# Mais pontes e menos ilhas
# Estratégias para a integração de software educacional

André Santanchè[1], Cesar Augusto Camillo Teixeira[1,2]

[1]Núcleo de Pesquisa em Redes de Computadores / Núcleo de Pesquisa e Projetos em Educação a Distância – Universidade Salvador (UNIFACS)

40.210-730 – Salvador – BA – Brasil

santanche@unifacs.br

[2]Núcleo de Pesquisas em Computação – Faculdades COC (UNICOC)

Ribeirão Preto – SP – Brasil

cesar.teixeira@coc.com.br

**Resumo**. A facilidade de compartilhamento de *software* que a Internet tem proporcionado nos faz perceber que a simples possibilidade de acesso a uma extensa gama de *software* educacional, espalhada por todo o mundo, não é isoladamente fator decisivo para o uso efetivo deste *software*.
Este trabalho investiga como a fragmentação nas iniciativas de produção de *software* educacional destituído, na maioria dos casos, de padrões para sua integração, tem multiplicado os esforços para a solução de problemas comuns e desmotivado o uso efetivo de grande parte do que é produzido. Como contribuição para a minimização desse problema, apresentamos um conjunto de estratégias para a construção de *software* educacional, que conduz a uma abordagem de "produzir para integrar". Apresentamos também um projeto prático no qual estas estratégias foram implementadas. Esta perspectiva proporciona a soma de esforços na produção de soluções que são mais integradas e mais interessantes para o usuário final.

**Palavras chave:** Avaliação e Desenvolvimento de Software Educativo, Arquiteturas Distribuídas para Software Educativo, Integração de *software*, componentes de *software*, *Anima*.

## 1. Introdução

A cada ano cresce em todo o mundo a legião de autores de *software* educacional. Neste grupo se misturam profissionais em projeto e construção de *software*, com educadores e entusiastas que muitas vezes dedicam seu tempo livre à elaboração de projetos pessoais.

Tal crescimento, somado à dimensão de integração proporcionada pela Internet, na qual qualquer autor pode tornar seu produto disponível a milhões de usuários, tem modificado significativamente a postura dos usuários frente ao *software* educacional.

Em uma parcela significativa de domínios de conhecimento, o problema começa a deixar de ser: "como produzir o *software* que necessito",

para ser "onde encontrar e como usar o *software* que necessito".

A grande quantidade de *software* educacional disponível na Internet resulta da confluência de origens, que utilizam os mais diversos padrões para organização e construção deste tipo de *software*. Ao buscar um produto que atenda sua necessidade, o usuário se defronta freqüentemente com uma coleção de peças de *software* – que são incompatíveis e não dispõem de mecanismos de integração – abordando diversos aspectos do mesmo problema, ou o mesmo problema sob perspectivas diferentes.

É comum:

- Que cada peça de *software* exija que o usuário aprenda e se adapte a um padrão diferente.
- Existirem poucos mecanismos de integração entre os produtos, os quais, quando disponíveis, são geralmente limitados a trocas simples de arquivos de imagem ou texto.
- Que o autor dispenda esforço desnecessário na solução duplicada de um mesmo problema. Este esforço poderia ser canalizado para a construção de um produto único e melhor.

Os educadores-usuários do *software* educacional não são em sua maioria especialistas em informática e têm dificuldade em aprender a lidar com uma diversidade muito grande de sistemas. Conseqüentemente, eles acabam aprendendo a trabalhar com alguns poucos sistemas abrangentes e, eventualmente, com um ou outro *software* dedicado, que atenda de forma satisfatória uma necessidade específica.

Uma fração significativa do *software* produzido acaba sendo muito pouco utilizada, ou até mesmo não é utilizada. Como observa Bastiaan:

"*O sonho de muitos desenvolvedores-educadores atuais é que alguém de uma grande editora verá que seu produto vale a pena e pagará muito dinheiro por ele. A realidade, certamente, é que isto acontece com muito poucas peças de software educacional. A tragédia não consiste nisto, mas sim no fato de que os outros programas raramente vêem a luz do dia em outro lugar além da sala de aula do educador*" [Bastiaan].

A integração do *software* pode acontecer em diversos níveis, de formas diferentes e com propósitos diferentes. Neste trabalho destacamos algumas estratégias para integração que consideramos de especial relevância no contexto do *software* educacional.

No tópico **Dimensões de Integração** são exploradas estas estratégias e o modo como têm sido aplicadas em iniciativas de escopo internacional.

Em seguida, no tópico **Articulando Estratégias para Integração**, apresentamos propostas práticas para a integração de *software* educacional, tomando como base a experiência na construção de um modelo inserido no projeto *Anima* [Santanchè 2001], que estabelece como um de seus objetivos a integração de *software*, independente de sua plataforma ou implementação.

## 2. Dimensões de Integração

A integração entre dois sistemas diferentes de *software* pode se dar em diversos graus e modalidades, conforme a necessidade de integração e certos aspectos de compatibilidade.

