

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA  
COMPUTAÇÃO**

**Carlos Alberto Zorzo**

**UMA ARQUITETURA PARA  
INTEGRAÇÃO DE APLICAÇÕES INTER-EMPRESA  
BASEADA EM AGENTES**

Dissertação submetida à Universidade Federal de Santa Catarina com parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

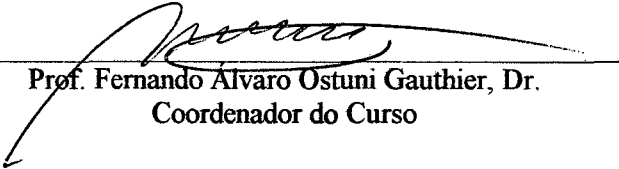
**João Bosco Manguiera Sobral**

Florianópolis, Dezembro de 2001.

# UMA ARQUITETURA PARA INTEGRAÇÃO DE APLICAÇÕES INTER-EMPRESA BASEADA EM AGENTES

Carlos Alberto Zorzo

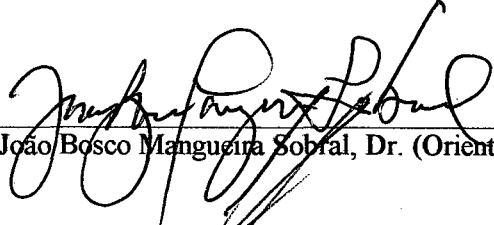
Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.



---

Prof. Fernando Alvaro Ostuni Gauthier, Dr.  
Coordenador do Curso

Banca Examinadora:

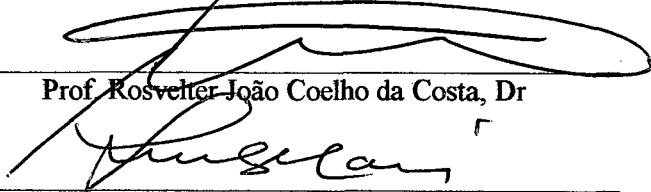


---

Prof. João Bosco Manguiera Sobral, Dr. (Orientador)

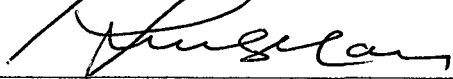
---

Prof. Hugo Fuks, PhD



---

Prof. Rosvelter João Coelho da Costa, Dr



---

Prof. Luiz Fernando Jacintho Maia, Dr

À Universidade Federal de Santa Catarina.  
À Associação Catarinense de Fundações Educacionais – ACADE.  
À Universidade do Contestado – Campus Caçador.  
Ao orientador Prof. Dr. João Bosco Manguiera Sobral,  
pelo acompanhamento pontual e competente.  
Aos professores e funcionários do Programa de  
Pós-Graduação em Ciência da Computação.  
A minha esposa, Patrícia, pelo apoio constante.  
A meus familiares e amigos.

A todos que direta ou indiretamente  
contribuíram para a realização  
deste trabalho.

## Sumário

<b>Lista de Figuras</b> .....	vi
<b>Lista de Tabelas</b> .....	vii
<b>Resumo</b> .....	viii
<b>Abstract</b> .....	ix
<b>1 INTRODUÇÃO</b> .....	1
<b>2 INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS</b> .....	4
<b>2.1 Definição</b> .....	4
<b>2.2 Histórico e Evolução</b> .....	6
<b>2.3 Tipos de Integração</b> .....	9
2.3.1 Integração ao Nível de Dados.....	10
2.3.2 Integração Funcional.....	11
2.3.3 Integração ao Nível de Interface de Usuário .....	13
<b>2.4 Tecnologias para Integração</b> .....	13
2.4.1 Chamada de Procedimentos Remotos (RPC) .....	14
2.4.2 Middleware Orientado a Mensagens .....	15
2.4.3 Objetos Distribuídos .....	16
2.4.4 Servidores de Aplicação .....	18
2.4.5 Message Brokers.....	19
<b>2.5 Perspectivas e Tendências</b> .....	20
<b>3 AGENTES DE SOFTWARE</b> .....	23
<b>3.1 Conceituação</b> .....	23
<b>3.2 Atributos de um Agente</b> .....	24
<b>3.3 Tipos de Agentes</b> .....	25
3.3.1 Agentes Colaborativos.....	26
3.3.2 Agentes de Interface .....	26
3.3.3 Agentes de Recuperação de Informação.....	26
<b>3.4 Comunicação entre Agentes</b> .....	27
<b>3.5 Aplicações de Agentes</b> .....	31
<b>4 A LINGUAGEM EXTENSÍVEL DE MARCAÇÃO (XML)</b> .....	34
<b>4.1 Histórico e Conceito</b> .....	34
<b>4.2 Especificação XML</b> .....	40
4.2.1 Documentos .....	40
4.2.2 Definição de Tipo de Documento (DTD).....	43
4.2.3 Validação .....	44
4.2.3.1 Documentos Bem Formatados.....	45
4.2.3.2 Documentos Válidos.....	45
4.2.4 Processadores XML.....	45
<b>4.3 Linguagem de Folha de Estilos e Transformações (CSS e XSL)</b> .....	46
<b>4.4 Linguagem de Ligação e Ponteiros (XLink e XPointer)</b> .....	48
<b>4.5 Modelo de Objetos do Documento (DOM)</b> .....	49
<b>4.6 Espaço de Nomes (XML Namespaces)</b> .....	50
<b>4.7 Esquemas (XML Schema)</b> .....	50
<b>4.8 Aplicações de XML</b> .....	51

<b>5 ARQUITETURA DE INTEGRAÇÃO DE APLICAÇÕES</b> .....	54
<b>5.1 Fundamentação</b> .....	54
<b>5.2 Camadas da Arquitetura</b> .....	56
<b>5.3 Elementos Básicos</b> .....	57
5.3.1 Processos Virtuais de Negócio .....	57
5.3.2 Agentes de Processo .....	59
5.3.3 Agentes de Interface de Aplicação .....	60
5.3.4 Protocolos de Comunicação.....	60
5.3.5 Objetos de Dados .....	61
<b>5.4 Sistemática de Funcionamento</b> .....	62
<b>5.5 Viabilidade da Arquitetura</b> .....	63
5.5.1 Processo Virtual.....	63
5.5.2 Objeto de Dados.....	64
5.5.3 Protocolo de Comunicação .....	65
5.5.4 Agente de Processo.....	67
5.5.5 Agente de Interface de Aplicação .....	69
5.5.6 Execução.....	71
<b>5.6 Avaliação e Perspectivas</b> .....	74
<b>6 CONCLUSÃO</b> .....	76
<b>7 BIBLIOGRAFIA</b> .....	78
<b>8 ANEXOS</b> .....	83
<b>8.1 ANEXO 1 - Código Fonte do Agente de Processo</b> .....	84
<b>8.1 ANEXO 2 - Código Fonte do Agente de Interface de Aplicação</b> .....	88

## Lista de Figuras

Figura 1: Estrutura em camadas da arquitetura de integração proposta .....	56
Figura 2: Protocolo de comunicação do processo virtual transferência de alunos .....	66
Figura 3: Interface de instanciação de agentes do ambiente Saci.....	73
Figura 4: Interface do processo virtual de transferência de alunos.....	73

## Lista de Tabelas

Tabela 1: Categorias e performativas KQML.....	30
Tabela 2: Mensagens KQML do protocolo de comunicação do processo virtual de transferência de alunos .....	67

## Resumo

A Integração de Aplicações Empresariais (EAI) é uma resposta da Tecnologia da Informação aos problemas verificados nas organizações a partir de mudanças tecnológicas, organizacionais e de mercado. Do ponto de vista tecnológico verificou-se o surgimento de novos paradigmas de desenvolvimento e de plataformas menores e mais abertas, gerando um acúmulo de tecnologias heterogêneas e de soluções individualizadas. Do ponto de vista organizacional a evolução natural dos ambientes empresariais implicou no surgimento de processos funcionais organizacionais mais complexos, o que elevou sobremaneira a complexidade das aplicações e a necessidade de compartilhar informações que residiam naquelas soluções individualizadas. Finalmente, do ponto de vista do mercado, o surgimento do *e-business*, que aumentou a necessidade das aplicações dialogarem entre si, o grande número de aquisições, fusões e incorporações de empresas e a migração dos sistemas para a Web tornou imperativa a questão da Integração de Aplicações Empresariais. Uma vez que a pura e simples troca das aplicações atualmente existente nas organizações se tornar inviável por questões de tempo e custo proibitivos, a EAI se coloca como solução uma vez que permite às aplicações díspares compartilhar dados e processos de negócios, quer seja dentro da organização, quer seja entre parceiros comerciais. Desta forma, pode-se fazer uso das estruturas de hardware e software já existentes nas organizações para a integração e desenvolvimento de aplicações que possibilitem novas e inovadoras maneiras de fazer negócios, preservando assim, o investimento já feito pelas organizações. Dentre as várias tecnologias disponíveis, investiga-se o uso de Agentes de Software e da Linguagem XML na integração de aplicações ao nível de dados, bem como o uso destas tecnologias na solução dos problemas de heterogeneidade dos ambientes, independência das aplicações, manutenção da semântica dos dados e a observância de padrões abertos de mercado, inerentes ao contexto da integração. Em suma, imagina-se como solução para a Integração de Aplicações Empresarias um ambiente povoado por agentes de software que se comunicam e cooperam através de mensagens KQML que, por sua vez, encapsulam objetos de dados XML a fim de manter a semântica dos dados compartilhados.



## Abstract

Enterprise Application Integration (EAI) is a reply of the information technology to the problems verified in the organizations from technological and organizational and market changes. From the technological point of view it was verified the rising of new paradigms of development and smaller and openness platforms, generating an accumulation of heterogeneous technologies and individualized solutions. From the organizational point of view the natural evolution of the enterprise environments implied in the rising of more complex organizational functional processes, what excessively raised the complexity of the applications and the necessity to share information that inhabited in those individualized solutions. Finally, from the point of view of the market, the rising of the e-business, that added the necessity of the applications dialogue among themselves, the great number of acquisitions, fusing and incorporation of companies and the migration of the systems to the Web became imperative the question of the Enterprise Application Integration. Since the pure and simple change of applications currently existing in the organizations become impracticable because of questions of prohibitive time and cost, EAI places itself as solution since it allows to the disparate applications to share data and business-oriented processes, on either inside the organization or among commercial partners. This way, it can be made use of the structures of the existing hardware and software in the organizations for the integration and development of applications that make possible new and innovative ways to make businesses, thus preserving, the investment already made by the organizations. Among several available technologies, the use of Software Agents and Language XML in the integration of applications is investigated in the data-level application integration, as well as the use of these technologies in the solution of heterogeneous environment problems, independence of applications, maintenance of the semantics of the data and the observance of open standards in the context of the integration. In short, it is imagined as solution for the Enterprise Application Integration an environment full of software agents that communicate and cooperate themselves through KQML messages that encapsulate XML data objects in order to keep the semantics of the shared data.

# 1 INTRODUÇÃO

A problemática de integração de aplicações empresariais vem tomando corpo dentro das organizações desde o momento em que as aplicações começaram a migrar de plataformas centralizadas para outras, menores e distribuídas. Com o advento destas, proliferaram nas organizações as soluções departamentalizadas e individualizadas, gerando um acúmulo de soluções e tecnologias díspares.

Com o passar do tempo as organizações começaram a se dar conta de que os dados de que elas precisavam para implementar seus processos funcionais normalmente extrapolavam os limites daquelas soluções individualizadas, residindo em várias delas. Tal problema, causado por uma evolução natural dos ambientes e modelos empresariais estabelecidos e que implicou no surgimento de processos funcionais organizacionais mais complexos, gerou um conseqüente aumento na complexidade das soluções computacionais. Mais ultimamente, o incremento do *e-business*, que aumentou a necessidade das aplicações dialogarem entre si, tornou a questão de integração de aplicações imperativa para a sobrevivência das organizações.

Dentro deste contexto, a integração de aplicações empresariais pode ser entendida como o processo de criação de uma infra-estrutura que permita à aplicações díspares compartilhar dados e processos de negócios. Desta forma, pode se fazer uso das estruturas de hardware e software já existentes na organização para o desenvolvimento de aplicações que habilitem novas e inovadoras soluções, preservando, assim, o investimento já feito pelas organizações. Cabe ressaltar, entretanto, que, se por um lado a integração de aplicações empresariais traria ganhos e diferenciais de competitividade para a organização e seus parceiros, por outro sua efetiva implementação esbarra em algumas barreiras, principalmente a existência de arquiteturas complexas e heterogêneas dentro das organizações.

Atualmente, a busca por soluções de integração passa pela utilização de soluções de *middleware* para combinar as funcionalidades e dados das aplicações já existentes nas organizações. Tais soluções passam, então, pelas tecnologias de Chamada Remota

de Procedimentos, *middlewares* orientados a troca e gerenciamento de mensagens, Objetos Distribuídos e Servidores de Aplicação. Cada uma destas tecnologias possui suas particularidades, sendo mais apropriada para a implementação de um dos tipos de integração, quer seja a integração ao nível de dados, a integração funcional ou a integração ao nível de interface de usuário. Dentro deste contexto, a tecnologia de agentes de software ultimamente tem se colocado como potencialmente promissora para a solução dos problemas de integração de aplicações empresariais. Algumas propostas de aplicação desta tecnologia na integração de aplicações empresariais podem ser encontradas em Maamar, Kettani & Sahli (2000), Pan & Tenenbaum (1991) e Berry & Pancarella (1999).

O presente trabalho pretende investigar a aplicabilidade de tecnologias abertas e padronizadas, mais especificamente as tecnologias de agentes de software e da linguagem XML dentro do escopo da integração de aplicações empresariais. Imagina-se com uma solução para os problemas de integração de aplicações empresariais um ambiente povoado por agentes de software que cooperam entre si de modo a propor soluções para tal problema. Além disso, espera-se que o uso da linguagem XML possibilite a manutenção semântica dos dados necessários aos procedimentos de integração. Em suma, o presente trabalho pretende investigar a viabilidade do uso destas tecnologias no desenvolvimento de soluções de integração, sendo, o mesmo, estruturado em 4 capítulos.

No capítulo 2 faz-se um estudo bibliográfico sobre a problemática da integração de aplicações empresariais. Desta forma, pretende-se conceituar a integração de aplicações e fornecer um panorama histórico que justifique a sua necessidade, bem como identificar os tipos de integração e as tecnologias atualmente utilizadas na solução deste problema. Por fim, pretende-se identificar algumas tendências em relação a esta problemática.

O capítulo seguinte aborda a tecnologia de agentes de software, fornecendo subsídios para sua posterior utilização na solução dos problemas de integração de aplicações empresariais. Assim, identifica-se o que são agentes, quais seriam seus atributos, principais tipos de agentes e, finalmente, a questão da comunicação entre eles,

mais especificamente a utilização da linguagem KQML neste processo. Por outro lado, procura-se também identificar algumas áreas mais propícias a utilização desta tecnologia.

No capítulo 4 aborda-se a linguagem XML. Um levantamento bibliográfico foi feito sobre o histórico e o surgimento da linguagem, bem como um apanhado sobre a especificação XML e as tecnologias a ela relacionadas como a sua linguagem de folha de estilos e *linkagem*, a interface DOM para a manipulação de documentos XML e as tecnologias de espaço de nomes e de esquemas de representação de dados.

O capítulo 5 propõe um modelo para uma arquitetura de integração de aplicações empresariais baseada na utilização das tecnologias de agentes de software e da linguagem XML. Inicialmente fundamenta-se e define-se uma estrutura de três camadas que suporte a arquitetura proposta, bem como os cinco blocos básicos que farão parte da mesma: processos virtuais de negócio, agentes de processo, agentes de interface de aplicação, protocolos de comunicação e objetos de dados. Uma posterior implementação, com base em um ambiente de execução de agentes, será feita e documentada para fins de validação das idéias contidas no modelo. Avaliações e perspectivas sobre o modelo proposto encerram o presente capítulo.

Finalmente, far-se-á uma avaliação do trabalho desenvolvido, incluindo-se também, como anexo, o código fonte dos agentes desenvolvidos para fins de validação da arquitetura proposta como solução à problemática de integração de aplicações.

## 2 INTEGRAÇÃO DE APLICAÇÕES EMPRESARIAIS

A Integração de Aplicações Empresariais é um conceito que persegue a área de Tecnologia de Informação das empresas desde os tempos em que começou a proliferar, nas mesmas, uma miscelânea de plataformas e tecnologias. No início, com as simples aplicações empresariais residindo em plataformas centralizadas, principalmente nos mainframes, os conceitos de integração de aplicações não eram sequer pensados. Entretanto, com a evolução da complexidade das aplicações e a introdução de plataformas menores e mais abertas, os ambientes computacionais das organizações evoluíram para ambientes heterogêneos, compostos de variadas plataformas e tecnologias de desenvolvimento. Aliado a isto, a mudança nos modelos empresariais estabelecidos e a proliferação do *e-business* tem colocado a Integração de Aplicações Empresariais como imprescindível para fornecer suporte aos novos processos funcionais das empresas, aumentando assim sua produtividade e competitividade no mercado global.

O presente capítulo tem por objetivo definir o que vem a ser a Integração de Aplicações Empresariais, bem como fornecer um panorama histórico que justifique esta urgente necessidade de integração dentro dos ambientes computacionais das organizações. Além disso, pretende explicitar os tipos de integração possíveis e as tecnologias e padrões existentes e que são aplicados à solução deste problema.

### 2.1 Definição

A integração de aplicações não é um conceito sonhado ou imaginado por algumas pessoas, mas sim uma resposta a um problema verificado em grande parte das empresas desde que as aplicações moveram-se de um processador central e que o incremento do *e-business* obrigou as aplicações a dialogarem entre si.

Com esta idéia em mente, pode-se definir então a Integração de Aplicações Empresariais ou simplesmente EAI (*Enterprise Application Integration*) como o

compartilhamento de dados e processos de negócios entre quaisquer aplicações e fontes de dados na empresa (Linthicum, 1999). Outra definição, um pouco mais abrangente, entende a EAI como sendo o processo de organização de uma infra-estrutura de modo a criar um ambiente lógico que permita às pessoas e organizações fazer negócios utilizando tecnologias de intercomunicações (Buyens, 1999).

Em outras palavras, EAI pode ser compreendida como a introdução de ciência, metodologia e tecnologia no processo de integração de sistemas de informações ou componentes de software. Entretanto, a medida que se agregam componentes de software cria-se uma certa interdependência entre os mesmos, dada pelo conceito de acoplamento. O acoplamento entre componentes de software define o grau de integração ou nível de interdependência entre os mesmos, medindo o impacto que trocas em um dos componentes terá nos outros (Ruh, Maginnis, Brown, 2000). Assim, a meta de qualquer processo de integração é gerar componentes de software fracamente acoplados de modo que haja pouca dependência entre eles.

Para um melhor entendimento da EAI como uma questão essencial às empresas no suporte aos seus processos funcionais e agregar valor ao seu negócio frente aos desafios atuais do mercado, pode-se dividi-la em dois macro domínios (Linthicum, 2000): a integração intra-empresa e inter-empresa. A integração intra-empresa relaciona-se à integração de aplicações e dados que servem a uma única empresa, encontrando-se dentro dos limites da mesma. Por outro lado, a integração inter-empresa é a utilização de mecanismos, técnicas e tecnologias para a interligação de aplicações e dados que residem em organizações distintas, sendo esta, a base para os cenários de *e-business*. Este último macro domínio da EAI é considerado o elo perdido e um dos fatores de inibição para o sucesso e alavancagem do *e-business* (Drummond, 1999).

Em última análise, o cenário perseguido pelas organizações através da implantação dos conceitos de EAI é aquele da existência de um sistema virtual que englobe toda a empresa, onde toda e qualquer informação requerida por qualquer transação estaria imediatamente disponível para processamento, independentemente de onde ela esteja localizada. Isto implicaria, então, no agrupamento das diversas aplicações da organização de modo que elas pareçam e funcionem como uma aplicação

monolítica unificada. Tal cenário traria, sem sombra de dúvida, enormes benefícios e ganhos ou vantagens competitivas para as organizações pois possibilitariam a ela melhorar seu relacionamento com clientes e com a cadeia de fornecedores e investidores, aumentar a eficiência de seus processos internos, simplificar grandemente as interações com outras empresas através da adoção de padrões de integração e disponibilizar uma infraestrutura básica para o desenvolvimento rápido e eficiente de novas aplicações (Ruh, Maginnis, Brown, 2000).

Entretanto, se por um lado, a implantação das técnicas de EAI possibilita grandes benefícios para as organizações, por outro sua utilização efetiva pelas empresas ainda esbarra em algumas barreiras (Masler, 2000) (Linthicum, 2000) (Ruh, Maginnis, Brown, 2000): existência de arquiteturas complexas e caóticas dentro das organizações, resultado do desenvolvimento de anos de soluções departamentalizadas sem uma visão global da empresa; carência de pessoal capacitado a trabalhar com as tecnologias de integração como a tecnologia de middleware orientado a mensagens (*MOM-Message Oriented Middleware*) e de gerenciadores de mensagens (*Message Brokers*), arquitetura CORBA (*Common Object Request Broker Architecture*), o modelo de componentes de objetos distribuídos da Microsoft (*DCOM-Microsoft Distributed Component Object Model*) e a plataforma Enterprise JavaBeans (*EJB*); e os problemas de segurança que assumem uma dimensão bem maior do que aquela existente nas aplicações tradicionais.

## **2.2 Histórico e Evolução**

Nos primórdios da utilização das Tecnologias de Informação nos ambientes organizacionais, as aplicações computacionais eram de baixa complexidade e se limitavam a replicar em computador procedimentos manuais com o intuito de obter ganhos de produtividade. Além disso, tais aplicações eram destinadas a automatizar isoladamente tarefas específicas dentro das empresas. Outra característica importante desta época era que tais aplicações eram processadas em plataformas centralizadas, normalmente *mainframes*.

Como resultado deste ambiente computacional, tanto os processos quanto os dados residiam em um ambiente homogêneo e a necessidade de integração, quando havia, geralmente era sanada através de alguma codificação adicional.

