Modelo Arquitetural para Evolução no Contrato de Serviços no Contexto de Aplicações de Comando e Controle

David de Souza França, Eduardo Martins Guerra
Instituto Nacional de Pesquisas Espaciais - INPE - Av. dos Astronautas, 1758, Jd. Granja - CEP 12227-010 - São José dos Campos - SP, Brasil

Resumo – Na tentativa de prover serviços de interoperabilidade e difusão de informações pertinentes a muitos ramos das Forças Armadas, muitas aplicações *web* são baseadas ou consomem dados de outras aplicações. Isto é um problema quando as aplicações evoluem em velocidades diferentes e compartilham modelos nas trocas de mensagens. Este artigo propõe um modelo arquitetural para evolução das aplicações sem a quebra da interoperabilidade utilizando-se para isso refatorações de XML, que geram automaticamente *scripts* para conversão de dados entre modelos distintos. As informações sobre as versões são gerenciadas por um conjunto de serviços *web* que centralizam as funções de transformação de dados, que pode ser utilizado por diversas aplicações. Como prova de conceito, foram utilizados dados e modelos atualmente em uso na Força Aérea Brasileira e viu-se que o modelo arquitetural proposto pode ser aplicado em *softwares* sem a necessidade de grandes mudanças, apenas informando as aplicações como consumir os serviços disponíveis.

Palavras-chave – Modelo de dados, Interoperabilidade, serviços *web*.

I. INTRODUÇÃO

No âmbito das Forças Armadas do Brasil existem diversas aplicações de cunho estratégico e operacional que fazem parte do Sistema de C2 (Comando e Controle), que auxiliam o planejamento, aplicação e tomada de decisão dos recursos das Forças. Essas aplicações foram construídas baseadas em muitas tecnologias diferentes e sob arquiteturas variadas, já que foram concebidas em épocas diferentes e para propósitos distintos.

Hoje, de acordo com [1], uma das peculiaridades das aplicações de C2 é a necessidade de interoperabilidade, flexibilidade, modularização e segurança. Paradoxalmente, o uso de muitas tecnologias e arquiteturas diferentes tornaram tais aplicações extremamente sensíveis a mudanças, pois qualquer alteração pode quebrar o possível contrato de serviços estipulado entre dois sistemas quaisquer do conjunto de aplicações.

Nem sempre é possível desenvolver ou evoluir todas as ferramentas existentes ao mesmo tempo, já que as atividades de desenvolvimento de sistemas de *software* em grande escala são inerentemente simultâneas e assíncronas [2]. Por conta disso, a evolução de aplicações dá-se de forma lenta e difícil, alterando-se apenas parte das aplicações por vez ou, como é mais comum, evoluir em velocidades diferentes as aplicações existentes no conjunto.

Na tentativa de prover serviços de interoperabilidade e difusão de informações pertinentes a muitos ramos das Forças Armadas, muitas aplicações *web* são baseadas ou consomem dados de outras aplicações que se baseiam em SOA (*Service Oriented Architecture*).

A adoção de uma arquitetura orientada a serviços visa transformar a camada heterogênea atual em uma camada homogênea, tanto em termos de formato quanto em termos tecnológicos. Essa arquitetura possibilita a exposição de funcionalidades na forma de serviços interoperáveis baseados em padrões abertos de mercado, e tem como boa prática o estabelecimento de um modelo comum de dados para troca de informações de forma a respeitar a autonomia de cada Força perante a suas aplicações, representando um incremento de interoperabilidade nos sistemas de C2 [3].

Um contrato neste contexto é a descrição da interface dos serviços, o que inclui, por exemplo, os parâmetros que recebem e a informação que retorna. Para tanto, utiliza-se o padrão WSDL *(Web Services Description Language)* para descrição das operações existentes em um serviço *web*, bem como o protocolo utilizado na troca de mensagens e definição dos tipos de dados que são transmitidos.

Este trabalho tem o objetivo de propor um modelo arquitetural para evolução nos contratos dos serviços entre as aplicações. O modelo de evolução permite que aplicações que evoluírem para novas versões possam continuar trocando informações com as aplicações que ainda utilizam versões anteriores de contratos de serviços.

Para se chegar ao objetivo, o modelo arquitetural proposto utiliza uma ferramenta de refatoração de dados, que gera *scripts* para alteração estrutural dos arquivos XML. Além disso, o novo modelo abrange um servidor para armazenamento e versionamento dos *scripts* gerados.