Destacamos aqui duas modalidades de integração de especial interesse em nosso estudo:

- intercâmbio de dados;
- integração de procedimentos.

O intercâmbio de dados pode acontecer de maneira síncrona ou não.

Intercambiar dados de maneira assíncrona já é uma atividade bastante corriqueira em alguns domínios da computação e isto geralmente se dá através da troca de arquivos, cujo formato pode ser compreendido por dois programas de computador diferentes.

Por exemplo, é muito comum que dois processadores de texto diferentes dêem suporte a formatos de arquivo que ambos são capazes de editar. Os formatos de arquivos de imagem têm migrado para uma crescente padronização. A maior parte dos novos sistemas que atuam diretamente com arquivos de imagem é capaz de fazer uso de formatos amplamente difundidos.

Em muitos domínios, no entanto, este tipo de integração ainda traz grande dificuldade.

O intercâmbio de dados de maneira síncrona entre sistemas diferentes constitui um desafio maior. Não apenas dois programas de computador devem ser capazes de compreender um formato comum para representação dos dados, mas precisam de um protocolo para realizar a troca destes dados dinamicamente.

As redes de computadores, e especialmente a Internet, têm motivado o intercâmbio de dados entre sistemas, mas a ênfase a este intercâmbio se dá entre programas em máquinas distintas. Além disto, os principais envolvidos são algum tipo de *software* de base (sistemas operacionais, bancos de dados, etc.) ou ferramentas diretamente envolvidas com a rede (navegação, comunicação, transferência de arquivos, etc.).

Como trataremos mais adiante, recentes iniciativas – especialmente aquelas relacionadas à introdução da linguagem XML [Bray 2000] - têm alterado gradativamente este panorama.

Em grande parte dos casos, desejamos mais que o intercâmbio de dados, estamos interessados em uma integração de processos, ou seja, um certo módulo de *software* deseja não apenas trocar dados com outro módulo, mas utilizar seus serviços como parte de sua tarefa, ou então trabalhar em colaboração em uma tarefa que é resultante da interação de ambos. Isto significa que os dois módulos precisam trabalhar em conjunto, como se pertencessem ao mesmo programa.

Entre as estratégias para integração de processos, alguns sistemas operacionais dispõem de mecanismos que permitem a integração de módulos, objetos ou componentes de *software*, que são projetados a partir da estrutura específica do S.O.

Uma das tecnologias que tem obtido grande sucesso em integração está baseada no paradigma da orientação a objetos. Padrões abertos como o CORBA [Vinoski 1997] – *Common Object Request Broker Architecture* – estruturam a aplicação na forma de objetos, que estabelecem interfaces públicas através das quais podem se comunicar.

A integração entre processos pode se originar da necessidade de uma organização, que isoladamente deseja integrar suas aplicações (geralmente em ambientes distribuídos), ou pode ser fruto de uma necessidade coletiva mais ampla de construir produtos em colaboração, onde integrar é condição necessária para a soma dos esforços e a combinação de resultados. Esta segunda situação é a que nos interessa neste trabalho. Ela exige um deslocamento de uma ótica de construção de aplicações monolíticas, para um enfoque baseado na produção de componentes de *software*, passíveis de combinação em diferentes configurações e, por este motivo, ideais para a colaboração. Os componentes de *software* são discutidos em mais detalhes no tópico 2.2.

## 2.1. Tecnologias de marcação na *Web*

XML é uma linguagem de marcação que possui raízes em SGML (*Standard Generalized Markup Language*). Ela tem a capacidade de, através do uso de marcadores (caracterizados pelos símbolos "<" e ">"), agregar informações adicionais a documentos, com os mais diversos propósitos. A expressão

**<indivíduo>** **<nome>**Gaspar**</nome>** tem **<peso>**82**</peso>** quilos. **</indivíduo>**

permite diferenciar o conteúdo da frase "Gaspar tem 82 quilos." das marcações (<indivíduo>, <nome> e <peso>) que agregam informações adicionais à frase. Estas informações são facilmente extraídas e isoladas por programas de computador.

Grande parte dos princípios de XML resulta de uma experiência anterior de grande sucesso, o HTML, que se mostrou adequado para diferentes plataformas e sistemas que compõem a Internet. XML amplia os horizontes, lançando-se como linguagem base não apenas para a descrição de páginas *Web*, mas de qualquer conteúdo registrado e trocado entre aplicações. Para isto, ela possui recursos de metalinguagem, que permitem a criação de linguagens baseadas em XML para domínios específicos de aplicação.

A proposta do XML se expande também para comunicação dinâmica entre aplicações, como acontece no SOAP [Williams 01] – *Simple Object Access Protocol* – cujo objetivo é a comunicação entre aplicações em ambientes distribuídos e heterogêneos através da transferência de mensagens codificadas em pacotes XML.

Quando se trata de trocar informações entre aplicações, é importante estabelecer não apenas o modo de se representar o conteúdo, mas a forma

como este conteúdo será interpretado. Em linguagens XML definidas para domínios específicos, a interpretação do conteúdo é definida pela própria comunidade que a criou.

