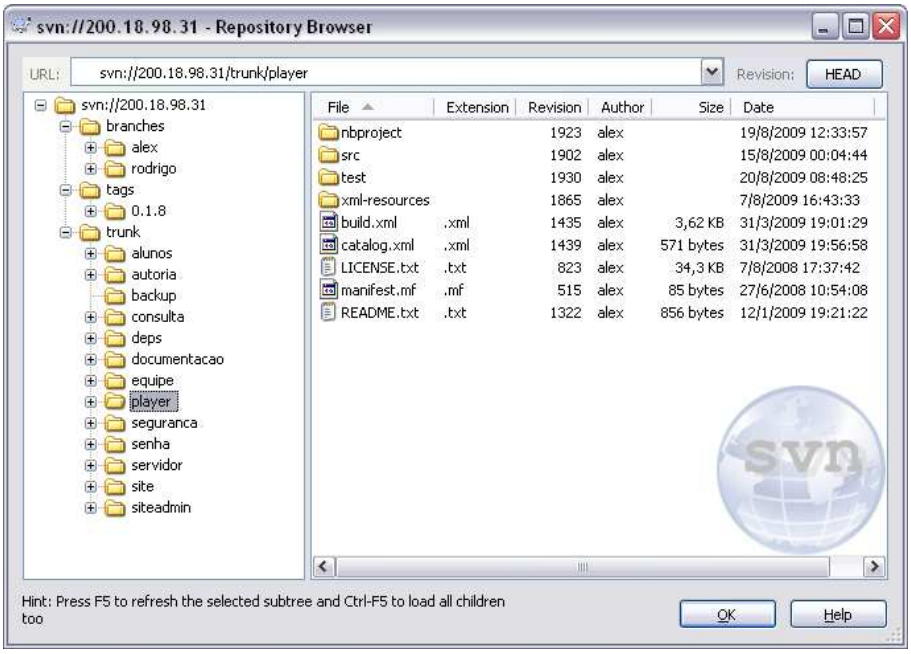


Figura 19 - Repositório *Subversion* sendo acessado através do *TortoiseSVN*

5.2.3.2 INTEGRAÇÃO

É uma prática para desenvolvimento de *software* onde os membros de uma equipe integram seu trabalho com freqüência levando a várias integrações por dia. Cada integração é verificada por uma construção automatizada para se detectar erros de integração o mais rápido possível. Muitas equipes de desenvolvimento consideram que esse processo reduz significativamente os problemas de integração e permite a uma equipe desenvolver *software* coeso mais rapidamente (FOWLER, 2009).

Ferramentas para integração contínua costumam se valer do mecanismo de construção presente em um projeto, ou seja, seu funcionamento é integrado a ferramentas do tipo *Ant* e *Maven*. Além de automatizar a construção, ferramentas de integração contínua também automatizam a realização de testes unitários, análise estática de código-fonte, análise de cobertura de testes, e notificação de engenheiros de *software* em caso de problemas.

Ferramentas populares com a finalidade de realizar integração contínua incluem *CruiseControl*[10], *Continuum*[11] e *Hudson*[12]. Para uso neste projeto, foi escolhido *Hudson*,

[10] http://cruisecontrol.sourceforge.net

[11] http://continuum.apache.org

[12] https://hudson.dev.java.net

pois além de ter a interface mais amigável possui *plug-ins* para integração com as outras ferramentas, tais como *Trac* e *Subversion*.

Uma vez a cada hora este servidor procura o repositório de código-fonte à procura de alterações e, caso alguma seja encontrada, ele inicia automaticamente o processo de construção de todo o sistema. Caso a construção seja bem sucedida, o próprio servidor inicia a execução de testes unitários, cobertura de testes, análise estática etc. Ao final, o sistema é implantado em um servidor de testes para que testes de sistema possam ser efetuados por programadores e especialistas de domínio.

Engenheiros de *software* e programadores são notificados sobre eventuais erros ao se integrar o sistema, normalmente através de *e-mails*. Para facilitar o trabalho da equipe de desenvolvedores em relação a notificações, foi instalado nos computadores de programação um *plug-in* do *Hudson* (*Hudson Build Monitor*[13]) para o navegador *Web Mozilla Firefox*[14]. Além disso, o *Hudson* foi integrado à ferramenta de gerenciamento de projetos *Trac* através dos *plug-ins HudsonTrac*[15] e *Hudson Edgewall Trac*[16]. A Figura 20 mostra uma visão da ferramenta *Hudson* sendo executada.

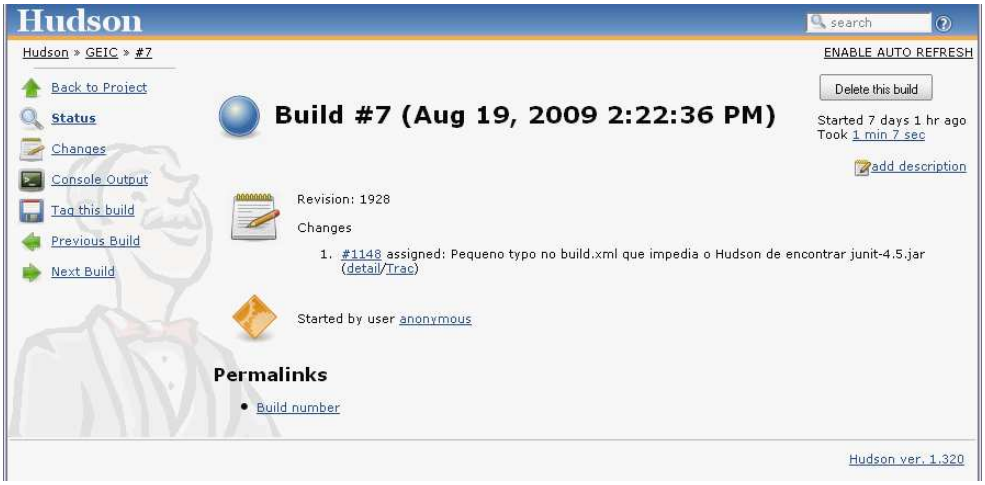


Figura 20 - Exemplo de construção automatizada através do Hudson

5.2.3.3 TESTES

[13] https://addons.mozilla.org/en-US/firefox/addon/7522

[14] http://www.mozilla.com

[15] http://trac-hacks.org/wiki/HudsonTracPlugin

[16] http://hudson.gotdns.com/wiki/display/HUDSON/Trac+Plugin

A única finalidade do servidor de testes é servir como uma cópia do servidor de aplicações, porém voltadas para testes de sistema. O sistema desse servidor é atualizado automaticamente pelo servidor de integração, e a partir dele programadores e especialistas de domínio realizam sessões de testes, depuração de componentes de *software* etc.

A grande vantagem de se usar um servidor de testes com as mesmas configurações e conteúdo do servidor de aplicações é reduzir ainda mais a chance de que ocorram problemas de integração inesperados. Evita também a ocorrência de defeitos que permanecem "dormentes", ou seja, que não se manifestam nos computadores dos programadores e que surgem apenas no ambiente de produção.

5.2.3.4 GERENCIAMENTO DE TAREFAS

Tarefas (*tickets*) são atividades, defeitos ou melhorias a serem realizados em um sistema. Como a quantidade de tarefas durante o decorrer de todo um projeto costuma ser bastante elevada, é comum o uso de ferramentas que auxiliem nesse processo. Além de facilitar a organização, uma ferramenta para o gerenciamento de tarefas auxilia na construção de uma base de conhecimento contendo informações sobre problemas, resoluções etc. Os grandes benefícios obtidos a partir desse tipo de ferramenta são maior produtividade, aumento da satisfação do cliente e melhora na comunicação entre programadores, engenheiros e gerentes.

Há ferramentas que auxiliam no processo de gerenciamento de tarefas, principalmente no que se refere à visualização da estrutura dos projetos e dos seus prazos. Outras funcionalidades dessas ferramentas costumam incluir calendários, diagramas de Gantt, gerenciamento de erros, gerenciamento de tempo gasto em cada tarefa etc. Ferramentas populares voltadas para esse tipo de atividade incluem *Trac*[17], *Redmine*[18] e *JIRA*[19].

Trac se mostrou a opção ideal para gerenciar a elaboração da infra-estrutura proposta, pois é um *software* livre (licença BSD modificada), possui uma filosofia minimalis-

[17] http://trac.edgewall.org

[18] http://www.redmine.org

[19] http://www.atlassian.com/software/jira

ta para gerenciamento de projetos via *Web*. Além disso, oferece recursos para planejamento (*roadmaps*), acompanhamento (*timeline*), escrita colaborativa (*wiki*) e integração com outras atividades de gerenciamento de projeto.

A partir dessa ferramenta são cadastrados e acompanhados todos os *tickets* do sistema, feitos os controles de *milestones* e são também feitas as integrações com *Subversion* e *Hudson*. A Figura 21 mostra uma visão da ferramenta *Trac* listando os *tickets* da um determinado *milestone*.

Figura 21 - Ferramenta *Trac* exibindo *tickets* de um *milestone*

5.2.3.5 BACKUP

Assim como no ambiente de produção, o objetivo do servidor de *backup* é prover maior segurança às informações armazenadas. Porém, aqui essas informações são o código-fonte e os itens de gerenciamento de projeto desenvolvidos pelos engenheiros de *software* e programadores. As cópias dos dados são realizadas em intervalos regulares, tanto localmente quanto remotamente.

5.2.3.6 E-MAIL

O servidor de *e-mails* tem como finalidade facilitar a comunicação entre engenheiros de *software* e programadores, tutores e especialistas de domínio. É utilizado também de maneira automatizada pelas ferramentas *Trac* e *Hudson* para notificar situações anormais, novidades e outros eventos, o que faz com que os membros da equipe estejam sempre a par das mudanças no projeto.

O agente para transferência de *e-mails* (*Mail Transfer Agent* – MTA) escolhido foi o *Postfix*[20], pois além de ser *software* livre é o MTA padrão em várias distribuições *Linux*.

5.3 PROJETO

Projeto (*design*) é uma representação significativa em engenharia de algo que será construído. Pode ser rastreado até os requisitos do domínio e possui quatro grandes áreas de interesse: dados, arquitetura, interfaces e componentes (PRESSMAN, 2001).

Engenheiros de *software* projetam sistemas computadorizados, mas as habilidades exigidas em cada nível de projeto são diferentes. Nos níveis de dados e arquitetura, o projeto é focado em padrões conforme eles se aplicam ao sistema a ser construído. No nível de interface, ergonomia geralmente dita o método de projeto. Já no nível de componente, um "método de programação" leva ao projeto efetivo de dados e procedimentos.

No escopo deste trabalho, as decisões a respeito da arquitetura da solução já foram apresentadas no capítulo 5.2 . O restante deste capítulo contém as decisões de projeto em relação aos padrões de projeto adotados, modelo de classes, módulos de *software*, estrutura do repositório, diretivas para criação de interfaces de usuário, e cenários de uso do sistema.

5.3.1 PADRÕES DE PROJETO

Cada padrão descreve um problema que ocorre repetidamente em nosso ambiente, e então descreve o núcleo da solução para aquele problema de maneira que a solução

[20] http://www.postfix.org

pode ser reutilizada várias vezes (ALEXANDER, 1977). Apesar de essa definição ter sido cunhada originalmente em relação a construções e edifícios, o mesmo pode ser dito a respeito de padrões de projeto (*design patterns*), que são expressos em termos de objetos e interfaces. No núcleo de ambos o tipos de padrão está uma solução para um problema em um contexto (GAMMA et al., 1994).

Em geral, um padrão possui quatro elementos: um nome (descrição sucinta em uma ou duas palavras), um problema a ser resolvido (quando aplicar um padrão), uma solução (elementos, relações, responsabilidade e colaborações) e conseqüências (resultados e compromissos de se aplicar o padrão). Assim, padrões de projeto de *software* são descrições de objetos comunicativos e classes que são customizadas para resolver um problema geral de projeto em um contexto particular.

Padrões de projeto residem no domínio de módulos e interconexões. Em um nível mais alto, há os padrões arquiteturais (descritos no capítulo 5.2.1) que são maiores em escopo, geralmente descrevendo um padrão geral seguido por um sistema inteiro (MARTIN, 2000).

No escopo deste trabalho foram adotados muitos padrões de projeto, os quais foram fundamentais para se obter um ambiente de produção padronizado, eficiente e de fácil entendimento e manutenção. Dentre eles, destacam-se o *Facade* e o *Data Transfer Object*.

5.3.1.1 FACADE

Estruturar um sistema em subsistemas ajuda a reduzir sua complexidade. Um objetivo comum em projetos é minimizar a comunicação e as dependências entre subsistemas, e uma maneira de se obter isso é através de uma fachada. O padrão de projeto fachada (*facade*) se refere a um objeto que provê uma interface única e simplificada para as partes de um subsistema (GAMMA et al., 1994), ou seja, define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

O uso de uma fachada também é aconselhável quando se quer ocultar do cliente a evolução de um ou mais subsistemas, provendo uma visão simples que é boa o suficiente para a maioria dos clientes. Outra aplicabilidade é quando se deseja definir apenas um ponto

de acesso a um subsistema, seja para fins de segurança ou de controle. Um exemplo de fachada pode ser visto na Figura 22.

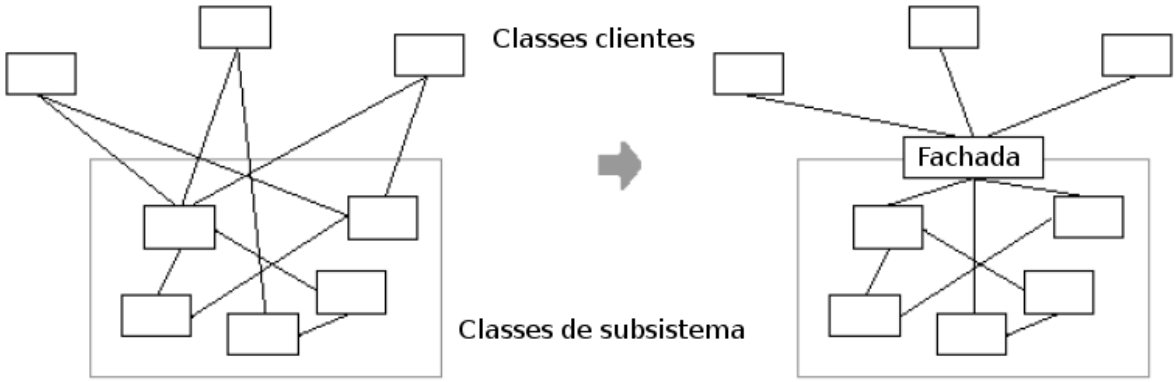


Figura 22 - Exemplo de fachada (adaptado de (GAMMA et al., 1994))

No contexto do sistema apresentado neste trabalho, o padrão fachada é utilizado principalmente no servidor de aplicação. Na subcamada de lógica de negócios, cada unidade de negócio é uma fachada que encapsula a complexidade das interações entre outras unidades de negócio e entidades. Esse tipo de fachada é também conhecido como “fachada de sessão” (*session facade*) (ALUR; CRUPI; MALKS, 2003).

5.3.1.2 REMOTE FACADE

Em um modelo orientado a objetos é aconselhável se trabalhar com pequenos objetos que possuam pequenos métodos, o que confere oportunidades para um maior controle e substituição de comportamento. Porém, quando estão envolvidas chamadas de métodos entre diferentes processos ou computadores isso se torna um problema, pois chamadas remotas são muito mais custosas: dados precisam ser serializados, segurança precisa ser conferida etc.

Como resultado, um objeto que tem como objetivo ser usado remotamente precisa de uma interface de alta granularidade que minimize o número de chamadas necessárias para se realizar uma tarefa. Essa mudança tem impacto em toda a estrutura de objetos, e programar se torna mais difícil e há diminuição de produtividade.

Uma fachada remota é uma fachada de alta granularidade em uma rede *Web* de objetos de baixa granularidade. Nenhum desses objetos possui uma interface remota, e a fa-

chada remota não contém lógica de domínio. Tudo o que a fachada remota faz é traduzir método de alta-granularidade em objetos de baixa-granularidade que se encontram em um subsistema (Fowler 2002). O principal objetivo deste padrão é melhorar a eficiência do uso da rede.

Neste trabalho, fachadas remotas são utilizadas entre as aplicações cliente e o servidor de aplicações, e se manifestam na forma de Serviços *Web*, que por sua vez se localizam na subcamada *Web* do servidor de aplicação. Dessa maneira, cada serviço é uma fachada remota para a subcamada de lógica de negócios, onde várias unidades de negócio podem ser envolvidas.

5.3.1.3 DATA TRANSFER OBJECT

Aplicações cliente geralmente requerem mais de um valor de um servidor de aplicação. O método *ad hoc* de se fazer isso é através de várias chamadas remotas, cada uma retornando um valor de interesse. Essa solução é muito ineficiente, pois além da penalidade paga por cada chamada de método, paga-se também pela latência de rede e pelo empacotamento e desempacotamento dos dados em cada camada.

Para se reduzir o número de chamadas remotas e para se evitar a penalidade associada, é aconselhado o uso de objetos de transferência de dados (*Data Transfer Objects* – DTOs, ou apenas TOs). Dessa maneira, uma única chamada de método é usada para enviar e receber o TO (ALUR; CRUPI; MALKS, 2003).

Um diagrama de classes envolvendo a criação e entrega de um TO encontra-se na Figura 23. Nela, um cliente faz uma requisição remota a uma unidade de negócio, que por sua vez cria um TO e o entrega ao cliente. O envio de dados do cliente para o servidor de aplicações também é feito na forma de TOs.

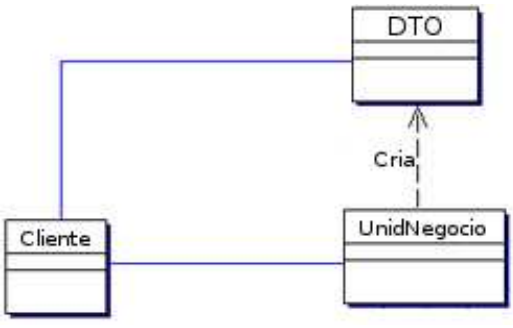


Figura 23 - Diagrama de classes envolvendo um DTO

Quando se está trabalhando com interfaces remotas, tais como uma fachada remota, cada chamada de método é muito custosa, e como resultado é necessário reduzir ao máximo o número de chamada, o que implica em se transferir mais dados em cada chamada. Uma maneira de se fazer isso seria retornando vários parâmetros, porém isso é impossível em linguagens como *Java*, que retornam apenas um valor por vez (FOWLER, 2002).

No contexto deste trabalho, a unidade de negócio ilustrada na Figura 23 encontra-se atrás de uma fachada remota (Serviços *Web*). Além disso, é permitido na rede o tráfego de tipos nativos de dados, tais como inteiros, *strings*, listas etc., que podem transitar sem o suporte de um TO. Para outros tipos complexos (por exemplo, tipo do domínio tais como *Aluno* e *Tutor*) o uso de TOs é aconselhado.

5.3.2 MODELO DE CLASSES

Um modelo de classes define a estrutura e o significado das classes (West 1999). Modelos de dados formalmente definem elementos de dados e a relação desses elementos para um domínio de interesse. Neste trabalho a preocupação principal é prover um modelo de classes que permita o reúso de dados por diferentes aplicações, tanto a partir da integração e compartilhamento em um mesmo banco de dados quanto a troca de dados por meios como transferência de arquivos. Modelos de dados de alta qualidade são coerentes com os requisitos do domínio, claros e não-ambíguos para todos os envolvidos, estáveis frente a mudanças nos requisitos, flexíveis frente a mudanças na lógica de negócios e reusáveis.

No contexto deste trabalho, as entidades do domínio são mapeadas em um modelo de classes conhecido como anêmico (FOWLER, 2002). Em um modelo de domínio anêmico a lógica de negócios é codificada fora dos objetos do domínio (entidades) em unidades de negócio (assim como ilustrado no detalhamento do servidor de aplicações na Figura 13). Com isso, essas classes externas é que programam a lógica do domínio, que por sua vez apenas transformam o estado das entidades.

As principais vantagens de se utilizar um modelo de domínio anêmico é a facilidade de se interagir com *frameworks* para persistência de dados (mapeamento objeto-relacional), além de facilitar em muito a geração de TOs a partir das entidades. A principal

desvantagem é a separação dos dados das entidades de seus comportamentos, fazendo com que elas passem a ser meros repositórios. O uso de modelos de domínio é encorajado em algumas plataformas de desenvolvimento, tais como *Java EE* (*Enterprise Edition*)[21].

A representação do modelo de classes pode ser feita através do diagrama de classes do UML. Porém, um diagrama completo representando todas as classes do domínio está fora do escopo deste trabalho, logo serão abordados apenas os cenários mais importantes para o entendimento da infra-estrutura, que são as unidades de ensino (PEIs, tentativas etc.), as pessoas (tutores, alunos etc.) e os registros de execução de sessão.

5.3.2.1 UNIDADES DE ENSINO

Unidades de ensino são representadas no domínio do problema como sendo aquelas responsáveis por ensinar ou avaliar os alunos. Elas foram organizadas da seguinte maneira:

- **Tentativas**: compostas basicamente por um estímulo de modelo, estímulos de comparação e mensagens de áudio de instrução, correção e conseqüência. Durante uma sessão de ensino, tentativas são as únicas coisas visíveis aos alunos. Frente a uma tentativa, um aluno deve emitir uma resposta;
- **Bloco**: conjunto de ocorrências (instâncias) de tentativas, sendo que uma tentativa pode ocorrer mais de uma vez dentro de um mesmo bloco. Além de ocorrências de tentativas, possui também as transições entre elas, que por sua vez podem ser efetuadas apenas mediante a satisfação de um critério;
- **Passo**: de maneira semelhante ao que acontece em um bloco, um passo é um conjunto de ocorrências de blocos, assim como as transições entre eles. Durante uma sessão de ensino, costuma-se executar um passo por vez;
- **Programa**: novamente de maneira semelhante a um bloco, um programa é um conjunto de ocorrências de passos. O PEI é a maior unidade de ensino no domínio, e é a única que pode ser associada a um aluno, assim alunos precisam estar

[21] http://java.sun.com/javaee

associados a PEIs para poderem executar tentativas. Um programa pode possuir três estados (situação): em edição, disponibilizado para os alunos e bloqueado.

Essa hierarquia de unidades de ensino é ilustrada na Figura 24, onde um PEI possui duas ocorrências de passos (Passo1 e Passo2), e cada um deles possui uma quantidade de passos, blocos etc. Para permitir um melhor agrupamento das tentativas dentro de um PEI, já que essas podem chegar às centenas, uma estrutura baseada no conceito de máquinas de estado finitas (MEF) foi adotada como base. Nessa estrutura, as unidades de ensino (estados de uma MEF) são interligadas por transições (transições de uma MEF) que são realizadas mediante a satisfação de um ou mais critérios de maneira determinística (MEF-det). Cada bloco, passo e programa possuem um estado inicial, e todos os estados são considerados finais, ou seja, o término de uma sessão é considerado válido independente da unidade de ensino executada por último.

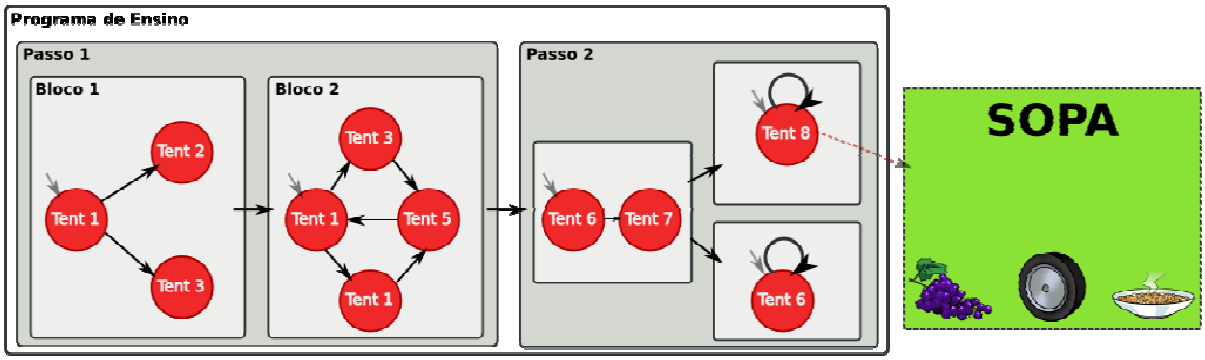


Figura 24 - Representação estrutural de um PEI

Uma definição mais formal do modelo de MEF-det utilizado para representar as unidades de ensino é um quíntupla (Σ,S,s_0,δ,F) onde:

- Σ é o alfabeto de entrada (respostas dos alunos);
- S é o conjunto finito e não-vazio de estados (ocorrências de entidades) dentro de um bloco, passo ou programa;
- s_0 é o estado inicial contido em S;
- δ é a função de estado-transição: $\delta : S \times \Sigma \rightarrow S$;
- F é o conjunto de estados finais (que no modelo do domínio é o próprio conjunto S).

O diagrama de classes referente às unidades de ensino encontra-se esquematizado na Figura 25. Nele, observa-se que *Tentativa* e *Estímulo* são classes abstratas, já que um sistema baseado nesta proposta pode definir quais tipos de tentativas e estímulos são mais adequados ao seu domínio. No estudo de caso do capítulo 6 foram codificados três tipos de tentativas e três tipos de estímulos (figura, som e texto).

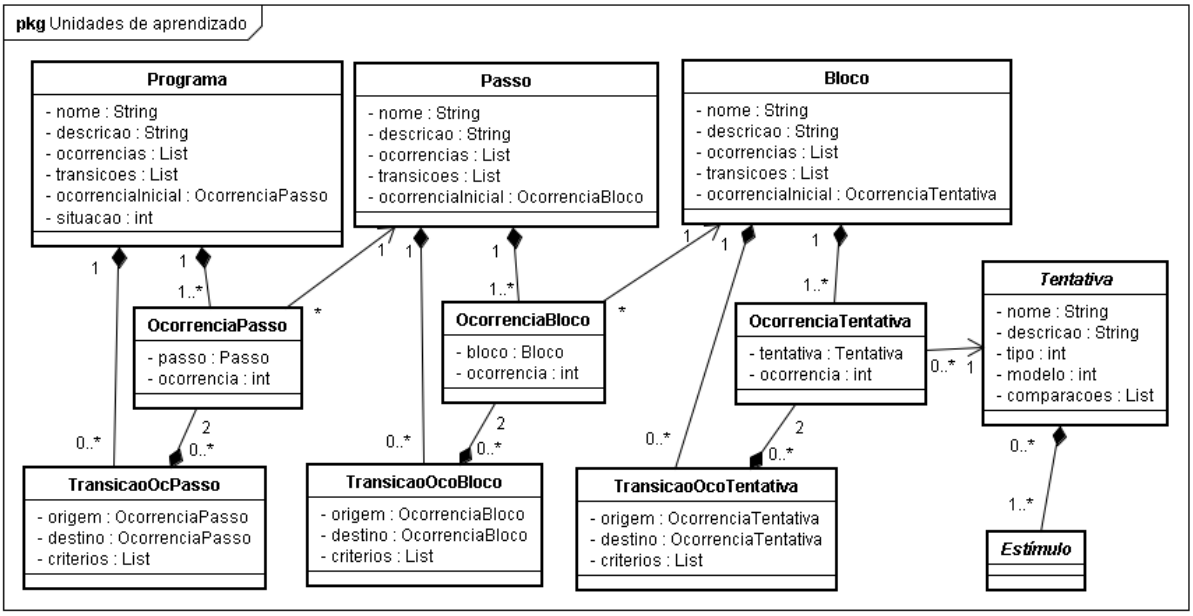


Figura 25 - Modelo de classes das unidades de ensino

5.3.2.2 PESSOAS

As pessoas do ambiente de produção foram organizadas neste projeto assim como ilustrado na Figura 26:

- **Pessoa**: a classe abstrata *Pessoa* é a base para tipos concretos como *Aluno* e *MembroEquipe*, e possui atributos importantes, tais como *nomeUsuario*, *nomeCompleto*, *senha* e *permissoes*. As permissões definem quais módulos cada pessoa pode executar, além de outras ações mais detalhadas, tais como remover alunos etc.;
- **Aluno**: especialização de *Pessoa*, possui atributos adicionais, tais como a escola em que estuda. Sua única responsabilidade é executar PEIs;
- **Membro de equipe**: também especialização de Pessoa, possui atributos adicionais, tais como *e-mail* e listas de alunos gerenciados e monitorados. Alunos ge-

renciados são aqueles que podem ser editados ou removidos, enquanto alunos monitorados são aqueles que podem ser monitorados durante uma sessão de ensino. Cada membro de equipe possui uma referência ao membro de equipe que o cadastrou.

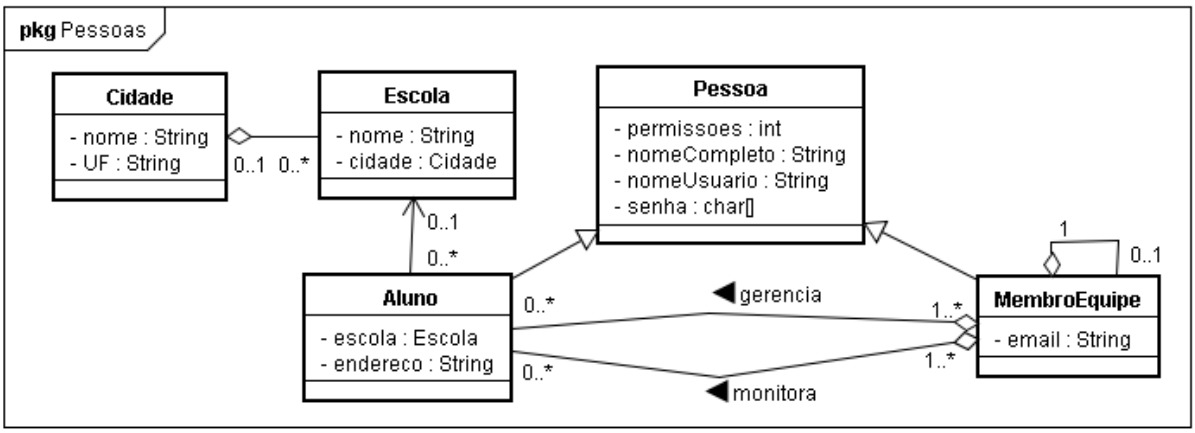


Figura 26 - Diagrama de classes das pessoas

Optou-se por não diferenciar no modelo de classes especialistas de domínio e tutores, já que o a diferença entre eles são apenas as permissões que possuem. Por exemplo, um membro de equipe com permissões para monitorar um aluno pode ser considerado um tutor, enquanto um membro de equipe com permissões para criar PEIs pode ser considerado um especialistas de domínio. Assim, tanto tutores quanto especialistas de domínio são representados pela classe *MembroEquipe*.

Caso possua as permissões necessárias, um membro de equipe pode criar outros membros de equipe a fim de repassar tarefas, delegar responsabilidades etc. Porém, o membro de equipe criador pode repassar aos seus subordinados apenas as permissões que ele já possui, garantindo que os subordinados tenham sempre poderes iguais ou menores que os seus. Adicionalmente, um membro de equipe tem controle sobre todos os novos membros de equipe e alunos que seus subordinados venham a cadastrar, gerando assim uma hierarquia completa de pessoas.

No topo da hierarquia de pessoas encontra-se o *super usuário*, que é o primeiro a ser cadastrado no sistema e o único que não possui um membro de equipe superior. Além de possuir controle sobre todos os alunos e membros de equipe, pode executar algumas ações exclusivas, tais como disponibilizar PEIs para execução, postar notícias no *site* etc. Uma ilustração da hierarquia de pessoas se encontra na Figura 27. As linhas pontilhadas que ligam o

super usuário aos grupos de alunos significam que, apesar do primeiro não ter criado os últimos, ele possui controle sobre eles de maneira indireta devido à sua maior posição na hierarquia.

