HOLGER WIEHEN

APLICAÇÃO DA LINGUAGEM VxBPEL EM CONTEXTOS MULTITENANT: UM ESTUDO PARA IMPLEMENTAR VARIABILIDADE

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

São Paulo

2014

HOLGER WIEHEN

APLICAÇÃO DA LINGUAGEM VxBPEL EM CONTEXTOS MULTITENANT: UM ESTUDO PARA IMPLEMENTAR VARIABILIDADE

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Dr. Kechi Hirama

São Paulo

2014

Catalogação-na-publicação

Wiehen, Holger
Aplicação da linguagem VxBPEL em contextos multitenant: um estudo para implementar variabilidade / H. Wiehen. -- São Paulo, 2014.
70 p.

Monografia (MBA em Tecnologia de Software) – Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1.Sistemas de informação 2.Linguagem VxBPEL I.Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II.t.

DEDICATÓRIA

Dedico este trabalho a minha esposa Tânia Cristina Gois de Araújo e ao meu gato Jake.

AGRADECIMENTOS

À minha esposa Tânia Cristina Gois de Araújo pelo seu apoio. A sua compreensão e tolerância em relação à minha ausência, possibilitou a dedicação ao curso de especialização e a esta monografia.

Aos meus pais, Bärbel Wiehen e Friedhelm Wiehen, pelo incentivo, durante toda minha vida. O seu exemplo de curiosidade, espírito investigador e dedicação formaram um pilar para esta pesquisa.

Ao professor Kechi Hirama, pela orientação durante o projeto de monografia. Os seus questionamentos, auxiliaram na evolução da ideia de pesquisa, no direcionamento do estudo e na comunicação dos resultados.

RESUMO

Este trabalho apresenta, inicialmente, o problema de atender, com um sistema *multitenant*, a *tenants* com requisitos variados. A pesquisa abrange quatro áreas complementares: sistemas *multitenant*, modelagem de variabilidade, tecnologia *web services* e a linguagem VxBPEL. Propõe-se uma solução, em nível arquitetural, para o problema de variabilidade *multitenant*, baseado na composição variável de *web services*, com a linguagem VxBPEL. O trabalho analisa as possibilidades e limitações da aplicação desta linguagem no contexto *multitenant.* Baseado nesta análise conclui-se, que uma orquestração de serviços com a linguagem VxBPEL, permite a construção de sistemas *multitenant*, que suportam a variabilidade exigida por múltiplos *tenants*.

Palavras-chave: SaaS, sistema *multitenant*, variabilidade, VxBPEL

ABSTRACT

This work studies the construction of multitenant systems, able to atend tenants with varying requirements. Research covers four foundational topics: multitenant systems, variability modeling, web services technology and the VxBPEL language. Based on this research, we propose a solution for the problem of multitenant variability, at the architecture level, based on the variable composition of web services. Therefore, this work analyzes the possibility and limitations to apply the VxBPEL language in a multitenant environment. According to this analysis, we conclude that service orchestration based on the VxBPEL language allows the construction of multitenant systems, able to support variability required by multiple tenants.

Keywords: SaaS, multitenant system, variability, VxBPEL

LISTA DE ILUSTRAÇÕES

Pág.

Figura 1 – Modelo Multitenant 18
Figura 2 – Modelo de camadas 20
Figura 3 - Níveis de abstração do modelo COVAMOF (Sinnema et al., 2004) 28
Figura 4 – Notação do Ponto de Variação COVAMOF (Sinnema et al., 2004) 30
Figura 5 – Notação da relação de Realização COVAMOF (Sinnema et al., 2004) ... 30
Figura 6 – Notação da Dependência COVAMOF (Sinnema et al., 2004) 31
Figura 7 – Tipos de Dependência COVAMOF (Sinnema et al., 2004) 32
Figura 8 – Notação Interação de Dependência COVAMOF (Sinnema et al., 2004) . 32
Figura 9 – Meta-modelo do framework COVAMOF (Sinnema et al., 2004) 32
Figura 10 – COVAMOF Variability View (Sinnema et al., 2004) 33
Figura 11 - Estrutura de um documento XML 38
Figura 12 – Estrutura de um esquema XSD 39
Figura 13 - Envelope SOAP com transporte HTTP 40
Figura 14 – Envelope SOAP no padrão RPC 41
Figura 15 – Estrutura do documento WSDL 42
Figura 16 – Estrutura do documento BPEL 45
Figura 17 – Composição de elementos VxBPEL e BPEL 48
Figura 18 – Exemplo de processo VxBPEL configurável (parte 1 de 2) 51
Figura 19 – Exemplo de processo VxBPEL configurável (parte 2 de 2) 52
Figura 20 – Visão estrutural da arquitetura multitenant referência 57

LISTA DE TABELAS

Pág.

Tabela 1 – Elementos VxBPEL para representar o Ponto de Variação 48
Tabela 2 – Elementos VxBPEL para representar a relação de Dependência..... 49
Tabela 3 – Elementos da arquitetura multitenant referência 56

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASP	Application Service Provider
BPEL	Business Process Execution Language
CPD	Centro de Processamento de Dados
CVV	COVAMOF Variability View
ESB	Enterprise Service Bus
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAAS	Infrastructure as a Service
MVC	Model View Controller
PAAS	Platform as a Service
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SAAS	Software as a Service
TI	Tecnologia de Informação
UDDI	Universal Description, Discovery and Integration
WSDL	Web service Description Language
XML	Extensible Markup Language
XSD	Xml Schema Definition

SUMÁRIO

1. INTRODUÇÃO 12
1.1 MOTIVAÇÕES 12
1.2 OBJETIVO 14
1.3 JUSTIFICATIVAS 15
1.4 ESTRUTURA DO TRABALHO 16
2. SISTEMAS MULTITENANT 18
2.1 MODELO MULTITENANT 18
2.2 CARATERÍSTICAS ARQUITETURAIS 20
2.3 SUPORTE À VARIABILIDADE 22
2.4 CONSIDERAÇÕES DO CAPÍTULO 23
3. MODELAGEM DE VARIABILIDADE 25
3.1 ENGENHARIA DE LINHA DE PRODUTO 25
3.2 MODELAGEM DE VARIABILIDADE 26
3.3 FRAMEWORK COVAMOF 27
3.3.1 Metas do Framework 27
3.3.2 Elementos de Modelagem 29
3.3.3 Representação 33
3.4 MECANISMO DE REALIZAÇÃO 34
3.5 CONSIDERAÇÕES DO CAPÍTULO 34
4. TECNOLOGIA WEB SERVICE 36
4.1 ARQUITETURA ORIENTADA A SERVIÇOS 36
4.2 PADRÕES PARA WEB SERVICES 37
4.2.1 Formato XML 37
4.2.2 Comunicação SOAP 39
4.2.3 Descrição WSDL 41
4.3 COMPOSIÇÃO DE *WEB SERVICES* 43
4.4 LINGUAGEM BPEL 44
4.5 CONSIDERAÇÕES DO CAPÍTULO 46
5. LINGUAGEM VXBPEL 47
5.1 OBJETIVO DA LINGUAGEM VXBPEL 47

5.2 EXTENSÃO DO PADRÃO BPEL 47
5.3 CONFIGURAÇÃO DO PROCESSO VXBPEL 52
5.4 CONSIDERAÇÕES DO CAPÍTULO 54
6. APLICAÇÃO DA VXBPEL NO MODELO MULTITENANT 55
6.1. ARQUITETURA *MULTITENANT* 55
6.2. REPRESENTAÇÃO DE VARIAÇÃO 58
6.3 CONSIDERAÇÕES DO CAPÍTULO 58
7. ANÁLISE DOS RESULTADOS 59
7.1. REQUISITOS DE VARIABILIDADE *MULTITENANT* 59
7.2. APLICAÇÃO DA VXBPEL NO CONTEXTO *MULTITENANT* 59
7.3. ADAPTAÇÃO DA TECNOLOGIA VXBPEL 61
7. 4 CONSIDERAÇÕES DO CAPÍTULO 62
8. CONSIDERAÇÕES FINAIS 63
8.1 CONTRIBUIÇÕES DO TRABALHO 63
8.2 TRABALHOS FUTUROS 64
REFERÊNCIAS 66

1. INTRODUÇÃO

Este capítulo apresenta as motivações, o objetivo, as justificativas e a estrutura da monografia.

1.1 Motivações

Iniciada com o surgimento da Internet comercial na década 1990 e reforçada pela sua consolidação na década subsequente, observa-se uma transformação do setor de TI (*Tecnologia de Informação*). Inicialmente concebida como meio de comunicação e de acesso à informação, reconheceu-se o potencial da Internet como meio de distribuição universal de aplicações e sistemas computacionais. Neste modelo de distribuição, o sistema de software é implantado em um CPD (*Centro de Processamento de Dados*) e acessado por uma interface executada pelo navegador *web* do usuário. A abordagem baseia-se em dois pilares introduzidos com a Internet: Uma infraestrutura de rede global e padrões *web* para a construção de interfaces gráficas, independente de plataforma e sistema operacional das estações de trabalho.

A Internet como meio de distribuição de sistemas computacionais abre novas possibilidades para satisfazer a demanda das corporações por TI. Por um lado continua viável a tradicional prática de adquirir, implantar e manter um sistema dentro do CPD da própria corporação. Por outro lado surge a oportunidade de substituir o sistema próprio pelo direito de acesso e uso de um sistema operado por um prestador de serviços no modelo SaaS (*Software as a Service*).

Observa-se o crescimento do modelo SaaS no mercado de TI, com base em fortes vantagens. Para o cliente, o software alugado como serviço reduz as despesas com capital. Com a redução do investimento, o cliente preserva opções de decisões futuras, flexibilizando a sua gestão. O cliente evita a complexidade associada à manutenção e operação de um sistema de software, mantendo o foco nos aspectos estratégicos do seu negócio. Do ponto de vista técnico, a hospedagem centralizada facilita a sustentação e operação do sistema, agilizando a entrega de melhorias e

correções. A execução de atividades como gestão de configuração, auditoria e monitoramento por um provedor, focado nestas atividades, aumenta a disponibilidade do sistema.

A realização do modelo SaaS varia de acordo com a sua proximidade com o tradicional modelo de software próprio. No modelo ASP (*Aplication Service Provider*), o provedor mantém para o tenant um sistema dedicado e exclusivo. Enquanto no modelo *Multitenant*, o provedor mantém para todos os clientes um sistema único.

O modelo *Multitenant* tem uma vantagem de custo, em comparação com a abordagem tradicional de provisionar um sistema dedicado para cada *tenant*: O uso de um único sistema significa o compartilhamento dos seus recursos de hardware, bem como a redução das atividades necessárias para manter o sistema operacional.

Enquanto o compartilhamento de uma mesma instância do software gera benefícios de custo, surgem desafios tecnológicos como objetos recentes da pesquisa em Engenharia de Software. São entre eles: O isolamento confiável e eficiente dos *tenants*, a gestão de SLA (*Service Level Agreement*), o provisionamento de hardware para distribuição dos *tenants* e o suporte para a variabilidade funcional demandada pelos *tenants*.

O suporte para a variabilidade, em particular, é um aspecto chave para o modelo *Multitenant*. Esse suporte permitirá a adesão de clientes com necessidades variadas a um serviço SaaS, estendendo os benefícios econômicos do modelo *Multitenant* para um número maior de sistemas e clientes.

1.2 Objetivo

Este trabalho tem como objetivo propor uma solução para a composição de variabilidade em aplicações *multitenant*, permitindo que o sistema processa de forma distinta as requisições de *tenants* diferentes.

O trabalho pressupõe que as variabilidades suportadas pela aplicação são modeladas e gerenciadas com técnicas da Engenharia de Linha de Produto de Software. A abordagem proposta neste trabalho permite a reconfiguração do sistema, em tempo de execução, para cada requisição dos *tenants*.

Para suportar esta reconfiguração dinâmica do sistema, a solução proposta por este trabalho se baseia em uma arquitetura SOA (*Service Oriented Architecture*) e utiliza a camada de orquestração como técnica de composição de variabilidade.

Para isto, utiliza a VxBPEL (KONNIG, 2008), uma extensão do padrão BPEL (*Business Process Execution Language*) para orquestração de *Web Services*. VxBPEL introduz à BPEL recursos específicos para expressar variabilidade. Porém a tecnologia não prevê o uso em um contexto *multitenant*.

O trabalho propõe o uso da VxBPEL para implementar variabilidade, em sistemas *multitenant*. Será mostrado como um sistema *multitenant* poderá incorporar a VxBPEL para esta finalidade e quais adaptações da VxBPEL serão necessário para o funcionamento desta proposta.

1.3 Justificativas

A relevância e a dificuldade de suportar variabilidade em aplicações *multitenant* são discutidas por e Bezemer e Zaidman (2010), Sengupta e Roychoudhury (2011) e Sun *et al.* (2008).