XML cumpre em grande parte dos casos a função da descrição de metadados, ou seja, reúnem um conjunto de dados cujo objetivo é a descrição de outros dados. Com o propósito de estruturar a estratégia utilizada para a codificação, troca e reutilização de metadados em XML, foi elaborado RDF [Lassila 1999] – *Resource Description Framework*. Para descrever um recurso, RDF utiliza uma instrução composta de três partes: recurso, tipo de propriedade e valor.

**[http://www.bib.org/livro.xml] –Autor→ "Gaspar"**

Neste caso, o Autor (tipo de propriedade) do livro definido no recurso http://www.bib.org/livro.xml é Gaspar (valor).

## 2.2. Componentes de *Software*

Em outubro de 1968, quando a engenharia de *software* estava dando seus primeiros passos, ao falar sobre os componentes de *software*, Mcilroy afirmou:

"Sem dúvida nós produzimos *software* com técnicas ultrapassadas. Sem dúvida nós alcançaremos rapidamente o fim da linha no confronto com o pessoal de *hardware*, porque eles são industrialistas e nós somos fazendeiros. A produção de *software* hoje em dia apresenta-se, na escala de industrialização, em algum lugar abaixo das mais atrasadas indústrias de construção." [Mcilroy 1968]

Para Mcilroy, os componentes de *software* se apresentaram como um caminho para superar a crise. Ele faz uma analogia com o uso de peças pré-fabricadas, muito comum em diversos campos da engenharia.

Reunindo características de reusabilidade, facilidade de distribuição, extensibilidade, facilidade de configuração e combinação, produzir um sistema 'orientado a componentes' significa uma mudança de perspectiva. Ao invés de se codificarem novos produtos a partir da estaca zero, eles são construídos a partir da combinação de componentes de *software*.

A tecnologia dos componentes evoluiu muito nos últimos trinta anos ou mais. Todavia, Szperski observa que apenas alguns tipos de *software* de base, como sistemas operacionais e bancos de dados, alcançaram altos níveis de maturidade [Szperski 1999].

No desenvolvimento de *software* educacional, observamos que houve uma evolução nas ferramentas genéricas para programação, que combinam a tecnologia de componentes com interfaces visuais de construção. Por outro lado, no domínio específico da educação, a prática na construção de componentes 'educacionais' ainda é inexpressiva, conseqüentemente, multiplicamos esforços na solução dos mesmos problemas.

Em matemática reconstruímos inúmeras vezes traçadores de curvas e módulos de resolução de expressões; em sistemas de educação à distância multiplicamos os sistemas de correio eletrônico, fórum e acompanhamento do aluno; e assim por diante.

## 2.3. Enfoques de integração

A integração de *software* educacional pode ocorrer sob dois enfoques:

- integração de ferramentas;
- integração de resultados.

Ao tratar do futuro da educação on-line, Downes afirma:

"*O modelo predominante para o projeto de cursos irá se assemelhar à arquitetura dos computadores modernos. Haverá um backbone (espinha dorsal), análogo à placa mãe do computador, que estabelece a estrutura básica do curso. No backbone serão conectados vários módulos de estudo, ferramentas de comunicação, e sistemas de informações do estudante.*" [Downes 1998]

Segundo Downes, ao contrário da estrutura predominante dos sistemas para cursos on-line, no futuro haverá fabricantes especializados na construção de módulos com tarefas específicas, tal como e-mail, *chat*, apresentação de simulações, etc. Estes módulos se integrarão através da troca de 'objetos educacionais', conforme padrão LOM – *Learning Object Metadata* – [Hodgins 2002] e sua respectiva versão XML definida pelo IMS [Anderson 2001].

Deste modo nos beneficiaremos com a combinação das melhores soluções em cada área e o aluno terá a liberdade de escolher a opção de

módulo que mais lhe aprouver, para o desenvolvimento de determinada tarefa.

O enfoque dado por Downes à integração de *software* educacional é o que classificamos como 'integração de ferramentas'.

Por outro lado, existe um segundo enfoque que é adotado pelo projeto ESCOT – *Educational Software Components of Tomorrow* – no qual ferramentas de *software* educacional já existentes, que não possuem necessariamente uma estrutura interna construída a partir de uma perspectiva de integração, são capazes de gerar produtos na forma de componentes de *software* aptos à integração [Roschelle 1999].

As referidas ferramentas consistem em sistemas, a partir dos quais um usuário produz algum tipo de resultado na forma de *software*. Por exemplo, ferramentas de: autoria em geral, de simulação ou para a criação de modelos matemáticos. No caso do ESCOT, os componentes gerados pela ferramenta devem estar implementados como componentes Java (JabaBeans), ainda que as próprias ferramentas sejam codificadas em outras linguagens, tais como LISP e C.

Classificamos este segundo enfoque como 'integração de resultados'. Nele, o produto resultante de ferramentas especializadas é matéria prima para integração.