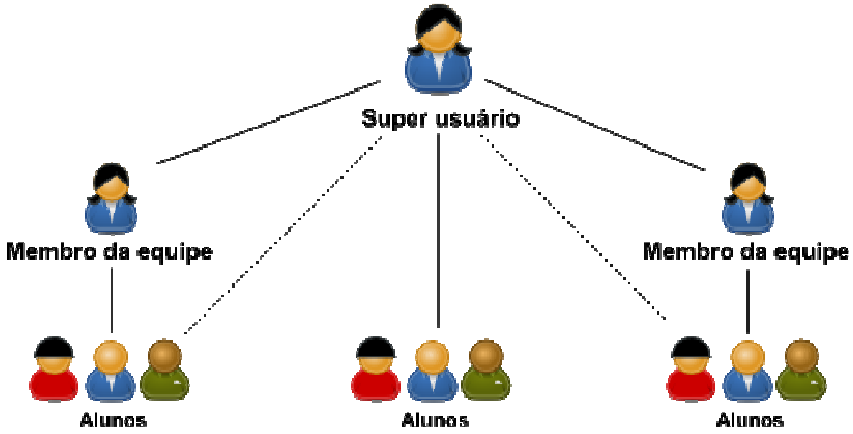


Figura 27 - Hierarquia de pessoas (subordinação)

5.3.2.3 REGISTRO DE EXECUÇÃO DE SESSÃO

O registro da execução de uma sessão por parte de um aluno é talvez o conjunto de informações mais valioso de toda a infra-estrutura. A partir desses dados é possível avaliar o desempenho de um aluno frente às tentativas, assim como avaliar o próprio PEI quanto ao seu funcionamento e eficácia.

As entidades envolvidas no registro de execução de sessão encontram-se no diagrama de classes da Figura 28. São elas:

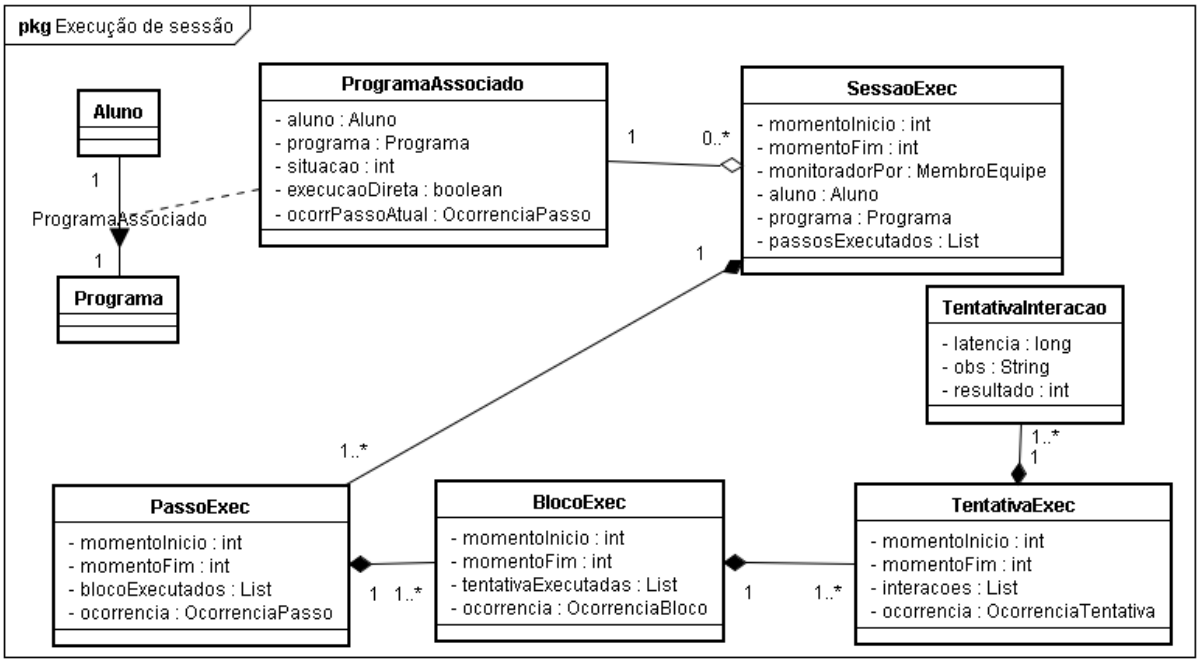


Figura 28 - Modelo de classes da execução da sessão

- ***ProgramaAssociado***: resulta da associação de um PEI a um aluno, e sem a qual uma sessão de ensino não pode ser iniciada. Outros atributos incluem a situação dessa associação (em execução ou finalizado), o passo em que o aluno se encontra dentro do PEI e se o aluno pode executar o PEI sem a presença de um tutor (execução direta);
- ***SessaoExec***: a entidade principal relacionada ao registro de execução de sessões, a execução de sessão é enviada ao servidor de aplicação logo após o término de uma sessão. Além de conter os momentos de seu início e término, contém o nome do monitor responsável (se aplicável), nome do aluno, o PEI associado e os passos que foram executados;
- ***PassoExec***: contém todas as informações coletadas durante a execução de um passo. Além de conter os momentos de início e fim do passo, contém a ocorrência de passo à qual se refere e uma lista de blocos executados. Normalmente um ou dois passos são executados por sessão;
- ***BlocoExec***: contém as informações coletadas durante a execução de um bloco de tentativas. Além de conter os momentos de início e fim do bloco, contém a ocorrência de bloco à qual se refere e uma lista de tentativas executadas;

- ***TentativaExec***: contém as informações coletadas durante a execução de uma única tentativa. Além de conter os momentos de início e fim, contém a ocorrência de tentativa à qual se refere e uma lista com o registro das interações do aluno;
- ***TentativaInteracao***: uma interação é cada uma das chances que o aluno teve de emitir uma resposta correta durante a execução de uma tentativa (uma execução de tentativa deve possuir pelo menos uma interação). Em uma interação estão registradas a latência da resposta, observações do tutor e o resultado da interação (correta, incorreta, cancelada etc.).

5.3.3 MÓDULOS DE *SOFTWARE*

A arquitetura do ambiente de produção favorece a construção de sistemas modulares, ou seja, facilita a construção de *software* composto de partes separadas, bem especificadas e com uma tarefa específica. No contexto deste trabalho, os módulos de *software* podem ser organizados em dois grupos: servidores e clientes, e que passarão a ser referenciados, respectivamente, como “módulos-servidor” e “módulos-cliente”.

5.3.3.1 MÓDULOS-SERVIDOR

Os módulos-servidor são aqueles que não são acessados diretamente pelos usuários, mas sim pelos módulos-cliente ou por outros módulos-servidor. Assim, suas funções geralmente consistem em realizar a ligação entre os usuários e a base de dados, assim como efetuar entre essas duas camadas o processamento relativo à lógica do domínio. Consistem também em realizar o intermédio (geralmente temporário) entre os usuários e o servidor de aplicações, com tarefas como adaptação de protocolos, troca de mensagens etc.

Cada módulo-servidor idealmente deve ser implantado em um computador (servidor) específico, o que garante uma melhor distribuição de carga entre todo o sistema e facilita a substituição de nós em caso de defeitos de *hardware* e comunicação. Em casos onde for conveniente, é possível o uso de recursos de malhas de processamento (*clustering*), visando distribuir a carga de processamento entre diferentes computadores, e espelhamento (*mirro-*

ring), visando manter redundância de dados e equipamentos para que, em caso de falha, um nó possa ser prontamente substituído.

Outro ponto comum entre os módulos-servidor é que todos eles possuem a estrutura baseada no padrão arquitetural multicamadas descrito no capítulo 5.2.1 , com separações claras entre as camadas *Web*, unidades de negócio e, em alguns casos, da subcamada de entidades. Os módulos-servidor da infra-estrutura proposta, cujos posicionamentos na infra-estrutura já foram discutidos no capítulo 5.2.2 , são:

- ***Servidor***: este módulo encontra-se implantado no servidor de produção, no núcleo do ambiente de produção. Sua estrutura obedece estritamente aquela descrita no padrão arquitetural multicamadas, com as camadas *Web*, lógica de negócio e entidades. Apresenta-se na forma de uma aplicação principal composta por dois submódulos: camada de lógica de negócios (unidades de negócio e as entidades) e camada *Web* (Serviços *Web*);
- ***Mensagens***: implantado no servidor de mensagens, sua estrutura difere do modelo multicamadas tradicional por não possuir uma subcamada de entidades, já que sua função é apenas repassar mensagens voláteis entre o módulo de execução de sessões e o módulo de tutoria. Para isso, sua camada de lógica de negócio contém filas de mensagens que permitem uma comunicação distribuída fracamente acoplada, confiável e assíncrona;
- ***Proxy TVDI***: realiza o intermédio entre o STB da TVDI e o servidor de produção assim como ilustrado na Figura 16 (arquitetura de comunicação na TVDI). Sua estrutura obedece parcialmente o modelo de *proxy* da infra-estrutura proposta já que a camada de adaptação de conteúdo não é necessária, visto que os televisores digitais possuem uma qualidade de som e vídeo suficientes;
- **Proxy Móvel**: realiza o intermédio entre dispositivos móveis básicos e o servidor de produção assim como ilustrado na Figura 17. Sua estrutura obedece estritamente o modelo de *proxy* da infra-estrutura proposta, sendo que o uso da camada de adaptação de conteúdo é especialmente importante. Ao requisitar uma sessão de ensino, o usuário deve informar sua preferência em relação à qualidade das mídias que receberá (dependendo da sofisticação de seu dispositivo), e o *proxy* deve levar em conta essa informação na hora de redimensionar as figuras e de reduzir a qualidade do som contidos na sessão.

5.3.3.2 MÓDULOS-CLIENTE

Os módulos-cliente, ao contrário dos módulos-servidor, são acessados diretamente pelos usuários a fim de que seja realizado um conjunto específico de tarefas, tais como gerenciar alunos, executar PEIs etc. Podem se apresentar na forma de aplicações tanto *Web* quanto *desktop*, opção que depende do tipo de funcionalidade que se deseja prover e dos requisitos de tempo, compatibilidade etc. O ponto de acesso a todos os módulos-cliente deve ser um módulo principal chamado *site* (descrito adiante), sendo que a lista de módulos disponíveis deve ser apresentada ao usuário mediante fornecimento de nome de usuário e senha.

Nenhuma informação é armazenada no disco rígido do computador de um usuário durante a execução de um módulo-cliente, sendo que todos os dados exibidos e editados devem ser obtidos diretamente do servidor de produção ou dos *proxies*. Esses dados são obtidos através de um ou mais Serviços *Web*, sendo que o mais comum é que cada módulo-cliente acesse um único Serviço *Web* de maneira direta. Em casos onde não é possível o acesso através de Serviços *Web*, devem ser utilizados os meios disponíveis na plataforma para se conectar a um *proxy* que, por sua vez, realizará a conexão com o servidor de produção.

Para se amenizar o impacto de ter que acessar Serviços *Web* constantemente, e para reduzir os custos de uso de redes de comunicação caras ou lentas, os módulos-cliente devem possuir mecanismos comuns para que o desempenho do sistema não seja prejudicado. Devem garantir também que durante certas atividades, tais como a execução de uma sessão de ensino, não haja interrupções devido a telas de carregamento, alta latência de rede etc. Os mecanismos adotados são a presença de *caches* locais, validação de parâmetros em tempo-real e acesso pontual às redes de comunicação.

Caches locais têm a finalidade de evitar acessos desnecessários à rede de comunicação, mantendo para isso uma lista de objetos recentemente requisitados em memória. Assim, é formada uma camada entre o módulo-cliente e um Serviço *Web* que só é atravessada caso um objeto requisitado não tenha sido encontrado localmente. Nesses casos onde um acesso à rede é inevitável, os objetos recém adquiridos devem ser adicionados ao *cache* seguindo uma política *Fist-In First-Out* (FIFO), favorecendo assim a reutilização de dados em intervalos de tempo relativamente curtos (localidade temporal).

Outra maneira de se fazer uso menos freqüente da rede de comunicação é através da validação em tempo-real de parâmetros de formulários. Isso impede que dados incon-

sistentes ou incompletos sejam enviados ao servidor de produção, avaliados, rejeitados e retornados junto a uma mensagem de erro. Um exemplo de validação em tempo-real de parâmetros de formulário ocorre em uma janela de cadastro de alunos, onde o botão para envio de dados é ativados apenas quanto todos os parâmetros obrigatórios tiverem sido preenchidos corretamente.

Finalmente, em módulos-cliente voltados para execução de PEIs, quando possível deve-se limitar o acesso à rede de comunicação a momentos muito pontuais, tais como o recebimento de uma sessão de ensino e envio do registro de execução. No meio tempo entre esses dois eventos, e uma vez que a sessão de ensino esteja no *cache* local, a rede de comunicação não deve ser acessada ou deve ser feita raramente. Essa técnica é válida apenas em situações onde a execução de uma sessão não esteja sendo acompanhada por um tutor remoto.

Além dos aspectos de desempenho que todo módulo-cliente deve realizar, a facilidade com que eles devem ser utilizados é indispensável, já que o público-alvo é tão diverso como alunos, professores, psicólogos, pedagogos e outros. Usabilidade é a facilidade com que o usuário pode utilizar uma ferramenta a fim de realizar uma tarefa específica, e para que isso aconteça é indispensável que, dentre outras coisas, todos os módulos-cliente possuam uma interface de usuário consistente e agradável.

Enquanto a aparência de uma aplicação é determinada por vários componentes individuais, alguma organização é necessária para se unificar a aparência e estrutura de conjuntos de ícones usados nesses componentes. Para isso, todos os módulos-cliente obedecem às recomendações do *Tango Desktop Project* (TANGO, 2009), que define um guia de estilo para cores e ícones que pode ser seguido por artistas e projetistas, além de oferecer uma biblioteca de ícones com representações para vários tipos de arquivos, ações, avisos etc. Pretende-se com esse projeto ajudar a criar uma experiência de interface gráfica de usuário consistente para *software* livre e de código-aberto.

Os módulos-cliente da infra-estrutura proposta são:

Site

O *site* é o ponto de acesso para todas as tarefas relacionadas ao gerenciamento de pessoas e PEIs. Após fornecer um nome de usuário e uma senha, o usuário é identificado como sendo ou um aluno ou um membro da equipe, e as próximas telas exibem as tarefas que podem ser iniciadas.

Além de ser o local a partir de onde todos os outros módulos-cliente são iniciados, o *site* funciona como um ponto central de acesso para todos os assuntos relacionados ao ambiente de produção, tais como notícias e documentação. Em adição, no *site* é disponibilizado gratuitamente o código-fonte de todo o sistema.

Alunos

O módulo de gerenciamento de alunos (*alunos)* é onde estão centralizadas as atividades relacionadas ao cadastro e gerenciamento de alunos, assim como das escolas e cidades em que eles se encontram. Além das funcionalidades básicas de cadastro, edição e remoção, pode-se utilizar este módulo para associar PEIs a alunos (um aluno pode estar associado a vários PEIs simultaneamente). Em casos muito raros, é possível também posicionar o aluno em um passo específico dentro de um PEI (essa atividade fica registrada na ficha do aluno).

A lista de alunos exibe apenas aqueles que podem ser gerenciados pelo membro de equipe corrente. Assim, evita-se que outros membros de equipe não subordinados tenham acesso a alunos que não lhe pertencem, ao mesmo tempo em que mantém a interface de usuário limpa e rápida.

Autoria

O módulo de autoria de PEIs (*autoria*) tem como finalidade gerenciar as unidades de ensino (programas, passos, blocos, tentativas e estímulos) do sistema. Também a partir deste módulo é que são disponibilizados os PEIs para que eles possam ser associados a alunos através do módulo *alunos*.

A principal funcionalidade deste módulo é a criação de tentativas. Contando com uma interface de usuário bastante intuitiva, é possível criar de maneira visual tentativas baseadas nos estímulos já cadastrados. Outras funcionalidades, tais como duplicação de tentativas, permitem elevar ainda mais a produtividade dos especialistas de domínio responsáveis pela criação dos objetos de ensino. Devido ao fato de muitas tentativas serem apenas pequenas variações de uma tentativa base, basta realizar cópias dela e alterar os detalhes relevantes.

Uma vez criadas as tentativas, o especialista de domínio deve criar unidades de ensino mais complexas, composta por conjuntos de tentativas. Essas unidades, representadas por blocos, passos e programas, são criadas de maneira bastante amigável através de recursos como arrastar e soltar (*drag and drop*) e ampliar e reduzir (*zoom in* e *zoom out*).

Para evitar que um PEI, que geralmente é composto por centenas de tentativas, seja disponibilizado contendo erros, o módulo de autoria fornece recursos para que cada unidade de ensino (tentativas, blocos e passos) seja testada individualmente. Esses testes consistem em exibir as unidades assim como elas serão mostradas ao aluno no módulo de execução de sessão, porém com várias informações complementares, tais como destaque visual para resposta correta, teclas de atalho etc.

Consulta

O módulo de consulta de resultados (*consulta*) é utilizado para conferir as medidas coletadas durante a execução pelos alunos dos PEIs. Após fornecer nome de usuário e senha, o membro de equipe visualiza uma lista com apenas os seus alunos que já executaram sessões. Após isso, podem ser consultados os PEIs que cada um deles executou, os passos contidos em cada PEI, e assim sucessivamente até chegar a cada tentativa específica.

Além de fornecer na tela os resultados das execuções, há também a possibilidade de se exportar esses resultados de várias maneiras. A exportação em formato tabular gera um arquivo CSV (*comma-separated values*), acessível por *software* como *Microsoft Excel* e *OpenOffice Spreadsheet*, que é mais adequada para fins de tratamento estatístico e utilização de macros. A exportação estatística também gera um arquivo em formato CSV, porém com algumas informações estatísticas (médias, medianas, mínimos, máximos etc.) já calculadas. Finalmente, a exportação de documento gera um arquivo contendo a sessão descrita em um formato mais legível e amigável, próprio para a geração de relatórios.

Equipe

O módulo de gerenciamento de membros da equipe (*equipe*) é onde se controla quem são os membros participantes do projeto e quais as suas permissões e responsabilidades. É através do *equipe* que a hierarquia de pessoas ilustrada na Figura 27 pode ser concretizada.

Através dele membros de equipe podem ser criados, editados e removidos. Depois de criados, podem ser modificadas suas permissões, que são do tipo “permitir acesso ao módulo de autoria”, “permitir criação de alunos” etc. Além disso, neste módulo é definido quais alunos o membro de equipe selecionado pode gerenciar (editar ou remover) ou monitorar (acompanhar durante a execução de uma sessão de ensino).

Player

O módulo de execução de PEIs (*player*) permite aos alunos executarem os programas que foram associados a eles pelos seus tutores. Do ponto de vista de um aluno, o *player* exibe apenas tentativas, sem distinção de bloco e passo, dando a ilusão de que o PEI é uma série linear de tentativas.

Um aluno, após fornecer seu nome de usuário e sua senha, tem acesso à lista de PEIs que foram associados a ele. Ao selecionar um desses PEIs, é exibido um breve resumo com o nome do programa e o passo atual, e em seguida a sessão de ensino é iniciada.

Um tutor também pode acessar o *player* através de seu nome de usuário e sua senha. Após suas permissões serem confirmadas, é exibida uma lista de alunos prontos para serem monitorados. Ao se selecionar um aluno, é exibida a lista de programas associados, e o restante ocorre como no caso onde o aluno inicia a sessão. Porém, quando uma sessão é iniciada por um tutor alguns recursos adicionais se tornam disponíveis, tais como o uso de teclas de atalho (forçar acerto, forçar erro etc.) e a possibilidade de se fazer anotações pelo teclado durante a sessão (essas anotações são exibidas no módulo *consulta*).

Player-Móvel

O módulo de execução de PEIs através de dispositivos móveis (*player-móvel*) permite aos alunos executarem PEIs através de seus telefones celulares, *palmtops* etc., exatamente como ocorre com o *player* tradicional. Após fornecer seu nome de usuário e sua senha, o aluno é requisitado a autorizar a conexão de seu aparelho com a Internet, e após isso a execução da sessão ocorre de maneira tradicional.

A principal diferença do *player-móvel* em relação ao *player* tradicional é que ele pode apenas ser acessado por alunos (o modo tradicional permite o acesso de alunos e tutores), e não há opções de monitoria remota. Dessa maneira, o único modo de execução de sessão é o direto (sem presença de tutor), restrição necessária devido às limitações de *hardware* e de rede dos dispositivos móveis.

Player-TVDI

O módulo de execução de PEIs através de sistemas de TVDI (*playertvdi*) permite aos alunos executarem PEIs através de suas televisões digitais, contanto que elas possuem um STB com canal de retorno suficiente para o recebimento de sessões de ensino.

A principal diferença deste módulo em relação ao *player* tradicional é que deve ser usado o controle remoto para realizar a seleção e confirmação de respostas.

Tutor

Um dos módulos mais importantes é o *tutor*, cujo objetivo é permitir o monitoramento remoto de alunos, contando para isso com recursos de comunicação síncrona de som e imagem, execução remota de comandos, registro de comentários etc. Esse módulo, que é acessado exclusivamente por tutores, se comunica com o módulo *player* sendo executado pelos alunos, utilizando para isso tanto o servidor de produção quanto o servidor de troca de mensagens.

De acordo com o conteúdo a ser ensinado e do perfil dos alunos, pode-se optar por várias formas de monitoramento que variam em função da quantidade de alunos e tutores simultâneos, forma de avaliação e tipo de comprometimento à sessão. Esses vários modos de monitoria permitirão que o tempo ocioso de tutores quando esses estiverem disponíveis para monitoramento seja praticamente eliminado, além de permitir a mais alunos serem atendidos no mesmo período.

Em relação à quantidade de alunos e tutores participando simultaneamente de uma sessão de ensino, pode-se optar pelos seguintes modos:

- **1:1** → Um aluno sendo monitorado por um tutor, ideal em sessões de ensino em que o aluno requer dedicação exclusiva. Situação exemplo: ensino de leitura e escrita a crianças com retardo mental sendo realizado por um tutor da área de psicologia;
- **1:N** → Um aluno sendo monitorado por um ou mais tutores, com a avaliação geral sendo gerado automaticamente a partir da avaliação de cada tutor. Esse modo é ideal para assuntos complexos que requerem a confirmação de uma banca de tutores, ou de um tutor supervisor sendo acompanhado por um conjunto de tutores menos experientes. Esse modo é útil também para evitar que um grupo de tutores permaneça ocioso por muito tempo a espera de um aluno para monitorar. Situação exemplo: ensino de uma habilidade acadêmica complexa avaliado por uma banca de tutores, ou situações que envolvem um tutor principal supervisionando um grupo de tutores menos experientes;
- **N:1** → Vários alunos sendo monitorados por um único tutor, ideal em situações em que o tutor realiza pouca ou nenhuma intervenção e apenas supervisiona a e-

xecução de tentativas de um grupo de alunos específico. Esse modo é útil também quando se deseja maximizar o aproveitamento do tempo disponível do tutor. Situação exemplo: ensino de uma habilidade onde o tutor deve intervir apenas após um comportamento específico de um dos alunos sob sua tutela;

- **N:N** → Vários alunos sendo monitorados por vários tutores, ideal em situações em que os tutores apenas supervisionam um grupo de alunos, o que proporciona um aproveitamento ótimo dos tempos disponíveis de alunos e tutores. Situação exemplo: ensino de uma habilidade simples onde há abundância de alunos e tutores, sendo que os últimos devem apenas supervisionar a sessão de ensino interagindo apenas quando indispensável.

Em relação ao momento em que a avaliação da resposta de um aluno é feita pelo tutor, pode-se optar por duas formas:

- **Síncrona** (s): realizada logo após a resposta, proporcionando um retorno imediato ao aluno. Aconselhada para situações onde o retorno é importante para o andamento da sessão, essa forma de avaliação pode ser aplicada em todos os tipos de PEIs;
- **Assíncrona** (a): realizada posteriormente à sessão de ensino. Aconselhado para situações onde o retorno imediato ao aluno não é desejável ou relevante, ou quando a quantidade de alunos excede em muito a quantidade de tutores disponíveis. Indicado também quando o canal de comunicação não o permite, assim como ocorre com canais de retorno insuficientes em sistemas de TVDI. Avaliações assíncronas são permitidas apenas em PEIs lineares, onde as transições entre unidades de ensino dentro de uma única sessão são independentes do desempenho do aluno.

Finalmente, em relação à dedicação do tutor em relação à cada sessão, há duas formas possíveis:

- **Exclusiva** (e): o tutor permanece comprometido a uma sessão de ensino por vez, não podendo participar de outra monitoria até que a sessão de ensino em que se encontra tiver terminado. Implica o acompanhamento tentativa por tentativa, independente do ritmo do aluno.

- **Inclusiva** (i): o tutor não permanece dedicado a uma única sessão de ensino por vez, podendo monitorar tentativas de vários alunos e de vários PEIs simultaneamente ou de maneira seqüencial.

Durante o decorrer deste trabalho, referências aos modos de monitoria serão feitos na forma: *(quantidade de alunos e tutores) (momento de avaliação) (dedicação)*. Onde um dos modos não precisar ser especificado, serão usadas as incógnitas *x*, *y*, *z* e *w* conforme necessário. Por exemplo, o modo de monitoria onde um aluno é monitorado por um tutor de maneira síncrona e exclusiva será referenciado simplesmente como *1:1se*. Já o modo de monitoramento onde um aluno é monitorado por vários tutores, independente do momento de avaliação e da dedicação, será referenciado como *1:nxy*.

As várias combinações possíveis de monitoria oferecem uma enorme flexibilidade aos tutores e alunos, porém algumas dessas combinações não são viáveis de um ponto de vista prático. Um exemplo de situação não prática é *N:1xe*, já que um tutor não pode monitorar mais de um aluno de maneira exclusiva.

Vale lembrar que o *tutor* é útil apenas quando se deseja acompanhar alunos de maneira mais próxima, já que o *player* já oferece o *feedback* imediato exigido pelo domínio através de mensagens pré-definidas.

5.4 PRÁTICAS E MÉTODOS

Além das atividades guarda-chuva (disciplinas de suporte) descritas no capítulo 5.2.3 tais como controle de revisões, integração contínua, gerenciamento de tarefas etc., outras práticas e métodos essenciais para o sucesso de um projeto de *software* foram. São elas:

5.4.1 TESTES

Testes de *software* são investigações empíricas conduzidas para prover a clientes informações sobre a qualidade do produto ou serviço sendo testado (KANER, 2006), e incluem o processo de validação e verificação de que o produto satisfaz os requisitos técnicos e

de negócio que guiaram seu projeto e desenvolvimento. Testes podem ser realizados nos seguintes níveis:

- **Testes funcionais**: visam à operação correta do produto no sentido de satisfazer o comportamento esperado na especificação de requisitos. Pode ser de vários tipos, inclusive:
  - **Testes unitários**: testam unidades mínimas de *software* (funções, classes, pacotes, componentes ou módulos) a fim de verificar se o projeto detalhado dessas unidades foi corretamente codificado. Realizados tanto pelos programadores quanto automaticamente durante o teste de integração. O conjunto de *frameworks* para execução de testes unitários mais popular é coletivamente conhecido como *XUnit*, e é composto por *frameworks* como *JUnit*[22] (para a plataforma *Java*), *CPPUnit*[23] (para a linguagem C++) etc.
  - **Teste de integração**: expõe defeitos nas interfaces e na interação entre os componentes integrados (módulos) do sistema, sendo geralmente efetuado após os testes unitários e antes do teste de sistema. Realizado tanto manualmente pelos engenheiros de *software* quanto automaticamente por um servidor de integração.
  - **Teste de sistema**: teste de um sistema completamente integrado a fim de verificar se os requisitos que guiaram seu desenvolvimento foram satisfeitos. Realizados tanto pelos programadores durante o processo de desenvolvimento quanto pelos especialistas de domínio e tutores.
- **Testes não-funcionais**: testam aspectos não-funcionais do sistema, tais como desempenho, estabilidade, usabilidade e segurança. Realizados tanto por engenheiros de *software* e programadores quanto por especialistas de domínio.

No âmbito deste trabalho, utilizou-se o *framework* de testes unitários *JUnit*, pois além de ser *software* livre e de possuir vasta documentação, é o padrão *de facto* para testes unitários na linguagem *Java*. Os testes de integração foram realizados tanto de forma manual pelos programadores quanto automatizados por um servidor de integração contínua. Testes de sistema foram realizados pelos programadores à medida que novas funcionalidades e-

[22] http://junit.org

[23] http://sourceforge.net/apps/mediawiki/cppunit

ram adicionadas, e também por especialistas de domínio e tutores logo antes da liberação de novas versões do sistema.

5.4.2 CONSTRUÇÃO AUTOMATIZADA

O processo de construção de um *software* (*build*) se refere ao processo de converter código-fonte em artefatos de *software* que possam ser executados em um computador. Etapas importantes da construção incluem o processo de compilação de código-fonte, ligação (*link*) de artefatos intermediários, execução de testes e implantação.

A construção comumente é gerenciada por uma ferramenta (*build tool*) que coordena a compilação e a ligação, na ordem correta, de todas as partes necessárias para se criar o produto, e opcionalmente no final realiza o processo de implantação desse produto em servidores ou *middlewares*. Ferramentas populares para construção de *software* incluem *Ant*[24], *Maven*[25] e *Make*[26].

No contexto deste projeto foi escolhido *Ant* como ferramenta de automação de construção, pois além de ser suportado nativamente em todos os IDEs *Java* facilita o uso de testes de integração com *JUnit*.

Além de ser utilizado para automatização de construção, scripts *Ant* devem ser utilizados muito para a automatização de várias outras etapas do processo de desenvolvimento, tais como:

- **Configurar banco de dados**: um conjunto de tarefas que costuma se repetir com grande freqüência é a configuração de SGBD para que esse possa ser utilizado pelo servidor de aplicação do sistema. Exemplos de tarefas desse conjunto incluem criar um banco de dados com as propriedades corretas, criar usuários com acesso a esse banco, definir permissões etc.;
- **Configurar servidor de aplicação**: depois de configurado o SGBD para poder receber os dados do ambiente de produção é necessário que o servidor de aplicação, que fará acesso ao SGBD através de uma camada de persistência, esteja ade-

[24] http://ant.apache.org

[25] http://maven.apache.org

[26] http://www.gnu.org/software/make

quadamente configurado. Desse modo ele poderá localizar exatamente quais classes e quais bancos de dados devem ser envolvidos no processo de persistência;

- **Povoar banco de dados**: outra tarefa bastante comum durante o processo de desenvolvimento é a recuperação de todo o conteúdo do banco de dados do ambiente de produção a fim de se realizar testes, verificações de rotina etc.
- **Criação de um novo módulo-cliente**: quando se deseja oferecer um novo conjunto de funcionalidade aos usuários uma série de atividades repetitivas deve ser executada, tais como a criação de um novo módulo-cliente, de um Serviço *Web* associado etc. Para eliminar o desperdício de tempo associado a essas tarefas um *script Ant*, cujos argumentos recebidos devem ser apenas o nome do novo módulo e o nome do Serviço *Web* relacionado;

5.4.3 ESTRUTURAÇÃO DO REPOSITÓRIO DE CÓDIGO-FONTE

Dados a complexidade do modelo de classes, a grande quantidade de módulos e a natureza distribuída da infra-estrutura apresentada, é natural que vários programadores estejam envolvidos simultaneamente e que um repositório de código-fonte é necessário (assim como discutido no capítulo 5.2.3).

Porém, apenas manter o código-fonte dentro do repositório sem nenhuma preocupação com sua estrutura não oferece vantagens para a produtividade dos programadores. Não são raras as situações onde se deseja manter simultaneamente duas versões do mesmo sistema, registrar a situação exata em que o sistema estava durante o lançamento de uma versão, ou manter uma versão do código-fonte apenas para experimentos.

Para resolver esses problemas, o repositório foi estruturado da seguinte maneira:

- ***trunk***: o “tronco” do repositório, onde está a versão mais recente do código-fonte, ou seja, a que está recebendo mais atualizações por parte dos programadores. O código-fonte que se encontra no *trunk* se tornará, no futuro, a próxima versão a ser entregue. Cada subpasta do *trunk* deve conter um módulo específico do sistema, como *autoria*, *consulta* etc. No momento em que o código-fonte do *trunk* é consi-

derado pronto para ser lançado e disponibilizado aos usuários, um *tag* deve ser criado, e o trabalho no *trunk* deve prosseguir normalmente;

- ***tags***: as "etiquetas", onde ficam versões do repositório que representam alguma coisa do mundo real. Por exemplo, no exato momento em que a versão *X.Y.Z* de um sistema é disponibilizada, deve-se criar um *tag* com o nome *X.Y.Z*. Com isso, caso se deseje uma cópia do *trunk* assim como ele estava no momento exato de um lançamento, basta obter o código-fonte do *tag* relacionado. O uso de *tags* também é útil para quando um defeito no código-fonte for descoberto, e a correção deve ser aplicada tanto no *trunk* quanto em versões que já foram lançadas;
- ***branches***: os "ramos" do repositório, onde ficam versões do código-fonte paralelas, de teste ou que ainda não estão prontas para serem eventualmente reintegradas ao *trunk*. O principal objetivo dos *branches* é não interferir com o *trunk* ao mesmo tempo em que permite aos programadores desenvolver código-fonte protegido pelo repositório. A pasta *branches* contém as subpastas *releases* e *users*:
    - ***releases***: são cópias de código-fonte que uma vez pertenceu ao *trunk*, mas que foram recebendo modificações e foram sendo liberadas independentemente. O uso de um *branch* de *releases* é muito importante quando um mesmo sistema está disponível no mercado em várias versões diferentes e que são mantidas em paralelo. Cada subpasta da pasta *releases* deve ser nomeada com o número de versão do *trunk* no momento da cópia. Assim como acontece no *trunk*, cada versão da pasta *releases* deve ter uma *tag* relacionada;
    - ***developers***: composta de código-fonte específico para cada programador, é normalmente usado para testes, experimentações, alterações de grande escala que não podem ficar no *trunk* etc. Ao contrário do *branch* de *releases*, as versões presentes nos *branches* dos programadores não costumam ser lançadas, mas sim eventualmente reincorporadas (*merge*) ao *trunk*. O *branch* dos programadores pode ser usado também para armazenar versões customizadas do *trunk*, que apesar de não serem lançadas como o sistema oficial ainda assim são de interesse de outro grupo de usuários ou programadores. A pasta *developers* contém subpastas com os nomes dos programadores, e todos são livres para colocar nessas pastas o que lhes for de interesse.

