

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

Programa de Pós-Graduação Lato Sensu

Engenharia de Software

ESTUDO DE CASO: ANÁLISE DA INTEGRAÇÃO DO SISTEMA

ALFANDEGÁRIO DE UM TERMINAL PORTUÁRIO

RENATO MARQUEZ VIANNA

São Paulo

Julho/2015

RENATO MARQUEZ VIANNA

**ESTUDO DE CASO: ANÁLISE DA INTEGRAÇÃO DO SISTEMA
ALFANDEGÁRIO DE UM TERMINAL PORTUÁRIO**

Monografia apresentada à Pontifícia Universidade
Católica como exigência parcial para obtenção do
título de Especialista em Engenharia de Software.

Prof. Msc. André Luiz Garcia Pereira

São Paulo

Julho/2015

AGRADECIMENTOS

Agradeço a todos os professores mestres e doutores da PUC-SP por contribuírem com a minha formação acadêmica e pessoal, partilhando conhecimento e valores éticos; devido ao aprendizado conquistado foi possível desenvolver esta pesquisa. Sinceros agradecimentos ao professor orientador André Luiz Garcia Pereira, pela sua paciência, compreensão e sinceridade, além de humildemente compartilhar parte de seu vasto conhecimento para que este trabalho fosse conduzido e elaborado da melhor forma possível.

Tão importante quanto, também agradeço à minha família pelo apoio oferecido durante a confecção deste trabalho. Agradeço especialmente à minha irmã por ceder hospedagem em sua residência durante todo o período letivo, em que precisava me deslocar e, pela distância da faculdade até minha casa, certamente sem sua ajuda essa trajetória seria mais dificultosa. Por fim, agradeço a todos os amigos que me apoiaram a concluir esta pesquisa.

“Nossa maior fraqueza está em desistir. O caminho mais certo de vencer é tentar mais uma vez.”

Thomas Edison

RESUMO

O presente trabalho propõe uma arquitetura moderna sustentada pelos conceitos SOA para a estrutura de sistemas de um terminal portuário, a partir da análise da proposta de integração empregada e dos resultados obtidos pós-implantação da arquitetura e das referências bibliográficas estudadas. Inicialmente a pesquisa aborda conceitos teóricos acerca de técnicas de integração de sistemas, como EAI, depois ESB e por fim SOA, apresentando métodos, padrões e boas práticas emergentes ao longo das duas últimas décadas no campo de arquitetura de sistemas, conceituando com maior ênfase o paradigma de serviços para representar os processos de negócio de uma organização. A partir das referências bibliográficas utilizadas para a explicação dos conceitos, o trabalho estuda o cenário anterior e atual do terminal portuário e o compara com uma estrutura sustentada pelo conceito de serviços, concluindo que a proposta de uma arquitetura moderna, atrelada a outras sugestões técnicas e de processo, teria retorno significativo ao terminal.

Palavras-chave: arquitetura, terminal portuário, integração, recinto alfandegado

ABSTRACT

This research proposes a modern architecture supported by the SOA concepts for the structure of port terminal systems, based on the analysis of the proposed integration used and the results obtained after implementation of the architecture and the references studied. Initially, the research addresses theoretical concepts about system integration techniques such as EAI, ESB and then finally SOA, presenting methods, standards and emerging good practices over the past two decades in systems architecture field, with greater emphasis conceptualizing the services paradigm to represent the business processes of an organization. Based on the references used for the explanation of concepts, the research studies the previous and current situation of the port terminal and compares with a structure sustained by the concept of services, concluding that the proposal of a modern architecture, linked to other techniques and suggestions process would have significant return to the terminal.

Key-words: architecture, port terminal, integration, bonded warehouse

LISTA DE ABREVIATURAS E SIGLAS

ERP	Enterprise Resource Planning
EAI	Enterprise Application Integration
B2B	Business To Business
CRM	Customer Relationship Management
SOA	Service-Oriented Architecture
OLAP	On-line Analytical Processing
XML	Extensible Markup Language
DBMS	Database Management System
RMI	Remote Method Invocation
CORBA	Common Object Request Broker Architecture
IIOP	Internet Inter-ORB Protocol
DCOM	Distributed Component Object Model
API	Application Programming Interface
GUI	Graphical User Interface
MOM	Message Oriented Middleware
RPC	Remote Procedure Call
UTF-8	8-bit Unicode Transformation Format
ASCII	American Standard Code for Information Interchange
COBOL	Common Business Oriented Language
SQL	Structured Query Language
IDL	Interface Definition Language
HTTP	Hypertext Transfer Protocol

SOAP	Simple Object Access Protocol
COM	Component Object Model
CIO	Chief Information Officer
FTP	File Transfer Protocol
ESB	Enterprise Service Bus
OOP	Object Oriented Paradigm
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
URL	Uniform Resource Locator
CGI	Common Gateway Interface
ASP	Active Server Pages
UDDI	Universal Description, Discovery, and Integration
IETF	Internet Engineering Task Force
DNS	Domain Name System
REST	Representational State Transfer
SAML	Security Assertion Markup Language
MEP	Message Exchange Pattern
WS-BEPL	Web Service Business Process Execution Language
BL	Bill Of Lading
TOS	Terminal Operating System
TOPO	Terminal Operational Package System
TAS	Truck Appointment System
SISCOMEX	Sistema Integrado de Comércio Exterior
MANTRA	Sistema Integrado de Gerência do Manifesto, do Trânsito e do Armazenamento
DTE	Declaração de Transferência Eletrônica
ADE	Ato Declaratório Executivo
OCR	Optical Character Recognition
ALM	Application Lifecycle Management

LISTA DE FIGURAS

Figura 1 – Caos na integração “ <i>link-a-link</i> ” de sistemas	18
Figura 2 – Message Brokers proposto pela EAI.....	19
Figura 3 – Exemplo de execução de um método de uma API.....	22
Figura 4 – Exemplo de resposta de um método de uma API	22
Figura 5 – Arquitetura centralizada	23
Figura 6 – Arquitetura de duas camadas	24
Figura 7 – Arquitetura de três camadas	25
Figura 8 – Transferência de arquivos	33
Figura 9 – Banco de dados compartilhado	34
Figura 10 – <i>Remote Procedure Call</i>	36
Figura 11 – <i>Middleware</i> Orientado a Mensagens.....	38
Figura 12 – Silos de informação gerados pela arquitetura accidental	42
Figura 13 – Web Service agindo como um provedor de serviço.....	49
Figura 14 – Exemplo de documento XML.....	51
Figura 15 – Bloco <i>definitions</i> de um documento WSDL	52
Figura 16 – Bloco <i>types</i> de um documento WSDL	52
Figura 17 - Bloco <i>message</i> de um documento WSDL	53
Figura 18 – Bloco <i>portTypes</i> de um documento WSDL	53
Figura 19 – Bloco <i>binding</i> de um documento WSDL	54
Figura 20 – Bloco <i>service</i> de um documento WSDL.....	54
Figura 21 – Exemplo de requisição com mensagem SOAP	55
Figura 22 – Exemplo de resposta com mensagem SOAP	55
Figura 23 – MEP <i>Request-Response</i>	59
Figura 24 – MEP <i>Fire-and-Forget</i>	60
Figura 25 – Exemplo de atividade de serviço complexa	61
Figura 26 – Exemplo de atividade de negócio	66

Figura 27 – Exemplo de fluxo regido pela orquestração.....	67
Figura 28 – Orquestração x coreografia	68
Figura 29 – Processo simplificado de importação de cargas.....	75
Figura 30 – Processo simplificado de exportação de cargas	77
Figura 31 – Primeira proposta de integração para o terminal portuário.....	80
Figura 32 – Dependências envolvendo a API DTE.....	83
Figura 33 – Proposta de arquitetura com serviços.....	89
Figura 34 – Processo de descarga de contêiner com <i>web services</i>	91
Figura 35 – Exemplo de utilização de <i>views</i>	94

LISTA DE TABELAS

Tabela 1 – Importância da integração na indústria (em % de respostas)	40
Tabela 2 – Relação de métodos HTTP na arquitetura REST	57
Tabela 3 – Resultados de desempenho dos sistemas do terminal portuário.....	81
Tabela 4 – Comparação de alguns resultados esperados x reais do terminal portuário	82
Tabela 5 – Resultados de desempenho dos novos sistemas do terminal portuário	85
Tabela 6 – Comparação de alguns resultados após implantação do novo HOST	86

SUMÁRIO

1	INTRODUÇÃO	16
2	CONCEITO E MELHORES PRÁTICAS DE EAI	17
2.1	DEFINIÇÃO DE EAI	17
2.2	TIPOS DE EAI	20
2.2.1	Integração de Dados	20
2.2.2	Integração de Aplicações	21
2.2.3	Integração de Métodos	25
2.2.4	Integração de Interfaces Gráficas de Usuário	28
3	BARRAMENTO SOA E EVOLUÇÃO DA EAI	30
3.1	CRITÉRIOS PARA INTEGRAÇÃO DE APLICAÇÕES	30
3.1.1	Acoplamento entre Aplicações	30
3.1.2	Simplicidade de Integração	31
3.1.3	Tecnologias de Integração	31
3.1.4	Formato de Dados	31
3.1.5	Disponibilidade dos Dados	31
3.1.6	Dados ou Funcionalidades	32
3.1.7	Assincronia	32
3.2	PARADIGMAS DE INTEGRAÇÃO	32
3.2.1	Transferência de Arquivos	33
3.2.2	Banco de Dados Compartilhado	34
3.2.3	Execução Remota de Procedimento	36

3.2.4	<i>Middleware</i> Orientado a Mensagens	38
3.3	TOPOLOGIAS DE INTEGRAÇÃO	40
3.3.1	Arquitetura Acidental	41
3.3.2	<i>Brokers</i> de Integração	43
3.3.3	ESB	45
4	CONCEITO E MELHORES PRÁTICAS DE SOA.....	47
4.1	FUNDAMENTOS DE SOA	47
4.2	WEB SERVICES	49
4.2.1	XML	50
4.2.2	WSDL	51
4.2.3	SOAP	54
4.2.4	UDDI	56
4.2.5	REST.....	56
4.3	EXTENSÕES WS-*.....	57
4.3.1	Padrões de Trocas de Mensagens	58
4.3.2	Atividade de Serviço.....	60
4.3.3	Coordenação	62
4.3.4	Transações Atômicas	63
4.3.5	Atividade de Negócio	64
4.3.6	Orquestração	66
4.3.7	Coreografia	68
5	ESTUDO DE CASO: INTEGRAÇÃO DE UM SISTEMA ALFANDEGÁRIO	70
5.1	SOBRE O TERMINAL PORTUÁRIO.....	70
5.1.1	Processo de Avaliação de Sistemas	71
5.1.2	Arquitetura de Sistemas	71
5.1.3	Processos de Negócio	74
5.2	PROPOSTA DA PRIMEIRA SOLUÇÃO DE INTEGRAÇÃO	78

5.2.1	Desempenho da Arquitetura	80
5.2.2	Resultados das Operações Portuárias	81
5.3	PROPOSTA DA SEGUNDA SOLUÇÃO DE INTEGRAÇÃO	83
5.3.1	Desempenho da Arquitetura	84
5.3.2	Resultados das Operações Portuárias	86
5.4	PROPOSTA DE SOLUÇÃO COM ARQUITETURA SOA.....	87
5.4.1	Utilização de Serviços	89
5.4.2	Utilização de Views.....	92
5.4.3	Melhorias aplicadas ao processo de desenvolvimento	95
6	CONCLUSÕES	97
	BIBLIOGRAFIA	98

1 INTRODUÇÃO

Desde o advento dos primeiros computadores até os dias atuais, é notável que a tecnologia da informação tenha tido papel importante no rumo de organizações de diferentes ramos, sejam de atividades primárias, secundárias ou terciárias. A cada novo desenvolvimento de tecnologias computacionais, maior a dependência da TI pelas organizações. Por anos a utilização de sistemas computacionais deixou de ser uma vantagem competitiva para se tornar uma necessidade essencial para a sobrevivência das empresas no mercado.

Como algo inovador e recente, os sistemas de computador, introduzidos no ambiente das organizações, eram utilizados com cada vez mais frequência, mesmo que não da melhor forma possível. A filosofia de apenas adquirir *softwares* e implantá-los a qualquer custo almejando um retorno rápido e recompensador, sem planejamento e nem profissionais especializados e dedicados, é refutada quando os resultados negativos e inexpressivos são divulgados após o término ou cancelamento de algum projeto de implantação de um novo sistema de computador.

Diante de inúmeros casos de fracassos, novas tendências sugeriram no ramo da TI, especificamente no de desenvolvimento de *software*, para reduzir ou eliminar os problemas inerentes a uma tecnologia recente e introduzida nos ambientes corporativos. No escopo desta pesquisa, os problemas concentram-se na integração entre os sistemas, na capacidade de programas de computador interagir com outros, com o objetivo de reutilizar informação, processos, rotinas e lógicas de negócio, transformando um sistema não como um elemento isolado em uma ilha, mas como uma parte de um todo que representa a estrutura de sistemas de uma organização.

Para resolver questões relativas à integração, o mercado experimentou uma das primeiras propostas concretas de arquitetura para solucionar o problema, conhecido como EAI. Após uma década de casos de insucessos, a necessidade do mercado fez surgir a proposta orientada a serviços, conhecida como SOA, utilizada em grande escala principalmente nos últimos anos e que tem correspondido às necessidades dos gestores de TI.

Este trabalho apresenta um estudo de caso de um terminal portuário que encontrou dificuldades na implantação de seus sistemas, comprometendo seriamente suas atividades, e propõe uma arquitetura moderna orientada a serviços para atingir o objetivo de criar a integração homogênea entre as suas aplicações. Os três primeiros capítulos correspondem à base teórica dos conceitos de EAI, ESB e SOA, e finaliza com o estudo de caso mencionado. Por questões éticas o nome do terminal portuário não é mencionado neste trabalho.

2 CONCEITO E MELHORES PRÁTICAS DE EAI

Este capítulo tem por objetivo introduzir o assunto sobre integração de sistemas, que será discutido ao longo dos capítulos 3 e 4, abordando-se, respectivamente, a evolução do conceito de EAI e apresentando-se a definição e melhores práticas de SOA. A introdução proposta pelo presente capítulo fornecerá embasamento teórico para compreender o desenvolvimento de EAI, bem como suas técnicas de integração, permitindo ao leitor realizar comparações com técnicas modernas, discutidas ao longo dos capítulos subsequentes.

Na seção 2.1 é apresentada a definição geral de EAI, contextualizada com o mercado dos anos 2000. Na seção 2.2 é discutida, em forma de tópicos, os tipos existentes de EAI, apresentando integração de dados, de interface da aplicação, de método e interface gráfica com usuário.

2.1 DEFINIÇÃO DE EAI

A EAI (do inglês *Enterprise Application Integration*) pode ser entendida como um conceito relacionado à unificação de soluções de negócio através da combinação de aplicações intermediada por um *middleware*¹. Além de centralizar repositórios de dados e funcionalidades das aplicações, o uso de uma arquitetura EAI viabiliza a integração dos processos das corporações (Ruh, Maginnis, & Brown, 2000, p. 2).

Até meados dos anos 2000, os departamentos das organizações desenvolviam seus próprios processos e aplicações para atender aos seus requisitos de negócio, individualmente, criando o que ficou conhecido como “ilhas de automação”, visto que as soluções não eram compartilhadas com os demais setores das empresas. A falsa impressão de que os objetivos estratégicos das organizações seriam mais bem atingidos tendo-se apenas departamentos informatizados foi notada a partir de pesquisas apuradas em meados de 1998 (Linthicum, 1999, p. 7), constatando-se que a elevada necessidade de aquisição por novos produtos deu-se por conta do suporte a novos requisitos de negócio, que mudavam constantemente em resposta ao mercado e que, portanto, não era suficiente adquirir novas aplicações, mas sim integrá-las de forma a canalizar os processos, dados e regras comuns.

Como forma de integrar os sistemas da organização, a fim de se aproveitar processos e informações, utilizava-se o conceito de integração “*link-a-link*”, consistindo na troca de

¹ Refere-se a uma tecnologia que provê serviços independentes para intermediar outras aplicações (Ruh, Maginnis, & Brown, 2000, p. 3)

dados através de um canal de comunicação construído diretamente entre os sistemas. Apesar de atender, inicialmente, as necessidades de integração dos *softwares*, a gama de tipos de sistemas e da multiplicidade fez com que este modelo de integração se tornasse complexo para se controlar, causando, conforme novos sistemas eram incorporados nas organizações, em uma situação caótica. Sempre que um *software* precisasse se comunicar com outro, um canal era construído entre ambos (ilustrado pela figura 1), fazendo com que, em um ambiente corporativo, contendo centenas de sistemas, o controle da informação se tornasse uma tarefa altamente dificultosa (Linthicum, 1999, pp. 12, 13).

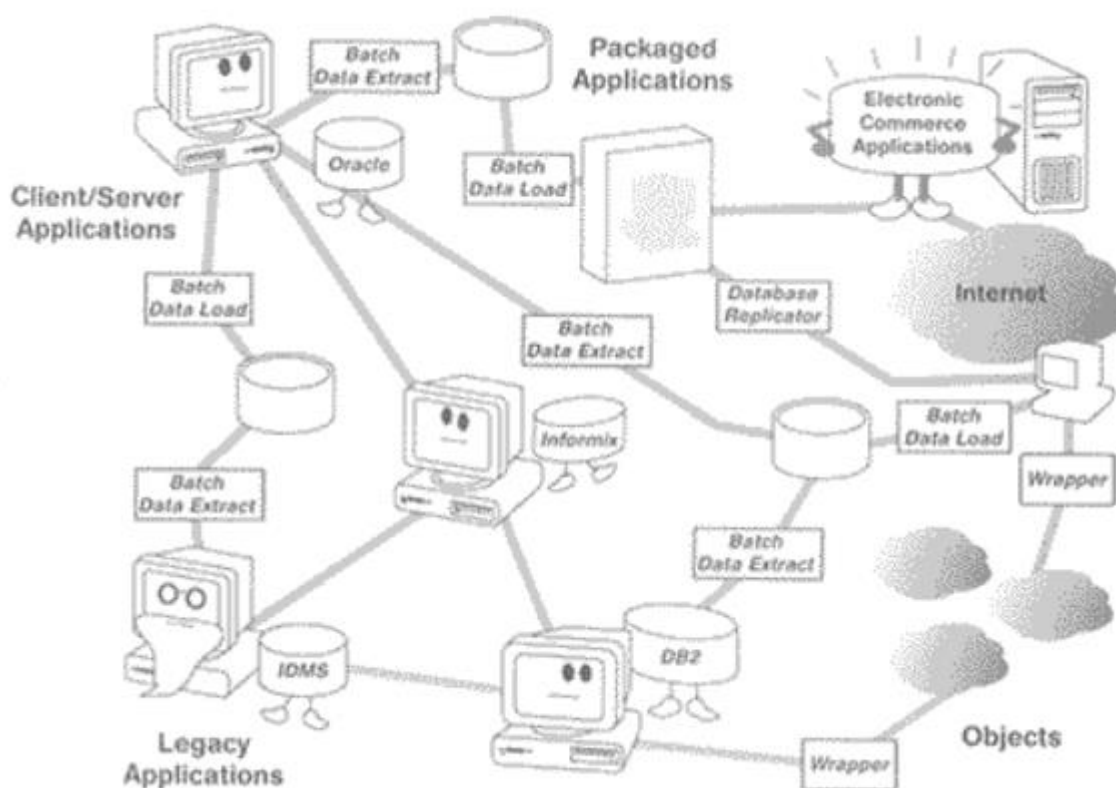


Figura 1 – Caos na integração “link-a-link” de sistemas
 Fonte: Linthicum (1999, p. 13)

Em um ambiente corporativo, as necessidades de negócio fazem com que diferentes tipos de aplicações se comuniquem, como por exemplo, um sistema legado e um ERP (do inglês *Enterprise Resource Planning*), contudo realizar a integração de recursos tão distintos não é uma tarefa trivial. A proposta de uma arquitetura EAI é de propiciar organização (ilustrado na figura 2), controle e segurança em um ambiente empresarial, no âmbito de integração de sistemas, sejam eles legados, distribuídos, de estações de trabalho ou produtos de terceiros, através de *message brokers*, que são responsáveis por realizar a transformação

das mensagens de acordo com as características de cada sistema, tornando a integração independente de aplicação, tecnologia ou protocolo.

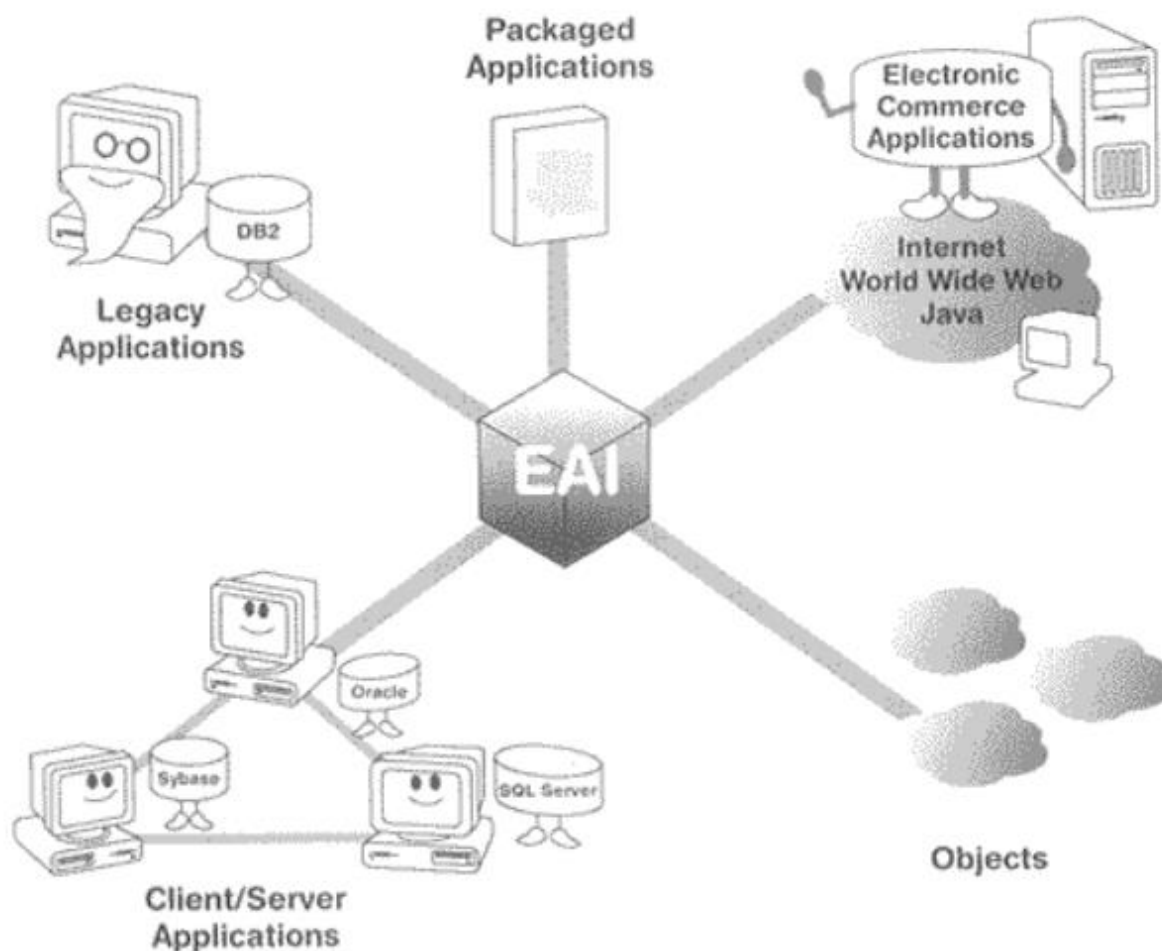


Figura 2 – Message Brokers proposto pela EAI
 Fonte: Linthicum (1999, p. 15)

Além do controle, uma arquitetura EAI fornece uma visão consolidada dos dados e processos da organização, pois seu foco não é simplesmente transmitir dados de uma ponta a outra da estrutura de sistemas, mas garantir fluxos de processos, integridade e segurança da informação e o correto transporte dos dados entre os sistemas.

As principais motivações para o uso de uma arquitetura EAI se resumem a redução de custos, visto que as integrações custavam no período de 2000 a 2001 cerca de 40% do orçamento destinado ao processamento das informações de negócio, e da possibilidade de criar vantagem competitiva a partir do uso mais apropriado de B2B (do inglês *Business to Business*), CRM (do inglês, *Customer Relationship Management*) e da própria Internet (Serain, 2002, p. 1).

2.2 TIPOS DE EAI

Apesar das razões descritas anteriormente, considerar a arquitetura EAI no contexto empresarial deve pautar alguns pontos importantes, pois a depender das necessidades de negócio *versus* custo-benefício, é possível que uma arquitetura mais simples atenda aos objetivos. Primeiramente, é necessário compreender a importância dos dados e dos processos de negócio para a organização, a ponto de verificar como os processos estão relacionados e se há necessidade deles trocarem dados em comum. Depois, em conjunto com o setor de TI, deve-se avaliar o que é e o que não é possível automatizar sob o âmbito de processo. Ambas as tarefas, dependendo do tamanho da organização, demandará grande esforço e tempo para que sejam concluídas, contudo faz-se necessário para que a arquitetura EAI a ser proposta não seja concebida equivocadamente, considerando processos que de fato serão automatizados e que dependem de integração entre si.