Para demonstrar o conceito do modelo arquitetural foi utilizado um modelo em uso hoje em dia pela Força Aérea Brasileira que descreve a estrutura que o XML (*eXtensible Markup Language*) deve seguir para trocar informações com outras aplicações. Esse arquivo tem o nome de XML *Schema*. Com base nesse XML *Schema* pode-se utilizar *scripts* XSLT (*eXtensible Stylesheet Language for Transformation*) para se alterar arquivos XML de uma versão para outra.

Como verificação de funcionamento do modelo proposto, uma aplicação migra seu XML *Schema* para uma nova versão e com isso arquivos baseados no modelo antigo devem ser alterados para a nova estrutura. A alteração de estrutura deve ficar a cargo do serviço desenvolvido, alterando os dados para a nova estrutura.

Para elucidar o desenvolvimento, este artigo está estruturado da seguinte forma: a Sessão 2 trata da revisão literária, descrevendo a arquitetura SOA e os padrões de criação de serviços. A Sessão 3 descreve a estrutura do modelo arquitetural dos serviços de versionamento e transformação proposto neste artigo. A sessão 4 apresenta as validações sobre a proposta e a Sessão 5 fala sobre as conclusões a que se chegou.

I. ARQUITETURA ORIENTADA A SERVIÇOS

Serviço, em computação, pode ser definido como uma entidade de uma aplicação que executa uma ação específica, oferecendo funcionalidades previamente definidas em um contrato, chamado de *service description*. Tal contrato possui características descritivas, informando ao utilizador do serviço, chamado de *consumer*, as necessidades semânticas e sintáticas da sua utilização [4].

Um paradigma da computação é a orientação a serviços, onde uma aplicação é desenhada e desenvolvida para oferecer serviços sem se importar quem o consumirá, seguindo o conceito de *request e reply* [5].

Existem muitas tecnologias para a criação, publicação, fornecimento e consumo de serviços. Cada tipo de tecnologia prove características peculiares e dá ao arquiteto de *software* opções que melhores se encaixam as necessidades da aplicação. Pode-se citar como tecnologias de orientação a serviços: *Web Services* [6], CORBA (*Common Object Request Broker Architecture*) [7], OSGi (*Open Services Gateway Initiative*) [8] e Java Beans [9].

Serviços de Comando e Controle

Segundo [10], sistemas de tecnologia da informação para C2 são recursos de TI, constitutivos do Sistema de C2, que proporcionam ferramentas por meio das quais as informações são coletadas, monitoradas, armazenadas, processadas, fundidas, disseminadas, apresentadas e protegidas.

Ainda de acordo com [10], a interoperabilidade é a capacidade de sistemas, unidades ou forças de intercambiarem serviços ou informações, ou aceitá-los de outros sistemas. Assim, tal interoperabilidade está de acordo com a proposta deste artigo e o que preconiza o Ministério da Defesa, como um dos pontos de manutenção da soberania nacional.

SOA Patterns

Com a difusão do *design* SOA muitas soluções de problemas recorrentes foram utilizadas naturalmente e documentadas individualmente em cada projeto. Assim, a esses padrões comuns de construção, publicação e fornecimento de serviços deu-se o nome de SOA *Patterns* [5].

O padrão *Data Model Transformation Pattern* é aplicável quando se tem o problema de serviços usando modelos diferentes para representar os mesmos tipos de dados.

Um serviço chamado *Claim Process* que utiliza um modelo de XML chamado *ProcessClaim.xsd* envia uma mensagem ao receptor que espera receber mensagens no modelo *Claim.xsd*. Para isso, o padrão *Data Model Trasformation* aplica um *script* XSLT de transformação de dados entre o envio e a recepção, marcado com o número 2 na Fig. 1.

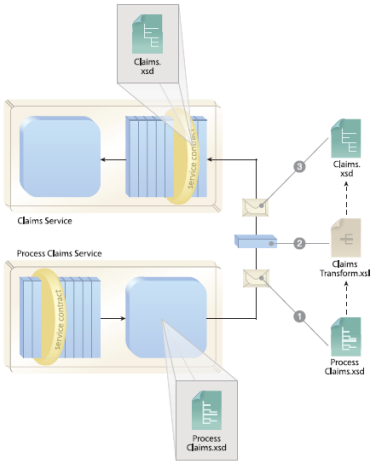

Fig. 1 – Padrão *Data Model Transformation* [5]

Referência [5] comenta que: quando os serviços são implementados como serviços *web*, XSLT é geralmente usado para definir a lógica de mapeamento que é posteriormente executada para realizar a transformação em tempo de execução. Na verdade, o uso de *scripts* XSLT representa a aplicação mais comum desse padrão.