## 3. Articulando estratégias para Integração

Ao tratarmos da integração de *software* no tópico anterior, abordamos muitos aspectos ligados principalmente a uma perspectiva de indústria de *software*. Isto aparentemente relega a questão a instituições privadas, especializadas na produção e comercialização de *software*, dado que outras instituições, como as de ensino e pesquisa, não têm como principal objetivo a construção de produtos com acabamento comercial, mas miram sobretudo suprir uma necessidade própria – mormente nas instituições de ensino – ou explorar as fronteiras da inovação – principalmente nas instituições de pesquisa.

Se por um lado parece um contra-senso falar em construção de *software*, utilizando, no contexto das instituições de ensino e pesquisa, técnicas de engenharia, apropriadas para a produção no contexto industrial, por outro, o *software* produzido nessas instituições é uma das mais ricas e inovadoras fontes disponíveis. A maior parte daqueles que produzem *software* nesse contexto pensam na possibilidade de seu uso.

Isto significa que, ao contrário de muitos domínios de conhecimento, o *software* educacional produzido em qualquer contexto, mesmo que experimentalmente, é sempre um potencial produto de aplicação prática.

Entretanto, a grande ênfase dada a um produto de *software* final isolado que atenda a uma necessidade, comprove ou demonstre uma inovação, relega a segundo plano a dimensão coletiva, no seu sentido mais amplo, que implica em 'produzir para colaborar'.

Este último modo de produzir é certamente mais trabalhoso, como observa Szperski ao discutir sobre a construção de componentes: "A tecnologia do componente de *software* é complexa de dominar e viabilizar, soluções baseadas em componentes só se desenvolvem se os benefícios estiverem claros" [Szperski 1999].

Antes de iniciar a produção de um novo *software*, algumas questões merecem especial atenção:

- Se estou construindo um pequeno *software*: como espero que o usuário dedique seu tempo a esta pequena peça, que tem seu modo particular de ser instalada, configurada e manuseada, em meio a tantas outras?

- Se estou construindo um grande sistema: será ele uma grande ilha isolada de todo o restante?

- Construindo um programa de computador como um único e grande bloco monolítico, não estarei desperdiçando todo o trabalho na solução de um problema, enquanto partes de meu sistema também são pedaços da solução de muitos outros problemas?

A questão que pretendemos tratar a seguir é: "Como podemos articular as diversas tecnologias que têm convergido para a integração em um modelo adequado ao desenvolvimento de *software* educacional?"

Este foi o desafio do projeto *Anima* [Santanchè 2001] [Santanchè 2002], que possui como uma de suas principais metas a integração de *software*, independente de plataforma ou implementação.

*Anima* se originou a partir de um ambiente destinado à construção de aplicações educacionais, denominado Casa Mágica [Santanchè 1999]. Conforme a classificação realizada no tópico 2.3, *Anima* inicialmente se concentrou em um enfoque de 'integração dos resultados' produzidos pelo sistema Casa Mágica.

Um projeto feito em Casa Mágica é organizado na forma de componentes de *software* intercomunicantes. A tarefa de *Anima* foi estabelecer um modelo, que permitisse a integração destes componentes com quaisquer outros, independente de sua origem, linguagem ou implementação.

O modelo definido por *Anima* se organiza sob a forma de objetos intercomunicantes. Os componentes de *software* são tomados como um tipo particular de objeto, deste modo, obtemos uma representação homogênea para o sistema completo.

Apesar de ter partido de um enfoque centrado na 'integração dos resultados', o modelo definido para *Anima* pode igualmente ser aplicado na construção de ferramentas.

A combinação de tecnologias para integração, utilizadas na concepção do modelo e na construção das ferramentas do projeto *Anima*, servirá de matéria prima para a discussão referente à articulação de estratégias de integração, que desenvolvemos a seguir.

## 3.1. Deslocamento para camada neutra

Linguagens de programação e sistemas computacionais utilizam formatos particulares para a representação de dados, no que diz respeito a seus programas, dados armazenados ou aqueles utilizados para a comunicação entre partes do programa.

Um dos segredos da integração consiste na crescente possibilidade de padronizar esses dados. Quanto mais dados puderem ser compreendidos pelas diversas aplicações, mais facilidade teremos de integrar.

Isto não é verdade apenas quando estamos falando de intercâmbio de dados. A integração de processos envolve a criação de mecanismos para que dois módulos possam se comunicar e, deste modo, trabalhem em conjunto. Este é um dos papéis fundamentais das interfaces entre objetos CORBA, por exemplo.

Conforme apresentamos, XML tem-se mostrado a base ideal sobre a qual se podem construir padrões de representação. Sua estrutura aberta, neutra e extensível não está suscetível a incompatibilidades de formatos internos de representação e a torna ideal para uso em qualquer plataforma, linguagem ou sistema.

Por este motivo, um dos princípios sobre o qual está estruturado *Anima*, consiste no deslocamento de dados utilizados por sistemas e linguagens baseados em objetos – que originalmente são representados em um formato particular – para uma camada neutra, independente de plataforma e implementação (Figura 1). Esta camada utiliza XML como base para representação.