Com o passar do tempo e o desenvolvimento tecnológico, verificou-se o surgimento de novas plataformas que, por sua vez, eram menores, mais abertas e alardeadas como mais baratas. O surgimento destas plataformas, bem como de novos paradigmas de desenvolvimento de software, tais como desenvolvimento baseado em componentes e orientado a objetos, incentivou uma migração das aplicações para estas novas plataformas. A adoção destas novas plataformas e tecnologias gerou, nas organizações, uma acumulação heterogênea de tecnologias e aplicações cujo gerenciamento tornava-se cada vez mais difícil, complexo e oneroso. Aliado a estas questões tecnológicas, o aumento da sofisticação das aplicações e a evolução natural dos ambientes e modelos empresariais estabelecidos implicou no surgimento de processos funcionais organizacionais mais complexos e que necessitam acessar e manipular informações que residem em diversas daquelas aplicações isoladas existentes da organização. Além destes motivos tecnológicos e organizacionais, a dependência cada vez maior das empresas por tecnologia, o incremento do *e-business*, mais especificamente o comércio *business-to-business*, o grande número de aquisições, fusões e incorporações de empresas e a migração dos sistemas empresariais para a Web tornou imperativa a questão de integração das aplicações empresariais (Masler, 2000) (Linthicum, 200).

A fim de acompanhar estas mudanças tecnológicas, organizacionais e de mercado, as organizações sentiram a necessidade de evoluir seus ambientes computacionais de cenários onde co-existiam um conjunto de aplicações isoladas e com tecnologias heterogêneas, para cenários onde estas aplicações trabalhem integradamente a fim de aumentar a eficiência da organização e dar suporte aos seus processos funcionais. Dentro deste complexo contexto, a EAI é tida então, como uma solução para os efeitos colaterais do desenvolvimento de aplicações isoladas, construídas com objetivos específicos e propósitos únicos, sem maiores aspirações de integrá-las a maiores ou múltiplas aplicações, bem como resultado direto de uma falta de previsão ou planejamento arquitetural dos ambientes computacionais das organizações.



Para agravar ainda mais o quadro atual e ressaltar a necessidade da EAI, duas questões se colocam como barreiras às organizações. Primeiramente, verifica-se que até bem pouco tempo atrás, não existiam planejamentos arquiteturais ao nível organizacional dentro das organizações. A maior parte das decisões em questões de Tecnologias de Informação eram tomadas departamentalmente onde cada um destes selecionava tecnologias e soluções que julgavam mais apropriada às suas necessidades, gerando, como resultado final, uma mistura de tecnologias e paradigmas que eram difíceis e custosos de integrar. Em segundo lugar, muitas daquelas aplicações isoladas existentes na organização foram desenvolvidas com tecnologias não padronizadas de desenvolvimento de aplicações e armazenamento de dados. Esta questão assume uma proporção ainda maior quando se verifica que, se por um lado estas tecnologias tem envelhecido, por outro as aplicações desenvolvidas permanecem críticas para as atividades da empresa, urgindo a necessidade de atualizá-las e integrá-las às outras aplicações da organização.

Desta forma, a integração das aplicações vem suprir uma lacuna uma vez que sempre há a possibilidade da pura e simples troca destas aplicações, geralmente descartada pelo seu custo e tempo de desenvolvimento proibitivos. Assim, a medida que a integração de aplicações permite a utilização das aplicações e bases de dados já existentes nas empresas, em vez de recriá-los, preserva preciosos investimentos já feitos em tecnologias e desenvolvimento, gerando uma margem competitiva para as organizações que necessitam compartilhar informações dentro de seus limites ou com parceiros comerciais (Linthicum, 1999).

No contexto da evolução destas tecnologias, nos primórdios, as formas mais comuns de integrar aplicações consistiam da implementação de bases de dados comuns e na troca assíncrona de arquivos (Masler, 2000). Depois do aparecimento e da proliferação das aplicações cliente-servidor, ligadas a plataformas menores e distribuídas, contribuindo para a acumulação heterogênea de tecnologias e aplicações e tornando os sistemas de informação das organizações cada vez mais difíceis de gerenciar, o aparecimento das soluções de ERP (*Enterprise Resource Planning*) surgiu como alternativa às necessidades de compartilhamento e integração de aplicações. Na medida em que as organizações substituíam muitas de suas aplicações por um único

sistema de ERP, cujos diferentes módulos compartilham as mesmas tecnologias e os mesmos modelos de dados, estes serviam como espinha dorsal dos sistemas de informação e facilitavam a integração das aplicações. A medida que os ERPs, soluções custosas e de difícil implementação, serviam a quesitos de integração, sua incapacidade de cobrir todas as necessidades funcionais das organizações derrubava o mito do ERP único, implicando na necessidade de integração dos mesmos com outras soluções integradas como gestão de *call centers* e da cadeia logística, entre outras (Masler, 2000) (Inmon, 1999). Assim como cada uma destas soluções integradas tem suas próprias interfaces e tecnologias proprietárias, a necessidade de integração volta à tona nos ambientes organizacionais.

Posteriormente, as soluções para a integração migraram para produtos de softwares de middleware *point-to-point* e de *message queuing*. Entretanto, tais soluções resolvem o problema de integração parcialmente e caminham, atualmente, para a criação de infraestruturas comuns entre as empresas com o objetivo de compartilhar informações, bem como para a utilização de tecnologias mais modernas, porém menos comprovadas, como *message brokers* e servidores de aplicação (Linthicum, 2000).

### **2.3 Tipos de Integração**

A implantação de soluções de integração de aplicações deve sempre ser precedida de uma análise visando um entendimento mais detalhado dos dados e processos de negócios da organização, indicando os requisitos de integração, bem como a forma mais indicada para aquela situação particular de integração. Desta forma, diferentes necessidades de integração podem se fazer necessárias em diferentes aplicações ou empresas. Dentro deste contexto existem diferentes tipos, pontos, dimensões ou modelos de integração de aplicações como, por exemplo, a integração ao nível de apresentação, funcional ou de dados (Ruh, Maginnis, Brown, 2000), a integração ao nível de interface de usuário, de interfaces de aplicação, de métodos e de dados (Linthicum, 1999) ou a integração ao nível de processos de negócios, de componentes, de serviços de mensagens e de transporte de dados (Buyens, 1999). Estas diferentes dimensões de integração, propostas por diferentes autores, possuem um conjunto de

conceitos comuns e, para fins de explicitação dos mesmos, adotar-se-á um modelo que contemple as dimensões de integração propostas nos três modelos acima citados.

### 2.3.1 Integração ao Nível de Dados

A integração de aplicações empresariais ao nível de dados é o processo que assegura o transporte ou movimento de dados entre diferentes bases de dados através do acesso direto às mesmas. Esta dimensão de integração implica, geralmente, na extração de informações de uma base de dados, seu processamento e posterior atualização em outra base de dados, podendo incluir também a transformação do formato dos dados e a aplicação de lógica de negócios para os dados que estão sendo manipulados. Diz-se então que a integração, neste nível, é conseguida através do acesso direto aos dados necessários às aplicações.

Como exemplo desta instância de integração pode-se imaginar uma integração de dados de múltiplas fontes para gerar um *data warehouse* único, completo e consistente, que seria utilizado por ferramentas de *data mining* e aplicações estatísticas para fins de análise e tomada de decisões.

Em termos comparativos, este tipo de integração provê o acesso a uma variedade maior de dados, implicando assim em maior flexibilidade se comparada ao modelo de integração através da interface de usuário ou apresentação. Outra vantagem é o fato de permitir a reutilização dos dados, uma vez que os dados acessados e transformados para fins de integração poderiam ser utilizados por outras aplicações. Além disso, a integração de aplicações ao nível de dados, na medida em que mantém as aplicações isoladas, não necessita de mudanças significativas no código das aplicações, diminuindo despesas de modificações, testes e desenvolvimento das mesmas. Tal característica, juntamente com os baixos custos dos mecanismos de movimentação e formatação de dados, colocam esta dimensão de integração acessível às organizações a um custo relativamente baixo. Por outro lado, a integração ao nível de dados pode implicar na necessidade de re-escrever lógicas de negócios, o que pode se tornar um grande problema. Outro empecilho encontrado neste modelo de integração é que a mesma está ligada a um modelo de dados de integração que pode ser alterado em função de

características evolucionárias das aplicações, implicando assim em significantes esforços para manter a integração.

Em suma, a movimentação e manipulação de dados, considerada muitas vezes fácil de se implementar, pode assumir proporções preocupantes uma vez que pode implicar na necessidade de manipulação de uma grande quantidade de bases e tabelas de dados.

### 2.3.2 Integração Funcional

O modelo de integração funcional engloba os tipos de integração ao nível de interface de aplicação e de métodos proposto em Linthicum (1999). A integração funcional pode ser utilizada tanto para a integração intra-empresa como também para a integração inter-empresa e baseia-se na integração das aplicações ao nível de lógica de negócios. Esta, por sua vez, corresponde ao código escrito para realizar as funções de negócios requeridas por um aplicação, incluindo as regras para a manipulação e interpretação dos dados, implementando, desta forma, o processamento de negócios em uma linguagem de programação. Pelo fato de integrar as aplicações ao nível de lógica de negócios, em oposição às integrações ao nível de dados e interface de apresentação, a integração funcional permite solucionar uma faixa mais abrangente de problemas de integração, tais como problemas de *consistência de dados* (propagação e atualização de informações de uma ou mais fontes de aplicações integradas, sendo esta solução, considerada como integração funcional em vez de integração ao nível de dados a medida que necessita da aplicação de lógica de negócios neste processo), *integração de processos de múltiplos passos* (processamento de ações de forma coordenada através de várias aplicações integradas sem intervenção humana ou redigitação de dados) e o *desenvolvimento de componentes de integração plug-and-play* (componentes com interfaces bem definidas para fins de integração, permitindo a fácil conexão destes sem a necessidade de recodificação).

Para a implementação da integração funcional as organizações podem se valer das interfaces de programação de aplicações (*API - Application Programming Interface*) fornecidas pelas aplicações para acessar e compartilhar tanto processos de negócios quanto informações. Tal tipo de solução é mais apropriada para a integração de pacotes

de aplicações (SAP, PeopleSoft e Bann, entre outros) que disponibilizam interfaces para o acesso a seus processos e dados, bem como para extrair informações, colocá-las em um determinado formato e transmiti-las para outras aplicações. Entretanto, se por um lado tais pacotes de aplicações disponibilizam APIs para o acesso a seus processos e dados, por outro o fazem baseadas em tecnologias proprietárias, com semânticas e funções específicas de cada interface de aplicação. Como um luz no final do túnel, este cenário começa a mudar e as organizações fornecedoras de pacotes, com suas respectivas APIs, estão começando a suportar padrões de mercado nestas suas interfaces, principalmente a linguagem XML.

Além da utilização de APIs, o uso da Chamada Remota de Procedimento (*RPC – Remote Procedure Call*) foi utilizada durante muito tempo para solucionar os problemas de integração, sendo, tal solução, substituída atualmente pela utilização de *middlewares* de processamento distribuído como os Middleware Orientados a Mensagens (*MOM – Message Oriented Moddeware*) que provêem a integração através da passagem de mensagens entre as aplicações, Tecnologias de Objetos Distribuídos que se utilizam de um *middleware* composto de objetos baseados em interfaces e protocolos padrão para se comunicar com outros objetos para fins de integração e os Monitores de Processamento de Transações que provêem suporte às aplicações de missão crítica das organizações. Além destes, os Servidores de Aplicação e os *Message Brokers* são tecnologias mais recentes também utilizadas para implementar a integração funcional.

De um ponto de vista comparativo, o modelo de integração funcional é o mais flexível, robusto e abrangente dentre os três tipos de integração, provendo também um maior grau de reusabilidade dos componentes de integração criados, bem como um fraco acoplamento no caso da solução de problemas de consistência de dados. Por outro lado, possui uma maior complexidade para ser implementado, implicando assim numa curva de aprendizagem maior para o domínio de suas tecnologias. Além disso, pode gerar em um forte acoplamento entre as aplicações integradas, principalmente na solução de problemas de integração de processos de múltiplos passos e, dependendo da complexidade da interface, no desenvolvimento de componentes de integração plug-and-play.

### 2.3.3 Integração ao Nível de Interface de Usuário

Esta é a mais simples e primitiva forma de integração, bem como a de menor custo. Neste contexto, a integração de múltiplas aplicações é conseguida através de uma interface de usuário que funciona como um ponto comum de integração. Assim, esta nova e unificada interface de usuário se apresenta como sendo uma aplicação única, muito embora ela possa estar acessando várias aplicações legadas. Por outro lado, cada interação do usuário deverá ser mapeada para a aplicação correspondente através das interfaces definidas, não havendo interconexão de aplicações e dados. Desta forma, este tipo de integração, acontecendo somente ao nível de interface do usuário, é tido como o mais limitado dentre os tipos aqui explicitados.

Podem ser considerados exemplos deste tipo de integração os Portais de Informação na Web que permitem integrar aplicações e apresentar em uma mesma interface de usuário informações extraídas de várias fontes ou aplicações.

## 2.4 Tecnologias para Integração

Como visto, a necessidade pela implantação de soluções de Integração de aplicações empresariais cresce cada vez mais dentro dos ambientes organizacionais na medida em que se constata, nestes ambientes, a existência de diversas aplicações desenvolvidas com tecnologias diferentes, bem como a existência de plataformas heterogêneas. Tais ambientes podem ser compostos por uma variedade de sistemas legados que contém aplicações de missão-crítica, pacotes de aplicações com tecnologias abertas e proprietárias, uma mistura de plataformas de hardware e sistemas operacionais, protocolos e equipamentos de rede, bem como aplicações e bases de dados distribuídas geograficamente. A integração desta miscelânea de tecnologias, se por um lado é uma tarefa árdua, por outro é considerada essencial para a obtenção de vantagens competitivas e para a sobrevivência da organização no novo cenário mundial.

A implantação de soluções para os problemas de integração passa, geralmente por softwares de *middleware* para combinar as funcionalidades das aplicações já existentes

nas organizações. Estas tecnologias referem-se, geralmente, a produtos de softwares independentes de aplicação que implementam serviços de mediação entre aplicações, reduzindo a complexidade da integração das mesmas (Ruh, Maginnis, Brown, 2000). As primeiras soluções de integração baseavam-se em tecnologias tradicionais de *middleware*, como a Chamada de Procedimentos Remotos (*RPC – Remote Procedure Call*), migrando posteriormente para soluções mais evoluídas como *middleware* orientados a mensagens, objetos distribuídos, servidores de aplicação e *message brokers*, sendo estes, vistos como uma nova geração de softwares de *middleware*.

#### **2.4.1 Chamada de Procedimentos Remotos (RPC)**

O modelo de Chamada de Procedimento Remoto é baseado na necessidade de executar um componente de uma aplicação em qualquer local de uma rede de computadores. Assim, baseia-se no desenvolvimento de aplicações distribuídas que se integram ao nível de procedimentos, permitindo que um processo em uma máquina execute um procedimento em outra. Este modelo utiliza uma construção tradicional das linguagens de programação, a chamada a procedimento, que é estendida de um único sistema ou máquina para um ambiente distribuído, permitindo assim, a execução remota de procedimentos em computadores ligados através de uma rede de computadores.

A Chamada de Procedimento Remoto é uma das formas mais populares para a implementação do modelo cliente-servidor em computação distribuída. Neste contexto, pressupõe-se a existência de processos clientes que iniciam uma interação com componentes de recursos ou servidores, transparente ao cliente, e aguardam uma resposta para retomar seu processamento, provendo, desta forma, uma conexão preponderantemente síncrona entre as aplicações.

Esta tecnologia é ainda hoje utilizada em muitas soluções, mas não tem sido utilizada nas modernas arquiteturas de integração de aplicações empresariais. Por estar fortemente associado ao modelo procedural de programação, sua importância para a EAI tem sido suplantada pela chegada da tecnologia de Objetos Distribuídos.

### 2.4.2 Middleware Orientado a Mensagens

Este tipo de *middleware* habilita a integração de aplicações através da troca de mensagens entre as mesmas. Uma mensagem, por sua vez, é uma construção que é transportada de uma aplicação para outra e contém, tanto dados, quanto informações de controle (Ruh, Maginnis, Brown, 2000). Em suma, todas as informações necessárias residem na própria mensagem. Dentro deste contexto de troca de mensagens, uma vez que as mensagens são transportadas de aplicações para aplicações e que tais aplicações podem possuir tecnologias heterogêneas, impõe-se a necessidade de codificação e decodificação das mesmas pelas aplicações uma vez que qualquer desvio deste formato estabelecido implicará na impossibilidade de interpretação dos dados e ações a serem desencadeadas. Além disso, tal tecnologia possui alto grau de simplicidade, razão para sua alta popularidade.

A tecnologia de Middleware Orientado a Mensagens (*MOM – Message Oriented Middleware*) provê um mecanismo de comunicação assíncrona para enfileirar mensagens (*message queuing*) que seriam transportadas entre aplicações, provendo assim a integração das mesmas. Desta forma, diminui-se o acoplamento entre as aplicações, uma cliente e outra servidora, implementando filas através das quais se dará a troca das mensagens. Neste contexto, um cliente produz uma mensagem, colocando-a em uma fila de mensagens. Em outras palavras, o cliente localiza uma fila que gerencia o tipo específico de mensagens que ele quer enviar, monta a mensagem no formato correto, colocando-a, em seguida, na fila definida. Um servidor será então responsável pelo gerenciamento das filas, fazendo as mensagens chegarem aos seus destinos.

A utilização de filas como forma de trocar informações entre aplicações pode se dar através da troca de mensagens ponto-a-ponto (*point-to-point*) ou por difusão (*broadcasting*). No primeiro caso, a transmissão das mensagens ocorre entre dois servidores ou aplicações específicas, podendo ser classificado como uma troca de mensagens com requisição e resposta (*request/reply messaging*). Este tipo de troca de mensagens é implementada através de duas filas: uma para as mensagens enviadas pela aplicação fonte e outra para as mensagens de resposta enviadas pela aplicação destino. Basicamente as filas gerenciam as mensagens que entram e saem através do método



FIFO (*First In First Out*), podendo, entretanto, implementar variações como a associação de prioridades e de complexas regras para o gerenciamento das mesmas. Com relação ao modelo de comunicação, forma como o qual as aplicações interagem, uma troca de mensagens ponto-a-ponto pode se dar de forma síncrona, onde a aplicação que enviou a mensagem deve aguardar até receber uma resposta para então continuar o seu processamento, ou assíncrona, onde a aplicação envia a mensagem e pode continuar seu processamento, não dependendo de uma resposta para tanto.

Na troca de mensagens por difusão as mensagens são difundidas para um conjunto de servidores ou aplicações conectadas. Neste contexto, uma alternativa de implementação baseia-se na troca de mensagens por publicação e assinatura (*publish/subscribe*) que permite o endereçamento das aplicações baseadas no seu interesse em vez de seu endereço ou localização na rede. Para tanto, as aplicações registram seus interesses por certos tipos de mensagens junto a um gerenciador. Quando este recebe mensagens, as envia às aplicações específicas. Ambientes avançados podem também suportar filtros que permitem especificar condições sob as quais uma mensagem deve ser repassada, bem como especificar ações alternativas como, por exemplo, direcionar a mensagem para filas alternativas.

A utilização de filas e mensagens como forma de passar informações entre aplicações implica no gerenciamento de questões como a tradução das mensagens, necessidade de conversão das mensagens para um formato que possa ser transmitido e a conseqüente restauração da mesma de modo que ela se torne legível à aplicação destino, bem como o gerenciamento e roteamento de múltiplas filas, uma vez que sistemas sofisticados de troca de mensagens implementam múltiplas filas que lidam com diferentes tipos de mensagens e suportam o roteamento dinâmico através dos nodos de uma rede.

#### **2.4.3 Objetos Distribuídos**

O surgimento das tecnologias de computação distribuída tem sido uma resposta aos requisitos de compartilhamento de informações distribuídas deste cenário global de negócios. As tecnologias mais recentes de computação distribuída são, por natureza,

orientadas a objetos, combinando o suporte ao desenvolvimento de aplicações distribuídas com a tecnologia de orientação a objetos.

A Tecnologia de Objetos Distribuídos (*DOT – Distributed Object Technology*) é considerada *middleware* porque permite a comunicação e a cooperação entre aplicações através da referência (criação e invocação) a objetos remotos independentemente de requisitos de hardware, software, linguagem de programação e sistema de rede de computadores (Linthicum, 1999). Por objetos distribuídos, pode-se entender pequenos programas de aplicação distribuídos geograficamente que usam interfaces e protocolos padrão para se comunicar com outros. Desta forma, desenvolvedores criariam objetos através de diferentes linguagens de programação, armazenados em plataformas dispersas e heterogêneas que, utilizando protocolos padrões, poderiam executar funções da aplicação pela invocação de métodos do outro objeto, implementando, desta forma, a integração ao nível funcional ou de métodos.

Esta tecnologia possibilita solucionar a questão da distribuição das aplicações a serem integradas por plataformas heterogêneas, permite bastante flexibilidade às aplicações pois estas podem invocar quaisquer objetos sem se preocupar com a sua localização física, bem como facilita a reutilização dos componentes de softwares ou objetos já desenvolvidos.

Dentro do contexto da utilização de objetos distribuídos para a integração de aplicações, três tecnologias estão em uso atualmente (Ruh, Maginnis, Brown, 2000): CORBA, DCOM/COM+ e Enterprise JavaBeans. A Arquitetura CORBA (*Common Object Request Broker Architecture*) é uma infraestrutura aberta de computação distribuída padronizada pela *Object Management Group* (OMG), uma organização internacional cuja carta de princípios inclui o estabelecimento de diretrizes industriais e especificações de gerenciamento de objetos para prover uma base única para o desenvolvimento de aplicações. Esta infraestrutura permite a objetos CORBA invocar um ao outro sem necessitar conhecer onde os objetos a serem acessados residem ou em que linguagem os objetos requisitados foram implementados. Para tanto, utiliza como elementos principais uma linguagem de definição de interfaces (*IDL - Interface Definition Language*) que define as interfaces entre os objetos e um gerenciador de

requisições (*ORB – Object Request Broker*), responsável por todos os mecanismos requeridos para a comunicação entre os objetos.

Outra tecnologia utilizada para o desenvolvimento de aplicações distribuídas, concorrente da arquitetura CORBA, é a tecnologia DCOM (*Distributed Component Model*), uma extensão da tecnologia COM (*Component Object Model*), da Microsoft, permitindo que os objetos possam agora residir em máquinas remotas. O DCOM, entendido então como o sistema de objetos distribuídos da Microsoft, é uma interface ActiveX para componentes de aplicações distribuídas. Como tal, fornece também transparência de rede e automação do processo de comunicação entre objetos, permitindo que estes possam estar em diferentes processos na mesma máquina ou em processos em máquinas diferentes e que as comunicações entre eles possam ser feitas sem que um precise saber a localização do outro. Este modelo utiliza a chamada de procedimentos remotos para permitir a interoperabilidade das aplicações e utiliza o serviço DNS (*Domain Name System*) para a localização de objetos na rede. Além disso, fornece também mecanismos de segurança através de autenticação e criptografia de dados.