Xml Refactoring

De acordo com [11], *refactoring* é utilizada para designar alteração de uma estrutura de *software* sem alterar seu comportamento. Pode dizer que XML *refactoring* é alteração na estrutura dos dados sem alteração de seu conteúdo. Já o *refactoring database* é a alteração da estrutura sem a alteração de sua semântica, isto é, os dados sob uma nova estrutura, porém continuam com o mesmo significado ou valor.

Para a refatoração de um arquivo XML qualquer é necessária a mudança do XML *Schema* e os *scripts* para a migração dos documentos XML. A ferramenta escolhida para refatoração de XML's foi a Chrysalis [12]-[15], um *plugin* para a plataforma de desenvolvimento Eclipse [16]. Além de prover ferramentas para alteração do XML *Schema* ainda gera os *scripts* XSLT para serem aplicados sobre os arquivos XML.

Utilizando um XML *Schema* como apresentado na Fig. 2.

```
<xs:schema
 <xs:element name="aeronave">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="nome" type="xs:string"/>
    <xs:element name="velocidade" type="xs:int"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Fig. 2 – Exemplo de XML Schema

A Fig. 3 apresenta um XML que segue as definições do XML Schema da Fig. 2.

```
<aeronave nome= “A29A” velocidade=420 />
```

Fig. 3 – exemplo de XML

Supõe-se necessário alterar o XML *Schema* para que a entidade “nome” passe a se chamar “id”. A primeira etapa é alterar o XML *Schema*. A Fig. 4 mostra como ficaria tal *Schema*.

```
<xs:schema
 <xs:element name="aeronave">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="id" type="xs:string"/>
    <xs:element name="velocidade" type="xs:int"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Fig. 4 – XML Schema após alteração

A Fig. 5 descreve como ficaria o *script* XSLT para a mudança da entidade “nome” para “id”.

```
<xsl:template match=”node()|@*”>
 <xsl:copy>
  <xsl:aplly-templates select=” node()|@*”>
 </xsl:copy>
</xsl:template>
<xsl:template match=”@nome”>
 <xsl:attribute name=”id”>
  <xsl:value-of select=”,”/>
 </xsl:attribute>
</xsl:template>
```

Fig. 5 – *Script* XSLT para mudança de nome

Finalmente, a Fig. 6 descreve o arquivo XML após a aplicação do *script* descrito na Fig. 6.

```
<aeronave id= “A29A” velocidade=420 />
```

Fig. 6 – XML após aplicação do *script* XSLT

Isso mostra que se pode alterar a estrutura dos dados, sem alterar o conteúdo, apenas com *scripts* XSLT. Para a aplicação dos *scripts* sobre os arquivos XML é necessário um processador da linguagem XSLT. Hoje no mercado pode-se encontrar algumas opções como, por exemplo: Saxon [18].

II. MODELO ARQUITETURAL

O modelo de arquitetura desenvolvido visa abranger ferramentas já existentes para a refatoração e versionamento de XML’s, e ainda disponibilizar tais serviços sob uma tecnologia SOA com um banco de dados remoto.

O *plugin* Chrysalis versiona seus *scripts* XSLT localmente e em arquivos, deixando a cargo dos usuários o controle de quais *scripts* devem ser utilizados. Para suprir a necessidade de automatização do versionamento, o modelo conceitual utiliza um banco de dados remoto, onde versões já produzidas são armazenadas. Também é disponibilizado funções de conversão de arquivo.

A Fig. 7 descreve o modelo arquitetural proposto, onde aplicações, chamadas de Aplicação 1 e Aplicação 2 trocam informações via XML. Um servidor com uma aplicação *web*, baseado na tecnologia SOA faz o intermédio da conversação, recebendo informações das aplicações referentes ao arquivo a ser enviado e para qual versão deve ser transformado.

Para o recebimento e envio de informações, foi implementado o *Data Model Trasformation Pattern*, visando atender de modo eficiente às requisições de transformações de arquivos XML. O padrão implementado no componente *proxy* da Fig. 7 funciona como intermediário entre aplicações.

O *Proxy* utiliza para gerenciamento das versões um banco de dado, garantindo que várias aplicações possam enviar seus novos modelos de dados a um mesmo local de armazenamento. Isso permite uma melhor qualidade de gerência de dados.