Para modelar a variabilidade neste contexto de *multitenant,* os autores recorreram amplamente às técnicas da Engenharia de Linhas de Produto de Software, como em Mietzner *et al.* (2009) e Moens (2012).

Para apoiar a composição da variabilidade no software Ghaddar, Tamzalit e Assaf (2011), Lee e Kotonya (2010) e Mietzner *et al.* (2009) sugeriram uma arquitetura SOA, porém omitiram como realizar essa composição de fato na camada de orquestração.

Esta questão é abordada por Koning *et al.* (2008), com o desenvolvimento de uma extensão (VxBPEL) do padrão BPEL para representar explicitamente a variabilidade na orquestração de *Web Services*. O desenvolvimento de Koning *et al.* (2008) refere-se a um contexto SOA geral.

A aplicação do desenvolvimento de Koning *et al.* (2008) em um contexto *multitenant,* conforme proposto por este trabalho, irá fechar a lacuna deixada por Ghaddar, Tamzalit e Assaf (2011), Lee e Kotonya (2010) e Mietzner *et al.* (2009), ao omitir como compor a variabilidade na arquitetura SOA. Assim este trabalho abre o caminho para um suporte sistemático de variabilidade em sistemas *multitenant*.

1.4 Estrutura do Trabalho

O Capítulo 1 INTRODUÇÃO apresenta as motivações, o objetivo, as justificativas e a estrutura do trabalho.

O Capítulo 2 SISTEMAS *MULTITENANT* apresenta o conceito de *multitentancy*, a sua aplicação em sistemas SaaS e a dificuldade de atender, com um mesmo sistema, *tenants* com requisitos variados.

O Capítulo 3 MODELAGEM DE VARIABILIDADE aborda o problema de lidar com variabilidade em sistemas de software. Identificam-se os métodos da Engenharia de Linha de Produto de Software como solução para a modelagem de variabilidade em geral e no contexto de sistemas *multitenant* em específico. Apresenta-se o método COVAMOF desenvolvido por Sinnema *et al.* (2004).

O Capítulo 4 TECNOLOGIA WEB SERVICE apresenta o estilo arquitetural orientado a serviço e as principais tecnologias para implementar esta arquitetura. Detalha-se a composição de Web Services, como elemento central da arquitetura SOA, utilizando a linguagem BPEL.

O Capítulo 5 LINGUAGEM VxBPEL apresenta esta extensão do padrão BPEL para orquestração de Web Services. Mostra-se como a VxBPEL, desenvolvida por Koning *et al.* (2008), permite expressar variabilidade na orquestração dos serviços. Abordam-se a linguagem e a implementação desta extensão.

O Capítulo 6 APLICAÇÃO DA VxBPEL NO MODELO *MULTITENANT* mostra como aplicar a VxBPEL em um sistema *multitenant*, para atender com um mesmo sistema *tenants* com requisitos variados.

O Capítulo 7 ANÁLISE DOS RESULTADOS constitui uma analise crítica da aplicação da VxBPEL como mecanismo de variação em sistemas *multitenant*, conforme sugerido. Analisa-se as limitações desta proposta e como superar-los.

O Capítulo 8 CONSIDERAÇÕES FINAIS apresenta as conclusões, contribuições e identifica aspectos com potencial para uma pesquisa futura.

REFERÊNCIAS relaciona a literatura base deste trabalho.

2. SISTEMAS MULTITENANT

Este capítulo apresenta o conceito de *Multitentancy*, a sua aplicação em sistemas de software e a dificuldade deste modelo em suportar variações demandadas pelos *tenants.*

2.1 Modelo Multitenant

O modelo *Multitenant* descreve sistemas de software compartilhados por múltiplas instituições clientes, os *tenants*. Este compartilhamento ocorre pelo acesso dos tenants a uma única instância do sistema. Ou sejam, ao contrario de abordagens tradicionais de reuso da Engenharia de Software, focado nos artefatos estáticos do software, no modelo *Multitenant* os tenants compartilham o sistema de software em tempo de execução, conforme Sengupta e Roychoudhury (2011). A Figura 1 ilustra este compartilhamento.

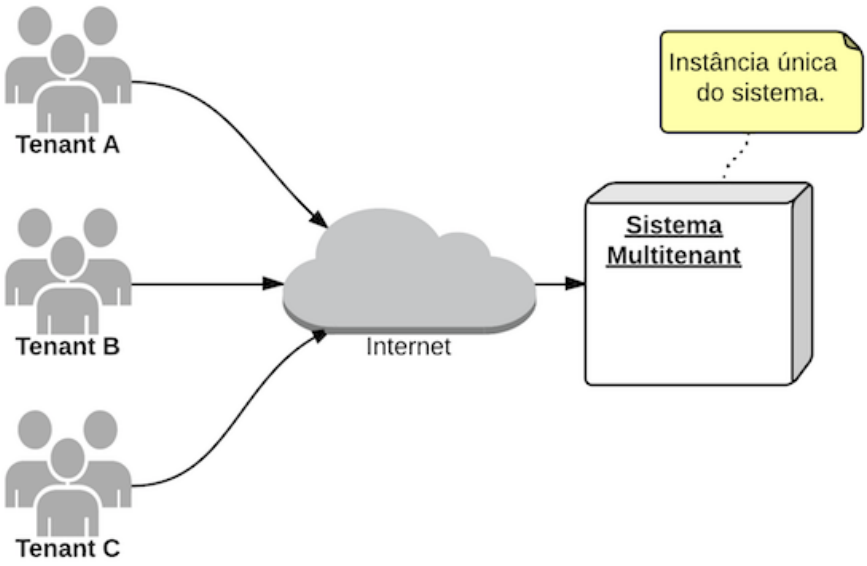

Figura 1 – Modelo Multitenant

O conceito de *multitenancy* surgiu, no âmbito de sistemas de informação, na época Pré-Internet, iniciada com os anos 90. Sistemas contábeis, por exemplo, com a capacidade de manter a contabilidade de múltiplos clientes de um contador, ganharam este atributo. Enquanto esses sistemas foram acessados somente por

uma organização usuária local, a empresa de contabilidade, eles gerenciavam e isolavam os dados de múltiplos clientes, os *tenants*. A expansão do modelo *Multitenant* para novas verticais ocorreu com dois desenvolvimentos (Benlian; Hess e Buxmann (2009).

Por um lado, o nascimento da Internet comercial, na segunda metade da década 90, criou novas possibilidades para o compartilhamento de sistemas ao permitir o acesso remoto de *tenants*, independentemente da sua proximidade geográfica. Por outro lado, a informatização crescente aumentou a quantidade de sistemas de informação utilizados pelas organizações.

Assim, surgiu o interesse de manter o foco em aspectos estratégicos do negócio e de terceirizar uma parte desses sistemas e consumi-los como serviço contratado. Esta tendência de contratar sistemas de software como serviço, chamado de SaaS (*Software as a Service*), criou uma nova relevância para o modelo *Multitenant*, pois o modelo Multitenant se mostrou um estratégica eficaz para a prestação de um serviço de locação de sistemas de software eficiente e escalável, conforme Harms e Yamartino (2010). Com a consolidação deste mercado SaaS, o conceito do sistema *multitenant*, se vinculou ao conceito SaaS, apesar da sua semântica ortogonal.

A inovação dos modelos SaaS e *Multitenant* ocorreu em um contexto de *Cloud Computing* (Computação em Nuvem) mais amplo. Observa-se uma flexibilização do setor de TI de forma geral. A oferta de recursos de TI variados como serviço cria, além do mercado SaaS, um mercado PaaS (plataforma como serviço) e IaaS (infraestrutura como serviço). PaaS se refere à comercialização de ambientes de execução pre-configurados e gerenciados pelo provedor, abrindo ao cliente a possibilidade de implantar sistemas individuais, porém terceirizar a operação destes sistemas. IaaS diz respeito à comercialização de infraestrutura básica como servidores, redes e armazenamento na forma de serviço. Observa-se uma sinergia entre esses mercados SaaS, PaaS e IaaS. Por exemplo, o serviço oferecido por um provedor SaaS pode contratar um hardware como serviço de um provedor IaaS ou PaaS. Neste caso, descrito por Ghaddar, Tamzalit e Assaf (2011), o provedor SaaS terceiriza o investimento e a complexidade da operação de um centro de processamento de dados. Assim, a expansão do mercado de sistemas SaaS e

consequentemente a aplicação crescente do modelo *Multitenant*, não podem ser explicadas sem as facilidades do ecossistema *cloud computing* complementar (Harms e Yamartino, 2010).

2.2 Caraterísticas Arquiteturais

Caraterizada pelo acesso remoto, a arquitetura do modelo *Multitenant* pode ser descrita naturalmente por camadas, iniciando com a camada mais externa, dedicada ao acesso do usuário, conforme ilustrada pela Figura 2.

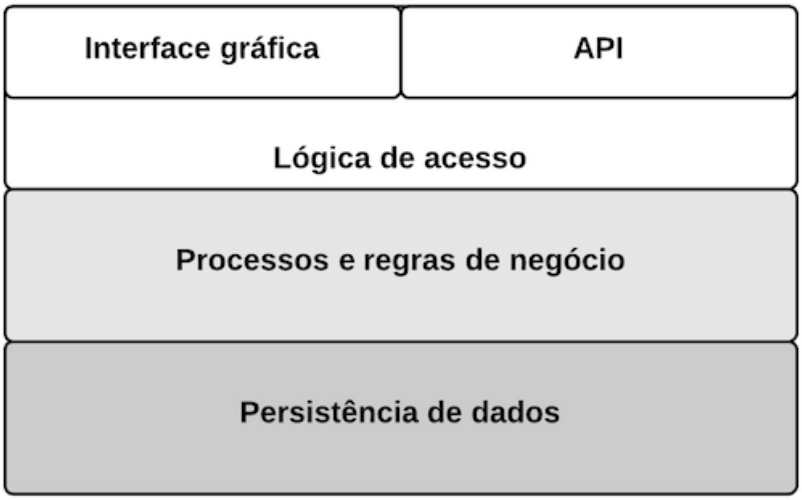

Figura 2 – Modelo de camadas

Enquanto, conceitualmente, o modelo *Multitenant* é independente do mecanismo de distribuição, não pode ser pensado sem a utilização de protocolos de Internet, pois como uma variedade do modelo *SaaS*, sistemas *multitenant* sugiram no contexto de inovação da Internet. Assim, identifica-se o protocolo HTTP (*Hypertext Transfer Protocol*) como principal mecanismo de distribuição, mediando a interação entre o usuário e o sistema. Com a utilização deste protocolo, o sistema *multitenant* se beneficia de uma infraestrutura de rede com base em instalação global. A escolha do protocolo HTTP para a camada de acesso atende então aos requisitos de acessibilidade global e de escalabilidade do sistema. Ao mesmo tempo o protocolo é agnóstico ao formato dos dados transportados entre o usuário e o sistema, preservando flexibilidades para o projeto da interface do sistema. Duas variações

comuns são: A interface gráfica *web*, baseado em documentos HTML e a interface programática, API (*Application Programming Interface*), permitindo uma integração de sistemas mantidos pelo *tenant* com o sistema *multitenant*.

Um aspecto chave de sistemas *multitenant* é o isolamento confiável dos dados associados a cada *tenant*. No conjunto arquitetural esta qualidade é garantida pela camada de persistência. Independentemente do modelo de dados da aplicação, a literatura discute duas abordagens opostas: O modelo multibanco prevê um banco de dados exclusivo para cada *tenant*. Neste modelo a conexão ao banco de dados restringe a visibilidade de uma determinada operação de acesso aos dados de um *tenant* só. Em contraste, o modelo compartilhado prevê a mesclagem dos dados em um banco único, mantendo para cada dado um identificador de *tenant*. Neste modelo o isolamento correto depende da presença de uma cláusula de restrição em cada operação de acesso, limitando a visibilidade aos dados com determinado identificador de *tenant.* As caraterísticas destes dois modelos de persistência alternativos, publicadas por Chong; Carraro e Wolter (2006) e Wang *et al.* (2008), possibilitam a escolha da abordagem adequada para cada sistema, em função dos seus requisitos.

Sistemas *multitenant* necessitam de capacidade de escalamento, como uma instância única do sistema que concentre o processamento de todos os *tenants*. Estratégicas arquiteturais adequadas permitem o escalamento vertical ou horizontal das camadas do sistema. O escalamento vertical consiste no provisionamento de um hardware com poder computacional maior, permitindo o processamento de um número maior de requisições. Enquanto isto o escalamento horizontal aumenta o processamento por paralelizar e distribuir a carga a múltiplos nodos. Observa-se na prática a estratégica de escalamento horizontal para a camada de processamento das regras de negócio. A camada de persistência, porém, representa o maior desafio de escalabilidade em sistemas *multitenant*, devido à dificuldade de distribuir o banco de dados a múltiplos nodos. Chong; Carraro e Wolter (2006) apresentam diversas estratégias para o escalamento de bancos de dados.