Por fim, a tecnologia *Enterprise JavaBeans* (EJB), desenvolvida pela SUN Microsystems, define um modelo para o desenvolvimento e disponibilização de componentes Java reusáveis. Esta tecnologia objetiva suportar o desenvolvimento de aplicações multi-camadas com objetos distribuídos, focando-se no conceito de clientes magros, ou mínimos, onde a maioria das funcionalidades das aplicações residiriam no lado do servidor para habilitar e suportar aplicações distribuídas em larga escala.

#### **2.4.4 Servidores de Aplicação**

Servidores de Aplicação (*Application Servers*) são *middlewares* utilizados para a integração de aplicações ao nível funcional ou de métodos, provendo uma localização para a lógica da aplicação, bem como a coordenação das conexões entre as aplicações utilizando-se de semânticas transacionais. Desta forma, usuários destes servidores podem integrar recursos de *back-end* e aplicações de missão crítica usando a noção de transação, uma ou mais funções de negócios baseada em regras de negócios estipuladas, onde a mesma é completada somente quando todas as funções de negócios requeridas

são completadas de acordo com o que está especificado pelas regras de negócios (Ruh, Maginnis, Brown, 2000). Além disso, os servidores de aplicação permitem disponibilizar informações daqueles recursos de *back-end* para *web browsers* conectados ou através de interfaces cliente-servidor.

Os servidores de aplicação tipicamente fornecem aspectos de desenvolvimento de aplicações tais como um ambiente de desenvolvimento de integração (*IDE – Integration Development Environment*), bem como facilidades de desenvolvimento e teste de aplicações. A nova geração dos servidores de aplicação tendem a suportar Java tanto como linguagem nativa como modelo de desenvolvimento (embora de diferentes maneiras), movendo-se, desta forma, em direção do mundo da *Enterprise JavaBeans* (Linthicum, 200). Em suma, os servidores de aplicação provêm uma solução moderna e atualizada, baseado em transações, para os problemas de integração de aplicações.

#### **2.4.5 Message Brokers**

*Message brokers* representam o ápice dos *middleware* direcionados à integração de aplicações; pelo menos esta é a maneira como eles estão sendo posicionados dentro do contexto de integração de aplicações. Sua importância reside no fato do seu posicionamento dentro da estrutura de Tecnologia de Informação das organizações uma vez que eles não são uma tecnologia habilitada ao desenvolvimento de aplicações, mas sim uma tecnologia que permite às aplicações se comunicarem com outras sem que elas necessitem quaisquer conhecimentos umas sobre as outras. Em suma, eles simplesmente mediam a troca de informações entre elas.

Estes produtos facilitam o movimento de informações entre dois ou mais recursos, aplicações destino e fonte, gerenciando as diferenças na semântica das aplicações e plataformas, sendo, desta forma, adequados à solução dos problemas de integração de aplicações empresariais que, normalmente, lidam com diferentes semânticas de dados. Eles também podem juntar várias aplicações usando regras comuns e máquinas de roteamento.

Os *message brokers* nada mais são do que servidores que fazem o gerenciamento de mensagens entre aplicações. Para tanto e se necessário, eles podem transformar os

esquemas ou estruturas das mensagens, bem como o conteúdo das mesmas, de modo que elas façam sentido às aplicações que a estão recebendo, na medida que fluem entre aplicações e bases de dados diversas. Em contraste aos servidores de aplicação que utilizam um modelo transacional, onde transações são invocadas dentro do servidor de aplicação, os *message brokers* tendem a mover informações usando mecanismos dirigidos a eventos: ficam atentos às trocas de estado nas aplicações, reagindo a elas de acordo com regras pré-estabelecidas, tipicamente movendo e reformatando informações de negócios. Dentro deste contexto, uma única troca de estado pode implicar na geração de centenas de eventos como, por exemplo, uma reação em cadeia de movimento de informações e tratamento de eventos de negócios entre aplicações conectadas. Por outro lado, a utilização de *message brokers* possibilita um fraco acoplamento entre as aplicações, ao passo que a utilização de servidores de aplicação geram soluções fortemente acopladas. As soluções fracamente acopladas são as mais desejáveis pois tipicamente não requerem trocas nas aplicações fonte ou destino, sendo também menos onerosas para implementar.

## 2.5 Perspectivas e Tendências

A Integração de Aplicações Empresariais tende a assumir um papel cada vez mais relevante dentro das arquiteturas de TI das organizações, visto que ela deverá ser um meio para externalizar informações para as novas aplicações de *e-business*. Algumas tendências podem ser apontadas para o futuro da área de Integração de Aplicações Empresariais (Linthicum, 2001) (Ruh, Maginnis, Brown, 2000) (King, 2000) (Rowell, 2001):

- *Crescimento das funcionalidades para automação de processos de negócios:* atualmente os esforços de integração têm sido direcionados para o movimento de informações entre as aplicações e, a medida que os conceitos e tecnologias de EAI atinjam uma maturidade maior, espera-se um direcionamento para o encapsulamento de processos entre as aplicações, provendo modelos de processos de negócios que seriam implementados entre as organizações;

- *Direcionamento maior das metodologias e tecnologias da EAI para o processo de integração inter-empresa:* devido ao crescimento do *e-business* verifica-se uma tendência de que a EAI supra este mercado com as ferramentas necessárias para solucionar os problemas de integração entre parceiros comerciais, englobando aspectos como gerenciamento de parceiros, colaboração, suporte a XML (*Extensible Markup Language*) e à transações em tempo real;
- *Normalização dos padrões:* verifica-se atualmente um grande número de padrões para a troca de dados entre aplicações como ebXML, RosettaNet e BizTalk, dentre uma grande lista. No futuro, espera-se que somente alguns destes sustentem-se como padrões de fato, podendo-se notar, neste contexto, duas claras tendências: o uso de XML (*Extensible Markup Language*) com meio de representação de informações a serem compartilhadas entre parceiros comerciais e do XSLT (*XML Stylesheet Language Transformations*) como mecanismo de transformação de dados;
- *Aumento de escalabilidade:* a maioria das soluções dos problemas de EAI possibilitam a integração de poucas aplicações, normalmente um número menor que 10. Com o aparecimento de problemas de grande domínio, onde centenas de aplicações deverão ser capazes de trocar informações e compartilhar processos, espera-se que as novas gerações de soluções suportem a integração simultânea de um grande número de aplicações, fornecendo aspectos de processamento distribuído de mensagens e suporte massivo ao processamento paralelo;
- *Fusão de tecnologias para a geração de produtos híbridos:* em termos do mercado de soluções tecnológicas para a EAI espera-se uma diminuição dos fornecedores de soluções, através de aquisições e fusões, e o surgimento de produtos híbridos através da fusão das tecnologias existentes atualmente. Neste sentido, nota-se uma aproximação cada vez maior entre as tecnologias dos servidores de aplicação e os *message brokers*, uma vez que estes estão incorporando capacidades de desenvolvimento de aplicações transacionais e aqueles, capacidades básicas de gerenciamento de mensagens.

Em suma, pode-se compreender a Integração de Aplicações Empresariais como uma combinação de metodologias e tecnologias, onde as metodologias conduziram a

uma solução que pode ser implementada através de tecnologias existentes. Deve-se pensar a EAI, então, como uma solução para o problema das ilhas de automação existentes nos ambientes de Tecnologia da Informação de muitas organizações gerados pela individualização na busca de soluções e a inexistência de macro-planejamentos para as arquiteturas de TI das organizações. Neste contexto, ela situa-se como uma solução não invasiva para as necessidades de comunicação e compartilhamento de dados e informações, tanto intra-empresa quanto inter-empresa, que preserva os investimentos já feitos em tecnologia pelas organizações e possibilita a integração com seus parceiros comerciais, gerando, assim, fatores diferenciados de competitividade para a mesma.

Por fim, a EAI é tida como o ponto mais avançado em relação à uma tendência verificada de aproximação da informática aos negócios das organizações. Para tanto, há de se atingir um nível de abstração maior que permita a integração total das aplicações das organizações, independente da sua localização física, plataforma, linguagem de programação e protocolos, como forma de compor e suportar os processos e serviços de negócios das organizações através da utilização dinâmica e flexível de componentes encerrados nas aplicações já existentes ou naquelas a serem desenvolvidas.

## **3 AGENTES DE SOFTWARE**

Análises recentes indicam tendências de crescimento em relação à complexidade dos softwares nas próximas décadas. As características dinâmicas e distribuídas das aplicações atualmente requerem que as soluções a serem desenvolvidas sejam providas de habilidades que facilitem a interação dos usuários com os sistemas informatizados, bem como suportem a questão da distribuição das aplicações. Dentro deste contexto, a tecnologia de agentes tem recebido grande atenção da comunidade científica pois permite o desenvolvimento de partes menores de código e mais confiáveis que permitem superar as limitações da manipulação direta das interfaces e a complexidade da computação distribuída. O desenvolvimento de software baseado em agentes implica em que os programas de aplicação sejam escritos como agentes, componentes de software que se comunicam e concorrem ou cooperam com outros agentes para a realização de suas tarefas.

Este capítulo aborda aspectos da tecnologia de agentes como a definição e conceituação de agentes de software. Além disso, alguns tipos de agentes são descritos, bem como o processo de comunicação entre eles. Finalmente são abordados alguns casos de aplicação de agentes que ressaltam a importância desta tecnologia e o grau de maturidade que a mesma já atingiu.

### **3.1 Conceituação**

O termo “agente” existe na linguagem dos computadores desde a década de 80. Entretanto, a definição do termo não é e está longe de ser única e unânime para a comunidade científica. Dentro deste contexto, conceituar agentes não é uma tarefa simples, a ser formulada em poucas palavras. Vários pesquisadores definem agentes de diferentes formas, fundamentados, basicamente, na área de pesquisa em que trabalham.

Um agente pode ser definido como um ente autônomo que, imerso em um meio ambiente, percebe e atua sobre este a fim de atingir seus objetivos (Franklin & Graesser,



1996); como um componente de software ou hardware que é capaz de atuar em nome de um usuário para a execução de tarefas (Nwana, 1996) ou mesmo como um elemento de software de funcionamento contínuo e autônomo e imerso em um ambiente composto por outros agentes e processos (Shoham, 1997).

A despeito da diversidade de definições encontradas para agentes, verifica-se alguns elementos ou perspectivas comuns a grande parte delas. Uma primeira perspectiva comum é a associação de um agente a uma entidade que é capaz de agir. Numa segunda perspectiva, normalmente um agente é considerado como um ajudante ou alguém que atua em nome de outrem que lhe delegou tarefas específicas. De um ponto de vista mais amplo, a partir de várias definições encontradas na literatura, pode-se chegar à conclusão de que um agente de software, especificamente, é um componente de software que possui a habilidade, capacidade e autorização para agir em nome de um usuário para fins de realização de uma tarefa. Para tanto, necessita, normalmente, a interação com o usuário e, possivelmente, também com outros agentes, bem como a interação com o meio ambiente em que está imerso.

Em outras palavras, eles podem ser vistos como programas de computador ou componentes de software que apresentam características ou atributos vinculados a um agente, assunto este que será abordado na próxima seção. Dentro deste contexto, eles podem ser tão simples como uma subrotina até complexos elementos de software que possuem capacidade de aprendizado, caracterizando então, um agente inteligente.

### **3.2 Atributos de um Agente**

Como visto, o conceito de agente não é único, mas pode-se dizer que um agente deve apresentar alguns atributos ou propriedades para que possam ser caracterizados como tal. Da mesma maneira que a conceituação de agentes, diversos autores nominam diferentes atributos que poderiam definir um agente de software. Deve-se ressaltar, entretanto, que, a despeito da existência de um conjunto de propriedades e atributos que caracterizam um agente, a escolha de quais atributos um agente específico deverá conter

está ligada diretamente às funcionalidades que o mesmo deverá apresentar, ficando, tal escolha, a cargo de quem o projeta.

Um apanhado de algumas destas propriedades, com base em Franklin & Graesser (1996), Nwana (1996) e Wooldridge & Jennings (1996), seriam:

- *Autonomia*: capacidade de um agente atuar sem a intervenção direta de uma pessoa ou outro agente ou com a mínima intervenção indispensável;
- *Habilidade Social*: capacidade de um agente se comunicar e interagir com outros agentes ou pessoas para solicitar informações ou expor seus resultados;
- *Reatividade*: possibilidade de um agente reagir, de maneira adequada, à mudanças no ambiente no qual ele se encontra imerso;
- *Orientação por objetivos*: atributo também conhecido como pró-atividade e representa a capacidade de um agente atuar em função de um objetivo para o qual ele foi projetado e não somente em função de mudanças no meio ambiente em que está imerso;
- *Cooperação*: capacidade de um agente trabalhar em conjunto com outros agentes ou pessoas a fim de concluir suas tarefas, característica esta, tido como fundamental em um ambiente multi-agentes; e
- *Mobilidade*: capacidade dos agentes se moverem entre nós de redes de computadores, implicando que, em determinado período de tempo, um agente suspenda a sua execução, armazene seu estado interno e se dirija a outro nó da rede para lá, continuar sua execução.

### 3.3 Tipos de Agentes

Com base nos atributos que um agente pode apresentar, alguns deles anteriormente citados, pode-se classificar os agentes de acordo com um conjunto de atributos comuns ou tarefas que executam. Também neste contexto, diferentes pesquisadores sugerem diferentes classificações de agentes. Dentre estas classificações, alguns tipos de agentes de software merecem destaque.

### **3.3.1 Agentes Colaborativos**

Agentes colaborativos são aqueles agentes que residem em uma comunidade de agentes onde estes cooperam entre si para a consecução de suas tarefas. Dentro deste contexto, evidencia-se a necessidade de esquemas de comunicação entre agentes que possibilitem tanto a cooperação quanto o intercâmbio de informações, bem como um alto grau de autonomia dos mesmos.

Sistemas deste tipo, chamados de multi-agentes, possibilitam a construção de soluções através da construção de unidades menores, cada qual com funções definidas, provendo maior funcionalidade que a construção de um único elemento maior. Problemas de interconexão de múltiplos sistemas e o tratamento de informações provenientes de fontes distribuídas são áreas de aplicação para este tipo de agentes (Tolosa & Bordignon, 1999).

### **3.3.2 Agentes de Interface**

São elementos de software que assistem a usuários quando estes interagem com aplicações. Tais agentes monitoram as atividades dos usuários na interface das aplicações, identificam ações repetitivas e aprendem atalhos, sugerindo aos usuários formas mais eficientes para a realização das suas tarefas. Desta forma, os agentes de interface podem agir como assistentes pessoais, facilitando as tarefas dos usuários. Para tanto, é imprescindível que tais agentes apresentem capacidade de aprendizagem, podendo adaptar-se às preferências e hábitos de seus usuários.

### **3.3.3 Agentes de Recuperação de Informação**

Devido à grande quantidade de informações disponíveis atualmente nos computadores espalhados geograficamente, ocasionando um fenômeno conhecido como *sobrecarga de informação*, a tarefa de encontrar informações relevantes pode se tornar um tanto quanto difícil. Neste campo, especificamente, os agentes de software podem ser de grande valia.

Agentes que tentam solucionar o problema supra-citado, conhecidos como agentes de recuperação de informação, agem especificamente recuperando informações em

nome de seus usuários. Tais agentes podem ser utilizados para acessar dados em fontes heterogêneas e dispersas, bem como para manusear diferentes formatos de dados. Isto permitiria às organizações integrar dados dispersos geograficamente como forma de possibilitar a transformação dos mesmos em informações de valor agregado para seus usuários.

### **3.4 Comunicação entre Agentes**

As soluções de software desenvolvidas atualmente assumem uma dimensão a mais de complexidade a medida que precisam solucionar problemas advindos das características distribuídas das atuais e futuras aplicações. A utilização da tecnologia de agentes de software, dentro deste contexto, normalmente pressupõe a existência de um ambiente povoado por agentes que cooperam ou competem (negociam) entre si a fim de implementar as soluções necessárias. Neste ambiente, normalmente distribuído, a comunicação entre os agentes assume um papel preponderante na medida em que ela possibilita a troca de informações e a coordenação das ações dos agentes visando a obtenção de um objetivo. Entretanto, para que tal processo de comunicação entre agentes possa realmente se efetivar, pressupõe-se a existência de um linguagem de comunicação comum que permita a troca de dados e informações entre eles.

A definição de uma linguagem universal de comunicação de agentes persegue os pesquisadores a um bom tempo. Uma linguagem que satisfaz estes requisitos e tem sido suportada por um grande número de soluções baseadas em agentes é a linguagem ACL (*Agent Communication Language*). Esta linguagem, resultado de trabalhos do grupo ARPA Knowledge Sharing Effort, possui uma abordagem declarativa, baseada na afirmação de que a comunicação entre agentes pode ser melhor modelada como a troca de sentenças ou estruturas declarativas (Geneesereth & Ketchpel, 1994). Ela foi a primeira tecnologia bem difundida para a comunicação entre agentes de software, sendo concebida como um padrão para interações entre agentes objetivando permitir interoperabilidade de sistemas de múltiplos agentes (Wiederhold, 1994).

A linguagem ACL pode ser entendida como sendo composta de três grandes partes: um vocabulário, uma linguagem interna e uma linguagem externa. O vocabulário da linguagem ACL é um dicionário de palavras apropriadas às áreas de aplicação comuns, contendo, tais palavras, uma descrição para uso, pelas pessoas, no entendimento e significado da mesma, bem como uma notação formal, escrita na linguagem interna da ACL, para uso pelos programas. Em suma, este dicionário pode conter múltiplas ontologias (compreensão comum e compartilhada de algum domínio que pode ser comunicada entre pessoas e computadores), podendo, os programas, utilizarem a que lhe é mais conveniente. Além disso, ele é aberto, permitindo o acréscimo de novas palavras. A segunda parte da linguagem ACL, denominada KIF (*Knowledge Interchange Format*), baseada no cálculo de predicados de primeira ordem, provê a codificação de dados simples, restrições e regras, entre outras, para formar sentenças KIF que farão parte das mensagens a serem trocadas entre os agentes. Em outras palavras, esta parte da linguagem é usada para a codificação dos conteúdos proposicionais internos às mensagens trocadas entre os agentes. Finalmente, a linguagem ACL possui uma camada linguística, utilizada para encapsular estruturas KIF, fornecendo informações que levem em conta o contexto da comunicação, promovendo assim mais eficiência no processo de comunicação. Em suma, uma mensagem na linguagem ACL pode ser entendida como uma expressão em KQML (*Knowledge Query and Manipulation Language*), sua linguagem externa, na qual os argumentos desta são sentenças formatadas em KIF que, por sua vez, são formadas por palavras do vocabulário ACL.

A linguagem KQML, concebida com o objetivo de se tornar um padrão de comunicação entre agentes, é baseada na teoria dos atos de fala, sendo formada de um conjunto extensível de performativas que determinam as interações entre os agentes (Finin et al., 1994), suportando a comunicação de alto nível entre agentes e abstraindo o mecanismo de transporte das mensagens. As performativas KQML, por sua vez, são sentenças que explicitam a intenção do agente emissor da mensagem. Na comunicação entre agentes a intenção costuma ser parte integrante da mensagem, definindo claramente a intenção do agente emissor de modo que o agente receptor da mensagem não tenha dúvidas sobre como processar a mensagem recebida, simplificando, assim, o projeto de agentes comunicantes (Bordini, Vieira, Moreira, 2001). Além disso, KQML

inclui também uma arquitetura para plataformas de comunicação de agentes baseada na presença de facilitadores (agentes que tem conhecimento sobre quais agentes estão acessíveis, bem como seus habilidades), além de induzir a um conjunto de protocolos (seqüências de trocas de mensagens entre agentes em determinadas situações de comunicação).

O formato de uma mensagem KQML é baseado na sintaxe da linguagem LISP sendo, os argumentos, identificados por palavras chaves precedidos por um sinal de dois pontos, conforme visualizado abaixo:

( <i>performative</i>			
	:language	word	} message layer
	:ontology	word	
	:sender	word	} communication layer
	:receiver	word	
	:reply-with	word	
	:in-reply-to	word	
	:content	expression	} content layer
)			

A primeira camada (*message layer*) possui informações que tem por objetivo auxiliar o agente receptor da mensagem a entender o seu conteúdo. O campo *performative* identifica a intenção do emissor com a mensagem, enquanto o campo *:language* é a linguagem na qual a mensagem é expressada e o campo *:ontology* é o vocabulário usado para as palavras na mensagem. A segunda camada (*communication layer*) descreve os parâmetros de comunicação, como o emissor (*:sender*) e o receptor (*:receiver*) da mensagem, bem como um identificador único para a mensagem (*:reply-with*). Uma terceira e última camada na mensagem (*content layer*) permite especificar um campo *:content* como conteúdo da mensagem.

Exemplos de mensagens KQML podem ser visualizadas a seguir:

```
( ask-one
  :language   SQL
  :ontology   bovespa
  :sender      ag1
  :receiver   stock-server
  :reply-with q1
  :content    "SELECT price FROM stocktable WHERE ent=Conectiva"
)
```

```
( tell
  :language   prolog
  :ontology   bovespa
  :sender      stock-server
  :receiver   ag1
  :reply-with a2
  :in-reply-to q1
  :content    "[price(10.0)]"
)
```

No exemplo acima, pode-se ver o uso de duas performativas da linguagem, *ask-one* e *tell*. Entretanto, KQML especifica um conjunto de outras performativas, divididas em diversas categorias. Para fins de exemplificação, algumas categorias e respectivas performativas são exemplificadas na Tabela 1.

Tabela 1: Categorias e performativas KQML

<b>Categoria</b>	<b>Performativas</b>
Query	evaluate, ask-one, ask-if, ...
Multiresponse query	stream-in, stream-all, ...
Response	reply, sorry, ...
Information	tell, achieve, cancel, untell, ...
Generator	standby, ready, next, ...
Capability	advertise, subscribe, ...
Networking	register, forward, broadcast, ...

Fonte: Hübner & Sichman, 2000.

Embora a linguagem KQML possua um conjunto pré-definido de performativas, este não é fechado, podendo ser expandido. Entretanto, agentes que desejam implementar alguma daquelas performativas reservadas devem fazê-lo de maneira padronizada, de acordo com a intenção da mesma.

### **3.5 Aplicações de Agentes**

As pesquisas na área de agentes de software tem recebido grande atenção de Universidades, empresas e centros de pesquisa. Atualmente já existem várias aplicações baseadas em agentes para facilitar a vida dos usuários em áreas como o comércio eletrônico, ensino à distância, gerenciamento de sistemas, correio eletrônico e busca de informação na WWW, dentre outras.