Diante de várias perspectivas de integração, que variam conforme as necessidades da organização, uma arquitetura EAI pode ser classificada em quatro níveis: dados, interface com aplicação, método e interface com usuário, conforme descrito nos tópicos seguintes de acordo com (Linthicum, 1999):

2.2.1 Integração de Dados

Neste nível de integração, uma arquitetura EAI dará foco, basicamente, aos dados contidos nas fontes de dados dos sistemas, fornecendo meios para trafegar informação de negócio entre as aplicações do contexto empresarial. A existência de técnicas e ferramentas no mercado para replicação e transformação de dados entre bancos de dados concede a esta abordagem de integração uma importante vantagem, poupando, em alguns casos, esforço para a construção de componentes de software para realizar esta tarefa.

Na maioria dos casos, uma integração de dados não afeta a lógica da aplicação detentora da informação, tampouco a estrutura do banco, reduzindo os impactos da implantação de uma arquitetura EAI sob esta perspectiva, em comparação com outros níveis de integração de aplicações abordados posteriormente. Uma situação apropriada para esta abordagem é em relação à necessidade de integrar um ERP, tradicionalmente uma solução privada e adquirida pela organização, com uma aplicação qualquer; por não ter acesso ao código-fonte do ERP e por conta dos custos de se realizar alterações no produto, uma

integração neste nível permite atingir o objetivo da organização sem interferir na estrutura nas camadas de aplicação ou de dados do ERP.

Normalmente uma arquitetura EAI com abordagem de integração de dados envolverá um *middleware* responsável por transferir os dados de uma origem a um destino, a partir de alguma notificação na origem ou a partir de um intervalo de tempo. Essa transferência pode consistir em uma transformação dos dados, previamente, antes que a informação seja entregue ao seu destino; embora em muitos casos atualmente os bancos de dados sejam relacionais, é possível que a tecnologia empregada na origem e no destino seja distinta, como por exemplo, Adabas na origem e banco relacional no destino (em casos envolvendo sistemas legados ou *mainframe*), ou banco relacional na origem e banco multidimensional no destino (em casos envolvendo sistemas que utilizam a tecnologia OLAP). Nesta abordagem, uma prática comum no tráfego de dados entre origem e destino é do emprego de XML para fins de estruturação e transformação.

2.2.2 Integração de Aplicações

Integração de aplicações tem por objetivo aproveitar a camada de negócio contida em uma aplicação, diferentemente da abordagem anterior, onde o foco é puramente o dado. Através do acesso direto a uma fonte de dados, seja um arquivo ou um DBMS, muito provavelmente as regras de negócio serão ignoradas, por normalmente estarem contidas no nível da aplicação. Nos casos em que a integração de sistemas tem por objetivo aproveitar os processos de negócio, a abordagem de dados deixa de ser a melhor a solução, dando margem a outras abordagens, como a que atua no nível de interface da aplicação.

O desenvolvimento de aplicações a partir dos anos 2000 passou a considerar os sistemas como sendo partes integrantes de um ambiente corporativo, e não meramente como aplicações isoladas que devem atender a um determinado setor da empresa sem necessidade de comunicação com outros *softwares*. A partir desta ideologia, padrões de mercado surgiram como uma forma de organizar as interfaces de comunicação dos sistemas, como por exemplo, Java RMI, CORBA, IIOP e Microsoft DCOM, e hoje se tornaram boas práticas para a criação de APIs, em contraste com padrões privados.

A forma de integração de interface da aplicação envolve a disponibilização de uma API, que basicamente atua como uma camada de acesso aos dados e às regras de negócio. Normalmente produtos particulares, como ERPs, apresentam APIs escritas de acordo com um padrão próprio, dificultando o seu entendimento e o processo de integração, diferentemente de

APIs escritas de acordo com padrões do mercado. As APIs normalmente são escritas em Java, C, C++ e .NET, e a figura 3 apresenta um exemplo de código escrito em Java consumindo uma função fornecida por uma API de um ERP, que basicamente retorna os dados do cliente através de um identificador exclusivo.

```

1 public void getCustomerDetailsFromERP(long customerId) throws ERPCommunicationException {
2     String customerDetailsXML = getCustomerDetails(customerId);
3     System.out.println(customerDetailsXML);
4 }