A comercialização de sistemas *multitenant* como serviço, conforme o modelo SaaS, implica na necessidade de garantir o funcionamento do sistema nos limites definidos

pelo SLA (*Service Level Agreement*) acordado entre provedor e *tenant*. Estratégicas arquiteturais adequadas garantem o nível de serviço, por permitirem o monitoramento e a inspeção do sistema em execução, a implantação do sistema com redundância, a troca de elementos de hardware e a atualização do software, sem interrupção do serviço.

2.3 Suporte à Variabilidade

A ideia de compartilhar uma instância de software entre múltiplos clientes é contrariada pela variação natural dos requisitos destes *tenants*. O provedor do software *multitenant*, interessado na ampla exploração comercial do seu produto procura atender, com um mesmo sistema, clientes com necessidades variadas. Sengupta e Roychoudhury (2011) relacionam o potencial comercial de um software *multitenant* ao seu grau de suporte à variabilidade funcional e não funcional. Essa importância de atender aos requisitos individuais dos *tenants* é confirmada pelo estudo de mercado de Kaplan (2008).

Para o provedor de software surge a questão em qual extensão irá suportar requisitos variados. O projeto e a manutenção de variabilidade no sistema de software elevam a sua complexidade e representa um custo adicional para o provedor. De ponto de vista econômico o suporte à variabilidade é indicado enquanto a receita adicional, proporcionada pela adesão de novos clientes, ultrapassa este custo da complexidade adicional. Sengupta e Roychoudhury (2011) afirmam que a ausência completa de variabilidade no sistema *multitenant* não é aceitável, em especial para provedores de pequeno e médio porte sem poder de barganha com os seus clientes. Assume-se então que todo sistema *multitenant* comercial, precisa suportar alguma variabilidade. A extensão deste suporte é definida individualmente para cada sistema, em função de critérios económicos e tecnológicos.

A complexidade, resultante de variabilidade em sistemas *multitenant*, reflete-se em vários aspectos do desenvolvimento deste software. No nível de análise e

especificação surge a necessidade de levantar sistematicamente as variações suportadas, o seu rastreamento durante o ciclo de desenvolvimento e a gestão de eventuais mudanças. Ghaddar, Tamzalit e Assaf (2011) e Mietzner *et al.* (2009), entre outros autores, recorrem à Engenharia de Linha de Produto de Software e identificam modelos para lidar sistematicamente com esta complexidade. Em especial identificam-se as diversas técnicas de modelagem de variabilidade, como auxilio adequado na análise e especificação da variabilidade.

Na questão de projeto arquitetônico, revela-se que a Engenharia de Linha de Produto de Software fornece uma orientação menor. Enquanto conceitos desta engenharia, como ponto de variação, variação e variante representam um modelo abstrato para o reflexo da variabilidade no sistema *multitenant*, os tradicionais mecanismos para implementar estes conceitos não são suficientes para sistemas *multitenant*. Nestes sistemas, o *binding*, ou seja a escolha de uma alternativa em um ponto de variação, precisa ocorrer em tempo de execução (*runtime*), enquanto a Engenharia de Linha de Produto de Software tradicional discute mecanismos de *binding* em tempo de compilação ou implantação. A necessidade de suportar o *binding* em tempo de execução deve-se ao compartilhamento da mesma instância de execução do software, no modelo *Multitenant*.

A literatura recente sobre o suporte à variabilidade em sistemas *multitenant* converge para a arquitetura SOA como mecanismo de implementação da variabilidade. Moens *et al.* (2012), Mietzner *et al.* (2009) e Ghaddar (2010), entre outros, descrevem uma estratégia em que cada ponto de variação corresponde à invocação de um *web service*. Existe uma implementação própria deste *web service* para cada variação especificada neste ponto de variação. A orquestração dinâmica permite a variação em tempo de execução.

2.4 Considerações do Capítulo

O estudo sobre sistemas *multitanent* revelou uma relevância comercial crescente deste modelo de software. Ao mesmo tempo este modelo apresenta alguns desafios, estudados pela Engenharia de Software. O presente trabalho foca o

suporte de requisitos individuais dos *tenants* em sistemas *multitenant*. Entre as abordagens de solução encontradas na literatura, considera-se a modelagem de variabilidade como recurso de análise e especificação e a arquitetura baseada em serviços como recurso de projeto.

3. MODELAGEM DE VARIABILIDADE

Este capítulo aborda o tratamento sistemático de variabilidade em famílias de software. Descrevem-se conceitos e métodos de modelagem da variabilidade, desenvolvidos pela Engenharia de Linha de Produto, com ênfase para o *framework* COVAMOF.

3.1 Engenharia de Linha de Produto

O tratamento sistemático de variabilidade, no contexto de software, foi desenvolvido pela Engenharia de Linha de Produto, conforme Pohl; Böckle e Van Der Linden (2005).

O conceito da Linha de Produto é antigo em outras engenharias. Bass; Clements e Kazman (2012) identificaram como origem histórica o século 19, quando se iniciou, na manufatura de armas, a produção eficiente de uma variedade de modelos, com base em uma plataforma central e peças opcionais. No século seguinte a Linha de Produto se confirmou em diversas indústrias como uma abordagem chave para aumentar a variedade de produtos com eficiência econômica. Pohl; Böckle e Van Der Linden (2005) descrevem como o conceito de Linha de Produto dominou a indústria automobilística a partir da década 1980 ao permitir o atendimento de inúmeros nichos de mercado com modelos específicos.

No setor de software Park; Sugumaran e Kang (2006) situam a Engenharia de Linha de Produto no campo da pesquisa clássica sobre reuso de software, iniciada com a primeira conferência sobre Engenharia de Software, organizada em 1969 pela OTAN.

O conceito da Linha de Produto de Software foi desenvolvido pelo *Software Engineering Institute* (SEI) da Universidade de Carnegie Mellon dos EUA e descrito em Clements (2002).

O objetivo de uma Linha de Produto é o desenvolvimento eficiente de uma família de softwares, pelo emprego de uma plataforma comum e a customização de componentes compartilhados. A Linha de Produto é indicada para um conjunto de softwares que compartilham um grande número de requisitos centrais e cujas partes comuns podem ser exploradas economicamente.

Para isto, Bass; Clements e Kazman (2012) introduziram o conceito do Escopo da Linha de Produto. Esse conceito define quais variações do software podem ser derivadas da linha, quais variações demandariam uma adaptação da linha, por exemplo uma introdução de novos Pontos de Variação e quais softwares estão completamente fora do escopo da Linha de Produto.

Bass; Clements e Kazman (2012) destacaram o tratamento sistemático da variação existente entre os membros da família, como uma preocupação chave. A disciplina de Gestão de Variação atende a esta preocupação, e inclui análise, especificação, rastreamento e evolução da variação suportada pela Linha de Produto.

3.2 Modelagem de Variabilidade

Elemento central da Gestão de Variação é a representação explícita das variações suportadas pela Linha de Produtos, com um modelo de variabilidade.

Existem múltiplas técnicas para modelagem de variação, no campo da Engenharia de Linha de Produto. Estes se distinguem pelo seu objetivo e o contexto da sua aplicação. Sinnema e Deelstra (2007) apresentaram uma seleção destes métodos e mostrou como eles concebem diferentes recursos de modelagem para suportar os seus cenários de aplicação.

Um conceito central à modelagem de variação é o Ponto de Variação. Este e os conceitos associados permitem uma análise e especificação estruturada da variação em famílias de software.

Como foi descrito por Koning *et al.* (2007), Ponto de Variação é toda parte de um sistema que pode variar, conforme os requisitos desse sistema. A variabilidade suportada em um determinado Ponto de Variação é dada por um conjunto de alternativas, chamadas de Variantes. A escolha de uma Variante em cada Ponto de Variação, resulta em um produto específico, chamado de Configuração.

A seleção de uma Variante para um determinado Ponto de Variação é chamada de *Binding*. O momento em que este *Binding* ocorre, dentro do ciclo de desenvolvimento do software, determina quanta flexibilidade existe para reconfigurar o sistema. Existem dois tipos de *Binding*. O *Binding* em tempo de compilação, distribuição ou implantação cria uma Configuração estática. Uma modificação desta Configuração requer voltar até essa fase do desenvolvimento, na qual ocorreu o *Binding*. Em contraste, um sistema que suporta o *Binding* em tempo de execução (*run-time*) permite a reconfiguração dinâmica do sistema sem necessidade de trocar ou reiniciar este sistema. Com suporte ao *Binding* em tempo de execução, uma mesma instância do sistema tem o potencial de assumir todas as Configurações, conforme destacado por Koning *et al.* (2007).

3.3 Framework COVAMOF

Os métodos de modelagem concretos partem do conceito do Ponto de Variação e definem uma notação, visões e restrições, entre outros. COVAMOF é um *framework* para modelagem de variabilidade em famílias de software desenvolvido por Sinnema *et al.* (2004).

3.3.1 Metas do Framework

COVAMOF se baseia em observações empíricas sobre a natureza da variabilidade em linhas de produto. Desta observação derivam-se quatro requisitos para o *framework* (*design goals*). Esses quatro requisitos para o *framework* COVAMOF são:

[R1] COVAMOF deve suportar a representação explícita e uniforme de variabilidade em todos os níveis de abstração.

Observa-se que a variação de uma Linha de Produto afeta todos os níveis de abstração da sua modelagem. COVAMOF assume um modelo com três níveis de abstração para organizar os artefatos elaborados no desenvolvimento da Linha de Produto. Os níveis de abstração considerados são: O Modelo de Caraterísticas, Arquitetura e a Implementação de Componentes, conforme apresentado na Figura 3.

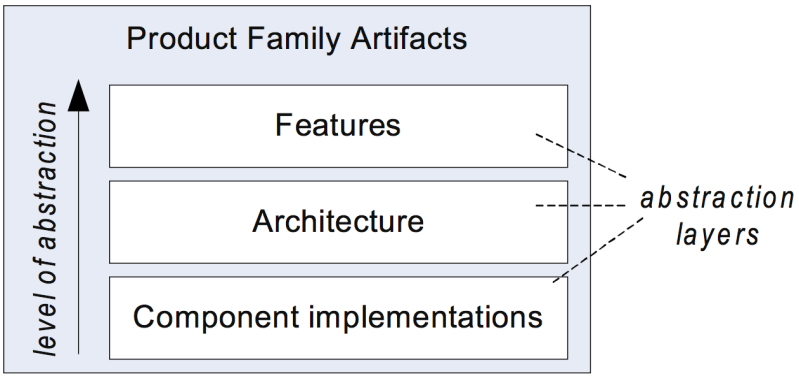

Figura 3 - Níveis de abstração do modelo COVAMOF (Sinnema *et al.*, 2004)

[R2] COVAMOF deve suportar a organização hierárquica da Variação.

A Linha de Produto costuma ser confrontada com um grande volume de variação, distribuído pelos três níveis de abstração. A organização dos Pontos de Variação através de relações hierárquicas permite lidar eficientemente com este volume de informação.

[R3] COVAMOF deve suportar a modelagem de restrições para o Binding dos Pontos de Variação.

Observa-se que a seleção da Variante de um ou mais Pontos de Variação pode ser regida por restrições. Por exemplo, a restrição que define um tempo de resposta máximo para determinado sistema, restringe o *Binding* de todos Pontos de Variação com impacto de latência. A seleção das Variantes deve formar uma configuração válida, que atende ao requisito de tempo de resposta máximo.

[R4] COVAMOF deve apoiar a análise e resolução de interdependências entre restrições distintas.

Assim como um Ponto de Variação pode influenciar outro Ponto de Variação, conforme [R3], observa-se que as restrições de *Binding* podem ser interdependentes. Como exemplo, assume-se um sistema com limite para o consumo de memória, além de uma latência máxima. Uma seleção de Variantes, que atendem ao requisito de latência, pode deixar a configuração inválida por exceder o limite de memória e vice-versa.

3.3.2 Elementos de Modelagem

Correspondendo aos quatro requisitos (*design goals*) do *framework*, COVAMOF concebe quatro elementos de modelagem. Estes elementos de modelagem são:

[E1] COVAMOF considera um Ponto de Variação uniforme.

O Ponto de Variação é utilizado para a modelagem de variabilidade em três níveis de abstração: Modelo de Caraterísticas, Arquitetura e Implementação de Componentes. O Ponto de Variação é caraterizado por atributos, dos quais os principais são: O conjunto de Variantes, o tempo de *Binding* e o tipo do Ponto de Variação. O tipo define restrições para o *Binding* de um Ponto de Variação. COVAMOF considera 5 tipos:

- O Ponto de Variação opcional requer a escolha de uma ou zero Variantes.
- O Ponto de Variação alternativo requer a escolha de exatamente uma Variante.
- O Ponto de Variação variante requer a escolha de pelo menos uma Variante.
- O Ponto de Variação variante opcional permite a escolha de Variantes sem restrição.
- O Ponto de Variação de valor requer a escolha de uma faixa predefinida.