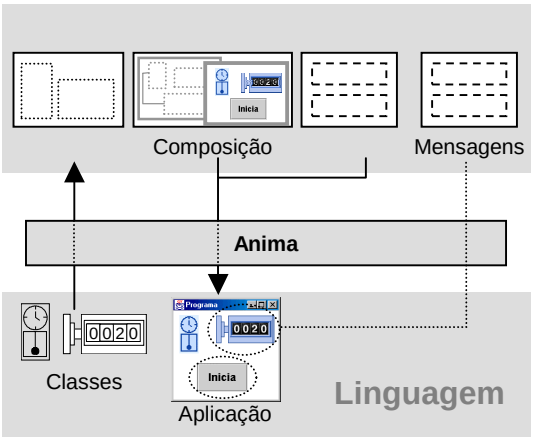


**Figura 1** – Deslocamento de representações para uma camada neutra.

Isto implica no desafio de identificar que dados são candidatos a este deslocamento.

Tomando como base um módulo de *software*, cujos dados desejamos deslocar para uma camada neutra, verificaremos a existência de dados que dizem respeito à representação do módulo de *software* propriamente dito – código do programa e dados associados – e outros representados apenas em tempo de execução, como as mensagens trocadas entre objetos. O modelo abrange ambas as situações.

Em sistemas orientados a objetos, que são alvo do modelo *Anima*, o módulo de *software* é codificado como um conjunto de objetos que pertencem a classes.

Estas classes representam os aspectos estáticos, compartilhados por conjuntos de objetos. Objetos da mesma classe se diferenciam pela sua

identidade e pelo seu estado, representado através de valores de seus atributos.

A separação entre classes e objetos, permite representar objetos através dos dados que os distinguem, na camada neutra em XML, mantendo a implementação de sua classe codificada na linguagem de programação original.

É importante ressaltar que estamos trabalhando em um padrão de integração, não de uniformidade. Por este motivo, *Anima* não estabelece uma única linguagem de programação, sob a qual devem estar codificados todas as classes e objetos; ao contrário, o modelo respeita a particularidade de cada linguagem e delimita os aspectos que podem ser representados de forma neutra.

Uma aplicação é representada, no modelo *Anima*, como a captura de uma configuração particular de um conjunto de objetos e o modo como eles estão interligados. Um recurso denominado Composição (Figura 1), contém a representação XML tanto da configuração dos objetos, quanto de sua interligação. Ele funciona como um espelho que reflete a constituição de uma aplicação em uma superfície neutra.

Estes 'espelhos' são capazes de circular livremente pela Internet, como documentos XML. Ao alcançarem o destino onde devem assumir a forma de uma aplicação, são utilizados mecanismos de conversão (Figura 1), baseados na linguagem XSLT [Adler 2001] – XSL *Transformations* – que é uma linguagem XML cuja função é realizar a transformação de um documento XML em uma saída diferente.

A representação neutra dos objetos e das ligações é importante quando o enfoque for na 'integração dos resultados', pois ela é um padrão de resultado a ser seguido pelas ferramentas de produção.

Se tomarmos os produtos de uma ferramenta de *software* educacional na forma de componentes, passíveis de serem integrados com outros, torna-se necessário um registro na camada neutra de informações associadas a:

- sua classe (tais como atributos que os caracterizam ou mensagens a que os objetos respondem);
- interfaces de que dispõem para sua integração;
- outros metadados.especialmente aqueles que os caracterizam como 'objetos educacionais', tal como foi definido pelo LOM [Hodgins 2002].

No modelo proposto isto é feito através de um recurso denominado CIM (Classe, Interface e Metadados – Figura 1).

Por se tratar de um conjunto de metadados, a descrição CIM utiliza RDF. Os metadados educacionais seguem uma especificação elaborada pelo IMS, ainda em processo de construção, que utiliza RDF para a representação do LOM [Anderson 2001].

No que diz respeito aos dados representados em tempo de execução, para fins de integração, interessa-nos especificamente a comunicação que é realizada pelos objetos em uma aplicação. Em geral, cada linguagem ou sistema adota uma estratégia e formato específicos para realizar a comunicação entre seus objetos.

Para viabilizar a comunicação entre objetos, independente de sua linguagem ou sistema, estas mensagens foram transferidas para a camada neutra em XML (Figura 1).

Cada classe de objetos, que irá se comunicar utilizando as mensagens XML, adiciona uma interface de conversão, cuja função é transformar as mensagens do formato interno para o externo, e vice-versa.

Os objetos de uma mesma aplicação são agrupados em um espaço de trabalho. Para cada espaço de trabalho existe um módulo denominado Mediador, responsável pelo trânsito de mensagens entre os objetos. Como objetos de diferentes linguagens ou sistemas utilizam espaços de trabalho distintos, os Mediadores destes espaços de trabalho estão habilitados a se comunicar e atuam como *proxy*, transferindo mensagens entre objetos de espaços distintos.