```

Figura 3 – Exemplo de execução de um método de uma API

A figura 4 exemplifica o retorno da função, no formato XML, após o método ter sido executado com sucesso.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <customer>
3     <id>921</id>
4     <name>James Smith</name>
5     <birthday>1970-04-12</birthday>
6     <address>831 Green Avenue</address>
7     <city>Atlanta</city>
8     <zipcode>22030</zipcode>
9 </customer>

```

Figura 4 – Exemplo de resposta de um método de uma API

Soluções fechadas, como SAP, Siebel e PeopleSoft, normalmente tornam-se grandes desafios de integração para um arquiteto EAI, visto que são produtos desenvolvidos por terceiros, que podem deixar de oferecer suporte a seus *softwares*, e o acesso aos dados e processos de negócio é condicionado pela limitação da API fornecida.

A integração de aplicações pode envolver acesso a dados, processos de negócio e objetos, descritos a seguir:

Dados: Resumidamente, diz respeito ao acesso direto aos dados da aplicação, ignorando regras de negócio que podem estar contidas em uma funcionalidade do *software*;

Processos de Negócios: Diz respeito ao acesso às rotinas de negócio da aplicação, contidas nas funcionalidades do produto, como por exemplo, uma validação de cadastro de fornecedor;

Objetos: Oriundo do conceito do paradigma de orientação a objetos, os objetos são recursos que combinam os dois elementos citados anteriormente. Os dados, representados pelos atributos, serão acessados e modificados através dos processos de negócios, representados pelas operações (ou métodos, como também são conhecidos) do objeto.

A API de um *software*, portanto, pode expor métodos que atuem no nível de dados, negócio ou objetos. Além disso, os produtos terceirizados, assim como as soluções caseiras, tendem a seguir uma das seguintes arquiteturas: centralizada, de duas camadas ou de três de camadas.

2.2.2.1 Arquitetura Centralizada

Nas soluções tradicionais, os dados (assim como a lógica de acesso) e a interface gráfica do usuário eram mantidos em uma mesma máquina (normalmente um *mainframe*), sendo acessados por terminais burros, conforme ilustrado pela imagem 5. Neste cenário, a manutenção é mais facilitada do que em sistemas distribuídos, por tudo residir em um mesmo computador; outra vantagem é em relação à integração, visto que tanto processo de negócio quanto dados podem ser acessados em conjunto, por um único ponto.

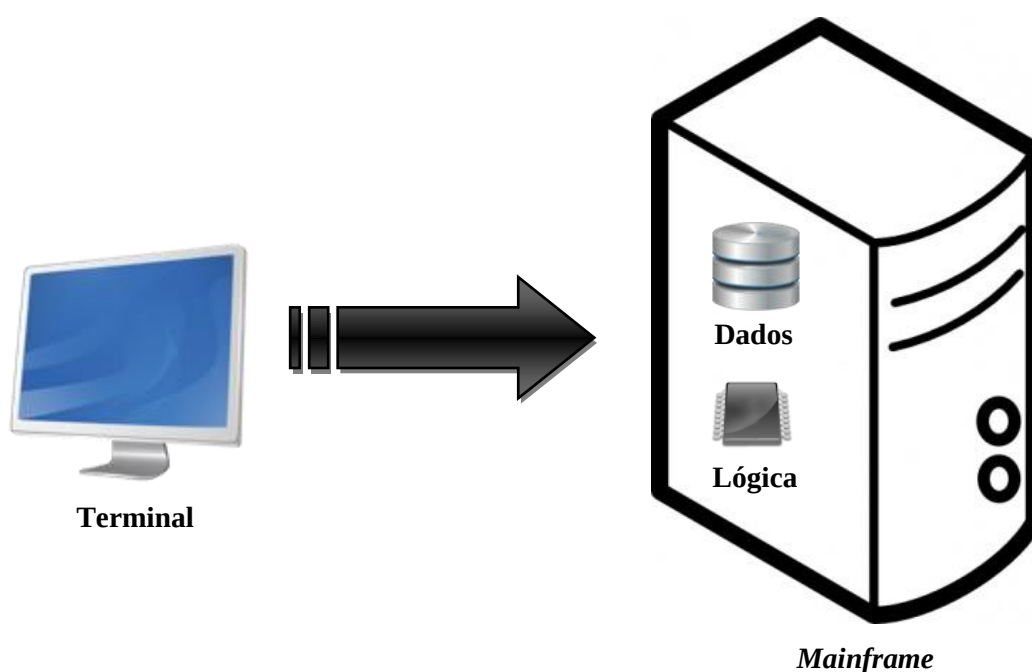


Figura 5 – Arquitetura centralizada
Fonte: Linthicum (1999, p. 43). Adaptado pelo autor

Em resposta às necessidades do mercado, o modelo distribuído de *software* surgiu. Embora não possua as vantagens da arquitetura centralizada, neste novo modelo é possível

combinar diferentes sistemas operacionais, servidores, DBMSs e outras aplicações, possibilitando às empresas diversificar sua estrutura de TI e torná-la adaptável a soluções mais sofisticadas.

2.2.2.2 Arquitetura de Duas Camadas

Neste modelo, a arquitetura é dividida em duas camadas, onde a interface gráfica com usuário, ou GUI (*Graphical User Interface*), é separada dos dados. A lógica de negócio reside também na mesma camada da GUI, conforme ilustrado na imagem 6. Nesta arquitetura, também conhecida como cliente/servidor, o lado cliente torna-se “pesado” por processar a lógica de negócio. Uma forma de reduzir a responsabilidade do lado cliente é através da criação de uma camada intermediária, proposta na arquitetura a ser discutida a seguir.

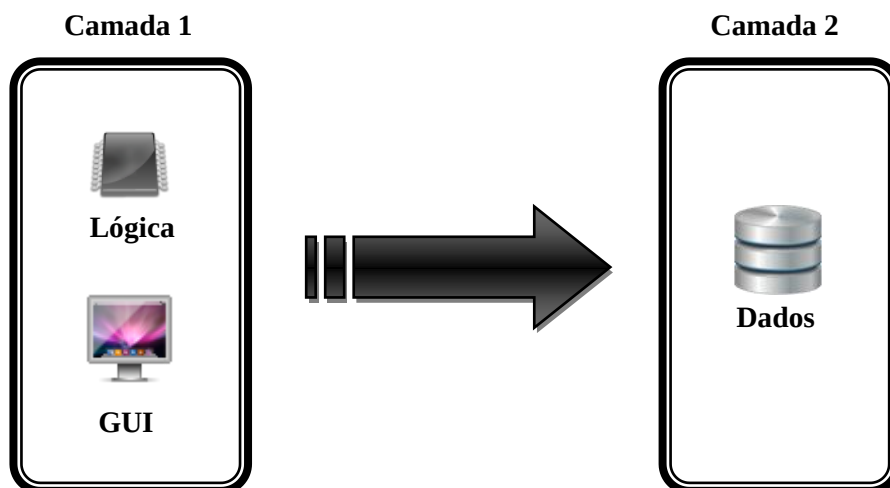


Figura 6 – Arquitetura de duas camadas
Fonte: Linthicum (1999, p. 44). Adaptado pelo autor

2.2.2.3 Arquitetura de Três Camadas

Similar ao modelo anterior, ao criar camadas, nesta arquitetura há criação de uma camada intermediária, responsável por processar a lógica de negócio do *software*. Neste modelo, conforme ilustrado na imagem 7, é introduzido um servidor de aplicação intermediário para representar a camada de regras de negócio. Nesta arquitetura, cada camada tem sua própria responsabilidade, e a interação do cliente com os dados é sempre intermediada pelo servidor da aplicação.

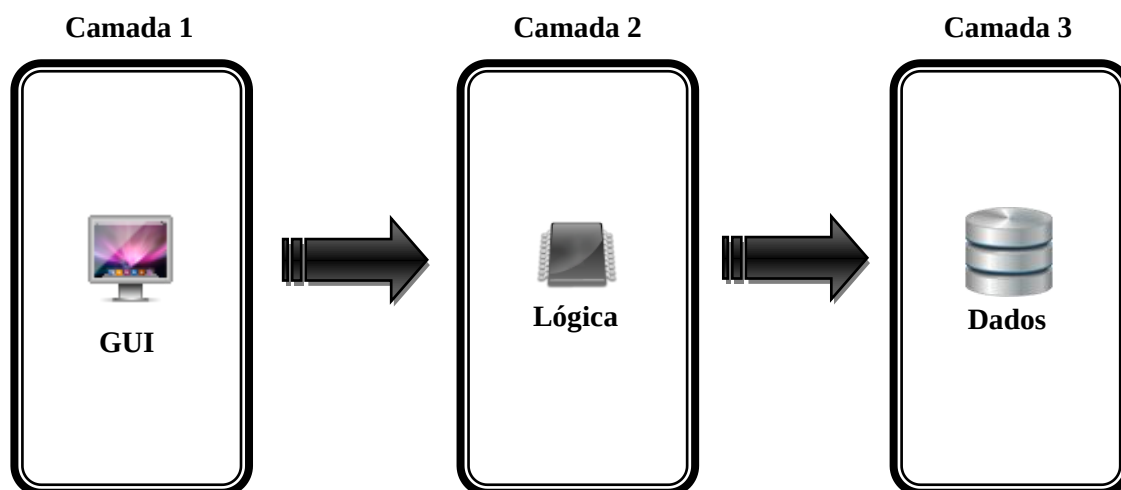


Figura 7 – Arquitetura de três camadas
Fonte: Linthicum (1999, p. 44). Adaptado pelo autor

Além de ser uma tendência entre os sistemas mais modernos, esta arquitetura provê uma limpa separação e organização entre as camadas, além de escalabilidade, característica fundamental em estruturas empresárias. No modelo de duas camadas, a conexão com o banco de dados é realizada de forma direta, fazendo com que cada cliente tenha como dependência uma conexão física com o DMBS, enquanto que no modelo distribuído em três camadas a responsabilidade de acesso aos dados passa a ser do servidor de aplicação, possibilitando a criação de poucas conexões para suportar diversos clientes através de um recurso chamado *pooling* de conexões.

2.2.3 Integração de Métodos

Neste nível de integração, um arquiteto EAI tem como objetivo criar uma infraestrutura para que as aplicações da organização compartilhem de mesmos métodos ou processos de negócio, que sejam comuns entre si. Neste modelo, o foco de integração é o reaproveitamento de rotinas pelas aplicações que constituem a estrutura de sistemas, de tal forma a evitar duplicidade de código e consequentes complicações na manutenção dos *softwares*.

Uma sugestão de arquitetura utilizando este modelo de integração é concentrando em um local centralizado todas as rotinas que sejam comuns às aplicações, servindo como um repositório de métodos, de tal forma que se um programa precisar de determinada funcionalidade bastará consumir a rotina disponível por esse repositório. Em termos práticos, esta arquitetura pode envolver um servidor de aplicação contendo os métodos disponíveis em

forma de objetos distribuídos, que poderiam ser consumidos via tecnologias como CORBA ou DCOM.

Dentre os desafios de implementação de uma arquitetura EAI utilizando este modelo de integração, o maior deles é relativo ao custo em submeter aplicações existentes a alterações em sua estrutura de código para passar a consumir métodos externos. Quanto mais sistemas existentes na organização, maior será o custo; e justificar esse investimento perante a alta direção aumenta o desafio de uso da integração no nível de método. Em contrapartida, se a arquitetura de sistemas da empresa utilizar este modelo desde a sua concepção, os benefícios tendem a ser altamente relevantes, visto que o impacto de alterações nos processos de negócio será menor e afetará automaticamente todos os sistemas integrados.

A integração entre aplicações no nível de método pode ser categorizada da seguinte forma:

Regras: são conjuntos de condições que devem ser atendidas para que os processos de negócio sejam realizados na organização; as regras podem envolver requisitos da própria empresa, ou requisitos externos, como normas do governo ou leis. Os sistemas implementam as regras em sua própria estrutura para o controle do fluxo e validação da informação. Em uma arquitetura EAI, regras comuns seriam disponibilizadas em um servidor para múltiplos acessos das aplicações, com uma infraestrutura que fornecesse processamento centralizado; dessa forma, a mudança de uma regra, como uma nova política da empresa, seria realizada em um único ponto, em vez de em todos os sistemas que implementassem esta restrição.

Lógica: lógicas são meramente blocos de instruções dentro de um programa, como por exemplo, blocos sequenciais, condicionais e de repetição, enquanto que regras são mais amplas, por definirem restrições de negócio. Centralizar lógicas pode trazer benefícios relativos a desempenho, visto que é possível ter blocos de código que realizem a mesma tarefa, todavia de forma mais ou menos eficiente.

Dados: neste contexto, dados são as informações trafegadas entre os sistemas e usuários ou computadores. Todos os sistemas compartilham informações, desde relatórios, passando por sistemas de contabilidade, até a ERPs. Neste caso, os dados seriam acessados via métodos e, portanto, o seu nível de acesso dependeria de quanto que os métodos iriam expor as informações.

Objetos: oriundo do paradigma da orientação a objetos, os objetos, como mencionados anteriormente, são elementos de computação que encapsulam dados que serão manipulados e acessados via operações, conhecidas como métodos. Identificar objetos em âmbito organizacional pode ser uma tarefa mais complexa do que identificar processos de

negócio isolados, exigindo, portanto, maior tempo de estudo para construir um modelo de objetos da organização.

2.2.3.1 Frameworks

Importantíssimo para a integração de métodos, um *framework* pode servir como uma coleção de rotinas e estruturas prontas para resolver diversos problemas comuns aos sistemas, e sob o âmbito de uma arquitetura EAI o seu papel pode ser ainda mais importante. Para conceito, Freedman (1993 *apud* Linthicum, 1999, p. 63) define que *frameworks* “consistem em classes abstratas e a colaboração de seus objetos bem como suas classes concretas. Enquanto que o paradigma da orientação a objetos apoia o reuso de *software*, *frameworks* apoiam o reuso de *design*”.

Para EAI, *frameworks* podem ser utilizados para reunir métodos que processem regras do negócio, concentrando estruturas já validadas e testadas no passado e que podem ser reutilizadas pelos sistemas, reduzindo esforço e tempo na realização do *design* da aplicação. Desta forma, as aplicações podem reutilizar não só rotinas de uso geral – como manipulação de arquivos, que é comum a vários sistemas, mas sem valor de negócio –, mas principalmente fluxos e estruturas de controle da informação baseadas em processos da organização; uma lógica para aprovação de crédito de um cliente, por exemplo, pode ser compreendida como um processo encapsulado por um método disponível em um *framework*, que por sua vez pode ser reutilizado por vários sistemas.

A codificação de *frameworks* voltados para uma arquitetura EAI requer esforço de muita análise dos processos de negócio da empresa, entender as regras e restrições envolvidas, elencar aqueles que serão implementados em código e que podem ser compartilhados entre os sistemas. Todo este procedimento é custoso, visto que uma grande organização pode concentrar até centenas de regras de negócio, e requer elevado conhecimento de *design* para que o *framework* a ser construído possa de fato ser utilizável. Outra característica a ser considerada é relativa à tecnologia empregada na construção do *framework*, visto que sistemas codificados em tecnologia diferente não poderão utilizá-lo – por exemplo, um *e-commerce* implementado em Java não poderá utilizar um *framework* baseado em Delphi. Por fim, para que o *framework* esteja pronto para ser incorporado no desenvolvimento dos sistemas, o seu processo de desenho da arquitetura e design, implementação e testes será custoso; *frameworks* mal projetados, escritos ou testados tendem a ser problemáticos para os sistemas.

2.2.4 Integração de Interfaces Gráficas de Usuário

Apesar de menos estruturada do que os níveis anteriores estudados, a integração de interfaces gráficas do usuário é uma opção a ser considerada por um arquiteto EAI, principalmente em situações em que o uso das demais técnicas de integração, por alguma razão, não é recomendada – como uma manutenção evolutiva que seja arriscada realizar em um sistema legado que não fornece integração no nível de método ou aplicação – ou não é possível – como em um sistema externo de código-fonte privado que não fornece mecanismos de integração.

A integração neste nível orienta-se puramente pela interface gráfica com o usuário, ou seja, com as telas do sistema. Neste cenário, a integração irá capturar (ou submeter) dados para as telas simulando um usuário humano, e realizar a tratativa de mensagens que serão respondidas pelo sistema. A integração neste nível deve considerar o fato de que as telas podem gerar dados que não necessariamente possuem origem no banco de dados (ou outro repositório utilizado pelo *software*), mas sim dados que podem ser gerados ou calculados a partir da camada de lógica da aplicação.

Como primeiro passo, o arquiteto EAI deverá compreender o funcionamento das telas do sistema. Neste momento, a documentação lhe será útil e poupará tempo em verificar quais informações são derivadas da camada de lógica e como as telas se relacionam; contudo, quando a documentação é inexistente ou obsoleta, o caminho será analisar o código-fonte ou dialogar com desenvolvedores do sistema. Outro ponto importante a ser observado é em relação ao lançamento de erros, que requer atenção para que a integração possa identificar quando e em quais condições uma falha é emitida pelo sistema. Este primeiro passo, se negligenciado, poderá influenciar no sucesso da integração, acarretando em possíveis falhas relativas à transmissão de dados incorretos ou de omissão de erros produzidos pelo sistema a ser integrado.

Em termos práticos, a integração de interfaces gráficas de usuário funcionará a partir da extração de dados da tela, que pode ser categorizada por duas formas distintas de acordo com (Linthicum, 1999, pp. 77, 78), listadas a seguir:

Extração Estática: Consiste em extrair os dados a partir de posições fixas na tela, orientando-se por coordenadas x e y do plano cartesiano. A vantagem desta técnica é em relação à facilidade de implementação, contudo ela é recomendada apenas nos casos em que as telas não possuem frequência de alteração, pois qualquer alteração de posição irá impactar na extração, resultando em dados errôneos para as aplicações integradas.

Extração Dinâmica: Neste cenário, a extração dos dados será orientada por identificadores dos campos ou dos rótulos contidos na tela. Desta forma, mesmo que um campo seja remanejado ou que seu tamanho seja reajustado, a integração continuará funcionando; é possível, inclusive, realizar validações na tela, como identificar ausência de informações obrigatórias. A desvantagem desta técnica é em relação à implementação, que tende a ser mais complexa do que a extração estática.

Há duas abordagens de como as telas podem ser tratadas durante a integração, descritas a seguir:

Tela como Dado: Ao utilizar esta abordagem, a tela por completo é processada como um texto, armazenada em uma variável *String*. Desta forma, através da análise da variável, a aplicação responsável pela integração deverá extrair os dados necessários através de rotinas que processem texto. A vantagem desta abordagem é a simplicidade de implementação, que dispensa preocupações em criar estruturas de dados mais sofisticadas.

Tela como Objeto: Aqui, a aplicação responsável pela integração representará a tela como um objeto, sendo um objeto Java, CORBA ou similar, fornecendo operações para acessar e modificar os dados da interface gráfica. Embora mais complexo, essa implementação apresenta excelente vantagem, por oferecer a possibilidade de se distribuir os objetos em um servidor de aplicação ou *message broker*, que por sua vez podem ser mais facilmente integrados às outras aplicações da arquitetura empresarial de sistemas.

3 BARRAMENTO SOA E EVOLUÇÃO DA EAI

O objetivo deste capítulo é o de apresentar mais conceitos relacionados à integração de sistemas, com foco para a evolução das técnicas utilizadas na EAI, e contextualizar o leitor com conceitos elementares do barramento SOA. O atual capítulo fornecerá informações para compreender a evolução da EAI, descrevendo alguns paradigmas e topologias utilizados para integração de sistemas.

Na seção 3.1 é descrito critérios para se construir uma integração entre aplicações, e como eles influenciarão o(s) estilo(s) empregado(s) pelo arquiteto EAI. Na seção 3.2 é descrito os estilos possíveis de integração EAI, como transferência de arquivos, banco de dados compartilhado, execução remota de procedimentos e *middleware* orientado a mensagens. Por fim, na seção 3.3 é explicado as diferentes topologias que podem ser utilizadas na integração entre aplicações.

3.1 CRITÉRIOS PARA INTEGRAÇÃO DE APLICAÇÕES

Optar por realizar integração entre aplicações, conforme brevemente discutido no capítulo anterior, não se trata de uma tarefa trivial e deve, portanto, envolver análise e cautela ao decidir se uma integração será realizada e, posteriormente, como ela será viabilizada. O primeiro critério a ser considerado é se a aplicação de fato deve ou não ser integrada com os demais sistemas da organização; o *software* a ser construído pode ser exclusivo para atender uma necessidade muito específica, isolada do contexto de sistemas da organização, e nestes casos, portanto, não há necessidade de se planejar uma estrutura para integração. O desafio, contudo, é justamente conseguir identificar se a aplicação não demandará comunicação com os demais sistemas, seja de imediato, a curto, médio ou longo prazo. Os tópicos seguintes descrevem outros critérios que devem ser considerados durante a decisão de se construir uma estrutura para integração. Segundo (Hohpe & Woolf, 2003):

3.1.1 Acoplamento entre Aplicações

As aplicações, mesmo integradas, devem reduzir o máximo possível sua dependência com outros sistemas. O alto acoplamento entre aplicações é prejudicial para a capacidade de manutenção, pois ela aumenta a possibilidade de uma aplicação deixar de funcionar ou de

realizar suas tarefas corretamente em decorrência de alterações no sistema integrado. Quanto menor a dependência, melhor para os sistemas no que diz respeito a futuras manutenções.

3.1.2 Simplicidade de Integração

Ao realizar a integração entre os sistemas, o arquiteto em conjunto com os desenvolvedores deve realizar o mínimo de alterações necessárias na aplicação e escrever o mínimo de código possível para viabilizar a integração, contudo é importante atentar à abordagem de integração utilizada, pois às vezes o estilo mais simples não necessariamente trará os melhores resultados para a organização. Portanto, deve-se equilibrar a simplicidade da integração *versus* os retornos esperados.

3.1.3 Tecnologias de Integração

O projeto de integração das aplicações poderá, naturalmente, envolver tecnologias específicas para atender as necessidades da organização, tanto como *software* quanto *hardware*. Este critério deverá ser verificado, visto que a equipe de desenvolvedores, nestes casos, deverá aprender a utilizar as tecnologias utilizadas no processo de integração, aumentando naturalmente a complexidade e o tempo do projeto.

3.1.4 Formato de Dados

Este importante critério diz respeito ao formato de dados utilizado por diferentes aplicações, como por exemplo, a codificação de dados que pode ser UTF-8 na origem e ASCII no destino. Para que a integração ocorra com sucesso, por vezes será necessário traduzir o dado a ser enviado pela aplicação de origem para que a aplicação de destino possa realizar o processamento. Se a integração envolver múltiplas aplicações, o uso de um *middleware* intermediário responsável por essa função pode ser compreendido como uma opção adequada.

3.1.5 Disponibilidade dos Dados

Ao realizar o projeto de integração entre as aplicações, o critério de disponibilidade dos dados deve ser considerado. Um requisito que pode ser priorizado pelas aplicações é o de

possuir acesso o mais rápido possível à informação gerada pelos demais sistemas e, portanto, a latência para que o dado esteja disponível para consumo deve ser a menor possível. Disponibilizar um grande conjunto de informação em pequenas partes pode ser uma abordagem a ser considerada para que as aplicações possam acessar as informações com maior agilidade.

3.1.6 Dados ou Funcionalidades

Aplicações integradas podem desejar compartilhar não apenas os seus dados, mas também suas funcionalidades, a fim de prover métodos que executem total ou parcialmente as regras de um processo de negócio, conforme comentado no capítulo anterior. Diante disto, cabe analisar qual o tipo de arquitetura EAI a ser implementada de acordo com a necessidade de integração da aplicação. As técnicas para alcançar o objetivo de compartilhamento de dados ou funcionalidade serão abordadas adiante.

3.1.7 Assincronia

De forma geral, o processamento das tarefas computacionais é realizado de forma síncrona, ou seja, uma etapa por vez é processada pelo computador. Contudo, pode existir a necessidade de determinada tarefa ser executada em segundo plano, de forma assíncrona, principalmente se o processamento a ser realizado possuir longa duração. No cenário da EAI, uma aplicação integrada pode não aguardar pela conclusão da execução de um método, principalmente nos casos de chamadas remotas de procedimento, para que a rede de computadores não fique sobrecarregada. Este critério, portanto, assume relevante importância para o arquiteto e a equipe de desenvolvedores.

3.2 PARADIGMAS DE INTEGRAÇÃO

Um ou mais critérios discutidos anteriormente podem ser atendidos por uma determinada abordagem de integração. Os estilos de integração podem ser resumidos em quatro grandes principais, segundo (Fowler, Rice, Foemmel, Hieatt, & Mee, 2002), analisados nos tópicos seguintes. É importante salientar que não existe o estilo perfeito, aquele que sempre deve ser utilizado para solucionar os problemas de integração, pois cada problema tem

as suas próprias particularidades; cabe ao arquiteto escolher o estilo mais apropriado de acordo com o problema a ser resolvido.

3.2.1 Transferência de Arquivos

Neste cenário de integração, os dados são compartilhados através da geração e consumo de arquivos. Cada aplicação gera um ou mais arquivos contendo as informações que outras aplicações precisam, dentro de uma periodicidade de tempo determinada por aqueles que estiverem projetando a integração entre os sistemas. A imagem 8 ilustra um esquema simples desta abordagem.

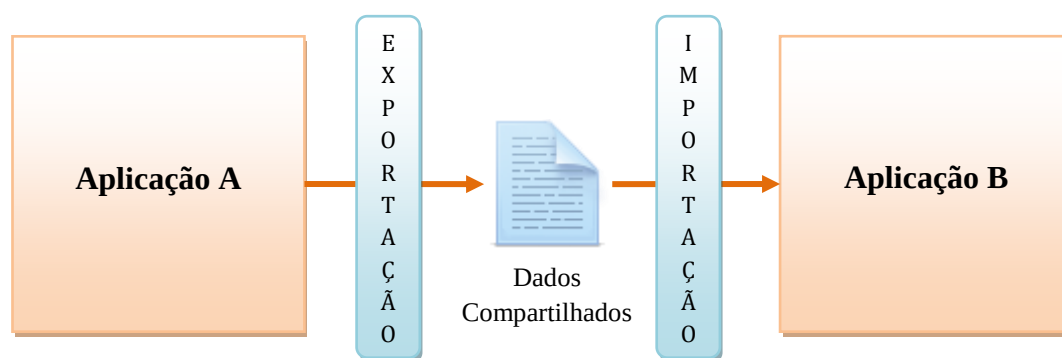


Figura 8 – Transferência de arquivos
Fonte: Hohpe & Woolf (2003, p. 66). Adaptado pelo autor

A transferência de arquivos promove uma característica importante para a integração de sistemas no que diz respeito ao baixo acoplamento. Manutenções na aplicação que provê o arquivo, contanto que não influenciem na geração do mesmo, não trará impacto nas aplicações integradas, pois o único fator de dependência é próprio arquivo, responsável por estabelecer a comunicação entre os sistemas.

Neste estilo de integração, o arquiteto deverá se preocupar em estabelecer um formato de arquivo a ser utilizado pelos sistemas. Em aplicações de *mainframe* é comum definir arquivos no padrão COBOL, já em aplicações Unix os arquivos de texto normalmente são os mais utilizados e em sistemas mais modernos tem-se adotado o padrão XML. Consequentemente, há diversas ferramentas para realizar operações de leitura, gravação e transformação entre formatos de arquivos, tornando a implementação deste estilo menos problemática.

Uma preocupação com a transferência de arquivos é em relação à frequente atualização da informação. Enquanto o sistema integrado não realizar a leitura do arquivo, a informação ficará temporariamente desatualizada. Há ambientes corporativos em que a atualização da informação é crucial para os processos de negócio, como por exemplo, um sistema de cobranças que utiliza dados do cliente, oriundos de um CRM, para notificar cobranças através de informações de endereço e contato; se estes dados estiverem desatualizados, é possível que o sistema envie a cobrança erroneamente.

Por fim, neste estilo de integração, caberá ao arquiteto projetar uma estrutura para controlar quando um arquivo já foi processado e não será mais necessário (e consequentemente, como ele será descartado), quais aplicações já efetuaram o consumo dos dados e controlar o acesso e gravação de arquivos. Dependendo da frequência de atualização das informações, este estilo de integração não será o mais indicado e, portanto, a alternativa de banco de dados compartilhado ou *middleware* orientado a mensagens poderá atender com mais eficácia o objetivo da integração.

3.2.2 Banco de Dados Compartilhado

Compartilhar banco de dados entre diversas aplicações consiste em criar entidades que serão utilizadas pelos sistemas integrados, realizando consultas, gravações, atualizações e exclusões de dados e, com isso, atingindo a integração neste nível. Para implementar esta abordagem de integração, basta disponibilizar um banco de dados central que será acessado pelas aplicações, conforme ilustrado pela imagem 9.

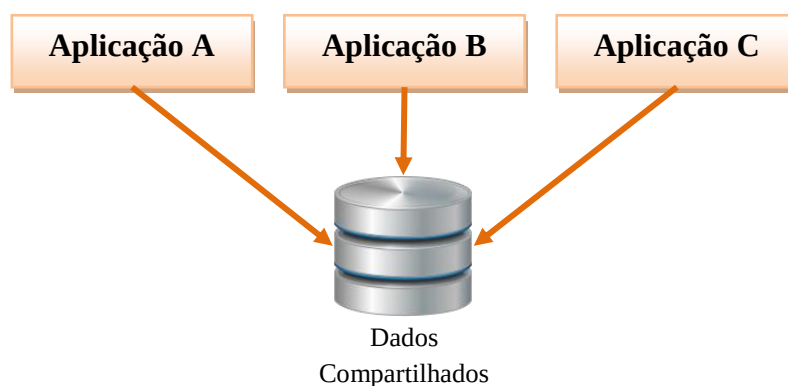


Figura 9 – Banco de dados compartilhado

Fonte: Hohpe & Woolf (2003, p. 69). Adaptado pelo autor

Enquanto que a consistência da informação utilizando transferência de arquivos era dificultosa de se atingir, com o uso de um banco de dados compartilhado entre as aplicações a informação estará consistente e disponível o tempo todo. Tanto criação quanto a modificação dos dados é controlada por gerenciadores de transação, garantindo que todas as aplicações possam submeter alterações nos dados de forma íntegra. A disponibilidade das alterações também é uma característica a se considerar, visto que a partir do momento que uma transação é concluída, a informação passa a estar disponível para consulta, sendo este processo realizado em intervalo de tempo consideravelmente menor do que com a transferência de arquivos.

Outra preocupação que o arquiteto EAI tem ao utilizar o estilo de integração citado anteriormente é em relação ao formato de arquivos a ser utilizado e padronizado, garantindo que os sistemas envolvidos se comuniquem no mesmo “idioma”. Ao utilizar um banco de dados relacional, todos os principais DBMSs do mercado suportam a linguagem SQL, garantindo assim uma forma padronizada de comunicação entre as aplicações e o repositório de dados central.

Apesar das vantagens, modelar um banco de dados que tenha uma estrutura comum a todos os sistemas integrados é uma tarefa delicada de se concretizar, em virtude das necessidades de cada aplicação. Sistemas diferentes demandam informações diferentes, e consolidar tabelas em um único esquema de objetos para armazenar os dados de tal forma que atenda as necessidades de persistência das aplicações requer cuidado para que não sejam criadas tabelas redundantes e, conseqüentemente, difíceis de serem utilizadas.

O número de aplicações a serem integradas, bem como a quantidade de conexões com o banco de dados que serão criadas com o intuito de consultar e modificar os dados, requer atenção por parte da equipe a implantar este modelo de integração. Acessos simultâneos em um ambiente mal configurado, ou acessos que submetam comandos SQL erroneamente estruturados, podem causar problemas de desempenho no banco de dados, assim como longos *locks*² nas operações de leitura e gravação em tabelas, prejudicando consideravelmente a integração entre os sistemas e a disponibilidade da informação.

² Trata-se de um mecanismo para bloquear temporariamente a operação em um dado que seja conflitosa com outra operação, em situações envolvendo duas ou mais transações simultaneamente (Bernstein & Newcomer, 2009, p. 141)

3.2.3 Execução Remota de Procedimento

Tanto o estilo conceituado na transferência de arquivos como no compartilhamento de um banco de dados, o foco da integração é em compartilhar dados entre as aplicações. No entanto, nem sempre as necessidades das aplicações envolvem apenas reaproveitar informações contidas em outros sistemas. A depender das regras de negócio da organização, a atualização do endereço de um cliente, por exemplo, pode consistir em uma simples modificação do dado, ou envolver um complexo processo consistindo em etapas jurídicas antes de concretizar a modificação da informação. E se diferentes aplicações compartilharem a mesma funcionalidade, como a do exemplo citado, então convém reaproveitar a função contida em um sistema para que não só os sistemas respeitem o processo de negócio, mas também para reduzir esforço de desenvolvimento, através da integração por RPC (*Remote Procedure Call*).

Um importante e poderoso conceito relacionado a *design* de aplicações é o encapsulamento, permitindo modificar e ler um dado através de uma interface de chamada de função. Através de funções é possível interceptar a alteração de um dado, possibilitando a uma aplicação iniciar as ações necessárias para responder à alteração. Banco de dados compartilhado oferece uma grande estrutura de dados não encapsulada, tornando difícil realizar esta tarefa e a transferência de arquivos, embora permita a uma aplicação tomar conhecimento de quando um dado foi alterado, fornece atraso neste processo, pois depende da criação do arquivo e da leitura do mesmo por parte da aplicação integrada.