Figura 4 mostra um exemplo para cada tipo de Ponto de Variação, em notação COVAMOF.

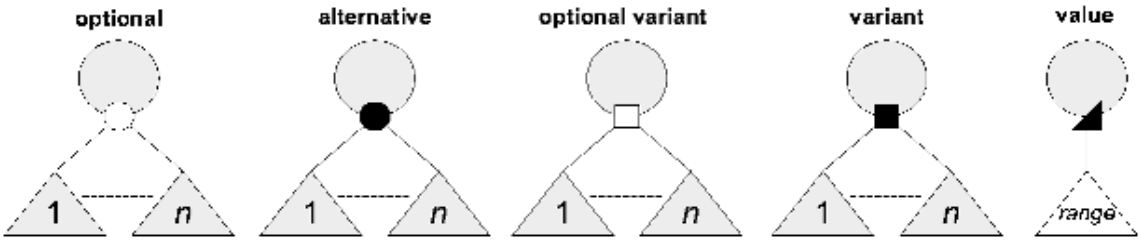

Figura 4 – Notação do Ponto de Variação COVAMOF (Sinnema *et al.*, 2004)

[E2] COVAMOF introduz o conceito da relação de Realização.

A organização hierárquica da variação ocorre envolve os Pontos de Variação sem mecanismo de realização associado. O Ponto de Variação em nível de abstração superior costuma ser realizado por um ou mais Pontos de Variação em nível inferior. Por exemplo, uma Variação modelada no nível de Caraterística reflete-se na Arquitetura e na Implementação de Componentes, através de Pontos de Variação relacionados. COVAMOF permite a modelagem explícita deste relacionamento hierárquico com o recurso da relação de Realização. A relação de Realização tem cardinalidade variada, conforme representada pela Figura 5, em notação COVAMOF.

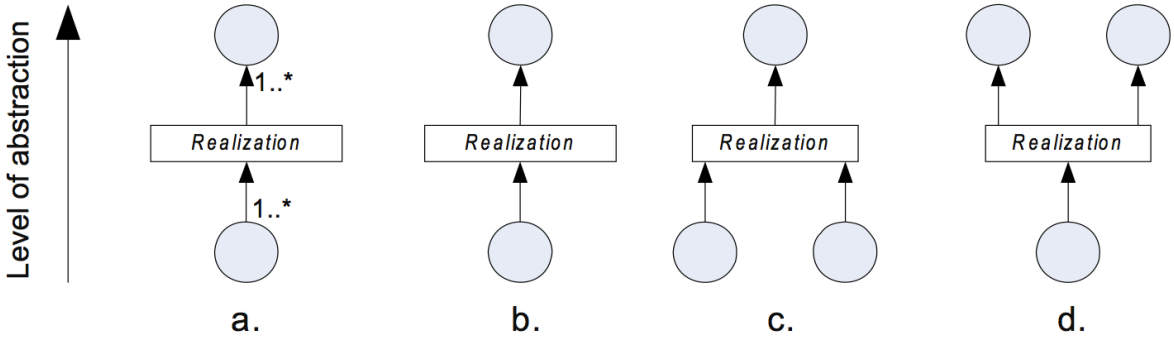

Figura 5 – Notação da relação de Realização COVAMOF (Sinnema *et al.*, 2004)

[E3] COVAMOF define o conceito da Dependência.

A modelagem de restrições existentes para o *Binding* de um ou mais Pontos de Variação ocorre como Dependência COVAMOF. Associado à noção de restrição é o conceito de validade. A Dependência especifica, através dos seus atributos, condições para determinar a validade de um *Binding*. Os atributos chave são o tempo de validação e a associação dos Pontos de Variação.

O tipo de associação de um Ponto de Variação diz respeito o quanto se pode classificar e quantificar o seu impacto na validade da Dependência e pode assumir os valores: previsível, direcional e desconhecido. Enquanto o impacto do *Binding* de um Ponto de Variação com associação previsível, para a validade da Dependência, pode ser analisado quantitativamente antes do *Binding*, sabe-se para uma associação direcional somente se o seu impacto na validade será positivo ou negativo.

O tipo de associação dos Pontos de Variação determina se a validade de uma Dependência pode ser verificada estaticamente ou se demanda um teste da configuração após a derivação do produto. Figura 6 mostra a Dependência, os Pontos de Variação associados e do tipo de associação.

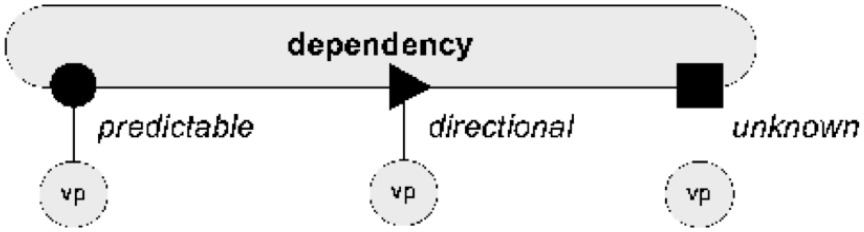

Figura 6 – Notação da Dependência COVAMOF (Sinnema *et al.*, 2004)

Além do tipo das associações com os Pontos de Variação, a Dependência possui um tipo próprio, dizendo respeito à forma como a sua validade é determinada. Para Dependências lógicas existe uma função de validade que mapeia a seleção de Variantes nos Pontos de Variação a um valor booleano. Dependências numéricas permitem derivar da seleção de Variantes um valor numérico e definem para ele uma faixa aceitável, para determinar a validade da seleção. Dependências nominais definem um mapeamento da seleção de Variantes a uma categoria nominal e definem um conjunto de categorias válidas. A notação da Dependência, incluindo seu tipo, é mostrado pela Figura 7.

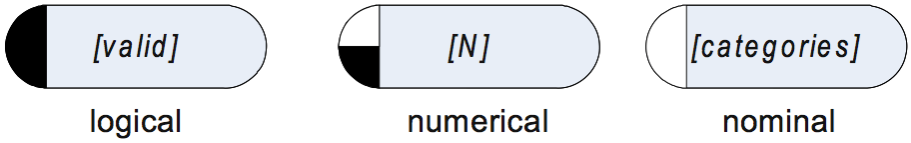

Figura 7 – Tipos de Dependência COVAMOF (Sinnema *et al.*, 2004)

[E4] COVAMOF formaliza a Interação de Dependências.

Dependências podem ter uma relação entre si, caso a validade de uma dependência impacte na validade de outra. COVAMOF permite a sua modelagem como Interação de Dependência. A especificação da Interação de Dependência, incluindo a sua direção de efeito, permite entender e resolver conflitos de validade na resolução de um produto. Figura 8 mostra a notação COVAMOF da Interação de Dependência.

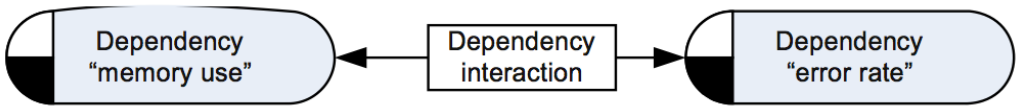

Figura 8 – Notação Interação de Dependência COVAMOF (Sinnema *et al.*, 2004)

A relação dos quatro elementos de modelagem do *framework* é formalizada pelo meta-modelo da Figura 9. Em COVAMOF o Ponto de Variação é o elemento de modelagem principal. Dois ou mais Pontos de Variação podem ser associados hierarquicamente por uma relação de Realização. Opcionalmente o Ponto de Variação é associado a uma ou mais Dependências, ou seja o seu *Binding* é restringido pela condição de validade dessas Dependências. Dependências com impacto mútuo são associadas através de uma Interação de Dependência.

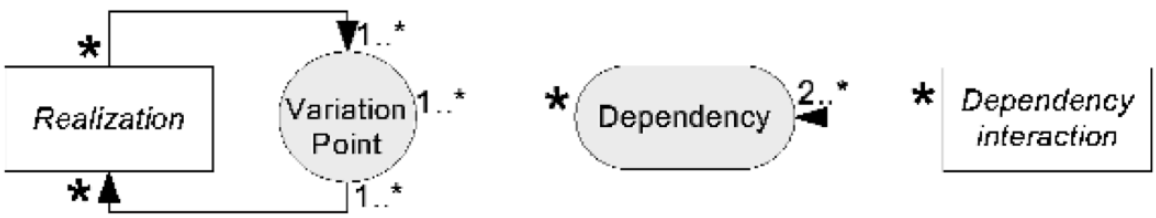

Figura 9 – Meta-modelo do framework COVAMOF (Sinnema *et al.*, 2004)

3.3.3 Representação

Os recursos de modelagem do *framework* permitem a elaboração de um modelo de variabilidade para a Linha de Produto, o COVAMOF *Variability View* (CVV). Existem duas formas de representar esse modelo:

A especificação textual é baseada em uma linguagem derivada da *Extensible Markup Language* (XML). Sinemma (2004) contém exemplos desta linguagem.

A especificação gráfica do modelo CVV envolve duas visões distintas e complementares. A visão de Ponto de Variação mostra a distribuição da variação e as relações de Realização. Já a visão de Dependência documenta as restrições que existem para o *Binding* dos Pontos de Variação e a Interação das Dependências.

Figura 10 ilustra como o modelo CVV é composto por estas duas visões complementares e como o modelo representa uniformemente a variabilidade existente nos diferentes níveis de abstração da Linha de Produto.

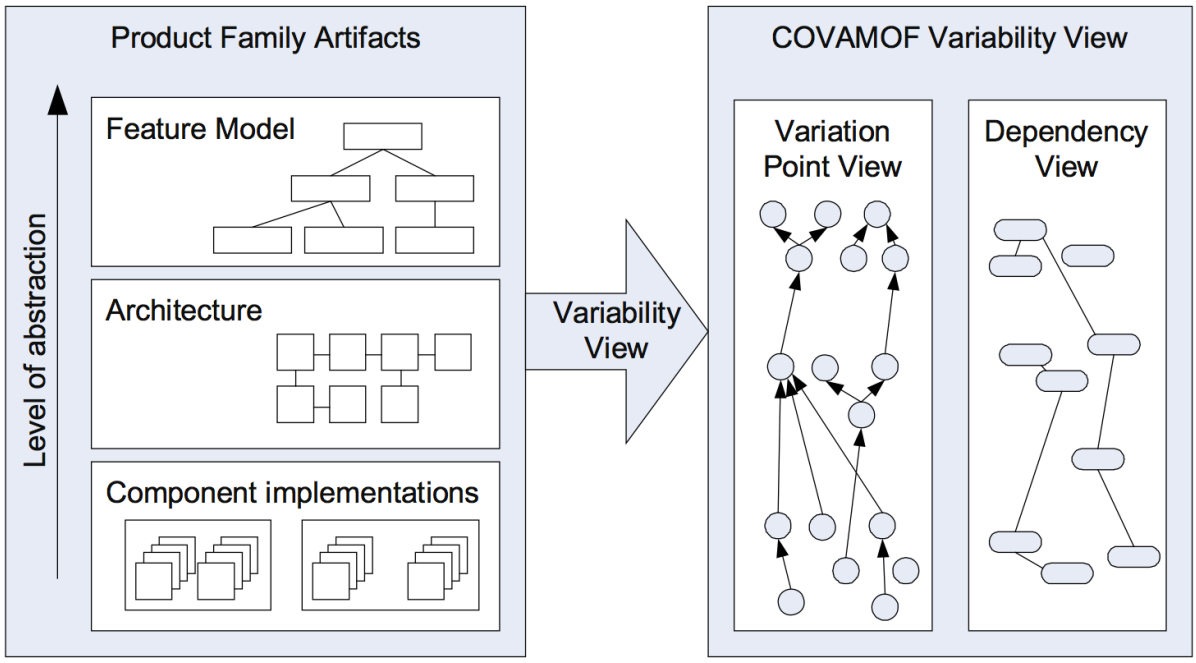

Figura 10 – COVAMOF Variability View (Sinnema *et al.*, 2004)

3.4 Mecanismo de Realização

Conforme descrito, a variabilidade suportada em uma Linha de Produto limita-se a Pontos de Variação bem definidos. A implementação desses Pontos de Variação no artefato de software requer um mecanismo de realização. Bass; Clements e Kazman (2012) consideraram mecanismos de nível arquitetural e mecanismos mais próximos da implementação. Comum a eles é o objetivo de permitir a derivação de produtos customizados, sem duplicação e alteração de código.