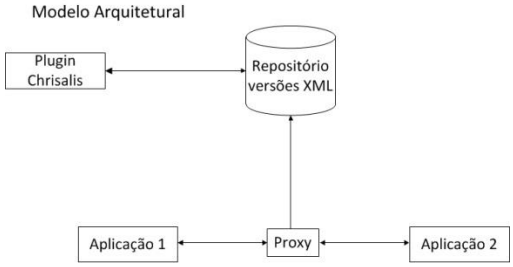

Fig. 7 - Modelo Arquitetural para Evolução no Contrato

Serviços disponíveis

A Fig. 8 mostra o modelo WSDL responsável por descrever os serviços disponíveis do *web service* utilizado.

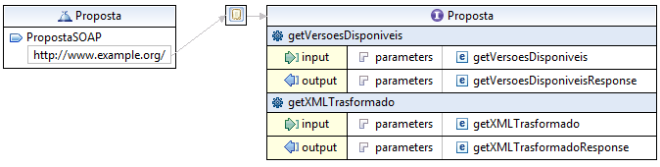

Fig. 8 – WSDL da aplicação proposta pelo modelo

Na Fig. 9 tem-se as especificações da operação *gerVersoesDisponiveis*. A operação recebe o nome do XML *Schema* e retorna como resultado as versões para qual existe *scripts* de transformações para outras versões.

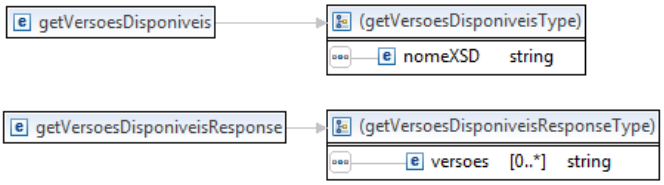

Fig. 9 – Operação *getVersoesDisponives*

Na Fig. 10 tem-se as especificações da operação *getXMLTrasformado*. A operação recebe um arquivo XML e o número da versão para a qual se pretende transformar devolvendo

como resposta um arquivo XML já transformado para o modelo pretendido.

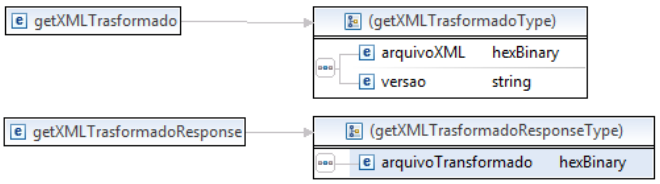

Fig. 10 – Operação *getXMLTrasformado*

Repositório de arquivos

A Fig. 11 mostra o Modelo Entidade-Relacionamento da base de dados. Esse modelo, simplificação de uma solução completa.

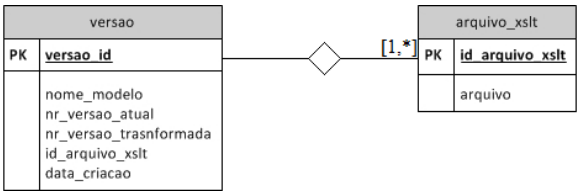

Fig. 11 – Modelo conceitual do banco de dados

O acesso ao banco de dados funciona da seguinte forma: o consumidor utilizando a operação *getVersoesDisponiveis* envia o nome do modelo a ser buscado retornando as versões disponíveis para transformação. Ao utilizar a operação *getXMLTransformado* passando a versão escolhida e o arquivo XML a ser transformado, o serviço aplica os *scripts* e retorna o novo XML. Para a aplicação dos *scripts*, utilizou-se a processador Saxon.

Assim, o modelo proposto se utiliza da ferramenta Chrysalis para gerar os *scripts* de transformações, disponibilizando na forma de *web service*, onde a transformação fica a cargo do serviço. Outro ponto é o gerenciamento das versões em um banco de dados, que centraliza os *scripts* criados.

III. AVALIAÇÃO

Para avaliar a arquitetura, utilizou-se o modelo de dados já existente na FAB (Força Aérea Brasileira), como mostra a Fig. 12, modelo este que estrutura os arquivos XML entre os sistemas POMA (Planejamento e Acompanhamento Operacional de Missão Aérea) e o sistema Máquina de Simulação.