A partir das fontes bibliográficas consultadas sobre a temática de agentes de software, pode-se identificar algumas áreas propensas à utilização desta tecnologia, algumas das quais já se notam resultados da utilização da tecnologia de agentes.

O gerenciamento de sistemas e redes de computadores foi uma das primeiras áreas de aplicação a empregar a tecnologia de agentes, especialmente os agentes inteligentes. Isto deve-se ao fato do uso crescente da arquitetura cliente/servidor, o que elevou a complexidade dos sistemas. Neste contexto, agentes podem ser empregados no gerenciamento dinâmico de grandes redes de computadores, aprendendo a reagir a determinados padrões no comportamento dos sistemas. Também em função da crescente distribuição das aplicações computacionais, a manipulação inteligente da transmissão das informações, necessária em ambientes móveis e de largura de banda limitada, torna-se uma área promissora para a tecnologia de agentes.

Outra área que está se utilizando de agentes de software a algum tempo é o correio eletrônico e a troca de mensagens. Aqui, agentes são empregados na priorização de mensagens e na organização das correspondências de usuários. Num próximo passo, espera-se que agentes inteligentes possam gerenciar o correio eletrônico e a troca de



mensagens para os usuários com base em regras que poderiam ser deduzidas a partir de padrões de comportamento retirados de observações dos hábitos dos usuários.

Agentes inteligentes de software também podem ser muito úteis para resolver o problema da sobrecarga de informações auxiliando no acesso e o gerenciamento de informações. Esta é uma área com grande efervescência em função da rápida popularização da Internet e a explosão da quantidade de informações disponíveis aos usuários. Neste caso, agentes inteligentes poderiam ser empregados para a pesquisa e filtragem de informações, bem como na sua categorização, priorização, disseminação seletiva e compartilhamento cooperativo.

Outra área de acelerado crescimento, alimentado pela popularização da Internet, é o comércio eletrônico. Se, por um lado, os consumidores que estão em busca de produtos e serviços necessitam de informações sobre o que estão comprando, por outro, os comerciantes necessitam localizar e atrair clientes, bem como oferecer informações e suporte sobre seus produtos. Tanto os clientes quanto os comerciantes necessitam automatizar suas participações neste mercado virtual e agentes poderiam auxiliá-los de diversas formas como, por exemplo, fazer compras para usuários, coletar informações sobre produtos e serviços e oferecer sugestões aos compradores com base em critérios por eles estabelecidos e participar de negociações entre clientes e comerciantes. Por outro lado, eles poderiam ser utilizados como forma de facilitar o comércio eletrônico entre as empresas, possibilitando a integração da cadeia de suprimento das empresas, viabilizando, desta forma, transações eletrônicas entre parceiros comerciais.

Vislumbra-se também a utilização de agentes de software em tarefas de gerenciamento administrativo das organizações como forma de torná-las mais eficientes. Neste contexto eles poderiam ser usados para gerenciar fluxos de trabalho, integrar computadores e serviços de telefonia, bem como automatizar e gerenciar processos de interesses dos usuários destas organizações. Além disso, poderiam ser utilizados no gerenciamento de processos colaborativos, onde usuários trabalham cooperativamente no desenvolvimento de projetos em um ambiente de rede de computadores.

A área de interfaces inteligentes é outra que está propensa a utilização de agentes. A medida em que a população de usuários de computadores cresce e se diversifica, as interfaces se tornam cada vez mais complexas para acomodar hábitos e preferências variadas, tornando os computadores mais difíceis de serem utilizados. Neste contexto, agentes inteligentes poderiam ser utilizados para monitorar ações dos usuários e identificar padrões que serviriam de base tanto para a personalização de interfaces quanto para ajudá-los automaticamente a resolver os problemas que poderiam surgir.

Por fim, uma área que tem oferecido significantes oportunidades para a utilização de agentes refere-se a integração de aplicações e empresas. Em tal ambiente, tipicamente heterogêneo e distribuído, agentes poderiam gerenciar e monitorar recursos das organizações e facilitar a interação dos usuários com estes. De fato, agentes inteligentes podem tornar-se a tecnologia central para o desenvolvimento das empresas virtuais.

Em suma, predizer qual será o papel dos agentes de software no futuro e como eles serão construídos não é uma tarefa fácil principalmente porque esta tecnologia e a sua utilização estão ainda em fase de desenvolvimento. Entretanto, espera-se uma utilização cada vez maior desta tecnologia, quer seja para resolver os problemas encontrados atualmente nos ambientes organizacionais, quer seja como tecnologia para a solução dos novos problemas que estão a surgir, principalmente aqueles advindos das características distribuídas e dinâmicas dos novos ambientes e aplicações.

## **4 A LINGUAGEM EXTENSÍVEL DE MARCAÇÃO (XML)**

Na tentativa de solucionar os problemas de extensibilidade, estruturação e validação verificados no HTML, além de possibilitar a fácil e rápida construção de aplicações mais robustas e flexíveis, o W3C (*World Wide Web Consortium*), organismo responsável por estudos e padronizações na Internet, incentivou o desenvolvimento e endossa a utilização de uma linguagem chamada Extensible Markup Language (XML) para o desenvolvimento de aplicações que requeiram funcionalidades além daquelas fornecidas pelo HTML.

Este capítulo tem por finalidade apresentar um histórico e descrever a linguagem XML, bem como parte do conjunto de tecnologias a ela associadas, como a sua linguagem de folha de estilos e de ligação. Por fim, uma avaliação da situação atual da XML e das aplicações mais propícias à utilização deste padrão são também relacionados.

### **4.1 Histórico e Conceito**

O espantoso crescimento da Internet, e em particular da World Wide Web, verificado nos últimos anos, tem tornado a mesma a mídia preferida para a transmissão e disseminação de informações, acesso a dados e comunicação pessoal. Assim sendo, inúmeras novas aplicações distribuídas e multi-plataforma, como comércio eletrônico, bibliotecas virtuais, educação à distância e sistemas de hipermídia, vindo sendo desenvolvidas para este ambiente, vindo se somar àquelas aplicações tradicionais já existentes na Web. Um dos pilares deste espantoso crescimento da Web está na possibilidade dada aos autores e desenvolvedores de, facilmente e a baixo custo, distribuir documentos e aplicações para uma audiência internacional. Isto decorre, principalmente, pelo fato da Web estar baseada em padrões simples, fáceis e abertos, como os protocolos TCP/IP e HTTP e a linguagem HTML, sendo, esta última, o formato de armazenamento e transmissão da grande maioria dos documentos na Web.

O HTML (*HyperText Markup Language*) é uma aplicação da SGML (*Standard Generalized Markup Language - ISO 8879*) e foi desenvolvida inicialmente para a publicação de documentos hipertexto na Web, tornando-se rapidamente um fenômeno devido a sua simplicidade. Durante um bom tempo o HTML serviu aos propósitos de desenvolvimento de aplicações para Web mas, a medida que uma gama de novas aplicações começaram a ser desenvolvidas, suas limitações começaram a aparecer, uma vez que a mesma precisou ser usada para fins para as quais ela não tinha sido criada. Desta forma, algumas limitações do HTML são um grande empecilho para o projeto de aplicações Web robustas, poderosas, flexíveis e evolucionárias (Johnson, 1999):

- *falta de extensibilidade*: HTML possui um conjunto fixo de tags com semântica bem definida, não permitindo aos desenvolvedores a customização dos mesmos para explicitar o significado dos dados de acordo com aplicações específicas;
- *centrada na apresentação*: HTML é considerada uma boa linguagem para propósitos de exibição de documentos na Web mas devido a sua mistura entre dados e marcação, deixa a desejar no desenvolvimento de aplicações;
- *dificulta a reusabilidade*: a mistura da estrutura lógica do documento com marcações de apresentação compromete a reusabilidade de documentos HTML, sendo que a separação entre conteúdo e forma e a especificação dos dados em termos de sua estrutura facilitaria a atualização e reutilização dos mesmos e novas formatações poderiam ser facilmente aplicáveis a conteúdos mutáveis;
- *provê somente uma visão dos dados*: um documento HTML é formatado para uma apresentação específica e a visualização do mesmo de diferentes formas, baseado nos requisitos de usuários, implica na necessidade de desenvolvimento de código adicional ou o desenvolvimento de um novo documento HTML;
- *falta de estrutura semântica*: HTML permite especificar somente o lay-out ou apresentação dos dados, mas não seu significado, trazendo isto, sérios problemas ao processo de busca na Internet.

O HTML foi o passo inicial para a publicação Web e o grande responsável para que os usuários assimilassem os conceitos de marcação. A medida que este amadurecimento se verificou, tentativas de eliminar as restrições impostas pelo HTML como forma de possibilitar o desenvolvimento da nova geração de aplicações Web

passaram a ser feitas, inicialmente cogitando-se a utilização direta do SGML na Web, ao invés de apenas utilizar uma aplicação sua, no caso o HTML.

A SGML é uma meta-linguagem, ou seja, uma linguagem que pode ser usada para construir outras linguagens (Johnson, 1999) (Bedunah, 1999) (Walsh, 1998), sendo considerada a precursora das linguagens de marcação. Uma característica marcante da SGML é permitir que os documentos descrevam sua própria gramática, ou seja, que eles especifiquem o conjunto de marcações utilizadas e as relações estruturais entre elas, permitindo características de extensibilidade, flexibilidade e estruturação (Bosak, 1997). Ela é um padrão maduro e independente de vendedor e plataforma que emergiu de pesquisas desenvolvidas pela IBM na representação de documentos texto, no final dos anos 60 e, após um período de evoluções, o primeiro padrão foi publicado em 1986. Atualmente é usada para o processamento de grandes quantidades de documentos nas indústrias aeroespacial, automotiva e de telecomunicações, bem como pelo Departamento de Defesa e o Serviço de Imposto de Renda dos EUA. Entretanto, se por um lado a SGML resolve os problemas associados ao HTML, por outro é uma linguagem extremamente complexa para ser utilizada pela grande maioria das pessoas, além de contemplar muitos aspectos opcionais que não seriam necessários para aplicações Web, provendo uma relação “custo X benefício” pouco atrativa aos produtores de Web browsers. Desta forma, o que ora se apresentava era, por um lado, SGML, extremamente complexa e sofisticada e, de outro, HTML, uma aplicação da SGML que não apresenta características de flexibilidade, extensibilidade e estruturação. Buscava-se então uma alternativa que unisse a simplicidade do HTML à força do SGML para o desenvolvimento de aplicações Web.

Neste momento, o W3C (*World Wide Web Consortium*), organismo responsável por estudos e padronizações na Internet, tomou as rédeas da situação e criou um Grupo de Trabalho com os pesos pesados do mundo da SGML. Tal Grupo de Trabalho teria por objetivo construir um conjunto de especificações que facilitariam e incentivariam o uso dos aspectos benéficos do SGML na Web com as mesmas facilidades que se trabalha com o HTML (Bosak, 1997), sendo guiado pelas seguintes metas (Bray et al, 2000):

- esta nova linguagem deveria ser diretamente utilizável sobre a Internet e os usuários deveriam ser capazes de visualizar seus documentos tão rápido e facilmente quando podem visualizar documentos HTML;
- deveria suportar o desenvolvimento de uma grande variedade de aplicações;
- deveria ser compatível com o SGML (a maioria das pessoas envolvidas neste esforço procedia de organizações que possuem grande quantidade de material em SGML), mantendo os padrões existentes e resolvendo problemas relativamente novos de envio de documentos ricamente estruturados sobre a Web;
- deveria ser fácil escrever programas que processassem documentos escritos nesta nova linguagem e, foi tomado como meta, que um competente estudante de Ciência da Computação deveria demorar em torno de duas semanas para construir programas deste tipo;
- número de aspectos opcionais desta nova linguagem deveria ser mantido o mais próximo possível de zero, evitando assim que estes aspectos opcionais trouxessem problemas de compatibilidade;
- documentos escritos com esta linguagem deveriam ser razoavelmente claros e legíveis aos seres humanos;
- as especificações desta linguagem deveriam ser desenvolvidas rapidamente, visto que os esforços de padronização são notoriamente lentos;
- projeto da linguagem deveria ser formal e conciso;
- documentos nesta linguagem deveriam ser fáceis de serem criados, quer seja através de simples editores de texto ou scripts;
- a concisão de marcação deveria ser de importância mínima.

O resultado dos esforços deste Grupo de Trabalho foi a especificação de um subconjunto simplificado do SGML que foi chamado de XML (*Extensible Markup Language*). Desta forma, XML pode ser definida como um subconjunto do SGML que mantém suas características de extensibilidade, estruturação e validação, permitindo que documentos ricamente estruturados possam ser criados, transmitidos e processados sobre a Web, sendo projetada para maximizar seu poder de expressão e facilidades de aprendizagem e implementação (Bosak, 1997). A XML é uma linguagem de baixa curva

de aprendizado, principalmente para conhecedores do HTML pois, como ela, se utiliza de tags e atributos para identificar semanticamente os dados, permitindo a criação e especificação de tags para domínios ou aplicações específicas, bem como o relacionamento estrutural entre eles. A grande diferença entre as duas reside, basicamente, no fato do HTML especificar como os dados devem ser mostrados em um Web browser enquanto o XML especifica semanticamente os dados, deixando a interpretação do mesmo a cargo das aplicações que o irão processar (Laurent, 1998). Quanto ao papel destas duas linguagens, num futuro não muito distante, espera-se que o XML complemente em vez de substituir o HTML e, enquanto esta, e suas evoluções, será usada para formatar e visualizar dados, XML proverá uma boa maneira de representar dados estruturadamente (Johnson, 1999).

A XML é considerada uma linguagem muito poderosa e apresenta como características básicas (Bosak, 1997) (Johnson, 1999) (Bedunah, 1999):

- *extensibilidade*: XML é considerada uma meta-linguagem, possibilitando a criação de outras linguagens de marcação, sendo, desta forma, extensível. Assim, usuários podem criar e definir novos elementos (tags e atributos) como forma de customizar documentos e especificar semanticamente os dados para domínios e aplicações específicas. Esta possibilidade de atribuir semântica aos dados, associada à possibilidade de separá-los da sua forma de apresentação, possibilita o desenvolvimento de aplicativos de busca muito mais eficientes, bem como aumenta a legibilidade dos documentos XML aos seres humanos;
- *estruturação*: XML permite uma representação de dados estruturados uniformes e independente de aplicações, plataformas e fornecedores. Desta forma, documentos XML podem especificar as estruturas necessárias para representar esquemas de base de dados e hierarquia de objetos, permitindo o aninhamento das mesmas em qualquer nível de complexidade. Esta estruturação clara e precisa, por sua vez, facilita o desenvolvimento de aplicações que processem automaticamente documentos XML;
- *validação*: um documento XML pode conter uma descrição opcional de sua gramática, ou seja, os elementos que podem aparecer no documento e o relacionamento entre eles. Esta descrição, por sua vez, pode ser utilizada por

aplicações que necessitem validar o documento, garantindo assim a sua correteude estrutural;

- *flexibilidade das aplicações*: XML permite o desenvolvimento de aplicações mais flexíveis que incluam possibilidades de integração de dados vindos de diferentes fontes, tratamento dos dados no cliente, múltiplas visões dos dados e atualizações granulares do conteúdo. Além disso, permite que estas aplicações sejam desenvolvidas mais rapidamente pois disponibiliza tecnologias adicionais para a validação de documentos e importação dos seus dados para objetos de linguagens de programação;
- *separação entre conteúdo, estrutura e apresentação*: pelo fato da XML se utilizar dos delimitadores somente para descrever dados, é mantida uma separação entre os mesmos e sua estrutura das suas formatações de visualização, expressas através a sua linguagem de folha de estilos (XSL). Desta forma, esta separação entre o conteúdo e a estrutura da sua forma de apresentação possibilita que documentos XML sejam facilmente distribuídos na Web e processados, nos seus destinos, para diferentes propósitos.

Entretanto, a linguagem XML não está sozinha. Para ser considerada o formato universal para documentos e dados estruturados na Web ela baseia-se num conjunto de tecnologias padrão comprovadamente otimizadas para a Web e definidas pelo W3C. São elas (Bos, 1999): a *Especificação XML 1.0* que define as regras para a criação de tags e atributos em documentos XML; a *Linguagem XSL* utilizada para expressar folhas de estilo para visualização dos documentos XML, bem como para transformar o mesmo em outros tipos de documentos; as *Linguagens Xlink e Xpointer/XFragments* que descrevem maneiras padrão de adicionar links e apontar partes de um documento XML; o *Document Object Model (DOM)* que provê formas de acesso aos dados estruturados XML através de linguagens de programação; o *XML Namespaces* que descreve sintaxes para criar prefixos para os tags, evitando assim confusões que possam surgir por questões de duplicidades e o *XML Schemas* que auxiliam desenvolvedores a precisamente definir seus próprios formatos baseados em XML. Além destes, existem outros módulos e ferramentas disponíveis, e em desenvolvimento, para trabalhar com a linguagem XML, estando, tais documentações, disponíveis no Web Site do W3C (*World Wide Web Consortium*).



## 4.2 Especificação XML

A especificação XML é uma recomendação do W3C que tem por objetivo descrever as regras sintáticas para a criação de documentos na linguagem XML.

### 4.2.1 Documentos

Um documento XML é um arquivo texto que contém dados formatados conforme a especificação da linguagem. Esta escolha por arquivos texto foi uma opção consciente dos projetistas do XML: embora eles necessitem de maior largura de banda para a sua transmissão, quando comparados com arquivos semelhantes em formato binário, tal característica, que pode ser compensada por mecanismos de compressão de dados, foi relegada em função das facilidades de manipulação de arquivos texto (Bos, 1999).

Os documentos XML são compostos de marcações e dados que devem ser combinados e escritos segundo regras notacionais e estruturais do XML para que eles possam ser processados automaticamente. Arquivos que não sigam estas regras serão rejeitados pelos programas que processam o XML. Dentre as regras notacionais do XML destacam-se (Johnson, 1999):

- cada delimitador de marcação deve possuir, obrigatoriamente, seu correspondente tag de fechamento;
- não deverá haver sobreposição de marcação, ou seja, a estrutura do documento deverá ser estritamente hierárquica (tags abertos dentro de outros tags devem ser fechados antes do fechamento do tag hierarquicamente superior);
- os valores de atributos deve, obrigatoriamente, ser colocados entre aspas;
- os nomes dos tags são *case-sensitive*;
- a utilização dos caracteres (>), (<) e (“) como dados e não como marcadores só pode feita através da utilização de caracteres de referência a entidades, respectivamente (&lt;), (&gt;) e (&quot;);
- todos os documentos devem possuir um elemento raiz que engloba os demais elementos do documento formando uma árvore de elementos do documento.

As entidades e marcações são organizadas, dentro de um documento XML, em duas seções: um cabeçalho, que define a versão da linguagem que está sendo utilizada e suas folhas de estilo e declarações de documento associadas, e uma árvore de elementos e instruções de processamento, contendo as marcações e os dados do documento. Estas, por sua vez, especificam uma estrutura lógica e uma estrutura física. Fisicamente o documento é composto por unidades chamadas entidades, sendo que a árvore de elementos do documento começa com uma entidade denominada raiz ou entidade do documento. Logicamente ela é composta de elementos, referência a entidades, comentários, instruções de processamento, seções marcadas (CDATA) e declarações de tipos do documento, todos indicados no documento por uma marcação específica e explícita (Bray & et al, 2000) (Walsh, 1998) (Johnson, 1999).

Os elementos são a forma mais comum de marcação no XML e correspondem às tags de marcação que, da mesma forma que no HTML, são delimitadas pelos sinais de menor (<) e maior (>) e devem obedecer as regras sintáticas explicitadas na especificação da linguagem. São também chamados de meta-dados pois fornecem informações sobre os dados por ele delimitados ou, em outras palavras, indicam semanticamente a natureza do conteúdo que delimitam. Elementos podem conter outros elementos, aninhados hierarquicamente com base na semântica ou na estrutura lógica do documento, e atributos, como forma de prover informações adicionais sobre o mesmo. Uma característica marcante do XML é a possibilidade de especificar elementos vazios, possuindo, para tanto, uma sintaxe diferenciada, na qual a marcação de início termina com o elemento “/”, suprimindo a marcação de fim.

Por outro lado, entidades XML são representações simbólicas para textos ou arquivos externos (Bedunah, 1999), prestando-se para dois propósitos: inserir no documento XML caracteres reservados para marcação como conteúdo ou caracteres Unicode e referenciar blocos de texto frequentemente repetidos ou para incluir o conteúdo de arquivos externos em um documento XML. As entidades em XML podem ser internas, externas ou parâmetros (Walsh, 1998): entidades internas são aquelas em que o bloco de texto a ela associado está definido internamente no documento; entidades externas são aquelas associadas a arquivos externos e entidades parâmetro são aquelas usadas nas declarações do documento para definir modelos de conteúdo

semanticamente idênticos, permitindo seu compartilhamento e reutilização. Quando do processamento de um documento XML, a ocorrência de uma entidade implica na substituição da mesma pelo conteúdo que ela representa.

Comentários também podem ser incluídos em um documento XML para tornar o próprio documento ou uma definição de tipo de documento mais legível e prover informações adicionais sobre os dados. Eles não fazem parte do conteúdo de dados de um documento e os processadores XML são orientados a não passá-los para a aplicação que processa o documento.

Instruções de processamento (PIs) e seções CDATA são construções usadas para fornecer informações adicionais a uma aplicação. Assim como os comentários, as PIs e seções CDATA não fazem parte do conteúdo de um documento XML, mas diferentemente deles, os processadores XML são orientados a passá-las para a aplicação que se encarregará de processar os dados de acordo com a sua lógica.

Declarações de tipos do documento também podem ser usadas em documentos XML para declarar o conjunto de elementos que podem ser usados no mesmo, bem como os relacionamentos entre eles. Além disso, permite especificar os atributos destes elementos, entidades e notações usadas no documento. Em suma, as declarações especificam restrições na estrutura do documento, além de permitir aos mesmos comunicar aos processadores XML meta-informações a respeito do seu conteúdo (Walsh, 1998). Estas declarações, embora não obrigatórias, são úteis para aplicações que necessitem de uma checagem estrutural do documento antes do seu processamento. Tais declarações podem ser feitas no cabeçalho do próprio documento XML ou, preferencialmente, em um documento externo denominado DTD (*Document Type Definition*), possibilitando, desta forma, a reutilização e compartilhamento da mesma.

Por fim, outro aspecto interessante da marcação de documentos XML diz respeito à manipulação de espaços em branco, podendo estes, ser significantes ou não, dependendo do modelo de conteúdo dos elementos. Em elementos que possuem somente sub-elementos com conteúdo os espaço em branco não são significativos. Entretanto, em elementos que possuem um modelo de conteúdo que possa conter, além de outros elementos, textos livres, os espaço em branco podem ser significativos. A

linguagem XML especifica um atributo que indicará ao processador XML se ele deverá ou não ignorar espaços em branco.