No nível arquitetural a variação pode ser realizada por 3 táticas: A inclusão ou exclusão de componentes; a seleção de componentes com uma mesma interface, porém implementação distinta; e a inclusão de um número diferente de elementos replicados, quando se trata de uma variação não funcional, como desempenho e disponibilidade.

Em um nível abaixo da arquitetura, Bass; Clements e Kazman (2012) identificaram como mecanismos típicos para realização de variabilidade: o recurso de herança e polimorfismo em tecnologia orientada a objeto; aspectos numa solução orientada a aspectos; e condicionais da linguagem de programação, entre outros.

Koning *et al.* (2008) enfatizaram a relação entre mecanismo de realização e tempo de *Binding*. Enquanto, o mecanismo de herança permite mudar o *Binding* de um ponto de variação dinamicamente em tempo de execução, a troca de um componente por uma implementação alternativa ocorre principalmente no momento de implantação. Ou seja, não todos os mecanismos de realização suportam a reconfiguração dinâmica de um software derivado da linha de produtos.

3.5 Considerações do Capítulo

O estudo da modelagem de variabilidade, desenvolvida pela Engenharia de Linha de Produto de Software, revela conceitos e modelos importantes para o suporte de variabilidade em sistemas *multitenant*. O Ponto de Variação constitui um recurso fundamental para modelar como o sistema varia e permite atender aos *tenants* com

requisitos variados. A sistemática, o detalhe e a notação do framework COVAMOF permitem elaborar um modelo de variabilidade formal para o sistema *multitentant*. Na implementação da variabilidade, destaca-se a importância de escolher um mecanismo de realização que suporta a variabilidade dinâmica demandada pelo sistema *multitenant*. Conforme apresentado no primeiro capítulo, este trabalho irá buscar o mecanismo de realização no domínio de Sistemas Orientados a Serviços (SOA).

4. TECNOLOGIA WEB SERVICE

Este capítulo introduz o conceito de *Web Service*. O assunto é contextualizado com uma breve descrição da Arquitetura Orientada a Serviço. A realização de um *Web Service* é abordado com uma apresentação dos padrões tecnológicos relevantes. A composição de sistemas a partir de múltiplos serviços é discutida, com foco na linguagem BPEL.

4.1 Arquitetura Orientada a Serviços

Uma arquitetura SOA (*Service Oriented Architecture*) descreve um sistema construído a partir de componentes distribuídos, segundo Bass; Clements e Kazman (2012). O autor diferencia a arquitetura SOA por basear-se em componentes independentes, expostos como serviços e mantidos potencialmente por departamentos ou instituições independentes. Caraterística chave, neste contexto, é a capacidade da arquitetura SOA de integrar serviços implementados em tecnologias heterogéneas através de uma interface padronizada.

Papazoglou (2007) identifica a arquitetura SOA como uma tendência para a tecnologia da informação corporativa. Como fatores de negócio para esta tendência, há a necessidade crescente de integrar processos de negócio que abrangem múltiplos departamentos e corporações. Ao mesmo tempo, os sistemas corporativos devem ter a flexibilidade de acompanhar as mudanças rápidas do ambiente de negócio.

Como fatores tecnológicos que contribuem com a adoção de arquiteturas SOA identifica-se a Internet e os padrões Web. Estes estabelecem um meio de comunicação com alcance global e presente em todas as instituições. *Web Services* mantidos por instituições independentes podem ser integrados com base nesta infraestrutura global.

Elemento principal da arquitetura SOA é o *Web Service*, que Papazoglou (2007) identifica como uma representação de uma função de negócio completa. A

realização do serviço garante três propriedades: 1) O serviço é autocontido. Enquanto um *Web Service* pode invocar outros serviços, ele deve manter seu próprio estado. 2) O serviço é independente de plataformas e tecnologias de implementação específicas. 3) O serviço pode ser dinamicamente recombinado, resultando em novas aplicações, não previstas durante o seu projeto original.

Além do *Web Service*, a arquitetura SOA contêm elementos de infraestrutura opcionais, enumerados por Bass; Clements e Kazman (2012). A localização dinâmica de serviços, isto é a determinação da sua interface e do seu endereço de comunicação, ocorre por consulta a um registro de serviços (*UDDI registry*). A comunicação entre o serviço e o seu consumidor acontece, no cenário mais simples, no padrão ponto-a-ponto. Alternativamente, a comunicação pode ser mediado por *hub* de comunicação central. Esta mediação pelo *Entreprise Service Bus* (*ESB*), visa facilitar a integração de serviços heterogêneos, oferecendo a tradução de protocolos, verificação de integridade, segurança e transações.

4.2 Padrões para Web Services

Para possibilitar a integração de sistemas, a realização de um *Web Service* é baseada em padrões tecnológicos. Enquanto existem abordagens e tecnologias alternativas, o padrão de fato é dado pelas tecnologias XML (*Extensible Markup Language*), SOAP (*Simple Object Access Protocol*) e WSDL (*Web Services Description Language*), conforme Curbera *et al.* (2002).

4.2.1 Formato XML

XML é uma linguagem para a representação textual de dados, especificada em W3C (2008). Em um documento XML os dados são representados hierarquicamente, em uma estrutura de árvore. O elemento básico do documento XML é a *Tag*, que funciona como identificador do dado útil. Para isto a *Tag* consiste de um elemento de abertura e de fechamento, que delimitam o dado. Informação adicional sobre o

elemento é expressada como atributo da *Tag*. Figura 11 mostra um exemplo de documento em XML.

```
<?xml version="1.0"?>
<catalogo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="catalogo.xsd">
        <livro id="1">
                <isbn>0596002521</isbn>
                <titulo>XML SCHEMA</titulo>
        </livro>
</catalogo>
```

Figura 11 - Estrutura de um documento XML

A correção de um documento XML é determinada em dois níveis. A gramática da linguagem XML permite avaliar se determinado documento é bem formado, isto é se obedece às regras genéricas da XML, conforme W3C (2008). A verificação se o documento obedece uma estrutura específica é chamada de validação e requer uma referencia além da gramática XML em si.

Como linguagem de marcação genérica, a XML prevê a definição de um formato próprio para cada classe de documentos. A definição desse formato, como esquema formal, é regido pelo padrão XSD (XML *Schema Definition*), apresentado em W3C (2004). Com o esquema definem-se elementos (*Tags*) e regras para a composição do documento XML. Na definição de um elemento define se o seu tipo de dado. Tipos de dados customizados são criados a partir de um conjunto de tipos primitivos, predefinidos pelo padrão XSD.

A existência de um esquema permite a validação de um documento XML. O documento válido possui uma estrutura compatível com as regras definidas pelo esquema associado. Figura 12 mostra o esquema XSD do documento XML da Figura 11.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="catalogo">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="livro" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="isbn" type="xs:string"/>
                        <xs:element name="titulo" type="xs:string"/>
                    </xs:sequence>
                    <xs:attribute name="orderid" type="xs:string" use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

Figura 12 – Estrutura de um esquema XSD

Os padrões tecnológicos para realização de Web Services, são baseados na linguagem XML, conforme mostrado a seguir.

4.2.2 Comunicação SOAP

A invocação de *Web Service* requer a troca de mensagens entre o consumidor do serviço e o serviço em si. Essa troca de mensagens é padronizada pelo protocolo SOAP (*Simple Object Access Protocol*), apresentado em W3C (2007). Enquanto SOAP é um protocolo flexível, permitindo a implementação de vários modelos de comunicação, o presente trabalho descreve um cenário de comunicação RPC (*Remote Procedure Call*), conforme Curbera *et al.* (2002).

SOAP permite modelar a comunicação entre o consumidor e o serviço como uma invocação de procedimento remoto RPC. As mensagens trocadas são documentos XML para qual o protocolo SOAP define uma estrutura simples: A mensagem, chamada de envelope SOAP, contem como carga a identificação da operação invocada e os parâmetros submetidos. Opcionalmente o envelope SOAP contêm elementos de cabeçalho com meta dados, por exemplo, credenciais para identificar o autor da mensagem. Figura X mostra a estrutura básica do envelope SOAP.

Curbera *et al.* (2003) enfatiza que o transporte concreto do envelope SOAP entre os participantes da comunicação está fora do escopo desse protocolo. SOAP assume a existência de um mecanismo de transporte e é compatível com diversos protocolos, sendo HTTP (*Hypertext Transfer Protocol*) uma escolha comum. O exemplo da Figura 13 mostra os cabeçalhos da mensagem HTTP acima do envelope SOAP.

```
POST /travelservice
SOAPAction: "http://www.acme-travel.com/checkin"
Content-Type: text/xml

<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
    <!-- meta dados, como credenciais -->
  </SOAP:Header>
  <SOAP:Body>
    <!-- remote procedure call para servico web -->
  </SOAP:Body>
</SOAP:Envelope>
```

Figura 13 - Envelope SOAP com transporte HTTP

A invocação de um *Web Service* como procedimento remoto (RPC), requer a identificação do procedimento e a passagem dos parâmetros. O protocolo SOAP define que esta informação seja incluída no elemento `<body>` do envelope SOAP, sem prescreve uma estrutura particular. A estrutura da mensagem SOAP é definida pelo autor do Web Service como interface WSDL, conforme detalhado na Seção 4.2.3. Figura 14 mostra um envelope SOAP para a invocação de um serviço no padrão RPC. Observa-se a identificação da operação invocada e a passagem dos parâmetros com informação sobre seu tipo de dado.

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <m:GetFlightInfo
        xmlns:m="http://www.acme-travel.com/flightinfo"
        SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xml:xsd="http://www.w3.org/2001/XMLSchema"
        xml:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <airlineName xsi:type="xsd:string">UL</airlineName>
        <airlineNumber xsi:type="xsd:int">506</airlineName>
    </m:GetFlightInfo>
  </SOAP:Body>
</SOAP:Envelope>
```

Figura 14 – Envelope SOAP no padrão RPC

4.2.3 Descrição WSDL

A interação com um *Web Service* requer uma definição formal da sua interface. WSDL (*Web Services Description Language*), publicada em W3C (2001), é um padrão para especificar a interface de um serviço, em formato XML.

Uma especificação WSDL consiste da descrição abstrata do serviço e da sua ligação (*Binding*) a um protocolo de comunicação. A descrição abstrata nomeia as operações do serviço e associa a cada operação os dados trocados na sua invocação. A divisão da especificação em descrição abstrata e *Binding* do serviço permite expor um mesmo Web Service através de protocolos (*Bindings*) alternativos, como menciona Curbera *et al.* (2003).

Figura 15 mostra a estrutura global de um documento WSDL, cujos elementos são explicados a seguir.

```
<?xml version="1.0"?>
<definitions>

    <types>
    </types>

    <message>
    </message>

    <portType>
        <operation>
        </operation>
    </portType>

    <binding>
    </binding>

    <service>
      <port>
      </port>
    </service>

</definitions>
```

Figura 15 – Estrutura do documento WSDL

A seção `<types>` define os tipos de dados relevantes para a interação com o serviço. Com um sistema de tipo baseado no padrão XSD, WSDL permite o uso de tipos primitivos e a definição de tipos customizados.

A seção `<messages>` define os dados trocados na invocação de uma operação do serviço. Para isto nomeia os parâmetros e os associa ao seu tipo de dado.

O elemento `<operation>` define uma operação (procedimento) oferecida pelo *Web Service*. Para isto nomeia a operação e associa seus parâmetros, conforme declarados na seção `<message>`. Na operação, os parâmetros podem ter o papel de `input`, `output` ou `fault` (condição de erro).

As definições de um conjunto de operações suportados pelo *Web Service* são unidas pelo elemento `<portType>`, completando a descrição abstrata do serviço.

O elemento <binding> define o protocolo de comunicação e o formato exato das mensagem para acessar as operações de um <portType>. WSDL prevê a comunicação via SOAP, mas permite a extensão para outros protocolos. Múltiplos elementos <binding>, para um mesmo <portType>, permitem expor as operações do serviço por protocolos alternativos.

A publicação do serviço em um endereço Web específico é realizado com o elemento <port>. O elemento <port> define um ponto de contato (*endpoint*) para a comunicação com o *Web Service* por associar um <binding> a um endereço, por exemplo uma URL HTTP usada na comunicação SOAP.

Os diversos pontos de contato do *Web Service* são unidos por um elemento <service>.

A especificação da interface de um *Web Service* com um documento WSDL elimina ambiguidades na comunicação com esse serviço. Essa comunicação livre de ambiguidade possibilita a integração e composição de serviços, conforme abordado pelos capítulos 4.3 e 4.4.

4.3 Composição de *Web Services*

Uma arquitetura orientada a serviços permite a criação de processos de negócio por combinar serviços de granularidade menor. Segundo Peltz (2003), a modelagem dessa composição de serviços envolve duas perspectivas complementares.

O modelo de Coreografia descreve a interação necessária entre serviços para alcançar o objetivo da composição. Barros; Dumas e Oaks (2006) afirmam que este modelo abstrai o processo interno dos serviços participantes, mas descreve, de uma perspectiva global, a troca de mensagens entre os serviços. Restrições para a composição dos serviços, como por exemplo uma sequencia de interação obrigatória ou a exclusão mútua de interações, são aspectos do modelo de coreografia.

O modelo de Orquestração descreve a composição de serviços a partir do ponto de vista de um dos participantes da colaboração, um processo de negócio particular, conforme Peltz (2003). Devido a esta perspectiva, o modelo inclui tanto a interação do processo com *Web Services* externos, quanto as atividades internas do processo. Barros; Dumas e Oaks (2006) associam à orquestração o conceito do processo executável, conforme detalhado no capítulo 4.4.

4.4 Linguagem BPEL

WS-BPEL (*Web Service Business Process Execution Language*), abreviada como BPEL, é uma linguagem que implementa a ideia da composição de serviços. A linguagem foi especificada em Jordan *et al.* (2007).

Conforme Peltz (2003) o padrão BPEL permite a modelagem da composição de *Web Services*, tanto pela perspectiva de orquestração, quanto de coreografia. Enquanto o processo BPEL executável é um modelo de orquestração, o processo BPEL abstrato modela a coreografia de serviços.

O processo BPEL executável é interpretado por um motor BPEL, resultando na realização das atividades modeladas, incluindo a invocação de *Web Services*. A interação do processo com serviços externos baseia-se na sua interface WSDL. Chen *et al.* (2006) enfatizam esta dependência entre os padrões BPEL e WSDL.

Louridas (2008) caracteriza BPEL como linguagem de *script* baseada em notação XML com recursos para uma programação estruturada, controle de fluxo e interação bidirecional com serviços externos. O autor descreve o documento BPEL como sendo composto por 4 seções elementares, mostradas pela Figura 16.