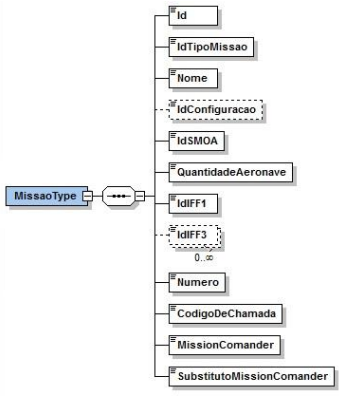

Fig. 12 – Modelo de dados utilizado pela FAB

Como exemplo, a nomenclatura chamada *missão* foi modificada para *ação*. Também se agrupou as entidades *Iff1* e *Iff3* numa única entidade chamada *Iffs*. A Fig. 13 descreve como ficou o XML *Schema* após as alterações propostas.

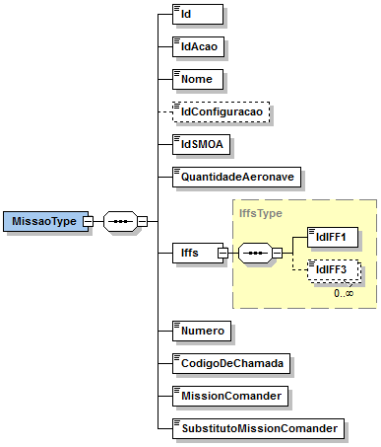

Fig. 13 – Modelo de dados utilizado pela FAB após alterações

Utilização dos serviços disponíveis

Suponha que a aplicação Máquina de Simulação deseja obter dados acerca das missões planejadas no sistema POMA, porém seu modelo de dados sofreu alterações e está utilizando a segunda versão do modelo de troca de dados entre os sistemas. Suponha também que o sistema POMA ainda não alterou seu modelo para a nova versão. Assim uma conversão entre versões de modelos é necessária.

A Máquina de Simulação utiliza a operação *getVersoesDisponiveis* enviando como entrada o dado *Missao.xsd* recebendo como resposta que existem conversões disponíveis entre as versões 1 e 2. Utilizando-se da operação *getXMLTrasformado* enviando para qual versão se quer a transformação e o arquivo XML a ser transformado. A Fig. 14 mostra o arquivo enviado, seguindo o modelo que está na versão 1.

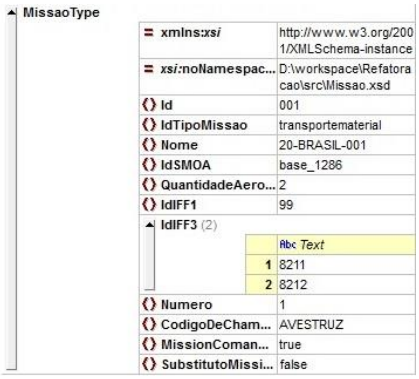

Fig. 14 – Arquivo XML na versão 1

O serviço obtém os *scripts* referentes à conversão da versão 1 para a 2 e os aplica sobre o XML recebido, passando para a nova estrutura, como mostra a Fig. 15. Então o serviço proposta envia como resposta ao pedido o novo XML.

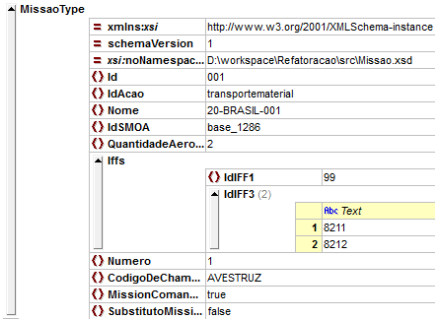

Fig. 15 – Arquivo XML após aplicação de *scripts* XSLT

Análise da Avaliação

Notou-se que com o serviço de transformação dos dados chegou-se ao objetivo proposto, onde o consumidor utiliza o serviço não se importando em como os dados são transformados nem como são armazenados, conhecendo qual o modelo referente ao XML e para qual versão deseja ir.

Pequenas transformações foram efetuadas, mas para muitas transformações ou para uma quantidade grande de XML *Schema's* a serem gerenciadas, o serviços exige muito poucas mudanças nas aplicações consumidoras, além de garantir uma melhor gerência e consistência dos dados armazenados.

Um dos pontos negativos é a limitação de tipos de transformação feitas pela ferramenta Chrysalis, porém é possível estender esse conjunto, implementando novos tipos, já que a ferramenta foi modelada na forma de *plugin.*

Outro fato é a inclusão dos arquivos manualmente no banco de dados, ficando a cargo do usuário. A ferramenta Chrysalis ainda não possui uma forma de enviar os arquivos gerados diretamente ao banco de dados remoto.