## 3.2. Integração de Resultados

Se por um lado é interessante a construção de *software* educacional sob a lógica dos componentes, por outro, a grande variedade de linguagens e sistemas disponíveis na Internet pode nos conduzir à criação de um extenso repositório destes componentes, incapazes de se integrarem.

A perspectiva do deslocamento para uma camada neutra, apresentada no tópico anterior, busca estender uma experiência bem sucedida no uso de uma linguagem de marcação como o HTML (cujos princípios foram adotados na definição do XML), que foi capaz de unificar um grande parque de plataformas e sistemas distintos, para o contexto da construção de aplicações.

Deslocar apenas uma parte da aplicação para uma camada neutra, parece uma solução incompleta. Somos levados a acreditar que a parte da aplicação – especificamente a implementação das classes – que se mantém em código nativo da linguagem de programação, irá ancorar o módulo na sua plataforma e no sistema de origem.

Para compreendermos como a combinação genérico/específica pode ser bem sucedida, vamos novamente estabelecer uma analogia com o HTML.

Um documento HTML representa em seu conteúdo, basicamente, texto, além das marcações que estabelecem sua estrutura e formato. Quando desejamos incluir em uma página HTML outras mídias, utilizamos marcadores que fazem referências a recursos externos, como imagens, vídeos e animações.

Estas mídias adicionais atuam como objetos estrangeiros, cujo conteúdo está descrito em um formato particular, diferente do HTML. A tarefa dos navegadores, que apresentam as páginas, é de estabelecer mecanismos capazes de apresentar os objetos estrangeiros referenciados.

Analogamente, a implementação de um componente de *software* atua como um objeto estrangeiro. A questão passa a ser: como executar o código que implementa um componente de *software* em diferentes plataformas e sistemas, tal como se processa com a apresentação das mídias nos navegadores *Web*?

A resposta a esta questão está nas linguagens de código móvel (*Mobile Code Language* – MCL). Elas se baseiam no princípio da mobilidade de código, que "pode ser definida como a capacidade de modificar dinamicamente as ligações entre fragmentos de código e o local onde eles são executados" [Fuggeta 1998]. Isto permite que componentes de *software* possam trafegar pela rede livremente, sendo executados em diferentes plataformas e sistemas.

Combinadas, as classes produzidas em linguagens de código móvel e representações CIM, resultam em pacotes altamente autônomos e portáveis, não apenas no que diz respeito a sua execução, como também em relação às informações necessárias para sua compreensão e seu uso.

Do ponto de vista da 'integração de resultados', representados na forma de composições, os objetos que as compõem poderão vir acompanhados de documentação, através da representação CIM da classe. Isto permite o entendimento das peças de uma composição independente do ambiente que as produziu.

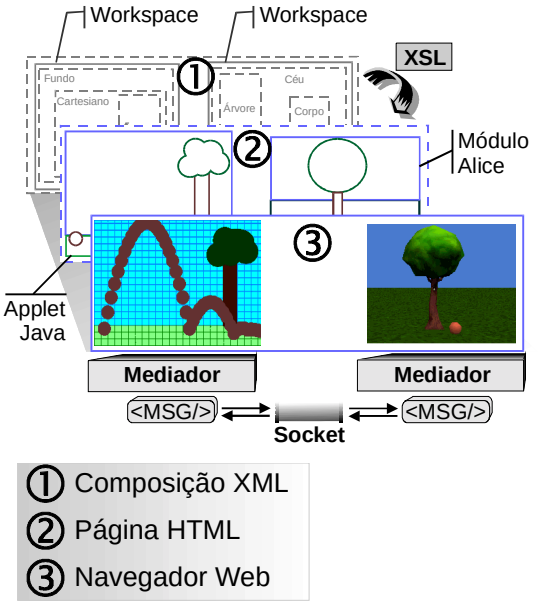


**Figura 2** – Diagrama ilustrando o processo de transformação, que parte da representação XML até o documento apresentado no navegador *Web*.

A Figura 2 ilustra um projeto que vem sendo desenvolvido utilizando *Anima*, cujo propósito é a integração de uma aplicação construída no sistema Casa Mágica com outra elaborada no sistema Alice[1].

O Alice, dirigido a usuários leigos em computação gráfica, consiste em um ambiente tridimensional para a construção de animações. Sua estrutura para organização de modelos e montagem das animações é toda orientada a objetos.

Utilizando *Anima*, a animação de um corpo lançado obliquamente é apresentada de forma sincronizada: em um plano bidimensional que registra seu trajeto (Casa Mágica) e em um espaço tridimensional (Alice).

[1] Página: http://www.alice.org.

Ambas as composições estão descritas em XML. Através do XSL, são transformadas em uma página HTML, que faz referência a uma *applet* Java para a aplicação Casa Mágica (o sistema Casa Mágica é implementado em Java e possui um módulo *runtime* que é executado na forma de *applet*), e a um *plugin* Alice para a animação tridimensional (o Alice possui um *plugin* que permite a execução de suas animações).