```
<?xml version="1.0"?>
<process>

  <partnerLinks>
    <partnerLink />
  </partnerLink>

  <variables>
    <variable />
  </variables>

  <sequence>
    <!-- atividades do processo -->
  </sequence>

</process>
```

Figura 16 – Estrutura do documento BPEL

A seção `<partnerLinks>` declara os participantes externos (*Web Services*) com qual o processo interage para alcançar seu objetivo.

A seção `<variables>` declara as variáveis usadas para manter estado durante a execução do processo, incluindo valores recebidos com a invocação de serviços externos.

O tratamento de erros ocorridos no decorrer do processo é definido na seção `<faultHandlers>`.

As atividades do processo são modeladas como fluxo. Atividades sequenciais são contidos por um elemento `<sequence>`, enquanto o elemento `<flow>` representa atividades paralelas. Fluxos sequenciais e paralelos podem ser aninhados.

A linguagem BPEL possui recursos adicionais, porém irrelevantes para o escopo do presente trabalho, incluindo controle transacional, invocação assíncrona de serviços e sincronização de atividades concorrentes.

4.5 Considerações do Capítulo

O *Web Service* constitui um importante recurso arquitetural. Enquanto a Arquitetura Orientada a Serviços (SOA) é frequentemente utilizada devido ao seu potencial de integrar sistemas heterogéneos, no contexto de sistemas *multitenant*, a escolha deste padrão arquitetural deve-se à sua propriedade de flexibilidade. O próximo capítulo discute a realização de variabilidade, através da orquestração de *Web Services*, baseado nos padrões tecnológicos apresentados.

5. LINGUAGEM VxBPEL

Este capítulo discute a representação e realização de variabilidade na composição de *Web Services*. Apresenta-se a linguagem VxBPEL para esta finalidade.

5.1 Objetivo da linguagem VxBPEL

VxBPEL, apresentada por Koning *et al.* (2008), é uma extensão da linguagem BPEL para descrever variação na composição de serviços. O sistema orientado a serviço que suporta variação na orquestração pode ser reconfigurado para se adequar a requisitos alterados.

A intenção da linguagem VxBPEL é tratar variabilidade como aspecto fundamental na modelagem de um sistema. Consequentemente, ela propõe representar a informação da variabilidade explicitamente no modelo de orquestração, com recursos dedicados para esta finalidade, em vez de utilizar recursos nativos da linguagem BPEL, como o desvio condicional.

VxBPEL suporta 4 tipos de variação na composição de serviços: 1) Invocação de implementações alternativas para um mesmo *Web Service*. 2) Invocação de serviços alternativos, com interfaces distintas. 3) Variação dos parâmetros utilizados na invocação de um serviço. 4) Variação do processo de negócio.

O modelo de variabilidade implementado pela linguagem VxBPEL orienta-se no *framework* COVAMOF. VxBPEL permite representar esses conceitos no contexto da modelagem de processos.

5.2 Extensão do padrão BPEL

Como extensão, a VxBPEL introduz novos elementos à linguagem BPEL. Os elementos VxBPEL encapsulam fragmentos do processo BPEL, para definir a variação prevista na orquestração de serviços.

A Figura 17 ilustra a combinação de elementos BPEL e VxBPEL. O exemplo a seguir modela um fluxo sequencial. Contido neste contexto BPEL é um ponto de variação. O ponto de variação VxBPEL substitui uma atividade BPEL regular, no fluxo. As variantes VxBPEL por sua vez contém atividades BPEL regulares, por exemplo a invocação de um *Web Service*. Observa-se a distinção dos elementos VxBPEL por um namespace XML próprio, o `vxbpel`.

```
<sequence>
  ...
  <receive></receive>
  ...
  <vxbpel:VariationPoint>
    <vxbel:Variant>
      <xbpel:VPBpelCode>
        <invoke/>
        <assign/>
      </xbpel:VPBpelCode>
    </vxbel:Variant>
    ...
  </vxbpel:VariationPoint>
  ..
  <reply></reply>
</sequence>
```

Figura 17 – Composição de elementos VxBPEL e BPEL

Os elementos introduzidos pela extensão VxBPEL são resumidas nas Tabela 1 e Tabela 2. Tabela 1 relaciona os elementos para modelagem de um *Ponto de Variação*.

Tabela 1 – Elementos VxBPEL para representar o Ponto de Variação

`<vxbpel:VariationPoint>`	Marca uma parte do processo BPEL como sendo variável.
`<vxbpel:Variants>`	Relaciona as variantes que existem para determinado ponto de variação.
`<vxbpel:Variant>`	Declara uma variante específica.
`<vxbpel:VBpelCode>`	Encapsula as atividades BPEL que constituem a implementação da variante.

Os elementos da Tabela 2 permitem modelar um *Ponto de Variação* configurável, o recurso VxBEPL que implementa o conceito da *Relação de Realização* no framework COVAMOF. Este recurso simplifica a gestão da variabilidade distribuída em todo processo BPEL. O *Ponto de Variação* configurável associa múltiplos *Pontos de Variação* regulares, e a sua configuração determina a configuração destes pontos. Baseado neste princípio, Koning *et al.* (2008) sugerem que a ação de configuração de todo processo BPEL se limita à escolha de variantes nos *Pontos de Variação* configuráveis.

Tabela 2 – Elementos VxBPEL para representar a relação de Dependência

`<vxbpel: ConfigurableVariationPoint>`	Gerencia a configuração de múltiplos pontos de variação do mesmo processo.
`<vxbpel:Rationale>`	Comunica a intenção e restrições para a escolha de uma configuração.
`<vxbpel:Variants>`	Relaciona as variantes alternativas que existem no ponto de variação.
`<vxbpel:Variant>`	Declara uma variante. Esta se distingue da variante do ponto de variação regular (tabela x).
`<vxbpel: RequiredConfiguration>`	A implementação da variante. Consiste em regras para a configuração dos pontos de variação regidos por este ponto configurável.
`<vxbpel:VPChoices>`	Encapsula as regras de configuração, sendo uma para cada ponto de variação regido.
`<vxbpel:VPChoice>`	A configuração exigida para um ponto de variação. Consiste na seleção de uma variante.

Os *Pontos de Variação* configuráveis constituem uma informação global para o processo BPEL. Ao contrario dos *Pontos de Variação* regulares, a sua definição não ocorre no fluxo de atividades BPEL. Porém, VxBPEL introduz uma seção de configuração na raiz do documento BPEL.

Figura 18 e Figura 19 mostram a combinação dos elementos VxBPEL de forma integrada. O exemplo a seguir modela uma orquestração de serviços para um

processo de compra e importação. O processo contêm dois *pontos de variação* no fluxo das atividades BPEL. Enquanto a primeira variação consiste na troca do *Web Service* invocado para realizar uma compra, a segunda variação representa uma alteração mais abrangente no processo de importação. Em ambos os *pontos de variação* há duas variantes. Para simplificar a gestão da variabilidade, o processo contêm um *ponto de variação* configurável. Este relaciona os *pontos de variação* regulares com o nível de suporte fornecido para determinado cliente. As duas variantes desse ponto configurável correspondem a dois níveis de suporte. Ao escolher um nível de suporte, define-se qual variante será considerada nos dois *pontos de variação* regulares, configurando completamente o processo BPEL.

```
<process>
  <partnerLinks></partnerLinks>
  <variables></variables>

  <vxbpel:ConfigurableVariationPoints>
    <vxbpel:ConfigurableVariationPoint defaultVariant="standartService">
      <vxbpel:Name>Customer Service Level</vxbpel:Name>
      <vxbpel:Rationale>Choose between standart and premium customer ser
      <vxbpel:Variants>
        <vxbpel:Variant name="standartService">
          <vxbpel:RequiredConfiguration>
            <vxbpel:VPChoices>
              <vxbpel:VPChoice
                vpname="purchaseService"
                variant="standartPurchase"/>
              <vxbpel:VPChoice
                vpname="customsProcess"
                variant="standartCustoms"/>
            </vxbpel:VPChoices>
          </vxbpel:RequiredConfiguration>
        </vxbpel:Variant>
        <vxbpel:Variant name="premiumService">
          <vxbpel:RequiredConfiguration>
            <vxbpel:VPChoices>
              <vxbpel:VPChoice
                vpname="purchaseService"
                variant="expressPurchase"/>
              <vxbpel:VPChoice
                vpname="customsProcess"
                variant="shortCustoms"/>
            </vxbpel:VPChoices>
          </vxbpel:RequiredConfiguration>
        </vxbpel:Variant>
      </vxbpel:Variants>
    </vxbpel:ConfigurableVariationPoint>
  </vxbpel:ConfigurableVariationPoints>
```

Figura 18 – Exemplo de processo VxBPEL configurável (parte 1 de 2)

```
  <sequence>
    <receive></receive>
    <vxbpel:VariationPoint name="purchaseService">
      <vxbel:Variant name="standartPurchase">
        <xbpel:VPBpelCode>
          <invoke partnerLing="purchase-a"/>
        </xbpel:VPBpelCode>
      </vxbel:Variant>
      <vxbel:Variant name="expressPurchase">
        <xbpel:VPBpelCode>
          <invoke partnerLing="purchase-b"/>
        </xbpel:VPBpelCode>
      </vxbel:Variant>
    </vxbpel:VariationPoint>
    <assign></assign>
    ...
    <vxbpel:VariationPoint name="customsProcess">
      <vxbel:Variant name="standartCustoms">
        <xbpel:VPBpelCode>
          <sequence>...</sequence>
        </xbpel:VPBpelCode>
      </vxbel:Variant>
      <vxbel:Variant name="shortCustoms">
        <xbpel:VPBpelCode>
          <flow>...</flow>
        </xbpel:VPBpelCode>
      </vxbel:Variant>
    </vxbpel:VariationPoint>
    ...
    <reply></reply>
  </sequence>