5.4.4 DOCUMENTAÇÃO

Documentação é o texto escrito que acompanha um sistema de *software*. Seu conteúdo contém explicações sobre como esse sistema é operado ou como pode ser usado, e deve levar em conta diferentes perfis de usuários. No contexto desta proposta um conjunto rico de documentos deve ser disponibilizado, de preferência on-line em mídias eletrônicas, a fim de auxiliar especialistas de domínio e programadores a se familiarizarem com a infra-estrutura, além de fornecer a membros de equipe já experientes uma referência para a execução de tarefas rotineiras. O conjunto mínimo de documentos, que compreende o necessário para que tanto usuários quanto equipe de desenvolvimento possam desempenhar seus papéis com facilidade, deve compreender:

- **Manual de instalação do ambiente de produção**: instruções sobre como preparar um computador para que esse possa executar todo o sistema. Inclui informação sobre instalação do sistema operacional, configurações de segurança, dependências de *software*, aplicações essenciais etc.;
- **Manual de instalação do ambiente de desenvolvimento**: instruções sobre como preparar um computador para que a partir dele possam ser desenvolvidos módulos para o ambiente de produção, assim para que possa participar do ambiente de desenvolvimento e de seu processo. Inclui informações sobre instalação de ambientes de desenvolvimento, bibliotecas de *software* etc.;
- **Manual de implantação do sistema**: instruções sobre como obter a versão mais recente do sistema a partir do repositório de código-fonte e implantá-la em um computador previamente configurado. Geralmente utilizados por engenheiros de *software* sempre que uma nova versão do sistema deve ser disponibilizada para os usuários;
- **Manual do programador**: instruções sobre como realizar tarefas relacionadas à criação de manutenção de módulos de *software*. Inclui informações sobre o processo de construção automatizada, padrões de projeto, convenções de codificação etc.;
- **Recomendações para a criação de interfaces de usuário**: recomendações gerais sobre como garantir uma interface de usuário unificada entre todos os módulos-

cliente. Possui informações sobre padrões de cores, conjuntos de ícones, comportamento de janelas, disposição de botões, dicas visuais para usuários etc.;

- **Manual do usuário**: instruções sobre como utilizar o sistema desenvolvido a partir dos pontos de vista de especialistas do domínio e alunos. Nele estão documentados todos os módulos-cliente e suas funcionalidades, sistema de permissões e hierarquia de pessoas, informações sobre contato com a equipe de desenvolvimento etc.;
- **Recomendações para a criação de estímulos**: voltado principalmente para especialistas de domínio que desejam adicionar novos estímulos (figura e som) ao sistema. Contém informações sobre qualidade mínima e máxima dos estímulos, dimensões, duração, estilo etc.

5.5 ABRANGÊNCIA DA SOLUÇÃO

Apesar de este trabalho ter sido criado com base em PEIs e no paradigma de equivalência de estímulos, tornou-se claro que toda a análise experimental do comportamento, além de outras teorias de aprendizado como as baseadas no construtivismo, podem se beneficiar com a flexibilidade da solução desenvolvida e a facilidade para se introduzir modificações que atendam a requisitos específicos. Outras áreas além da pesquisa em ensino podem ser atribuídas aos sistemas baseados na infra-estrutura proposta, tais como aplicação de avaliações onde nenhum *feedback* deve ser retornado a quem responde os testes.

De maneira geral pode-se afirmar que a solução apresentada possui grande abrangência e pode suportar qualquer forma de aprendizado que se baseie na resolução de problemas. Como no caso das tentativas utilizadas no paradigma de equivalência de estímulos, esses problemas devem ser discretos e não obrigatoriamente precisam apresentar situações em que o aluno deve, frente a um modelo e a um conjunto de alternativas, selecionar uma das opções desse conjunto. Um exemplo poderia ser o de um programa de ensino de algoritmos onde o estudante, frente a um enunciado (por exemplo, “elabore um algoritmo para ordenação de listas”), fornece um pseudocódigo verificável. Vale ressaltar que pode haver vários algoritmos válidos para um mesmo problema e que a conseqüência à resposta do aluno não precisa neces-

sariamente ser “certo” ou “errado”, podendo ser, por exemplo, a eficiência do pseudocódigo na forma de complexidade algorítmica.

Os problemas devem ser arranjados em conjuntos sucessivamente maiores e mais abstratos (a exemplo dos blocos e passos do paradigma de equivalência de estímulos). Além disso, todas essas unidades (da mais concreta até a mais abstrata) devem ser ligadas por fluxos (transições) condicionais e incondicionais que possibilitam a definição de um roteiro de estudos, permitindo assim aos especialistas de domínio controlar a ordem e a velocidade com que os programas de ensino serão aplicados.

Visualizando um programa de ensino como uma estrutura de dados em forma de árvore, pode-se pensar nos problemas como sendo as “folhas”. Assim, basta alterar o conteúdo dessas folhas (tentativas, testes de múltipla-escolha, avaliações subjetivas etc.) para se obter uma nova forma de ensino ou avaliação (PEIs, exames para obtenção de carteira de motorista, provas escolares etc.).

Em termos computacionais, para que cada “folha” possa ser generalizada para atender aos requisitos de vários domínios, finalidades e formas de ensino, basta aproveitar o modelo de classes especificado na Figura 25 e disponibilizar implementações concretas da classe abstrata *Tentativa*. Pode ser ainda necessário especificar novos critérios (específicos para certo domínio) para que as transições entre tentativas, blocos e passos ocorram. Um esquema da representação em árvore de problemas e transições pode ser visto na Figura 29.

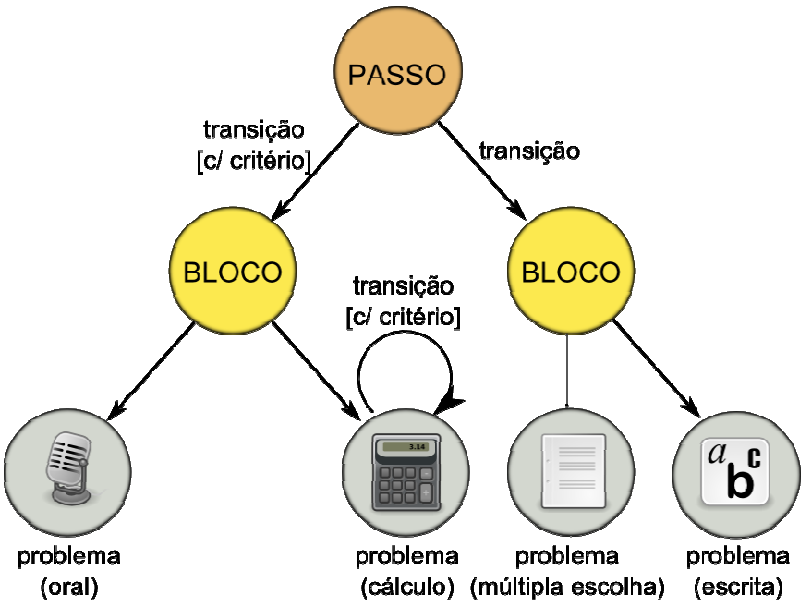


Figura 29 - Árvore de tentativas (problemas), transições e critérios

Além de mudanças no modelo de classes das unidades de ensino, o suporte a novos tipos de problemas exige pequenas adaptações nos módulos da infra-estrutura computacional especificados. Os módulos e um resumo das mudanças requeridas são:

- ***Autoria***: novas telas para a criação visual de tentativas (problemas).
- ***Player*** (e suas variantes *playertvdi* e *playermovel*): novas telas para a exibição de tentativas, assim como a lógica para o registro do comportamento do aluno nas novas tentativas;
- ***Tutor***: novas telas para a exibição das tentativas sendo executadas pelos alunos no módulo *player*;
- ***Consulta***: novas telas e gerações de estatísticas e relatórios;
- ***Servidor***: alteração da lógica de negócio e da camada de persistência.

Os outros módulos, tais como *equipe* e *alunos*, permanecem inalterados. A criação de novos módulos pode ser necessária especialmente em casos onde a avaliação da resposta do usuário é computacionalmente custosa ou não trivial. Nesse caso específico espera-se que módulos de avaliação tornem-se cada vez mais freqüentes uma vez que as respostas a problemas passarão a ser mais elaboradas (assim como as conseqüências a essas respostas). Outro cenário possível é a criação de módulos de avaliação que façam intermédio com serviços externos, tais como aqueles especializados na avaliação de um tipo específico de conteúdo.

6 ESTUDO DE CASO

O Laboratório de Estudos do Comportamento Humano (LECH), do Departamento de Psicologia (DPsi) da Universidade Federal de São Carlos (UFSCar), vem desenvolvendo e aplicando há vários anos programas de ensino baseados no paradigma de equivalência de estímulos (SOUZA, 2009).

Com o objetivo de informatizar a aplicação de PEs, foi desenvolvida pelo LECH uma ferramenta computacional chamada *Aprendendo a Ler e a Escrever em Pequenos Passos* (ALEPP, também conhecido como *ProgLeit*). Criada a partir da linguagem de programação *Object Pascal* (*Delphi* 4), é uma aplicação *stand-alone* (*desktop*) que realiza a única tarefa de apresentar PEs e registrar as interações dos alunos.

O ALEPP é um tanto limitado e não atendia à totalidade das necessidades envolvidas na aplicação dos PEIs. Além disso, precisa ser instalado em cada computador e obriga os alunos a se locomoverem até o centro de pesquisas, o que se torna inviável quando o objetivo não é mais a pesquisa e sim a disponibilização dos PEs para o maior número de alunos. Esses alunos que requerem ensino remediativo em sua maioria apresentam-se em maior desvantagem econômica, estudam em escolas das periferias urbanas ou moram em regiões isoladas ou de difícil acesso (BRANDÃO; BAETA; ROCHA, 1983).

Embora o ALEPP tenha sido amplamente empregado com um grande contingente de alunos e permitido o desenvolvimento de pesquisas que fundamentam os PEs (SOUZA et al., 2009), atende poucas das necessidades descritas nos capítulos anteriores. Logo, tornou-se necessária a criação de um sistema de *software* que atenda aos requisitos no domínio da análise experimental do comportamento.

O LECH participa do projeto *Tidia-Ae* (Tecnologia da Informação no Desenvolvimento da Internet Avançada - Aprendizado Eletrônico) - fase II como laboratório associado ao núcleo São Carlos – Intermídia. O *Tidia-Ae* tem como finalidade auxiliar as atividades de ensino eletrônico, oferecendo suporte ao ensino presencial (TIDIA-AE, 2009), com foco tanto na pesquisa quanto no desenvolvimento de ambientes colaborativos e ensino a distância, suportados por redes de alta velocidade (FAPESP, 2009).

O Laboratório de Inovação em Computação e em Engenharia (LINCE) do Departamento de Computação (DC) da UFSCar, participante do *Tidia-Ae* como laboratório de desenvolvimento associado, estabeleceu uma parceria com o LECH no âmbito desse projeto.

O objetivo dessa parceria é desenvolver um sistema de *software* que resolva o problema do ensino individualizado por computador e que permita também avaliar, de um ponto de vista prático, técnicas avançadas de engenharia de *software*, redes de computadores, computação distribuída e novas plataformas de *hardware*.

O sistema desenvolvido a partir da infra-estrutura proposta e do conjunto de tecnologias escolhido foi denominado GEIC (Gerenciador de Ensino Individualizado por Computador), e a equipe responsável pelo seu desenvolvimento é composta por pesquisadores e bolsistas técnicos tanto do LECH quanto do LINCE. Essa equipe, além de auxiliar no processo de criação do sistema, foi fundamental também para a definição da infra-estrutura, visto que em vários momentos partes tanto da infra-estrutura quanto do sistema foram realizados simultaneamente.

Todo o código-fonte do sistema está disponível gratuitamente através do *site* oficial[27], sendo que a licença escolhida foi a GPL versão 3 ou superior (FSF, 2007). Já a documentação completa (manuais, diagramas etc.) está disponível sob a licença GFDL versão 1.3 ou superior (FSF, 2008).

6.1 ESTÍMULOS E TENTATIVAS

Escolhidas as tecnologias para se desenvolver um sistema a partir da infra-estrutura proposta, foi necessário escolher um conjunto básico de estímulos e tentativas que serviriam como unidades básicas para a criação de PEIs. O LECH possui um grande histórico de desenvolvimento e aplicação de PEIs, assim como um conjunto já consolidado de técnicas e métodos baseados no paradigma de equivalência de estímulos (SOUZA; DE ROSE, 2006). Assim, decidiu-se por utilizar no sistema as unidades de ensino mais consagradas e em uso pelo maior número de pesquisadores.

Os estímulos disponíveis no GEIC são:

- **Figura**: gravuras, símbolos e fotografias;
- **Som**: mensagens de instrução, correção e conseqüência, palavras e sílabas, normalmente diferenciadas por voz masculina e feminina;

[27] http://www.dc.ufscar.br/geic

- **Texto**: letras, sílabas e palavras.

Os tipos de tentativas disponíveis são:

- **Emparelhamento com o modelo**: representada pela sigla MTS (do original em inglês *matching-to-sample*), consiste em um estímulo de modelo e até três estímulos de comparação, onde um estímulo de comparação deve ser selecionado pelo aluno. A Tabela 1 apresenta as diferentes combinações (subtipos) que podem ser geradas a partir dos diferentes tipos de estímulo. Observe que comparações de som não são utilizadas. A tentativa de MTS assim como ela é exibida aos alunos durante a execução de uma sessão pode ser conferida na Figura 39 (esquerda).

Tabela 1- Subtipos de tentativas MTS

Modelo/Comparação	**Figura**	**Som**	**Texto**
Figura	MTS_FF	**X**	MTS_FT
Som	MTS_SF	**X**	MTS_ST
Texto	MTS_TF	**X**	MTS_TT

- **Resposta construída**: representada pela sigla CR (do original em inglês *constructed response*), consiste em um estímulo de modelo e um conjunto de estímulos de texto (geralmente letras e sílabas), onde esses devem ser selecionados para se formar a palavra correta. A Tabela 2 apresenta as diferentes combinações de tentativas CR. Observe que apenas comparações do tipo texto são utilizadas. A tentativa de CR assim como ela é exibida aos alunos durante a execução de uma sessão pode ser conferida na Figura 39 (centro).

Tabela 2 - Subtipos de tentativas CR

Modelo/Comparação	**Figura**	**Som**	**Texto**
Figura	**X**	**X**	CR_FT
Som	**X**	**X**	CR_ST
Texto	**X**	**X**	CR_TT

- **Nomeação ou ditado**: representada pela sigla NOM, consiste em apenas um modelo, frente ao qual o aluno deve ou emitir uma resposta verbal ou escrita (em uma folha de papel, por exemplo). Dessa forma, tentativas NOM não possuem estímulos de comparação e seus subtipos se diferenciam apenas pelo tipo do modelo. Como se pode observar na Tabela 3, as combinações possíveis são NOM_F,

NOM_S e NOM_T. A tentativa de NOM assim como ela é exibida aos alunos durante a execução de uma sessão pode ser conferida na Figura 39 (direita).

Tabela 3 - Subtipos de tentativas NOM

Modelo/Comparação	Figura	Som	Texto
Figura	NOM_F	**X**	**X**
Som	NOM_S	**X**	**X**
Texto	NOM_T	**X**	**X**

6.2 MÓDULOS DESENVOLVIDOS

Devido à infra-estrutura proposta neste trabalho ter sido em sua maior parte elaborada sem se restringir a tecnologias específicas, foi necessário selecionar para o desenvolvimento dos módulos do sistema, entre várias opções, as tecnologias que melhor se adaptavam aos requisitos do domínio e às especificações de arquitetura e projeto.

Alguns dos critérios por trás de todas as escolhas foram que a tecnologia deveria ser livre (*free-software*) ou pelo menos ter código-fonte aberto (*open-source*), estar em desenvolvimento ativo (isto é, não estar há mais de um ano sem atualizações), possuir uma grande base de usuários, ter vasta documentação disponível e não exigir a instalação de bibliotecas adicionais nos computadores dos clientes. As decisões gerais e que afetam todos os módulos são:

- **Plataforma de desenvolvimento geral**: a plataforma *Java*[28] se mostrou a mais adequada, pois está presente em praticamente todos os tipos de dispositivos (computadores pessoais, TVDI, dispositivos móveis etc.), possui ótima documentação disponível, é gratuita e possui uma grande base de programadores;
- **Plataforma de desenvolvimento para computadores pessoais**: *Java SE 1.6*[29], devido à sua rápida evolução, documentação disponível, rica biblioteca padrão de desenvolvimento e à grande quantidade de produtos disponíveis (servidores, *frameworks* etc.);

[28] http://java.sun.com

[29] http://java.sun.com/javase

- **Ambiente de desenvolvimento**: *Netbeans* 6[30], pois além de ser um dos ambientes mais populares para o desenvolvimento de aplicações *Java*, possui uma ótima integração com o servidor *Glassfish*. Outros pontos favoráveis são o fato do *Netbeans* ser desenvolvido pela própria *Sun Microsystems* (criadora da plataforma *Java*) e de possuir várias facilidades para criação de interfaces de usuário utilizando a biblioteca Swing;

Assim como especificado na etapa de projeto, os módulos desenvolvidos para o GEIC dividem-se em dois conjuntos: módulos-servidor e módulos-cliente.

6.2.1 MÓDULOS-SERVIDOR

Todos os módulos servidor, assim como especificado no capítulo 5.3.3.1 , possuem uma estrutura muito semelhante baseada nos padrões arquiteturais SOA e multicamadas, logo é natural que todos sejam codificados a partir de um conjunto comum de tecnologias. As opções utilizadas foram:

- **Plataforma empresarial**: *Java EE 5*, uma plataforma bastante popular para o desenvolvimento de aplicações empresariais e servidores, foi escolhida devido à sua maturidade e disponibilidade de documentação e ferramentas auxiliares;
- **Contêiner *Web* e servidor de aplicação**: *Glassfish V2*[31], já que atualmente ele é a codificação de referência oficial da plataforma *Java EE 5*, é totalmente integrado com o IDE *Netbeans 6* e possui um rico console de administração de recursos via web;
- **Sistema operacional**: *GNU/Linux Ubuntu 8.04 LTS*, já que sistemas *GNU/Linux* são bastante populares para uso em servidores. A distribuição *Ubuntu* 8.04 foi escolhida já que a equipe de desenvolvimento possuía familiaridade com esse sistema e também por ser uma versão com suporte estendido (*Long Term Support*);

[30] http://www.netbeans.org

[31] https://glassfish.dev.java.net

A especificação *Java EE 5* define um tipo de aplicação chamada "aplicação empresarial" ("*enterprise application*") na qual vários módulos *Java EE* podem ser agregados para se formar uma solução completa. Dessa forma, todos os módulos-servidor foram criados como aplicações empresariais *Java EE* compostas de sub-módulos, cada qual representando uma camada do modelo multicamadas.

Atualmente, por limitações de recursos, todos os módulos-servidor foram implantados em um único servidor. Felizmente, o servidor de aplicação *Glassfish* pode ser modularizado com a criação de servidores virtuais ou domínios (*domains*) isolados, o que permitiu que cada módulo-servidor utilizasse um domínio específico sem interferir nos demais.

Os módulos-servidor desenvolvidos para o GEIC são:

6.2.1.1 SERVIDOR

O servidor de produção do GEIC (*servidor*) é o núcleo do GEIC, fornecendo todos os Serviços *Web* necessários para a realização da interface com os módulos-cliente e os *proxies*, além de conter as unidades de negócio do domínio.

Na camada Web são disponibilizados Serviços *Web* (geralmente um para cada módulo cliente), que funcionam como fachadas remotas para o sistema interior. Toda esta camada é empacotada como sendo uma aplicação *Web* chamada *servidor-war* composta exclusivamente de Serviços *Web*.

Na camada de lógica de negócios encontram-se EJBs de sessão (*stateless session beans*) locais baseados na especificação EJB 3.0 e, para realizar a persistência dos dados, são utilizados entidades (*entity beans*) em conjunto com um *framework* para mapeamento objeto-relacional. Toda a camada de lógica é reunida em um pacote de *beans* chamado *servidor-ejb*.

O API para mapeamento objeto-relacional utilizado foi *o Java Persistence API* (JPA)[32], pois com ela os programadores ficam livres para escolher, em tempo de implantação, qual *framework* de mapeamento objeto-relacional (*Object-Relational Mapping* – ORM) usar,

[32] http://java.sun.com/developer/technicalArticles/J2EE/jpa

tais como *ToplinkEssentials*[33] e *Hibernate*[34]. Além disso, o uso de JPA em conjunto com o provedor de persistência gratuito *Toplink Essentials* se mostrou uma solução extremamente simples e eficaz.

O banco de dados utilizado foi *MySQL 5*[35], devido à maior base instalada de usuários, melhor documentação, maturidade, facilidade de uso e ótima integração com a plataforma *Java*.

6.2.1.2 MENSAGENS

O servidor de troca de mensagens (*mensagens*) é utilizado para permitir a troca assíncrona de mensagens entre os módulos-cliente *player* e *tutor* (descritos adiante). Possui uma camada *Web* (*mensagens-war*) com apenas um Serviço *Web* e uma camada de negócios (*mensagens-ejb*) onde estão localizadas as filas de mensagens.

A tecnologia utilizada para a codificação das filas de mensagens foi JMS[36] (*Java Message Service*), que é uma API Java para envio entre dois ou mais clientes. JMS é parte do *Java EE*, e é um padrão que permite a módulos criarem, enviarem, receberem e lerem mensagens, realizando assim comunicação distribuída fracamente acoplada, confiável e assíncrona.

6.2.1.3 *PROXY MÓVEL*

O *proxy* para dispositivos móveis (*proxymovel*) é uma aplicação empresarial contendo na camada *Web* (*proxymovel-war*) alguns *Servlets*[37] para fazerem a interface HTTP com os dispositivos móveis, e na camada de negócios (*proxymovel-ejb*) o *cache* e EJBs que acessam os serviços do servidor de aplicação e adaptam o conteúdo das sessões de ensino.

---

[33] http://www.oracle.com/technology/products/ias/toplink/jpa/index.html

[34] https://www.hibernate.org

[35] http://www.mysql.com

[36] http://java.sun.com/products/jms

[37] http://java.sun.com/products/servlet

6.2.1.4 *PROXY TVDI*

O *proxy* para a TVDI (*proxytvdi)* é uma aplicação empresarial contendo na camada *Web* (*proxytvdi-war*) alguns *Servlets* para fazerem a interface HTTP com os STBs, e na camada de negócios (*proxytvdi-ejb*) o *cache* e EJBs que acessam os serviços do servidor de aplicação.

Foi utilizado como canal de descida, emulando um sistema de difusão terrestre, a mesma comunicação utilizada no canal de interatividade para comunicação com o cliente, ou seja, rede IP.

6.2.2 MÓDULOS-CLIENTE

Todos os módulos-cliente, assim como especificado no capítulo 5.3.3.2 , possuem uma estrutura semelhante. Na plataforma PC o seguinte conjunto de tecnologias foi utilizado:

- **Implantação remota de aplicações cliente:** foi utilizado *Java WebStart*[38] para fazer com que as aplicações *desktop* pudessem ser implantadas remotamente. Isso fez com que os usuários não precisassem instalar o módulo localmente, dando a sensação de que uma aplicação *Web* estava sendo executada.
- **Bibliotecas para desenvolvimento de interfaces de usuários**: *Java Swing*[39], devido à escolha da linguagem oficial ter sido *Java*. *Swing*, ao contrário de outras bibliotecas para interfaces de usuário, acompanha o kit de desenvolvimento *Java*, o que garante que arquivos adicionais não precisarão ser instalados nos computadores dos usuários;

Com exceção do módulo *site* (descrito a seguir), todos os outros módulos-cliente para plataforma PC foram codificados como aplicações *desktop* acessíveis através do navegador via *Java WebStart*. A opção por não codificar os módulos-cliente como aplicações

[38] http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp

[39] http://java.sun.com/docs/books/tutorial/uiswing

Web partiu do requisito do melhor uso possível da rede de comunicação, além da exigência de que nenhuma interrupção poderia ser feita em determinados momentos, tais como durante uma sessão de ensino.

Garantir esse tipo de comportamento, mesmo em aplicações *Web* ricas com uso de AJAX (*Asynchronous JavaScript And XML*), ainda hoje é um desafio. Essa opção se tornará viável a partir do momento em que os principais navegadores *Web* disponíveis no mercado passem a seguir de maneira fiel alguns padrões, tais como HTML, CSS e *JavaScript*, evitando assim que código-fonte precise ser adaptado para navegadores específicos.

Os módulos-cliente são:

6.2.2.1 SITE

O *site* oficial do GEIC (módulo *site*) é o ponto de acesso para todas as tarefas relacionadas ao gerenciamento de PEIs. Ao contrário de todos os outros módulos-cliente para PC, o *site* é uma aplicação *Web*. A tecnologia escolhida para a geração dinâmica de páginas foi *Java ServerPages* (JSP)[40], pois permite o desenvolvimento rápido de aplicações *Web* independentes de servidor e plataforma.

Na Figura 30 pode ser observada a tela que é exibida logo após o super-usuário fornecer seu nome de usuário e sua senha. Na coluna da esquerda (módulos disponíveis) estão os módulos que podem ser executados (no caso de um aluno, apenas o módulo de execução de sessões estaria disponível). Na coluna central estão as notícias relacionadas ao GEIC, tais como novas versões do sistema, reuniões, avisos etc. Na coluna da direita está um placar com um resumo da situação geral do sistema, contendo informações como número de sessões executadas, número de PEIs disponíveis, cidades cadastradas etc.

[40] http://java.sun.com/products/jsp

Figura 30- *Site* do GEIC na visão do super-usuário

6.2.2.2 ADMINISTRAÇÃO DO SITE

O módulo para administração do *site* (*siteadmin*) foi criado para permitir que aspectos dinâmicos do módulo *site*, tais como notícias e chamadas na página principal, fossem feitos de maneira mais confortável e, acima de tudo, com um controle de permissões.

A princípio apenas o super-usuário do GEIC possui permissões para realizar mudanças no *site*, controlar notícias etc., mas no futuro essa tarefa deverá ser representada na forma de permissões específicas. Assim, essas permissões poderão ser repassadas para outros membros de equipe, que poderão colaborar com a adição de conteúdo no *site*.

Além de permitir o controle de notícias, é possível através do módulo *siteadmin* obter informações importantes como o número de acessos em cada notícias, a origem dos acessos no *site*, número total de acessos na página principal etc.

Uma das telas do módulo *siteadmin* pode ser conferida na Figura 31, onde pode ser vista a lista de notícias cadastradas, assim como a quantidade de acessos de cada uma.

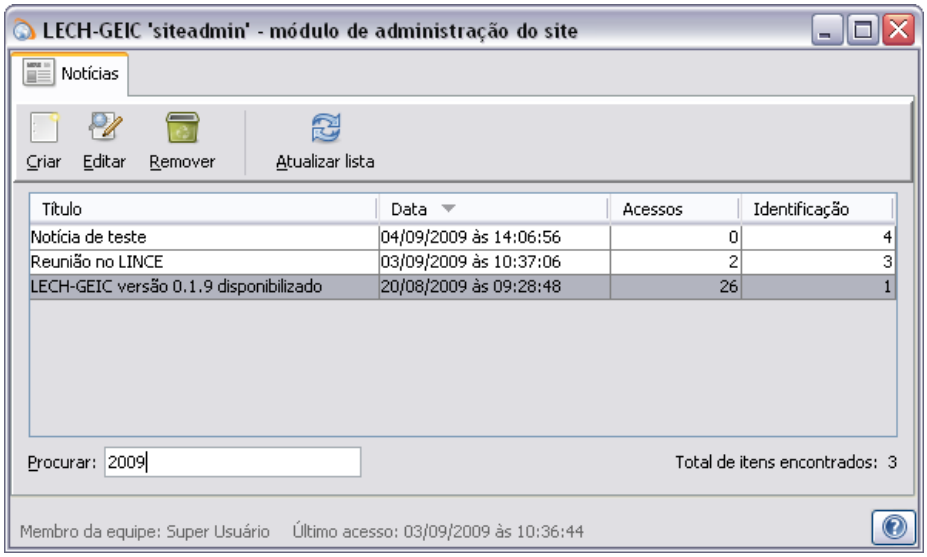


Figura 31 - Lista de notícias na administração do site

6.2.2.3 ALUNOS

O módulo para gerenciamento de alunos (*alunos*) é bastante simples e permite o gerenciamento de alunos (criar, editar, remover, associar programas etc.), escolas e cidades. Na Figura 32 pode ser observado o uso de um campo de busca retornando os alunos encontrados (esse campo de busca retorna resultados em tempo-real e está presente em todos os módulos-cliente).

Figura 32 - Busca no módulo de gerenciamento de alunos

6.2.2.4 AUTORIA

O módulo de autoria de PEIs (*autoria*) é um dos mais utilizados pelos especialistas de domínio, já que a tarefa de se cadastrar estímulos, criar tentativas e compor um PEI é bastante demorada. Na Figura 33 pode-se observar a lista de PEIs já criados. Além do nome, descrição e número de identificação, é exibida também a situação (*status*) de cada programa, que pode ser:

- **Em edição**: os trabalhos nele ainda não estão concluídos e, logo, alunos ainda não podem executá-lo;
- **Disponibilizado**: sua edição foi concluída e alunos já podem executá-lo. PEIs nesse estado nunca mais poderão ser editados;
- **Bloqueado**: alunos não podem mais executá-lo. Um PEI é colocado nesse estado geralmente devido a um defeito ter sido encontrado após a disponibilização, ou ainda quando uma versão melhorada do PEI foi criada e os novos alunos devem ser movidos.

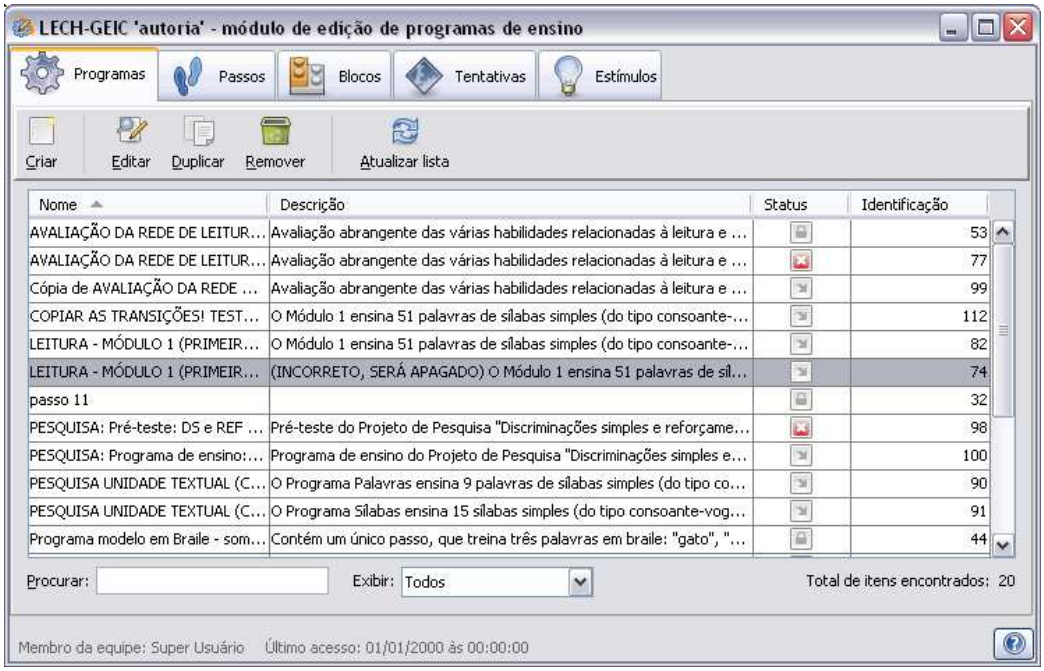


Figura 33 - Lista de PEIs no módulo autoria

Na Figura 34 está um diálogo de criação de tentativa MTS-FF, onde pode ser observado que tanto as mensagens quanto os estímulos de modelo e comparação podem ser

conferidos (assim como eles aparecerão para os alunos durante a execução de um PEI) em tempo de edição. Essa facilidade diminui as chances de uma tentativa incorreta ser disponibilizada.