#### **4.2.2 Definição de Tipo de Documento (DTD)**

Um dos maiores poderes da XML é a possibilidade dada ao usuário de definir marcações próprias a fim de quantificar semanticamente os dados do documento para algum domínio ou aplicação específica, sendo o objetivo das marcações, descrever as unidades de armazenamento dos dados. Estas unidades de armazenamento de dados são relacionadas entre si com o intuito de formar uma estrutura lógica do documento. Por sua vez, a especificação da linguagem XML provê um mecanismo, chamado DTD (*Document Type Definition*), que especifica a gramática do documento, ou seja, as formas de utilização das marcações na montagem da estrutura lógica do documento (Bray et al, 2000).

Um DTD pode ser entendido então, como um arquivo externo, também com sintaxe XML, que é utilizado para especificar a estrutura de um documento XML. Em outras palavras, ele define linguagens de marcação específicas, representando, desta forma, a parte extensível do XML. Além disso, um DTD permite que documentos XML tornem-se auto-descritivos pois comunicam aos processadores XML informações a respeito do seu conteúdo. Sua utilização é opcional mas, uma vez associado a um documento XML, este deverá seguir rigidamente a gramática nele definida, sendo, suas informações, utilizadas pelos processadores XML para a validação estrutural do documento. Existem quatro tipos de declarações que podem ser feitas em um DTD: declaração de tipos de elementos, declaração de listas de atributos, e declarações de entidades e notações.

A declaração de elementos tem por objetivo explicitar quais elementos podem ser usados em um documento XML para fins de marcação dos dados, bem como o relacionamento entre eles. Para tanto, deve explicitar o nome dos elementos, seu modelo de conteúdo (que tipo de conteúdo ele pode conter), operadores lógicos de agrupamento (AND ou OR) e os indicadores de ocorrência do elemento (um elemento pode ocorrer uma única ou nenhuma vez, uma ou mais vezes, zero ou mais vezes ou uma e somente uma vez).

Por outro lado, os elementos de um documento podem conter atributos como forma de prover informações adicionais sobre os mesmos. Tais atributos também devem ser declarados especificando-se o nome do elemento e a sua lista de atributos que deve especificar nome do atributo, seu tipo (CDATA, Enumerado, ID, IDREF ou IDREFS, NMTOKEN ou NMTOKENS, ENTITY ou ENTITIES) e um indicador de ocorrência ou valor default (#REQUIRED, #IMPLIED, #FIXED ou um valor inicial para o atributo).

Outra declaração feita nos DTDs refere-se à declaração de entidades internas, externas ou parâmetro, permitindo associar um identificador a algum fragmento de texto ou a arquivos externos que poderão conter textos ou dados binários. A invocação de uma entidade por um documento XML causará sua substituição pelo modelo de conteúdo a ela associado.

Por fim, a linguagem XML permite também a declaração de notações específicas como forma de referir-se de maneira consistente e identificar tipos específicos de dados binários externos. Estas informações são passadas diretamente pelo analisador XML à aplicação que processa o documento, sendo ela, encarregada de fazer uso desta informação conforme desejar.

Como citado, a utilização de DTDs não é obrigatória e as declarações que ele especifica podem ser feitas internamente, no cabeçalho do documento XML. Entretanto, tal procedimento compromete a reutilização e o compartilhamento do mesmo. Desta forma, a declaração de elementos, atributos, entidades e notações deve ser feita, preferencialmente, através de um DTD, um arquivo externo associado a um documento XML via sua área de cabeçalho.

#### **4.2.3 Validação**

Como visto, documentos XML são compostos somente de dados e marcações. Estas, por sua vez, devem ser especificadas segundo algumas regras sintáticas constantes na especificação da linguagem. Devido a estas restrições e a possibilidade do usuário criar sua própria gramática, através de um DTD, os documentos XML podem ser classificados em documentos bem formatados e documentos válidos.

#### 4.2.3.1 Documentos Bem Formatados

Um documento XML é considerado bem formatado se o mesmo segue as regras notacionais e estruturais da especificação de marcações. Tais regras constam na especificação da linguagem e, dentre elas, destacam-se a obrigatoriedade de abertura e fechamento e a não sobreposição das marcações (precisão de estruturas), bem como o encapsulamento dos valores dos atributos entre aspas. Em suma, diz-se que um documento é bem formatado se o mesmo obedece a sintaxe XML e, por definição, se um documento não está bem formatado (inclui sequências de caracteres de marcação que não podem ser analisadas) ele não é considerado um documento XML (Walsh, 1998). A boa formatação dos documentos é uma restrição imposta pela linguagem para facilitar o processamento automático de documentos e os processadores XML rejeitarão qualquer documento XML que não seja bem formatado. Por outro lado, os documentos considerados somente bem formatados não possuem um DTD associado, não permitindo desta forma, uma validação estrutural do mesmo.

#### 4.2.3.2 Documentos Válidos

Um documento XML bem formatado é válido se, e somente se, ele possui um DTD associado e se o documento satisfaz as restrições da gramática nele especificadas (Bray & et al, 2000). Neste caso, cada vez que o documento é processado, o DTD é acessado de modo que a validade do documento possa ser checada. Tal checagem implica, entre outras coisas, em verificar se os elementos do documento seguem a sequência e o aninhamento especificado no DTD e se os atributos necessários foram fornecidos e seus valores são do tipo correto. Diferentemente do HTML, o XML é muito severo sobre a fidelidade às regras e o resultado da quebra de uma delas gerará como resultado um documento inválido. Tais restrições objetivam facilitar o desenvolvimento de aplicações que processem o XML.

#### 4.2.4 Processadores XML

Os processadores genéricos XML, também chamados de analisadores ou *parsers*, são elementos de software que registram violações nas especificações das restrições de bem formatação de documentos XML (Bray & et al, 2000). Além disso, estes *parsers*

também podem verificar a validade de um documento XML. Desta forma, eles podem ser categorizados em *parsers* de validação e de não-validação (Johnson, 1999). Os *parsers* de não-validação verificam somente a boa formatação de documentos XML. Eles recebem estes documentos como entrada e produzem mensagens de erros quando estes não seguem as regras de boa formatação. Esta categoria de *parser* é relativamente fácil de escrever, sendo utilizada quando não há um DTD associado ao documento e este não precisa ser validado estruturalmente. Por outro lado, *parsers* de validação, além de verificar a boa formatação do documento, vão além e verificam também a sua validade. Isto é feito com base no DTD associado ao documento e implica em checar se a estrutura do documento XML está de acordo com a gramática especificada no DTD.

Os *parsers*, de forma genérica, são as ferramentas utilizadas para se acessar o conteúdo de arquivos XML que lhe são fornecidos como entrada. As saídas do *parser* serão mensagens de erros, caso os documentos sejam mal formatados ou inválidos (dependendo do *parser* que se esteja utilizando). Por outro lado, se o documento for bem formatado e válido, se for caso de utilização de um *parser* de validação, ele fornece acesso aos elementos do documento via uma API. Assim um *parser* pode também ser categorizado em relação à forma como ele acessa e disponibiliza os elementos de um documento XML para o processamento pelas aplicações. Desta forma, os *parsers* podem ser baseados em árvore ou baseados em evento. Um *parser* baseado em árvore produz, em memória, uma árvore de elementos do documento e, por isto, pode consumir muita memória para arquivos muito grandes. Por sua vez, os *parsers* baseados em eventos não geram uma árvore de elementos em memória, pois somente percorrem o documento XML e disparam eventos quando um elemento é encontrado. Desta forma, eles são mais simples, consomem menos memória e são úteis para pesquisa.

### **4.3 Linguagem de Folha de Estilos e Transformações (CSS e XSL)**

Como visto, a linguagem XML permite a separação entre o conteúdo de um documento e sua forma de apresentação, residindo, no documento XML, puramente as informações relacionadas ao mesmo. Se por um lado esta separação facilita o processo de manutenção de Web sites e permite que aplicações manuseiem facilmente o conteúdo

de um documento XML, por outro implica na necessidade de uma tecnologia adicional para a estilização e visualização de um documento XML em um Web browser. Esta visualização pode ser feita através do uso de folhas de estilo tal como a CSS (*Cascading Style Sheets*) e a XSL (*Extensible Stylesheet Language*), ambas linguagens declarativas de estilo utilizáveis para a visualização de documentos XML em Web browsers.

A linguagem CSS existe como recomendação corrente do W3C como uma linguagem para ser aplicada tanto para HTML quanto para XML. Ela é uma linguagem mais simples e menos poderosa que o XSL, sendo suportada pela grande maioria dos Web browser da atual geração. CSS permite especificar, através de um documento separado, um estilo específico para os elementos e, a partir de então, todos estes elementos serão mostrados de acordo com o estilo a ele associado. Esta separação permite a troca de estilo dos elementos de um documento simplesmente através da troca da folha de estilos associada. Apesar de poderosa para fins de estilização de elementos, a CSS tem uma grande limitação: ela não pode transformar os dados que estiliza (Johnson, 1999). Ela pode aplicar formatação a um documento HTML ou XML e mesmo esconder elementos, mas não pode reformular, fazer referências cruzadas ou reestruturar elementos pois simplesmente aplica um estilo a uma estrutura existente. Para a realização de tais operações pode-se utilizar a linguagem XSL, uma linguagem de folha de estilos estendida.

A linguagem XSL é um subconjunto da linguagem DSSSL (*Document Style Semantics and Specification Language*), a linguagem de folha de estilo e transformação de dados do SGML. Entretanto, esta linguagem é um tanto quanto complexa e, para solucionar tal problema, o W3C incluiu uma linguagem de estilos própria para o XML, denominada XSL (*Extensible Stylesheet Language*), mantendo muito das potencialidades do seu predecessor. A especificação da Linguagem XSL é formada por duas partes: uma linguagem de transformação de dados (XSLT) e uma linguagem ou conjunto de objetos de formatação usados para definir lay-out para a Web e impressão, podendo uma funcionar independentemente da outra. XSL é uma linguagem extremamente poderosa que permite rearranjar completamente os elementos de um determinado documento XML. Entre outras coisas ela poderia transformar uma estrutura XML em uma estrutura HTML ou em uma estrutura XML diferente, além de



permitir gerar outros formatos de documento. Além disso, múltiplas representações de uma mesma informação podem ser produzidas por diferentes folhas XSL aplicadas à mesma entrada XML, permitindo, por exemplo, a fácil integração entre aplicações que falam diferentes dialetos do XML (poder-se-ia utilizar XSL para traduzir a saída XML de uma aplicação para algum formato XML compatível com a entrada da segunda aplicação). Uma questão interessante relacionada à sintaxe da XSL é que ela é especificada em XML. Então, pode-se definir a XSL como um documento XML que especifica como transformar outro documento XML (Johnson, 1999).

#### **4.4 Linguagem de Ligação e Ponteiros (XLink e XPointer)**

Muito do sucesso da Web está relacionado a sua capacidade de ligar recursos. Entretanto, a linguagem HTML apresenta algumas limitações quando se trata de sistemas hipertexto, suportando somente a forma mais simples de ligação entre documentos: links unidirecionais para localizações fortemente codificadas (Bosak, 1997). Tal tipo de link não está mais sendo suficiente para suportar a demanda das novas aplicações para a Web que estão a surgir. Um sistema hipertexto mais eficaz deveria apresentar mecanismos de linkagem mais sofisticados e flexíveis, sendo este o objetivo das especificações XLink (*XML Linking Language*) e XPointer (*XML Pointer Language*), introduzindo um modelo de linkagem padrão para a linguagem XML.

A linguagem XLink baseia-se na idéia de prover ao desenvolvedor atributos globais que podem então ser utilizados para marcar elementos como elementos de ligação. Desta forma, o desenvolvedor poderá marcar seus elementos como localizadores utilizando, para tanto, atributos da XLink. Além disso, pode-se valer do uso de ponteiros estendidos, através da especificação XPointer, que fornece um método sofisticado para a localização de recursos. Particularmente, XPointer permite a localização de recursos arbitrários em um documento sem que seja necessário que o recurso seja expressamente identificado com um atributo particular.

Basicamente a especificação XLink define as seguintes funcionalidades (Walsh, 1998):

- *link simples*: identifica uma ligação entre dois recursos, assemelhando-se ao link existente no HTML;
- *link estendido*: permitem expressar relacionamentos entre mais de dois recursos;
- *ponteiros estendidos*: sintaxes que permitem localizar recursos através da árvore de elementos que contém o recurso;
- *grupos de links estendidos*: que introduzem a possibilidade dos processadores XML processarem vários arquivos para efetuar a linkagem entre recursos.

Em suma, a especificação da linguagem XLink, embora compacta, é poderosa o suficiente para fornecer todos os mecanismos de ligação necessários ao desenvolvimento da nova geração de aplicações Web que estão para surgir, tais como a nomeação de recursos independentemente da localização, links bi-direcionais, especificação e gerenciamento das ligações fora dos documentos aos quais eles se aplicam, ligações n-árias e ligações para múltiplas fontes, dentre outros (Bosak, 1997).

#### **4.5 Modelo de Objetos do Documento (DOM)**

Os documentos XML contém informações que serão processadas para alguma finalidade, desde a simples visualização em web browsers até complexos processamentos por aplicações comerciais e científicas. Desta forma, faz-se necessário o acesso aos elementos e atributos que compõem o documento XML. Tal acesso se faz através da API DOM (*Document Object Model*). Esta nada mais é do que uma interface de programação em linguagem e plataforma neutras que permite a programas de computador acessar e atualizar dinamicamente o conteúdo de documentos XML através de qualquer linguagem de programação que o implemente (Bedunah, 1999).

Em outras palavras, a especificação DOM descreve um conjunto de interfaces capaz de representar qualquer documento HTML ou XML bem formatado, como um objeto árvore. Desta forma, uma vez parseado um documento XML será visualizado como um objeto nativo de uma linguagem de programação, em forma de uma árvore de elementos e atributos em cima dos quais pode-se operar processamentos. Um dos primeiros usos populares do DOM é na DHTML (*Dynamic HyperText Markup*

*Language*) onde scripts no lado do cliente manipulam um documento HTML em resposta às ações do usuário, tendo como pano de fundo uma árvore do documento HTML manipulada pelas interfaces do DOM.

#### **4.6 Espaço de Nomes (XML Namespaces)**

A medida que uma aplicação possa manipular diferentes documentos XML, cada qual possuindo um vocabulário próprio de marcação, o problema de reconhecimento e colisão entre nomes de elementos e atributos pode se tornar comum. Da mesma forma, coloca-se como relevante a questão da reutilização de vocabulários de marcação XML já desenvolvidos antes da recriação de tais vocabulários. Dentro deste contexto de reutilização de vocabulários e solução dos problemas de reconhecimento e colisão de nomes de elementos e atributos, o W3C recomenda a utilização da especificação XML namespaces.

A especificação XML namespaces é uma recomendação do W3C utilizada para evitar problemas de colisão em nomes e elementos de atributos de um documento XML, bem como possibilitar a reutilização de vocabulários de marcação através de um método simples para qualificar nomes de elementos e atributos usados em documentos XML. Tal método pressupõe uma coleção de nomes, dentro de um espaço determinado, identificando, tais nomes, através da referência a um URI. Desta forma, pode-se construir documentos XML que incluam elementos e atributos de domínios diferentes sem preocupar-se com a questão da colisão entre os nomes destes elementos e atributos, permitindo também a reutilização de vocabulários de marcação já existentes.

#### **4.7 Esquemas (XML Schema)**

XML Schema é uma recomendação do W3C para fins de estruturação de documentos XML, da mesma forma que um DTD o faz. Entretanto, a utilização de XML Schema, em vez de DTD, permite a definição de estruturas de dados mais complexas e precisas. Para tanto, um XML Schema utiliza uma sintaxe XML e introduz

o conceito de tipos de dados na definição de uma estrutura de dados a ser utilizada em um documento XML.

A especificação XML assume a utilização de, pelo menos, dois documentos XML: um documento denominado *esquema* descrevendo a estrutura a ser aplicada em um documento XML e outro documento denominado *instância*, um documento XML contendo as informações propriamente ditas, de acordo com um esquema (Box, Skonnard, Lam, 2000). Para fins de entendimento da diferença entre esquema e instância, pode-se utilizar a distinção feita entre objeto e classe em linguagens de programação orientada a objetos: uma classe descreve um objeto assim como um esquema descreve um documento instância.

#### **4.8 Aplicações de XML**

Desde o seu surgimento, a linguagem XML tem sido apontada como a panacéia para todos os males nas questões relacionadas à Internet sendo, algumas vezes, difícil separar os exageros da realidade. Inicialmente previu-se a utilização da XML no desenvolvimento daquelas aplicações Web que não poderiam ser desenvolvidas dentro das limitações do HTML. Ele coloca-se como uma alternativa para o desenvolvimento daquelas aplicações que hoje se utilizam cada vez mais de applets e scripts para estender as limitações do HTML, prova esta da sua estagnação. Entretanto, a utilização da XML transcende os limites da Web e, estendendo Bosak (1997), pode-se dizer que as aplicações que poderão tirar proveito do formato livre, padronizado e independente de fornecedor e ferramentas da XML podem ser divididas em quatro grupos :

- *aplicações que requeiram a mediação entre bases de dados heterogêneas*: nesta categoria, enquadram-se aquelas aplicações em que a XML pode ser usada como um formato independente de plataforma e fornecedor para a representação e troca de dados entre aplicações que utilizem formas díspares de representação de dados;
- *aplicações que distribuam a carga de processamento entre servidores e clientes*: esta categoria engloba aquelas aplicações que distribuem dados XML,

possivelmente oriundos de várias fontes, a clientes de software que poderiam facilmente operar processamentos sobre tais dados, balanceando a carga de processamento em uma arquitetura cliente-servidor;

- *aplicações que requeiram dos clientes a apresentação de diferentes visões de um mesmo dado*: neste caso, aplicações poderiam oferecer diferentes visões de um mesmo dado para os usuários sem a necessidade que estes dados sejam novamente solicitados ao servidor;
- *aplicações que necessitem de organização e reconhecimento semântico de dados*: um futuro domínio de aplicações surgirão quando agentes inteligentes de software começarem a requisitar maior estruturação e significação semântica para os dados permitindo, por exemplo, o reconhecimento e processamento de vocabulários XML específicos e a busca mais precisa na Internet.

Além disso, a característica de extensibilidade desta meta-linguagem aberta e padronizada a tem colocado como uma ferramenta propícia para a solução de problemas que requeiram tanto modelagem quando a especificação semântica de dados, quer seja para a publicação de informações, descrição de recursos computacionais, troca de dados e integração de aplicações, comércio eletrônico e definição de protocolos de comunicação, entre outras. Maiores detalhes sobre aplicações da XML e iniciativas acadêmicas e industriais que a estão utilizando podem ser encontradas no web site <http://www.oasis-open.org/cover/siteIndex.html#toc-applications>.

Dentro desta gama de aplicações, uma das áreas que se mostra promissora para a XML é a troca de mensagens e informações entre aplicações, possibilitando desta forma a integração de aplicações díspares. Neste caso ela funcionaria como uma linguagem universal de intercâmbio de dados visando a unificação estrutural e semântica dos dados compartilhados entre as aplicações envolvidas neste processo. Para tanto, XML apresenta alguns pontos fortes como a sua simplicidade, separação entre conteúdo e formato de apresentação, o poder de representação da meta-linguagem e o fato de ser um padrão aberto e, por outro lado, como pontos fracos algumas limitações na interpretação semântica e nas suas facilidades de transformação de dados, ineficiência dos documentos baseados em formato texto, bem como a ausência de mecanismos de roteamento de mensagens com base no conteúdo das mensagens XML (Yee, 1999).

A despeito disto, uma série de soluções e padronizações que utilizam o XML para a troca de dados e integração de aplicações tem surgido, dentre eles os *Frameworks* eCo, BizTalk e cXML que padronizam documentos de negócios a serem trocados entre empresas. Por outro lado, verifica-se também que grande parte dos fornecedores de Banco de Dados e Softwares Integrados de Gestão estão incluindo interfaces para tratamento de XML em seus pacotes de software.

Em suma, entende-se que a linguagem XML surgiu como uma alternativa ao HTML para o desenvolvimento daquelas aplicações que necessitem de maior poder de representação de dados, bem como a associação de semântica aos mesmos. Ela apresenta como grandes benefícios o fato de ser uma meta-linguagem aberta e padronizada, estruturada e extensível, independente de plataforma e vendedor que possibilita a separação entre os dados e a sua forma de visualização, bem como a validação estrutural dos mesmos, permitindo assim, o fácil e rápido desenvolvimento de aplicações mais flexíveis. Por estes motivos sua utilização tem sido crescente principalmente em problemas que requeira a modelagem e a especificação semântica dos dados. Neste contexto, uma das áreas promissoras da XML é o messageamento e intercâmbio de dados entre aplicações, possibilitando assim a integração das mesmas.

## **5 ARQUITETURA DE INTEGRAÇÃO DE APLICAÇÕES**

A implementação de soluções de integração de aplicações empresariais em qualquer organização requer tanto metodologia quanto tecnologia. Se por um lado uma metodologia permite a combinação de definições e normas para organizar e executar o desenvolvimento de soluções mais eficientes, por outro, a tecnologia é utilizada para implementar tais soluções. A combinação destes dois elementos possibilita a definição de uma arquitetura que permite a integração entre aplicações internas à própria organização, bem como com aplicações de parceiros comerciais, fornecendo uma plataforma para o compartilhamento de informações e a implementação de processos de negócios. Além disso, uma arquitetura de integração poderá servir de base para o desenvolvimento de novas aplicações que habilitem novas e inovadoras maneiras de fazer negócios, objetivando, desta forma, ganhos de competitividade para a organização.

Este capítulo propõe uma arquitetura de integração de aplicações, baseado em padrões abertos, que permite o processamento de negócios inter-empresa. Aqui serão apresentados os elementos básicos que permeiam a arquitetura proposta, bem com seus objetivos, camadas de software e estrutura de funcionamento. Além disso, a viabilidade da arquitetura é verificada através da implementação de um processo virtual de transferência de alunos entre Instituições de Ensino Superior, juntamente com uma avaliação e indicação de perspectivas futuras visando o aprimoramento da mesma.

### **5.1 Fundamentação**

O novo cenário de comércio mundial, altamente competitivo e globalizado, implica em que as organizações sejam capazes de processar negócios de forma mais rápida e eficiente com seus parceiros comerciais. Para tanto, a automatização destes processos de negócios coloca-se como um fator chave para se obter ganhos diferenciais competitivos, bem como para permitir o oferecimento de novos serviços para os usuários e parceiros comerciais da organização.