Cada módulo possui seu próprio Mediador, escrito na respectiva linguagem de programação; os mediadores repassam mensagens de seus objetos através de um *socket* TCP/IP.

O navegador *Web* constitui um excelente espaço para a integração de aplicações, a cada dia um número crescente de linguagens e sistemas desenvolve mecanismos, na forma de *plugins*, para permitir que seus programas possam ser executados em navegadores *Web*. Segundo diSessa:

"Componentes são usados em um 'ambiente container' (como um navegador *Web* ou um processador de textos) que provê recursos genéricos (navegação, *bookmarking*, organização de texto e layout) para aumentar os recursos dos componentes de modo a alcançar um efetivo ambiente de trabalho". [diSessa 2001]

## 3.3. Integração de Ferramentas

Apesar do projeto *Anima* ter sido concebido inicialmente dentro do enfoque de 'integração de resultados', seu modelo é igualmente aplicável ao enfoque de 'integração de ferramentas'.

Vamos ilustrar como esta integração pode acontecer com *Anima*, utilizando a organização modular de cursos on-line prevista por Downes [Downes 1998] para o futuro.

A integração de componentes de porte médio e grande, em um sistema para gerenciar cursos on-line, exige uma extensão do modelo definido em *Anima*. Neste caso, será necessária uma especial atenção aos objetos de dados que irão circular entre os componentes.

Em um sistema deste porte, muitas mensagens que circularão entre os componentes conterão estruturas altamente especializadas, tais como: mensagens de correio eletrônico e fórum, respostas dadas por um aluno a uma avaliação, informações cadastrais de um aluno, etc.

Por este motivo, além da estrutura de integração proporcionada por *Anima*, faz-se necessária a definição de um padrão para a representação dos objetos de informação que serão registrados e transmitidos, tal como está descrito em [Santanchè 1999-2], que utiliza RDF para esta tarefa.

As mensagens *Anima* em XML, que serão transmitidas entre módulos, podem fazer uso de uma estrutura de classificação baseada em esquemas RDF, que será comum a todos os módulos. Isto garante uma estrutura e interpretação universais para o conteúdo das mensagens.

Além de estabelecer uma rica infra-estrutura para a definição de esquemas, com recursos de classes semelhantes aos utilizados na orientação a objetos, RDF possui um poderoso mecanismo de extensão baseado na herança, que permite a criação de novos esquemas, aproveitando aspectos de estrutura e semântica de esquemas já existentes.

## 4. Considerações Finais

Se, por um lado, há uma abundância de *software* educacional, por outro, tem havido uma carência de iniciativas em direção a um trabalho amplamente colaborativo, de modo que os esforços possam ser combinados e os resultados usufruídos por todos e também contextualizados, de acordo com as peculiaridades de cada grupo ou região.

A fragmentação das iniciativas conduz não apenas a uma repetição de esforços para a solução do mesmo problema, como também divide projetos futuros instalados sobre bases diferentes.

Hoje em dia, por exemplo, contamos com diversos sistemas destinados ao gerenciamento de cursos à distância, desenvolvidos por instituições de ensino e pesquisa, dos quais podemos destacar o AulaNet[2], desenvolvido pelo Laboratório de Engenharia de Software da PUC-Rio, o TelEduc[3], desenvolvido pelo Núcleo de Informática Aplicada à Educação (Nied) da Unicamp e o

[2] Página: http://anauel.cead.puc-rio.br/aulanet2/.
[3] Página: http://hera.nied.unicamp.br/teleduc/.

LearnLoop[4], traduzido pelo Núcleo Avançado de Computação Sônica e Multimídia da Universidade Federal de Uberlândia.

Apesar de alguns disporem de código fonte aberto, eles não definem mecanismos de integração entre si. Como conseqüência, projetos de pesquisa desenvolvidos sobre esses ambientes – como a "Categorização e Estruturação de Mensagens Textuais em Cursos", para o AulaNet [Fucks 2002] e do "Ambiente de Autoria de Cursos à Distância", para o TelEduc [Tessarollo 2000] – limitam seus benefícios ao sistema em que foram desenvolvidos, ainda que seus princípios possam futuramente ser estendidos a outros sistemas, com um esforço multiplicado de codificação.

Adicionalmente, conforme abordamos no início deste trabalho, a fragmentação de programas exige do usuário um esforço muito grande para trabalhar com sistemas que se apresentam de várias maneiras e, além disso, não se integram.

A proposta prática apresentada, baseada em *Anima*, não tem a intenção de se configurar como um modelo acabado e definitivo para a integração entre sistemas. Seu principal objetivo é mostrar a riqueza de recursos de que dispomos para integração e como eles podem atuar de forma articulada para alcançarmos nosso objetivo. A partir deste rascunho inicial, podemos atuar em colaboração, na busca de um modelo de integração adequado à comunidade como um todo.