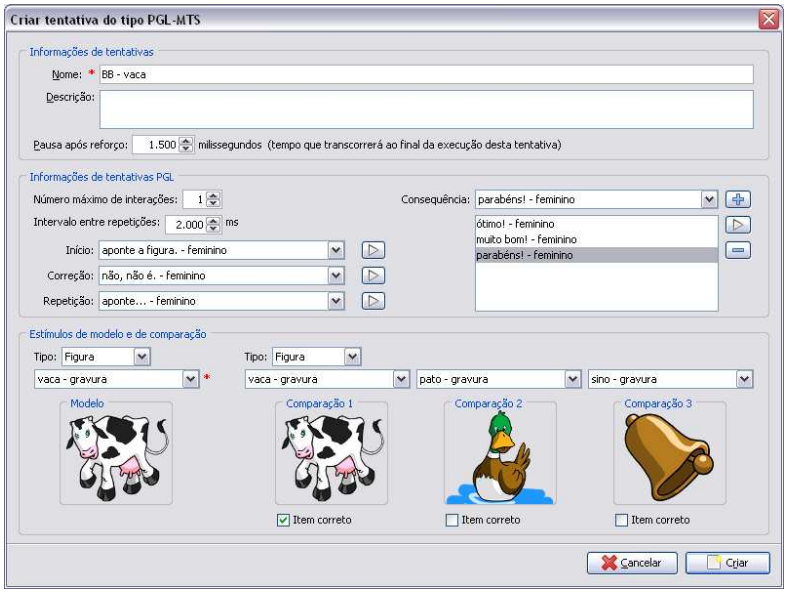


Figura 34 - Criação de tentativa de emparelhamento no módulo de autoria

Na Figura 35 pode ser observada a edição de um programa, que por sua vez é um conjunto de passos. Na porção central da tela encontram-se as ocorrências de passos e as transições entre elas, e pode-se perceber que conceitos da teoria de máquinas de estado finitas, tais como estados, estados iniciais, transições e auto-transições, estão devidamente representados.

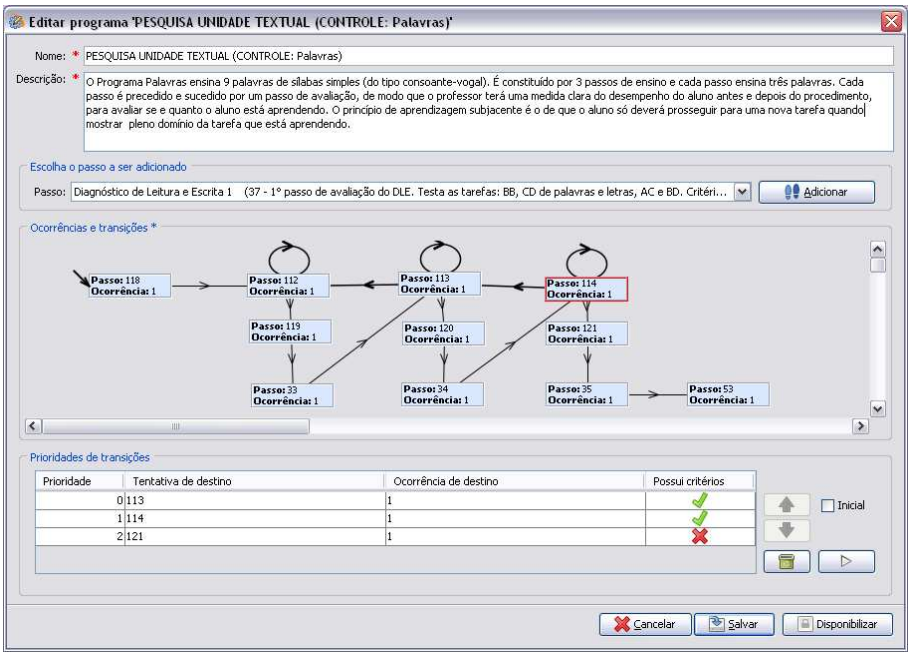


Figura 35 - Criação de programa (conjunto de passos) no módulo de autoria

Na Figura 36 observa-se um teste de passo. Na parte superior da tela são mostradas as teclas de atalho disponíveis. Na parte inferior são mostrada a identificação e o nome das unidades de ensino correntes. Já no centro da tela é mostrada a tentativa assim como ela será mostrada ao aluno, porém com dicas visuais destacando qual o estímulo de comparação correto. A funcionalidade de testes reduziu drasticamente a quantidade de erros cometidos pelos especialistas de domínio, já que as unidades de ensino mais básicas podem ser testadas antes de serem incorporadas a unidades mais complexas.

Figura 36 - Teste de sessão no módulo de autoria

6.2.2.5 CONSULTA

Um exemplo de consulta das tentativas executadas em um bloco pode ser visto na Figura 37. Podem ser observados atributos como latência das respostas, estímulos selecionados, mensagens de conseqüência recebidas etc. Caso o especialista deseje conferir como é a tentativa à qual aquela execução se refere, basta pressionar o botão de informações para que seja exibida a tentativa assim como ela foi criada no módulo de autoria.

Figura 37 - Bloco de tentativas executadas visto através do módulo *consulta*

6.2.2.6 EQUIPE

E através do módulo de gerenciamento de equipe (*equipe*) que a hierarquia de pessoas ilustrada na Figura 27 pode ser concretizada. Este módulo é bastante sucinto, assim como pode ser visto através da Figura 38.

Figura 38 - Lista de membros de equipe

6.2.2.7 PLAYER

As tentativas são exibidas aos alunos assim como na Figura 39. Na coluna da esquerda está uma tentativa *MTS_SF*, na coluna do meio uma tentativa *CR_ST* e na coluna da direita uma tentativa *NOM_T*.

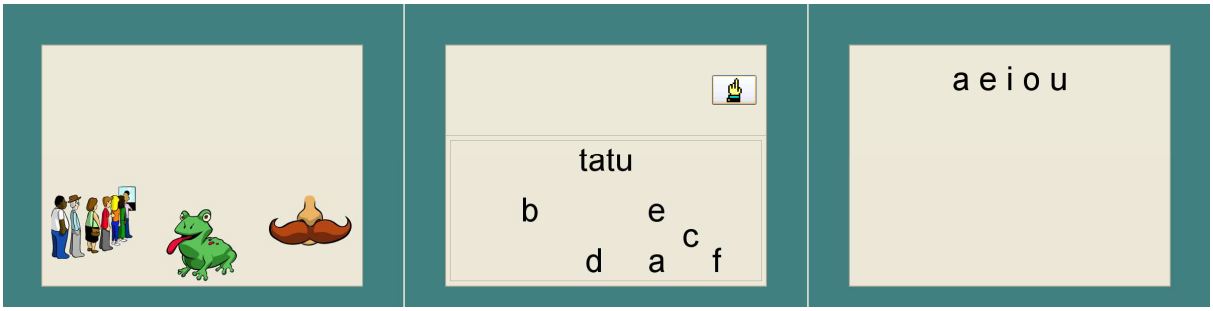


Figura 39 - Tentativas exibidas no *player*

6.2.2.8 PLAYER MÓVEL

Para o desenvolvimento de um *player* para dispositivos móveis foi utilizada a plataforma de desenvolvimento *Java ME*[41], pois é bastante popular entre programadores casuais e profissionais, possui código-fonte disponível e também ótimas ferramentas para a realização de testes e simulações. O tipo de plataforma utilizado foi *CLDC 1.1/MIDP 2.1*, já que esse se encontra presente na maioria dos telefones celulares.

Na Figura 40 pode ser vistas duas telas do *playermovel* sendo executadas em dois simuladores distintos de dispositivos móveis. Na esquerda observa-se a tela principal do módulo em um celular colorido simples, enquanto na direita observa-se a execução de uma tentativa em um celular com teclado *QWERTY*.

[41] http://java.sun.com/javame

Figura 40 - Player para dispositivos móveis

6.2.2.9 PLAYER TVDI

A Figura 41 ilustra uma das telas de tentativas produzidas para apresentação via TVDI. Note que, ao contrário do que ocorre no *playermovel*, a aparência de uma tentativa é praticamente idêntica daquela exibida no *player* tradicional. A única diferença visível é a presença de um cursor controlado pelo controle remoto (em torno da palavra “bolo”), já que não se pode fazer uso de um mouse para seleção de estímulos.

Este *player* encontra-se em fase de protótipo, sendo que para seu desenvolvimento foi utilizada uma versão de desenvolvimento do *Ginga* (SOARES; FILHO, 2007)[42], denominação do *middleware* do Sistema Brasileiro de TV Digital (SBTVD), além do ambiente declarativo *Ginga-NCL* (SOARES; RODRIGUES; MORENO, 2007)[43], que utiliza a lin-

[42] http://www.ginga.org.br

[43] http://www.gingancl.org.br

guagem *Nested Context Language* (NCL)[44] e a linguagem de script *Lua*[45]. O ambiente declarativo foi escolhido em detrimento do ambiente procedural *Ginga-J*[46] (Filho 2007), baseado na plataforma *Java* e que seria a escolha mais apropriada para a programação altamente interativa, devido a incertezas à época quanto sua adoção e a questões de licença.

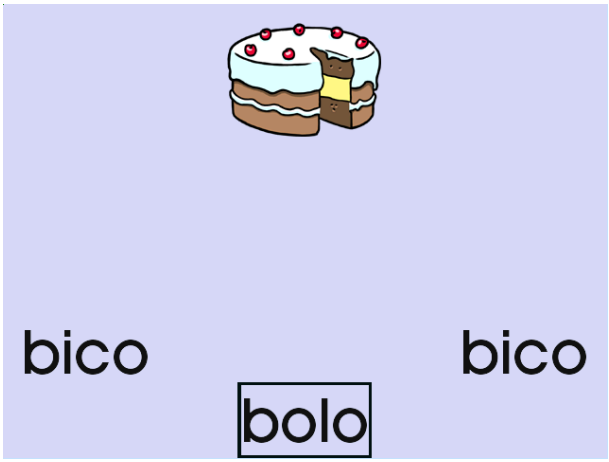


Figura 41 - Tentativa MTS no player para SBTVD

6.2.2.10 TUTOR

Além de apenas acompanhar a sessão, o tutor pode participar ativamente através da análise subjetiva de algumas tentativas (especialmente as de nomeação – NOM), a criação de comentários (que serão registrados ao final da sessão) e, dependendo do ambiente computacional, enviar comandos de voz.

Conforme as principais necessidades da equipe de especialistas do LECH, o módulo *tutor* suporta o modo *1:1se* (um aluno por tutor, avaliação síncrona de resposta, acompanhamento exclusivo). Modos adicionais, tais como avaliação assíncrona e dedicação inclusiva, serão adicionados nas próximas versões do sistema.

Na Figura 42 podem-se observar os elementos gráficos exibidos ao tutor durante uma sessão de ensino, com monitoria do tipo *1:1se*. A aparência é muito semelhante ao teste de sessões do módulo *autoria*, porém aqui existem informações adicionais como o nome do

[44] http://www.ncl.org.br

[45] http://www.lua.org

[46] http://www.lavid.ufpb.br

aluno sendo avaliado. Há também botões que facilitam para que o tutor interfira no resultado de alguma tentativa.

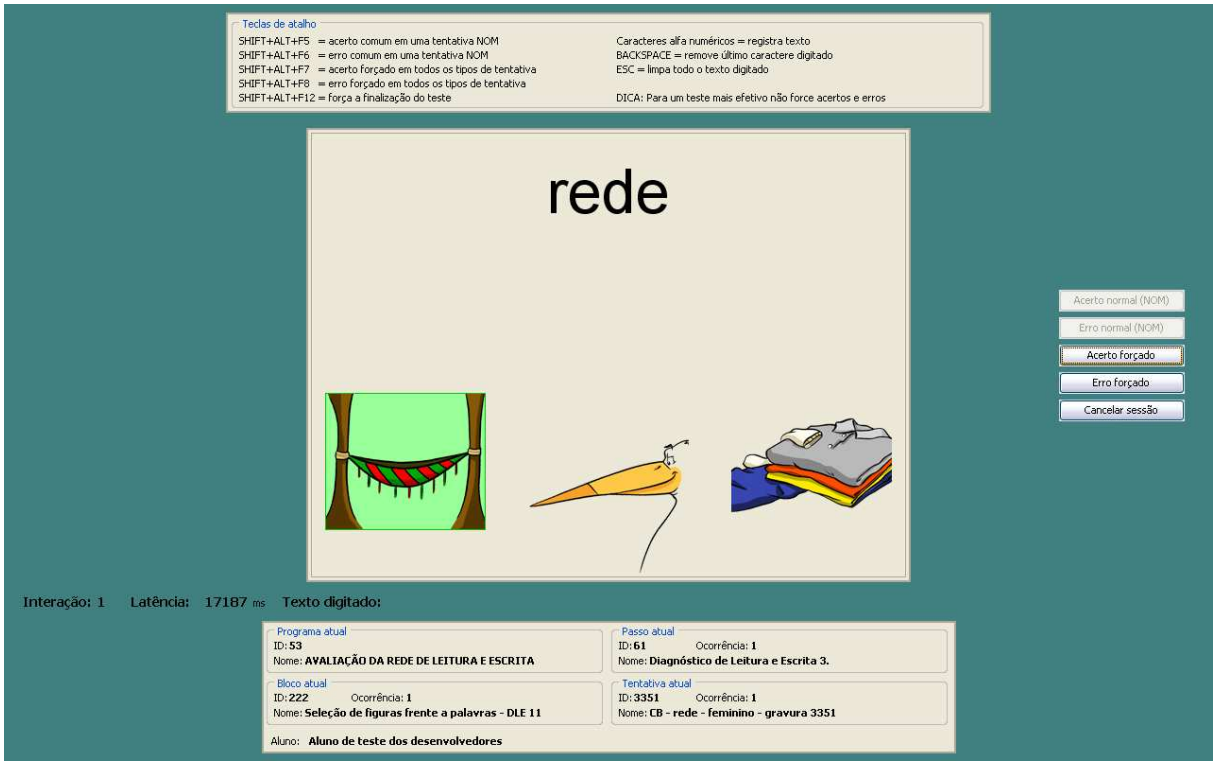


Figura 42 - Tutoria durante sessão de ensino

6.3 ESTÁGIO ATUAL

Assim como discutido na metodologia (capítulo 4), versões funcionais do sistema foram liberadas à medida que progressos foram sendo realizados na parte de desenvolvimento, e desde o princípio especialistas da área de psicologia ofereceram opiniões e críticas e participaram de sessões de teste. Em paralelo a isso, outro conjunto de especialistas desenvolveu seus PEIs tanto para fins de pesquisa quanto para oferecer *feedback* à equipe de desenvolvimento.

Conforme o sistema foi sendo desenvolvido tornou-se possível criar e aplicar PEIs inteiramente de maneira informatizada. Um exemplo de PEI já em fase de aplicação inclui o ensino de leitura e escrita para crianças com atraso no desenvolvimento inseridas em escolas especiais de ensino, utilizando para isso discriminações simples e reforçamento espe-

cífico (ZAINE, 2009). Outros PEIs, ainda em fase de desenvolvimento ou teste, incluem ensino de Braile, leitura e escrita para crianças do ensino fundamental, ensino de matemática etc.

Apesar de já estar em uso por vários pesquisadores, o sistema se encontra em constante desenvolvimento. Se forem levadas em conta a quantidade de universidades interessadas neste projeto e a velocidade com que o sistema está sendo adotado, pode-se concluir que ele evoluirá ainda mais para atender às demandas dos especialistas da área de ensino. Até o momento foram feitas palestras no LINCE, LECH e CECH (Centro de Educação e Ciências Humanas) da UFSCar e Pontifícia Universidade Católica de São Paulo (PUCSP), e parcerias com pesquisadores da Universidade Federal de Brasília (UnB) e Universidade Federal do Pará (UFPA).

A edição de produção (implantada nos servidores oficiais) do GEIC encontra-se atualmente na versão 0.1.9, o que significa que uma grande versão foi liberada seguida de nove pacotes de funcionalidade e correções de erros. Nessa edição estão presentes os módulos *alunos*, *autoria*, *consulta*, *equipe*, *player*, *site* e *siteadmin*. Esses módulos obtiveram prioridade no processo de desenvolvimento, já que são indispensáveis para que o GEIC pudesse ser usado em ambiente de produção.

A edição de desenvolvimento (0.2.0), que ainda não foi liberada para os usuários, contém o restante dos módulos especificados no capítulo 5.3.3 , que são: *playermovel*, *playertvdi*, *proxytvdi* e *proxymovel*. Essa versão já se encontra em um estágio bastante avançado de desenvolvimento, com liberação para os usuários agendada para o final de 2009. O sistema será também integrado ao ambiente colaborativo do projeto *Tidia-Ae*, baseado no *Sakai CLE* (SAKAI, 2009). Isso implicará na criação de dois módulos adicionais: *playersakai* e *tutorsakai*. Essa integração proverá recursos importantíssimos para a realização remota de monitorias, tais como comunicação síncrona de áudio e vídeo entre aluno e tutor, lousas virtuais etc.

7 AVALIAÇÃO

O GEIC, que foi construído a partir da infra-estrutura apresentada neste trabalho, já está em uso por diversos pesquisadores e alunos. Além disso, sua adoção como sistema para desenvolvimento e aplicação de PEIs facilitou em muito o trabalho dos especialistas de domínio e permitir que, ao mesmo tempo em que mais PEIs fossem criados, mais alunos fossem atendidos.

Porém, para se obter uma avaliação mais precisa a respeito da qualidade tanto da infra-estrutura quanto do sistema desenvolvido, é necessária a realização de estudos adicionais com controles melhores de execução, medição, investigação e repetição. Dessa forma, foram realizados, além do estudo de caso, alguns estudos experimentais voltados tanto para especialistas de domínio quanto para engenheiros de *software* e programadores.

Essas experimentos seguem uma forma simplificada do estudo experimental realizado por Travassos (2002), e são compostas por um questionário voltado para os especialistas de domínio (psicólogos e pesquisadores) envolvidos durante o desenvolvimento do sistema (estudo 1), um experimento voltado para especialistas do domínio sem prévio conhecimento do sistema (estudo 2) e um experimento voltado para engenheiros de *software* e programadores sem prévio conhecimento da infra-estrutura (estudo 3).

Foi realizada também uma avaliação subjetiva a respeito dos conceitos e técnicas utilizados para o desenvolvimento deste trabalho. Essa avaliação (estudo 4), apesar de ser anedótica e não contar com o mesmo formalismo presente nos outros três estudos, proporciona uma visão mais prática a respeito da evolução do projeto e das vantagens e desvantagens de cada escolha realizada.

7.1 ESTUDO 1 – ESPECIALISTAS DO DOMÍNIO ENVOLVIDOS COM O SISTEMA

7.1.1 DEFINIÇÃO DOS OBJETIVOS

- **Objetivo global**: Avaliar a qualidade do sistema, levantar seus pontos fortes e fracos, coletar sugestões para futuras melhorias e descobrir se o processo adotado pa-

ra o desenvolvimento da solução evoluiu adequadamente conforme as necessidades dos usuários. Tudo isso do ponto de vista dos especialistas de domínio envolvidos durante o processo de desenvolvimento da solução.

- **Objetivo da medição**: tendo como base as atuais funcionalidades do sistema, caracterizar:
    - A qualidade do sistema;
    - Os principais pontos fortes do sistema;
    - Os principais pontos fracos do sistema;
    - A adequação do processo de desenvolvimento frente às necessidades dos usuários;
    - Sugestões para futuras melhorias.

7.1.2 PLANEJAMENTO

- Descrição da instrumentação:
    - **Dados pessoais**: 10 questões abertas sobre linha de pesquisa seguida pelo indivíduo, tempo de uso do GEIC, familiaridade com cada módulo etc.;
    - **Qualidade do sistema**: 16 questões fechadas baseadas no padrão ISO 9126 (ISO 1991), com respostas na escala de 0 a 3;
    - **Futuras versões**: 3 questões abertas sobre as funcionalidades que o indivíduo acha que deveriam ser adicionadas em futuras versões do sistema, assim como sugestões para melhorias específicas em cada módulo;
    - **Em relação sistemas similares**: caso o indivíduo possua experiência prévia com algum sistema semelhante, 3 questões comparando esse outro sistema com o GEIC;
    - **Geral**: 2 questões gerais a respeito do sistema, tais como os principais pontos positivos e negativos de toda a solução.
- **Seleção do contexto**: O estudo supõe o processo *off-line* porque os especialistas não estão sendo entrevistados durante todo seu uso do sistema, mas em certo instante. Os participantes são os especialistas do domínio que participaram como pesquisadores durante o desenvolvimento do sistema. O estudo é modelado por-

que são utilizadas notas subjetivas. O contexto possui um caráter geral, pois são feitas comparações com outros sistemas.

- **Seleção dos indivíduos**: 5 alunos de graduação ou pós-graduação em psicologia da UFSCar, com a condição de haverem participado do processo de desenvolvimento do sistema e terem acompanhado a evolução do projeto de *software*, tanto de forma passiva como usuários quanto de forma mais ativa, participando nas reuniões de coleta de requisitos e sessões de teste.

7.1.3 MATERIAIS

Os materiais utilizados no experimento podem ser consultados no Anexo A. São eles:

- Termo de consentimento livre e esclarecido (TCLE), onde o indivíduo aceita participar deste projeto de pesquisa;
- Questionário com 34 questões.

7.1.4 OPERAÇÃO

Os participantes foram contatados pessoalmente e, após assinarem o TCLE, receberam o questionário por *e-mail*. Não era necessário responder o questionário no momento do recebido. Depois de preenchidos, os questionários foram enviados de volta por *e-mail*.

7.1.5 RESULTADOS

Perfil dos participantes

Tabela 4 - Perfis dos participantes (membros da equipe)

Participante	Curso	Período (semestre)	Linha de pesquisa	Experiência com o GEIC (meses)
P1	1	6	1	9
P2	1	8	2	27
P3	2	2	3	9
P4	2	2	4	12
P5	2	concluído	5	2

Tabela 5 - Legenda (perfis dos participantes membros da equipe)

Curso		Linha de pesquisa	
1	Graduação em Psicologia	**1**	Ensino e aprendizagem
2	Mestrado em Psicologia	**2**	Análise e programação de condições de ensino
		3	Comportamento social e processos cognitivos
		4	Análise experimental do comportamento
		5	Informatização de ensino

A partir da Tabela 4 é possível perceber que o perfil dos membros da equipe é bastante variado e com formação acadêmica bastante distribuída, indo desde um participante do sexto semestre da graduação (P1) até um participante com mestrado concluída (P5), e com variados tempos de experiência com o sistema (de 2 a 27 meses).

Nenhum dos participantes segue a mesma linha de pesquisa, porém vale notar que apenas um não realiza pesquisas diretamente relacionadas ao projeto (P3, linha de pesquisa 3), enquanto as linhas de pesquisa do restante possuem relação com o tema do projeto.

A freqüência com que os módulos-cliente foram acessados está esquematizada na Figura 43. De uma escala com os valores 0 (nunca), 1 (pouco), 2 (freqüentemente) e 3 (muito), observa-se que o módulos mais acessado é o de execução de sessão (*player*), seguido pelo de autoria de PEIs (*autoria*), consulta de resultados (*consulta*) e gerenciamento de alunos (*autoria*). Isso se justifica já que esses módulos são de uso rotineiro durante o processo de aplicação de elaboração e aplicação de PEIs, ao contrário do módulo de gerenciamento de equipe (*equipe*), que é raramente acessado apenas pelo super-usuário. O módulo de tutoria remota (*tutor*) encontra-se em fase de testes e ainda não foi liberado aos usuários.

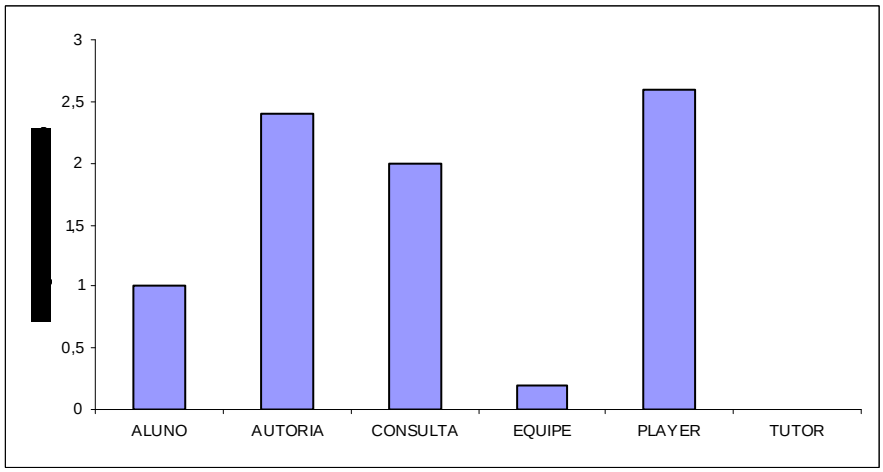


Figura 43 - Freqüência em que os módulos-cliente foram acessados pelos membros da equipe

Qualidade do sistema

Analisada a Figura 44 com a pontuação média nas questões sobre a qualidade do sistema, obtêm-se média 2,4 e mediana 2,5 (de um total de 3), o que é um resultado bastante satisfatório e revela que os usuários que acompanharam o desenvolvimento do sistema estão satisfeitos com sua evolução e seu estado atual.

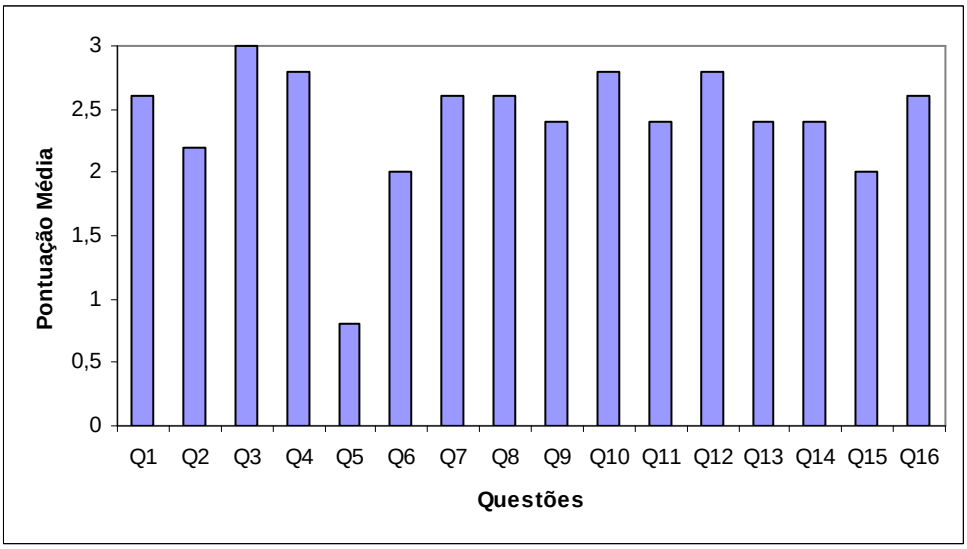


Figura 44 - Pontuação média nas questões sobre a qualidade do sistema (membros da equipe)

Porém, alguns pontos merecem nota. Q3 apresentou a maior média (nota máxima, 3) dentre todas as questões, o que indica que os usuários estão satisfeitos com os quesitos de segurança do sistema. Já Q5 apresentou uma média muito baixa (0,8), o que indica que

os usuários estão insatisfeitos com o sistema em relação à maneira como ele se comporta em caso de ausência de rede de comunicação.

A causa mais provável para essa desaprovação se deve ao fato do sistema ser baseado na *Web* e só funcionar corretamente quando há uma conexão com a Internet disponível, de preferência com grande largura de banda. Isso por si só não constituiria um problema, já que atualmente aplicações Web puras são bastante populares. Porém, um comentário realizado por P3 aponta a real causa do problema: a falta de disponibilidade do servidor (localizado nas dependências do LINCE), que faz com que as tarefas de todos os usuários precisem ser interrompidas em caso de queda de energia ou falha na rede de computadores da universidade (o que é bastante freqüente).

Uma possível solução para esse impasse seria mover o servidor do sistema para fora das dependências da universidade e prover meios para que ele se mantenha disponível a maior parte do tempo. Outra solução seria contratar o serviço de empresas especialistas em hospedagem de *sites*, já que elas oferecem planos variados de qualidade de serviço e alta disponibilidade.

Futuras versões

A Figura 45 aponta os principais interesses dos membros da equipe no que se refere a novos tipos de estímulos a serem suportados pelo sistema. Não é surpresa que o tipo de estímulo composto “figura+som” sejam um dos mais requisitados, já que a classe de estímulos compostos já é utilizada em outros sistemas e os usuários desejam que isso se repita no GEIC. Esse requisito já se encontra aprovado no gerenciamento de tarefas da equipe de desenvolvimento. O tipo “vídeo” também é uma preferência natural, dado seu apelo visual e suas possibilidades de uso. Sua adição ao sistema certamente teria um impacto positivo entre os especialistas do domínio e alunos.

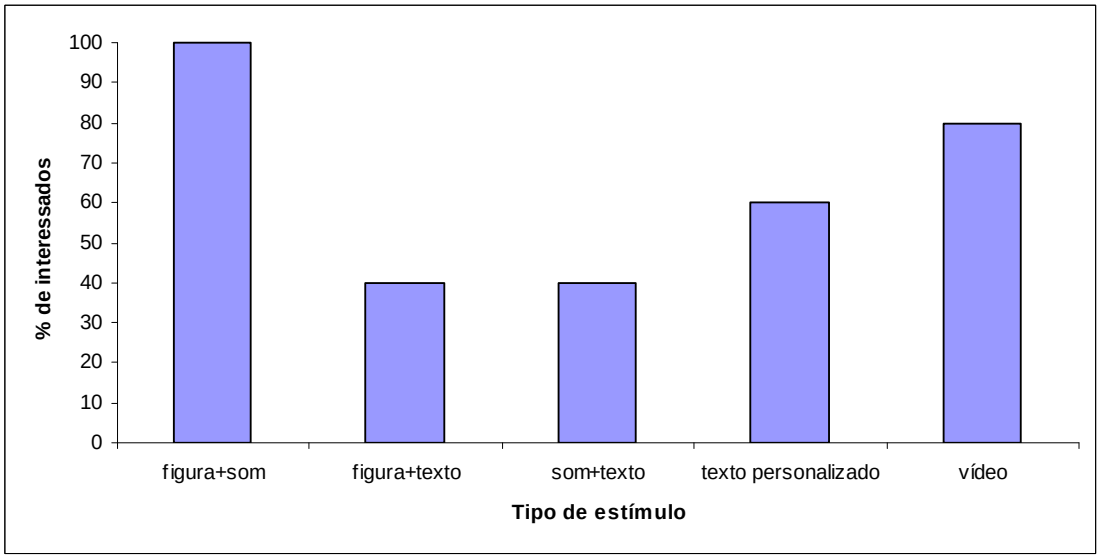


Figura 45 - Interesse pelos membros da equipe em novos tipos de estímulos

A Figura 46 aponta os principais interesses dos membros da equipe no que se refere a novos tipos de tentativas. O tipo "CR com *feedback* seleção a seleção" já vem sendo requisitado há algum tempo e já se encontra aprovado no gerenciador de tarefas da equipe de desenvolvimento. Sua execução é relativamente simples, já que é uma variação do tipo CR tradicional. O tipo "discriminações simples simultâneas" teve o menor interesse, o que talvez se explique pelo fato de ela poder ser "imitada" a partir de uma tentativa MTS comum (bastando para isso não utilizar um estímulo de modelo).