Dentro deste contexto, o compartilhamento e a troca de informações entre as aplicações das organizações é condição básica para suportar a implementação de processos de negócios automatizados. Pode-se dizer então, que a integração necessária ao suporte de processos de negócios entre parceiros comerciais extrapola os limites da empresa, caracterizando, assim, um processo de integração de aplicações inter-empresa.

Um grande número de desafios se colocam quando se trata da integração de aplicações inter-empresa, inerentes a um ambiente distribuído e a grande variedade de aplicações heterogêneas existentes nas organizações envolvidas. Dentre estes problemas, alguns merecem destaque: a heterogeneidade dos ambientes, independência das aplicações, semântica dos dados e a observância de padrões. A arquitetura de integração aqui proposta pressupõe a integração de aplicações ao nível de dados e propõem-se a solucionar os problemas acima citados valendo-se das tecnologias de Agentes de Software e da Linguagem XML (*Extensible Markup Language*).

A questão da heterogeneidade dos ambientes e da independência das aplicações é solucionada através da presença de agentes de software que interagem com as aplicações das organizações envolvidas em um determinado processo. Eles são projetados para interagir com aplicações e bases de dados específicas, mantendo assim a independência das mesmas em relação ao processo de integração e às outras aplicações. Desta forma, pode-se dizer que a arquitetura de integração proposta é não invasiva e fracamente acoplada, uma vez que não são necessárias trocas nas aplicações ou estruturas de dados envolvidas no processo de integração, permitindo também que processos de negócios possam ser implementados e alterados sem requerer alterações nas aplicações. Por outro lado, a questão da semântica de dados é solucionada através da definição de objetos de dados na Linguagem XML que podem ser validados estruturalmente e contém as informações e dados que serão compartilhados e transportados entre as aplicações para a consecução de um processo de negócios. Por último, a observância aos padrões é mantida a medida que os objetos de dados são formatados utilizando a Linguagem XML, padrão aberto adotado pelo W3C (*World Wide Web Consortium*) e por um grande número de organizações para a permuta de dados, bem como pelo desenvolvimento dos agentes de software valendo-se da linguagem de programação JAVA, aberta e independente de plataforma.

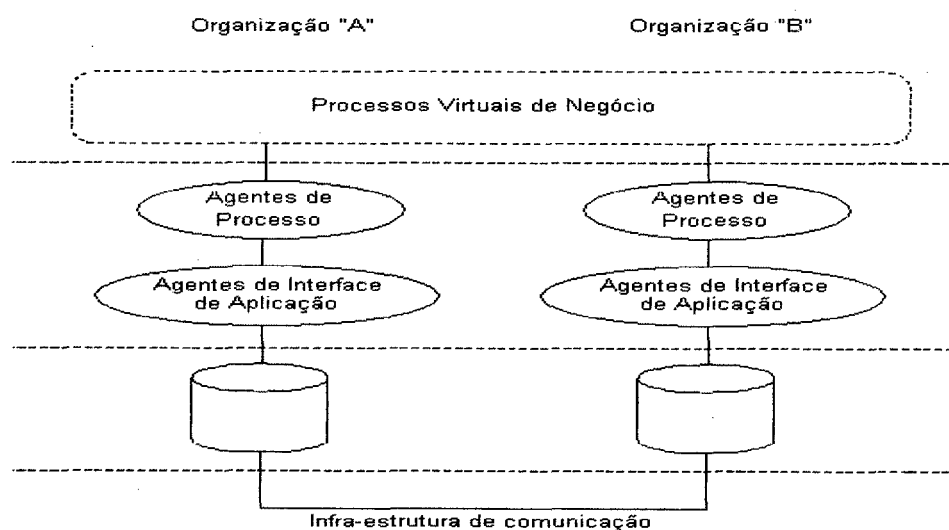


Em suma, pode-se dizer que a arquitetura de integração proposta propõe-se a solucionar o problema de compartilhamento e troca de informações como forma de suportar processos de negócios entre organizações. Tais processos, por sua vez, podem representar desde o oferecimento, a usuários e organizações, de informações integradas de diversas fontes, bem como permitir a execução on-line de transações de negócios entre organizações. Para tanto, a arquitetura se utiliza da lógica de negócios encapsulada em processos virtuais, executados por agentes de software, que, por sua vez, interagem com outros agentes, via mensagens KQML, para prover a interação entre as aplicações e estruturas de dados, compartilhando e trocando informações modeladas através de objetos de dados XML.

## 5.2 Camadas da Arquitetura

O arquitetura de integração proposta é definida com base em uma estrutura de três camadas, objetivando separar as definições dos processos virtuais de negócios, os agentes que executam as operações para a consecução de tais processos e a camada de dados. Esta estrutura em camadas pode ser visualizada na Figura 1.

Figura 1: Estrutura em camadas da arquitetura de integração proposta



A camada superior da arquitetura de integração proposta mantém as definições dos processos virtuais de negócios. A camada intermediária, ou camada de agentes, engloba os agentes de processos e os agentes de interface. Por fim, a camada inferior contém as bases de dados que são acessadas pelos agentes de interface. Com esta estrutura em três camadas pretende-se isolar, através de interfaces bem definidas, as definições dos processos de negócios e sua lógica das bases de dados, permitindo que alterações nos elementos de cada uma das camadas não implique, necessariamente, em alterações nos outros elementos.

Nesta estrutura em camadas da arquitetura integração proposta os agentes de processos e de interface de aplicação interagem através de uma infra-estrutura de comunicação, possibilitando, desta forma, a troca de mensagens e dados entre eles.

### **5.3 Elementos Básicos**

A arquitetura de integração proposta se baseia na definição e construção de cinco blocos básicos: processos virtuais de negócio, agentes de processo, agentes de interface de aplicação, protocolos de comunicação e objetos de dados.

#### **5.3.1 Processos Virtuais de Negócio**

Os processos virtuais de negócio são elementos lógicos localizados na camada superior da arquitetura de integração proposta. Um processo de negócio pode ser entendido como um ente abstrato que define uma ação que implementa, no seu todo, uma função de negócios da organização. Estas funções de negócios podem ser tanto internas, realizadas dentro dos limites de uma organização e, portanto, implicando apenas na integração de aplicações intra-empresa, como também externas, implicando na execução de operações além dos limites da empresa, necessitando, assim, da integração de aplicações inter-empresa. Neste caso, os processos podem ser compreendidos como virtuais pois combinam serviços oferecidos por diferentes empresas parceiras comerciais. No escopo da arquitetura de integração aqui proposta aborda-se somente a automatização de processos virtuais, aqueles realizados entre

organizações parceiras comerciais não se atendo, portanto, a automatização de processos intra-empresa. Em suma, pode-se entender então um processo virtual de negócios como uma seqüência de eventos ou ações que disparam a integração entre componentes ou aplicações visando a implementação de uma função de negócios entre organizações parceiras comerciais. Desta forma, tais processos extrapolam os limites de uma única organização, necessitando, portanto, de um processo de integração inter-empresa.

De outro ponto de vista, um processo virtual de negócios pode também ser entendido como uma troca de dados e informações entre aplicações de organizações parceiras para o processamento de um negócio. Para a consecução de tal processo, entretanto, poderá haver a necessidade de transformações no formato, bem como a aplicação de lógica de negócios aos dados que estão sendo intercambiados. Em outras palavras, o processo virtual de negócios especifica o objetivo final de integração.

Uma vez que processos virtuais de negócios implicam numa operação entre organizações, eles podem representar desde operações mais simples como o oferecimento de informações integradas de fontes diferentes e distribuídas, até aqueles que permitem a realização de transações on-line. Para fins de exemplificação, uma transferência de alunos entre duas instituições de ensino, uma integração de dados oriundos de diferentes fontes para gerar um *data warehouse* único para fins de análise e suporte a decisões, a propagação de dados por várias aplicações visando manter a consistência dos mesmos nas várias aplicações e um processo de tomada de preços para a compra de materiais entre clientes e fornecedores, entre outros, podem ser entendidos como processos virtuais de negócios.

Como visto, o processo virtual de negócios é o elemento superior na estrutura de camadas da arquitetura de integração proposta e envolve, normalmente, dois ou mais parceiros comerciais. Desta forma, a definição dos processos de negócios deve se dar de forma compartilhada pelos parceiros comerciais envolvidos e implica em definir os elementos básicos envolvidos no processo, como uma interface de usuário padrão para iniciar o processo, as aplicações e bases de dados a serem acessadas, a lógica de negócio

a ser aplicada aos dados, seu protocolo de comunicação e o formato do objeto de dados a ser intercambiado.

Em suma, a definição de um processo virtual de negócios corresponde então à etapa de discussão, entendimento e modelagem do mesmo de forma que ele possa ser implementado através da comunicação e cooperação entre agentes de software.

### **5.3.2 Agentes de Processo**

Os agentes de processo residem em uma camada intermediária na estrutura de camadas da arquitetura de integração proposta. Eles são elementos de software, associados a processos virtuais de negócios que, em nome dos usuários, coordenam a execução dos mesmos. Assim, compreende-se a existência de agentes de processo específicos para cada um dos processos virtuais de negócios implementados.

Para cumprir a sua finalidade, implementar um processo virtual de negócios, um agente de processo necessita interagir com usuários e com outros agentes de software a fim de solicitar serviços de acesso às bases de dados das aplicações envolvidas para fins de recuperação e atualização das mesmas. Neste caso, vislumbra-se um processo de comunicação e cooperação entre agentes de software e, para tanto, deve-se haver tanto uma infra-estrutura quanto um protocolo de comunicação previamente acordados. A arquitetura de integração aqui definida propõe a implementação dos protocolos de comunicação dos processos através da linguagem KQML de comunicação de agentes. Desta forma, mantém-se coerência com as diretrizes norteadoras da arquitetura de integração proposta, quer seja a utilização de tecnologias abertas e padronizadas.

Por fim, cabe também aos agentes de processo a responsabilidade pelo roteamento das mensagens e objetos de dados para os agentes de interface das aplicações envolvidas, bem como a aplicação de lógica de negócios, se necessário, a estes dados. Além disso, ele sinaliza aos usuários e aplicações envolvidas no processo os resultados da execução do mesmo.

### **5.3.3 Agentes de Interface de Aplicação**

Para solucionar a questão da heterogeneidade dos ambientes e manter a independência e autonomia das aplicações, a arquitetura de integração proposta define um conjunto de agentes que residem em uma camada intermediária na estrutura do modelo. Tais agentes, nominados agentes de interface de aplicação, conhecem os protocolos através dos quais as aplicações aceitam requisição e provêm resultados, sendo utilizados para o acesso às aplicações e bases de dados envolvidas nos processos virtuais de negócios. Desta forma, a arquitetura de integração proposta sugere a existência de um agente de interface específico para cada uma das aplicações e bases de dados envolvidas na execução de um processo de negócios.

Os agentes de interface de aplicação recebem requisições e objetos de dados dos agentes de processo e executam operações de atualização de bases de dados, recuperação de informações e formatação das mesmas de acordo com o objeto de dados em questão. Além disso, podem disparar funções nas aplicações a que estão vinculados. Em suma, eles aceitam como entrada ou requisições solicitando objetos de dados ou objetos de dados a serem atualizados nas bases de dados das aplicações e respondem, respectivamente, com objetos de dados recuperados das bases de dados ou respostas apropriadas sobre a conclusão das operações de atualização das bases de dados.

Tais agentes de interface de aplicação são, então, coordenados pelos agentes de processo para, juntos, implementar a solução de integração. Desta forma, verifica-se uma estreita cooperação entre eles, baseada num protocolo de comunicação previamente acordado e implementado através de mensagens na linguagem de comunicação de agentes KQML. Assim, mantém-se coerência com a meta de propor uma solução com base em padrões abertos da área de tecnologia da informação.

### **5.3.4 Protocolos de Comunicação**

Um quarto bloco básico na implementação da arquitetura de integração aqui proposta diz respeito à definição de um protocolo de comunicação entre os agentes de software. Tal protocolo corresponde ao conjunto e à seqüência de troca de mensagens entre os agentes de processo e de interface de aplicação a fim de possibilitar a troca de

objetos de dados e informações entre eles visando a consecução do processo virtual de negócios.

Na arquitetura proposta, tal protocolo é definido e acordado entre os parceiros comerciais no momento da definição do processo virtual de negócio e será baseado em mensagens KQML, uma linguagem de comunicação de agentes. Tal protocolo será, então, implementado pelos agentes de processo e de interface de aplicação e norteará as requisições e respostas solicitadas ou esperadas por tais agentes, bem como definirá o conteúdo de tais mensagens.

### 5.3.5 Objetos de Dados

A automatização dos processos virtuais de negócio implica no compartilhamento e troca de informações entre as aplicações e bases de dados envolvidas. Em função das diferenças de semântica e formatação dos dados entre as aplicações, colocá-se como necessário a definição de uma semântica única para os dados que devam fluir entre as aplicações para a consecução dos processos virtuais de negócio.

Um objeto de dados provê o mecanismo básico de transporte e manipulação de dados na arquitetura de integração proposta, manipulação esta que pode se dar pelo agente de processo quando, por exemplo, da aplicação de lógica de negócios a tais dados. Tal objeto consiste de um *container* para informações estruturadas, segundo um padrão previamente acordado entre as organizações envolvidas.

Seguindo a filosofia de se propor uma arquitetura de integração de acordo com padrões abertos de mercado, optou-se por modelar os objetos de dados com a Linguagem XML. Desta forma, os agentes de processo e interface de aplicação podem validar estruturalmente tais objetos, via um DTD (*Document Type Definition*), bem como manipulá-los através da utilização da API DOM (*Document Object Model*) que provê formas de acesso aos dados estruturados com a linguagem XML através de linguagens de programação que a implementam. Por fim, uma folha de estilos XSL ou CSS poderá ser também associada ao objeto de dados, permitindo a fácil visualização do mesmo.

#### 5.4 Sistemática de Funcionamento

A execução de um processo virtual de negócios implica na necessidade de uma integração inter-empresa, mais especificamente através das aplicações e bases de dados envolvidas no mesmo. Para tanto, algumas etapas devem cumpridas.

Inicialmente, os agentes de interface de aplicação devem ser instanciados nas organizações que irão executar o processo. Como visto, tais agentes implementam os serviços de acesso às bases de dados das aplicações e são coordenados pelo agente de processo através de mensagens KQML. Desta forma, tais agentes podem receber requisições dos agentes de processo para acessar bases de dados e formatar seus resultados de acordo com o objeto de dados do processo ou proceder à atualização de bases de dados de acordo com os objetos de dados recebidos. Uma vez instanciados, tais agentes ficam aguardando por mensagens KQML vindas do agente de processo para então, executar seus serviços.

Posteriormente, a organização que deseja executar o processo virtual deverá iniciar agente de processo vinculado ao mesmo. Através da interface do processo o usuário poderá parametrizar algumas opções do mesmo e disparar o agente de processo que será responsável por coordenar sua execução. Este agente, por sua vez, irá coordenar a execução do processo virtual através da interação e cooperação com os agentes de interface de aplicação para requisitar ou enviar objetos de dados para fins de atualização das bases de dados das aplicações.

Já de posse das informações que retornam dos agentes de interface, os agentes de processo concluem a execução destes e sinalizam o usuário com mensagens sobre o *status* da execução. Em função do resultado final da operação, os usuários podem dar o processo por finalizado ou agir de acordo com o status da mensagem recebida.

Cabe ressaltar também que, nesta sistemática de funcionamento da solução de integração proposta, os agentes de processo podem aplicar, se necessário, regras de negócios aos dados que foram recuperados de uma aplicação antes do seu encaminhamento aos agentes de interface de outras aplicações.

## **5.5 Viabilidade da Arquitetura**

Para fins de verificação de viabilidade da arquitetura de integração proposta, optou-se pela implementação de um processo virtual de transferência de alunos entre Instituições de Ensino Superior. Tal opção baseou-se numa necessidade comprovada de um ambiente real, na total ineficiência do processo manual feito atualmente, na inexistência de soluções automatizadas para este problema e pelo fato de tal exemplo possuir grande parte das características necessárias à validação da solução aqui sendo proposta.

### **5.5.1 Processo Virtual**

O processo virtual de negócio denominado transferência de alunos se dá entre Instituições de Ensino Superior no momento de uma transferência, ou seja, no momento em que um aluno transfere-se de uma Instituição para outra. Neste momento, um aluno solicita o desligamento de uma Instituição e a matrícula em outra. Atualmente este processo se dá de forma totalmente manual e consiste da realização de um conjunto de ações tanto na Instituição de onde o aluno está se transferindo, quanto na Instituição para onde ele está se transferindo.

A Instituição de onde o aluno está sendo transferido, aqui denominada de fonte, recupera as informações necessárias do aluno a partir das bases de dados de suas aplicações acadêmicas. Tais informações consistem do código de identificação do aluno e suas informações pessoais, juntamente com o conjunto de disciplinas e resultados que o mesmo cursou na Instituição fonte. Tais informações são então impressas e remetidas, via correio, à Instituição na qual o aluno está se matriculando, denominada de destino. Na Instituição destino os dados do aluno, recebidos via correio, servirão então para alimentar as bases de dados das suas aplicações acadêmicas. Neste processo, um conjunto de regras podem ser aplicados aos dados do aluno que está sendo transferido como, por exemplo, um processo de validação de disciplinas uma vez que pode haver discrepâncias entre as grades curriculares dos cursos da Instituição fonte e destino. Tal procedimento é, no mínimo, ineficaz pois implica na extração automática dos dados de uma fonte informatizada, na Instituição fonte, e a posterior reintrodução dos mesmos dados em uma fonte informatizada na Instituição destino. Estimativas sugerem que 70%



de todas as entradas em um sistema de computador foram saídas prévias de outros sistemas (Nelson, 1994).

O processo virtual de transferência de alunos, a ser implementado, é então sintetizado como sendo a recuperação, na Instituição fonte, dos dados pessoais e acadêmicos do aluno sendo transferido, sua formatação de acordo com o objeto de dados definido para o processo e envio à Instituição destino. Na Instituição destino tais dados são validados e alimentarão as bases de dados de suas aplicações acadêmicas. Este procedimento é então mais fácil e eficaz pois as informações fluem ao nível das aplicações acadêmicas das Instituições envolvidas no processo.

### 5.5.2 Objeto de Dados

O objeto de dados é utilizado para manter a semântica dos dados que estão sendo manipulados durante a execução do processo virtual de negócios. Tal objeto de dados, correspondendo a uma versão simplificada do procedimento de transferência de alunos, é formatado na linguagem XML, podendo ser visualizado na abaixo:

```
<?xml version="1.0"?>
<!DOCTYPE SYSTEM " "
<TransferenciaAluno>
  <CodigoAluno>          </CodigoAluno>
  <NomeAluno>           </NomeAluno>
  <Disciplinas>
    <Disciplina>
      <CodigoDisciplina>          </CodigoDisciplina>
      <NomeDisciplina>           </NomeDisciplina>
      <PeriodoOferta>           </PeriodoOferta>
      <NotaFinal>                </NotaFinal>
      <NumeroFaltas>            </NumeroFaltas>
      <ResultadoFinal>          </ResultadoFinal>
    </Disciplina>
    .
    .
    .
    <Disciplina>
      <CodigoDisciplina>          </CodigoDisciplina>
      <NomeDisciplina>           </NomeDisciplina>
      <PeriodoOferta>           </PeriodoOferta>
      <NotaFinal>                </NotaFinal>
      <NumeroFaltas>            </NumeroFaltas>
      <ResultadoFinal>          </ResultadoFinal>
    </Disciplina>
  </Disciplinas>
</TransferenciaAluno>
```

Tal objeto encapsula os dados do aluno que está sendo transferido, podendo ser mais ou menos complexo, dependendo das necessidades do procedimento que será implementado pelo processo virtual. Em suma, o objeto de dados consiste em um *container XML* que será preenchido com os dados dos alunos.

Para permitir a validação estrutural deste objeto de dados um DTD (*Document Type Definition*) é a ele associado, conforme visualizado a seguir, permitindo a sua validação estrutural quando da utilização na aplicação destino.

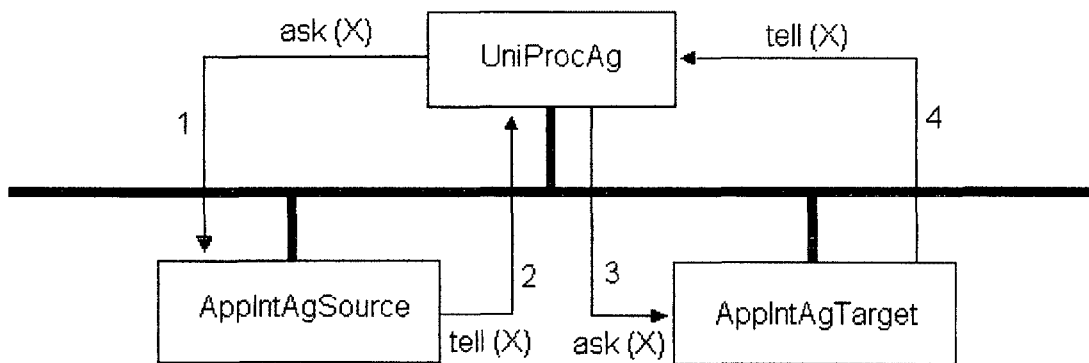
```
<?xml version="1.0" ?>
<!ELEMENT TransferenciaAluno (CodigoAluno,
                             NomeAluno,
                             Disciplinas)>
<!ELEMENT CodigoAluno (#PCDATA)>
<!ELEMENT NomeAluno (#PCDATA)>
<!ELEMENT Disciplinas (Disciplina+)>
<!ELEMENT Disciplina (CodigoDisciplina,
                     NomeDisciplina,
                     PeriodoOferta,
                     NotaFinal,
                     NumeroFaltas,
                     ResultadoFinal)>
<!ELEMENT CodigoDisciplina (#PCDATA)>
<!ELEMENT NomeDisciplina (#PCDATA)>
<!ELEMENT PeriodoOferta (#PCDATA)>
<!ELEMENT NotaFinal (#PCDATA)>
<!ELEMENT NumeroFaltas (#PCDATA)>
<!ELEMENT ResultadoFinal (#PCDATA)>
```

Além disso, o manuseio deste objeto de dados é possibilitado pela utilização da API DOM que permite visualizar tais dados como uma árvore de objetos. Desta forma, o processamento destes dados, quer seja pelo agente de processo, quer seja pelo agente de interface de aplicação é, sobremaneira, facilitado.