## 5. Referências


Adler, Sharon et al. Extensible Stylesheet Language (XSL) Version 1.0 – W3C Recommendation 15 October 2001. W3C - World Wide Web Consortium, 2001 [online] http://www.w3.org/TR/xsl/.

Anderson, Thor & McKell, Mark. IMS Learning Resource Meta-data - Version 1.2 Final Specification. IMS Global Learning Consortium, Inc., May 17, 2001 [online] http://www.imsproject.org/metadata/imsmdv1p2/imsmd_infov1p2.html.

Bastiaan, Martin Koning. Connected and Scalable: A revolutionary Structure for Online Communities. EOE – Educational Object Economy, (sem data).

Bray, Tim et al. (editor). Extensible Markup Language (XML) 1.0 (Second Edition) - W3C Recommendation 6 October 2000. W3C - World Wide Web Consortium, October 6, 2000 [online] http://www.w3.org/TR/2000/REC-xml-20001006.

diSessa, Andrea A. The Web/comp project – Proposal. School of Education, University of California, Berkeley, 2001 [online] http://www.soe.berkeley.edu/~boxer/webcomp/proposal.html.

Downes, Stephen. The Future of Online Learning. Online Journal of Distance Learning Administration, Volume I, Number 3, Fall 1998, State University of West Georgia, Distance Education Center [online] http://www.westga.edu/~distance/downes13.html.

Fuks, Hugo. Aprendizagem e Trabalho Cooperativo no Ambiente AulaNet. Rio de Janeiro: Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro, Fevereiro de 2000 [online] http://139.82.193.20/aulanet2/misc/aprendizagem.pdf.

Fuggeta, Alfonso & Picco, Gian Pietro & Vigna, Giovanni. Understanding Code Mobility. IEEE Transactions on Software Engineering, volume 24, 1998 [online] http://www.elet.polimi.it/Users/DEI/Sections/Compeng/GianPietro.Picco/papers/tse98.ps.gz.

Hodgins, Wayne (chair). Draft Standard for Learning Object Metadata. IEEE Learning Technology Standards Committee (LTSC), January 18, 2002 [online] http://ltsc.ieee.org/doc/wg12/LOM_WD6_3a.pdf.

Lassila, Ora & Swick, Ralph R. Resource Description Framework (RDF) Model and Syntax Specification - W3C Recommendation 22 February 1999, W3C - World Wide Web Consortium (W3C), February 22, 1999 [online] http://www.w3.org/TR/REC-rdf-syntax/.

Mcilroy, M. D. Mass Produced Software Components in Naur, P. & Randell, B. (Eds.). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968 [online] http://www.cs.ncl.ac.uk/people/brian.randell/home.formal/NATO/nato1968.PDF.

Roschelle, Jeremy et al. Developing Educational Software Components. IEEE Computer Society – Computer, volume 32, number 9, September, 1999.


[4] Página da versão nacional: http://www.ead.ufu.br/learnloop/.

Santanchè, André & Teixeira, Cesar Augusto Camillo. Integrando Instrucionismo e Construcionismo em Aplicações Educacionais através do Casa Mágica. V Workshop de Informática na Escola – XIX Congresso da SBC, 20 de Julho de 1999 [online] http://www.geocities.com/santanche/publicado/WIE99.pdf.

Santanchè, André & Teixeira, Cesar Augusto Camillo. Explorando Linguagens de *Markup* Extensíveis na Construção de Sistemas de Educação Baseados na *Web*. XXVI Seminário Integrado de Software e Hardware – XIX Congresso da SBC, 20 de Julho de 1999 [online] http://www.geocities.com/santanche/publicado/SEMISH1999.pdf.

Santanchè, André & Teixeira, Cesar Augusto Camillo. *Anima*: Promovendo Integração de Componentes na *Web*. VII Simpósio Brasileiro de Multimídia e Hipermídia, 16 de julho de 2001 [online] http://www.geocities.com/santanche/publicado/SBMidia2001-WTD.pdf.

Santanchè, André. Anima: Representando e Integrando Objetos Educacionais na *Web*. Universidade Salvador – UNIFACS – Dissertação de Mestrado, 2002.

Szperski, Clemens. Components and Objects Together. Software Development Magazine, 1999 [online] http://www.sdmagazine.com/documents/s=758/sdm9905b/.

Tessarollo, Marcia Renata Matero. AutorWeb - Ambiente de Autoria de Cursos à Distância. Dissertação de Mestrado. Unicamp – Universidade Estadual de Campinas, 2000 [online] http://hera.nied.unicamp.br/teleduc/publicacoes/marcia_disser.pdf.

Vinoski, Steve. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communications Magazine, volume 35, number 2, February 1997 [online] http://www.infosys.tuwien.ac.at/Research/Corba/archive/intro/Vinoski-IEEE-CM.pdf.

Williams, Stuart & Jones, Mark (editors). XML Protocol Abstract Model –W3C Working Draft 9 July 2001. W3C - World Wide Web Consortium, 2001 [online] http://www.w3.org/TR/xmlp-am/.