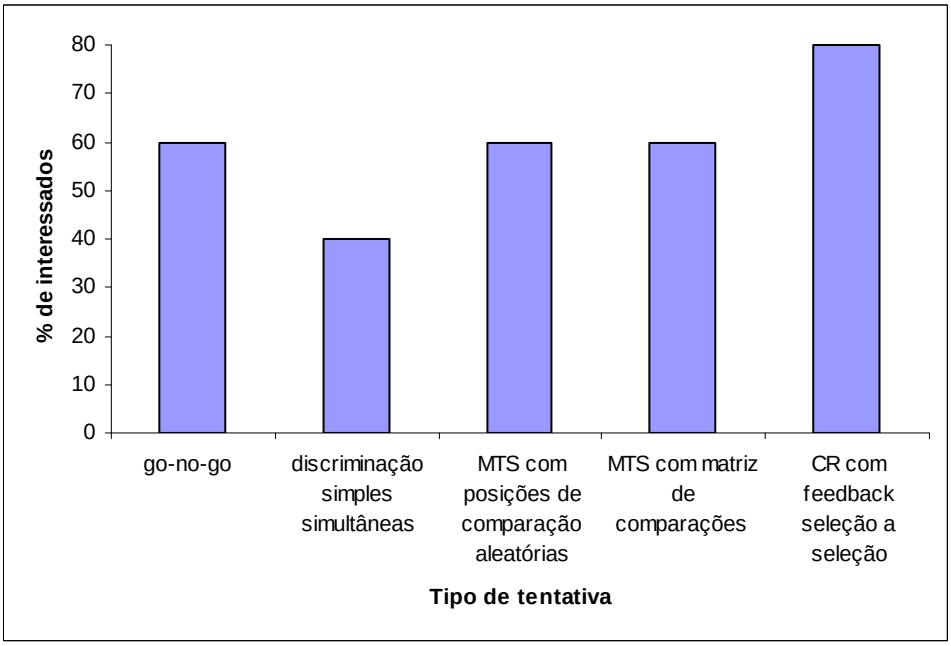


Figura 46 - Interesse pelos membros da equipe em novos tipos de tentativas

A Tabela 6 mostra as melhorias desejadas para cada módulo. Observa-se claramente que em relação ao gerenciamento de pessoas (módulos *alunos* e *equipe*) está sendo requisitado um suporte melhor à formação de grupos e ao controle de permissões sobre esses membros. Já no módulo *autoria* referem-se a uma maior flexibilidade de operação, principalmente no que se refere a disponibilizar PEIs e a alterar unidades de ensino com mais liberdade. Finalmente, no módulo *consulta* há reclamações sobre a apresentação das tabelas de exportação de dados (que são visualmente confusas), além do desejo de se poder gerar gráficos automaticamente, sem a necessidade de se exportar uma tabela e gerar gráficos a partir de uma planilha eletrônica. O módulo *player* não recebeu nenhuma crítica, talvez por ser o mais usado e, logo, um dos mais testados.

Tabela 6 - Melhorias desejadas para cada módulo (membros de equipe)

Módulo	**Melhoria**
Alunos	Aplicar ao aluno por herança algumas funções atribuídas ao grupo ao qual ele está vinculado (escola)
Autoria	Mais autonomia para disponibilizar programas
	Permitir alterar tentativas e blocos sem necessariamente desvincular toda a estrutura do programa
Consulta	As tabelas geradas ainda estão confusas, ficando difícil saber onde começa um conjunto de dados e termina outro
	Geração de gráficos
equipe	Permitir atribuir manualmente quais grupos o membro da equipe será vinculado e limitar quais programas ele terá acesso para associar a cada aluno (ou grupo de aluno, por escola, por exemplo)

Em relação a sistemas similares

Apenas P4 possui experiência prévia com um sistema para avaliação de escolhas entre textos explicativos e exemplificadores de conceitos científicos, na área de Psicologia Experimental, desenvolvido durante seu trabalho de mestrado. De acordo com ele, a principal diferença entre o GEIC e esse outro sistema é que o GEIC se mostrou mais flexível quanto à possibilidade de criar novos treinos, e apresenta uma interface de interação com o usuário mais rica e profissional. Além disso, ele complementa que o GEIC exige um planejamento prévio dos objetivos de ensino a serem alcançados com o programa, condizente com a proposta de prática educacional no qual a Psicologia Experimental faz uso.

Geral

A questão explorada na Figura 47 se refere aos principais pontos positivos do sistema, que são a facilidade para se criar PEIs, as possibilidades de criação de PEIs para dife-

rentes objetivos/populações, o reaproveitamento de unidades de ensino e o sistema visual de encadeamento dessas.

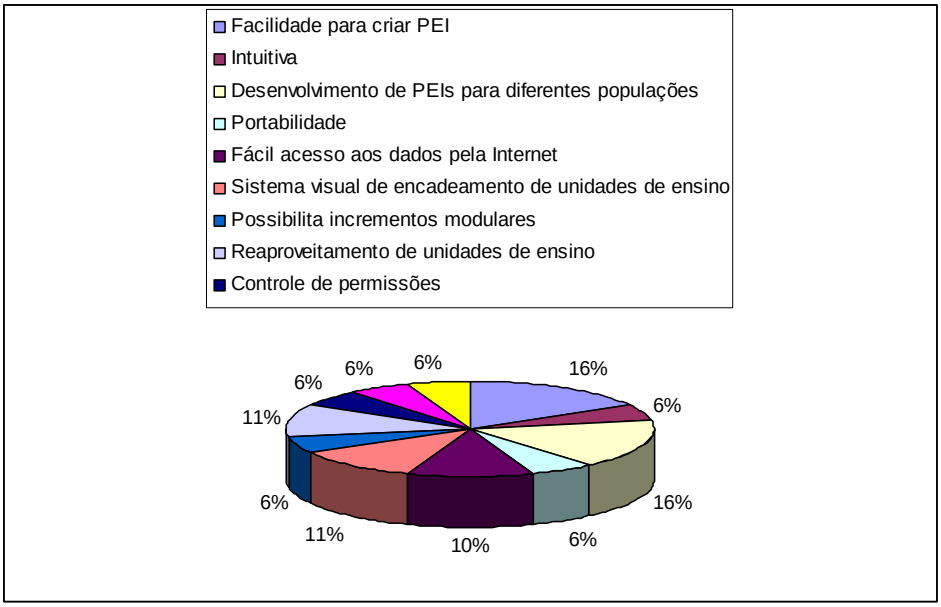


Figura 47 - Principais pontos positivos do sistema (membros de equipe)

Já a questão explorada na Figura 48 se refere aos principais pontos negativos do sistema, alguns diretamente relacionados aos pedidos de melhorias descritos na Tabela 6. São eles a impossibilidade de se consertar erros em unidades de ensino já associadas e as constantes quedas de energia e Internet do servidor do GEIC. Outros pontos são o excesso de detalhes (o que exige um treinamento para novos usuários), não poder recuperar sessões em caso de interrupção de sessão, exigência de Internet rápida e esquema de permissões limitado.

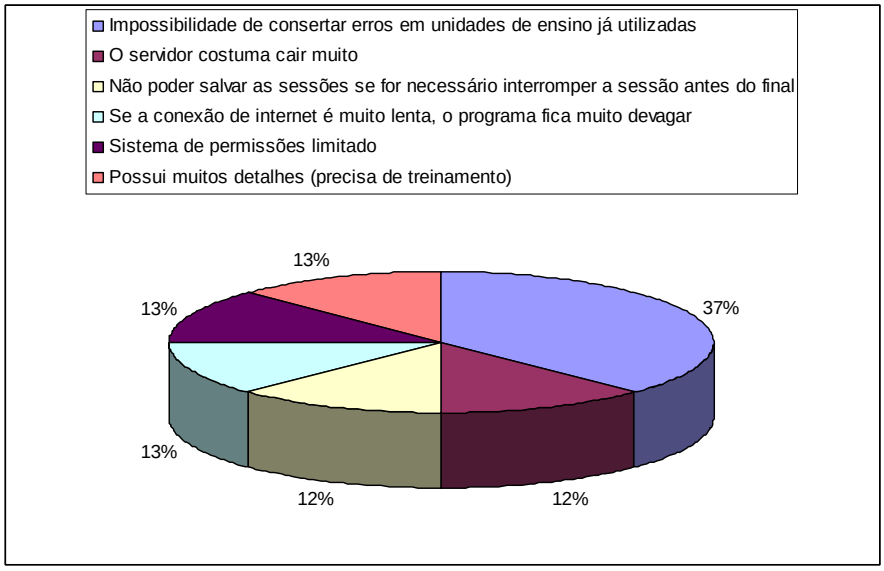


Figura 48 - Principais pontos negativos do sistema (membro de equipe)

7.2 ESTUDO 2 – ESPECIALISTAS DO DOMÍNIO INGÊNUOS EM RELAÇÃO AO SISTEMA

7.2.1 DEFINIÇÃO DOS OBJETIVOS

- **Objetivo global**: Avaliar a qualidade do sistema, levantar seus pontos fortes e fracos e coletar sugestões para futuras melhorias, do ponto de vista dos especialistas de domínio ingênuos com relação ao sistema desenvolvido no estudo de caso.
- **Objetivo da medição**: tendo como base as atuais funcionalidades do sistema, caracterizar:
    - A qualidade do sistema;
    - Os principais pontos fortes do sistema;
    - Os principais pontos fracos do sistema;
    - Sugestões para futuras melhorias.

7.2.2 PLANEJAMENTO

- Descrição da instrumentação:
    - **Dados pessoais**: 8 questões abertas sobre linha de pesquisa seguida pelo indivíduo, sua formação acadêmica etc.;
    - **Qualidade do sistema**: 10 questões fechadas baseadas no padrão ISO 9126 (ISO 1991), com respostas na escala de 0 a 3;
    - **Futuras versões**: 3 questões abertas sobre as funcionalidades que o indivíduo acha que deveriam ser adicionadas em futuras versões do sistema, assim como sugestões para melhorias específicas em cada módulo;
    - **Em relação a sistemas similares**: caso o indivíduo possua experiência prévia com algum sistema semelhante, 3 questões abertas comparando esse outro sistema com o GEIC;
    - **Geral**: 2 questões gerais a respeito do sistema, tais como os principais pontos positivos e negativos de toda a solução.

- **Seleção do contexto**: O estudo supõe o processo *off-line* porque os especialistas não estão sendo entrevistados durante todo seu uso do sistema, mas em certo instante. Os participantes são especialistas do domínio sem conhecimento prévio do sistema. O estudo é modelado porque são utilizadas notas subjetivas. O contexto possui um caráter geral, pois são feitas comparações com outros sistemas.
- **Seleção dos indivíduos**: 10 alunos de graduação em psicologia ou alunos de pós-graduação em psicologia e educação especial da UFSCar. Esses participantes não devem ter conhecimento prático prévio sobre o sistema, nem terem acompanhado a evolução do projeto de *software*, tanto de forma passiva como usuários do sistema quanto de forma mais ativa, participando nas reuniões de coleta de requisitos e sessões de teste.

7.2.3 MATERIAIS

Os materiais utilizados no experimento podem ser consultados no Anexo B. São eles:

- Termo de consentimento livre e esclarecido, onde o indivíduo aceita participar deste projeto de pesquisa;
- Treinamento sobre o domínio e sobre o sistema GEIC: foram exibidos alguns slides sobre o domínio do problema (programas de ensino individuais e equivalência de estímulos), assim como slides explicando o funcionamento geral do GEIC;
- Tarefa: realização de várias tarefas relacionadas ao sistema, tais como cadastrar alunos, criar e aplicar PEIs e conferir resultados;
- Questionário: 26 questões.

7.2.4 OPERAÇÃO

- **Treinamento**: Aos participantes foram exibidos slides sobre o domínio do problema, contendo tópicos como aprendizagem, paradigma de equivalência de estí-

mulos, PEIs e PSI. Foram exibidos também slides sobre o sistema, com explicações sobre meio de acesso, utilidade e funcionalidade de cada módulo. O treinamento teve duração aproximada de 45 minutos;

- **Tarefa**: Foi entregue a cada participante um enunciado com as atividades a serem desempenhadas durante a tarefa, e foi estipulado um tempo máximo de 60 minutos para que essa tarefa fosse concluída. Perguntas sobre as tarefas foram respondidas livremente.
- Ao final desse prazo os participantes foram convidados a responderem ao questionário.

7.2.5 RESULTADOS

Perfil dos participantes

Tabela 7 - Perfis dos participantes (especialistas ingênuos)

Participante	Curso	Período (semestre)	Linha de pesquisa
P1	1	6	-
P2	1	10	-
P3	1	10	-
P4	1	Concluído	-
P5	2	4	3
P6	3	2	1
P7	2	2	3
P8	2	4	2
P9	2	3	3
P10	2	-	-

Tabela 8 - Legenda (perfis dos participantes ingênuos)

Curso		Linha de pesquisa	
1	Graduação em Psicologia	**1**	Prevenção de deficiências
2	Mestrado em Psicologia	**2**	Análise comportamental da cognição
3	Mestrado em Educação Especial	**3**	Comportamento social e processos cognitivos

A partir da Tabela 8 é possível ver uma distribuição quase igual entre alunos de graduação (não seguem nenhuma linha de pesquisa) e pós-graduação (com três participantes seguindo a linha de pesquisa 3). Todos os participantes possuíam conhecimento prévio sobre análise do comportamento e equivalência de estímulos, já que esses conteúdos são ensinados durante o curso de graduação em psicologia.

Qualidade do sistema

Analisada a Figura 49 com a pontuação média nas questões sobre a qualidade do sistema, obtêm-se média 2,7 e mediana 2,82 (de um total de 3), o que é um resultado mais do que satisfatório e revela que em geral o sistema é bastante atrativo para especialistas de psicologia sem conhecimento prévio.

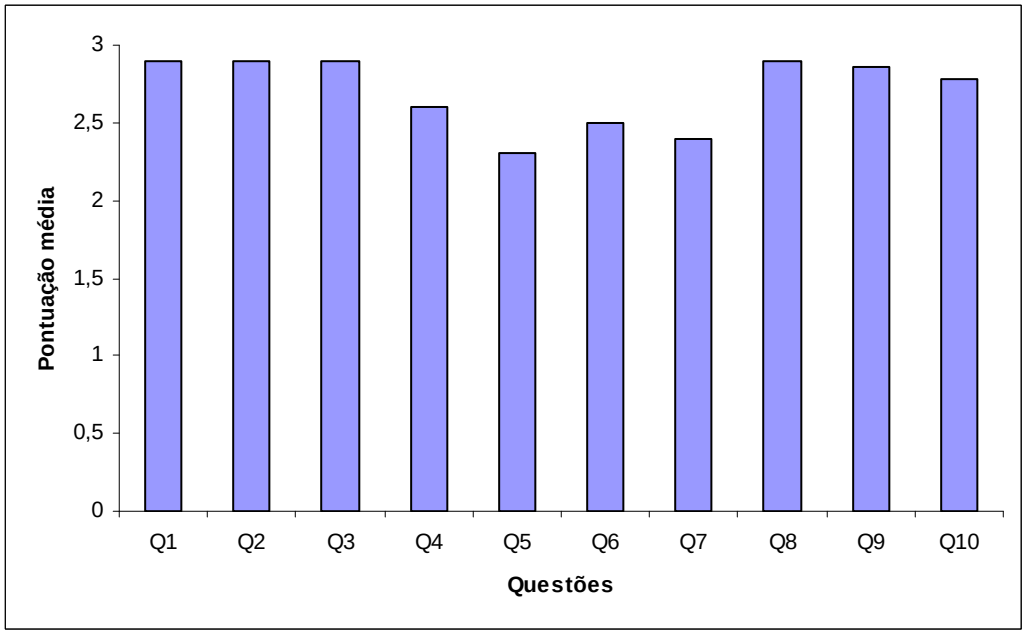


Figura 49 - Pontuação média nas questões sobre a qualidade do sistema (especialistas ingênuos)

Apesar de todas as questões terem uma média muito boa (acima de 2), vale novamente observar a média mínima a fim de se tentar detectar algum padrão de interesse. As questões com menor média foram Q5 e Q7, relacionadas respectivamente à facilidade de aprendizado do sistema e ao potencial de atrair novos usuários, o que pode ser um indício de que o sistema pode confundir novos usuários. Porém, isso pode se justificar pelo curto período de tempo a que os participantes tiveram para se familiarizar com o sistema, além da grande quantidade de tarefas exigidas para o término do experimento. Pode se justificar também pela grande quantidade de recursos que alguns módulos, tais como *autoria*, oferecem, o que a princípio pode ser intimidativo para um especialista não treinado. Estudos posteriores de usabilidade devem ser conduzidos a fim de procurar facilitar ainda mais o aprendizado da ferramenta.

Futuras versões

A Figura 50 aponta os principais interesses dos participantes no que se refere a novos tipos de estímulos a serem suportados pelo sistema. Como nem todos os participantes

eram totalmente familiarizados com o domínio foi natural observar que o tipo “vídeo” foi o mais requisitado, dado seu apelo visual, em detrimento de outros tipos menos chamativos e de talvez mais relevância para o ensino de leitura e escrita.

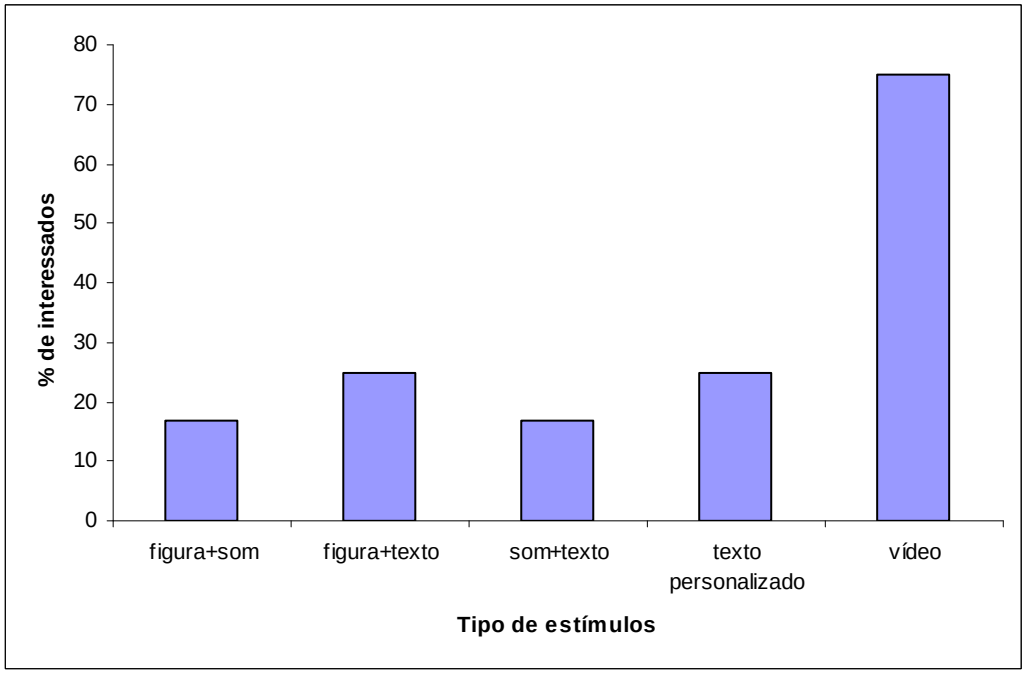


Figura 50 - Interesse em novos tipos de estímulos (especialistas ingênuos)

A Figura 51 aponta os principais interesses dos participantes no que se refere a novos tipos de tentativas. Os tipos “*go-no-go*” e “MTS com matriz de comparações” foram os mais requisitados. Uma causa que pode explicar a preferência pelo segundo tipo é que ele segue o modelo clássico apresentado por Sidman (1971) com uma matriz de estímulos de 3x3, um procedimento de emparelhamento com o modelo utilizado desde a década de 70.

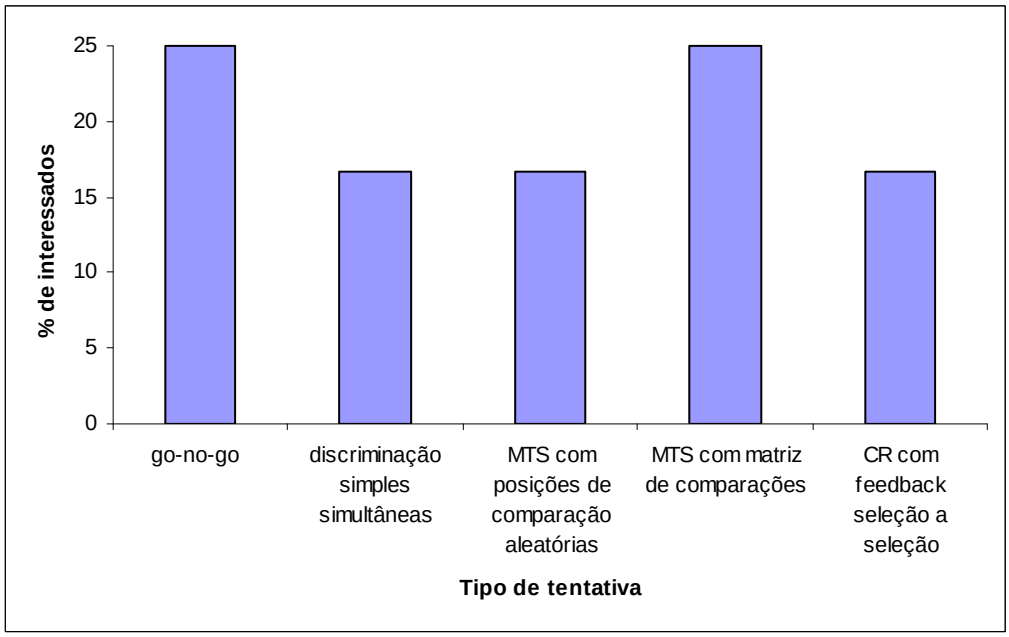


Figura 51 - Interesse em novos tipos de tentativas (especialistas ingênuos)

Em relação sistemas similares

Seis dos participantes possuíam experiência prévia com o sistema *ProgLeit*, já que os estudantes de psicologia têm contato com ele durante a graduação (Liga da Leitura). Em geral esses participantes relataram que a produtividade no GEIC em relação ao *ProgLeit* é maior, dada a quantidade de recursos para se salvar registros. Também que o GEIC oferece mais opções para a criação de PEIs, e que a interface é bem mais explicativa. Relataram ainda que, apenas do GEIC ser mais complexo, é também mais completo.

Geral

A Figura 52 ilustra os principais pontos positivos do sistema, que são os estímulos de som e figura padronizados, facilidade para se utilizar em vários locais, monitoramento remoto de alunos, facilidade para programar sessões, segurança e riqueza de opções.

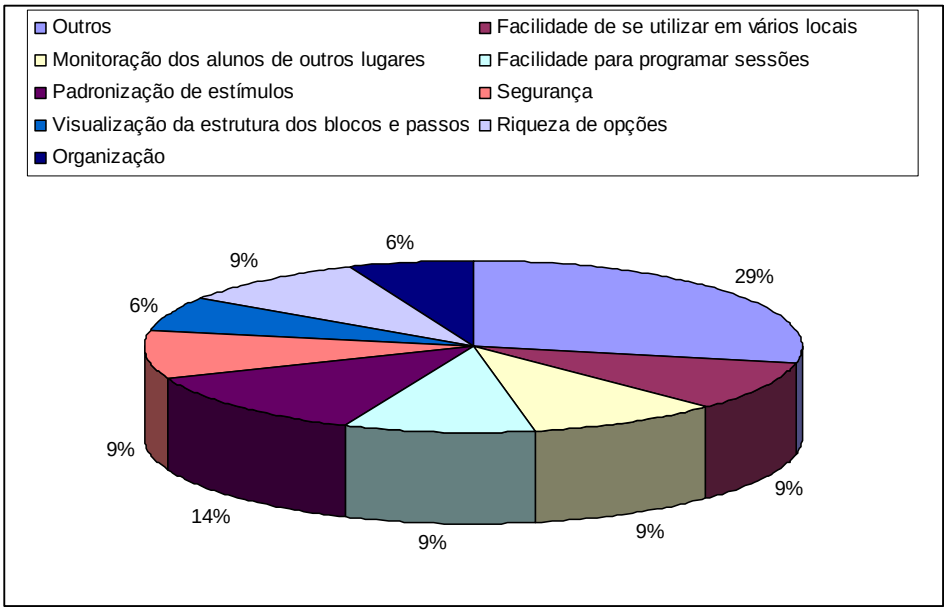


Figura 52 - Principais pontos positivos do sistema (especialistas ingênuos)

A Figura 53 se refere aos principais pontos negativos do sistema, que são a dificuldade para se aprender a utilizar, diversidade de opções (deixa o usuário confuso), acesso obrigatório à Internet, é fácil ocasionar erros em PEIs com muitos passos, depende de um único super-usuário para autorizar versões de PEIs.

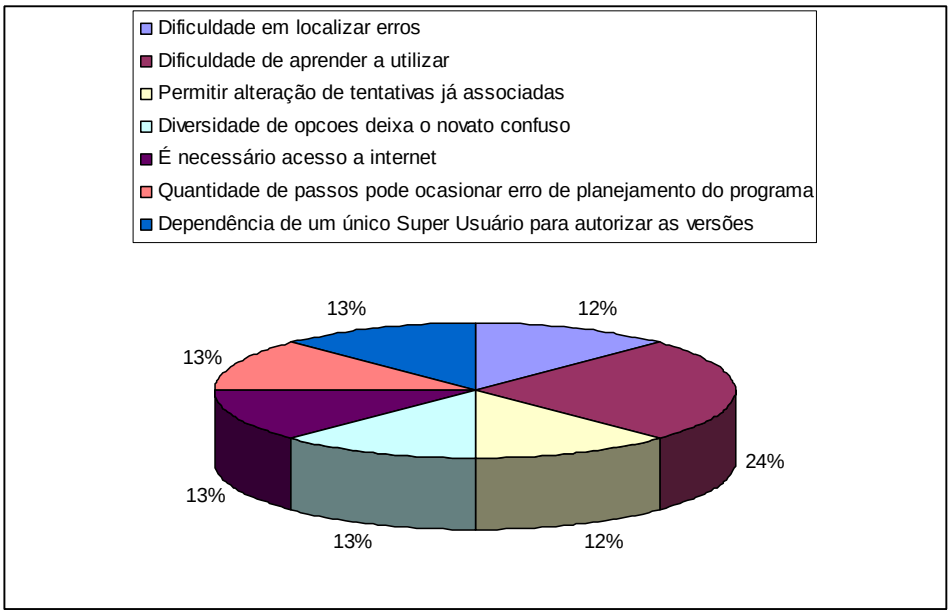


Figura 53 - Principais pontos negativos (especialistas ingênuos)

7.3 ESTUDO 3 – DESENVOLVEDORES DE *SOFTWARE* INGÊNUOS EM RELAÇÃO À INFRA-ESTRUTURA

7.3.1 DEFINIÇÃO DOS OBJETIVOS

- **Objetivo global**: Avaliar a qualidade da infra-estrutura, levantar seus pontos fortes e fracos e coletar sugestões para futuras melhorias. Tudo isso do ponto de vista de engenheiros e desenvolvedores de *software* sem contato prévio com a infra-estrutura;
- **Objetivo da medição**: Tendo como base as atuais funcionalidades da infra-estrutura, caracterizar:
    - A qualidade da infra-estrutura;
    - Os principais pontos fortes da infra-estrutura;
    - Os principais pontos fracos da infra-estrutura;
    - Sugestões para futuras melhorias.

7.3.2 PLANEJAMENTO

- **Descrição da instrumentação**
    - **Dados pessoais**: 15 questões abertas sobre linha de pesquisa seguida pelo indivíduo, sua formação acadêmica, experiência em desenvolvimento de *software* etc;
    - **Qualidade da infra-estrutura**: 18 questões fechadas com respostas na escala de 0 a 3;
    - **Em relação a infra-estruturas similares**: caso o indivíduo possua experiência prévia com alguma infra-estrutura semelhante, 3 questões abertas comparando essa solução e a apresentada neste trabalho;
    - **Geral**: 3 questões gerais a respeito da infra-estrutura, tais como os principais pontos positivos e negativos de toda a solução.

- **Seleção do contexto**: O estudo supõe o processo *off-line* porque os especialistas não estão sendo entrevistados durante todo seu uso da infra-estrutura, mas em certo instante. Os participantes são especialistas do domínio sem conhecimento prévio da infra-estrutura. O estudo é modelado porque são utilizadas notas subjetivas. O contexto possui um caráter geral, pois são feitas comparações com outras infra-estruturas.
- **Seleção dos indivíduos**: Dez alunos de graduação em engenharia/ciência da computação ou alunos de pós-graduação em computação da UFSCar. Esses participantes não devem ter conhecimento prático prévio sobre a infra-estrutura, nem terem acompanhado a evolução do projeto de forma ativa.

7.3.3 MATERIAIS

Os materiais utilizados no experimento podem ser consultados no Anexo C. São eles:

- Termo de consentimento livre e esclarecido, onde o indivíduo aceita participar deste projeto de pesquisa;
- Treinamento sobre o domínio, sobre o sistema GEIC e sobre a infra-estrutura: foram exibidos alguns slides sobre o domínio do problema (programas de ensino individuais e equivalência de estímulos), slides explicando o funcionamento geral do GEIC e, finalmente, slides sobre a infra-estrutura da solução;
- Tarefa: realização de várias tarefas relacionadas à infra-estrutura, tais como implantar uma versão do GEIC, alterar unidades de negócio, criar um novo módulo etc.;
- Questionário: 39 questões.

7.3.4 OPERAÇÃO

- **Treinamento**: Aos participantes foram exibidos slides sobre o domínio do problema, contendo tópicos como aprendizado, paradigma de equivalência de estímulos, PEIs e PSI. Foram exibidos também slides sobre o sistema, com explicações sobre meio de acesso, utilidade e funcionalidade de cada módulo. Finalmente, foram exibidos slides sobre a infra-estrutura proposta, incluindo organização dos servidores, processo, padrões etc. O treinamento teve duração aproximada de 50 minutos;
- **Tarefa**: Foi entregue a cada participante um enunciado com as atividades a serem desempenhadas durante a tarefa, e foi estipulado um tempo máximo de 60 minutos para que essa tarefa fosse concluída. Perguntas sobre as tarefas foram respondidas apenas quando o conceito em questão era indispensável para a continuidade da atividade, ou quando mais de um participante possuía a mesma dúvida;
- Ao final desse prazo os participantes foram convidados a responderem ao questionário.

7.3.5 RESULTADOS

Perfil dos participantes

Tabela 9 - Perfis dos desenvolvedores

Participante	Curso	Período (semestre)	Linha de pesquisa
P1	1	4	-
P2	2	8	1
P3	4	1	2
P4	3	4	-
P5	3	Concluído	3
P6	1	4	1
P7	3	2	-
P8	2	6	-
P9	3	Concluído	4
P10	3	Concluído	5

Tabela 10 - Legenda (perfis dos desenvolvedores)

Curso		Linha de pesquisa	
1	Bacharelado em Ciência da Computação	**1**	TV Digital
2	Engenharia da Computação	**2**	Ontologias Fuzzy e recuperação da informação
3	Mestrado em Ciência da Computação	**3**	Inteligência artificial
4	Doutorado em Ciência da Computação	**4**	Engenharia de software
		5	Segurança em ambientes de ensino colaborativo

A partir da Tabela 9 é possível ver uma distribuição equilibrada entre alunos de graduação e mestrado, com apenas um aluno de doutorado. As linhas de pesquisa também são distribuídas, com uma ligeira superioridade de alunos que trabalham com TV Digital Interativa (LINCE).

Observa-se a partir da Figura 54 que os participantes possuem bastante experiência com desenvolvimento de *software* dentro da universidade (sozinhos e em grupos), porém poucos possuem alguma experiência em empresas, seja programando sozinho ou em grupo.