### 5.5.3 Protocolo de Comunicação

O protocolo de comunicação definido para o processo virtual de transferência de alunos especifica a sequência de mensagens que serão trocadas entre os agentes de processo e interface de aplicação, caracterizando um processo de colaboração que visa implementar o processo virtual proposto. Tal protocolo é visualizado na Figura 2.

Figura 2: Protocolo de comunicação do processo virtual transferência de alunos



O protocolo acima define a seqüência de troca de dados entre os agentes na implementação da transferência de alunos entre Instituições de Ensino Superior. Nele, uma mensagem do tipo *ask(X)* é enviada, pelo agente de processo, ao agente de interface da aplicação fonte, solicitando, via uma String SQL, a recuperação de dados da base de dados da aplicação. O agente de interface desta aplicação processa a mensagem, gerando como resultado um objeto de dados XML que, via uma mensagem *tell(X)*, é retornado ao agente de processo. Este, por sua vez, envia uma mensagem *ask(X)*, contendo o objeto de dados XML recebido, ao agente de interface da aplicação destino. O processamento desta mensagem pelo agente de interface da aplicação destino implica na atualização da base de dados da aplicação destino e o posterior envio de uma mensagem *tell(X)* ao agente de processo sinalizando se a atualização dos dados foi bem sucedida ou não. Finalmente o agente de processo sinaliza ao usuário que delegou a tarefa de executar o processo virtual de transferência de alunos se o processo foi executado com êxito ou não.

Como visto, o protocolo de comunicações definido para o processo virtual de transferência de alunos baseia-se na troca de mensagens KQML entre os agentes de software sendo, tais mensagens, especificadas na Tabela 2.

Tabela 2: Mensagens KQML do protocolo de comunicação do processo virtual de transferência de alunos

Ask (X)	Tell (X)
<pre>( ask   :ontology UniversityVirtualProcess   :language SQL or XML   :receiver AppIntAgSource or ApplIntAgTarget   :replay-with rRD or rUD   :content StringSQL or DataObjectXML )</pre>	<pre>( tell   :ontology UniversityVirtualProcess   :receiver UniProcAg   :replay-with rRD or rUD   :content DataObjectXML or Status )</pre>

#### 5.5.4 Agente de Processo

O agente de processo utilizado para coordenar a execução do processo virtual transferência de alunos foi implementado com a linguagem de programação JAVA. Tal agente é responsável pelo roteamento das mensagens KQML entre os agentes de interface de aplicação, coordenando, desta forma, o acesso às bases e a troca de dados envolvidas no processo. Alguns detalhamentos sobre a implementação do agente de processo merecem destaque, mais especificamente aqueles relacionados ao ciclo de vida do agente e ao processo de comunicação do mesmo com os agentes de interface de aplicação.

Um extrato do código do agente é dado abaixo.

```
import saci.*;                                // API SACI
public class UniProcAg extends Agent {
    public static void main (String args[]) {
        UniProcAg uvpa = new UniProcAg();    // instancia o agente
        if ( uvpa.enterSoc("NomedoAgente") ) { // introduz o agente na sociedade
            uvpa.run();                       // executa o agente
            uvpa.leaveSoc();                  // abandona a sociedade
            System.exit(0);
        }
    }
    public void run() {
        // implementa o comportamento do agente
    }
} // classe UniProcAg
```

Inicialmente o agente é instanciado dentro do método *main()*. Em seguida, ele é introduzido em uma sociedade de agentes através do método *enterSoc("NomeDoAgente")*. O parâmetro *NomeDoAgente* identifica o agente na sociedade, quer seja perante os outros agentes, quer seja perante o facilitador do ambiente.

Em seguida, o método *run()* implementa o comportamento do agente. Durante a sua execução, o agente de processo é responsável pela montagem de uma mensagem KQML que será enviada sincronamente ao agente de interface da aplicação fonte através do método *ask()*. Tal mensagem solicita um serviço de recuperação de dados do agente de interface da aplicação fonte através de uma String SQL que está contida no campo de conteúdo da mensagem. De posse da resposta recebida, o agente de processo monta uma nova mensagem, agora ao agente de interface da aplicação destino, contendo como conteúdo o objeto de dados recebido na resposta do agente de interface da aplicação fonte. Após o envio síncrono desta mensagem ao agente de interface da aplicação destino e o recebimento da sua correspondente resposta, o agente sinaliza ao usuário se a consecução do processo foi bem sucedida ou não. Um extrato do código que explicita este comportamento do agente é mostrado a seguir.

```
public void run() {
    try {
        Message rd = new Message ("ask" +
            ":ontology UniversityVirtualProcess" +
            ":language SQL" +
            ":receiver AppIntAgUNC" +
            ":replay-with rRD" );
        rd.put("content",StringSQL);

        Message ard = mbox.ask (rd);
        Message ud = new Message ("ask" +
            ":ontology UniversityVirtualProcess" +
            ":language XML" +
            ":receiver AppIntAgFURB" +
            ":replay-with rUD" );
        ud.put("content",ard.get("content"));

        Message aud = mbox.ask (ud);
        // sinaliza Status do processo ao usuário
    } catch (Exception e) {
        System.err.println("Error " + e);
    }
}
```

Este conjunto de procedimentos que correspondem a instanciar o agente, colocá-lo em uma sociedade, executá-lo e fazê-lo deixar a sociedade identificam o ciclo de vida do agente. Para maiores esclarecimentos, a totalidade do código fonte do agente de processo implementado para a validação do modelo proposto pode ser encontrada em anexo (Anexo 1).

### 5.5.5 Agente de Interface de Aplicação

Para finalizar a verificação da solução de integração proposta foram implementados os agentes de interface de aplicação envolvidos no processo de transferência de alunos. Tais agentes, desenvolvidos com a linguagem de programação JAVA, são responsáveis por interagir com as aplicações acadêmicas residentes nas Instituições fonte e destino. Em resposta à mensagens KQML eles executam serviços de acesso, recuperação e atualização de informações nas bases de dados, bem como a formatação das mesmas de acordo com o objeto de dados definido para o processo virtual. Para tanto, tais agentes se utilizam das tecnologias JAVA JDBC, para acesso às bases de dados, e XML JAXP para o acesso e manipulação dos objetos de dados XML.

Uma síntese do código do agente implementado é mostrado a seguir.

```
import saci.*;
import java.sql.*;
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;

public class AppIntAg extends Agent {
    public static void main (String args[]) {
        Agent aia = new AppIntAg();           // instancia o agente
        if ( aia.enterSoc("NomeDoAgente") ) { // introduz o agente na sociedade
            aia.initAg(null);                 // inicia o agente
            aia.run();                         // executa o agente
        }
    }
    public void initAg (String[] args ) {
        // inicia o agente e comunica ao facilitador suas habilidades
    }
    public void run() {
        // implementa o comportamento do agente
    }
} // classe AppIntAg
```

Da mesma forma que o agente de processo, inicialmente o agente de interface de aplicação é instanciado, no método *main()*. Após isto, ele será introduzido em uma sociedade de agentes através do método *enterSoc("NomeDoAgente")* que possui como parâmetro um nome para o agente; nome este que o identificará na sociedade perante os outros agentes e o facilitador do ambiente.

Em seguida, o método *initAg()* permite inicializar o agente, indicando ao facilitador que o referido agente está apto a responder mensagens KQML *ask()*, definidas dentro da ontologia *UniversityVirtualProcess*, conforme o código do método abaixo explicitado.

```
public void initAg () {
    try {
        mbox.advertise("ask", null, "UniversityVirtualProcess", null);
    } catch (Exception e) {
        System.err.println("Error starting agent: " + e);
    }
}
```

Por último, o método *run()* implementa o comportamento do agente que fica em estado de espera para receber mensagens. A mensagem a ser recebida pelo agente possuirá como conteúdo uma String SQL para ser feita uma consulta às bases de dados da aplicação fonte ou um objeto de dados XML para que seja feita a atualização destes dados nas bases da aplicação destino. Ao receber uma mensagem, através do método *polling()*, o agente verifica se está apto a processá-la e, em isto sendo possível, analisa a mensagem e identifica o tratamento a ser dado a ela. O tratamento das mensagens se dará, então, através do método *dataXML()* que recupera dados das bases de dados das aplicações e os formata de acordo com o objetos de dados XML associado ao processo ou do método *dataStatus()* que atualiza as bases de dados das aplicações a partir do objeto de dados XML recebido. Neste último caso, o agente se utiliza de um parser XML para, a partir do objeto de dados, obter uma objeto DOM e acessar e manipular os elementos do objeto de dados. Por fim, o agente monta uma mensagem *tell()* em resposta ao agente de processo, tendo esta, como conteúdo, uma resposta apropriada à

mensagem recebida. Um extrato do código JAVA que implementa este comportamento do agente é dado a seguir.

```

public void run() {
    while (true) {
        try {
            Message m = mbox.polling(); // recebe mensagem
            if ( m.get("performative").equals("ask") &&
                m.get("ontology").equals("UniversityVirtualProcess") ) {
                Message a = new Message ("(tell)");
                a.put("ontology", m.get("ontology"));
                a.put("receiver", m.get("sender"));
                a.put("in-reply-to", m.get("reply-with"));
                if ( m.get("language").equals("SQL") )
                    a.put("content", dataXML( (String)m.get("content") ));
                if ( m.get("language").equals("XML") )
                    a.put("content", dataStatus( (String)m.get("content") ));
                mbox.sendMsg(a); // envia mensagem ao Agente de Processo
            }
        } catch (Exception e) {
            // implementa o tratamento de exceções
        }
    }
}

```

Para maiores esclarecimentos, a totalidade do código fonte do agente de interface de aplicação implementado pode ser encontrada em anexo (Anexo 2).

### 5.5.6 Execução

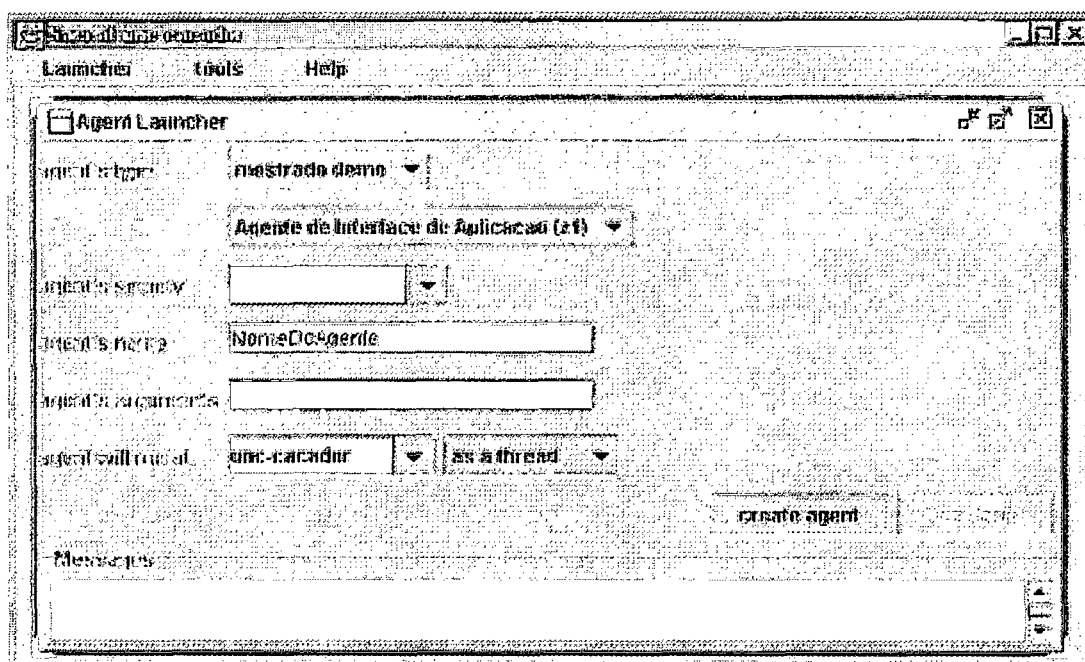
A verificação da viabilidade da arquitetura de integração proposta se deu através da execução dos agentes anteriormente descritos em uma rede de computadores. Neste contexto, simulou-se a existência de 4 Instituições de Ensino tendo como base 4 máquinas com o protocolo TCP/IP instalados. Cada uma destas máquinas pressupunha a existência de uma aplicação acadêmica e, para tanto, criou-se um modelo simplificado das bases de dados de uma aplicação acadêmica real nas plataformas Access, Paradox, Oracle e Dbase.



Para a execução dos agentes optou-se por um ambiente de execução de agentes denominado Saci (*Simple Agent Communication Infrastructure*). Tal ambiente torna transparente ao programador dos agentes os aspectos de comunicação distribuída, sendo um ambiente de bom desempenho quando comparado com outras ferramentas similares: JKQML versão 5.1a, desenvolvida sob o projeto AlphaWorks da IBM; Jackal versão 3.1, desenvolvido na UMBC e o FIPA-OS versão 1.1, desenvolvido pela Nortel Networks (Hübner & Sichman, 2000). O Saci foi desenvolvido com base na especificação KQML e possui como principais características o fato dos agentes utilizarem KQML para se comunicar (o ambiente possui funções para compor, enviar e receber mensagens KQML); os agentes são identificados por um nome, sendo transparente sua localização na rede; um agente pode conhecer outros agentes por meio de um serviço de páginas amarelas; agentes podem ser implementados como *applets*; agentes podem ser iniciados remotamente e podem também ser monitorados. Por fim, a arquitetura do ambiente Saci permite a entrada e saída de agentes de sociedades de agentes e o envio e recebimento de mensagens, bem como o anúncio das habilidades dos agentes ao facilitador do ambiente.

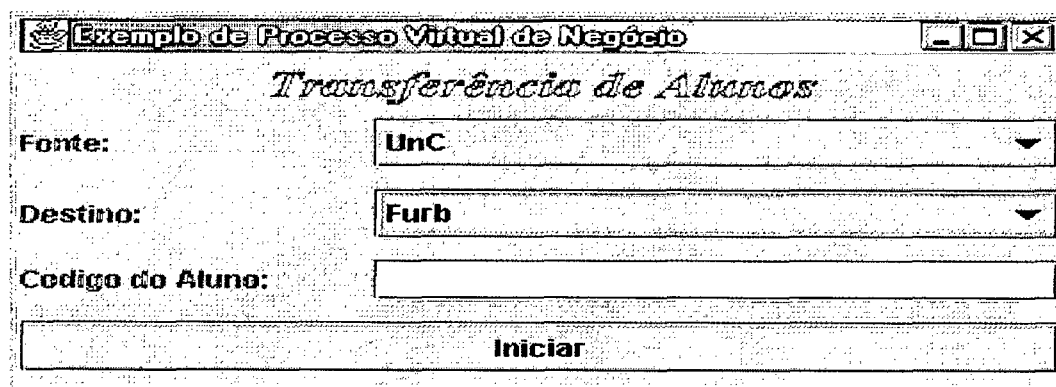
Para a execução da solução de integração inicia-se, primeiramente, o ambiente Saci nas máquinas que contém as aplicações acadêmicas. Nestes ambientes instanciam-se os agentes de interface das aplicações que foram projetados para acessar aquelas bases de dados. A partir de então, os agentes de interface de aplicação ficarão em estado de espera por mensagens dos agentes de processo. O recebimento de uma mensagem implicará em que o agente de interface de aplicação execute, nas bases de dados, os serviços apropriados àquela mensagem, quer seja recuperando dados das mesmas, quer seja atualizando-as de acordo com o objeto de dados recebido. Para fins de esclarecimento, a interface de instanciação de agentes do ambiente Saci é mostrada na Figura 3.

Figura 3: Interface de instanciação de agentes do ambiente Saci



Uma vez instanciados os agentes de interface de aplicação nas máquinas de cada uma das Instituições de Ensino, a instituição de desejar executar uma transferência de alunos inicia este processo carregando a sua interface. Tal interface é visualizada na Figura 4.

Figura 4: Interface do processo virtual de transferência de alunos



Através desta interface é permitido ao usuário configurar o agente de processo através da indicação das instituições fonte e destino, bem como do código do aluno que será transferido. Uma vez configurado, o agente de processo será instanciado e coordenará a execução do processo através da cooperação, por meio das mensagens KQML, com os agentes de interface da aplicação. Por fim, o agente sinalizará ao usuário final o resultado da execução do processo virtual, concluindo, desta forma, o seu ciclo de execução.

### **5.6 Avaliação e Perspectivas**

O processo de verificação da viabilidade da solução de integração proposta possibilitou que a mesma fosse executada dentro de uma arquitetura de computação distribuída. Dentro deste contexto, foi possível comprovar a viabilidade das idéias contidas na solução e a sua correteude de execução, dentro do que a mesma se propunha. Após superadas as dificuldades iniciais de configuração das máquinas para possibilitar a execução do Saci, o restante das simulações correram a contento. O agente de processo implementado comunicou-se corretamente com os agentes de interface que, por sua vez, realizavam com correteude as operações de recuperação e atualização das dados a partir das bases de dados. Da mesma forma, a manipulação do objeto de dados ficou sobremaneira facilitada a partir da utilização da API DOM, levando a crer que a utilização da XML no desenvolvimento de aplicações não introduz grandes complexidades ao processo. Do ponto de vista do processo virtual, este foi executado a contento através do processo de cooperação entre os agentes.

De um ponto de vista mais pessoal, a definição e verificação da viabilidade da arquitetura proposta possibilitou o contato com tecnologias interessantes como a linguagem XML e as tecnologias e ela associadas, a API DOM, a linguagem de comunicação de agentes KQML e o ambiente de execução de agentes Saci com todos os conceitos e tecnologias e ele inerentes, bem como as tecnologias Java e JDBC.

Por outro lado, o trabalho ora apresentado caracteriza-se como um ponto de partida em uma incursão na problemática de integração de aplicações empresariais

como forma de suportar o processamento de negócios entre organizações parceiras comerciais. Assim vislumbram-se algumas perspectivas futuras oriundas deste trabalho.

Uma primeira investigação que merece ser aprofundada diz respeito a definição de uma linguagem para a modelagem dos processos virtuais. Poder-se-ia investigar os benefícios advindos da definição de uma linguagem que formalizasse a especificação de tais entes abstratos da arquitetura. Cabe aqui também uma investigação das propriedades da Linguagem XML como ferramenta para a definição de tal linguagem de modelagem de processos virtuais. Este formalismo, então, serviria como elemento de entrada para os agentes de processo durante a execução do processo virtual. Ainda dentro deste contexto, esta linguagem de modelagem poderia também definir e isolar do resto da arquitetura as regras de negócios a serem aplicadas aos dados. Neste caso, ter-se-ia uma menor interdependência dos agentes e da lógica dos processos virtuais de negócio.

Outra perspectiva que se aponta na solução proposta é a possibilidade de inclusão de inteligência nos agentes de processo e de interface de aplicação. Neste caso, agentes de processo poderiam aplicar complexas regras de negócios aos dados envolvidos em um processo virtual de negócios.

Deve-se também aprofundar investigações no que diz respeito à performance da arquitetura proposta e de como ela se portaria em ambientes reais de execução. Além disso, o estudo e avaliação de outras tecnologias para a implementação da arquitetura proposta, como *message brokers* e objetos distribuídos, poderia propiciar *benchmarks* comparativos, bem como identificar pontos fortes e fracos da arquitetura, além de melhorias e evoluções da mesma.

Um último problema que deveria ser investigado seria a questão da segurança. Uma vez que os processos virtuais incluem a transferência e manipulação de dados entre os parceiros comerciais e que tais dados podem ser sigilosos para as organizações, a questão da segurança no transporte destes dados por uma infra-estrutura de comunicação, como por exemplo a Internet, assume papel preponderante na arquitetura.

## 6 CONCLUSÃO

Num primeiro momento deve-se ressaltar a importância da integração de aplicações empresariais para as organizações. O novo cenário mundial de negócios, cada vez mais globalizado, competitivo e distribuído, implica em que as organizações se utilizem das tecnologias de telecomunicações existentes para estreitar os relacionamentos com clientes e parceiros comerciais a fim de gerar novas formas de fazer negócios e obter ganhos competitivos frente a seus concorrentes. Para tanto, cada vez mais as organizações terão que investir em soluções de integração, uma vez que o desenvolvimento de novas soluções e a pura e simples troca das aplicações atualmente existentes se inviabiliza por questões de tempo e custo proibitivos. Desta forma, a solução para tais problemas passa pela combinação das arquiteturas de hardware e software das organizações como forma viabilizar novas formas de fazer negócio.

Parece também ser clara uma tendência de utilização cada vez maior da linguagem XML como língua franca para a representação, transporte e manipulação de dados em soluções de integração de aplicações empresariais. Como comprovação deste fato verifica-se uma adoção cada vez maior do XML como formato de permuta de dados dos softwares de gestão empresarial, como o SAP R/3, e a disseminação de padrões para comércio eletrônico, como o BizTalk, ebXML e RosettaNet, que se utilizam do XML para fins de padronização de documentos de negócios a serem compartilhados por empresas. Dentro deste contexto, espera-se também uma adoção cada vez maior da linguagem XSLT, a linguagem de transformação de dados associada ao XML, como mecanismo de transformação de dados, facilitando assim a integração de aplicações empresariais que utilizem diferentes dialetos do XML.

Com relação à arquitetura de integração proposta, a utilização de agentes de software e XML se mostrou adequada para a solução do problema de integração. Com a utilização destas tecnologias foi possível manter a independência das aplicações das organizações envolvidas no processo de integração, bem como solucionar a questão da heterogeneidade dos ambientes. Além disso, foi possível manter a semântica dos dados que foram utilizados no processo de integração, sendo a manipulação destes, facilitada pela utilização de tecnologias que visualizam um documento XML como uma árvore de

objetos. Desta forma, a arquitetura proposta se caracteriza como uma solução padronizada, aberta, e que se utiliza de tecnologias de baixa curva de aprendizagem para a sua implementação. Naturalmente, novas simulações e aprofundamentos devem ser feitos na arquitetura como forma de levantar seus pontos fortes e fracos, referendando ou não sua viabilidade.

Por outro lado, se verifica também que o desenvolvimento do trabalho foi de grande valia pois propiciou um aprofundamento nas questões e tecnologias de integração de aplicações empresarias e no uso de agentes de software e da linguagem XML na solução de tais problemas. Além disso, foi possível um contato com um ambiente de comunicação de agentes de software, com a problemática de comunicação de agentes e a linguagem KQML, bem como o desenvolvimento baseado na linguagem Java, na tecnologia JDBC e na manipulação de documentos XML através das APIs padronizadas.

Finalmente, pode-se dizer que o presente trabalho possibilitou uma primeira incursão na problemática de integração de aplicações e, desta forma, o aprofundamento de estudos no escopo deste problema se faz necessário, principalmente em relação a uma maior avaliação das tecnologias aqui propostas, bem como a utilização de um leque de outras tecnologias na solução de tais problemas.