Em linhas gerais, a implementação deste estilo de integração envolve basicamente uma aplicação que fará a chamada de uma função em outra aplicação, detentora do processo de negócio a ser compartilhado. O programa enviará os dados necessários, de acordo com a interface estabelecida na comunicação, e aguardará a resposta após o término do processamento da função. A imagem 10 ilustra graficamente este estilo de integração.

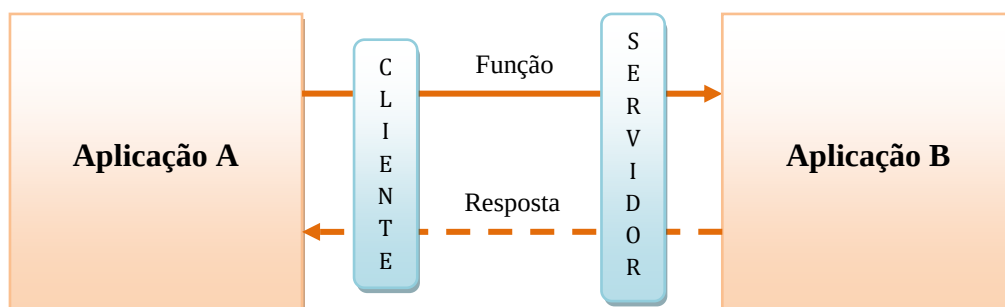


Figura 10 – *Remote Procedure Call*

Fonte: Hohpe & Woolf (2003, p. 71). Adaptado pelo autor

A arquitetura RPC envolve um contrato que delimita a interface fornecida pelo servidor e estabelece os serviços disponíveis e os parâmetros necessários para o cliente consumir. Segundo (Serain, 2002, p. 55), o contrato que estabelece as diretrizes para a comunicação entre as aplicações possui as seguintes características:

- Ele é identificado por um número identificador único conhecido como *uuid* (*universal unique identifier*). O *uuid* é utilizado em todas as requisições do cliente para que ambas as partes saibam que estão se comunicando sobre a mesma coisa;
- Ele define e nomeia a interface fornecida pelo servidor ao cliente. A interface contém os serviços que o servidor pode fornecer e contém a assinatura dos serviços desejados pelo cliente;
- Ele é escrito em uma linguagem de informação especial chamada IDL (*Interface Definition Language*). Há algumas versões desta linguagem e uma foi normatizada para a tecnologia RPC. Ela é chamada de OSF (*Open Software Foundation*) IDL RPL;
- Tanto cliente quanto servidor recebe uma cópia do contrato. Uma cópia representa a ponta do cliente e a outra a ponta do servidor.

O modelo RPC aplica o conceito de encapsulamento na integração entre os sistemas. Se uma aplicação requisitar acesso ou modificação de uma informação, deverá realizar esta ação por intermédio da chamada de uma função fornecida pelo servidor. Existem diversas abordagens RPC no mercado, como CORBA, COM, .NET Remoting, Java RMI e outros; a escolha da abordagem dependerá dos sistemas a serem integrados. Estes ambientes fornecem algumas capacidades adicionais, como controle transacional das chamadas. Outra abordagem, mais moderna e independente tecnologicamente, é o uso de *Web Services* através de padrões como SOAP e XML, que funcionam através do protocolo HTTP (esta abordagem será discutida com mais ênfase no capítulo seguinte).

Ao optar em utilizar RPC para realizar a integração entre sistemas é necessário se ater a um importante detalhe. Ao se distribuir os serviços em um servidor para que sejam consumidos pelos clientes, deve-se ter a ciência de que as chamadas serão realizadas remotamente, e esta característica, se negligenciada, poderá trazer problemas de desempenho

na integração ao utilizar demasiadamente este recurso ou se arquitetura não for corretamente projetada.

Embora o acoplamento entre as aplicações diminua ao utilizar o conceito de encapsulamento e eliminar a dependência de estruturas de dados compartilhados (como observado no estilo de banco de dados compartilhado), os sistemas ainda permanecerão com alto acoplamento entre si, em virtude da dependência dos serviços fornecidos pelo servidor. Conforme a quantidade de sistemas que utilizam o modelo RPC aumentar, maior será a dificuldade de manutenção no lado servidor, detentor das rotinas fornecidas aos clientes.

3.2.4 *Middleware Orientado a Mensagens*

Um dos grandes desafios da integração de sistemas, conforme destacado ao longo deste presente trabalho, é aumentar a colaboração entre as aplicações com o mínimo de acoplamento possível, de tal forma que não aumente a incerteza do funcionamento entre os sistemas tanto em tempo de desenvolvimento da solução quanto durante a execução das aplicações. Conforme observado nos tópicos anteriores, os estilos de integração utilizando banco de dados compartilhado e RPC geram relevante acoplamento, e a transferência de arquivos apresenta o atraso na entrega da informação, além das questões de controle do gerenciamento do arquivo em si.

Utilizar um *middleware* orientado a mensagens reúne o melhor da transferência de arquivos, com características particulares extras que tornam este estilo mais apropriado para maioria dos casos de integração entre as aplicações. De forma simplificada, uma aplicação pode enviar ou receber pequenos pacotes de dado, chamados de mensagens, a um componente intermediário e, se houver necessidade, aguardar a resposta do receptor de forma assíncrona. A imagem 11 ilustra a relação entre os componentes que compõem este estilo.

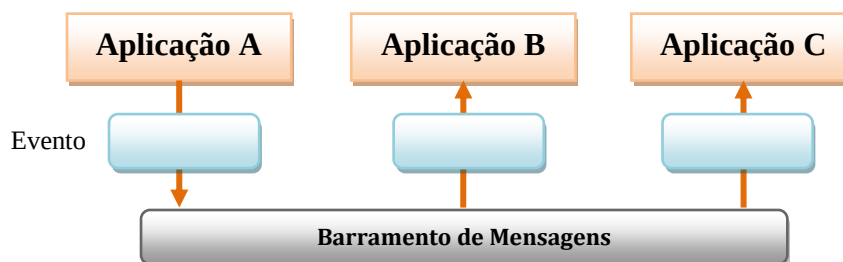


Figura 11 – *Middleware Orientado a Mensagens*
Fonte: Hohpe & Woolf (2003, p. 73). Adaptado pelo autor

As aplicações integradas a um sistema de mensageria são associadas, cada uma, a duas filas. Uma que permite o envio de mensagens (fila de saída) e a outra permitindo a receção de mensagens (fila de entrada). Segundo (Serain, 2002, p. 150), o diálogo entre duas aplicações através de mensagens apresenta os passos a seguir:

- Aplicação A (emissor) inicia o diálogo. Ela mesma se associa a duas filas de comunicação, realizando esta ação a partir de uma operação fornecida pelo *middleware*. As filas de mensagens representam o acesso ao barramento de mensagens;
- O emissor introduz a mensagem a ser enviada na fila de saída (através de alguma primitiva fornecida pelo barramento de mensagens). A mensagem contém, além do dado a ser enviado, o nome da fila de destino. A partir do momento que a mensagem é anexada à fila de saída, a aplicação emissora pode aguardar a resposta de forma assíncrona ou síncrona;
- O barramento de mensagens lê a mensagem e transfere à fila de destino;
- A partir do momento que a aplicação de destino se conecta às filas, ela aguarda por novas mensagens. Quando uma mensagem estiver disponível na fila de entrada, ela submete uma primitiva de leitura para realizar a operação de leitura da mensagem;
- Após processar a mensagem recebida, o receptor pode retornar uma mensagem para a aplicação emissora. Neste caso, ela submete uma mensagem na fila de saída;
- O *middleware* recupera a mensagem da fila e transmite para o sistema que iniciou o diálogo;
- A aplicação emissora que estava aguardando pela resposta lê a mensagem que foi transmitida pelo *middleware*, concluindo assim o diálogo.

Uma importante característica ao utilizar este estilo de integração é a assincronia. A aplicação que deseja disponibilizar uma mensagem não precisa aguardar pela resposta dos sistemas que dependem de sua mensagem, assim como a aplicação que precisa da mensagem não depende necessariamente que o emissor esteja disponível, mas sim que o mesmo tenha enviado a mensagem ao *middleware*.

Sistemas de mensageria fornecem baixo acoplamento, pois a única dependência das aplicações integradas é em relação ao barramento de mensagens, que deve ser conhecido por todas as aplicações visto que as mensagens dos emissores são direcionadas a ele, assim como o *middleware* é o responsável por direcionar os pacotes de dados aos respectivos

destinatários. Outro papel do barramento de mensagens é o de realizar as devidas transformações necessárias dos dados de acordo com o sistema receptor.

3.3 TOPOLOGIAS DE INTEGRAÇÃO

Conforme observado no capítulo anterior, a necessidade das organizações em realizar integração entre os sistemas tem crescido gradativamente, especialmente após a popularização e crescimento de mercados voltados a ERPs, CRMs e a própria Internet. Uma pesquisa realizada com 557 CIOs em novembro de 2003 pelo grupo IDC (*International Data Corporation*), a respeito das prioridades de investimento em TI para o ano de 2004, constatou que a integração com os sistemas ocupa o terceiro lugar no *ranking*, demonstrando portanto a crescente demanda por investimento, superando inclusive itens como a segurança dos dados e da informação (Chappell, 2004). A tabela 1 apresenta o resultado da pesquisa, exibindo em números percentuais o orçamento dedicado ao item relativo a TI dividido por setor da indústria.

Tabela 1 – Importância da integração na indústria (em % de respostas)

	Total	Instituições Financeiras	Manufatura	Governo / Educação / Saúde	Serviços	Telecomunicações	Transporte	Atacado / Distribuição
Infraestrutura	21	17	24	26	17	15	26	29
Custo de TI	20	23	24	18	16	31	22	14
Integração	16	16	17	9	20	15	30	4
Segurança	12	18	9	16	12	8	-	14
Projetos de Internet	9	11	6	10	9	4	11	7
Linux	4	1	4	3	8	8	-	-
Otimização de Software	4	4	6	4	1	8	-	11
Dispositivos Móveis	2	-	1	3	4	-	-	-
Redes sem fio	2	1	2	3	2	4	-	-
Outros	1	9	8	8	11	8	11	22
N=	557	82	109	116	143	26	27	28

Fonte: Chappell (2004, p. 34)

Diante dos resultados da pesquisa, evidencia-se que a integração de sistemas tem importante papel no mercado, exigindo das equipes de TI selecionar as melhores opções de solução para os problemas em suas respectivas organizações. Além das técnicas e estilos de

integração observados anteriormente, outro importante fator a ser estudado é em relação à topologia a ser empregada na solução, ou seja, a arquitetura como um todo a ser construída para suportar a integração entre os sistemas. Os tópicos a seguir descrevem uma arquitetura acidental, que deve ser evitada, e as topologias que podem ser utilizadas no processo de integração de sistemas de uma organização, de acordo com (Chappell, 2004) *et al.*

3.3.1 Arquitetura Acidental

Conforme observado na figura 1, apresentada no capítulo anterior, uma arquitetura acidental representa um estado de caos no que diz respeito à integração de sistemas. Ela é concebida a partir de pequenas soluções para problemas específicos de comunicação entre aplicações, originando as integrações “*link-a-link*”. Estas implementações são extremamente específicas para integrar dois sistemas e normalmente não são planejadas tendo-se a visão de toda a estrutura relevante de *softwares* que constituem a organização, o grau de reaproveitamento da solução a médio e longo prazo e as melhores práticas de mercado que podem ser estudadas e utilizadas na resolução do problema (Linthicum, 1999).

Em alguns cenários, a arquitetura é planejada pela equipe de TI e/ou arquiteto, contudo sob a perspectiva departamental, considerando apenas sistemas utilizados pelo setor a ser beneficiado pelo projeto de integração. Porém, esta forma de planejamento é equivocada, visto que em longo prazo soluções *link-a-link* gerarão silos (ou “ilhas”) de informação (ilustrado na figura 12) por conta da impossibilidade de se adaptar rapidamente a arquitetura de acordo com as novas necessidades da organização, tendo-se como possível consequência a geração de dados redundantes e dificuldade no compartilhamento de informações entre os departamentos da empresa, bem como entre seus clientes e parceiros.

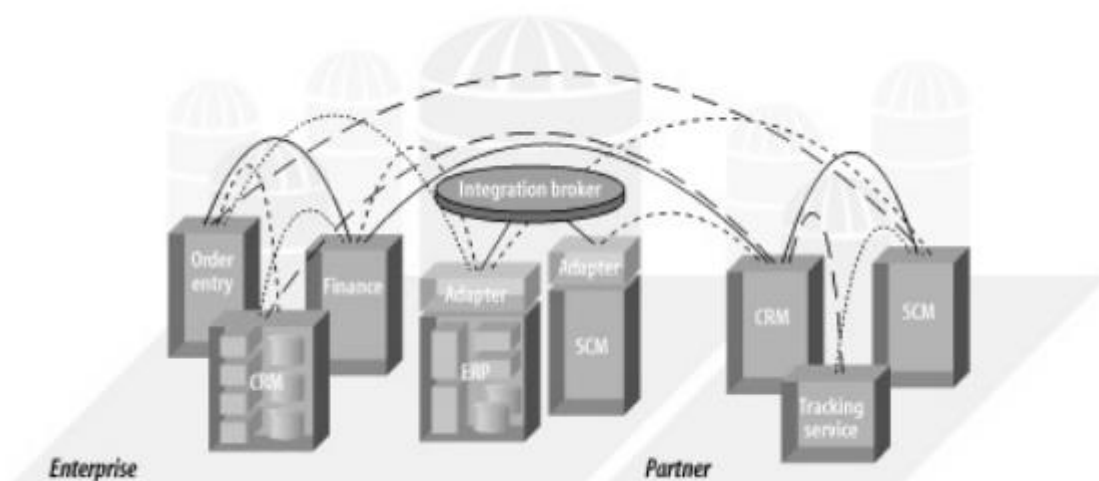


Figura 12 – Silos de informação gerados pela arquitetura acidental
 Fonte: Chappel (2004, p. 40)

Na figura 12, as linhas em diferentes formatos representam distintas tecnologias empregadas na resolução dos problemas de integração. A infraestrutura pode apresentar técnicas como transferência de arquivos por FTP, conexões diretas através de *sockets*, MOM proprietário, CORBA ou algum outro mecanismo de RPC. Além disso, os sistemas alvo das integrações podem transferir dados através de algum padrão XML, como SOAP. O *broker* central na figura representa uma ilha de integração, desempenhando o papel de integração a nível departamental apenas, em vez de central, característica comum em arquiteturas acidentais (Chappell, 2004).

As organizações estabelecem políticas de segurança dos dados e da informação, tanto para alinhamento estratégico como em virtude de leis que estejam em vigor nos países. Quando há existência de silos de informação, além dos possíveis problemas citados anteriormente, a organização pode encontrar dificuldades para tornar viáveis as políticas concebidas pela alta gerência, visto que em cada projeto de integração pode haver distintos mecanismos de segurança da informação nas conexões criadas entre as aplicações, ou simplesmente não existir, inviabilizando uma forma de garantir que os sistemas estejam em conformidade com as normas de segurança em TI da empresa.

Por fim, uma arquitetura acidental representa um alto investimento em infraestrutura que não possui características próprias para suportar adequadamente integração, como flexibilidade a mudanças. A longo prazo, alterações na arquitetura apresentam maior complexidade à medida que novas soluções *link-a-link* aumentam, visto que os sistemas tornam-se mais acoplados e, portanto, dependentes entre si. Como consequência a este fator,

alterações em um sistema podem afetar em grande escala outros em cascata, reduzindo o tempo de reação às mudanças da organização e de novas oportunidades de negócio. O alto acoplamento entre as aplicações promove fragilidade e insegurança a novas alterações, tornando difícil a realização de manutenções futuras. Sumarizando, de acordo com (Chappell, 2004, p. 44), a arquitetura accidental possui como pontos negativos:

- Alto acoplamento, fragilidade e inflexibilidade a mudanças;
- Elevados custos com manutenções por conta das múltiplas soluções *link-a-link*;
- Alterações em uma aplicação afetam outras aplicações;
- Lógica de roteamento da informação normalmente é *hardcoded*;
- Não há um modelo de segurança da informação comum;
- Normalmente não há uma API;
- Não há protocolos de comunicação comuns;
- Não há uma base comum em que possa se estabelecer e construir boas práticas;
- Dificuldade em suportar processamento assíncrono;
- Não confiável;
- Não há ferramentas para monitorar e gerenciar *deploy* de aplicações e componentes integrados.

3.3.2 *Brokers de Integração*

Como uma alternativa à arquitetura accidental, em meados dos anos 90 surgiu o conceito dos *brokers* de integração, que consiste em *middlewares* responsáveis por centralizar o roteamento entre as aplicações através da topologia *hub-and-spoke*. Estas arquiteturas são comumente constituídas por MOM ou servidor de aplicações e permitem, em comparação a uma arquitetura accidental, a separação do código responsável pela integração dos sistemas dos processos de negócio da empresa (Linthicum, 1999).

A perspectiva de uma topologia *hub-and-spoke* no que diz respeito à integração é de todo o conjunto de aplicações que constituem a estrutura de sistemas, contudo a visão de integrações locais ou regionais não é permitida. Os processos de negócio que são construídos no topo da arquitetura não contemplam regras específicas de departamentos ou pequenas

unidades da empresa, mas sim apenas da organização como um todo. A integração por *brokers* pode ter limitações de acordo com o MOM utilizado.

Ao longo da década de 90 muitas organizações implantaram a topologia *hub-and-spoke* com o objetivo de atender suas necessidades de integração. A fim de verificar o resultado das implantações ao longo da década, um estudo promovido pela Forrester Research em dezembro de 2001 constatou que muitos projetos conduzidos com este objetivo-fim extrapolaram o *deadline* e o orçamento planejados, colocando em dúvida o sucesso oferecido pelos *brokers* de integração (Chappell, 2004). A seguir, algumas estatísticas extraídas do estudo:

- Em média, os projetos de integração consomem 20 meses para concluir;
- Menos de 35% dos projetos finalizam dentro do tempo e orçamento planejados;
- 35% do orçamento dedicado à manutenção de *software* são utilizados para manter comunicações *link-a-link*;

De acordo com os padrões de mercado, os custos da implantação EAI são considerados elevados. Em linhas gerais, um projeto de integração de um único software pode variar entre \$250.000,00 a \$1.000.000,00 considerando apenas a licença de uso. Este valor pode aumentar de cinco a 20 vezes ao incluir custos de serviços de consultoria, oriundos por conta da alta gama de ferramentas proprietárias utilizadas em uma topologia *hub-and-spoke*.

Outro complicador importante a ser destacado é em relação às habilidades e conhecimentos necessários para desempenhar as atividades associadas à implantação da arquitetura. As ferramentas e tecnologias normalmente utilizadas possuem característica proprietária e, portanto, os treinamentos e os manuais para capacitação podem não ser facilmente acessíveis. Quando disponível, a organização pode optar em treinar seus colaboradores e compor uma equipe interna responsável por implantar a topologia, porém a curva de aprendizagem consome em média seis meses. Diante dos obstáculos, e também da complexidade da instalação, configuração, implantação e gerenciamento das ferramentas, a opção normalmente empregada é a de contratar consultores especialistas, resultando em maiores custos no projeto e dificuldade em construir bases de conhecimento que possam ser aproveitadas futuramente.

Como resultado de uma década de projetos empregando EAI como solução, muitas organizações criaram silos de integração, ou seja, distintos conjuntos de sistemas integrados parcialmente ou de forma a não oferecer extensibilidade a projetos futuros. Os resultados

negativos e insatisfatórios da arquitetura EAI abriram campo para o surgimento de uma nova topologia, que propõe as soluções dos problemas identificados com o uso da topologia *hub-and-spoke*.

3.3.3 ESB

Barramento de Serviços Empresariais, ou apenas ESB (*Enterprise Service Bus*, em Inglês), é uma topologia fundamentada em padrões abertos de mensageria síncrona e assíncrona, criada em meados dos anos 2001, que provê interoperabilidade segura entre aplicações empresariais através de XML, interfaces para comunicação via *web services* e roteamento de documentos baseado em regras padronizadas. A principal diferença em relação à topologia *hub-and-spoke*, é que enquanto os *brokers* de integração concentram todas as funcionalidades possíveis que um sistema pode utilizar, na arquitetura ESB as funcionalidades são compreendidas como componentes separados, desacoplados, que podem ou não ser incorporados na integração entre os sistemas.

Aproveitando algumas características da arquitetura EAI, uma topologia ESB fornece um meio centralizado para configuração dos pontos de comunicação de mensageria, transformação de mensagens de acordo com o formato esperado pelas aplicações, conversão dos principais protocolos de comunicação, capacidade de roteamento das mensagens para os destinatários apropriados, habilidade para construir mensagens *off-line* concatenando dados remanescentes até que o pacote de informação esteja completo para envio, facilidade de monitoramento e administração dos recursos fornecidos pela arquitetura e segurança da informação através de canais confiáveis de comunicação.

Uma topologia ESB se sobressai a uma arquitetura EAI em alguns aspectos, muito em virtude de sua abordagem orientada a serviços independentes. A sua arquitetura é mais leve, pois em vez de concentrar todas as funcionalidades em um único *hub*, ela dispersa as funções em serviços que são interoperáveis, tornando facultativa a escolha do que será utilizado nas integrações. ESB também apresenta maior capacidade para integrar novas aplicações com poucas mudanças, maior escalabilidade e distribuição geográfica por permitir que específicos serviços sejam replicados e forte base para a implantação de uma arquitetura SOA.

Em relação a custos, de acordo com o grupo Forrester Research, enquanto a licença dos *brokers* de integração poderia iniciar com um custo de \$100.000,00 por projeto e, em média, apresentar valores entre \$400.000,00 a \$750.000,00, as licenças ESB podem custar em

torno de 10 a 15 vezes menos. Isto ocorre devido ao fato de que cada projeto de integração demandará apenas os serviços que são necessários para a implantação. Outra característica que contribui com custos menores é em relação à menor necessidade de contratar consultores, visto que por se basear em padrões conhecidos e abertos, a organização poderá formar a sua própria equipe e aproveitar o conhecimento adquirido em projetos futuros, minimizando custos com consultores externos (Chappell, 2004).

4 CONCEITO E MELHORES PRÁTICAS DE SOA

Após contextualizar o leitor com o início das primeiras técnicas de integração difundidas no mercado, como os *message brokers* propostos pela EAI, o presente capítulo tem por objetivo fornecer conceitos relativos às técnicas atuais de integração propostas pela SOA, tendo-se como alicerce principal o conceito de serviços *web*, sustentados por protocolo de comunicação bem definido e específico.

As seções que constituem esta divisão do trabalho estão organizadas nos seguintes itens: 4.1 abordará os fundamentos de SOA, explicando-se termos chave para entendimento da arquitetura, tendo-se tópicos descrevendo tecnologias que representam a primeira geração de *Web Services* como WSDL, SOAP e UDDI; o item 4.2 descreve conceitos relativos à segunda geração, tais como coreografia, orquestração, atividades de negócio e serviço, padrões de troca de mensagem e transações atômicas.

4.1 FUNDAMENTOS DE SOA

Embora seja um conceito que tem se destacado nos últimos anos e sido empregado em anúncios de grandes vendedores de tecnologia e em soluções propostas por gestores de TI, SOA é muitas vezes confundido com outros termos da Engenharia de Software, tornando sua compreensão confusa por profissionais do ramo de desenvolvimento de sistemas (Hau, Ebert, Hochstein, & Brenner, 2008). Para desmitificar o conceito, é necessário compreender dois termos chave a respeito deste assunto, conforme listado a seguir:

- **Arquitetura:** pode ser compreendida como a descrição de um *software* em termos de seus componentes principais, descrevendo suas relações e informações trafegadas entre eles. A arquitetura de um *software* não é projetada apenas concentrando no comportamento e estrutura de um sistema, mas também na sua utilização, desempenho, resiliência, funcionalidade, reuso, segurança, restrições tecnológicas, *tradeoffs* e outros (Rosen, Lublinsky, Smith, & Balcer, 2008).
- **Serviço:** é a abstração de uma atividade empresarial, descrevendo-se as entidades (como clientes, produtos, taxas, equipamentos, funcionários etc.), processos (como manufatura, vendas, recursos humanos etc.) e relacionamentos (fabricante e fornecedor, ou empregador e empregado, entre outros) envolvidos no domínio da atividade. Neste conceito, o provedor do serviço é responsável por realizar algum

trabalho útil para o solicitador do serviço determinado por um contrato que descreve o que será realizado e quais são as entidades envolvidas (Erl, et al., 2014).

Compreendido os termos-chave que compõem SOA, cabe salientar que há dois princípios fundamentais da arquitetura orientada a serviços, que basicamente são: baixo acoplamento e orientação a componentes, ambos com o objetivo de tornar os sistemas mais facilmente gerenciáveis. Para alcançar esta proposta, é prudente adotar as regras seguintes na implantação:

- **Orientação a Interfaces:** assemelha-se ao paradigma da orientação a objetos, onde em linhas gerais a interface define o contrato de utilização de uma rotina, descrevendo os possíveis parâmetros de entrada e saída; a forma como é executada ou implementada pouco importa para o utilizador;
- **Padronização:** para que se possa alcançar a proposta de uma arquitetura SOA no que diz respeito à componentização, é importante que os serviços sejam padronizados para que se possam trocar dados mais facilmente e assim possibilitarem integrações que constituam novas funcionalidades;
- **Autonomia e Modularização:** os serviços devem ser independentes entre si, ou seja, o funcionamento de um não necessariamente deve depender da existência de outro, tornando-o autônomo. Esta capacidade faz com que os serviços possam ser modularizados, diminuindo o acoplamento entre estes elementos de *software*;
- **Orientação a Negócio:** assim como na OOP um objeto tem como responsabilidade representar um elemento do mundo real no ambiente computacional, um serviço deve representar uma abstração de uma tarefa do mundo empresarial. Por exemplo, enquanto em um sistema um objeto pode representar a entidade “Cliente”, em SOA um serviço pode representar a tarefa “Pagamento com Cartão de Crédito”.

Embora o conceito de SOA não defina qual eixo tecnológico se deve empregar na implantação dos serviços, cabendo ao arquiteto e sua equipe adotar a melhor opção dentre as possíveis, uma forma comum e muitas vezes associada ao uso de SOA é o de *Web Services*, e por essa razão o presente trabalho abordará esta tecnologia nos tópicos a seguir.

4.2 WEB SERVICES

Conforme discutido anteriormente, um serviço é uma abstração de uma atividade da organização, cabendo definir a abrangência deste serviço, respeitando os princípios básicos apresentados no tópico anterior. Já um serviço *web*, ou mais conhecido como *web service*, é um método de comunicação empregado na integração e comunicação entre aplicações através de uma rede de computadores. Sua interface de utilização é descrita por um contrato, normalmente definido em WSDL, e pode possuir um ou mais esquemas XML e/ou políticas WS (Erl, et al., 2008).

Em linhas gerais, outros sistemas se comunicam com um *web service* através de mensagens serializadas em XML, encapsuladas pelo protocolo SOAP e enviadas via HTTP em conjunto com outros padrões empregados na implementação destes serviços. O contrato expõe as capacidades públicas do *web service* como operações, ou seja, define quais são os métodos que podem ser invocadas pelo consumidor do serviço. Uma característica importante dos *web services* é que eles são baseados em modelos universais e públicos, em particular o HTTP e XML, tornando-os atrativos.

O objetivo de um *web service* é o de disponibilizar recursos de um **software** em uma rede de forma normalizada, podendo o recurso consistir em uma tarefa computacional simples ou complexa. Respeitando-se obviamente o desenho técnico da solução, a priori qualquer sistema poderá consumir os recursos fornecidos pelo *web service*, pois independe de plataforma e linguagem de programação. A lógica encapsulada no serviço pode envolver, por exemplo, um recurso de uma aplicação legada, que poderia ser exposta via um contrato *web service*, conforme imagem 13. Normalmente os *web services* são desenvolvidos em tecnologias modernas, como Java ou .NET (Rosen, Lublinsky, Smith, & Balcer, 2008).

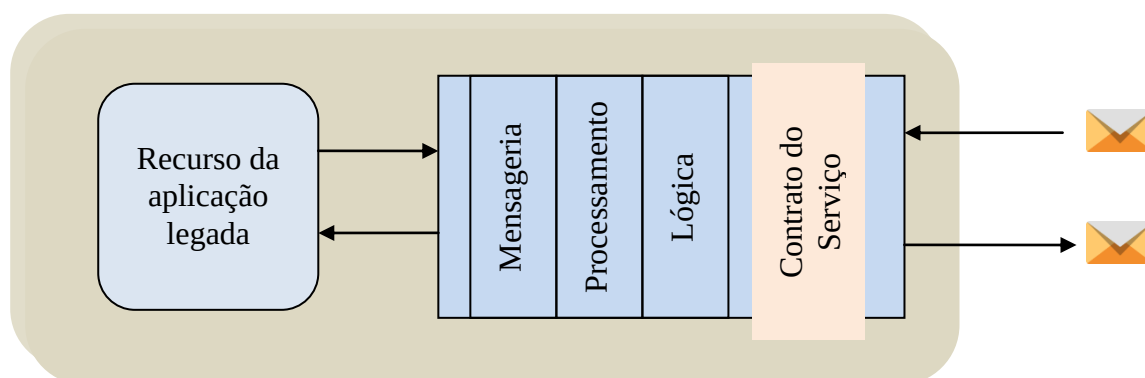


Figura 13 – Web Service agindo como um provedor de serviço
Fonte: Erl, et al. (2008, p. 34). Adaptado pelo autor

Os tópicos a seguir descrevem conceitos centrais da tecnologia de *Web Services*, como XML, WSDL, SOAP e UUID. O item sobre REST, embora não seja a respeito de *Web Service* propriamente dito, será brevemente discutido para exemplificar que uma arquitetura SOA não necessariamente precisa expor os serviços através de *Web Services*, mas sim de outras tecnologias, como a citada.

4.2.1 XML

Acrônimo de *eXtensible Markup Language* (Linguagem de Marcação Extensível), XML é uma linguagem baseada em texto utilizada para estruturar dados em documentos eletrônicos, com a vantagem de não ser limitada a um conjunto específico de rótulos. É um padrão extremamente poderoso para compartilhar informações estruturadas através da Internet e das redes internas e externas presentes nas corporações (Erl, et al., 2008).

Os dados presentes em um documento XML são auto descritíveis, ou seja, a própria estrutura do arquivo descreve como os dados são relacionados entre si, tornando a leitura do documento facilitada pelos transformadores de arquivos XML existentes no mercado. Com o auxílio desta linguagem, as empresas podem criar seus próprios padrões de documentos para compartilhar informações entre si. Por ser uma recomendação da *World Wide Web Constortium* (W3C), a linguagem acabou se popularizando e tornou-se uma das principais alternativas para transmissão de dados empregadas nas soluções de *middleware* e integração de sistemas (Fawcett, Quinn, & Ayers, 2012).

Em linhas gerais, o XML é constituído por elementos, definidos por *tags*. Um elemento sempre tem uma *tag* de início e fim, delimitando o seu escopo e conteúdo. Todos os elementos de um documento XML constituem o elemento principal, conhecido como “*root*” ou raiz, permitindo desta forma que os elementos possam ser agrupados em estruturas hierárquicas. O nome do elemento descreve o seu conteúdo, enquanto que a estrutura descreve o seu relacionamento com outros elementos. Por fim, um documento XML pode conter atributos nos elementos, introduzidos na *tag* inicial, atuando como descritores de suas características (Fawcett, Quinn, & Ayers, 2012). A imagem 14 exemplifica um simples documento XML.