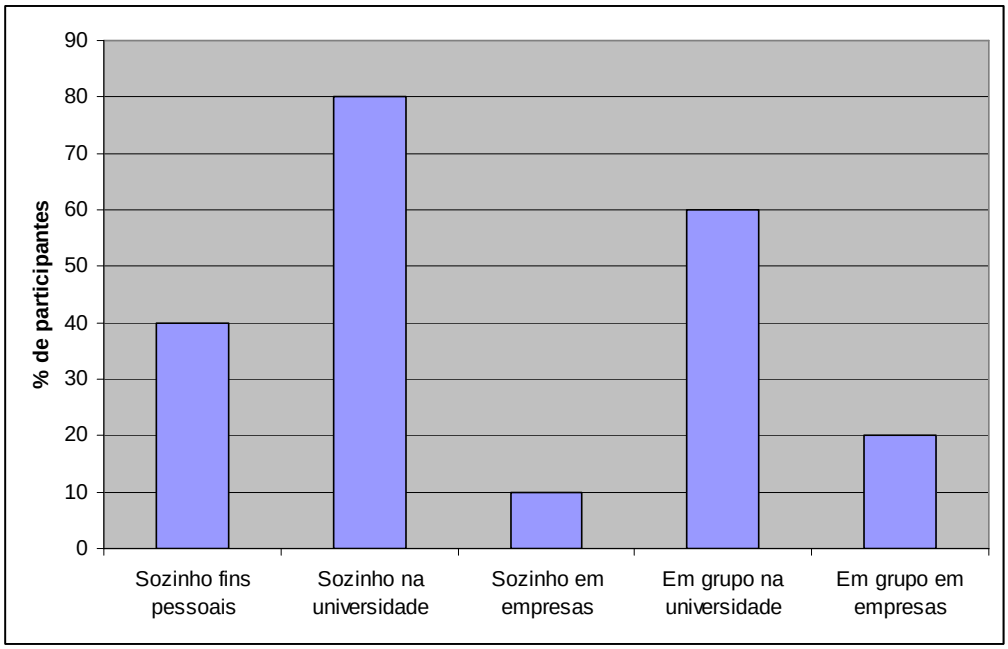


Figura 54 - Experiência em desenvolvimento de software

Analisando-se os dados da Figura 55 nota-se que, talvez como conseqüência da pouca experiência no mercado de trabalho, os participantes possuem pouca experiência com

ferramentas normalmente utilizadas nesse meio. Um exemplo notável é o de integração contínua, com o qual nenhum participante possuía experiência.

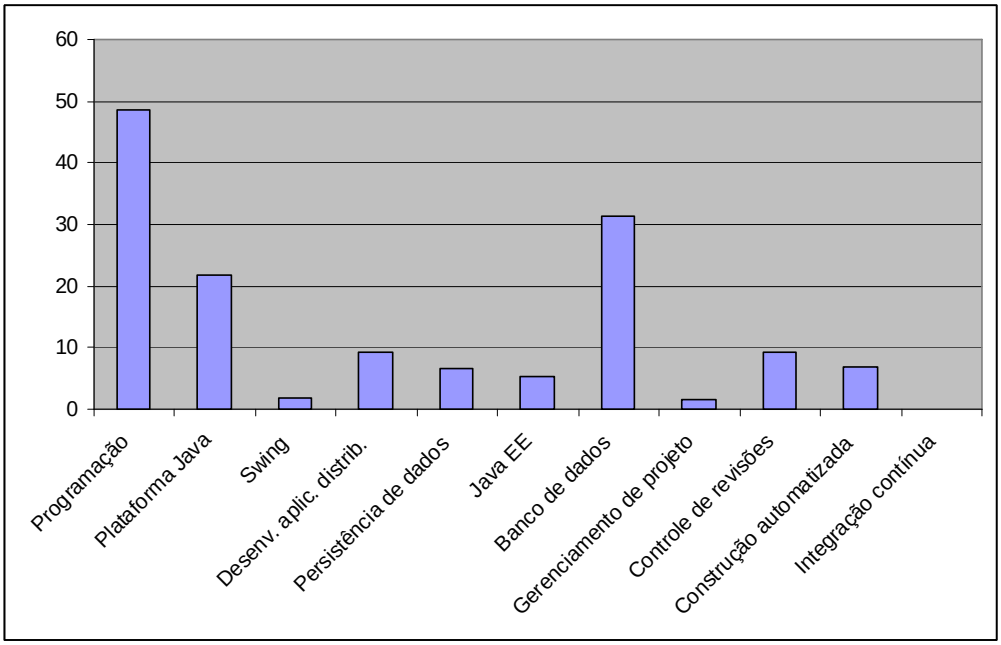


Figura 55 - Experiência em ferramentas de software específicas

Qualidade da infra-estrutura

Analisada a Figura 56 com a pontuação média nas questões sobre a qualidade da infra-estrutura, obtêm-se média 2,66 e mediana 2,68 (de um total de 3), o que é satisfatório.

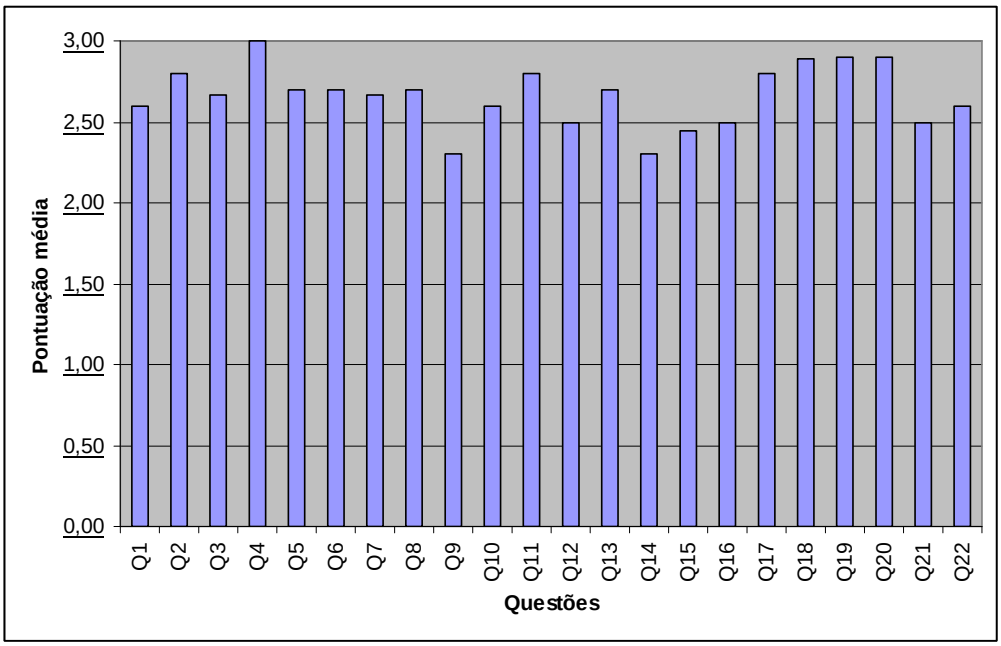


Figura 56 - Pontuação média nas questões sobre a qualidade do sistema (desenvolvedores)

As questões com menor média foram Q9 e Q14, relacionadas respectivamente à facilidade de desenvolvimento e facilidade de modificação, o que pode ser conseqüência da pouca experiência dos participantes no mercado de trabalho. Outro fator que pode ter contribuído para esse resultado é o curto período de tempo a que os participantes tiveram para se familiarizar com a infra-estrutura, além da grande quantidade de tarefas exigidas para o término do experimento. Pode se justificar também pela grande quantidade de ferramentas aprendidas e utilizadas em um período muito curto de tempo.

Em relação a infra-estruturas similares

Três participantes possuem experiência com o *Sakai*/*Tidia-Ae*, já que esse ambiente é utilizado para fins de desenvolvimento no LINCE. Dois dos participantes relataram que sua produtividade com a infra-estrutura apresentada tem potencial para ser melhor do que com o *Sakai*, em boa parte devido às ferramentas de automatização, fácil criação de módulos e integração contínua. Um dos participantes possui mais familiaridade com o *Sakai* e se sente mais confortável com ele. Um participante possui experiência com *Moodle* e o acha muito semelhante à infra-estrutura proposta. Outro participante possui experiência com a plataforma Eclipse, mas não fez mais comentários.

Geral

A questão explorada na Figura 57 se refere aos principais pontos positivos da infra-estrutura, que são a política de controle de versões, organização geral, automatização, mecanismos de testes, documentação e usabilidade.

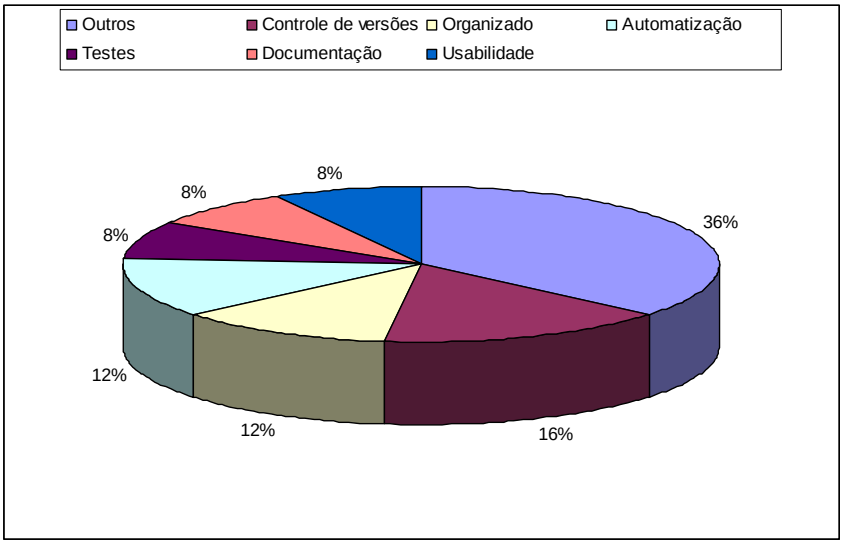


Figura 57 - Principais pontos positivos da infra-estrutura

Já a questão explorada na Figura 58 se refere aos principais pontos negativos do sistema, que são a existência de muitos servidores, exigência de conhecimento de muitas tecnologias, falta de um idioma padronizado (muitos termos em português e inglês usados alternadamente nas interfaces de usuário, documentação etc.) e falta de uma interface mais amigável para a realização de algumas tarefas administrativas.

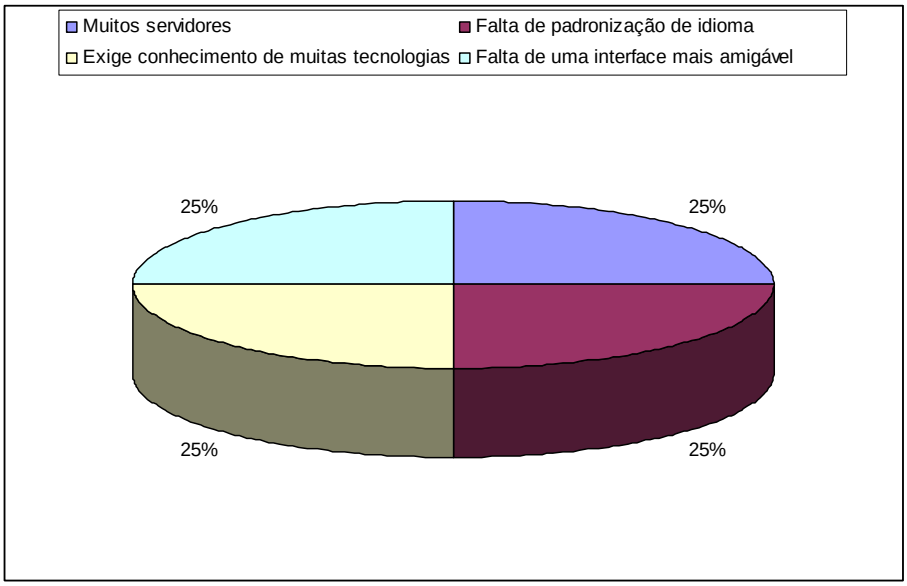


Figura 58 - Principais pontos negativos da infra-estrutura

7.4 ESTUDO 4 – AVALIAÇÃO SUBJETIVA

Neste estudo foi realizada uma avaliação subjetiva a respeito dos principais conceitos e técnicas utilizados para o desenvolvimento deste trabalho. Os elementos avaliados incluem padrões arquiteturais, disciplinas de gerenciamento de projeto, plataformas de *hardware*, plataformas de desenvolvimento e bibliotecas de *software*.

Esta avaliação é anedótica e, apesar de não contar com o mesmo nível de formalismo presentes nos outros três estudos, proporciona uma visão mais prática a respeito da evolução do projeto e das vantagens e desvantagens de cada escolha realizada. Muitas das observações são difíceis de serem obtidas a partir de outro tipo de estudo, já que levam em conta a experiência acumulada durante todo o desenvolvimento do trabalho.

7.4.1 ANÁLISE DE REQUISITOS

A coleta e análise dos requisitos foram realizadas tanto no início do projeto quanto no início de cada ciclo de desenvolvimento. Devido à natureza multidisciplinar do trabalho, que contou com o envolvimento de disciplinas como ciência da computação e psicologia, esse início poderia ter sido mais dinâmico caso uma maior integração entre os especialistas de cada área tivesse ocorrido.

Também no início de cada ciclo de desenvolvimento, cada um com uma duração aproximada entre 30 e 60 dias, o projeto como um todo teria se beneficiado de uma proximidade maior entre os diversos especialistas. Apesar disso, a escolha de ciclos de curta duração facilitou a alteração de eventuais requisitos incompletos ou mal especificados.

O uso mais intenso de práticas de desenvolvimento ágil, tais como desenvolvimento orientado a comportamento (*Behavior Drivem Development* – BDD) poderia ter aliviado alguns dos problemas reportados. BDD (NORTH, 2006) encoraja uma maior colaboração durante todo o processo de desenvolvimento entre desenvolvedores, analistas de qualidade, especialistas do domínio e não-técnicos.

De maneira similar, técnicas de programação extrema (*eXtreme Programming* – XP) (BECK, 1999), tais como “time coeso” (*whole team*) teriam ajudado na colaboração entre equipe de desenvolvimento e clientes. Nesta prática o “cliente” não apenas requisita os serviços e realiza os pagamentos, mas também se mostra disponível a todo instante para responder dúvidas. Se possível, o cliente (ou um representante com conhecimento sobre o domínio) deve estar presente fisicamente no local onde se encontram os programadores.

7.4.2 GERENCIAMENTO DE PROJETO

O uso disciplinado de algumas disciplinas de gerenciamento de projeto foi fundamental para manter organizada a equipe de desenvolvimento (dois desenvolvedores de software e uma quantidade variável de psicólogos e pesquisadores) envolvida neste trabalho. Além disso, será fundamental no futuro próximo para organizar e acomodar a demanda por novas funcionalidades, o suporte a novas plataformas e o envolvimento de novos engenheiros de *software* e programadores.

O controle de revisões, realizado através da ferramenta *Subversion*, permitiu o desenvolvimento em paralelo dos vários módulos do sistema. Permitiu também o registro de todas as alterações realizadas no repositório de código-fonte, possibilitando entre outras coisas armazenamento confiável de dados e maior rastreabilidade de alterações. A opção por um controle de revisões centralizado foi adequada, visto o tamanho reduzido da equipe de desenvolvimento e da proximidade geográfica entre os membros da equipe. Porém, conforme novos engenheiros de *software* e programadores forem se envolvendo no projeto, talvez surja a demanda por um controle de revisões distribuído através de ferramentas como *Git* ou *Mercurial*.

O gerenciamento de tarefas, realizado através da ferramenta *Trac*, permitiu a organização das tarefas, melhorias e defeitos a serem trabalhados, o que facilitou bastante o trabalho dos programadores. Permitiu também uma definição de metas bastante precisa, com recursos de linha do tempo do projeto (*timeline*) e gerenciamento de *milestones*.

Além das funcionalidades disponíveis nativamente no *Trac*, seus recursos de integração com outras ferramentas também foram fundamentais durante o decorrer do projeto. Exemplos incluem a integração com o controle de versões (*Subversion*), permitindo assim a associação entre tarefas e revisões de *software*, e a integração com o servidor de integração contínua (*Hudson*), permitindo assim a associação entre tarefas e revisões de *software* a um *build* automatizado.

A integração contínua, prática ainda desconhecida por parte dos engenheiros de *software* e programadores, foi realizada com o auxílio da ferramenta *Hudson*. Seu uso permitiu à equipe localizar erros de integração de *software* com bastante antecedência, e será ainda mais útil à medida que mais programadores passarem a modificar o código-fonte simultaneamente.

O conjunto de ferramentas utilizado (controle de revisões, gerenciamento de tarefas e integração contínua) é composto exclusivamente por *software* livre, o que não gerou encargos financeiros adicionais à equipe. Mais ainda, a integração entre elas é muito boa, o que fez com que o ambiente de desenvolvimento se tornasse bastante produtivo e controlado.

A opção por uso exclusivo de *software* livre nas ferramentas de gerenciamento de projeto se mostrou acertada, porém há alternativas proprietárias que oferecem soluções integradas bastante convidativas. Exemplos incluem *Rational Team Concert* e *JIRA*.

7.4.3 ARQUITETURA DE *SOFTWARE*

O uso de uma arquitetura de *software* personalizada para o domínio de aprendizado permitiu o desenvolvimento de *software* compatível com os requisitos especificados. Mais importante ainda, o uso de padrões arquiteturais consagrados como SOA e MVC permitiram acesso a uma vasta base de conhecimento sobre como organizar sistemas de grande porte.

Para realizar uma arquitetura orientada a serviços, foram utilizados serviços *Web* baseados nos protocolos WSDL e SOAP. Esses protocolos, suportados em praticamente todos os servidor de aplicação atuais e presentes em forma de bibliotecas em todas as linguagens de programação comerciais, permitiram prover real interoperabilidade entre diversos tipos de dispositivos e plataformas de desenvolvimento.

Porém, o protocolo SOAP, que é baseado em XML, se mostrou um limitante em alguns cenários de uso da rede de comunicação. A sobrecarga envolvida na codificação e decodificação de objetos para XML, associada à complexidade do protocolo, exigiram cuidado adicional por parte dos programadores a fim de evitar problemas com alta latência e baixa vazão. Alternativas mais leves, porém menos expressivas, existem na forma de arquiteturas *RESTful* (*Representational State Transfer*) (FIELDING, 2000) e de formatos de troca de informação como JSON (*Javascript Object Notation*) (CROCKFORD, 2006).

A infra-estrutura de servidores dos ambientes de produção e desenvolvimento, apesar de ter sido projetada de maneira a suprir as presentes e futuras necessidades dos sistemas desenvolvidos, se mostrou de difícil execução na prática, principalmente no que se refere à alocação de recursos físicos e financeiros.

A falta de um espaço físico dedicado na universidade, assim como limitações nas redes de comunicação e de energia, contribui para que o sistema desenvolvido não possua uma disponibilidade (*availability*) ótima. A solução para esse problema seria um espaço físico reservado para os futuros desenvolvimentos do sistema, o que se justifica pela grande quantidade de pesquisadores e alunos que vem sendo beneficiados com a solução. Já a maior disponibilidade de recursos financeiros possibilitaria a compra de unidades suficientes de computadores para funcionarem como servidores dedicados, além da aquisição de equipamentos que auxiliam no aumento da disponibilidade do serviço, tais como *no-breaks* e *racks* apropriados.

7.4.4 PROJETO

O projeto da infra-estrutura, apoiado pelo uso de padrões consolidados como *Transfer Object* e *Façade*, facilitou a codificação de módulos de *software* adequados para a realização das tarefas especificadas. Facilitou também a comunicação dentro da equipe de desenvolvimento, visto que os programadores passaram a contar com um vocabulário bem definido de padrões para realizarem as soluções de *software* e para documentarem todo o processo.

O modelo de domínio, estruturado de maneira “anêmica”, mostrou-se ideal para a representação das entidades e das unidades de negócio. Apesar de domínios anêmicos não serem considerados “boa prática“ por alguns especialistas (já que o modelo de domínio pode se tornar apenas um conjunto de estruturas de dados, e não classes com dados e comportamentos), seu uso na infra-estrutura proposta correu sem problemas e facilitou bastante o trabalho dos programadores.

A separação dos sistemas em módulos, formulados a partir de um conjunto específico de tarefas, foi indispensável para acelerar e facilitar os processos de especificação e desenvolvimento, teste e manutenção. Assim, mudanças em um módulo específico geralmente foram feitas pelo programador com melhor conhecimento daquelas funcionalidades, e os testes e implantação desse módulo puderam ser feitos sem influenciar os restantes.

Os módulos-cliente, projetados para serem aplicações *desktop* executadas a partir de um navegador, mostraram-se bastante eficientes e garantiram a satisfação dos requisitos de precisão de tempo exigidos no domínio de ensino. Caso tivessem sido projetados como aplicações *Web* utilizando AJAX, todo o processo de desenvolvimento teria sido prejudicado por conta de incompatibilidades com navegadores *Web* e imaturidade de bibliotecas de *software*. Talvez nos anos seguintes, com navegadores *Web* mais confiáveis e rápidos, novos padrões como *HTML5* e *CSS3*, além de bibliotecas de *software* mais maduras para o desenvolvimento de aplicações *AJAX*, o desenvolvimento dos módulos desta infra-estrutura possam ser projetos para serem exclusivamente aplicações *Web*.

7.4.5 TECNOLOGIAS

A escolha correta das tecnologias permitiu a realização do projeto da infra-estrutura e deu origem ao sistema GEIC. O conjunto dessas tecnologias é bastante extenso, dada a amplitude da proposta e a quantidade de dispositivos e plataformas de *software* suportadas. O uso de algumas dessas tecnologias é discutido a seguir.

A plataforma *Java* foi escolhida para o desenvolvimento dos módulos-servidor e da maioria dos módulos-cliente. Essa escolha facilitou imensamente a realização da interoperabilidade entre os diversos módulos, visto que a plataforma *Java* é suportada tanto em *PCs* quanto em dispositivos móveis, TV Digital etc. Porém, por já estar consolidada no mercado há vários anos e por possuir uma preocupação especial com compatibilidade entre versões anteriores, a plataforma *Java* em geral (linguagem de programação, bibliotecas, máquina virtual etc.) se mostrou em alguns momentos demasiado complexa e lenta.

No início do trabalho foram estudadas alternativas para a plataforma *Java*, tais como as linguagens de programação *C++* (eficiente e previsível, porém muito complexa) e *Python* (elegante e moderna, porém lenta e pouco adotada). Porém, optou-se por prosseguir utilizando *Java*, dada sua popularidade e farta documentação.

Como conseqüência da escolha pela plataforma *Java*, optou-se por utilizar a plataforma empresarial (*enterprise*) *Java EE 5*. Essa última versão da especificação *Java EE* melhora bastante em relação a sua predecessora, tanto em termos de simplicidade (uso de anotações, redução drástica da quantidade de arquivos *XML* exigida etc.) quanto em termos de recursos (*EJB3*, *JPA*, *JMS* etc.).

A codificação de referência do *Java EE 5*, conhecido como *Glassfish*, se mostrou a princípio instável, ineficiente e com uma quantidade de defeitos altíssima. Porém, até o término do projeto várias correções foram liberadas para corrigir várias das falhas reportadas, o que o tornou um produto mais aceitável do ponto de vista empresarial. Infelizmente, até então *Glassfish* era o único servidor certificado para *Java EE 5*, o que impediu a adoção de alternativas. Atualmente, entretanto, há outras opções disponíveis, tais como *JBoss 5*.

Para facilitar o processo de persistência de dados utilizou-se a API para persistência da plataforma *Java* (*Java Persistence API* – JPA) junto ao provedor de persistência *ToplinkEssentials*. O uso de JPA simplificou bastante a tarefa de codificação das unidades de negócio e das entidades, uma vez que se tornou desnecessário interagir diretamente com o

banco de dados. Além disso, a estrutura do banco de dados passou a ser gerada e atualizada automaticamente a partir do modelo de classes, o que garantiu ainda mais liberdade para a codificação ágil dos requisitos. Porém, o uso de uma API de persistência se tornou problemática em situações onde o desempenho era essencial, o que exigiu que comandos em SQL fossem usados em lugares estratégicos para sobrepor algumas funcionalidades do JPA.

Em relação ao SBTVD, foi desenvolvido um protótipo do *player* a partir do uso das linguagens NCL e Lua e do *middleware* Ginga. Devido ao fato dos programadores não disporem ainda de uma versão final do *middleware*, além de não possuírem acesso a antenas transmissoras de sinal digital, foi utilizado um simulador durante a maior parte do tempo.

A versão preliminar do *middleware* se mostrou bastante instável, com erros e travamentos freqüentes, bibliotecas de *software* deficitárias e falta de alguns recursos básicos que inviabilizaram o desenvolvimento de um *software* funcional e pronto para implantação. Além disso, os programadores responsáveis pelo *middleware* se mostraram incomunicáveis, impedindo assim que correções e novas funcionalidades fossem reportadas/sugeridas. A documentação disponível para a comunidade de programadores em geral se mostrou também bastante distante do ideal, com informação descentralizada, instruções confusas e falta de profundidade na abordagem de alguns tópicos.

Além dos problemas técnicos e gerenciais relacionados ao SBTVD, sua utilização como meio de difusão de programas de ensino só se tornará prático uma vez que parcerias com empresas de telecomunicação forem realizadas. Isso porque o custo dos equipamentos necessários para se efetuar transmissões digitais é bastante alto, e atualmente só pode ser arcado por grandes empresas ou canais estatais. Assim, um plano de cooperação em longo prazo deve ser estabelecido caso se deseje utilizar efetivamente o SBTVD como ferramenta de auxílio ao ensino.

Finalmente, foi desenvolvido um *player* para dispositivos móveis baseado na tecnologia *Java ME*. A tecnologia se mostrou bastante adequada para o desenvolvimento de aplicativos multimídia (como os exigidos pelo domínio), porém a fidelidade com que essa tecnologia é suportada pelos diferentes dispositivos varia largamente, tornando uma tarefa bastante complexa utilizar o mesmo código-fonte para execução em várias plataformas. Além dessas dificuldades em relação à plataforma de desenvolvimento, desenvolver *software* para dispositivos tão limitados como telefones celulares é um enorme desafio, dadas suas restrições severas de processamento e memória.

8 TRABALHOS CORRELATOS

Várias pesquisas tiveram como objetivo informatizar o processo de instrução personalizada, porém até então o foco havia sido em desenvolver sistemas de *software* (ferramentas) para auxiliar nesse processo, e não em desenvolver uma infra-estrutura que viabilizasse a construção e uso desses sistemas de forma colaborativa e visando o reúso. Assim, serão avaliados trabalhos relacionados apenas parcialmente à proposta aqui apresentada.

8.1 CAPSI E *WEBCAPSI*

O sistema personalizado de instrução auxiliado por computador CAPSI (*Computer-Aided Personalized System of Instruction*) e sua versão *Web WebCAPSI*[47] (PEAR; KINSNER, 1988; PEAR; NOVAK, 1996; PEAR; CRONE-TODD, 1999) são sistemas onde são disponibilizadas unidades de ensino (cursos) na forma de materiais de estudos e testes. Cada aluno pode manter seu próprio ritmo, e as respostas aos testes são avaliadas por revisores ou tutores.

O sistema *WebCAPSI* possui diferenças fundamentais em relação à infra-estrutura proposta neste trabalho, a maioria delas relacionada ao contexto em que cada um foi desenvolvido. Por ter surgido em uma universidade e ter sido aplicado inicialmente em alunos universitários, os cursos oferecidos visavam o ensino de habilidades mais elaboradas, tais como conhecimento de conceitos de psicologia. Além disso, há disponibilização de material de estudo, possibilidade de avaliação de alunos por alunos mais adiantados nos cursos etc. Já a infra-estrutura proposta surgiu no contexto de ensino de habilidades pré-lingüísticas a crianças com dificuldades de ensino, o que inviabiliza a disponibilização de material de estudos e o *peer-review*. Aqui, o foco está no processo de aplicação de testes e avaliação.

Em relação ao processo de avaliação, no *WebCAPSI* as respostas aos testes são enviadas aos tutores em conjuntos, e os tutores as avaliam assincronamente, ou seja, não necessariamente no exato momento em que as respostas foram enviadas. Já no ambiente

[47] http://www.webcapsi.com

proposto, são favorecidas aplicações onde o *feedback* às respostas é proporcionado imediatamente, tanto pelo arranjo do PEI quanto pelo tutor presencial/remoto.

8.2 APRENDENDO A LER E A ESCREVER EM PEQUENOS PASSOS

O sistema *Aprendendo a Ler e a Escrever em Pequenos Passos* (ALEPP) (SOUZA et al., 2009), embora tenha sido amplamente empregado (e em alguns centros de pesquisa ainda é) com um grande contingente de alunos e permitido o desenvolvimento de pesquisas que fundamentam os PEs, atende poucos dos requisitos descritos ao longo deste trabalho. Desenvolvido por volta de 1997, é uma aplicação *stand-alone* que realiza a única tarefa de apresentar PEs e registrar as interações dos alunos. Com o passar do tempo, tornou-se pouco eficiente para atender a todas as demandas dos especialistas do domínio, especialmente no que se refere às possibilidades de difusão e acessibilidade.

Para criar um PE que possa ser interpretado e apresentado pelo ALEPP, os especialistas do domínio precisam editar de forma manual, um conjunto complexo de pastas e arquivos com nomes e funções bem definidas. Além disso, antes do sistema ser usado ele deve ser instalado no computador do usuário para então poder interpretar e executar os arquivos que contêm informações sobre o PE, o que implica muitas vezes em transportar cuidadosamente o conteúdo a ser ensinado e copiá-lo para outros computadores que já possuam o ALEPP instalado. Essa abordagem acarreta uma série de restrições e problemas, tais como redundância, descentralização de informações, dificuldade na coleta e interpretação de resultados, possibilidade de adulterações etc.

Embora tenha desempenhado um importante papel auxiliar nas pesquisas sobre desenvolvimento de PEs, tornou-se obsoleto e pouco eficiente para atender a todas as demandas dos especialistas do domínio, especialmente no que se refere às possibilidades de difusão e acessibilidade.

8.3 SISTEMAS DE GERENCIAMENTO DE APRENDIZAGEM

Um *Sistema de Gerenciamento de Aprendizagem* (*Learning Management System* – LMS) é composto de ferramentas que possibilitam especialistas a gerenciar o processo que circunda a aprendizagem. Alguns desses processos incluem gerenciamento de usuários (alunos, tutores e membros da equipe), calendários de cursos, fóruns de discussões e geração de currículos e portfólios.

Outra classe de sistemas conhecida como *Sistema de Gerenciamento de Conteúdo de Aprendizagem* (*Learning Content Management System* – LCMS) é composta por ferramentas que gerenciam especificamente a criação, reutilização e aplicação de pequenas unidades de ensino chamadas objetos de aprendizagem (*Learning Objects* – LOs ou ainda *Learning Content Ob-jects* – LCOs).

Na literatura há discordâncias no que se refere à classificação do que é um LMS ou LCMS. Para definições de LMS e LCMS, assim como uma discussão detalhada do contexto em que esses sistemas surgiram, consulte (WATSON; WATSON, 2007).

Apesar de haver um grande número de LMSs e LCMSs disponíveis, em geral eles foram concebidos com o objetivo de serem genéricos, não se limitando a um paradigma de ensino em particular. Por esse motivo, é muito difícil codificar os requisitos específicos à aplicação de PEIs usando esses sistemas. Porém, devido a algumas semelhanças com a proposta desta dissertação, serão avaliados brevemente os dois LMSs mais populares, *Moodle* e *Sakai*.

8.3.1 MOODLE

Moodle (MOODLE, 2009) é um LMS *Web* que pode ser usado por educadores para criar *si-tes* de aprendizagem. Codificado usando a linguagem de script PHP, pode ser executado na maioria das plataformas de *hardware* e sistemas de bancos de dados disponíveis no mercado. Sua natureza modular é suficientemente flexível, o que permite que a partir dele sejam construídas comunidades de ensino colaborativo. É possível ainda adicionar funcionalidades diretamente ao seu código-fonte e vendê-las ou distribuí-las.

Em relação ao módulo de aplicação de testes/avaliação disponível no ambiente *Moddle*, chamado *Quiz*, seu principal ponto fraco é a falta de precisão ao se registrar o tempo do comportamento dos alunos. Enquanto nesse módulo a precisão de registro é na casa de segundos, no ambiente proposto nesta dissertação a precisão é de milissegundos, o que permite a criação de relatórios de análise de dados mais ricos e detalhados. Uma maior precisão no registro do tempo permite também o acompanhamento de uma sessão de aprendizado por um tutor remoto, o que seria impossível caso os dados fossem atualizados a cada segundo.

8.3.2 SAKAI CLE

Sakai CLE (*Collaboration and Learning Environment*) é um LMS *Web* que gerencia colaborações e cursos e que provê aos usuários ferramentas projetadas para assistir tutores, pesquisadores e alunos. Exemplos de ferramentas incluem calendário, apresentação de *slides*, notificações e anúncios, fóruns e gerenciamento de recursos, testes de alunos etc. É baseado na plataforma *Java* e possui suporte nativo a orientação a serviços.

O principal problema em relação ao *Sakai CLE* é sua arquitetura, que ao contrário do que acontece com o *Moodle*, é extremamente complexa, especialmente no que se refere à sua separação das camadas de apresentação e de ferramentas. Essa complexidade faz com que a tarefa de se criar uma eventual ferramenta seja extremamente custosa.