</process>
```

Figura 19 – Exemplo de processo VxBPEL configurável (parte 2 de 2)

5.3 Configuração do Processo VxBPEL

Os recursos da VxBPEL permitem expressar as variações previstas em uma orquestração de serviços. A execução do processo, porém, requer que seja considerada somente uma das variantes em cada ponto de variação. VxBPEL prevê que esta configuração do processo ocorra no momento da sua instanciação. Esse

modelo de configuração permite que duas instâncias consecutivas de um mesmo processo resultam em orquestrações distintas.

A execução de um processo VxBPEL segue o modelo do padrão BPEL. Identificam-se três fases neste modelo de execução: A primeira fase consiste no carregamento do processo. Um motor de execução (BPEL *execution engine*) lê a especificação XML do processo e cria uma representação interna. A segunda fase consiste na instanciação do processo. Esta instanciação ocorre para cada ordem de execução do processo. O motor constrói uma representação executável do processo. Na terceira fase ocorre a execução propriamente dita. O motor caminha pela representação executável do processo e executa as atividades. Com a geração de uma representação executável dedicada para cada instância do processo, observa-se um isolamento entre duas ordens de execução.

Neste modelo, a configuração do processo VxBPEL ocorre na segunda fase, a instanciação do processo. Koning *et al.* (2008) descrevem a adaptação de um motor de execução BPEL para suportar a configuração, que abrange a primeira e a segunda fase.

Na fase de carregamento do processo, constrói-se a representação interna do processo, contendo elementos VxBPEL entre as atividades BPEL regulares. Paralelamente constrói-se uma representação da configuração do processo. Esta estrutura de dados armazena os *Pontos de Variação* configuráveis e a sua configuração (seleção de variantes). A seleção das variantes nos Pontos de Variação configuráveis pode mudar dinamicamente. Porém, em cada momento existe uma configuração válida para o processo.

Na fase de instanciação do processo é necessário produzir uma representação executável. Isto envolve a substituição dos elementos VxBPEL pelas atividades BPEL corretas. Esta substituição ocorre pelo *binding* dos *Pontos de Variação* presentes no processo. Para cada *Ponto de Variação* consulta-se a configuração atual dos *Pontos de Variação* configuráveis, que informa a escolha da Variante correta.

Resultado do *binding* na fase de instanciação do processo é uma representação executável, livre de elementos VxBPEL, que descreve uma variação particular da orquestração. O motor de execução BPEL irá executar a instancia do processo regularmente.

A reconfiguração do sistema ocorre em tempo de execução, ou seja sem necessidade de reiniciar o motor de execução ou recarregar o documento VxBPEL. Koning *et al.* (2008) prevêm para o motor de execução adaptado uma API (*Application Programming Interface*) que expõe a seleção das *Variantes* nos *Pontos de Variação* configuráveis.

5.4 Considerações do Capítulo

VxBPEL permite especificar orquestrações de *Web Services* com variabilidade. A representação explícita da variabilidade no modelo de orquestração e a configuração do processo em tempo de execução, fazem da VxBPEL um candidato para a realização de sistemas *multitenant,* capazes de atender aos *tenants* com requisitos variados. O próximo capítulo mostra como empregar a VxBPEL como mecanismo de variação no contexto de sistemas *multitenant*.

6. APLICAÇÃO DA VxBPEL NO MODELO MULTITENANT

Este capítulo estuda a aplicação da linguagem VxBPEL no contexto de sistemas *multitenant*. Propõe-se uma solução, centrada na VxBPEL, para suportar variação funcional entre *tenants*.

6.1. Arquitetura *Multitenant*

A aplicação da VxBPEL, como mecanismo de variação ocorre em um contexto arquitetural maior. Para estudar, como um sistema baseado na VxBPEL permite suportar variação funcional entre *tenants*, define-se uma arquitetura referência.

A arquitetura é centrada na orquestração de *Web Services.* A funcionalidade do sistema é modelada como conjunto de processos de negócio. Operações elementares são implementadas como *Web Services*. A execução dos processos de negócio resulta na invocação dos *Web Services* elementares. Esta execução pode resultar em uma orquestração distinta, para cada *tenant*, de acordo com a variação modelada, nos processos VxBPEL.

Conforme caraterizado, o núcleo da arquitetura é constituído por: 1) Um motor de execução VxBPEL 2) Um conjunto de processos VxBPEL, contendo informação de variação 3) Um conjunto de *Web Services*.

O núcleo dessa arquitetura é apoiado por elementos complementares. Para a interação do *tenant*, o sistema expõe uma interface *web* ou uma Interface de Programação de Aplicativos (API – *Application Programming Interface*), caso o *tenant* invoca as funções do sistema programaticamente. No conjunto arquitetural, a responsabilidade desta camada é capturar as requisições dos *tenants*, construir o contexto para processar cada requisição e encaminha-la no sistema.

O contexto de processamento em sistema *multitenant*, inclui, além da identidade do usuário requisitante, a informação a qual organização *tenant* este pertence. Para isto, consulta-se uma base de *tenants*, mantida pelo sistema.

Elemento central, para apoiar a execução dos processos VxBPEL é um gerenciador de configurações. O gerenciador mantém uma configuração para cada *tenan*t e permite a adição, modificação e remoção de configurações para suportar a adesão e o desligamento de *tenants*.

A Tabela 3 relaciona estes elementos, enquanto a Figura 20 mostra uma visão estrutural da arquitetura.

Tabela 3 – Elementos da arquitetura multitenant referência

Elemento	**Papel**
Processo VxBPEL	Especificar a lógica de negócio como composição de Web Services.
Motor VxBPEL	Executar a lógica de negócio por invocar Web Services.
Web Service	Implementar uma operação básica da lógico de negócio.
Interface web	Capturar a requisição do *tenant*. (interface humana).
API web	Capturar a requisição do *tenant*. (interface de programa).
Base de *tenants*	Identificar o autor da requisição.
Gerenciador de configurações	Gerenciar a configuração dos *tenants* (configuração VxBPEL).

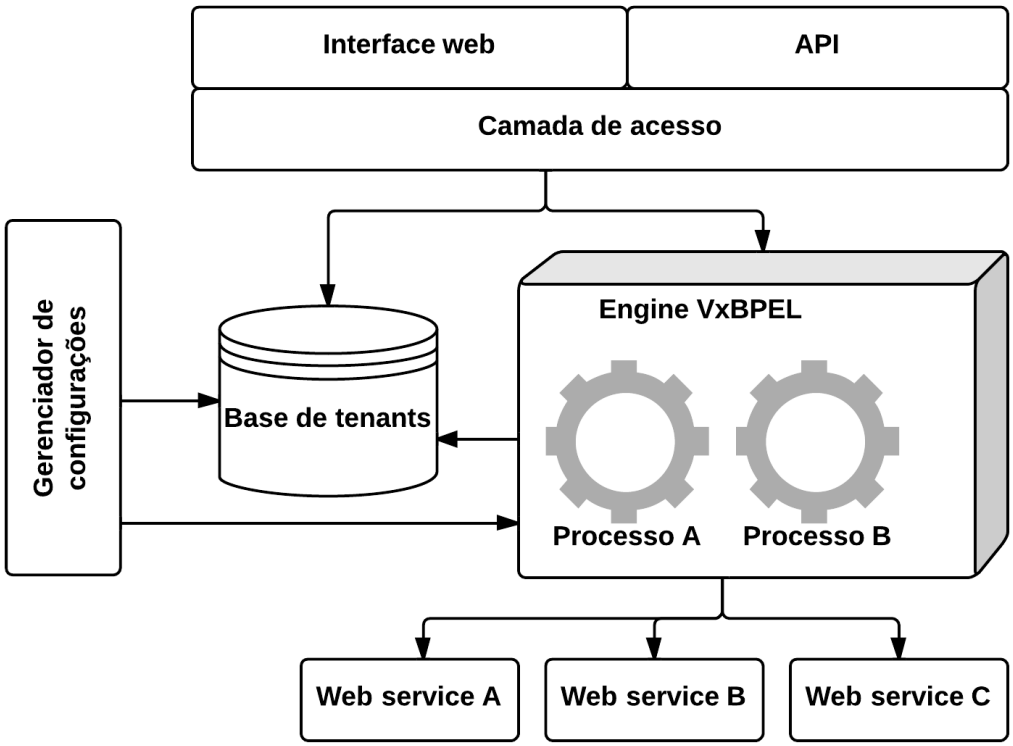

Figura 20 – Visão estrutural da arquitetura multitenant referência

O processamento de uma requisição segue um fluxo definido. O primeiro passo é autenticar e autorizar a requisição, isto é identificar o usuário autor da requisição, e verificar que este usuário tem a permissão de executar a operação requisitada. Esta autenticação e autorização é realizado na camada de acesso.

Ainda na camada de acesso ocorre o segundo passo. Identifica-se o *tenant* da requisição, por consulta à base de tenants, que mantem a associação entre cada tenant e seus usuários. Com esta informação será possível configurar o sistema corretamente, e executar a operação, conforme demandado pelos requisitos deste *tenant*.

A execução da operação de negócio é delegada, pela camada de acesso, ao motor VxBPEL. Neste componente ocorre o terceiro passo, a instanciação do processo VxBPEL, que corresponde à operação requisitada. É neste passo que o sistema é configurado especificamente para os requisitos do *tenant*, ao aplicar a sua configuração particular, na instanciação.

O quarto passo é a própria execução do processo. A operação requisitada pelo usuário é realizada, pelo motor VxBPEL, com a invocação coordenada de *Web Services*. Esta orquestração é específica para cada *tenant*, como a instância do processo é configurada de acordo com os seus requisitos, no passo anterior.

6.2. Representação de Variação

A arquitetura proposta suporta *tenants* com requisitos variados por adaptar a implementação das operações, dinamicamente, em função dos requisitos de cada *tenant*. Esta adaptação é realizada com pontos de variação nos processos VxBPEL.

O processo VxBPEL contem um ponto de variação para cada função variável. A gama de variação disponível para os *tenants*, em um determinado ponto é dada pelas variantes definidas para este ponto de variação.

A configuração de um *tenant* relaciona a cada ponto de variação a variante válida. Este mapeamento é construído em função dos requisitos do *tenant*. O sistema *multitenant* é capaz de atender a um determinado *tenant*, se é possível traduzir os seus requisitos, em uma seleção de variantes. A adaptação do sistema aos requisitos do *tenant*, em tempo de execução, consiste na aplicação das variantes selecionadas pelo *tenant*, em cada ponto de variação.

6.3 Considerações do Capítulo

Apresentou-se, através de um arquitetura referência, como a VxBPEL pode ser utilizada como mecanismo de variação em um sistema *multitenant*. A abordagem constitui uma solução sistemática para suportar, com um mesmo sistema, *tenants* com requisitos variados.

7. ANÁLISE DOS RESULTADOS

Este capítulo constitui uma analise crítica da aplicação da VxBPEL como mecanismo de variação em sistemas *multitenant*, conforme sugerido no capítulo anterior.

7.1. Requisitos de Variabilidade *Multitenant*

A aplicação da VxBPEL, como elemento central da arquitetura de um sistema *multitenant*, deve garantir que este sistema suporta o cenário de variabilidade previsto. Baseado no estudo sobre sistemas *multitenant*, no Capítulo 2, concebe-se um cenário de variabilidade, cujas caraterísticas são capturadas por quatro requisitos, R1 – R4:

R1: O sistema possui uma capacidade de variação fixa. A definição da variabilidade, suportada pelo sistema, ocorre em tempo de desenvolvimento.

R2: O sistema atende, com uma única instância de execução, a múltiplos *tenants*. Cada *tenant* tem requisitos variados, dentro da gama de variação suportada pelo sistema.

R3: A especificação da configuração, demandada por um *tenant* em particular, ocorre no momento da sua adesão ao sistema *multitenant*. O sistema suporta essa adesão sem interrupção de serviço.

R4: O sistema recebe um fluxo constante e imprevisível de requisições, originadas por diversos *tenants*. O sistema atende a cada requisição de acordo com os requisitos do *tenant*.

7.2. Aplicação da VxBPEL no Contexto *Multitenant*

Para questionar a aplicação da VxBPEL, conforme sugerida no capítulo anterior, analisa-se a sua adequação em relação aos requisitos R1 – R4. Considera-se tanto a linguagem VxBPEL, quanto a sua implementação (o motor de execução).

A VxBPEL atende ao **requisito R1**. Em tempo de desenvolvimento define-se uma capacidade de variação fixa para o sistema. Conforme a arquitetura referência, implementa-se esta variação com um ou mais processos VxBPEL. Ou seja, em tempo de desenvolvimento, define e implementa-se uma gama de variação, suportada pelo sistema, quando em execução.

Introduzir uma variação nova, não prevista no desenvolvimento original, significa alterar a capacidade de variação do sistema. Com a arquitetura referência, esta alteração implica na modificação manual dos processos VxBPEL. Neles, a variação é implementada por novos pontos de variação ou por variantes adicionais de pontos existentes. A alteração dos processos VxBPEL requer o seu recarregamento pelo motor de execução. Após este procedimento, que ocorre em tempo de desenvolvimento, a variação nova está disponível, para ser selecionada dinamicamente, em tempo de execução.

A VxBPEL atende ao **requisito R2**. A arquitetura referência prevê que as operações do sistema sejam modeladas como processos VxBPEL. Dinamicamente, cada requisição de processamento, implica na instanciação de um processo. A possibilidade de poder aplicar uma configuração distinta na instanciação do processo VxBPEL, permite atender, com uma mesma instância do sistema, a *tenants* com requisitos variados.

A VxBPEL atende ao **requisito R3**. VxBPEL permite especificar a configuração a ser considerada na instanciação de um processo com os pontos de variação configuráveis. Para preparar a adesão de um *tenant* novo, define-se para este uma seleção de variantes nos pontos de variação configuráveis. Conforme apresentado no Capítulo 5, a respeito de VxBPEL, o ponto de variação configurável age como mecanismo de configuração de alto nível, determinando a configuração de múltiplos pontos de variação dependentes, distribuídos no processo. A VxBPEL fornece então

um recurso que simplifica a definição de uma configuração, e assim a adesão de um novo *tenant*.