```

1  <?xml version="1.0"?>
2  <carta>
3      <para>Mãe</para>
4      <de>Bob</de>
5      <mensagem>Feliz dia das mães!</mensagem>
6      <presentes>
7          <item ordem="1">Flores</item>
8          <item ordem="2">Perfume</item>
9          <item ordem="3">Chocolate</item>
10     </presentes>
11 </carta>

```

Figura 14 – Exemplo de documento XML

Uma das principais vantagens do XML é em relação à sua simplicidade, permitindo que aplicativos facilmente possam gerar ou ler arquivos escritos sob esta linguagem. Desta forma, diversos outros padrões acabaram utilizando o XML como base, conforme os tópicos seguintes deste capítulo, que farão referência constante a esta linguagem de marcação de texto.

4.2.2 WSDL

Web Service Description Language, ou apenas WSDL, é uma recomendação da W3C para descrever as funcionalidades fornecidas por um *web service*. Basicamente é um documento escrito em XML, portanto independente de plataforma, que provê uma completa descrição sobre o serviço e suas capacidades, incluindo formato da requisição inicial e o resultado da resposta, além de detalhes de como exatamente o serviço deverá ser executado, incluindo informações sobre o servidor, porta de comunicação e padrão de URL (Sweeney, 2010).

Realizando um paralelo com o que foi discutido anteriormente neste presente trabalho, o WSDL é utilizado para descrever um contrato de utilização entre o *web service* e o restante do mundo, assim como acontece com o paradigma de objetos sustentado pelos padrões COM e CORBA, em que nestes casos o contrato de utilização é descrito em IDL e que basicamente descreve quais são os métodos disponíveis, bem como seus parâmetros de entrada e retorno. O documento WSDL delimita quais mensagens um servidor SOAP espera para disponibilizar os serviços, assim como quais mensagens ele retornará (Fawcett, Quinn, & Ayers, 2012).

Um documento WSDL é composto, basicamente, pelas seguintes seções: *definitions*, *types*, *message*, *portType*, *binding* e *service*. A seguir, alguns exemplos de cada trecho que compõe o arquivo e uma breve explicação.

4.2.2.1 Definitions

É o elemento raiz do documento, contendo todas as definições do *web service*, conforme exemplificado na imagem 15. Todos os blocos seguintes estão contidos entre as *tags* de abertura e fechamento *definitions*.

```

1  <?xml version="1.0"?>
2  <definitions name="ServicoOlaWS"
3      targetNamespace="http://www.examples.com/wsd1/ServicoOlaWS.wsd1"
4      xmlns="http://schemas.xmlsoap.org/wsd1/"
5      xmlns:soap="http://schemas.xmlsoap.org/wsd1/soap/"
6      xmlns:tns="http://www.examples.com/wsd1/ServicoOlaWS.wsd1"
7      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
8  </definitions>

```

Figura 15 – Bloco *definitions* de um documento WSDL

4.2.2.2 Types

Bloco opcional que define tipos de dados particulares, como por exemplo, um objeto complexo que será enviado ou retornado por um método. A imagem 16 demonstra um exemplo deste trecho.

```

1  <types>
2      <xsd:schema xmlns=""
3          xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
4          targetNamespace="http://www.examples.com/wsd1/ServicoOlaWS.wsd1">
5          <xsd:complexType name="MensagemOla">
6              <xsd:sequence>
7                  <xsd:element name="primeiroNome" type="xsd:string"/>
8                  <xsd:element name="segundoNome" type="xsd:string"/>
9              </xsd:sequence>
10         </xsd:complexType>
11     </xsd:schema>
12 </types>

```

Figura 16 – Bloco *types* de um documento WSDL

4.2.2.3 Message

Define os tipos de dados incluídos em mensagens SOAP enviadas e retornadas pelo servidor. A figura 17 ilustra um exemplo, em que a mensagem de nome “ServicoOlaRequestMsg” define o tipo de dado a ser enviado para o servidor na requisição e “ServicoOlaResponseMsg” define o tipo de dado a ser retornado pelo servidor:

```

1  <message name="ServicoOlaRequestMsg">
2    <part name="getMensagemOla" type="typens:MensagemOla"/>
3  </message>
4  <message name="ServicoOlaResponseMsg">
5    <part name="mensagemOlaResponse" type="xsd:string"/>
6  </message>

```

Figura 17 - Bloco *message* de um documento WSDL

4.2.2.4 PortTypes

Este trecho descreve as operações fornecidas pelo *web service*, contendo as mensagens de requisição (incluída na *tag input*) e resposta definidas anteriormente (incluída na *tag output*). Observe que na figura 18 as mensagens são as que foram definidas no exemplo anterior:

```

1  <portType name="ServicoOlaPortType">
2    <operation name="EnviarOla">
3      <input message="typens:ServicoOlaRequestMsg"/>
4      <output message="typens:ServicoOlaResponseMsg"/>
5    </operation>
6  </portType>

```

Figura 18 – Bloco *portTypes* de um documento WSDL

4.2.2.5 Binding

Até este momento, o que foi definido no documento WSDL foram os tipos complexos, caso existam no *web service*, as mensagens a serem enviadas e recebidas e as operações existentes, contudo é o trecho *binding* que fornece qual o protocolo de comunicação que deve ser utilizado para que a chamada do método aconteça. A figura 19 exemplifica este trecho, em que a transmissão incluída no exemplo trata-se do SOAP:



```

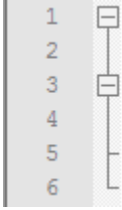
1 <binding name="ServicoOlaBinding" type="typens:ServicoOlaPortType">
2   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
3   <operation name="EnviarOla">
4     <soap:operation/>
5     <input>
6       <soap:body use="encoded"
7         encodingStyle="http://www.w3.org/2003/05/soap-encoding"
8         namespace="urn:examples:servicoola"/>
9     </input>
10    <output>
11      <soap:body use="encoded"
12        encodingStyle="http://www.w3.org/2003/05/soap-encoding"
13        namespace="urn:examples:servicoola"/>
14    </output>
15  </operation>
16 </binding>

```

Figura 19 – Bloco *binding* de um documento WSDL

4.2.2.6 Service

Por fim, o último trecho do documento WSDL define de fato as informações de URL em que o *web service* está localizado para que possa ser encontrado e consumido. A figura 20 exemplifica este bloco final:



```

1 <service name="ServicoOla">
2   <documentation>Documento WSDL para ServicoOla</documentation>
3   <port name="ServicoOlaPort" binding="typens:ServicoOlaBinding" >
4     <soap:address location="http://www.examples.com/EnviarOla/" />
5   </port>
6 </service>

```

Figura 20 – Bloco *service* de um documento WSDL

4.2.3 SOAP

É uma especificação da W3C que estabelece um protocolo de comunicação utilizado em ambientes distribuídos e descentralizados. SOAP define padrões das mensagens a serem trafegadas na implementação dos *web services*, baseado no formato XML. Este protocolo utiliza a camada de aplicação para transportar os dados, como RPC e normalmente HTTP (Sweeney, 2010).

Dentre as principais vantagens do SOAP, pode-se destacar: independência de linguagem, plataforma e fabricante; facilidade de implementação, necessitando apenas de um

servidor *web* e um script CGI ou uma página ASP, por exemplo; e menor esforço de configuração em segurança e *firewalls* assumindo que o HTTP seja o protocolo de comunicação utilizado (Fawcett, Quinn, & Ayers, 2012).

Uma mensagem SOAP é essencialmente um documento XML, contendo um elemento chamado *envelope*, que é o principal no documento e responsável por basicamente acoplar os filhos, além dos elementos *header*, opcional e responsável por conter informações como *timeout*, e *body*, contendo em si a mensagem a ser enviada ou devolvida pelo servidor. A figura 21 exemplifica uma mensagem SOAP de requisição.

```

1  <?xml version="1.0"?>
2  <soap:Envelope
3      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5      <soap:Body>
6          <m:EnviarOla xmlns:m="http://www.examples.com/EnviarOla">
7              <m:MensagemOla>
8                  <m:primeiroNome>Renato</m:primeiroNome>
9                  <m:segundoNome>Marquez</m:segundoNome>
10             </m:MensagemOla>
11         </m:EnviarOla>
12     </soap:Body>
13 </soap:Envelope>

```

Figura 21 – Exemplo de requisição com mensagem SOAP

Enquanto que a figura 22 ilustra uma possível mensagem SOAP de resposta do servidor.