A ferramenta de testes de alunos do *Sakai CLE*, chamada *SAMIGO*, se mostrou tão instável e apresentou tantos erros e exceções de *software* que tornou inviável sua avaliação mais detalhada. Conseqüentemente, a ferramenta no estado em que foi testada não é adequada para uso em ambientes reais, tais como escolas com centenas de alunos. Além disso, essa ferramenta não possui métodos documentados para se criar testes contendo imagens, sons ou vídeos (apesar de ser compatível com padrões para interoperabilidade de testes, tais como QTI – *Question and Test Interoperability*).

9 CONSIDERAÇÕES FINAIS

Neste trabalho foram exploradas e avaliadas técnicas de engenharia de *software* e computação distribuída na busca de uma solução que pudesse proporcionar a redução de custos envolvidos na criação, aplicação e gerenciamento de programas de ensino individualizados. Foi proposta também uma infra-estrutura de software que viabilizou a realização desses objetivos, assim como a construção de um sistema que permitiu oferecer uma codificação operacional e robusta para auxiliar no processo de ensino individualizado por computador.

9.1 TRABALHOS DECORRENTES

A infra-estrutura desenvolvida é flexível e de fácil uso até mesmo para desenvolvedores de *software* com pouca experiência, o que deve permitir a criação de um ecossistema de módulos de software. Com isso, aumentará também a oferta de recursos voltados para especialistas das áreas relacionadas ao ensino.

Alguns trabalhos acadêmicos já utilizam a infra-estrutura para prover facilidades adicionais aos especialistas de domínio. Bela (2009) desenvolveu uma metodologia para a utilização do conhecimento de domínio expresso em ontologias para a geração de anotações semânticas como meio de enriquecer e melhorar dados brutos, sendo que os dados armazenados pelo GEIC são de grande utilidade para a compreensão dos comportamentos apresentados por alunos. No futuro espera-se incorporar este trabalho, tanto na forma de um novo módulo-cliente responsável pela exibição de resultados quanto na forma de um módulo-servidor responsável por efetuar a mineração de dados diretamente a partir de base de dados.

Esforços para se criar módulos com interfaces de usuário mais amigáveis e interativas já se iniciaram. Já estão em andamento planos para se criar um módulo para a execução de sessões de ensino (*players*) de maneira análoga ao que já acontece com os módulos existentes, porém com a utilização de uma interface rica e conceitos de teoria de jogos. O principal objetivo será fornecer uma interface mais amigável a alunos, e comparar seu desempenho acadêmico com o de alunos que utilizaram os métodos tradicionais de execução de sessão.

Há várias possibilidades para se aumentar a quantidade de recursos disponíveis aos especialistas de domínio, principalmente no que se refere à criação de PEIs. Atualmente as unidades de ensino são organizadas de acordo com o modelo de máquina de estados finita determinística (MEFdet), ou seja, as possibilidades de transição de unidade para unidade são sabidas com antecedência. Contudo, uma camada de aleatoriedade poderia ser adicionada ao se adotar o modelo de máquina de estado finita não-determinística (MEFind) onde, para o mesmo par "unidade de ensino" e "comportamento", possam haver várias próximas unidades de ensino e a escolhe de uma delas seja feita de maneira aleatória. Isso contribuiria em muito para diversificar o fluxo dentro de um PEI sem que haja intervenção manual do especialista de domínio.

Outros modelos mais complexos, tais como os de diagramas da UML (diagrama de estados, diagrama de seqüência etc.), poderiam ser incorporados para permitir construções de fluxo mais ricas e expressivas dentro de um PEI. Exemplos de elementos que aprimorariam a gama de opções são comentários (presentes em todos os diagramas da UML), "*fork/join*" ou "*split/join*" (para atividades concorrentes, presente no diagrama de atividade) etc. Além de elementos adicionais, o fluxo entre unidades poderia ser controlado de maneira mais livre por especialistas de domínio através do uso de linguagens de *script*, tais como Lua, aliviando dessa maneira os desenvolvedores da tarefa de se criar novos critérios de transições e, assim, agilizando a elaboração de novos PEIs.

Novas versões do GEIC estão planejadas, o que deve fazer com aumente ainda mais o apelo desse sistema aos especialistas da área de educação. Para breve estão planejados suporte a novos tipos de estímulos (além dos habituais figura, som e texto), tais como animações simples, vídeos e estímulos compostos, e também a novos tipos de tentativas, tais como discriminação simples (sucessiva e simultânea), resposta construída com *feedback* seleção a seleção, testes de múltipla escolha etc. Com isso espera-se que o sistema seja flexível o suficiente para atender às necessidades de um conjunto mais variado de especialistas de domínio e alunos.

Padrões para interoperabilidade de unidades de ensino, tais como SCORM, LOM e QTI (SUN, 2002), constituem uma área em franco crescimento e isso tende a se acentuar conforme os LMSs e LCMSs mais populares o adotarem. Futuras pesquisas sobre como suportar nativamente esses padrões na infra-estrutura proposta são de enorme interesse e ajudarão os sistemas desenvolvidos a se moverem em direção aos grandes produtos disponíveis no mercado de sistemas educacionais. Outro benefício, dessa vez em longo prazo, de se adotar

padrões abertos e amplamente adotados para as unidades de ensino é que todo o conhecimento produzido (tanto em termos de unidades de ensino criadas quanto de execuções registradas) pode ser movido para outros sistemas que também adotem os padrões, impedindo assim que um enorme investimento realizado por uma comunidade seja perdido ou abandonado.

Trabalhos adicionais também podem ser desenvolvidos no sentido de obter um melhor aproveitamento do uso das redes de comunicação, diminuindo assim a latência e aumentando o fluxo de dados. Opções incluem a exploração de um modelo de transmissão de dados sob demanda (ao contrário do que ocorre atualmente, onde os dados são obtidos em momentos muito pontuais e em rajadas), o que deve beneficiar principalmente usuários da TVDI. Ainda em relação às redes de comunicação, podem ser pesquisadas medidas para torná-las ainda mais transparentes aos usuários e que, em situações onde elas falhem ou apresentem irregularidades, os sistemas desenvolvidos possam se adaptar de maneira mais satisfatória.

Como resultado da avaliação das técnicas e conceitos utilizados, foi possível localizar pontos na infra-estrutura passíveis de melhorias, principalmente aquelas relacionadas à alta demanda computacional que será exigida no futuro. Pesquisas em áreas como engenharia de sistemas (campo interdisciplinar da engenharia cujo foco é em como projetos complexos podem ser projetados e gerenciados) e sistemas de alta disponibilidade (*High Availability* – HA) tem grande potencial para beneficiar os sistemas desenvolvidos ao fornecer métodos para melhora de disponibilidade, segurança e escalabilidade.

Finalmente, a solução firmou as bases para a criação de uma comunidade virtual (rede social) interdisciplinar composta de desenvolvedores de software, especialistas de domínio, tutores e alunos, cujo objetivo comum é promover o ensino de qualidade. A colaboração entre esses indivíduos facilitará sua comunicação, troca de informações e um fluxo cada vez mais intenso de experiências.

9.2 RESULTADOS E CONTRIBUIÇÕES

A partir deste trabalho será possível a redução nos custos envolvidos no gerenciamento de PEIs, já que tutores poderão acompanhar à distância a execução de PEIs, enquanto alunos não precisarão mais se locomover até as escolas ou centros de pesquisa para pode-

rem participar de programas de ensino. Além disso, a criação de unidades de ensino foi bastante facilitada, assim como foram diminuídas as chances de erros de operação, o que permitirá uma economia ainda maior de recursos por parte das instituições de ensino.

No futuro, especialistas de domínio contarão com uma plataforma para a realização de pesquisas tanto na área de ensino de leitura e escrita quanto em outras, tais como ensino de matemática e notação musical, além de áreas que não envolvam pesquisa, tais como avaliação do conhecimento de sinais de trânsito. A difusão em larga escala de PEIs será possível graças ao aumento da base instalada dos módulos-cliente, já que a infra-estrutura promove a interoperabilidade de aplicações de ambiente *Web*, dispositivos móveis e de TV Digital Interativa.

Programadores e engenheiros de *software* passarão a dispor de uma infra-estrutura integrada, construída sobre uma arquitetura orientada a serviços, onde contarão com ampla documentação, padrões de projeto, componentes de *software* e infra-estrutura de servidores, dessa maneira reduzindo os custos relacionados ao desenvolvimento de módulos.

Foram exploradas e avaliadas técnicas de arquitetura de *software* e computação distribuída, além de um grande número de metodologias, ferramentas de *software* e tecnologias. Espera-se contribuir para a pesquisa em computação através da avaliação desses itens baseada em seu uso num ambiente de produção (e não apenas no âmbito acadêmico). O uso dessas novas tecnologias deverá tornar a solução atrativa também para pesquisadores das áreas de computação e engenharia, já que esses terão um ambiente real para efetuarem suas pesquisas. Além disso, todo o material desenvolvido durante o trabalho, incluindo o código-fonte e documentação, foi disponibilizado sob licenças livres, o que permitirá total liberdade aos interessados em se beneficiarem dos esforços realizados pela equipe.

A partir da análise dos dados coletados durante os experimentos conclui-se que todas as populações avaliadas se mostraram extremamente satisfeitas com a solução proposta neste trabalho. Os especialistas de domínio com experiência prévia (membros da equipe) avaliaram positivamente o sistema e sua evolução, e as ressalvas feitas já se encontram registradas e aprovadas no gerenciador de tarefas do projeto. Os especialistas de domínio ingênuos em relação ao sistema avaliaram muito positivamente o sistema, apesar do pouco conhecimento específico do conteúdo sendo testado. Os especialistas de computação aprovaram as decisões feitas em relação à infra-estrutura computacional da solução, e fizeram elogios principalmente no que se refere à facilidade de uso (mesmo para programadores com pouquíssima experiência em programação) e automação.

Com este trabalho teve início a criação de um ecossistema de pesquisas em torna da infra-estrutura computacional proposta, o que deve fomentar o desenvolvimento de vários trabalhos nas áreas de computação e educação. A união de metodologias de ensino consolidadas, conceitos de arquitetura de *software* e computação distribuída, tecnologias modernas e boas práticas de gerenciamento de projetos, todos unidos sob a disciplina e a sistemática da engenharia de *software*, contribuirão de maneira substancial para um fim social tão importante quando a educação.

REFERÊNCIAS BIBLIOGRÁFICAS

AARRENIEMI-JOKIPELTO, Päivi. **T-learning Model for Learning via Digital TV**. Proceedings of the 16th annual conference on innovation in education for electrical and information engineering, Lappeenranta, Finland. 2005.

AINSWORTH, L. L. **Self-paced instruction:** An innovation that failed. Teaching of Psychology, 6(1), 42-46, 1979.

ALEXANDER, Christopher et al. **A Pattern Language:** Towns, Buildings, Construction. New York: Oxford University Press. 1977.

ALUR, Deepak; CRUPI, John; MALKS, Dan. **Core J2EE Patterns:** Best Practices and Design Strategies. 2ª edição. Prentice Hall / Sun Microsystems Press. 2003.

BARESI, Luciano; NITTO, Elisabetta Di; GHEZZI, Carlo. **Toward Open-World Software:** Issue and Challenges. Computer, 39, 10, 36-43. IEEE Computer Society. 2006.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**http://www.amazon.com/Software-Architecture-Practice-Len-Bass/dp/0201199300**.** Addison-Wesley Professional. 1998.

BECK, Kent. **Extreme Programming Explained**: Embrace Change. Addison-Wesley Professional. 1999.

BELA, Rodrigo Estevan. **Ontologias de domínio auxiliando tarefas de mineração de dados: anotações semânticas e filtros de regras**. São Carlos: UFSCar, 2009. Dissertação (mestrado) – Programa de Pós-Graduação em Ciência da Computação, Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos, São Carlos. 2009.

BOEHM, B. **A Spiral Model of Software Development and Enhancement.** Computer, IEEE, 21(5):61-72, 1988.

BRANDÃO, Zaia; BAETA, Ana Maria; ROCHA, Any Dutra Coelho. **O Estado da Arte da Pesquisa sobre Evasão e Repetência no Ensino do 1º grau no Brasil (1971- 1981).** Revista Brasileira de Estudos Pedagógicos, 64, p. 38-69, 1983.

BRASIL. **Decreto Nº 4.901**. Presidência da República. 26 de Novembro de 2003.

BUSCHMANN, Frank et al. **Pattern-Oriented Software Architecture**: A System of Patterns. Wiley. Volume 1, 1ª edição. 1996.

CHORIANOPOULOS, Konstantinos; LEKAKOS, George. **Learn and Play with Interactive TV**. ACM Computers in Entertainment, 5, 2. Agosto de 2007.

CLELAND, David I.; GAREIS, Roland. **Global project management handbook.** McGraw-Hill Professional. p.1-4. 2006.

CPQD. **Sistema de TV Digital**. Disponível em <http://www.cpqd.com.br/img/historico_tv_digital.pdf>. Acesso: 01 maio 2009.

CPQD. **A TV digital está chegando**. Disponível em <http://www.cpqd.com.br/1/3590+a-tv-digital-esta-chegando-transmissao-tv-digital-cpqd.html>. Acesso em: 01 maio 2009.

CROCKFORD, Douglas. **RFC 4627** – The application/json Media Type for JavaScript Object Notation (JSON). Networking Working Group, The Internet Society. 2006.

CROWELL, C. R., QUINTANAR, L. R., GRANT, K. L. **PROCTOR**: An on-line student evaluation and monitoring system for use with PSI format courses. Behavior Research Methods & Instrumentation, 13(2), 121-127, 1981.

DE ROSE, Julio Cesar Coelho et al. **Aquisição de leitura após historia de fracasso escolar:** Equivalência de estímulos e generalização. Psicologia: Teoria e Pesquisa, 5, p. 325-346, 1989.

ECKERSON, Wayne W. **Three Tier Client/Server Architecture:** Achieving Scalability, Per-for-mance, and Efficiency in Client Server Applications. Open Information Systems 10, 1, 3(20), 1-12, 1995.

EDEN, Amnon H.; KAZMAN, Rick. **Architecture, Design, Implementation.** International Conference on Software Engineering. 2003.

ERL, Thomas. **Service-oriented Architecture:** Concepts, Technology, and Design. Prentice Hall PTR. 2005.

FAPESP. **TIDIA-AE.** Disponível em <http://tidia-ae.incubadora.fapesp.br/portal>. Acesso em: 01 jul. 2009.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. Cap. 5. Dissertação de Doutorado. University of California, Irvine, USA. 2000.

FILHO, Guido Lemos de Souza; LEITE, Luiz Eduardo Cunha; BATISTA, Carlos Eduardo Coelho Freira. **Ginga-J**: The Procedural Middleware for the Brazilian Digital TV System. Jôurnal of the Brazilian Computer Society, 4, 13, 47-56. 2007.

FOWLER, Martin. **Continuous Integration.** Disponível em <http://www.martinfowler.com/articles/continuousIntegration.html>. Acesso em: 01 ago. 2009.

FOWLER, Martin. **Patterns of Enterprise Application Architecture**. Addison-Wesley Professional. 2002.

FSF, Free Software Foundation. **GNU Free Documentation License version 1.3**. Disponibilizada em: nov. 2008. Disponível em <http://www.gnu.org/licenses/gfdl.html>. Acesso em: 01 ago. 2009.

FSF, Free Software Foundation. **GNU General Public License version 3**. Disponibilizada em: jun. 2007. Disponível em <http://www.gnu.org/licenses/gpl.html>. Acesso em: 01 ago. 2009.

GAMMA, Erich et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. 1994.

GEORGIEV, Tsvetozar; GEORGIEVA, Evgenia; SMRIKAROV, Angel. **M-Learning - a New Stage of e-Learning**. Proceedings of the 5th international conference on Computer systems and technologies, Rousse, Bulgaria. Junho, 2004.

GRANT, Lyle. K.; SPENCER, Robert. E. **The personalized system of instruction:** Review and applications to distance education. International Review of Research in Open and Distance Learning, 4(2), 1-12, 2003.

HENRY, Paul. **E-learning technology content and services**. Education + Training, 43, 4, p. 249-255. MCB University Press. 2001.

IBGE. **Pesquisa Nacional por Amostra de Domicílios (PNAD) 2005**. Disponível em <http://www.ibge.gov.br/home/estatistica/populacao/trabalhoerendimento/pnad2005/tabsintese.shtm>. Acesso em: 01 maio 2009.

IEEE. **IEEE Std 1220-1998, Standard for Application and Management of the Systems Engineering Process.** IEEE Press, Piscataway, N. J., 1999.

IET. Institution of Engineering and Technology. Software Infrastructure and Business Applications Specialist Topic. Disponível em < http://kn.theiet.org/communities/software/index.cfm>. Acesso em: 27 fev. 2010.

ISO/IEC. **Product Quality.** ISO/IEC 9126:1991. Software Engineering. 1991.

ITU-T. **Basic principles for a worldwide common family of systems for the provision of interactive television services**. ITU-T J.110. 1997.

JENDROCK, Eric et al. **Java(TM) EE 5 Tutorial.** Java Series. 3ª edição. Prentice Hall PTR. 2006.

KANER, Cem. **Exploratory Testing.** Worldwide Annual Software Testing Conference, Orlando, Florida. 2006.

KELLER, Fred Simmons. **Engineering Personalized Instruction in The Classroom**. Revista Interamericana de Psicologia, 1, (3), p. 144-156, 1967.

KELLER, Fred Simmons. **Neglected rewards in the educational process.** Proc. 23rd Amer. Conf. Acad. Deans, Los Res. Ass., New York, Feb. Pp 9-22, 1967.

KELLER, Fred Simmons. **Good bye teacher.** Journal of Applied Behavior Analysis, 1, (1), p. 79-89, 1968.

KRAFZIG, Dirk; BANKE, Karl; SLAMA, Dirk. **Enterprise SOA:** Service-Oriented Architecture Best Practices**.** Prentice Hall, 2004.

KULIK, James A.; KULIK, Chen-Lin C.; COHEN, Peter A. **A meta-analysis of outcome studies of Keller's Personalized System of Instruction.** American Psychologist, 34, 307-318, 1979.

KULIK, Chen-Lin C.; KULIK, James A.; BANGERT-DROWNS, Robert L. **Effectiveness of mastery learning programs:** A meta-analysis. Review of Educational Research, 60, 265-299, 1990.

KUO, Tien-Ying; HSIAO, Li-Hsien; CHIN, Po-Yun. **Multi-shot Framework With Preloading Architecture for Low Latency MHP Application Delivery**. International Conference on Multimedia and Expo. 1179-1182. IEEE Computer Society. 2007.

KUO, Tien-Ying; LO, Yi-Chung; LAI, Chih-Chun. **Optimal Carousel Stream for Interactive Digital TV Service**. International Symposium on Broadband Multimedia Systems and Broadcasting. 1-6. Abril de 2008.

LYTRAS, Miltiadis; CHOZOS, Polyneikis; POULOUDI, Athanasia. **Interactive Television and e-Learning Convergence:** Examining the Potential of t-Learning. ECEL 2002, The European Conference on e-learning, Brunel University. 2002.

MARTIN, Robert C. **Design Principles and Design Patterns**. Disponível em <http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf>. Acesso em: 01 ago. 2009.

MICROSOFT. **Deployment Patterns.** Microsoft Patterns & Practices. Disponível em <http://msdn.microsoft.com/en-us/library/ms998478.aspx>. Acesso em: 01 ago. 2009.

MANHÃES, Marcus Aurélio Ribeiro; SHIEH, Pei Jen. **Padrão para TV Digital Brasileira**: o Limite do Sistema Canal de Interatividade. CPqD, publicação interna. 2005.

MILLER, George A. et al. **Introduction to WordNet:** An On-line Lexical Database, 1993.

MOODLE. **Moodle.org.** Disponível em <http://moodle.org>. Acesso em: 01 jul. 2009.

NORTH, Dan. **Introducing BDD**. Better Software Magazine. Março de 2006.

O'MALLEY, C. et al. **Guidelines for learning/teaching/tutoring in a mobile environment**. MOBIlearn MP4 Public Deliverables. 2004.

OASIS. **UDDI**. Disponível em <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>. Acesso em: 01 maio 2009.

OMG. **UML 2.2.** Disponível em <http://www.omg.org/spec/UML/2.2>. Acesso em: 01 jul. 2009.

PEAR, Joseph J.; KINSNER, Witold. **Computer-aided personalized system of instruction:** An effective and economical method for short- and long- distance education. Machine- Mediated Learning, 2, 213-237, 1988.

PEAR, Joseph J.; NOVAK, Mark. **Computer-Aided Personalized System of Instruction:** A Program Evaluation. Teaching of Psychology, 23, 119-123, 1996.

PEAR, Joseph J.; CRONE-TODD, Darlene E. **Personalized system of instruction in cyberspace.** Journal of Applied Behavior Analysis, 32, 205-209, 1999.

PRESSMAN, Roger S. **Software Engineering:** A Practitioner's Approach. 5ª edição. McGraw-Hill Science/Engineering/Math. 2001.

REENSKAUG, Trygve Mikkjel Heyerdahl. **MVC**. Disponível em <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>. Acesso em: 01 ago. 2009.

REVELLE, Glenda L. **Educating via Entertainment media**: The Sesame Workshop Approach. Computers and Entertainment, 1, 1, Novembro de 2003.

ROCHA, José Antonio Meira da. **Modelo de monografia e Trabalho de Conclusão de Curso (TCC).** Documento digital do programa MS Word. Disponível em <http://www.meiradarocha.jor.br/news/wp-content/uploads/2007/09/modelo_tcc-2006-09-11b.zip>. Acesso em: 17 jul. 2009.

SAKAI. **Sakai CLE.** Disponível em <http://sakaiproject.org/portal>. Acesso em: 01 jul. 2009.

SHERMAN, J. Gilmour. **Individualizing instruction is not enough.** Educational Technology, 17, 56-60, 1977.

SHERMAN, J. Gilmour. **Reflections on PSI:** Good news and bad. Journal of Applied Behavior Analysis, 25, 59-64, 1992.

SIDMAN, Murray. **Reading and auditory-visual equivalences.** Journal of Speech and learning Research, 14, (1), p. 5-13, 1971.

SIDMAN, Murray.; TAILBY, W. **Conditional discrimination vs. matching to sample:** An expansion of the testing paradigm. Journal of the Experimental Analysis of Behavior, 37, p. 5-22, 1982.

SIDMAN, Murray. **Equivalence relations:** A research history. Authors Cooperative, 1994.

SKINNER, Burrhus Frederic. **Science and Human Behavior.** The Macmillan Company. 1953.

SKINNER, Burrhus Frederic. **Teaching machines.** Science 128, 969-977, 1958.

SOARES, Luiz Fernando Gomes; RODRIGUES, Rogério Ferreira; MORENO, Márcio Ferreira. **Ginga-NCL**: The Declarative Environment of the Brazilian Digital TV System. Journal of the Brazilian Computer Society. SBC. Agosto de 2007.

SOARES, Luiz Fernando Gomes; FILHO, Guido Lemos de Souza. **Interactive Television in Brazil:** System Software and the Digital Divide. European Interactive TV Conference - EuroITV2007. Amsterdam, The Netherlands. Maio de 2007.

SOMMERVILLE, Ian. **Software Engineering.** 8ª edição. Addison Wesley. 2006.

SOUZA, Deisy das Graças de.; DE ROSE, Julio Cesar Coelho. **Desenvolvendo programas indi-vidualizados para o ensino de leitura.** Acta Comportamentalia, 14, (1), p. 77-98, 2006.

SOUZA, Deisy das Graças de. et al. **Teaching Generative Reading Via Recombination of Minimal Textual Units:** A Legacy of Verbal Behavior to Children in Brazil. International Journal of Psycology and Psychological Therapy, 9, (1), p. 19-44, 2009.

STROMER, Robert; MAKAY, Harry A.; STODDARD, Lawrence T. **Classroom applications of stimulus equivalence technology.** Journal of Behavioral Education, 2, p. 225-256, 1992.

SUN Microsystems, Inc. **e-learning Interoperability Standards**. White Paper. 2002.

TANGO. **Tango Desktop Project**. Disponível em <http://tango.freedesktop.org/Tango_Desktop_Project>. Acesso em: 01 junho 2009.

TAVEGGIA, Thomas C. **Personalized instruction:** A summary of comparative research. 1967-1974. American Journal of Physics, 44, 1028-1033, 1976.

TAYLOR, Art. **J2EE and Beyond**: Design, Develop, and Deploy World-Class Java Software. Prentice Hall. 2002.

TELECO. **Por que a Internet tem penetração menor que o celular?** Disponível em <http://www.teleco.com.br/comentario/com175.asp>. Acesso em: 01 maio 2009.

TERGUJEFF, Renne et al. **Mobile SOA**: Service Orientation on Lightweight Mobile Devices. International Conference on Web Services (ICWS 2007). IEEE Computer Society. 2007.

TIDIA-AE. **TIDIA-AE.** Disponível em <http://tidia-ae.usp.br/portal>. Acesso em: 01 jul. 2009.

TRAVASSOS, Guilherme Horta; GUROV, Dmytro; AMARAL, Edgar Augusto Gurgel do. **Introdução è Engenharia de Software Experimental**. COPPE/UFRJ (Relatório Técnico RT-ES-590/02). 2002.

W3C. **Extensible Markup Language (XML)**. Disponível em <http://www.w3.org/XML>. Acesso em: 01 maio 2009.

W3C. **HTTP - Hypertext Transfer Protocol**. Disponível em <http://www.w3.org/Protocols>. Acesso em: 01 maio 2009.

W3C. **SOAP.** Disponível em <http://www.w3.org/TR/soap>. Acesso em: 01 maio 2009.

W3C. **Web Services.** Disponível em <http://www.w3.org/2002/ws>. Acesso em: 01 maio 2009.

W3C. **Web Services Architecture** – W3C Working Group Note. Disponível em <http://www.w3.org/TR/ws-arch>. Acesso em: 01 Maio 2009.

W3C. **Web Services Description Language (WSDL)**. Disponível em <http://www.w3.org/TR/wsdl>. Acesso em: 01 maio 2009.

WAINS, Shahid Islam, MAHMOOD, Waqar. **Integrating M-Learning with E-Learning**. SIGITE'08, Cincinati, Ohio, USA. 2008.

WAISMAN, Thais. **TV Digital Interativa na Educação**: Afinal, Interatividade para quê? 9º Congresso Internacional de Educação a Distância, São Paulo. 2002.

WATSON, William R.; WATSON, Sunnie Lee. **An argument for clarity:** What are learning management systems, what are they not, and what should they become? TechTrends, vol. 51, nº. 2, 2007.

WEST, Matthew; FOWLER, Julian. **Developing High Quality Data Models.** The European Process Industries STEP Technical Liaison Executive (EPISTLE). Relatório técnico. 1999.

YNOGUTI, Adriana Erlinda Nolasco. **O uso do computador como instrumento de ensino:** analisando sua eficiência no ensino individualizado de leitura. Dissertação de Mestrado. Programa de Pós-Graduação em Educação Especial. Universidade Federal de São Carlos, 2002.

ZAINE, Isabela. **Discriminações Simples e Reforçamento Específico e Diferencial para cada Classe no Ensino de Leitura a Indivíduos com Atraso no Desenvolvimento**. São Carlos: UFSCar, 2009. Dissertação (mestrado) – Programa de Pós-Graduação em Psicologia, Centro de Educação e Ciências Humanas, Universidade Federal de São Carlos, São Carlos. 2009. Trabalho não publicado.

GLOSSÁRIO

Arquitetura de *software* – A arquitetura de um sistema de *software* é sua estrutura e inclui componentes de *software*, suas propriedades externas e visíveis e a relação entre elas.

***Cache* –** Componente de um sistema cujo objetivo é melhorar o desempenho armazenando, em níveis de memória mais rápidos, dados freqüentemente requisitados. Esses dados, que geralmente são resultado de cálculos anteriores, são fornecidos de maneira transparente de modo que o cliente do sistema não saiba que o *cache* foi acessado.

Código-fonte – Conjunto de declarações e sentenças, geralmente em formato legível e amigável, voltados para uma linguagem de programação e que, posteriormente, é compilado ou interpretado.

Desktop – No contexto de computadores um *desktop* é um computador pessoal para uso geralmente em uma locação fixa.

Framework de *software* – Um projeto reusável utilizado para a geração de novos sistemas de *software*, geralmente possui bibliotecas de *software* e linguagens de *script* que auxiliam no processo de ligação dos diferentes componentes. *Frameworks* são comumente chamados de aplicações incompletas.

Hardware – Componentes físicos de um sistema computacional, tais como placas, processadores etc.

Infra-estrutura computacional – Estrutura básica ou características de um sistema de software. Relaciona-se a sistemas de software, o que inclui tópicos como sistemas operacionais, *middlewares* e outros programas que ofereçam suporte a aplicações de *software*.

Interface de *software* – Conjunto de operações que podem ser invocadas por aplicações cliente. Essas operações formam uma camada abstrata que isola o comportamento de uma entidade interna e as expõe para comunicação externa.

Interface de usuário – A interface entre humanos (usuários) e máquinas (software), ou seja, onde ocorre a interação entre eles. Pode ser constituída tanto de componentes físicos (*hardware*, tais como teclado, *mouse* etc.) como lógicos (*software*, tais como cursores, janelas etc.).

Middleware – É a camada de *software* que se localiza entre o sistema operacional e as aplicações em cada lado de um sistema distribuído.

Multiplataforma – Se diz que um software é multiplataforma quando ele pode operar em diferentes plataformas computacionais (*software* e *hardware*).

Plug-in – Um *software* menor que interage com um *software* principal, funcionando como um hospedeiro deste e fornecendo extensão de funcionalidade.

***Proxy* –** Um serviço de rede que serve como um intermediário nas trocas de requisições entre clientes e servidores.

Roadmap – Um plano que detalha objetivos de curto, médio e longo prazo, assim como métodos para se alcançar esses objetivos.

Servidor – *Software*, *hardware* ou uma combinação desses dois com o objetivo de fornecer serviços a clientes.

Sistema de *Software* – Conjunto de entidade de *software* interdependentes que interagem entre si com o objetivo de formar uma solução integrada.

Software – Programas ou outros tipos de informação que podem ser executados, lidos ou escritos por um computador.

Timeline – Representação gráfica ou tabular de uma seqüência de eventos.

Web – Abreviação (assim como WWW) de "*World Wide Web*", que é um sistema de hipertextos conectados ao longo da Internet.

ANEXO A - Materiais do Estudo 1

Neste anexo estão os materiais utilizados durante a aplicação do Estudo 1, voltado aos especialistas do domínio de psicologia que participaram durante o desenvolvimento do sistema (descrito em detalhes no capítulo 7.1).

Os materiais são o termo de consentimento livres e esclarecidos (TCLE), onde o indivíduo aceita participar deste projeto de pesquisa, e um questionário com 34 questões.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Eu, **Alex Fernando Orlando**, estou desenvolvendo a pesquisa de mestrado "**Infra-estrutura para o Gerenciamento de Programas de Ensino Individualizados**" sob a orientação do **Prof. Dr. Cesar Augusto Camillo Teixeira**, membro do Laboratório de Inovação em Ciência e Engenharia (LINCE), do Departamento de Computação (DC) da Universidade Federal de São Carlos (UFSCar).

O objetivo da pesquisa é explorar e avaliar técnicas de arquitetura de software e computação distribuída, buscando uma solução interdisciplinar que proporcione a redução dos custos envolvidos no gerenciamento de programas de ensino individualizados (PEIs). É proposta uma infra-estrutura computacional que viabiliza o desenvolvimento de módulos de sistema voltados para a autoria, aplicação e gerenciamento de PEIs. Como estudo de caso da pesquisa, foi desenvolvido o sistema GEIC (Gerenciador de Ensino Individualizado por Computador), já em uso por especialistas do departamento de psicologia.

Uma das avaliações do sistema GEIC será em relação à sua qualidade. O ponto de vista adotado será o dos especialistas do domínio de educação que participaram durante a evolução do sistema e que contribuíram com sugestões, críticas e testes. Gostaríamos de convidá-lo(a) a responder um questionário para avaliação do projeto. Este questionário poderá ser respondido e entregue posteriormente.

Caso deseje participar, você terá liberdade e direito de desistir da realização da tarefa e poderá, a qualquer momento, discutir conosco qualquer questão ou dúvida e retirar seu consentimento, caso sinta-se desconfortável com sua participação.