O requisito R3 exige ainda, que após especificar a configuração de um novo *tenant*, esta informação seja introduzida ao sistema. O motor de execução proposto por Koning *et al.* (2008) prevêm que se informe uma nova configuração ao sistema, sem necessidade de reimplantação ou reinicialização. Para este fim, o motor expõe uma Interface de Programação de Aplicativos (API).

A VxBPEL atende só parcialmente ao **requisito R4**. Como detalhado na avaliação do requisito R2, VxBPEL permite considerar uma configuração distinta a cada instanciação do processo. Conforme a arquitetura referência, o sistema identifica para cada requisição recebida o *tenant* que a originou. Isto permite que o sistema processe cada requisição de acordo com os requisitos individuais do *tenant*, mesmo que as requisições cheguem ao sistema em sequência ou de forma concorrente. Como o sistema cria instâncias isoladas do processo VxBPEL, para processar requisições concorrentes, consegue assumir configurações distintas em um mesmo instante.

Porém, a implementação da VxBPEL, proposta por Koning *et al.* (2008), não é suficiente para um cenário *multitenant*. Koning *et al.* (2008) limitam o motor de execução a manter somente uma configuração por vez. O sistema *multitentant* requer que o motor mantenha N configurações distintas simultaneamente, para poder aplicar a configuração correta ao instanciar o processo para o *tenant* requisitante.

7.3. Adaptação da Tecnologia VxBPEL

Uma adaptação do motor VxBPEL, que elimine esta restrição de manter somente uma configuração única por vez, permitirá utilizar a VxBPEL como mecanismo de variação em um cenário *multitenant*.

Para manter múltiplas configurações de processo, o motor de execução VxBPEL deverá ser modificado em três pontos:

- A estrutura de armazenamento interna do motor deve ser alterada para armazenar uma configuração por *tenant* e para relacionar cada *tenant* com a sua configuração. O armazenamento das configurações em um TAD (Tipo Abstrato de Dados) *Vetor Associativo*, permite manter múltiplas configurações e associar cada configuração ao seu *tenant*, quando se considera o identificador do *tenant* como chave, do par chave-valor.

- A interface API de configuração do motor deve ser estendida para receber em cada operação (adicionar, alterar e remover uma configuração) o identificador do *tenant*. Assim o motor consegue executar a operação solicitada sobre a configuração correta, localizada no *Vetor Associativo*, através do identificador de *tenant*.

- Igualmente, a interface do motor para execução do processo deve ser estendida para receber na solicitação o identificador do *tenant*. Com este identificador, o motor localiza a configuração correta para o *tenant*, a aplica na instanciação do processo e assim consegue executar um processo personalizado para o requisitante.

7. 4 Considerações do Capítulo

A análise revelou que a VxBPEL é, principalmente, um mecanismo viável para implementar variabilidade em um sistema *multitenant.* Identificou-se uma limitação da implementação original da linguagem, que inibe a sua aplicação no contexto *multitenant.* Porém, argumentou-se que é viável modificar o motor de execução, para superar esta limitação, possibilitando a aplicação da VxBPEL como mecanismo de variação, conforme sugerida.

8. CONSIDERAÇÕES FINAIS

Este capítulo apresenta os resultados da pesquisa. Destacam-se as contribuições do trabalho e identificam-se pontos com potencial para um aprofundamento maior, em pesquisas futuras.

8.1 Contribuições do Trabalho

Problema alvo desta pesquisa é o suporte de variabilidade em sistemas *multitenant*, ou seja a capacidade de atender, com um mesmo sistema, a *tenants* com requisitos variados. A literatura sobre sistemas *multitenant* indica como solução uma arquitetura orientada a serviços, ou seja, um sistema cuja variabilidade é implementada com uma variação dinâmica na composição de *Web Services*. Porém a literatura abstrai da questão de como implementar, de forma sistemática, esta variação na composição.

Este trabalho levantou a hipótese de que a linguagem VxBPEL, uma extensão do padrão BPEL para orquestração de *Web Services*, será adequada para preencher a lacuna do mecanismo de implementação, deixada pela literatura. Para examinar esta hipótese desenvolveu-se, no capítulo 6, uma arquitetura referência para sistemas *multitenant*, baseada na VxBPEL e mostrou-se, no capítulo 7, que esta arquitetura suporta um cenário típico de variabilidade *multitenant*, quando se adapta o motor de execução. Assim, o trabalho forneceu evidências que a aplicação da VxBPEL, para esta finalidade, é viável.

Para contextualizar a questão do suporte à variabilidade, este trabalho abordou tópicos complementares. Este estudo complementar permitiu identificar quatro elementos para a construção sistemática de um sistema que atende a *tenants* com requisitos variados:

- O **primeiro elemento** é a definição explicita de um escopo de variabilidade para o sistema. O estudo do Capítulo 2 revelou que o suporte de variabilidade implica em duas consequências opostas: enquanto um suporte amplo a requisitos

variados levanta o potencial comercial do sistema, implica também em uma complexidade e um custo maior. Considerando estes fatores opostos, a pesquisa sugere como critério para definir o escopo de variabilidade do sistema o resultado econômico;

- O **segundo elemento** é a modelagem explícita da variabilidade do sistema. Enquanto o escopo define quais variações o sistema deve suportar, o modelo de variabilidade detalhou cada uma destas variações. A pesquisa do Capítulo 3 mostrou como o *framework* COVAMOF permite formalizar a variação do sistema com pontos de variação e variantes. A organização hierárquica dos pontos de variação como relações de realização foi identificada como um recurso de modelagem capaz de reduzir a complexidade e o custo para atender a uma variação de requisitos maior;

- Após especificação e modelagem formal da variabilidade, o **terceiro elemento** é suportar a variação sistematicamente, com uma estratégica arquitetural adequada. A pesquisa do Capitulo 2, sobre sistemas *multitenant*, e do Capítulo 4, sobre a abordagem SOA, indicou a orquestração de *Web Services* como abordagem arquitetural; e

- O **quarto** e último **elemento** é uma tecnologia concreta para implementar a ideia de realizar a variação dinâmica do sistema com a orquestração de *Web Services*. O estudo do Capítulo 5 afirma o potencial da tecnologia VxBPEL para implementar variação. Os Capítulos 6 e 7 confirmaram a aplicabilidade da VxBPEL no contexto *multitenant*.

8.2 Trabalhos Futuros

A presente pesquisa permitiu identificar novas questões sobre a construção de sistemas *multitenant* variáveis. Ainda sem resposta detalhada, estas questões são candidatos para pesquisas futuras.

Devido ao impacto que o planejamento do escopo de variabilidade tem para o sucesso de um projeto *multitenant*, identifica-se a demanda por um modelo sobre o custo-benefício da variabilidade de um sistema. Para este estudo recomenda-se, à priori, a VxBPEL como mecanismo de variação. Assim será possível fazer previsões de custo mais precisas e desenvolver um modelo de custo-benefício que oriente o planejamento do escopo de variabilidade.

Uma segunda pesquisa diz respeito aos padrões que se podem identificar na variabilidade demandada de diversos sistemas. Sistemas corporativos pertencem frequentemente a um domínio recorrente como finanças e logística. Ao identificar uma recorrência na variabilidade, exigida nestes domínios comuns, será possível elaborar padrões para suportar esta variabilidade. Este padrões poderão existir tanto no nível da especificação COVAMOF, como na implementação VxBPEL, e orientar no desenvolvimento de um sistema *multitenant* variável.

REFERÊNCIAS

BACHMANN, F.; BASS, L. Managing Variability in Software Architectures. **SIGSOFT Software Engineering Notes**, New York, ACM, v. 26, n. 3, p. 126 – 132, 2001

BARROS, A.; DUMAS, M. e OAKS P. Standards for Web Service Choreography and Orchestration: Status and Perspectives. **Business Process Management Workshops. Lecture Notes in Computer Science**, Berlin, Springer, v. 3812, p. 61 – 74, 2006

BASS, L.; CLEMENTS, P. e KAZMAN, R. **Software Architecture in Practice**, 3ª edição, 2012, Upper Saddle River: Addison-Wesley, 2012

BENLIAN, A.; HESS, T. e BUXMANN, P. Drivers of SaaS-adoption–an empirical study of different application types. **Business & Information Systems Engineering**, v. 1, n. 5, p. 357 - 369, Berlin, Springer, 2009

BEZEMER, C. P. e ZAIDMAN, A. Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare? In: **Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)**, New York, ACM, p. 88 – 92, 2010

CHEN, L.; WASSERMANN, B.; EMMERICH, W. e FOSTER, H. Web service orchestration with BPEL. In: **Proceedings of the 28th international conference on Software engineering (ICSE '06)**, New York, ACM, p. 1071 – 1072, 2006

CHONG, F.; CARRARO, G. e WOLTER, R. Multi-tenant data architecture. **MSDN Library**, Microsoft Corporation, 2006.

COHEN, S. e KRUT, R. **Managing variation in services in a software product line context**. Pittsburgh, Carnegie-Mellon University, Software Engineering Institute, 2010.

CURBERA, F.; DUFTLER, M.; KHALAF, R.; NAGY,W.; MUKHI, N. e WEERAWARANA S. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. **IEEE Internet computing**, v. 6 n. 2, p. 86 – 93, 2002

GHADDAR, A.; TAMZALIT, D. e ASSAF, A. Decoupling variability management in multi-tenant SaaS applications. In: **Proceedings of The 6th IEEE International Symposium on Service Oriented System Engineering (SOSE 2011)**, p. 273 – 279, IEEE, 2011

HARMS, R. e YAMARTINO, M. The economics of the cloud. **Microsoft whitepaper**, Microsoft Corporation, 2010.

KAPLAN, J. M. How SaaS is overcoming common customer objections. **Cutter Consortium: Sourcing and Vendor Relationships, Advisory Service, Executive Update**, v. 8, n. 9, 2008

KONING, M.; SUN, C. A.; SINNEMA, M. e AVGERIOU, P. VxBPEL: Supporting variability for Web services in BPEL. **Information and Software Technology**, v. 51, n. 2, p. 258 - 269, Elsevier**,** 2009

LEE, J. e KOTONYA, G. Combining service-orientation with product line engineering. **Software**, v. 27, n. 3, p. 35 - 41, IEEE, 2010

LOURIDAS, P. Orchestrating web services with BPEL. **Software**, v. 25, n. 2, p. 85 - 87, IEEE, 2008.

MIETZNER, R.; METZGER, A.; LEYMANN, F. e POHL, K. Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In: **Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems**, p. 18 - 25 IEEE Computer Society, 2009

MOENS, H.; TRUYEN, E.; WALRAVEN, S.; JOOSEN, W.; DHOEDT, B. e DE TURCK, F. Developing and managing customizable Software as a Service using feature model

conversion. In: **Network Operations and Management Symposium (NOMS)**, p. 1295 - 1302, IEEE, 2012

JORDAN, D. e EVDEMON, J. (chairs). **Web services business process execution language version 2.0**, OASIS standard, 2007.

PELTZ, C. Web services orchestration and choreography. **Computer**, v. 36, n. 10, p. 46 - 52, IEEE, 2003.

POHL, K.; BÖCKLE, G. e VAN DER LINDEN, F. **Software product line engineering**, Berlin, Springer, 2005.

SENGUPTA, B.; ROYCHOUDHURY, A. Engineering multi-tenant software-as-a-service systems. In: **Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems**, p. 15 - 21, ACM, 2011

SINNEMA, M.; DEELSTRA, S.; NIJHUIS, J. e BOSCH, J. Covamof: A framework for modeling variability in software product families. **Software Product Lines**, p. 197 – 213, Berlin, Springer, 2004

SINNEMA, M. e DEELSTRA, S. Classifying variability modeling techniques. **Information and Software Technology**, v. 49, n. 7, p. 717 - 739, Elsevier, 2007

Park, S.; Sugumaran, V. e Kang, K. C. Software product line engineering. **Communications of the ACM**, v. 49, n. 12, p. 29 - 32, ACM, 2006.

SUN, W.; ZHANG, X.; GUO, C. J.; SUN, P. e SU, H. Software as a service: Configuration and customization perspectives. In: **Congress on Services Part II, 2008**, p. 18 – 25, IEEE, 2008

W3C. **Web Services Description Language (WSDL) 1.1**. W3C, 2001

W3C. **XML Schema Part 0: Primer (Second Edition)**. W3C, 2004

W3C. **SOAP Version 1.2 Part 0: Primer (Second Edition)**. W3C, 2007

W3C. **Extensible Markup Language (XML) 1.0** W3C, 2008

WANG, Z.; GUO, C. J.; GAO, B.; SUN, W.; ZHANG, Z. e AN, W. H. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In: **IEEE International Conference on e-Business Engineering (ICEBE 2008)**, p. 94 - 101, IEEE, 2008