```

1  <?xml version="1.0"?>
2  <soap:Envelope
3      xmlns:sopa="http://www.w3.org/2001/12/soap-envelope"
4      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5      <soap:Body>
6          <EnviarOlaResponse>
7              <string>Boa noite, sr. Renato Marquez</string>
8          </EnviarOlaResponse>
9      </soap:Body>
10 </soap:Envelope>

```

Figura 22 – Exemplo de resposta com mensagem SOAP

4.2.4 UDDI

As tecnologias apresentadas anteriormente têm seus papéis bem definidos, onde o SOAP é o responsável por transportar os dados e o WSDL por descrever os *Web Services*, contudo é o UDDI responsável por permitir que os serviços *web* sejam encontrados em uma rede de computadores. É uma plataforma independente baseada em XML utilizada para registrar os serviços *web* em listas que podem ser pesquisadas através da Intranet, Extranet ou Internet, a depender das configurações empregadas pela organização. UDDI utiliza padrões da W3C e da IETF, como o XML, HTTP e DNS (Sweeney, 2010).

De acordo com Rosen, Lublinsky, Smith, & Balcer (2008, p. 323) o funcionamento do UDDI é similar a uma lista telefônica, dividida em três tipos de “páginas” diferentes, listadas a seguir:

Páginas brancas: contém informações a respeito do provedor do *web service*, como nome, telefone para contato, endereços e também detalhes a respeito do serviço a ser fornecido, como por exemplo o link do arquivo WDSL descrevendo sua interface de utilização. Estes dados podem ser utilizados para pesquisas por clientes, filiais ou parceiros de negócio que queiram buscar os serviços por dados da empresa;

Páginas amarelas: apresenta listagens comerciais baseadas nos tipos de negócios oferecidos pelas empresas, categorizadas por taxonomias comuns. Útil para se pesquisar *web services* fornecidos por provedores que atuem em ramos parecidos ou iguais;

Páginas verdes: orientado a descrever detalhes técnicos dos serviços fornecidos pelo provedor, possibilitando encontrar *web services* a partir de suas funcionalidades, por exemplo. Pode sumarizar informações a respeito do propósito do serviço, bem como estrutura da informação a ser transmitida e respondida e assim por diante.

4.2.5 REST

REST é um estilo de arquitetura e uma abordagem diferenciada de comunicação empregado na construção de *web services*, como uma alternativa ao uso do protocolo SOAP, que em comparação é uma alternativa mais leve e que consome menos banda da rede para transmissão de dados entre o cliente e o provedor do serviço (Fielding, 2000). Frente a esta característica de possuir melhor desempenho, REST tem sido cada vez mais utilizado na construção de APIs na nuvem, em aplicações *mobile* e em redes sociais.

Um importante princípio que norteia REST é o de que os recursos são representados nas URLs. Resumidamente se, por exemplo, desejar consultar informações de um produto identificado pelo código 120, a URL pode ter um formato similar a `http://meuServidor.com/produtos/120`. Esta não é uma regra para que um *web service* construído sobre arquitetura REST funcione, contudo é uma forte recomendação, justificada pelo segundo princípio.

A segunda característica da arquitetura diz respeito à ação que o cliente deseja desempenhar com o recurso procurado, que basicamente será traduzido com o verbo HTTP a ser utilizado. Se, por exemplo, desejar recuperar informações do produto 120, a requisição deve ser realizada com o método GET (Fawcett, Quinn, & Ayers, 2012). A tabela 2 exibe a relação dos verbos HTTP com as suas respectivas ações.

Tabela 2 – Relação de métodos HTTP na arquitetura REST

Método HTTP	Ação na Arquitetura REST
GET	Recuperar dados de um recurso
POST	Criar um novo recurso
PUT	Atualizar dados de um recurso existente
DELETE	Excluir um recurso

4.3 EXTENSÕES WS-*

Os conceitos apresentados anteriormente definem o que é conhecido como a primeira geração dos *web services*. WSDL, SOAP e UDDI estão relacionados a questões importantes envolvendo serviços *web*, como protocolo de comunicação e envio/recebimento dos dados, descrição do contrato de utilização dos serviços e descoberta dos *web services*, contudo em um ambiente corporativo, muitas outras questões precisam ser atendidas, como governança, segurança, políticas, controle transacional, garantia de envio de mensagens e assim por diante.

Para atender a estas questões, a segunda geração de *web services* foi concebida, fornecendo as especificações necessárias para que cada um dos requisitos fosse atingido. As extensões atuam como complemento ao que fora criado na geração anterior, fortalecendo a orientação a serviços, forte conceito empregado em SOA (Erl, 2005). De acordo com (Rosen, Lublinsky, Smith, & Balcer, 2008), o prefixo WS-* define o conjunto de padrões utilizados na construção dos *web services*, como os listados a seguir:

- **WS-Addressing:** especificação para estabelecer regras e parâmetros de comunicação entre *web services*, como roteamento das mensagens SOAP trafegadas entre consumidor e provedor do serviço;
- **WS-Discovery:** especificação técnica que define um protocolo de descoberta *multicast*³ de *web services* em uma rede local de computadores;
- **WS-Federation:** é uma especificação para identificação federada, relativa à autenticação e autorização para fins de acesso e comunicação de *web services* através de múltiplos domínios;
- **WS-Policy:** é uma especificação que define um conjunto de regras de segurança a serem empregadas na comunicação dos *web services*, definindo por exemplo, a obrigatoriedade ou não do uso de *tokens* de segurança, algoritmos de criptografia suportados e políticas de segurança empregadas pelo provedor do serviço;
- **WS-Security:** define questões de segurança na comunicação de *web services*, como integridade e confidencialidade na troca de mensagens e permitindo o uso de diversos mecanismos de *tokens* de segurança, como SAML, Kerberos e X.509;
- **WS-Trust:** é uma extensão de WS-Security, que define mecanismos de garantia, renovação e validação de *tokens* de segurança previamente definidos no diálogo entre consumidor e provedor do serviço.

Os tópicos descritos a seguir referem-se às características presentes em uma arquitetura SOA suportadas pelas primeira e principalmente segunda gerações de *web services*.

4.3.1 Padrões de Trocas de Mensagens

Também conhecido como MEPs (*Message Exchange Patterns*), os padrões de troca de mensagens representam um conjunto genérico de padrões a serem aplicados na troca de mensagens entre dois ou mais serviços, definindo sequências possíveis de mensagens que podem ser enviadas e recebidas (Erl, 2005). Antes do surgimento de SOA e seus conceitos, *middlewares* orientados a mensagens, ou MOMs, já utilizavam alguns MEPs, portanto alguns

³ Em redes de computadores, *multicast* é um grupo de comunicação no qual uma informação é direcionada a um grupo de computadores de destino simultaneamente (Harte, 2008, p. 1).

padrões primitivos oriundos de antigas implementações foram incorporadas nos conjuntos mais moderno e complexo de trocas de mensagens.

O MEP a ser empregado na comunicação entre *web services* pode ser definido no cabeçalho da mensagem SOAP a ser enviada ao serviço de destino, permitindo desta forma que o padrão escolhido seja utilizado na troca de mensagens entre emissor e destinatário. Os tópicos a seguir descrevem apenas os dois tipos mais comuns de MEPs:

4.3.1.1 *Request-Response*

O mais popular dos padrões utilizado nas aplicações distribuídas, define basicamente que cada mensagem enviada por um emissor ao provedor do serviço deve ser respondida pelo mesmo provedor ao emissor. Este padrão pode ser incluído em comunicações síncronas e assíncronas entre serviços. A figura 23 exemplifica este padrão:

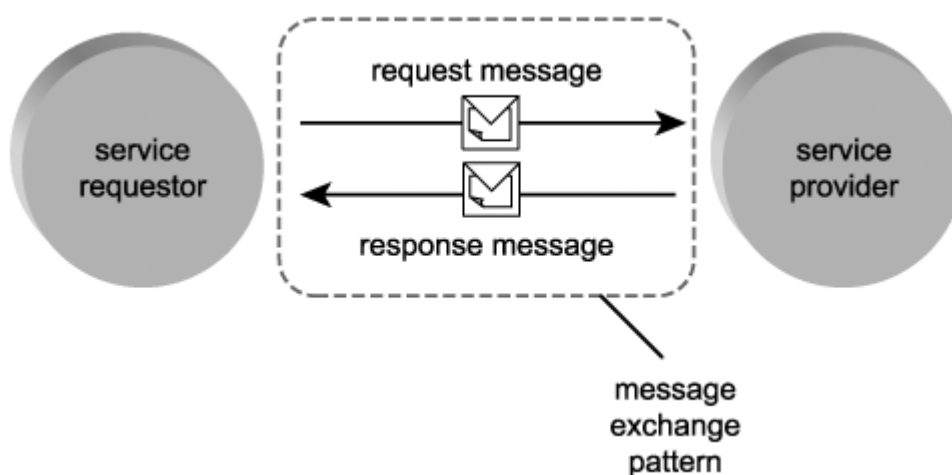


Figura 23 – MEP *Request-Response*
Fonte: Erl (2005, p. 163)

Um possível exemplo de utilização deste padrão é nos casos em que o consumidor do serviço (na imagem, representado pelo *service requestor*) envia uma mensagem contendo dados a serem validados pelo provedor do serviço (representado pelo *servidor provider*) e este por sua vez devolve uma mensagem com o resultado da validação. Este é um padrão muito comum e amplamente utilizado nas comunicações entre os serviços (Rosen, Lublinsky, Smith, & Balcer, 2008).

4.3.1.2 Fire-and-Forget

Este padrão assíncrono de troca de mensagens é fundamentado na transmissão unidirecional de mensagens de uma origem a um ou vários destinos. A principal diferença deste padrão para o anterior é que a mensagem de resposta não é esperada pelo emissor. O padrão de envio pode ser único, de emissor a um destino; pode ser múltiplo, enviando a um conjunto definido de destinatários; e *broadcast*, que é enviado a uma quantidade mais ampla de destinos (Erl, 2005). A imagem 24 ilustra o padrão *fire-and-forget*.

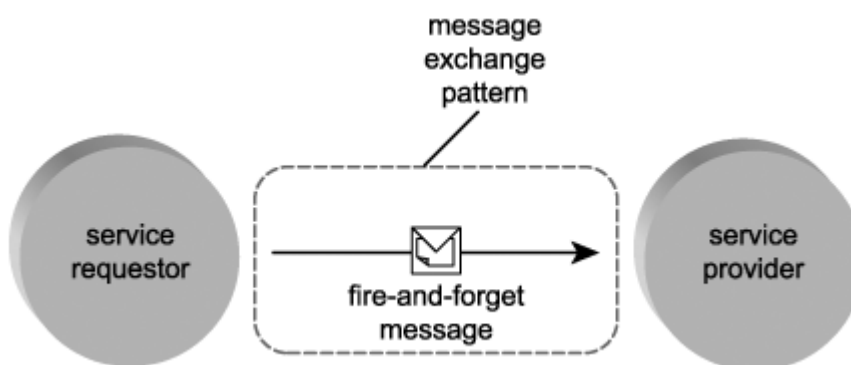


Figura 24 – MEP *Fire-and-Forget*
Fonte: Erl (2005, p. 164)

Um exemplo de utilização deste padrão pode ser encontrado no cenário seguinte: suponha que uma aplicação envie dados para serem validados por um serviço, contudo a mensagem de retorno contém informações de que os dados estão incorretos. Diante disto, o serviço de origem trata este caso como um erro interno, criando logs no servidor *web* e envia um e-mail de notificação. O e-mail de notificação pode ser realizado por outro serviço e, neste caso, o emissor envia uma mensagem no padrão *fire-and-forget* para o provedor responsável por enviar a notificação por e-mail.

4.3.2 Atividade de Serviço

Uma tarefa empresarial para ser completamente concluída muitas vezes envolve a realização de outras tarefas auxiliares. Traduzindo para o mundo dos serviços *web*, haverá situações em que para se atender um objetivo será necessária a execução de outros serviços *web*, cada qual desempenhando seu papel dentro do aglomerado em uma rede computadores. A interação de um conjunto de serviços trabalhando em conjunto para completar uma tarefa pode ser compreendida como uma atividade de serviço (Erl, 2005).

A realização de uma atividade de serviço pode ser concretizada apenas entre dois serviços, um emissor e um único destinatário, conhecido como uma atividade primitiva. Já atividades envolvendo três ou mais serviços são conhecidas como complexas, e podem de fato incluir muitos *web services*, além de diversos MEPs diferentes, para que uma determinada tarefa seja alcançada. A figura 25 ilustra um exemplo de uma atividade de serviço complexa:

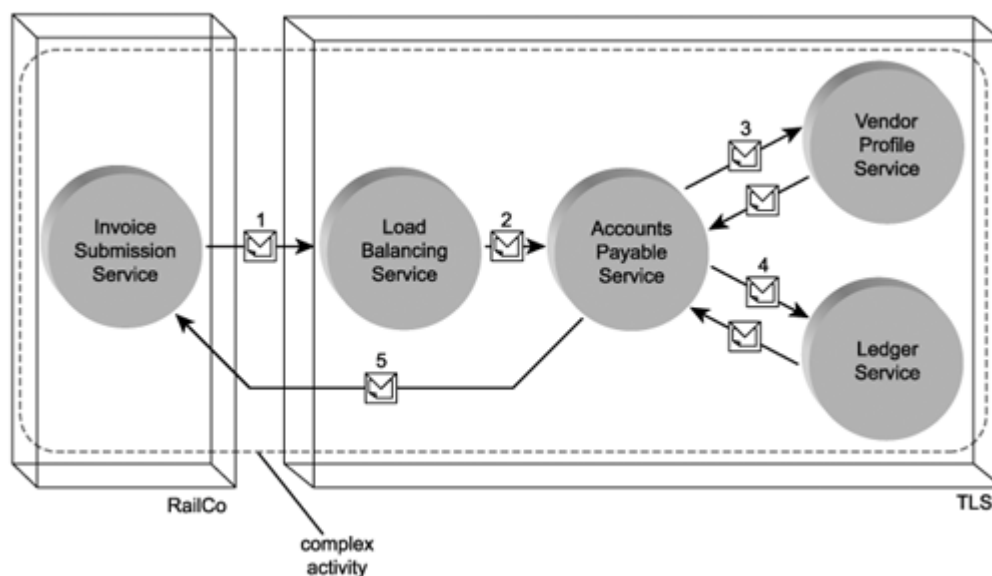


Figura 25 – Exemplo de atividade de serviço complexa
 Fonte: Erl (2005, p. 169)

A figura 25 exemplifica um cenário complexo envolvendo os passos a seguir:

- O serviço inicial, contido no ambiente RailCo (um sistema representando o lado emissor do exemplo), transmite uma mensagem contendo dados de nota fiscal;
- A mensagem é recebida por um serviço intermediário, contido no ambiente TLS (um sistema representado o lado provedor de serviço do exemplo), responsável por rotear a mensagem a outro serviço de acordo com as condições envolvidas. No exemplo, ele encaminha a mensagem ao serviço responsável por contas a pagar;
- O serviço de contas a pagar agirá como um controlador e iniciará a composição de serviços, processando inicialmente a mensagem. Em seguida, encaminhará ao serviço responsável pelo perfil do fornecedor para validar os dados de cabeçalho da nota fiscal;

- Após a resposta do *web service* do passo 3, agora o serviço de contas a pagar extrairá informações como taxas, descontos e valor total da nota e encaminha para o *web service* do livro-razão, responsável por registrar todas as notas fiscais da empresa;
- Finalmente, a atividade se encerra com o serviço de contas a pagar enviando uma mensagem de resposta ao serviço inicial decretando fim do processamento da requisição.

4.3.3 Coordenação

Cada atividade de serviço introduz um nível de contexto no ambiente de execução de uma aplicação. Informações do contexto consistem em detalhes a respeito da atividade, como características e ciclo de vida de execução. Quanto mais complexa for a atividade, mais informações de contexto serão encontradas (Erl, 2005). A complexidade pode estar relacionada a diversos fatores, conforme os destacados a seguir:

- Quantidade de serviços que compõem a atividade;
- Duração da atividade;
- A frequência com que a natureza da atividade muda;
- Possibilidade de existir múltiplas instâncias concorrentes da mesma atividade.

A coordenação é um mecanismo para gerenciar, preservar e/ou atualizar as informações de contexto para os participantes de atividades complexas. A coordenação pode ser desempenhada por um serviço, responsável por estabelecer um controle de chamadas aos outros serviços, para que a atividade seja concluída na ordem esperada. A composição de serviços responsáveis pela coordenação dos demais serviços consiste nos seguintes:

- **Serviço de Ativação:** Responsável pela criação de um novo contexto e pela associação deste contexto a uma atividade em particular a ser executada;
- **Serviço de Registro:** Permite que outros serviços, que serão envolvidos na execução da atividade, recebam informações do contexto enviadas pelo serviço de ativação;

- **Coordenador:** é o serviço controlador da composição, também conhecido como serviço de coordenação.

Na imagem 25, o trecho envolvendo os passos 3 e 4 ilustram um exemplo de coordenação. Embora não representado na figura, os serviços de contas a pagar, perfil de fornecedor e livro-razão são gerenciados por um coordenador, responsável por delimitar a ordem de chamada dos serviços: o primeiro, no exemplo, foi o de perfil de fornecedor que, respondendo com os dados necessários para o serviço de contas a pagar, o segundo serviço é enfim envolvido no processamento.

4.3.4 Transações Atômicas

Em um ambiente de sistemas corporativos, transações ocorrem a todo o momento. Quando se está manipulando determinados tipos de dados, a necessidade de encapsulá-los em uma única ação é fundamental para diversos requisitos de processo de negócio. Transações atômicas basicamente suportam as funcionalidades de *commit* e *rollback* para garantir integridade nas comunicações entre os *web services* (Erl, 2005).

As transações atômicas são gerenciadas pelo serviço coordenador, discutido no tópico anterior. Ele é o responsável por determinar se todos os serviços participantes responderam positivamente e, portanto, se as alterações deverão ser confirmadas ou abortadas. Em uma atividade, todos os serviços retornarão ao coordenador uma mensagem avisando que a operação foi concluída com sucesso ou com erro; se um deles informar que houve falha, o coordenador envia uma mensagem a todos os participantes orientando que a transação sofreu *rollback*.

De acordo com (Bernstein & Newcomer, 2009), ao discutir a respeito deste assunto, há quatro conceitos importantes acerca de transações atômicas, que formam o acrônimo ACID, citados a seguir:

4.3.4.1 Atomicidade

Esta característica garante que para que uma transação seja concluída com sucesso todas as operações envolvidas devem ser confirmadas. Se ao menos uma delas falhar, todo o processo deverá ser revertido e nenhuma alteração ser acatada. Como exemplo pode-se citar o caso de uma operação bancária envolvendo duas contas, A e B, onde um montante é

transferido da conta A para a B; não pode haver apenas crédito na conta B ou apenas débito da A.

4.3.4.2 *Consistência*

Esta propriedade define que uma transação realizada por um programa de computador deve garantir que a consistência inicialmente encontrada antes da sua execução deve permanecer após a sua conclusão, seja o término com uma reversão ou confirmação dos dados. Em termos de banco de dados, significa dizer que ao fim da transação, todas as chaves primárias, estrangeiras e demais regras das tabelas devem ser respeitadas e não podem estar violadas; se houver violação de qualquer regra que garanta a consistência, a transação deve ser abortada e nenhuma ação realizada poderá ser submetida.

4.3.4.3 *Isolamento*

Basicamente, o conceito de isolamento está relacionado à característica de que a execução de duas ou mais transações simultaneamente não afetarão os resultados de umas com as outras. Ou seja, o resultado das execuções deve ser necessariamente o mesmo se as transações fossem executadas em conjunto ou sequencialmente.

Um clássico exemplo de uma situação não isolada é em sistema bancário, em que duas transações deverão descontar os últimos R\$ 100,00 de uma determinada conta. Se as duas transações realizarem uma leitura na conta antes de uma delas concretizar a operação, ambas realizarão o desconto incorreto do montante, o que não aconteceria se a execução das transações fosse realizada sequencialmente.

4.3.4.4 *Durabilidade*

A quarta e última característica de uma transação atômica é em relação à capacidade dos dados que uma vez submetidos não poderão ser desfeitos, seja por falhas subsequentes de outras transações ou por erros do próprio sistema operacional. Este conceito é importantíssimo, pois uma vez que as alterações sofram *commit*, é esperado que transações futuras possam utilizar destas alterações para realizar novos processamentos.

4.3.5 **Atividade de Negócio**

Atividades de negócio regem longas atividades complexas de serviço, que podem demandar horas, dias ou até semanas para que uma atividade de negócio seja concluída. Neste intervalo de tempo, a atividade pode desempenhar inúmeras tarefas envolvendo diversos

serviços participantes. A diferença entre uma atividade de negócio e uma atividade comum de serviço é que os participantes deverão seguir regras específicas do negócio no tratamento de exceções, não sendo resumido apenas em *commit* e *rollback*, enquanto que em uma atividade regular os participantes devem apenas atender às regras de transações atômicas (Erl, 2005).

Por demandar, em algumas situações, semanas para que a atividade seja concluída, é de se esperar que não se utilize transações atômicas para tratar as possíveis exceções de negócio. Em vez disso, provêm-se cenários alternativos a serem seguidos pelos participantes nos casos em que haja situações inesperadas durante a execução da atividade de negócio. É válido ressaltar, no entanto, que as subatividades de serviço que compõem a atividade de negócio podem ser controladas por transações atômicas, visto que são neste caso processamentos isolados (Rosen, Lublinsky, Smith, & Balcer, 2008). Não se pode confundir neste contexto exceções técnicas (como uma falha de processamento, que cabe reversão das ações) com exceções de negócio (como uma resposta negativa de um fornecedor, que não cabe um *rollback*, porém pode ser realizado um plano alternativo).

Assim como o controle das transações atômicas, o controle de uma atividade de negócio também é regido pelo coordenador. Há essencialmente dois protocolos que ditam como os participantes devem se comportar no contexto global da execução da atividade de negócio, que são:

- **BusinessAgreementWithParticipantCompletion:** permite que um participante notifique ao coordenador de que a sua parte da atividade de negócio foi concluída;
- **BusinessAgreementWithCoordinatorCompletion:** permite que um participante notifique ao coordenador de que ele não possui mais responsabilidades de processamento em tarefas da atividade de negócio.

Para fins de exemplo, considere a figura 26 que exemplifica uma atividade de negócio envolvendo uma etapa facultativa. Na imagem, o serviço inicial de ordem de compra, contido no sistema TLS, envia uma mensagem SOAP contendo os detalhes da ordem ao serviço de atendimento de pedidos, contido no sistema RailCo. Por sua vez, o segundo serviço possui um período de 48 horas para devolver uma resposta, visto que o fornecedor deverá verificar se o seu estoque poderá ou não atender à ordem de compra. Uma vez respondido positivamente à solicitação, caso a ordem de compra ultrapasse R\$ 100.000,00 haverá uma etapa em que o gerente do sistema TLS deverá aprovar ou recusar a solicitação.

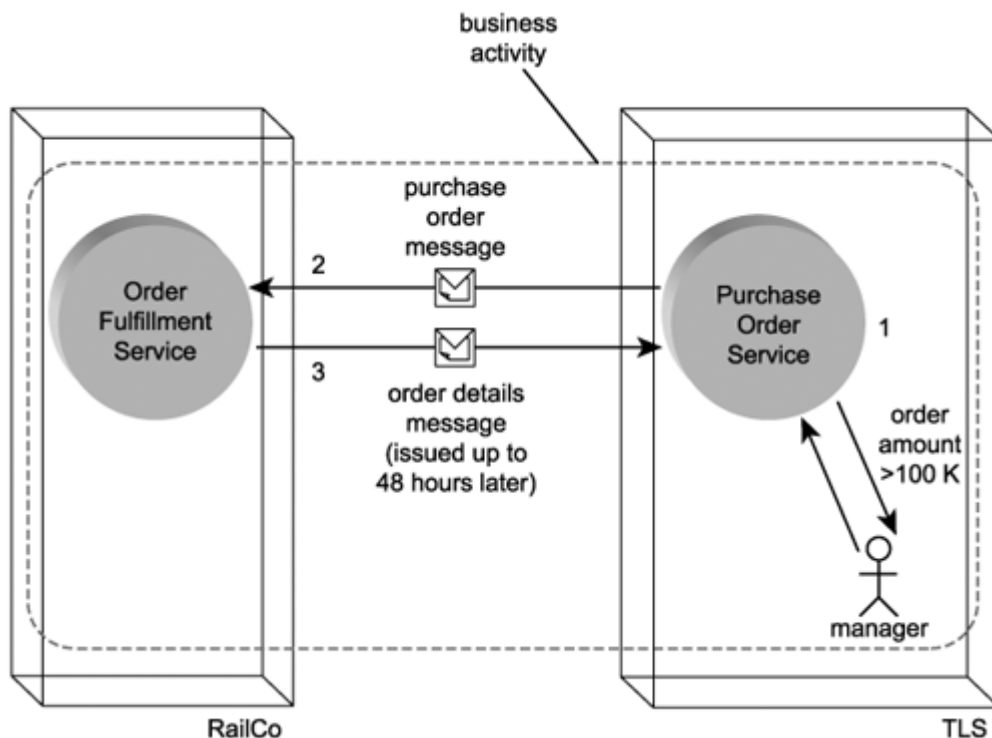


Figura 26 – Exemplo de atividade de negócio
 Fonte: Erl (2005, p. 198)

4.3.6 Orquestração

A orquestração dentro do conceito de *web services* descreve o fluxo de trabalho de como os serviços devem interagir, incluindo a lógica de negócio e a ordem das interações. A orquestração é realizada por um único controlador, similar ao conceito de *hub-and-spoke* nas implementações de EAI. O controlador central se responsabiliza por realizar a integração entre aplicações diferentes (Rosen, Lublinsky, Smith, & Balcer, 2008).

Para realizar composição dos serviços é utilizada uma linguagem de orquestração chamada WS-BEPL (*Web Service Business Process Execution Language*). Através da linguagem é possível construir *scripts* de orquestração, definindo regras e sequências de interação, que podem ser executados em tempo de execução por um serviço controlador da orquestração. A linguagem divide a construção de um *workflow* em atividades primitivas classificadas em básicas e estruturadas (Erl, 2005). A imagem 27 ilustra, por exemplo, um fluxo de processo de negócio que pode ser orquestrado por um controlador.

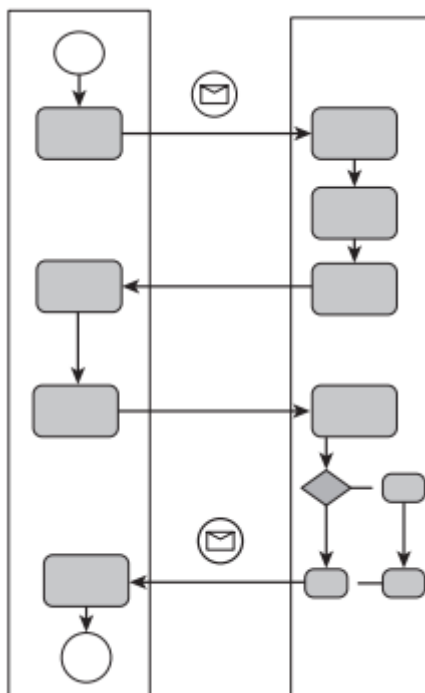


Figura 27 – Exemplo de fluxo regido pela orquestração
 Fonte: Rosen, Lublinsky, Smith, & Balcer (2008, p. 277)

É possível organizar as atividades básicas e estruturadas em ordem que satisfaçam a lógica de execução em sequências, fluxos ou *links*, descritos a seguir:

- **Sequências:** Alinha grupos de atividades relacionadas em uma lista que determina a execução em ordem sequencial, um seguido do outro. Sequências são comumente empregadas em situações em que um trecho da lógica da aplicação depende do resultado de alguma outra;
- **Fluxos:** Também contém grupos de atividades correlatas, mas introduz requisitos diferentes durante a execução das atividades. Trechos da lógica da aplicação podem executar em paralelo através de um fluxo, não sendo necessário que um conjunto de atividades aguarde o término das demais. Contudo, o fluxo não finaliza enquanto todas as atividades encapsuladas não concluírem seu processamento;
- **Links:** São úteis para estabelecer dependências formais entre atividades que constituem um fluxo. Antes de uma atividade se encerrar por completo, ela deve garantir que todos os requisitos estabelecidos nos *links* associados foram atendidos.

4.3.7 Coreografia

A coreografia descreve a sequência das mensagens entre os serviços, contudo no foco das trocas de mensagens e estados de comunicação públicos. Diferentemente da orquestração, que possui perspectiva puramente interna, a coreografia está intrinsicamente relacionada à perspectiva de ambientes externos à empresa, como de fornecedor, envolvendo, portanto canal B2B de comunicação (Rosen, Lublinsky, Smith, & Balcer, 2008), conforme representa a figura 28.

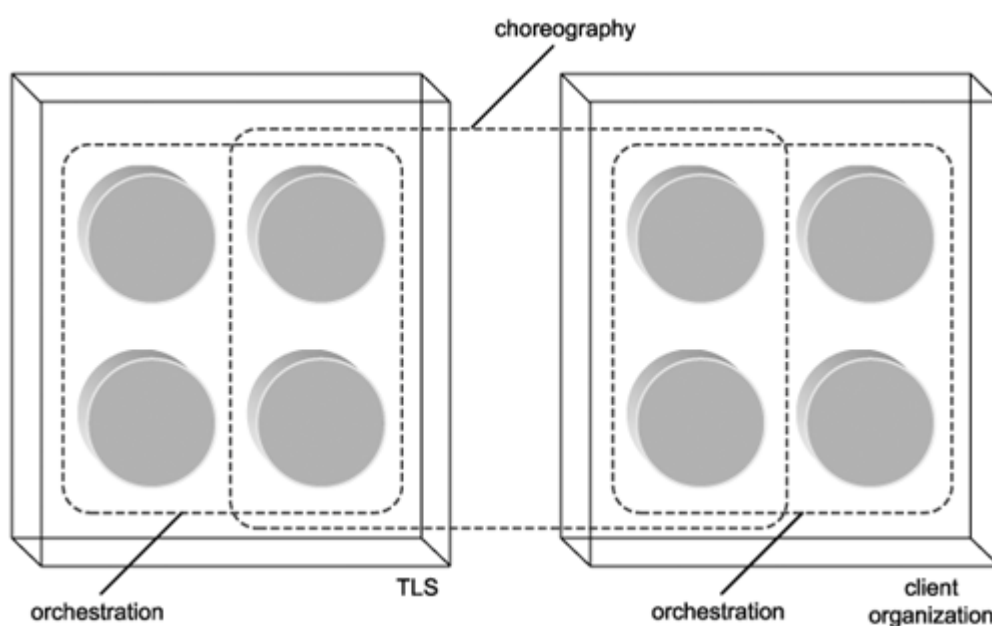


Figura 28 – Orquestração x coreografia
Fonte: Erl (2005, p. 212)

Assim como na orquestração, é necessária uma especificação para descrever a coreografia. Uma das existentes é a WS-CDL (*Web Services Choreography Description Language*), sendo possível através desta linguagem descrever e organizar a troca de informações entre os serviços contidos em diversas organizações, enfatizando a colaboração pública entre os ambientes (Erl, 2005). Entre os conceitos que envolvem a coreografia, destacam-se:

4.3.7.1 Colaboração

Diz respeito ao objetivo de estabelecer um tipo de colaboração organizada entre serviços que farão parte da atividade a ser atingida. A coreografia estabelece padrões

universais para uma interorganização entre os ambientes empresariais envolvidos, a fim de se concluir as tarefas de negócio.

4.3.7.2 *Papeis e participantes*

Um *web service* assume um de um conjunto predeterminado de papeis disponíveis. Isto estabelece os limites de responsabilidade que um serviço tem dentro de um contexto de uma atividade de negócio, delimitando o que o serviço pode ou não fazer. Os papeis podem ser incluídos na definição WSDL do *web service*, e todos os serviços serão agrupados em participantes.

4.3.7.3 *Relacionamentos e canais*

Um relacionamento consiste de necessariamente dois papeis, que basicamente pode ser conceituado como uma troca individual de mensagem entre dois papeis. Esta troca de mensagem pode, em conjunto com outras permutações de mensagens, formar uma determinada ação. Uma vez definido o relacionamento, é necessário que para ambos os serviços exista um canal de comunicação para que se estabeleça a conversação.

4.3.7.4 *Interações e unidades de trabalho*

Por fim, a lógica por trás de uma troca de mensagem é encapsulada em uma interação, que basicamente é o bloco fundamental para se construir coreografias em uma arquitetura SOA. As unidades de trabalho reúnem regras e restrições que devem ser respeitadas para que uma interação seja concluída com êxito.

5 ESTUDO DE CASO: INTEGRAÇÃO DE UM SISTEMA ALFANDEGÁRIO

Os conceitos de EAI, ESB e SOA apresentados e discutidos nos capítulos anteriores forneceram conhecimento e base teórica para se avaliar as propostas de integração de sistemas empregadas em uma empresa do setor portuário, bem como compreender a sugestão de solução descrita neste capítulo, envolvendo basicamente uma arquitetura sob os princípios SOA.

Inicialmente o capítulo abordará a introdução do estudo de caso, apresentando as características do terminal portuário, dos sistemas envolvidos e de alguns processos de negócio que constituem as principais atividades do terminal. Em seguida, será discutida a primeira solução de integração utilizada nos primeiros meses de funcionamento da empresa, avaliando seus pontos negativos. Posteriormente, será abordada a segunda solução de integração, utilizada até o momento em que este estudo foi realizado. Por fim, será apresentada a proposta de arquitetura de sistemas sob os conceitos de SOA e de boas práticas observadas ao longo deste trabalho.

5.1 SOBRE O TERMINAL PORTUÁRIO

O terminal portuário situa-se na região do porto de Santos, em São Paulo, e é um empreendimento concebido por dois grupos internacionais com sede na Holanda. O terminal multiuso, destinado a operar tanto contêineres quanto graneis líquidos, possui área projetada em 490 mil m² e capacidade de movimentação de 1,2 milhão de TEUs⁴ e 1,4 milhão de graneis líquidos. O empreendimento teve conclusão de suas principais obras em março de 2012 e início oficial de suas operações em agosto de 2013, marcado pela atracção do primeiro navio, o MSC Challenger, em seu cais.

A empresa pode ser subdividida em duas grandes partes: a primeira, referente ao terminal portuário, diz respeito às operações no cais, como atracções de navios, carregamento e descarregamento de contêineres e planejamento de operações das embarcações; a segunda, referente ao recinto alfandegado, diz respeito às operações de armazenamento dos contêineres, desconsolidação e averbação de documentos, registros de entrada e saída de cargas, envio de notificações aos órgãos aduaneiros, dentre outros. Apesar de mais complexo e exigir maior infraestrutura, quando uma empresa do ramo portuário

⁴ Abreviação de “Twenty foot Equivalent Unit”, unidade que representa a capacidade de carga de um contêiner marítimo normal de 20 pés de comprimento por 8 de largura e 8 de altura (Maritime Administration, 2008, p. 100).

contempla tanto atividades portuárias quanto de recinto alfandegado, ela torna-se um diferencial na região, além de ser vantajoso financeiramente.

5.1.1 Processo de Avaliação de Sistemas