Os procedimentos da pesquisa aos quais você será submetido não devem representar nenhum risco (físico ou psicológico). Trata-se de procedimentos de uso corrente em pesquisas da área e não temos na literatura indicações da possibilidade de qualquer risco. A despeito da previsão da ausência de riscos, o pesquisador responsável compromete-se a, diante de qualquer desconforto ou mal-estar, suspender a tarefa.

A divulgação do conjunto de dados ocorrerá em forma de artigo científico, apresentação de trabalhos em eventos científicos e em dissertação de conclusão de mestrado, sempre preservando a sua identidade, com total sigilo sobre seu nome e quaisquer outros elementos que permitam sua identificação.

_____

Alex Fernando Orlando
Aluno de Pós-Graduação - Mestrado
Para contato com o responsável: Alex - (16) 3351-8614 (LINCE)

AUTORIZAÇÃO

Eu,_____, aceito participar do projeto de pesquisa "Infra-estrutura para o Gerenciamento de Programas de Ensino Individualizados". Declaro que li o Consentimento Livre e Esclarecido e que estou de acordo com minha participação nos termos descritos.

São Carlos, _____/ _____/ 2009.

_____

Assinatura

Questionário de Avaliação do Sistema GEIC

Público-alvo: especialistas de psicologia já familiarizados com o sistema

Dados pessoais

1. Nome completo:
2. E-mail:
3. Idade:
4. Curso:
5. Semestre:
6. Orientador:
7. Linha de pesquisa:
8. Há quanto tempo trabalha com o GEIC:
9. De quais lugares você costuma acessar o GEIC? Se possível, coloque também a velocidade da Internet nesses locais.
10. Com que freqüência você utiliza os módulos:

	nunca	**pouco**	**freqüentemente**	**muito**
Aluno	☐	☐	☐	☐
Autoria	☐	☐	☐	☐
Consulta	☐	☐	☐	☐
Equipe	☐	☐	☐	☐
Player	☐	☐	☐	☐
Tutor	☐	☐	☐	☐

Qualidade do sistema

(escala de 0 a 3. Se possível, justifique sua resposta.)

1. O conjunto de funcionalidades é adequado suas necessidades?
2. O sistema fornece resultados precisos ou com a precisão dentro do que foi acordado/solicitado?
3. O sistema protege suas informações e as fornece apenas (e sempre) às pessoas autorizadas?
4. O software consegue lidar com falhas decorrentes de defeitos no software? Ou seja, você recebe informações e direções claras quando um erro no sistema ocorre?
5. O software mantém o funcionamento adequado mesmo quando ocorrem defeitos nele ou nas suas interfaces externas? Ou seja, quando o sistema falha ou uma conexão com a rede não é conseguida, são exibidas alternativas que permitam a você continuar trabalhando?
6. O software se recupera após uma falha, restabelecendo seus níveis de desempenho e recuperando os seus dados? Ou seja, em caso de erro do sistema de sistema, o desempenho da aplicação é mantido em futuras execuções e você consegue recuperar seus dados?
7. O sistema pode ser compreendido com facilidade, permitindo a você avaliar se o mesmo pode ser usado para satisfazer as suas necessidades específicas?
8. O sistema é fácil de ser aprendido?
9. O sistema facilita a sua operação por parte do usuário, incluindo a maneira como ele tolera erros de operação? Ou seja, caso o usuário cometa erros de operação, o sistema evita ou facilita suas correções?

10. O sistema possui características que possam atrair um potencial usuário para o sistema?
11. Os tempos de resposta (ou de processamento) estão dentro das especificações?
12. O sistema consome adequadamente os recursos computacionais disponíveis?
13. O sistema facilita o diagnóstico de eventuais problemas, assim como a identificação as causas das deficiências ou falhas?
14. É fácil para os desenvolvedores modificar o comportamento do software?
15. O sistema é capaz de evitar efeitos colaterais decorrentes de modificações introduzidas?
16. É possível se testar o sistema modificado, tanto quanto as novas funcionalidades quanto as não afetadas diretamente pela modificação?

Futuras versões

1. Quais novos tipos de estímulo você gostaria que fossem adicionados:
   ☐nenhum ☐figura+som ☐figura+texto ☐som+texto
   ☐texto com fonte personalizável ☐vídeo Outros:
2. Quais novos tipos de tentativa você gostaria que fossem adicionados
   ☐nenhum ☐go-no-go ☐discriminação simples
   ☐MTS com posições de comparação aleatórias
   ☐MTS com matriz de comparações (oito posições, como as usadas por Sidman)
   ☐CR com feedback seleção a seleção
   Outros:
3. Quais outras melhorias você gostaria que fossem feitas em cada módulo:
   Alunos:
   Autoria:
   Consulta:
   Equipe:
   Player:
   Site:
   Tutor:

Em relação a outros sistemas

1. Possui experiência prévia com outro software para gerenciamento de programas de ensino? Se sim, diga qual.
2. Caso possua experiência prévia com outro software para gerenciamento de programas de ensino, descreva sua produtividade ao usar o GEIC em relação a esse outro software.
3. Você percebe alguma alteração no seu fluxo/forma de trabalho ao usar o GEIC em relação a esse outro software? Descreva.

Geral

1. Quais os principais pontos positivos do GEIC?
2. Quais os principais pontos negativos do GEIC?

ANEXO B – Materiais do Estudo 2

Neste anexo estão os materiais utilizados durante a aplicação do Estudo 2, voltado aos especialistas do domínio de psicologia sem conhecimento anterior do sistema desenvolvido (descrito em detalhes no capítulo 7.2).

Os materiais são o termo de consentimento livres e esclarecidos (TCLE), onde o indivíduo aceita participar deste projeto de pesquisa, um conjunto de slides contendo uma visão geral sobre o sistema e o enunciado de uma atividade, e um questionário com 26 questões.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Eu, **Alex Fernando Orlando**, estou desenvolvendo a pesquisa de mestrado "**Infra-estrutura para o Gerenciamento de Programas de Ensino Individualizados**" sob a orientação do **Prof. Dr. Cesar Augusto Camillo Teixeira**, membro do Laboratório de Inovação em Ciência e Engenharia (LINCE), do Departamento de Computação (DC) da Universidade Federal de São Carlos (UFSCar).

O objetivo da pesquisa é explorar e avaliar técnicas de arquitetura de software e computação distribuída, buscando uma solução interdisciplinar que proporcione a redução dos custos envolvidos no gerenciamento de programas de ensino individualizados (PEIs). É proposta uma infra-estrutura computacional que viabiliza o desenvolvimento de módulos de sistema voltados para a autoria, aplicação e gerenciamento de PEIs. Como estudo de caso da pesquisa, foi desenvolvido o sistema GEIC (Gerenciador de Ensino Individualizado por Computador), já em uso por especialistas do departamento de psicologia.

Uma das avaliações do sistema GEIC será em relação à sua qualidade e facilidade de uso. O ponto de vista adotado será o de especialistas da área de psicologia que nunca utilizaram o sistema. A avaliação é baseada em um questionário que será aplicado após uma palestra

Os participantes terão liberdade e direito de desistir da realização da tarefa caso sintam-se desconfortáveis ou prejudicadas, não havendo qualquer ônus de sua parte. Os participantes poderão, a qualquer momento, discutir conosco qualquer questão ou dúvida e retirar seu consentimento, caso sintam-se desconfortáveis com sua participação.

Os procedimentos da pesquisa aos quais os participantes serão submetidos não devem representar nenhum risco (físico ou psicológico). Trata-se de procedimentos de uso corrente em pesquisas da área e não temos na literatura indicações da possibilidade de qualquer risco. A despeito da previsão da ausência de riscos, a pesquisadora responsável compromete-se a, diante de qualquer desconforto ou mal-estar apresentado pelos participantes (manifestado por eles ou percebido pelo pesquisador), suspender a tarefa realizada no momento para analisar as possíveis variáveis envolvidas na situação e planejar procedimentos alternativos que cessem a possibilidade de reaparecimento de desconforto.

A divulgação do conjunto de dados ocorrerá em forma de artigo científico, apresentação de trabalhos em eventos científicos e em uma dissertação, sempre preservando a identidade dos participantes, com total sigilo sobre os nomes e quaisquer outros elementos que permitam a identificação deles.

_____

Alex Fernando Orlando

Aluno de Pós-Graduação - Mestrado

Para contato com o responsável: Alex - (16) 3351-8614 (LINCE)

AUTORIZAÇÃO

Eu,_____, aceito participar do projeto de pesquisa "Infra-estrutura para o Gerenciamento de Programas de Ensino Individualizados" nas dependências do LINCE. Declaro que li o Consentimento Livre e Esclarecido e que estou de acordo com minha participação nos termos descritos.

São Carlos, _____/ _____/ 2009.

_____

Assinatura

Slides

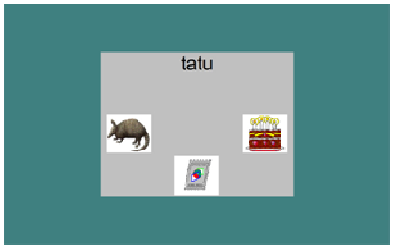
GEIC

Gerenciador de Ensino Individualizado por Computador

Introdução

- Aprendizagem de leitura e escrita
- Discriminações simples e condicionais
- Estímulos, tentativas, blocos, passos e programas
- Emparelhamento com modelo
- Equivalência de estímulos: reflexividade, simetria e transitividade

Introdução (cont.)



Introdução (cont.)

- Programas de ensino individualizados (PEIs)
- Sistemas de Instrução Personalizados (PSI)
  - Ritmo individualizado
  - Domínio do material
  - Instrutores apenas para motivação
  - Ênfase à palavra escrita
  - Uso co tutores

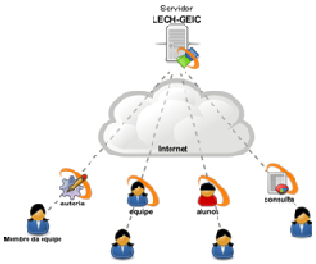
GEIC

- Gerenciador de Ensino Individualizado por Computador
- Sucessor do ProgLeit (Aprendendo a Ler e a Escrever em Pequenos Passos)
- Uso de tecnologias modernas como Java
- Acessível através da Internet (basta apenas um navegador e o plugin Java)

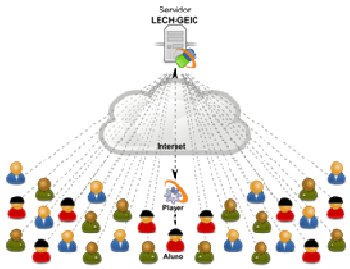
GEIC (cont.)

- Sistema é dividido em módulos:
  - Alunos
  - Autoria
  - Consulta
  - Equipe
  - Player
  - Tutor
  - Site
  - etc...

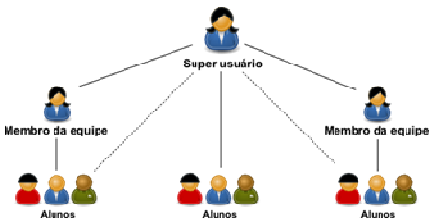
GEIC (cont.)



GEIC (cont.)



GEIC (cont.)



Tarefa

- Adicione os estímulos de figura e som da pasta "Experimento". Crie também estímulos de texto relativos a esses estímulos
- Crie tentativas: CR (CR_FT, CR_ST) e MTS (MTS_FF, MTS_SF, MTS_TT), totalizando 5 com os estímulos que você preferir
- Crie um bloco onde o aluno, caso erre em algum lugar, volte para a primeira ocorrência de tentativa. Testar cada ocorrência antes de salvar.

Tarefa (cont.)

- Crie um passo com dois blocos, em seqüência. Teste cada ocorrência de bloco antes de encerrar
- Cria um programa com apenas o passo que você criou. Teste a ocorrência de passo antes de encerrar
- Disponibilize seu programa
- Crie um aluno

Tarefa (cont.)

- Associa seu programa ao aluno
- Crie um membro de equipe com permissão apenas para monitorar alunos (tutor)
- Atribua monitoria sob esse aluno ao novo tutor
- Inicie uma sessão com o tutor recém criado, e referente ao aluno recém criado
- Consulte os resultados

Questionário de Avaliação do Sistema GEIC

Público-alvo: especialistas de psicologia não familiarizados com o sistema

Dados pessoais

1. Nome completo:
2. E-mail:
3. Idade:
4. Curso:
5. Semestre:
6. Orientador:
7. Linha de pesquisa:
8. Possui experiência em análise do comportamento e equivalência de estímulos?

Qualidade do sistema

(escala de 0 a 3. Se possível, justifique sua resposta.)

1. O conjunto de funcionalidades é adequado?
2. O sistema fornece resultados precisos?
3. O sistema protege suas informações e as fornece apenas (e sempre) às pessoas autorizadas?
4. O sistema pode ser compreendido com facilidade, permitindo a você avaliar se o mesmo pode ser usado para satisfazer as suas necessidades específicas?
5. O sistema é fácil de ser aprendido?
6. O sistema facilita a sua operação por parte do usuário, incluindo a maneira como ele tolera erros de operação? Ou seja, caso o usuário cometa erros de operação, o sistema evita ou facilita suas correções?
7. O sistema possui características que possam atrair um potencial usuário para o sistema?
8. Os tempos de resposta (ou de processamento) são adequados?
9. O sistema consome adequadamente os recursos computacionais disponíveis?
10. O sistema facilita o diagnóstico de eventuais problemas, assim como a identificação as causas das deficiências ou falhas?

Futuras versões

1. Quais novos tipos de estímulo você gostaria que fossem adicionados:
   ☐nenhum ☐figura+som ☐figura+texto ☐som+texto
   ☐texto com fonte personalizável ☐vídeo Outros:

2. Quais novos tipos de tentativa você gostaria que fossem adicionados
   ☐nenhum ☐go-no-go ☐discriminação simples
   ☐MTS com posições de comparação aleatórias
   ☐MTS com matriz de comparações (oito posições, como as usadas por Sidman)
   ☐CR com feedback seleção a seleção
   Outros:
3. Quais outras melhorias você gostaria que fossem feitas em cada módulo:
   Alunos:

Autoria:
Consulta:
Equipe:
Player:
Site:
Tutor:

Em relação a outros sistemas

1. Possui experiência prévia com outro software para gerenciamento de programas de ensino? Se sim, diga qual.
2. Caso possua experiência prévia com outro software para gerenciamento de programas de ensino, descreva sua produtividade ao usar o GEIC em relação a esse outro software.
3. Você percebe alguma alteração no seu fluxo/forma de trabalho ao usar o GEIC em relação a esse outro software? Descreva.

Geral

1. Quais os principais pontos positivos do GEIC?
2. Quais os principais pontos negativos do GEIC?

ANEXO C – Materiais do Estudo 3

Neste anexo estão os materiais utilizados durante a aplicação do Estudo 3, voltado aos especialistas de computação sem conhecimento anterior da infra-estrutura desenvolvida (descrito em detalhes no capítulo 7.3).

Os materiais são o termo de consentimento livres e esclarecidos (TCLE), onde o indivíduo aceita participar deste projeto de pesquisa, um conjunto de slides contendo uma visão geral sobre a infra-estrutura e o enunciado de uma atividade, e um questionário com 37 questões.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Eu, **Alex Fernando Orlando**, estou desenvolvendo a pesquisa de mestrado "**Infra-estrutura para o Gerenciamento de Programas de Ensino Individualizados**" sob a orientação do **Prof. Dr. Cesar Augusto Camillo Teixeira**, membro do Laboratório de Inovação em Ciência e Engenharia (LINCE), do Departamento de Computação (DC) da Universidade Federal de São Carlos (UFSCar).

O objetivo da pesquisa é explorar e avaliar técnicas de arquitetura de software e computação distribuída, buscando uma solução interdisciplinar que proporcione a redução dos custos envolvidos no gerenciamento de programas de ensino individualizados (PEIs). É proposta uma infra-estrutura computacional que viabiliza o desenvolvimento de módulos de sistema voltados para a autoria, aplicação e gerenciamento de PEIs. Como estudo de caso da pesquisa, foi desenvolvido o sistema GEIC (Gerenciador de Ensino Individualizado por Computador), já em uso por especialistas do departamento de psicologia.

Uma das avaliações será em relação aos processos de gerenciamento de projeto e desenvolvimento de software utilizando a infra-estrutura proposta. O ponto de vista adotado será o de engenheiros e desenvolvedores de software da área de computação que nunca utilizaram a infra-estrutura. A avaliação é baseada em um questionário que será aplicado após uma palestra e algumas atividades no computador.

Os participantes terão liberdade e direito de desistir da realização da tarefa caso sintam-se desconfortáveis ou prejudicadas, não havendo qualquer ônus de sua parte. Poderão também, a qualquer momento, discutir conosco qualquer questão ou dúvida e retirar seu consentimento, caso sintam-se desconfortáveis com sua participação.

Os procedimentos da pesquisa aos quais os participantes serão submetidos não devem representar nenhum risco (físico ou psicológico). Trata-se de procedimentos de uso corrente em pesquisas da área e não temos na literatura indicações da possibilidade de qualquer risco. A despeito da previsão da ausência de riscos, a pesquisadora responsável compromete-se a, diante de qualquer desconforto ou mal-estar apresentado pelos participantes (manifestado por eles ou percebido pelo pesquisador), suspender a tarefa realizada no momento para analisar as possíveis variáveis envolvidas na situação e planejar procedimentos alternativos que cessem a possibilidade de reaparecimento de desconforto.

A divulgação do conjunto de dados ocorrerá em forma de artigo científico, apresentação de trabalhos em eventos científicos e em uma dissertação, sempre preservando a identidade dos participantes, com total sigilo sobre os nomes e quaisquer outros elementos que permitam a identificação deles.

_____

Alex Fernando Orlando

Aluno de Pós-Graduação - Mestrado

Para contato com o responsável: Alex - (16) 3351-8614 (LINCE)

AUTORIZAÇÃO

Eu,_____, aceito participar do projeto de pesquisa "Infra-estrutura para o Gerenciamento de Programas de Ensino Individualizados" nas dependências do LINCE. Declaro que li o Consentimento Livre e Esclarecido e que estou de acordo com minha participação nos termos descritos.

São Carlos, _____/ _____/ 2009.

_____

Assinatura

Slides

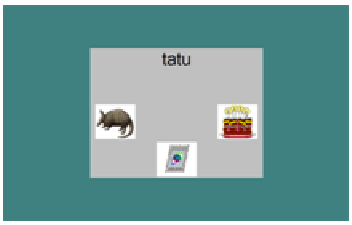
GEIC

Gerenciador de Ensino Individualizado por Computador

Introdução

- Aprendizagem de leitura e escrita
- Discriminações simples e condicionais
- Estímulos, tentativas, blocos, passos e programas
- Emparelhamento com modelo
- Equivalência de estímulos: reflexividade, simetria e transitividade

Introdução (cont.)



Introdução (cont.)

- Programas de ensino individualizados (PEIs)
- Sistemas de Instrução Personalizados (PSI)
  - Ritmo individualizado
  - Domínio do material
  - Instrutores apenas para motivação
  - Ênfase à palavra escrita
  - Uso de tutores

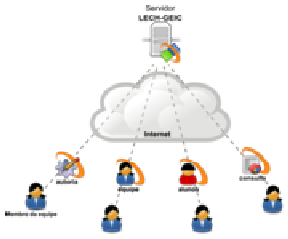
GEIC

- Gerenciador de Ensino Individualizado por Computador
- Sucessor do ProgLeit (Aprendendo a Ler e a Escrever em Pequenos Passos)
- Uso de tecnologias modernas como Java
- Acessível através da Internet (basta apenas um navegador e o plugin Java)

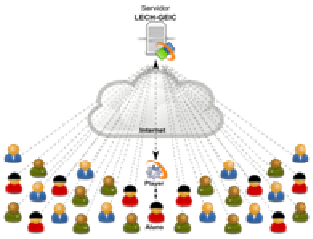
GEIC (cont.)

- Sistema é dividido em módulos:
  - Alunos
  - Autoria
  - Consulta
  - Equipe
  - Player
  - Tutor
  - Site
  - etc...

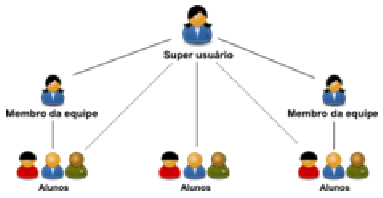
GEIC (cont.)



GEIC (cont.)



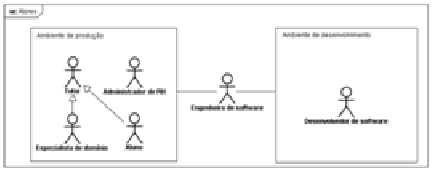
GEIC (cont.)



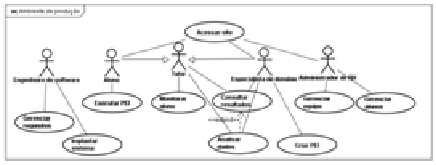
Interoperabilidade

- PC / Web
- TV Digital Interativa (SBTVD)
- Dispositivos móveis
- Tidia-Ae/Sakai

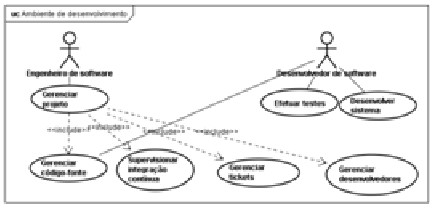
Atores



Atores (ambiente de produção)



Atores (ambiente de desenv.)



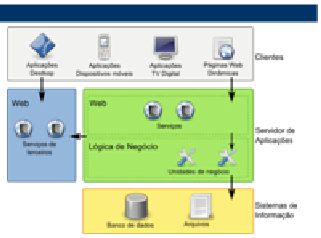
Gerenciamento de Projeto

- Controle de revisão
- Testes (unitários, funcionais, integração)
- Automação de construção
- Gerenciamento de tarefas
- Integração contínua

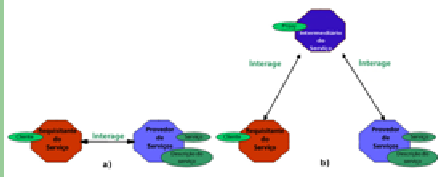
Arquitetura

- Padrões arquiteturais
  - SOA
  - MVC
  - Multicamadas

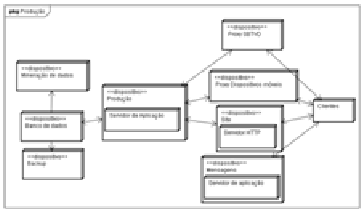
Multicamadas + SOA



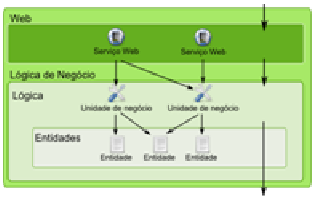
SOA (sem descoberta dinâmica)



Implantação (ambiente de prod.)

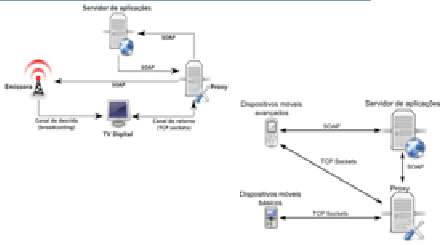


Servidor de aplicações

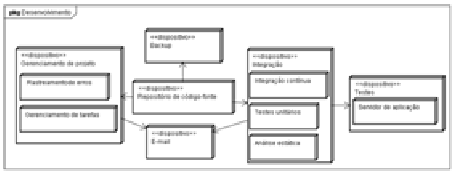


Proxy

Comunicação (SBTVD e mobile)



Implantação (ambiente de desenv)

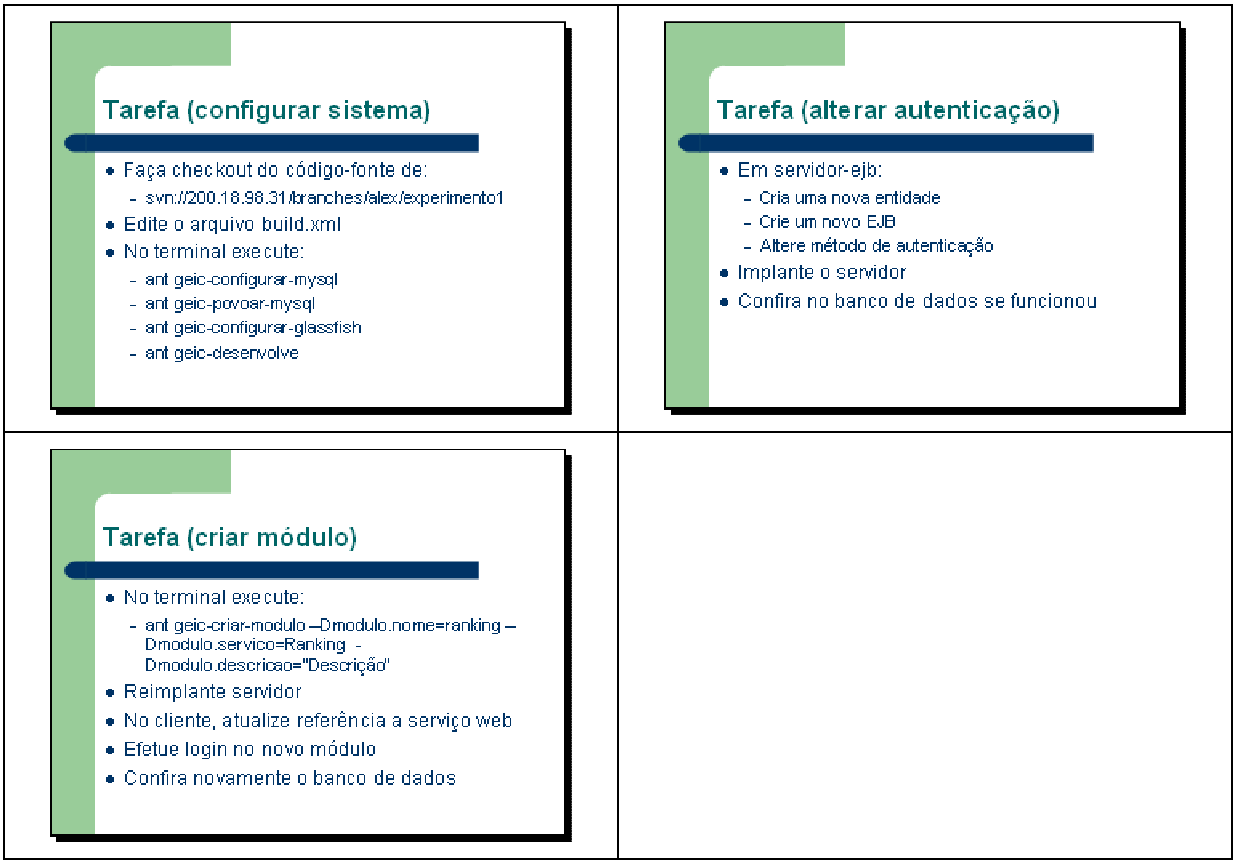


Padrões de projeto

- Fachada (façade)
- Fachada remota (remote façade)
- Objeto de transferência de dados (data transfer object – TO)

Tarefa (preparar ambiente)

- Java e Netbeans já instalados
- Instale o Apache Ant
- Instale o MySQL
- Instale o Glassfish
- Adicione referência ao Glassfish no Netbeans
- Instale TortoiseSVN


Tarefa (configurar sistema)
Faça checkout do código-fonte de:
svn://200.18.98.31/branches/alex/experimento1
Edite o arquivo build.xml
No terminal execute:
ant geic-configurar-mysql
ant geic-povoar-mysql
ant geic-configurar-glassfish
ant geic-desenvolve
Tarefa (alterar autenticação)
Em servidor-ejb:
Cria uma nova entidade
Crie um novo EJB
Altere método de autenticação
Implante o servidor
Confira no banco de dados se funcionou
Tarefa (criar módulo)
No terminal execute:
ant geic-criar-modulo –Dmodulo.nome=ranking –Dmodulo.servico=Ranking -Dmodulo.descricao="Descrição"
Reimplante servidor
No cliente, atualize referência a serviço web
Efetue login no novo módulo
Confira novamente o banco de dados

Questionário de Avaliação da Infra-estrutura proposta

Público-alvo: especialistas de computação não familiarizados com a infra-estrutura

Dados pessoais

1. Nome completo:
2. E-mail:
3. Idade:
4. Curso:
5. Semestre:
6. Orientador:
7. Linha de pesquisa:
8. Qual sua experiência anterior com desenvolvimento de software?
   - ☐ Já desenvolvi software sozinho para fins pessoais
   - ☐ Já desenvolvi software sozinho na universidade
   - ☐ Já desenvolvi software sozinho em empresas
   - ☐ Já desenvolvi software em grupo na universidade
   - ☐ Já desenvolvi software em grupos em empresas
9. Quanto tempo de experiência você possui em programação?
10. Possui experiência com a plataforma Java? Se sim, quanto tempo?
11. Possui experiência com a API JFC/Swing para desenvolvimento de GUIs em Java? Se sim, quanto tempo?
12. Possui experiência com desenvolvimento de aplicações distribuídas? Se sim, quanto tempo?
13. Possui experiência com APIs para persistência de dados? Se sim, diga quais APIs e há tempo tempo trabalha com elas.
14. Possui experiência com a plataforma Java EE? Se sim, qual versão e quanto tempo de experiência.
15. Possui experiência com bancos de dados? Se sim, diga qual SGBD e quanto tempo de experiência.
16. Possui experiência com ferramentas para gerenciamento de projeto (Trac, dotProject, JIRA etc.)? Se sim, diga qual e quanto tempo.
17. Possui experiência com ferramentas para controle de revisões (Subversion, CVS, Git etc.)? Se sim, diga qual e quanto tempo.
18. Possui experiência com ferramentas para construção de software automatizada (Ant, Mavem, Makefile etc.)? Se sim, diga qual e quanto tempo.
19. Possui experiência com ferramentas para integração contínua (Hudson, CruiseControl, Continuum etc.)? Se sim, diga qual e quanto tempo.

Qualidade da infra-estrutura

(escala de 0 a 3. Se possível justifique sua resposta.)

1. Os padrões de arquitetura utilizados são adequados?
2. Os padrões de projetos utilizados são adequados?
3. O modelo de classes descreve bem as entidades envolvidas e seus relacionamentos?
4. A camada de persistência realiza adequadamente o intermédio entre as camadas de dados e de negócio?

5. A infra-estrutura tem potencial para crescer de maneira controlada e sustentável (escalabilidade)?
6. A infra-estrutura facilita a manutenção dos sistemas desenvolvidos (manutenibilidade)?
7. A documentação disponível é suficiente para o entendimento da infra-estrutura?
8. A infra-estrutura permite o desenvolvimento disciplinado e eficiente de software?
9. A infra-estrutura facilita o desenvolvimento, incluindo a maneira como ele tolera erros de codificação?
10. A infra-estrutura possui características que possam atrair um potencial desenvolvedor para o sistema?
11. Os tempos de resposta (ou de processamento) estão dentro do que você considera aceitável?
12. A infra-estrutura consome adequadamente os recursos computacionais disponíveis?
13. A infra-estrutura facilita o diagnóstico de eventuais problemas, assim como a identificação as causas das deficiências ou falhas?
14. É fácil modificar o comportamento dos sistemas desenvolvidos?
15. A infra-estrutura é capaz de evitar efeitos colaterais decorrentes de modificações introduzidas?
16. É possível se testar o sistema modificado, tanto quanto as novas funcionalidades quanto as não afetadas diretamente pela modificação?
17. As facilidades da infra-estrutura relativas à interoperabilidade entre diferentes plataformas é satisfatória?
18. A infra-estrutura permite controlar e localizar alterações (rastreabilidade)?
19. A infra-estrutura possibilita o gerenciamento adequado das tarefas, melhorias e defeitos a serem realizadas?
20. A infra-estrutura possibilita o gerenciamento adequado de revisões de software?
21. A infra-estrutura permite que os sistemas desenvolvidos sejam construídos (*build*) automaticamente, ou com um número mínimo de intervenções manuais?
22. A infra-estrutura facilita a organização e o cumprimento de prazos para entrega de sistemas?

Em relação a outras infra-estruturas/ambientes

1. Possui experiência prévia com outra infra-estrutura / ambiente de desenvolvimento? Se sim, diga qual.
2. Caso possua experiência prévia com outra infra-estrutura, descreva sua produtividade ao usar a infra-estrutura proposta em relação a essa outra.
3. Você percebe alguma alteração no seu fluxo/forma de trabalho ao usar a infra-estrutura apresentada em relação a essa outra? Descreva.

Geral

1. Quais os principais pontos positivos da infra-estrutura?
2. Quais os principais pontos negativos da infra-estrutura?
3. Quais melhorias você acredita que poderiam ser realizadas na infra-estrutura?