## 7 BIBLIOGRAFIA

ABATE, Robert; BURKE, Joseph P.; AIKEN, Peter. *Achieving EAI with Service-Based Architectures*. Addison-Wesley Pub Co, 2001.

BEDUNAH, John B. *XML: The Future Of The Web*. ACM Crossroads Student Magazine, 1999, obtido via <http://www.acm.org/crossroads/xrds6-2/future.html> em 28/11/2000.

BERRY, Nina M.; PANCERELLA, Carmen M. *Agent-Based Enterprise Integration*. Proceedings of The Fourth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents (PAAM 99), April 19-21, 1999, London, UK.

BORDINI, Rafael Heitor; VIEIRA, Renata; MOREIRA, Álvaro Freitas. *Fundamentos de sistemas multiagentes*. XX Jornada de Atualização em Informática (JAI), volume 2, pp. 3-44, Fortaleza, 2001.

BOS, Bert. *XML in 10 points*. W3C Consortium Web Page, 1999, obtido via <http://www.w3.org/XML/1999/XML-in-10-points> em 21/11/2000.

BOSAK, Jon. *XML, Java, and the future of the Web*. Sun Microsystems, Março 1997, obtido via <http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm> em 22/08/2000.

BOX, Don; SKONNARD, Aaron; LAM, John. *Essencial XML*. Addison Wesley, July 2000.

BRAY, Tim et al. *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Consortium, October 2000, obtido via <http://www.w3.org/TR/2000/REC-xml-20001006> em 22/11/2000.

BUYENS, Marc. *Enterprise Application Integration (EAI)*. Xpragma, September 1999, obtido via [http://www.xpragma.com/eai\\_wp.htm](http://www.xpragma.com/eai_wp.htm) em 20/04/2001.

DRUMMOND, Rik. *BizTalk, eCo, e'speak and cXML – Why do we need all of these?* CommerceNet's Technology Report, Vol 1, No. 2, June, 1999, obtido via [http://www.commerce.net/research/reports/1999/99\\_02\\_ec.pdf](http://www.commerce.net/research/reports/1999/99_02_ec.pdf) em 28/08/2000.

FININ, T. et al. **KQML as an agent communication language**. In Proceedings of the Third International Conference on Information and Knowledge Management, ACM Press, 1994.

FRANKLIN, S.; GRAESSER, A. **Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents**. In Proceedings of the Third International Workshop on Agent Theories, Springer-Verlag, 1996.

GENEESERETH, M. R.; KETCHPEL, S. P. *Software Agents*. Communications of the ACM, July 1994, pp. 48-53, 147.

GREENSPAN, Jay. *Understanding XSLT*. HotWired Webmonkey, October, 1998, obtido via <http://hotwired.lycos.com/webmonkey/98/43/index2a.html?tw=authoring> em 20/12/2000.

HAROLD, Elliotte Rusty. *XML Bible*. 2ª ed., IDG Books, 2001, 1206 p.

HÜBNER, Jomi Fred; SICHMAN, Jaime Simão. *Saci Programming Guide*. Laboratório de Técnicas Inteligentes da Escola Politécnica da USP, 2001 obtido via <http://www.lti.pcs.usp.br/saci/> em 26/09/2001.

HÜBNER, Jomi Fred; SICHMAN, Jaime Simão. *SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes*. In: IBERAMIA/SBIA, 2000, Atibaia, São Paulo.

INMON, William H. *A brief history of Integration*. EAI Journal, 1999, obtido via <http://www.eaijournal.com/Article.asp?ArticleID=119&DepartmentId=5> em 10/05/2001.



JOHNSON, Mark. *XML for the absolute beginner: A guided tour from HTML to processing XML with Java*. Java World, Abril 1999, obtido via <http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml.html> em 24/08/2000.

KHARE, Rohit; RIFKIN, Adam. *X Marks the Spot: Using XML to Automate the Web*. IEEE Internet Computing, July/August 1997, Volume 1, Number 4, Pages 78-87, obtido via <http://www.cs.caltech.edu/~adam/papers/xml/x-marks-the-spot.html> em 24/07/2000.

KING, Nelson. *EAI Directions*. Intelligent Enterprise, March, 2000 Volume 3, Number 4, obtido via <http://www.intelligententerprise.com/000301/feat2.shtml> em 28/05/2001.

LAURENT, Simon S. *XML: A Primer*. M&T Books, IDG Books Worldwide, 348pp, 1998.

LINTHICUM, David S. *EAI – Application Integration Exposed*. Software Magazine, Feb/Mar 2000, obtido via <http://www.softwaremag.com/archive/2000feb/EAI.html> em 24/04/2001.

LINTHICUM, David S. *Enterprise Application Integration Exposed*. Addison-Wesley Pub Co, 1999.

LINTHICUM, David S. *Next-Generation EAI: Eight Prophecies for 2001*. ebizQ: The Portal for e-Business Integration. February 2001, obtido via [http://eai.ebizq.net/str/linthicum\\_1.html](http://eai.ebizq.net/str/linthicum_1.html) em 07/05/2001.

MAAMAR, Zakaria; KETTANI, Driss; SAHLI, Nabil. *Software Agents for Enterprise Application Integration*. OOPSLA Workshop on Business Object Design and Implementation, October 15-19/2000, Minneapolis, Minnesota USA, obtido via <http://jeffsutherland.org/oopsla2000/zakaria/zakaria.htm> em 31/07/2001.

MARUYAMA, Hiroshi, URAMOTO, Naohico; TAMURA, Kent. *The Power of XML: XML's Purpose and Use in Web Applications*. IBM Research, Outubro 1998, Obtido via <http://www-4.ibm.com/software/developer/library/xmlpower/xmlpower.html> em 25/09/2000.

MASLER, Jean-François. *EAI na convergência das aplicações empresariais*. Revista Redes, Setembro de 2000, obtido via <http://www.fbnet.pt/red/0900/a03-00-00.shtml> em 16/04/2001.

MORGENTHAL, J.P. *Enterprise Applications Integration with XML and Java*. Prentice Hall, 2000.

NAGARAJAN, Rajaram; WHITMAN, Larry; CHERAGHI, S. Hossein. *Enterprise Integration*. Proceedings of The 4th Annual International Conference on Industrial Engineering Theory, Applications and Practice November 17-20, San Antonio, Texas, USA.

NELSON, Chris. *The ABC of EDI*. The Electronic Commerce and EDI Centre for Wales, 1994, obtido via <http://www.edi.wales.org/feature4.htm> em 18/10/2000.

NWANA, H. *Software Agents: An Overview*. Knowledge Engineering Review, Vol. 11, N. 3, pp. 1-40, September 1996.

PAN, J. Y. C.; TENENBAUM, J. M. *An Intelligent Agent Framework for Enterprise Integration*. IEEE Transactions on Systems, Man and Cybernetics, vol 21, n° 6, November/December, 1991.

PARUNAK, H. V. D. *Technologies for Virtual Enterprises*. Agility Journal, 1997, obtido via <http://www.erim.org/~vparunak/papers.htm#VEandEC> em 08/08/2001.

ROWELL, Michael. *Understanding EAI: Enterprise Application Integration*. Sams, 2001.

RUH, William A.; MAGINNIS, Francis X.; BROWN, William J. *Enterprise Application Integration: a Wiley tech brief*. New York: John Wiley & Sons, 2000.

SCHLOSS, Bob. *Ten best bets for XML applications: wondering where to begin with XML? These projects won't turn into dead ends*. IBM T. J. Watson Research Center, Abril 2000, Obtido via <http://www-4.ibm.com/software/developer/library/tenxmlapps/> em 30/10/2000.

SHOHAM, Y. *An Overview of Agent-Oriented Program*. In Software Agents, ed. J. M. Bradshaw, AAAI Press, 1997.

TOLOSA, Gabriel Hernan; BORDIGNON, Fernando Raul Alfredo. *Revisión: tecnología de agentes de software*. IBICT – Revista Ciencia da Informação: Brasília, v.28, n.3, p. 302-309, 1999, obtido via <http://www.ibict.br/cionline/280399/28039909.htm> em 23/08/2001.

WALSH, Norman. *A Technical Introduction to XML*. World Wide Web Journal, October 1998, obtido via <http://www.xml.com/pub/a/98/10/guide0.html> em 22/11/2000.

WHITESIDE, R. A.; FRIEDMAN-HILL, E. J.; DETRY, R. J. *PRE: A framework for enterprise integration*. Proceedings of Design of Information Infrastructure Systems for Manufacturing 1998 (DIISM ' 98), Fort Worth, Texas, May 1998.

WIEDERHOLD, G. *Interoperation, mediation and ontologies*. In Proceedings of the International Symposium on Fifth Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge-Bases, volume W3, pages 33-48, Tokyo, Japan, 1994.

WOOLDRIDGE, M.; JENNINGS, N. R. *Pitfalls of Agent-Oriented Development*. Proceedings of the Second International Conference on Autonomous Agents (Agents ' 98), Minneapolis, Minnesota, May 1998.

WOOLDRIDGE, M.; JENNINGS, N. R. *Software Agents*. IEEE Review January 1996, pp. 17-20.

## **8 ANEXOS**

## 8.1 ANEXO 1 - Código Fonte do Agente de Processo

```

import saci.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class UniProcAg extends Agent {

    String sourceAG, targetAG, consultaSQL;

    public UniProcAg() {

        enterSoc("UniProcAg");
        String universityNames[] = {"UnC", "Furb", "Univalli", "UFSC" };
        JFrame frame = new JFrame("Exemplo de Processo Virtual de Negócio" );

        Container container = new Container();
        container = frame.getContentPane();
        GridBagLayout gbLayout = new GridBagLayout();
        container.setLayout( gbLayout );
        GridBagConstraints gbConstraints = new GridBagConstraints();

        JLabel titulo = new JLabel("Transferência de Alunos", JLabel.CENTER);
        titulo.setFont( new Font("TimesRoman", Font.BOLD + Font.ITALIC, 20) );
        gbConstraints.fill = GridBagConstraints.HORIZONTAL;
        gbConstraints.gridx = 0;
        gbConstraints.gridy = 0;
        gbConstraints.gridwidth = 2;
        gbConstraints.gridheight = 1;
        gbLayout.setConstraints( titulo, gbConstraints );
        container.add(titulo);

        JLabel fonte = new JLabel("Fonte:");
        gbConstraints.fill = GridBagConstraints.HORIZONTAL;
        gbConstraints.gridx = 0;
        gbConstraints.gridy = 1;
        gbConstraints.gridwidth = 1;
        gbConstraints.gridheight = 1;
        gbLayout.setConstraints( fonte, gbConstraints );
        container.add(fonte);

        final JComboBox unifonte = new JComboBox(universityNames);
        unifonte.setMaximumRowCount(3);
        gbConstraints.weightx = 1;
        gbConstraints.weighty = 1;
        gbConstraints.gridx = 1;
    }
}

```

```
gbConstraints.gridy = 1;
gbConstraints.gridwidth = 2;
gbConstraints.gridheight = 1;
gbLayout.setConstraints( unifonte, gbConstraints );
container.add(unifonte);

JLabel destino = new JLabel("Destino:");
gbConstraints.fill = GridBagConstraints.HORIZONTAL;
gbConstraints.gridx = 0;
gbConstraints.gridy = 2;
gbConstraints.gridwidth = 1;
gbConstraints.gridheight = 1;
gbLayout.setConstraints( destino, gbConstraints );
container.add(destino);

final JComboBox unidestino = new JComboBox(universityNames);
unidestino.setMaximumRowCount(3);
gbConstraints.weightx = 1;
gbConstraints.weighty = 1;
gbConstraints.gridx = 1;
gbConstraints.gridy = 2;
gbConstraints.gridwidth = 2;
gbConstraints.gridheight = 1;
gbLayout.setConstraints( unidestino, gbConstraints );
container.add(unidestino);

JLabel codigoAluno = new JLabel("Codigo do Aluno:");
gbConstraints.fill = GridBagConstraints.HORIZONTAL;
gbConstraints.gridx = 0;
gbConstraints.gridy = 3;
gbConstraints.gridwidth = 1;
gbConstraints.gridheight = 1;
gbLayout.setConstraints( codigoAluno, gbConstraints );
container.add(codigoAluno);

final JTextField campoCodAluno = new JTextField(20);
gbConstraints.gridx = 1;
gbConstraints.gridy = 3;
gbConstraints.gridwidth = 1;
gbConstraints.gridheight = 1;
gbLayout.setConstraints( campoCodAluno, gbConstraints );
container.add(campoCodAluno);

JButton iniciar = new JButton( "Iniciar" );
gbConstraints.weightx = 1;
gbConstraints.weighty = 1;
gbConstraints.fill = GridBagConstraints.HORIZONTAL;
gbConstraints.gridx = 0;
```

```

gbConstraints.gridy = 4;
gbConstraints.gridwidth = 2;
gbConstraints.gridheight = 1;
gbLayout.setConstraints( iniciar, gbConstraints );
container.add(iniciar);

ActionListener handler = new ActionListener() {
    public void actionPerformed (ActionEvent evt) {
        run((String)unifonte.getSelectedItemAt(),
            (String)unidestino.getSelectedItemAt(),
            campoCodAluno.getText());
    }
};

iniciar.addActionListener(handler);
campoCodAluno.addActionListener(handler);

frame.setSize( 400, 200 );
frame.show();
frame.addWindowListener(
    new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            System.exit( 0 );
        }
    }
);
}

```

```

public void run(String f, String d, String alu) {
    if ( f == "UnC" )
        sourceAG = "AppIntAgUNC";
    else if ( f == "Furb" )
        sourceAG = "AppIntAgFURB";
    else if ( f == "Univalli" )
        sourceAG = "AppIntAgUNIVALLI";
    else if ( f == "UFSC" )
        sourceAG = "AppIntAgUFSC";
    if ( d == "UnC" )
        targetAG = "AppIntAgUNC";
    else if ( d == "Furb" )
        targetAG = "AppIntAgFURB";
    else if ( d == "Univalli" )
        targetAG = "AppIntAgUNIVALLI";
    else if ( d == "UFSC" )
        targetAG = "AppIntAgUFSC";
}

```

```

consultaSQL = "\"SELECT CodigoAluno, NomeAluno FROM Alunos WHERE
                CodigoAluno=\"" + alu + "\"\"";

try {
    Message rd = new Message ("(ask" +
        " :ontology UniversityVirtualProcess" +
        " :language SQL" +
        " :replay-with rRD )" );
    rd.put("receiver",sourceAG);
    rd.put("content",consultaSQL);
    Message ard = mbox.ask (rd);
    Message ud = new Message ("(ask" +
        " :ontology UniversityVirtualProcess" +
        " :language XML" +
        " :replay-with rUD )" );
    ud.put("receiver",targetAG);
    ud.put("content",ard.get("content"));
    Message aud = mbox.ask (ud);
    String status = (String)aud.get("content");
    if ( status.equals("OK") )
        JOptionPane.showMessageDialog( null,"Processo Realizado com Sucesso!");
    else
        JOptionPane.showMessageDialog( null,"Problemas na complementação do
                                        processo!" + "\n" + status + "\n" );
    } catch (Exception e) {
        System.err.println("Error " + e);
    }

    leaveSoc();
}

public static void main (String args[]) {
    new UniProcAg();
}

} // classe UniProcAg

```



## 8.1 ANEXO 2 - Código Fonte do Agente de Interface de Aplicação

```

import saci.*;
import java.sql.*;
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;

public class AppIntAg extends Agent {

    String codAuxAlu, nomAuxAlu;

    public static void main (String args[]) {
        Agent aia = new AppIntAg();
        if ( aia.enterSoc("AppIntAg") ) {
            aia.initAg(null);
            aia.run();
        }
    }

    public void initAg ( ) {
        try {
            mbox.advertise("ask", null, "UniversityVirtualProcess", null);
        } catch (Exception e) {
            System.err.println("Error starting agent: " + e);
        }
    }

    public void run() {
        while (true) {
            try {
                Message m = mbox.polling(); // recebe a mensagem

                if ( m.get("performative").equals("ask") &&
                    m.get("ontology").equals("UniversityVirtualProcess") ) {
                    Message a = new Message ("(tell)");
                    a.put("ontology", m.get("ontology"));
                    a.put("receiver", m.get("sender"));
                    if ( m.get("language").equals("SQL") ) {
                        a.put("in-reply-to", m.get("reply-with"));
                        a.put("content", dataXML( (String)m.get("content") ));
                    }
                }
            }
        }
    }
}

```

```

        if ( m.get("language").equals("XML") ) {
            a.put("in-reply-to", m.get("reply-with"));
            a.put("content", dataStatus( (String)m.get("content") ));
        }
        mbox.sendMessage(a); // envia a mensagem ao UniProcAg
    }
} catch (Exception e) {
    System.err.println("Erro AppIntAgUNC: " + e);
    e.printStackTrace();
}
}
}
}

```

```

public String dataXML(String SQL) {
    Connection con = null;
    String dbUrl = "jdbc:odbc:TransfDB";
    String user = "";
    String password = "";
    String buffer = new String();
    String identacao = " ";
    String cabecalho = "<?xml version='1.0' encoding='ISO-8859-1'?>" + "\n";
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection(dbUrl,user,password);
        Statement stm = con.createStatement();
        SQL = SQL.substring(1,SQL.length()-1);
        ResultSet rs = stm.executeQuery(SQL);
        buffer = cabecalho;
        buffer += "<TransferenciaAluno>" + "\n";
        while (rs.next()) {
            codAuxAlu = rs.getString("CodigoAluno");
            nomAuxAlu = rs.getString("NomeAluno");
        }
        buffer += identacao+"<CodigoAluno>"+codAuxAlu+"</CodigoAluno>"+ "\n";
        buffer += identacao + "<NomeAluno>" + nomAuxAlu + "</NomeAluno>" + "\n";
        String SQL1 = "SELECT CodigoDisciplina, NomeDisciplina, PeríodoOferta,
                        NotaFinal, NumeroFaltas, ResultadoFinal FROM Disciplinas
                        WHERE CodigoAluno='"+ codAuxAlu + "'";
        ResultSet rs1 = stm.executeQuery(SQL1);
        buffer += identacao + "<Disciplinas>" + "\n";
        while (rs1.next()) {
            buffer += identacao + identacao + "<Disciplina>" + "\n";
            buffer += identacao + identacao + identacao + "<CodigoDisciplina>" +
                rs1.getString("CodigoDisciplina") + "</CodigoDisciplina>" + "\n";
            buffer += identacao + identacao + identacao + "<NomeDisciplina>" +
                rs1.getString("NomeDisciplina") + "</NomeDisciplina>" + "\n";
            buffer += identacao + identacao + identacao + "<PeríodoOferta>" +

```

```

        rs1.getString("PeriodoOferta") + "</PeriodoOferta>" + "\n";
    buffer += identacao + identacao + identacao + "<NotaFinal>" +
        rs1.getString("NotaFinal") + "</NotaFinal>" + "\n";
    buffer += identacao + identacao + identacao + "<NumeroFaltas>" +
        rs1.getString("NumeroFaltas") + "</NumeroFaltas>" + "\n";
    buffer += identacao + identacao + identacao + "<ResultadoFinal>" +
        rs1.getString("ResultadoFinal") + "</ResultadoFinal>" + "\n";
    buffer += identacao + identacao + "</Disciplina>" + "\n";
    }
    buffer += identacao + "</Disciplinas>" + "\n";
    buffer += "</TransferenciaAluno>";
    con.close();
    return buffer;
} catch (ClassNotFoundException cnfex) {
    return "Erro ao carregar o Driver JDBC/ODBC";
} catch (SQLException sqlex) {
    return "Erro na Conexão com o Banco de Dados";
}
}
}

```

```

public String dataStatus(String XML) {
    Document document;
    Connection con = null;
    String dbUrl = "jdbc:odbc:TransfMDB";
    String user = "";
    String password = "";
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con = DriverManager.getConnection(dbUrl,user,password);
        Statement stm = con.createStatement();
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.parse(new InputSource(new StringReader(XML)));

        NodeList codAluno = document.getElementsByTagName("CodigoAluno");
        NodeList nomAluno = document.getElementsByTagName("NomeAluno");

        String insertAlunos = "INSERT INTO Alunos (" +
codAluno.item(0).getNodeName() + ", " + nomAluno.item(0).getNodeName() + ")
VALUES (" + codAluno.item(0).getChildNodes().item(0).getNodeValue() + "," +
nomAluno.item(0).getChildNodes().item(0).getNodeValue() + ")";

        int statusInsercaoAlu = stm.executeUpdate(insertAlunos);

        NodeList disAluno =
document.getElementsByTagName("Disciplinas").item(0).getChildNodes();

```

```

        if (disAluno != null) {
            for (int i=1; i<disAluno.getLength(); i++) {
                Node nodes = disAluno.item(i);
                if ( nodes.hasChildNodes() ) {
                    NodeList aux = nodes.getChildNodes();
                    String dados =
""+codAluno.item(0).getChildNodes().item(0).getNodeValue()+"";
                    for (int j=1; j<aux.getLength(); j++) {
                        if ( aux.item(j).hasChildNodes() ) {
                            dados += ","+aux.item(j).getChildNodes().item(0).getNodeValue() + "";
                        }
                    }
                    String insertDisciplinas = "INSERT INTO Disciplinas
(CodigoAluno,CodigoDisciplina,NomeDisciplina,PeriodoOferta,NotaFinal,NumeroFalt
as,ResultadoFinal) VALUES (" + dados + ")";
                    int statusInsercaoDisc = stm.executeUpdate(insertDisciplinas);
                }
            }
        }
        con.close(); // fecha conexão com o banco de dados
        return "OK";
    } catch (SAXException sxe) { // erro gerado durante o parsing do arquivo
        Exception x = sxe;
        if (sxe.getException() != null)
            x = sxe.getException();
        x.printStackTrace();
        return "Erro ao parsear o arquivo/string";
    } catch (ParserConfigurationException pce) { // erro de configuração do parser
        pce.printStackTrace();
        return "Erro de configuração do parser";
    } catch (IOException ioe) { // erro de IO
        ioe.printStackTrace();
        return "Erro de I/O";
    } catch (ClassNotFoundException cnfex) {
        System.err.println("Falha ao carregar o Driver JDBC/ODBC");
        cnfex.printStackTrace();
        return "Falha ao carregar o Driver JDBC/ODBC";
    } catch (SQLException sqllex) {
        System.err.println("Conexao Impossivel!!!");
        sqllex.printStackTrace();
        return "Conexao Impossivel!!!";
    }
}

} // classe AppIntAg

```