Para avaliar se os sistemas iriam corresponder às expectativas operacionais estabelecidas e, também, para atestar se as rotinas estavam implementadas de acordo, o terminal portuário estabeleceu baterias de testes denominadas *Full Cycle* que verificavam os processos mais críticos e alguns cenários menos frequentes; os próprios usuários-chave de cada setor realizavam os testes e forneciam suas sugestões de melhorias. A bateria de testes verificava um ciclo inteiro de um contêiner, sendo ele de importação, exportação ou outra categoria, e a partir dos testes criava-se uma lista de correções ou modificações a serem realizadas nos sistemas.

A proposta do terminal portuário era a realização de quatro *Full Cycles* antes do *Go Live*, data marco para a implantação dos sistemas em produção e início das operações portuárias. No segundo *Go Live*, marcado para novembro, apenas alguns testes foram realizados, não caracterizando um *Full Cycle*, devido à paralelização das atividades do dia-a-dia com os testes; esta questão acabou culminando em problemas relatados nos tópicos seguintes. O rastreamento de bugs e controle das correções eram realizados puramente por e-mails disparados pelos usuários, ocasionando em alguns casos o esquecimento das aberturas de chamados.

5.1.2 Arquitetura de Sistemas

Os sistemas destacados a seguir referem-se àqueles com maior grau de relevância nas operações do terminal, fortemente relacionados às principais atividades desempenhadas ao objetivo-fim da empresa. A equipe de TI do terminal englobava, até o momento deste estudo, menos de 20 pessoas incluindo analistas de sistemas, desenvolvedores, analistas de infraestrutura, técnicos de suporte e coordenador. Alguns dos *softwares* listados a seguir foram desenvolvidos por terceiros, outros são internos e outros são de instituições do governo.

5.1.2.1 TOS

É um dos mais importantes sistemas do terminal, utilizado essencialmente pelos planejadores de pátio e de navio, atividades essenciais da empresa, além de operadores de

gates e prontificação de cargas de importação e exportação. O TOS também realiza troca eletrônica de dados com sistemas embarcados dos maquinários, como RTGs e *transtainers*, além de disponibilizar e receber arquivos EDI dos armadores de navios. A solução compreende dois módulos, chamados de TOPO (ambiente mais amigável destinado a usuários não relacionados às operações fins do terminal) e TOPX (destinado aos planejadores de pátio e de navio, especificamente). O módulo TOPO foi desenvolvido em C#/Visual Basic .NET e TOPX em C++.

5.1.2.2 *HOST*

Sistema desenvolvido por uma empresa nacional para ser usado exclusivamente em grandes armazéns, contudo modificado para as necessidades do terminal. Neste sistema foram criadas novas telas e ajustadas funcionalidades existentes para os setores de controle aduaneiro e exportação. Ele também é o responsável por realizar integração com os sistemas MANTRA e DTE. O sistema possui integração também com o TOS. O produto tinha ambiente desktop e *web*. O nome HOST não retrata um acrônimo, mas sim apenas um nome batizado pelo setor de TI do terminal.

5.1.2.3 *BHS*

Sistema desenvolvido pela equipe de TI do terminal portuário em conjunto com consultores terceirizados para substituir a primeira versão do HOST. Esta versão possuía apenas o ambiente desktop e foi desenvolvido em C#. A camada de integração com o TOS foi implementada em serviços Windows desenvolvidos em C#. A sigla não é divulgada por possuir o nome do terminal, não citado neste trabalho.

5.1.2.4 *TAS*

Plataforma *web* B2C e B2B desenvolvida pelo terminal portuário, fornecendo funcionalidades como agendamento de retirada e depósito de contêineres, consulta de atracções previstas no cais, solicitação de posicionamento de contêineres para retirada de amostras ou fotografias. É um sistema crítico por ser o portal com os clientes do terminal, e que deve estar disponível 24 horas por dia e 7 dias da semana, assim como o TOS e HOST. Ele possui integração com os sistemas HOST e TOS. O sistema foi desenvolvido em C# e é hospedado em servidor local.

5.1.2.5 *BBS*

Refere-se ao sistema financeiro do terminal portuário. Ele utiliza dados operacionais como tempo das atracções, quantidade de operações realizadas, quantidade de contêineres

envolvidos, tipos das cargas (se são refrigeradas, explosivas, perigosas etc.), movimentação de contêineres e diversos outros fatores utilizados nos cálculos financeiros. O sistema BBS possui integração no nível de banco de dados com o SAP, com o TOS e HOST. É uma solução *desktop* em C# com *Windows Forms*. Igualmente ao BHS, o nome deste sistema também não é detalhado em virtude do nome do terminal portuário.

5.1.2.6 SISCOMEX

SISCOMEX é um sistema do governo do Brasil para o controle do comércio exterior através de um fluxo único e automatizado de informações. Nele é possível acompanhar a saída ou entrada de mercadorias no país visto que os diferentes órgãos governamentais podem controlar e interferir no processamento de operações. Através dele é possível realizar o registro de exportação, importação e solicitação de despacho. O seu acesso é realizado através de um *browser*.

5.1.2.7 MANTRA

Sistema desenvolvido pelo governo federal brasileiro para controlar a entrada, movimentação e trânsito de cargas precedentes do exterior. Todo recinto alfandegado deve utilizar este sistema para informar quais cargas estão depositadas em seus respectivos pátios ou armazéns. A única integração formal possível com esta aplicação é através do envio de arquivos de texto, e o seu acesso é por intermédio de um emulador de terminal 3287 feito em Java Applet.

5.1.2.8 DTE

Este sistema é mantido pela ABTRA (Associação Brasileira de Terminais e Recintos Alfandegados) e é utilizado pelos terminais portuários e recintos alfandegados da baixada santista para informarem à Alfândega da Receita Federal do Porto de Santos a transferência de contêineres e cargas de importação que transitam pelo Porto de Santos e pelos recintos alfandegados da região. A integração com este sistema ocorre por intermédio do envio de arquivos de texto com estrutura particular da DTE, além de fornecer um *website* para acompanhamento do envio e aceitação/recusa destes arquivos de texto, também conhecidos como pacotes.

5.1.2.9 PREVINE

Produto customizado para o terminal portuário com o propósito de disponibilizar informações sobre o trânsito de veículos, contêineres, cargas e pessoas no recinto na forma de consultas através de um *website* seguro identificado por um certificado digital para agentes

fiscais da alfândega. O sistema deve estar sempre disponível para acesso pelas pessoas autorizadas do governo. PREVINE trata-se apenas de um nome para o sistema, assim como HOST.

5.1.2.10 ADE 02

Portal *web* desenvolvido em Java e hospedado em um Apache Tomcat para substituir o Previne. Além de se eliminar a dependência do sistema anterior, este novo precisou ser criado após as reformulações no banco de dados e na arquitetura de sistemas proposta pela segunda tentativa de integração entre os *softwares*.

5.1.2.11 OCR SYSTEM

Sistema desenvolvido por terceiros para identificar caracteres através de imagens, utilizado nos *gates* de entrada e saída do recinto alfandegado. O *software* depende da utilização de câmeras instaladas próximas às cabines, que capturam imagens das placas dos caminhões e de seus reboques, bem como a numeração dos contêineres. É um processo para automatizar a identificação dos veículos e cargas que entram e saem do terminal. O sistema possui integração com o TOS e HOST.

5.1.3 Processos de Negócio

Dentre os processos de negócio do terminal, encontram-se dois fluxos importantes e mais constantes, que são o processo de importação e o de exportação de cargas. Ambos são complexos, em especial o de importação, contudo, para fins práticos, serão abordados apenas os respectivos fluxos normais (sem cenários alternativos) enfatizando a relação entre os setores envolvidos nos dois grandes processos.

5.1.3.1 Processo de Importação

O processo de importação do terminal portuário inclui diversos cenários alternativos e, dentro os existentes na empresa, é o mais complexo exigindo, portanto, sincronia entre os sistemas para que o processo funcione adequadamente e com o menor índice de erros possível. A figura 29 ilustra o fluxo e a relação entre os setores, em um cenário simplificado.

O início do processo dá-se pela confirmação e formalização da atracação no cais do terminal, procedimento que envolve a Marinha do Brasil, a CODESP, o armador do navio e o terminal portuário em si. A confirmação da atracação dá-se pelo comum acordo entre as

partes envolvidas, e o sistema DTE envia um arquivo identificado por evento de código 12 ao terminal portuário contendo informações da atracação. A CODESP disponibiliza uma numeração que identifica a viagem do navio conhecida como RAP. A Marinha regulariza a movimentação do navio em mares brasileiros através de um documento chamado CE (Conhecimento Eletrônico) Mercante.

Após confirmar, dois processos iniciam em paralelo: o primeiro envolve o envio periódico de arquivo BAPLIE via EDI do armador ao terminal, contendo a relação de cargas que estão contidas no navio, bem como seus posicionamentos; o segundo diz respeito em às cargas manifestadas pelos importadores no SISCOMEX. O terminal tem a obrigação de permitir desembarcar apenas as cargas que estão manifestadas no sistema do governo federal brasileiro e, até que o navio atraque no cais, a comparação da relação das cargas do arquivo Manifesto com o Baplie é constantemente realizada. Este processo de comparação pode ser automatizado através da leitura dos dois arquivos.

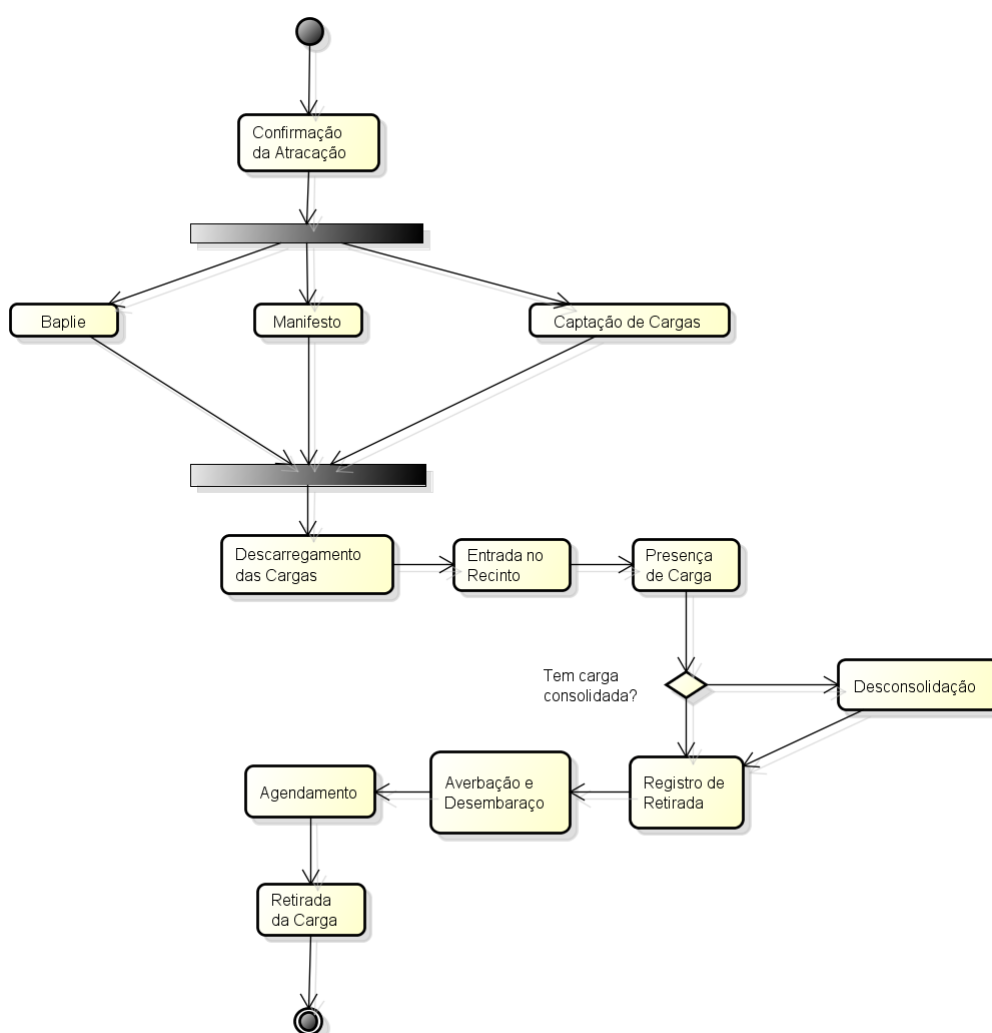


Figura 29 – Processo simplificado de importação de cargas

Paralelo ao processo descrito anteriormente, o setor de captação de cargas negocia com os clientes para que as suas cargas sejam depositadas no recinto; embora o navio atraque no cais do terminal, as cargas não captadas deverão ser redestinadas após o desembarque aos recintos que conseguirem tal feito. No processo de captação, quanto maior a agilidade sistêmica, melhores poderão ser os resultados, pois a relação entre o lucro da operação do navio com a quantidade de cargas captadas é diretamente proporcional. A cada lote captado o recinto recebe uma guia, enviado pelo sistema DTE; o documento que descreve um lote, que pode conter de um a vários contêineres, é o BL (Bill Of Lading).

Com a chegada do navio no cais, inicia-se a operação de descarga de contêineres. Conforme os contêineres são descarregados, o recinto deve, para cada um que foi captado, notificar ao sistema DTE sobre a sua entrada através de um arquivo conhecido como evento 14. Quando todos os contêineres pertencentes a um BL darem entrada ao recinto, a presença de carga deve ser informada no sistema MANTRA (que pode ser realizado manualmente ou através de um arquivo conhecido como evento SIS).

Ao realizar a presença de carga do lote no recinto alfandegado, o arquivo manifesto disponibilizado no início do processo de importação é atualizado informando quais são as cargas que estão consolidadas. Um BL consolidado pode agrupar, por exemplo, um contêiner com cargas de diferentes clientes e a desconsolidação neste caso envolve dois pontos: primeiramente, no nível documental, desmembrar o BL original em BLs *coloaders*, para que a retirada do contêiner possa ser realizada; a segunda, não praticada por este terminal portuário, envolve a abertura do contêiner para que as cargas sejam desovadas e possam ser retiradas pelos clientes. A desconsolidação documental pode ser realizada automaticamente através da leitura do arquivo manifesto atualizado.

Apenas após a presença de carga gerada, o importador pode realizar o procedimento de registro de retirada do contêiner no SISCOMEX, que basicamente envolve o registro de um documento que descreva a relação de cargas contidas no contêiner e pagamento de taxas e impostos cobrados pelo governo. Após o registro do documento, o recinto alfandegado deverá averbar o documento, ou seja, validar se as informações contidas estão de acordo, bem como desembaraçar, que envolve verificar se a carga precisará ser inspecionada ou não por órgãos anuentes do governo e, em caso positivo, acompanhar as inspeções necessárias.

Concluído o processo burocrático do registro documental da retirada da carga, o importador poderá realizar o agendamento da visita dos seus caminhões ao recinto para que o(s) contêiner(es) possam ser retirados. Ao chegar aos *gates*, o operador validará a entrada do caminhão e motorista ao recinto através da verificação dos dados provenientes do veículo e do

motorista com o agendamento realizado (a leitura da placa do caminhão pode ser realizada automaticamente pelo sistema OCR ou manualmente através da digitação da numeração pelo operador do *gate*). Validada a entrada, o caminhão entra no recinto, retira o contêiner e sai pelo *gate* de saída do recinto finalizando o processo; o recinto deverá notificar a saída do contêiner ao sistema da DTE.

5.1.3.2 Processo de Exportação

Em comparação com o processo de importação, o de exportação é mais simples principalmente em virtude do fluxo de documentos envolvidos para liberar a retirada do contêiner. A figura 30 exemplifica simplificada o processo de exportação de cargas praticado pelo terminal portuário.

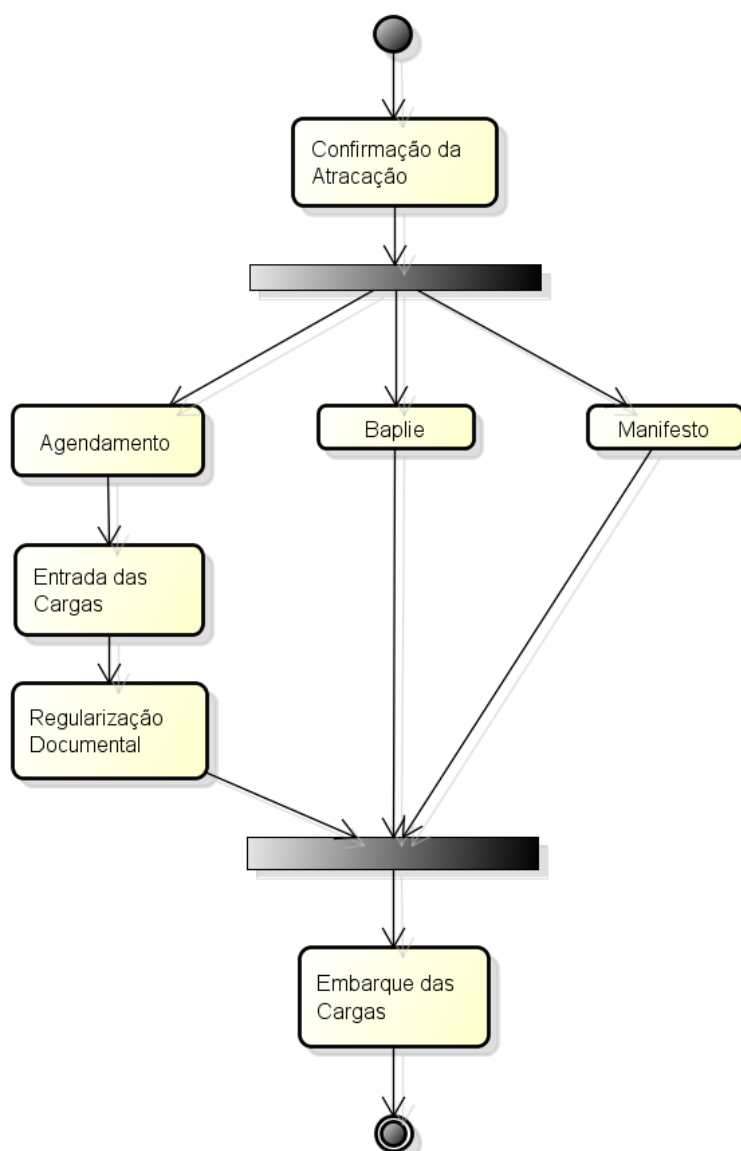


Figura 30 – Processo simplificado de exportação de cargas

Similar ao processo de importação, também o de exportação inicia com a confirmação da atracação do navio no cais. Uma vez confirmada, os procedimentos posteriores dão início. As etapas do BAPLIE e do manifesto são iguais ao de importação, ou seja, também são conferidos para averiguar se a relação de cargas que estão previstas para embarcar no navio é a mesma das cargas que foram manifestadas para serem exportadas.

Diferentemente do processo de importação, não há captação de cargas praticada da mesma forma. No caso da exportação, os acordos com os clientes são tratados diretamente entre o setor comercial e os clientes, não sendo necessário um setor como no caso de importação, pois neste caso não há utilização de guias enviadas pelo sistema DTE. Assim que a atracação é confirmada, é liberado aos clientes o agendamento dos veículos que trarão os contêineres a serem embarcados e despachados.

No dia e hora agendada, o caminhão contendo o contêiner é verificado no *gate* de entrada pelo operador e realiza os mesmos procedimentos no caso da importação, com exceção de uma verificação a mais, referente à conferência do contêiner. O conferente checa o número do agendamento, chamado de *visit code*, além da placa do caminhão e do contêiner, e dos dados pessoais do motorista. Após entrar, o caminhão deposita o contêiner no recinto e sai pelo *gate* de saída com o reboque vazio.

Após a entrada do contêiner, o setor de prontificação de cargas de exportação verifica a numeração dos lacres do contêiner com o documento de despacho entregue pelo cliente, bem como a checagem de peso, avarias e numeração. Caso todas as informações estejam de acordo, a documentação é regularizada e o contêiner confirmado para embarcar. Todos os contêineres confirmados para embarque são carregados no navio para que o mesmo, na hora estipulada, desatracue do cais e siga rumo ao porto de destino carregando os contêineres despachados.

5.2 PROPOSTA DA PRIMEIRA SOLUÇÃO DE INTEGRAÇÃO

A primeira proposta da arquitetura de sistemas foi elaborada pela gerência de TI do terminal portuário em conjunto com o engenheiro de *software* e de negócios da fornecedora do TOS. A ideia principal da arquitetura era a criação de duas ilhas de sistemas, uma voltada para o operacional com o TOS como principal sistema, e outra para o administrativo, com o HOST intermediando a integração entre as duas pontas. O objeto do terminal portuário era de poder facilmente substituir o TOS futuramente, por essa razão o HOST foi o único sistema

eleito para a integração; outro ponto pautado para a escolha da arquitetura era em relação ao custo da customização do TOS para integração, que deveria ser o menor possível.

A primeira proposta da arquitetura de sistemas elaborada para o terminal portuário envolvia uma infraestrutura contendo máquinas virtuais com elevados recursos computacionais – exatamente 8 máquinas virtuais Windows com 8 GB de memória RAM disponível e quatro núcleos de processamento cada uma. O HOST, nesta abordagem, seria o grande *middleware* da arquitetura, recebendo e enviando dados ao TOS e fornecendo informações ao TAS, PREVINE, DTE e MANTRA. O banco de dados dos sistemas era o Oracle na versão 11 e a plataforma do servidor era Windows. A integração entre os sistemas era realizada através do banco de dados e arquivos de texto.

O terminal portuário, em conjunto com a fornecedora do TOS, criou o conceito de mensagem. Cada mensagem representava um acontecimento no terminal, como por exemplo, um carregamento de contêiner, uma entrada de caminhão pelo *gate* ou um movimento de contêiner na quadra, cada qual identificado por um número. Ao todo eram 38 mensagens, e cada uma tinha uma estrutura de campos que identificavam o seu conteúdo, como número de contêiner, data e hora da operação, número de viagem do navio, peso da carga e assim por diante. Em resumo, toda vez que determinado evento ocorresse no terminal, uma mensagem deveria ser enviada de um sistema para o outro, dependendo de quem fosse a responsabilidade da geração da mensagem: do TOS para o HOST ou vice-versa.

As mensagens idealizadas na primeira proposta de arquitetura eram, basicamente, registros gravados em uma tabela no banco de dados, compartilhada entre os dois sistemas. O TOS utilizava de *threads*, uma para cada tipo de mensagem a receber, e cada uma delas realizava sucessivas consultas ao banco de dados para checar se haviam novas mensagens registradas, dentro de um intervalo de tempo de dois segundos. Uma lógica similar era realizada no sistema HOST, contudo, com acréscimo de inteligência artificial para realizar o tratamento das mensagens.

A integração entre TAS e HOST era de forma ainda mais direta. Quando um sistema precisava fornecer dados ao outro, este apenas realizava a gravação dos dados nas tabelas envolvidas através da chamada de *stored procedures*. A integração com os sistemas DTE e MANTRA eram realizadas através de arquivos de texto gerados em uma pasta da rede; um *middleware* terceirizado, dentro de um intervalo de tempo parametrizado, verificava a existência de novos arquivos no diretório e, em caso positivo, realizava o envio via FTP à DTE ou ao MANTRA. A responsabilidade em criar ou processar os arquivos de resposta da

DTE e MANTRA era exclusiva do HOST. A figura 31 ilustra o fluxo de comunicação entre os sistemas:

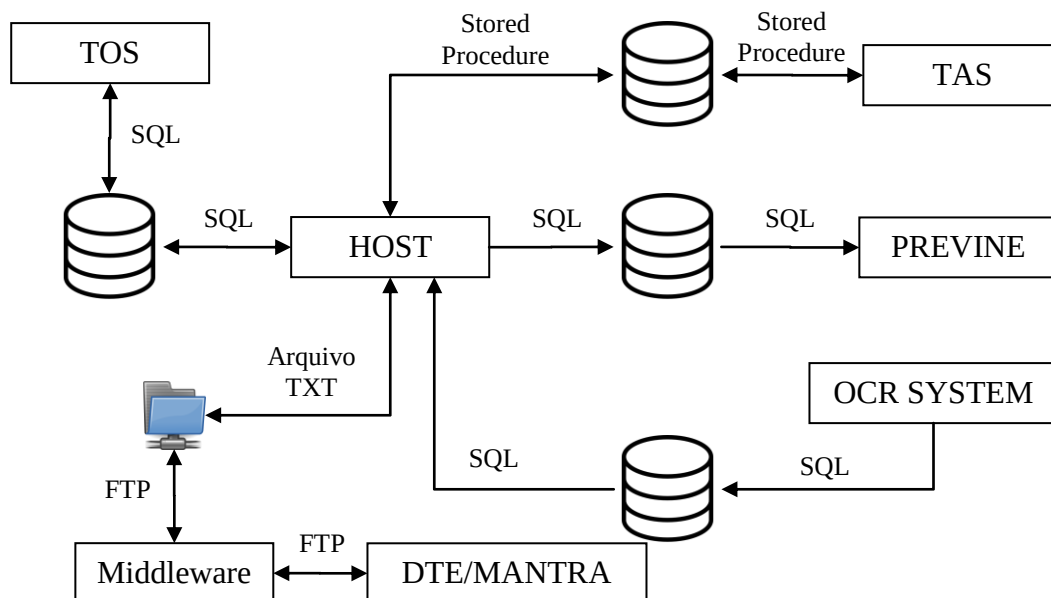


Figura 31 – Primeira proposta de integração para o terminal portuário

Por fim, o HOST realizava integração com o sistema PREVINE através de operações DML diretamente em seu banco de dados, e consumia dados gerados pelo sistema de OCR através de *views* do banco de dados que consultavam as informações das leituras das placas dos veículos e contêineres. As tabelas do sistema PREVINE possuíam estruturas e informações parecidas com as dos sistemas TAS, HOST e TOS.

5.2.1 Desempenho da Arquitetura

O sistema HOST foi modificado para as necessidades do terminal portuário, contudo originalmente ele é um produto de prateleira destinado a armazéns de grande porte. Desta forma, o seu banco de dados continha cerca de 100 tabelas e algumas destas foram ajustadas para suportar os requisitos do terminal resultando, por exemplo, em algumas tabelas com cerca de 100 colunas. Os usuários se queixavam da falta de conhecimento em porto por parte da empresa, resultando em entendimento incompleto ou equivocado das necessidades dos usuários, retrabalho e requisitos mal implementados.

A geração e o processamento das mensagens por parte do HOST, em momentos de pico das operações, exigiam cerca de 2 horas em média para serem concluídas. Este elevado

tempo foi um dos principais fatores para que as operações apresentassem indicadores de desempenho abaixo do esperado. Atrelado à lentidão computacional, a geração de dados por parte do HOST apresentavam inconsistências, devido à incorreta implementação de *triggers* e funcionalidades no sistema. Para minimizar os erros nos dados, intervenções diretas ao banco de dados de produção foram realizadas pela equipe de analistas de sistemas do terminal portuário.

O consumo de recursos das máquinas virtuais disponíveis era quase total, resultando em alguns casos a necessidade do reinício do servidor em virtude de travamentos ocasionados por *threads* e pelo mecanismo de inteligência artificial encontrado no sistema HOST. Devido à geração errônea dos dados, o envio dos pacotes de informações para o sistema DTE e MANTRA foi suspenso e o sistema TAS retirado do ar. O sistema HOST também fazia a leitura dos dados produzidos pelo OCR, contudo a rotina de pesquisa e validação dos dados estava incorretamente implementada, falhando em aproximadamente 95% dos casos, conforme destacado na tabela 3.

Tabela 3 – Resultados de desempenho dos sistemas do terminal portuário

Resultado	Esperado (média)	Real (média)
Índice de falha da leitura do sistema OCR	10%	95%
Tempo de processamento de mensagens críticas do TOS	1 min	2 h
Tempo de processamento de arquivos MANIFESTO	5 min	30 min
Delay para disparo da presença de carga (para o sistema MANTRA)	imediato	20h

A equipe de TI realizava plantão todos os dias, com revezamentos entre os analistas, para atender aos casos mais críticos, onde normalmente a resolução envolvia intervenção nas tabelas do banco de dados. Muitas horas extras foram pagas aos colaboradores e, atrelado a este custo, muitas solicitações de mudanças de requisitos foram criadas para tentar corrigir os pontos mais críticos, ultrapassando o orçamento estipulado para o centro de custo de TI.

5.2.2 Resultados das Operações Portuárias

Após o terminal portuário dar início às suas atividades em agosto de 2013, as operações, cerca de três semanas depois, estavam seriamente comprometidas pela ineficiência da arquitetura idealizada. A quantidade de movimentações de contêineres, de atracações de navios, de cargas captadas pelo setor comercial, de clientes conquistados e da agilidade das

operações estavam aquém do planejado no início da construção do terminal portuário. A tabela 4 ilustra a comparação entre alguns resultados esperados e os atingidos até fim de outubro de 2013.

Tabela 4 – Comparação de alguns resultados esperados x reais do terminal portuário

Resultado	Esperado (média)	Real (média)
Caminhões por hora em cada <i>gate</i> de entrada	20	3
Tempo de permanência de caminhões no pátio (minutos)	5	40
Navios por dia	12	2
Contêineres captados por navio	300	70
Faturamento mensal (milhões de R\$)	80	20

Dentre os resultados ilustrados acima, um dos mais expressivos e importantes é em relação ao faturamento mensal, que possui como causa todos os demais resultados adversos. A lentidão de algumas operações, como a entrada de caminhões no recinto, bem como a estadia dos veículos para carregar ou descarregar contêineres, fizeram com que a movimentação das cargas fosse baixíssima no período observado. Atrelado a isso, a reduzida quantidade de navios atracados também tiveram impacto negativo no faturamento mensal, fazendo com que os indicadores do terminal portuário atingissem valores alarmantes.

As longas permanências no pátio tornavam, durante alguns períodos do dia de trabalho, o recinto demasiadamente cheio, dificultando o trânsito de outros veículos e dos maquinários. A espera nos *gates* de entrada consumiam tempo mais do que o necessário, causando filas de caminhões para entrar no terminal portuário e irritação por parte dos motoristas, resultando em discussões e brigas com os operadores de *gates*, necessitando em algumas raras vezes a intervenção da polícia e seguranças para amenizar o ambiente e garantir a integridade física dos trabalhadores. As filas de caminhões chegaram a interferir no trânsito da avenida que dava acesso ao terminal.

Os setores de controle aduaneiro e prontificação de cargas recorreram muitas vezes ao uso de planilhas e criação/edição de arquivos de texto (a serem enviados para DTE) manualmente através de editores de texto, resultando em trabalhos exaustivos e suscetíveis a erros. Como consequência, o estresse e a má organização resultaram em pedidos de demissão por alguns colaboradores.

Após as três primeiras semanas de atividade apresentando resultados ruins e abaixo do esperado, a alta diretoria em conjunto com o setor de TI da empresa decidiu que uma

reformulação sistêmica imediata era necessária para que a realidade pudesse ser revertida e o investimento no terminal não fosse perdido. Após algumas reuniões, ficou decidido que o prazo máximo para a substituição dos sistemas que estavam interferindo negativamente nos processos fosse até dezembro do mesmo ano, com forte possibilidade de suspender as atividades do terminal caso o prazo não fosse atendido.

5.3 PROPOSTA DA SEGUNDA SOLUÇÃO DE INTEGRAÇÃO

Após aproximadamente três semanas do início das operações do terminal e dos inexpressivos resultados obtidos no período, foi estabelecido que uma reformulação em especial no sistema HOST fosse necessária, agregando ainda mudanças ao sistema TAS como correções e melhorias em geral, além de ajustes no sistema TOS. Infelizmente, o tempo estipulado para uma nova implantação foi demasiado curto, limitando-se em aproximadamente quatro meses.

O novo HOST, batizado de BHS, seria composto por três grandes partes: um conjunto de serviços Windows responsáveis por gerar e processar as mensagens enviadas pelo TOS; o sistema *desktop* utilizado pelos setores de prontificação de cargas de importação e exportação, controle aduaneiro e operadores de *gate*; API contendo rotinas para gerar os arquivos texto a serem enviados sob forma de pacote para a DTE, além de um serviço Windows para ler os arquivos texto de resposta e processá-las e outro para gerar a presença de carga. A imagem 32 ilustra um desenho técnico das dependências entre estes artefatos, incluindo o TAS que também tinha autonomia para disparar mensagens através do uso da API.

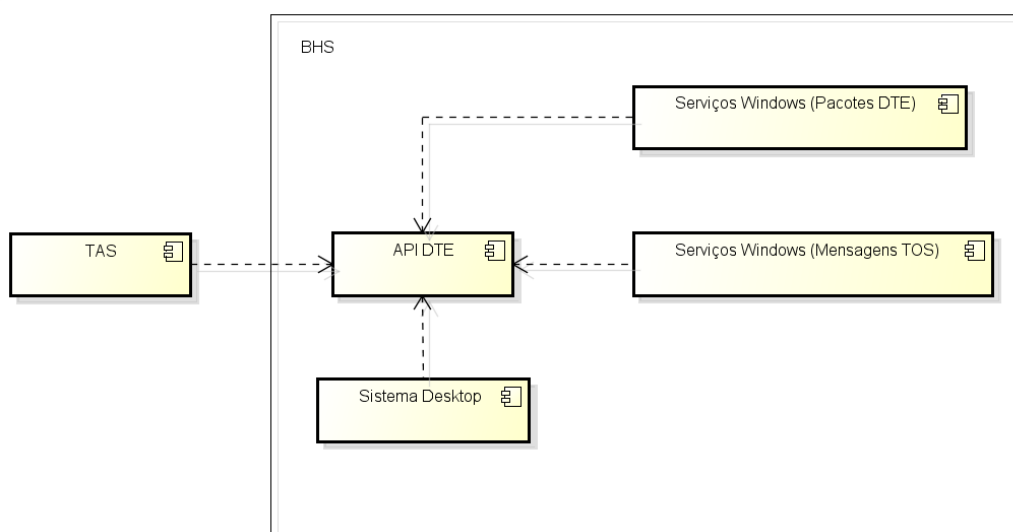


Figura 32 – Dependências envolvendo a API DTE

A motivação para esta decisão arquitetural foi de desmembrar as rotinas da DTE em uma API que pudesse ser utilizada a princípio por qualquer sistema da arquitetura de sistemas do terminal portuário em C#, como o próprio BHS ou o TAS. O tempo do projeto também influenciou na decisão, pois a equipe pôde ser subdivida de tal forma que integrantes pudessem desenvolver a API sem influenciar no desenvolvido do novo HOST e vice-versa.

O sistema PREVINE também foi removido, visto que também era um produto da mesma empresa fornecedora do *software* customizado para ser o HOST do terminal portuário, e foi substituído por um sistema *web* nomeado como ADE 02 desenvolvido em Java e implantando em um servidor Apache Tomcat. Em vez de ter uma base de dados própria, este sistema utilizava *views* que consultavam tabelas já existentes nos sistemas BHS, TOS e TAS, reduzindo a incidência de inconsistência dos dados.

O sistema *desktop* BHS era distribuído via *ClickOnce Deployment*, e o seu controle de versão era manual. Os serviços Windows tinham, cada um, uma *thread* que verificava se uma nova mensagem tinha sido criada pelo TOS, e nos casos em que a direção da mensagem era oposta, gerava uma nova mensagem ao TOS. O serviço responsável pelo disparo da presença de carga também consistia em uma *thread* que consultava, sob um intervalo parametrizável, uma *view* que retornava os BLs que precisavam do disparo.