IV. CONCLUSÕES

Através da arquitetura proposta é possível criar soluções aplicáveis a sistemas computacionais atualmente utilizados com o objetivo de manter a interoperabilidade. Isso significa que com pequenas mudanças é possível garantir que aplicações continuem se comunicando mesmo que tenham modelos de dados distintos. A utilização de *web services* e uma base de dados distribuída contribui para o gerenciamento dos dados gerados pela Chrysalis, onde todas as aplicações centralizam suas informações para conversões.

Outra vantagem dos serviços de versionamento e transformação dos modelos através da arquitetura SOA foi a criação da aplicação proposta como uma interface permitindo que as aplicações consumidoras utilizem as funcionalidades do *plugin* Chrysalis sem a necessidade de terem conhecimento de sua implementação ou gerenciamento das informações.

Com um modelo arquitetural simples, baseado em tecnologias plenamente conhecidas, pode-se propor uma solução viável para ser aplicada em alguns dos problemas de interoperabilidade existentes hoje nas Forças Armadas. Outra necessidade é a melhoria da ferramenta Chrysalis, aumentando o número de tipos de refatoração. Outro ponto é a evolução para que se possa salvar os *script* diretamente em um banco de dados sendo.

Quanto à aplicação, viu se a necessidade de permitir que os dados sejam transformados em seu local de origem, diminuindo assim o tráfego da rede. Isto se dá pelo fato de que normalmente os *scripts* de transformação têm um tamanho menor do que os arquivos XML.

REFERÊNCIAS

[1] SÁ, Odair O. Comunicações. Comando, Controle e Inteligência nas Forças Armadas (C3I). UNIFA. Rio de Janeiro, p. 9.

[2] AOYAMA, Mikio. Agile Software Process and Its Experience. Niigata Institute of Technology. Japão. 2008, p.10.

[3] BARROS, Breno. GUERRA, Eduardo M. PALMEIRA, Alisson. Arquitetura Orientada a Serviços para o Suporte a Interoperabilidade de Aplicações de C2. In: XII Simpósio de Aplicações Operacionais em Áreas de Defesa, 2010, São José dos Campos. XII Simpósio de Aplicações Operacionais em Áreas de Defesa, 2010.

[4] STAJANOVIC, z. DAHANAYAKE, A. "Service-Oriented Software System Engineering: challenges and Practices", Estados Unidos: Idea Group Publishing, 2005, P. 05-06.

[5] ERL, T, SOA Design Patterns. 1ª edição. Estados Unidos: Prentice Hall, 2009, p. 814.

[6] CERAMI, Ethan. Web Service Essentials. 1ª edição. Estados Unidos: O'Reilly. 2002, p. 304.

[7] TARI, Zahir, BUKHRES, Omran. Fundamentals of Distributed Object Systems: The CORBA Perspective. Estados Unidos: John Wiley & Sons, Inc.2001, p. 282.

[8] The OSGi Architecture. OSGi Alliance. Disponível em <http://www.osgi.org/Technology/WhatIsOSGi>. Acessado em 15 jul. 2013.

[9] HAEFEL, Richard Monson. Enterprise Java Beans. 3ª edição. Estados Unidos: O'Reilly. 2001, p. 500.

[10] MINISTÉRIO DA DEFESA. Doutrina Militar de Comando e Controle. 1ª edição. Brasilia, 2006, p. 68.

[11] FOWLER, M., et. al.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co. (1999)

[12] CASTRO, Elvis H. S. Suporte Automatizado para Refatoração de Documentos XML. São José dos Campos, ITA, 2012.

[13] DUARTE, Arthur P. Ferramenta para Refatoração de Documentos XML. São José dos Campos, Ita, 2010.

[14] SALERNO, Guilherme R. PEREIRA, Marcela S. Um Modelo de Refatoração em Modelos XML. São José dos Campos, ITA, 2009.

[15] SALERNO. Guilherme, PEREIRA, Marcela; GUERRA, E. M. FERNANDES, Clovis. T. A Refactoring Model for XML Documents. In: XML: Aplicações e Tecnologias Associadas (XATA 2010), 2010, Vila do Conde.

[16] ECLIPSE PLATFORM. The Eclipse Foundation, 2010. Disponível em: http://www.eclipse.org. Acessado em 01 jul. 2013.

[17] SAXON. Saxonica Limited. Disponível em: http://www.saxonica.com/welcome/welcome.xml. Acessado em 01 jul 2013.