A arquitetura de sistemas, de forma geral, obedeceu aos padrões estipulados na primeira proposta, com a diferença de substituir dois sistemas do conjunto por novos dois *softwares* com tecnologias e estruturas mais apropriadas, além do uso da API como forma de integração entre os sistemas. Contudo, o conceito de mensagens estipulados pelo terminal permaneceu o mesmo, bem como o uso do *middleware* para o envio e recebimento de arquivos da DTE e de *stored procedures* para integração de alguns dados entre BHS e TAS.

5.3.1 Desempenho da Arquitetura

O início do desenvolvimento da nova solução caminhou conforme o planejado nas duas primeiras semanas, tendo suas primeiras dificuldades com a ausência dos analistas do terminal portuário no auxílio da nova implementação, decorrente dos problemas oriundos do sistema anterior ainda em produção. Consequentemente, o projeto abriu lacunas na implementação das funcionalidades, que não estavam sendo validadas por completo, abrindo margem para o aparecimento de problemas maiores ao fim do projeto.

O projeto de desenvolvimento do novo TAS teve problemas no dia da implantação do sistema, em decorrência de má configuração do servidor IIS. Em virtude a essa situação, o portal *web* foi publicado apenas 48 horas depois da reabertura do terminal portuário, e na estação de trabalho de um dos programadores utilizada para o desenvolvimento do novo sistema. Após cerca de três semanas a situação foi regularizada e o TAS enfim publicado corretamente no servidor.

O BHS, embora tivesse melhor desempenho frente ao antigo HOST conforme destacado na tabela 5, não estava completo sob o ponto de vista dos usuários, restando inúmeras funcionalidades relevantes. Diante desta circunstância, os requisitos eram implementados sob demanda e consequentemente, com a ausência de testes unitários e de integração, novas versões eram disponibilizadas com *bugs*. Não haviam testes automatizados para minimizar a incidência de erros e de verificar a integridade de funcionalidades que já estavam funcionando. Toda nova versão era diretamente distribuída para o ambiente de produção, com ausência de etapas como homologação.

Tabela 5 – Resultados de desempenho dos novos sistemas do terminal portuário

Resultado	Esperado (média)	Real (média)
Índice de falha da leitura do sistema OCR	10%	30%
Tempo de processamento de mensagens críticas do TOS	1 min	5 s
Tempo de processamento de arquivos MANIFESTO	5 min	6 min
<i>Delay</i> para disparo da presença de carga (para o sistema MANTRA)	imediatos	10 min

A API para os eventos da DTE reduziu a dependência do TAS para o novo HOST, contudo a solução como uma biblioteca para as aplicações aumentou a complexidade da implantação de novas versões. Pela falta de controle e pressa em publicar sempre novas distribuições corrigindo os erros e acrescentando novas funcionalidades, cada sistema possuía uma versão diferente da API, gerando inconsistência.

Apesar do desafio das operações portuárias, os sistemas apresentavam muitas falhas, em virtude das causas já destacadas anteriormente. Desde a implantação, em novembro, até meados de abril, haviam registrados, ao todo, um total de mais de 300 *bugs* incluindo os principais sistemas envolvidos. A pressão da alta diretoria para que o terminal realizasse atividades mais diferenciadas com o propósito de aumentar o faturamento mensal acelerou o desenvolvimento de novas funcionalidades em paralelo às correções, tornando um cenário interminavelmente instável.

5.3.2 Resultados das Operações Portuárias

Os primeiros dias após implantação no novo HOST, do ADE 02, do novo TAS e de melhorias no TOS não foram exatamente conforme planejado. Em virtude do pouco tempo de projeto, o desenvolvimento foi acelerado, os testes foram poucos e como consequência as situações não previstas durante a fase final do projeto começaram a extrapolar em falhas dos sistemas, afetando algumas operações.

As operações do terminal, de uma forma geral, apresentaram melhorias significativas, desde o início, apesar dos erros sistêmicos. A produtividade aumentou, os fluxos de entrada e saída de caminhões eram mais frequentes e menos demorados, bem como as atracções de navios tornou-se mais constante. As movimentações de contêineres também aumentaram, em consequência. Pós implantação, durante aproximadamente o mês de novembro e início de dezembro, os *gates* tiveram poucos problemas relativos aos excessos de filas, e a quantidade de navios aumentou, conforme ilustra a tabela 6, apresentando dados referentes a dois meses com a nova implantação.

Tabela 6 – Comparação de alguns resultados após implantação do novo HOST

Resultado	Esperado (média)	Real (média)
Caminhões por hora em cada <i>gate</i> de entrada	20	18
Tempo de permanência de caminhões no pátio (minutos)	5	7
Navios por dia	12	8
Contêineres captados por navio	300	200
Faturamento mensal (milhões de R\$)	80	60

Apesar da melhora em todos os números, o terminal portuário permaneceu por alguns meses apresentando problemas em suas operações. Se antes o grande problema era relacionado ao fato de que sequer as operações triviais estavam sendo realizadas, agora, com as atividades operacionais fluindo, as falhas sistêmicas relativas às atividades alfandegadas ficaram expostas.

A primeira era o fato de que, apesar do baixo número de operações realizadas antes da implantação do novo HOST, muitos pacotes da DTE deveriam ser enviados, visto que manualmente não é possível enviar todas as informações em tempo (em situações normais, que não há desvios no cenário da importação de carga, um único contêiner demanda cerca de oito arquivos texto a serem enviados à DTE). Para isso, os pacotes começaram a ser enviados,

contudo em virtude da falta de integridade dos dados gerados anteriormente, muitos foram recusados e o esforço para corrigi-los passou a aumentar gradativamente, atrelado ao crescente número de operações proporcionado pela nova implantação e que também apresentavam falhas na geração dos arquivos. O envio incorreto dos pacotes também prejudicou o setor de captação de cargas, freando um aumento mais agudo do percentual de contêineres captados.

A segunda falha no controle aduaneiro era relativa à geração da presença de carga dos BLs. A geração deste evento para a receita federal tem tamanha importância, que quando uma presença de carga é gerada equivocadamente, o recinto é sujeito a multas e penalizações; se os erros forem constantes, as penalizações podem chegar à suspensão das atividades do terminal por até um ano através de ordem judicial. Durante os primeiros meses, o terminal gerou duas presenças de carga equivocadamente, recebendo notificação de fiscais da alfândega de que o terceiro erro causaria uma penalização.

Dentre os problemas apresentados, o mais crítico era relativo ao processamento incorreto das informações na validação da entrada e saída dos caminhões. Em uma madrugada, dois contêineres foram retirados equivocadamente do terminal, visto que eles estavam bloqueados para inspeção de agentes da alfândega e não podiam ser retirados até que a inspeção fosse concluída e os contêineres liberados. O problema não tomou proporções maiores, pois os contêineres foram devolvidos a tempo para o recinto e o caso foi minimizado pela diretoria.

5.4 PROPOSTA DE SOLUÇÃO COM ARQUITETURA SOA

Comparando as duas propostas de solução, é possível constatar que um ponto em comum que resultou no fracasso dos projetos de integração é na escolha equivocada da arquitetura empregada no terminal portuário. Tanto na primeira quanto na segunda, é possível observar a alta concentração de responsabilidades no HOST, que além de *middleware* entre as duas grandes ilhas de sistemas, era utilizado como um sistema *desktop* pelos usuários internos do terminal para desempenhar atividades de leitura de arquivos Manifesto, cadastros e desconsolidação de BLs, controle de eventos DTE, reprocessamento de mensagens etc., aumentando sua complexidade e sobrecarregando suas responsabilidades perante a arquitetura.

Como consequência a este acúmulo de papéis, e sem o uso de testes automatizados, novas atualizações do sistema geravam impacto direto aos demais sistemas, como relatado

pela equipe de suporte em um dos *bugs*, em que o sistema TAS deixou de funcionar por conta de uma atualização da assinatura de um método. O uso de *stored procedures* torna os testes e a detecção de erros mais difíceis, a implementação mais limitada ao comparar PL/SQL com recursos de linguagens orientadas a objetos e aumenta a dependência do banco de dados. O uso do conceito de mensagens poderia ser preservado, contanto se de fato tivesse sido utilizado MOM para suprir tal condição, em vez de uma solução envolvendo uma única tabela que facilmente chegava aos milhões de registros, aumentando mais uma vez a dependência do banco de dados e a complexidade na solução.

Os sistemas TAS, TOS e BHS armazenavam dados similares em seus respectivos esquemas de tabelas no banco de dados, gerando redundância. Informações de contêineres, por exemplo, poderiam ser consultadas nos três sistemas, contudo em virtude de erros na integração, cada um poderia apresentar informações diferentes, desatualizadas e inconsistentes. A necessidade da integração por “mensagens” e *stored procedures* surgiu em grande parte em virtude da replicação de dados entre um sistema e outro. Assim que um contêiner era atualizado no TOS, por exemplo, este era informado para o BHS para que ele também mantivesse os seus dados atualizados, porém a redundância da informação implicou em muitos erros nos sistemas em virtude de falhas nos envios e processamentos das “mensagens”.

Apesar dos pontos negativos, a segunda arquitetura resolveu em boa parte o problema de desempenho encontrado anteriormente. A remodelagem do banco de dados, bem como a reescrita do sistema HOST impulsionou a melhora do processamento das transações. A substituição do sistema PREVINE pelo ADE 02 reduziu a complexidade de integração que, embora já fosse por SQL, existia um esquema de tabelas repetidas para o sistema, agora inexistente.

Os tópicos a seguir descrevem a proposta de solução para a arquitetura de sistemas do terminal portuário, sustentando pelo conceito de serviços discutido no capítulo anterior, e na integração de dados com o uso de *views* do Oracle. A sugestão de uso de serviços, com o uso de *views*, é para agregar à segunda solução uma arquitetura moderna que resolva os problemas criados ou agravados após implantação, visto que o problema de desempenho já foi solucionado com a remodelagem e reescrita dos sistemas principais. Por fim são pautados alguns pontos de melhoria no processo de desenvolvimento de *software*.

5.4.1 Utilização de Serviços

Um dos alicerces do conceito SOA é a abstração de serviços empresariais em elementos de *software* que os representem. No contexto do terminal portuário, há diversas atividades que podem ser compreendidas como serviços e, portanto, abstraídas para que os sistemas possam processá-las. A criação de um barramento de *web services* que seriam consumidos pelas aplicações, combinado com os pontos ajustados com a arquitetura anterior, unificaria e facilitaria as mudanças das regras de negócio, seria mais escalável, mais seguro e gerenciável.

Como ponto de partida, um dos primeiros pontos a se verificar é analisar as atividades existentes no terminal portuário e elencar quais podem se tornar serviços e quais devem ser tratados em conjunto para considerar um único serviço. Lembrando as características essenciais predominantes em uma arquitetura orientada a serviços adequada, deve-se idealizar a nova estrutura orientando-se ao negócio, além de considerar pontos como modularização, autonomia, baixo acoplamento e padronização. A figura 33 exemplifica a nova arquitetura com utilização de serviços:

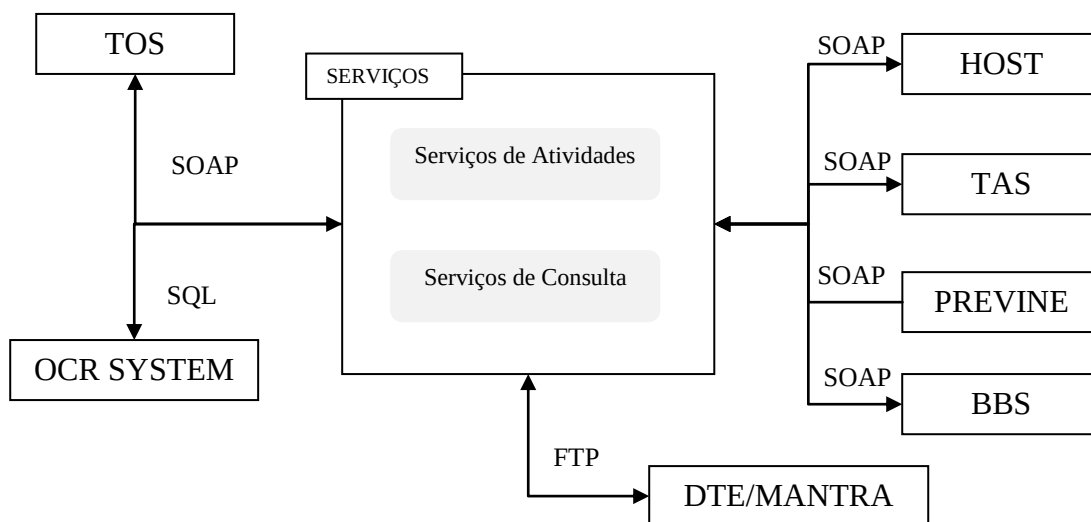


Figura 33 – Proposta de arquitetura com serviços

Os sistemas realizariam requisições SOAP com o barramento de serviços, sendo este constituído por *web services* que representassem atividades do terminal portuário e outros que representassem meramente consultas unificando os bancos de dados dos sistemas. A comunicação com DTE e MANTRA seria realizado diretamente via FTP (único protocolo disponível pela Alfândega e Receita Federal), dispensando o uso do *middleware* terceirizado,

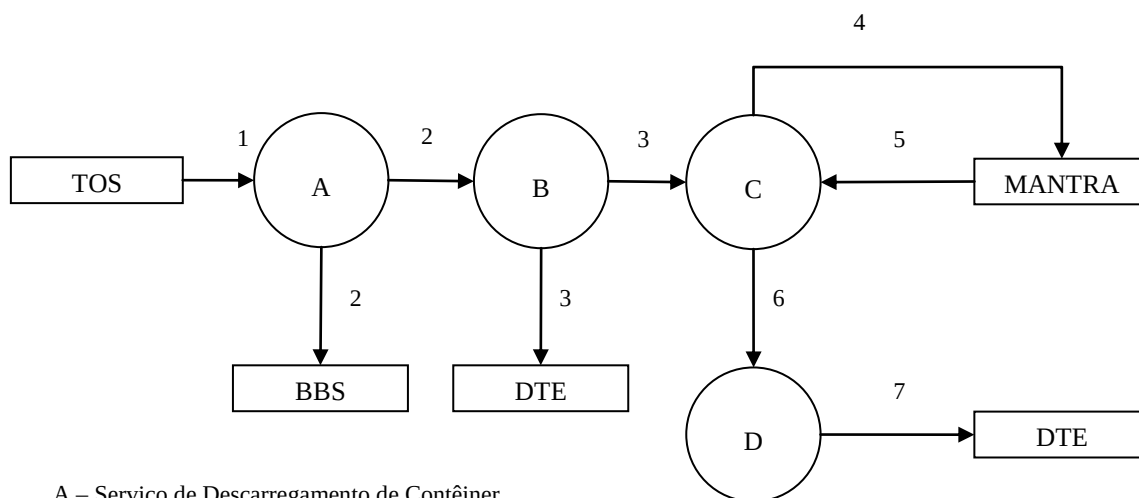
agilizando o tráfego de dados e reduzindo custo com suporte da aplicação. A lista abaixo exemplifica alguns serviços, representando atividades do terminal:

- Serviço de Descarregamento de Contêiner;
- Serviço de Entrada de Caminhão;
- Serviço de Saída de Contêiner do Recinto;
- Serviço de Presença de Carga;
- Serviço de Desconsolidação de BL;
- Serviço de Averbação de BL;
- Serviço de Atracação de Navio.

E cada um destes serviços apresentaria uma interface de utilização bem definida, descrevendo operações que poderiam ser executadas pelos consumidores dos serviços. Tomando o primeiro como exemplo, Serviço de Descarregamento de Contêiner, alguns métodos poderiam ser:

- NotificarDescarregamentoContêinerCheio;
- NotificarDescarregamentoContêinerVazio;
- NotificarReversaoDescarregamentoContêiner;

É válido lembrar que esses serviços seriam disparados, por exemplo, pelo sistema que possui a responsabilidade de fato em sistematizar uma descarga de contêiner, por exemplo, que é o TOS. Contudo, esta atividade, assim como as demais, pode desencadear outros eventos, como uma cobrança a ser realizada pelo BBS, ou um pacote a ser disparado para a DTE, ou até mesmo alguma notificação ao cliente. Os serviços aqui teriam a responsabilidade de garantir a ordem destes eventos, concentrando as regras dos fluxos envolvidos. A imagem 34, por exemplo, ilustra o processo de descarga de contêiner com a utilização de *web services*.



A – Serviço de Descarregamento de Contêiner
 B – Serviço de Entrada de Contêiner no Recinto
 C – Serviço de Presença de Carga
 D – Serviço de Desconsolidação de BL

Figura 34 – Processo de descarga de contêiner com *web services*

Os processos identificados pelos números são descritos a seguir:

1. TOS identifica que um descarregamento de um determinado contêiner terminou de ser concluído, e então ele notifica o serviço A através da chamada do método `NotificarDescarregamentoContêinerCheio`;
2. O serviço notifica ao sistema BBS que uma operação de descarga acabou de ser concluída e que precisa ser contabilizada para fins de faturamento. Caso o contêiner seja captado pelo terminal portuário, então o serviço B é invocado;
3. O serviço B dispara um pacote para a DTE, notificando a alfândega de que um contêiner entrou no recinto e que deve ser contabilizado como uma unidade no depósito do terminal portuário. Caso este contêiner seja o último do seu lote a entrar no recinto, então o serviço C é executado;
4. O terceiro *web service* assume a responsabilidade de controlar o disparo da presença de carga verificando, por exemplo, se determinado BL já não teve este evento disparado para o MANTRA manualmente por algum usuário via sistema BHS;
5. Quando o MANTRA responder positivamente ao evento do disparo da presença de carga, a data e hora do evento seria então confirmado no sistema e o lote

liberado para que o cliente desse entrada na documentação necessária para retirar seus contêineres do recinto;

6. O serviço D, o último deste processo, daria início ao processo de desconsolidação automática, recuperando as informações do arquivo Manifesto que poderia ser disponibilizado em alguma pasta de rede. Após a conclusão da desconsolidação, uma notificação seria enviada ao sistema da DTE.

Estes e outros processos do terminal portuário poderiam ser abstraídos em serviços, onde cada um teria sua autonomia e controle sobre as validações de suas regras de negócio. Além disto, segurança para permitir ou negar o uso de determinados serviços, além de, em conjunto com os conceitos de orquestração e transações atômicas, discutidos anteriormente, garantir o fluxo de execução dos serviços.

5.4.2 Utilização de Views

Os principais sistemas do terminal utilizam o SGBD Oracle como repositório dos dados, portanto, a partir desta premissa seria possível reutilizar estruturas de dados entre os sistemas para minimizar a criação de tabelas redundantes e armazenamento de dados repetidos, e encapsulá-las em serviços exclusivamente de consultas, minimizando problemas de integridade da informação entre os sistemas que são constantes com a arquitetura idealizada.

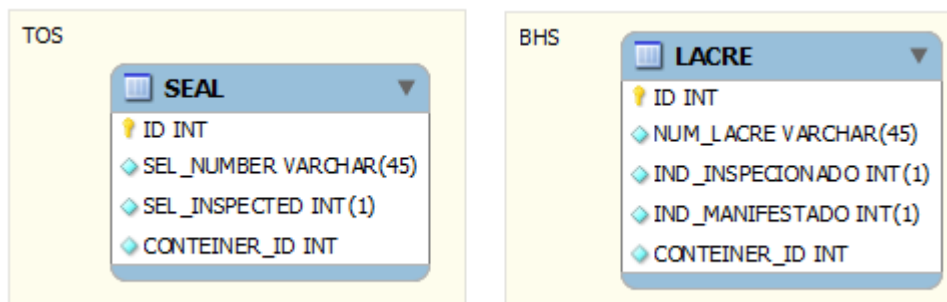
Frequentemente tanto o BHS, quanto BBS e TAS precisam consultar informações presentes nos demais sistemas, algumas vezes já combinando uniões entre tabelas, para a geração de relatórios operacionais, listagens diversas envolvendo contêineres, arquivos para alfândega e telas de consulta utilizadas tanto pelos setores do terminal quanto por agentes fiscais. Desta forma, as buscas ficariam concentradas no barramento de serviços.

Conforme mencionado anteriormente, com a arquitetura concebida, um dos problemas mais frequentes é o da falta de integridade das informações, muito em virtude das falhas das “mensagens” trocadas entre TOS e BHS. Como observado, o BHS é a fonte principal de informação dos demais sistemas, e falhas de integridade acarretam em problemas em efeito cascata nas outras aplicações. Para exemplificar, eis um cenário recorrente no terminal ao menos nos primeiros meses pós-implantação da segunda proposta de integração:

1. No fim do processo de importação de cargas, um caminhão estaciona no *gate* de saída do terminal portuário com o contêiner alocado em seu reboque para ser retirado do recinto;
2. Para validar as informações do motorista, do caminhão e do contêiner prestes a ser removido, a fim de checar se confere com o agendamento realizado pela transportadora, o sistema TOS inicia diálogo com o BHS;
3. Primeiramente, o sistema TOS dispara uma “mensagem” para o BHS solicitando que seja realizada a validação dos dados;
4. BHS identifica que uma nova “mensagem” foi criada no banco de dados e então realiza o seu processamento, validando os dados;
5. BHS identifica que os lacres do contêiner não foram vistoriados e nega a sua saída, informando que a vistoria deve ser realizada pelos funcionários do terminal;
6. Operador do *gate* checa então as informações no sistema TOS, e verifica que os lacres na verdade já foram vistoriados anteriormente. Por conta disto, o caso é anotado, e o motorista liberado para sair com o contêiner, ignorando o restante das validações e podendo causar futuros problemas;
7. Departamento da TI identifica posteriormente que uma mensagem enviada anteriormente não havia sido processada corretamente pelo BHS, ficando os lacres com um status incorreto se comparado com o TOS.

Apenas com a utilização de serviços, neste caso o problema poderia acontecer da mesma forma, pois a vistoria dos lacres é persistida em duas tabelas distintas, desnecessariamente. O serviço responsável por realizar as validações de saída dos contêineres, com a utilização de *views*, realizaria a consulta da vistoria em uma única fonte de informação, evitando erros como o relatado anteriormente, conforme ilustrado na figura 34. A coluna `CONTAINER_ID` encontrada em ambas as tabelas representam chaves estrangeiras das respectivas tabelas dos sistemas.

Atual



Sugestão

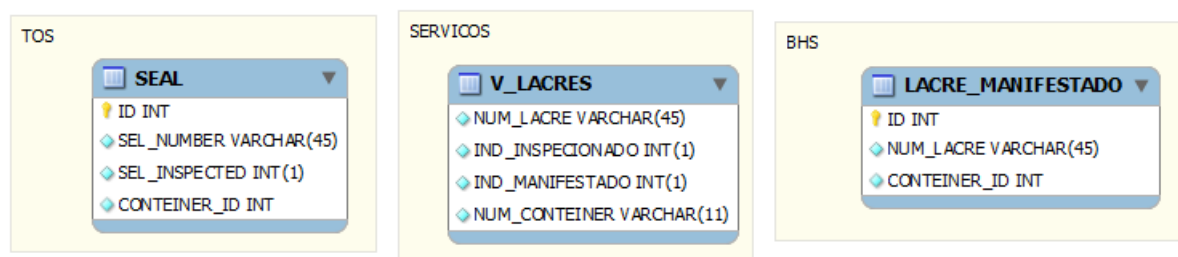


Figura 35 – Exemplo de utilização de views

De forma simplificada, o BHS persiste informações enviadas pelo TOS em suas tabelas, utilizando posteriormente ao validar as regras empregadas nas operações de entrada e saída de caminhões. Uma falha na geração ou processamento da mensagem responsável pela inspeção do lacre, por exemplo, pode acarretar em falhas descritas anteriormente. Ora, se o TOS detém a informação de lacres inspecionados, e o BHS de quais foram manifestados pelo cliente no sistema da receita federal, então a melhor forma de unir as informações que estão persistidas em um mesmo banco de dados é através de um *view*.

No exemplo, a *view* V_LACRES seria o resultado da união de dados entre as tabelas SEAL, do sistema TOS, e LACRE (que seria renomeada para LACRE_MANIFESTADO, sua real responsabilidade), do sistema BHS. Lacres não encontrados em um dos dois sistemas seriam aqueles manifestados incorretamente ou não encontrados no contêiner e aqueles encontrados em ambos os sistemas os que foram corretamente informados e encontrados. Presente em um esquema destinado à camada de serviços, a *view* poderia ser consumida por todos os sistemas e atualizações em um dos sistemas seriam refletidas instantaneamente. O mesmo conceito seria aplicado a outras informações, como avarias, descarga/embarque de contêineres, bloqueios de cargas, dentre outras.

5.4.3 Melhorias aplicadas ao processo de desenvolvimento

Outras sugestões ao terminal portuário incluem essencialmente três pontos, necessários para a melhoria do processo de desenvolvimento de *software*, da qualidade e da gestão como um todo do departamento de TI. Eis os pontos:

- **Utilização de testes unitários:** desenvolver *softwares* requer a realização de testes para que se hajam garantias de que uma determinada entrega irá funcionar conforme o que foi requisitado. O teste mais simples a ser aplicado é o unitário e, a partir de frameworks disponíveis no mercado, é possível automatizá-los para que sejam executados com maior frequência e facilidade durante o processo de desenvolvimento do *software*. Alguns erros detectados pós-implantação poderiam ter sido evitados com o uso deste conceito;
- **Criação de ambiente de homologação:** um ambiente propício para a realização de testes mais próximos da realidade e executados por usuários-chave ou por pessoas que detenham conhecimento suficiente para simular cenários reais nas novas versões dos sistemas é essencial para evitar que *bugs* sejam identificados e corrigidos em produção. Falhas no ambiente de produção podem não só aumentar a gama de problemas para a organização, como fornecer o tempo menor de reação ao problema do que em um ambiente de homologação. A depender da maturidade do setor de qualidade, pode-se criar um ambiente de pré-produção, além de um de contingência da produção para mitigar ao máximo os problemas que podem ocorrer no dia-a-dia do terminal, que deve manter seus sistemas disponíveis sem interrupções;
- **Ferramenta ALM:** a utilização de uma ferramenta para gerenciamento de ciclo de vida das aplicações auxilia a todos os envolvidos, desde usuários, como gerentes dos departamentos, até aos desenvolvedores responsáveis pelas correções dos erros e acompanhamento das aprovações ou reprovas dos testadores. Há bons produtos no mercado, que poderiam ser pesquisados e então um eleito para ser investido pelo terminal portuário, garantindo assim melhor controle sob os erros abertos, priorização de correções, acompanhamento do desenvolvimento, emissão de relatórios, dentre outras vantagens;

- **Criação de manuais e treinamentos:** funcionalidades dos sistemas eram desconhecidas por parte dos usuários, ocasionando em alguns casos o uso equivocado ou simplesmente o total desconhecimento da existência de determinada função em algum sistema. O uso de manuais e a realização de treinamentos poderiam reduzir estes problemas e maximizar a utilização dos sistemas utilizados pelos usuários;

6 CONCLUSÕES

A presente monografia estudou conceitos e técnicas de integração elementares para a construção de arquiteturas de sistemas, iniciando com EAI e apresentando a evolução das tecnologias ao estudar sobre ESB e finalizando o embasamento teórico do estudo com SOA e estruturas baseadas em serviços. Ao fim, foram exploradas as duas propostas de arquitetura empregadas no terminal portuário, bem como a sugestão de integração dos sistemas com a utilização de um barramento de serviços *web* representando as atividades da organização.

A primeira proposta de arquitetura trouxe consigo decisões equivocadas relativas à integração, como a utilização da tecnologia de inteligência artificial para o processamento de tarefas que dispensam tal paradigma, e o próprio conceito de isolar os sistemas por completo, tornando-os ilhas de informação. Apesar das consequências negativas, a primeira tentativa de integração forneceu lições importantes para a proposta seguinte, como utilização de tecnologias mais simples e necessidade de integração entre os sistemas reaproveitando suas estruturas de informações.

Pode-se concluir que, a partir dos resultados obtidos pelas integrações, um ineficiente planejamento da TI de uma organização pode comprometer um alto investimento e o trabalho de outros setores. No estudo de caso, o setor operacional foi diretamente afetado negativamente pelos sistemas na primeira integração, e beneficiado na segunda, outrora o setor alfandegado do terminal prejudicado na segunda tentativa de integração. A sugestão de do barramento serviços de fato poderia potencializar os pontos positivos já atingidos anteriormente e promover vantagens, contudo a principal conclusão a ser obtida é a de que não bastam soluções técnicas e inovações tecnológicas, mas sim processos bem definidos e planejamentos realizados com eficiência possuem maior influência nos resultados.

O estudo realizado permitiu conhecer técnicas de integração que poderiam ser empregadas no terminal portuário, mesmo nas primeiras tentativas de estrutura de sistemas e, como propostas de trabalhos futuros, pode-se realizar uma pesquisa analisando uma possível implantação da arquitetura SOA no contexto portuário, coletando resultados positivos e negativos, e os desafios políticos para atingir o referido objetivo de implantação.

BIBLIOGRAFIA

- Bernstein, P. A., & Newcomer, E. (2009). *Principles of Transaction Processing* (2^a ed.). Burlington: Elsevier.
- Chappell, D. A. (2004). *Enterprise Service Bus: Theory In Practice*. Sebastopol: O'Reilly Media.
- Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Boston: Prentice Hall PTR.
- Erl, T., Cheliah, P. R., Winterberg, T., Trops, B., Hajo, N., Maier, B., et al. (2014). *Next Generation SOA: A Real-World Guide to Modern Service-Oriented Computing*. Indianapolis: Prentice Hall.
- Erl, T., Karmarkar, A., Walmsley, P., Haas, H., Yalcinalp, U., Liu, C. K., et al. (2008). *Web Service Contract Design and Versioning for SOA*. Boston: Prentice Hall.
- Fawcett, J., Quinn, L. R., & Ayers, D. (2012). *Beginning XML* (5^a ed.). Indianapolis: John Wiley & Sons, Inc.
- Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.
- Fowler, M., Rice, D., Foemmel, M., Hieatt, E., & Mee, R. (2002). *Patterns of Enterprise Application Architecture*. Boston: Addison Wesley.
- Harte, L. (2008). *Introduction to Data Multicasting*. Fuquay Varina: Althos.
- Hau, T., Ebert, N., Hochstein, A., & Brenner, W. (2008). Where to Start with SOA. *Proceedings of the 41st Hawaii International Conference on System Sciences*, (pp. 1-9). Waikoloa.
- Hohpe, G., & Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston: Addison-Wesley Professional.
- Linthicum, D. S. (1999). *Enterprise Application Integration*. Boston: Addison-Wesley Professional.
- Maritime Administration. (2008). *Glossary of Shipping Terms*. Washington.

- Rosen, M., Lublinsky, B., Smith, K. T., & Balcer, M. J. (2008). *Applied SOA: Service-Oriented Architecture and Design Strategies*. Indianapolis: Wiley Publishing, Inc.
- Ruh, W. A., Maginnis, F. X., & Brown, W. J. (2000). *Enterprise Application Integration: A Wiley Tech Brief*. New York: John Wiley & Sons, Inc.
- Serain, D. (2002). *Middleware and Enterprise Application Integration: The Architecture of e-Business Solutions*. London: Springer.
- Sweeney, R. (2010). *Achieving Service Oriented Architecture: Applying an Enterprise Architecture Approach*. Hoboken: John Wiley & Sons, Inc.