UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

PROVENDO SEMÂNTICA PARA IMPLEMENTAÇÃO DE ASPECTOS EM BPM

Hércules Sant'Ana da Silva José

Orientadores
Claudia Cappelli
Flávia Maria Santoro

RIO DE JANEIRO, RJ - BRASIL
MARÇO de 2017

PROVENDO SEMÂNTICA PARA IMPLEMENTAÇÃO DE ASPECTOS EM BPM

Hércules Sant'Ana da Silva José

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Prof. Claudia Cappelli, D.Sc. – UNIRIO

Prof. Flávia Maria Santoro, D.Sc – UNIRIO

Leonardo Guerreiro Azevedo, D.Sc. – IBM Research – Brasil; UNIRIO

Prof. Thaís Vasconcelos Batista, D.Sc. – UFRN

RIO DE JANEIRO, RJ – Brasil
MARÇO de 2017

Catalogação informatizada pelo(a) autor(a)

José, Hércules Sant'Ana da Silva
J83 Provendo semântica para implementação de aspectos em BPM / Hércules Sant'Ana da Silva José. -- Rio de Janeiro, 2017.
90 f.

Orientadora: Claudia Cappelli.
Coorientadora: Flavia Maria Santoro.
Dissertação (Mestrado) - Universidade Federal do Estado do Rio de Janeiro, Programa de Pós-Graduação em Informática, 2017.

1. Orientação a aspectos. 2. Processos de negócio. 3. Semântica. 4. Descoberta automática de serviços. 5. WSMO. I. Cappelli, Claudia, orient. II. Santoro, Flavia Maria, coorient. III. Título.

Em gratidão ao Deus que eu amo. Louvado seja seu Santo nome.

Agradecimentos

Agradeço primeiramente a Deus por permitir estar com saúde e disposição para caminhar por esta estrada e permitir alcançar mais um degrau em minha vida acadêmica.

Meus sinceros agradecimentos aos meus pais, familiares e amigos por me educarem no caminho que me conduziu a me tornar uma pessoa de bem, favorecendo assim chegar onde cheguei nas áreas pessoal, social, profissional, acadêmica e espiritual.

Meus agradecimentos aos meus colegas de trabalho da UFRJ, que a atenção e compreensão de todos vocês me permitiu alcançar mais esta etapa na minha vida. Sei que estou em débito com vocês, mas ao longo do tempo esta dívida será paga com minha dedicação e comprometimento nas tarefas diárias.

Agradecimento especial para minhas orientadoras Claudia Cappelli e Flávia Santoro por me acompanharem nesse período do Mestrado. Sei que não fui aquele aluno excepcional, mas consegui atender as tarefas dentro das expectativas. Agradeço também aos professores do PPGI que ministraram as disciplinas do Mestrado e do corpo de servidores da UNIRIO que me atendeu para as questões administrativas.

Meus agradecimentos para meus colegas de turma, tanto do Mestrado quanto do Doutorado, pois compartilhar suas experiências me fizeram continuar seguindo em frente, e que em hipótese alguma eu devia desistir.

E um agradecimento especial para Eliane Capra e Carlos Matta. Graças a incentivo recebido de fazer faculdade em um tempo que eu não tinha este ideal para minha vida, permitiu-me alcançar a carreira de analista de TI, com bons conhecimentos em análise e desenvolvimento de software, e especializado na linguagem Java.

JOSÉ, Hércules Sant'Ana da Silva. **Provendo semântica para implementação de aspectos em BPM**. UNIRIO, 2017. 90 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

Em desenvolvimento de software, interesses transversais como segurança, auditoria, controle de acesso, autenticação, *logging*, persistência, transação, tratamento de erro entre outros, podem ser modularizados usando-se o paradigma de orientação a aspectos. Em processos de negócio os interesses transversais veem sendo tratados para reduzir a complexidade na visualização, aumentar o reuso e melhorar a manutenibilidade dos modelos. Na literatura encontramos técnicas que utilizam aspectos nas fases de modelagem e implementação de processos de negócio, contudo essas técnicas adotam diferentes representações semânticas, dificultando a integração entre as fases dentro do ciclo de vida de BPM. Este trabalho propõe um método de descoberta de serviços capaz de selecionar aspectos que atendam aos objetivos estabelecidos na fase de modelagem, tornando-os mais flexíveis e adaptáveis em tempo de execução. Foi realizada uma prova de conceito para verificar o método proposto e os artefatos produzidos. Além disso, foi realizado um experimento como forma de validação da proposta. Os resultados obtidos mostram que a definição de um objetivo operacional torna os aspectos mais flexíveis e adaptáveis, permitindo aos especialistas de negócio se concentrem na modelagem do aspecto sem necessariamente se preocupar com a implementação, uma vez que a mesma será descoberta durante a execução do processo.

Palavras-chave: Orientação a aspectos, processos de negócio, semântica, descoberta automática de serviços, modelagem de aspectos em processo de negócio, wsmo

ABSTRACT

In software development, crosscutting concerns, such as security, audit, access control, authentication, logging, persistence, transaction, error handling, among others, can be modularized using aspect-oriented paradigm. In business process, aspects have been used to reduce visualization complexity, increase reuse and improve maintainability of the models. In literature, we found techniques that address aspects of the modeling and implementation phases of business process, however these techniques adopt different semantic representations, hindering the integration between these phases in BPM lifecycle. This work proposes an discovery method of services capable of select aspects that achieve the objectives established in the modeling phase, thus becoming them more flexible and adaptable at runtime. A proof of concept was performed to verify the proposed method and artifacts produced. In addition, an experiment was conducted as a proposal validation. The results show that the definition of an operational objective makes the aspects more flexible and adaptable, allowing the business specialists to concentrate on the modeling of the aspect without necessarily worrying the implementation, since it will be discovery during the execution of the process.

Keywords: Aspect orientation, business process, semantics, automatic discovery of services, aspect-oriented modeling in business process, wsmo

Sumário

1. INTRODUÇÃO 1
1.1 MOTIVAÇÃO 1
1.2 CARACTERIZAÇÃO DO PROBLEMA 2
1.3 ENFOQUE DA SOLUÇÃO 7
1.4 METODOLOGIA DE PESQUISA 7
1.5 ORGANIZAÇÃO DO TRABALHO 8
2. FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS 9
2.1 *BUSINESS PROCESS MANAGEMENT* (BPM) OU GESTÃO DE PROCESSOS DE NEGÓCIOS 9
2.2 O PARADIGMA DE ORIENTAÇÃO A ASPECTOS 11
2.3 O PARADIGMA DE ORIENTAÇÃO A ASPECTOS EM BPM 15
2.4 WEB SERVICE MODELING ONTOLOGY (WSMO) 21
2.5 ONTOLOGIA, WEBSERVICE, GOAL E MEDIADOR DO WSMO 23
2.6 WEB SERVICE EXECUTION ENVIRONMENT 32
2.7 TRABALHOS RELACIONADOS 34
3. INTEGRAÇÃO SEMÂNTICA EM AO-BPM 39
3.1 COMPORTAMENTO DO ASPECTO 39
3.2 OBJETIVO OPERACIONAL 42
3.3 FORMALIZAÇÃO DO OBJETIVO NO AO-BPM 43
3.4 ESQUEMA CONCEITUAL 45
3.5 ARQUITETURA E ARTEFATOS 48
4. AVALIAÇÃO DA PROPOSTA 54
4.1 PROVA DE CONCEITO 54
4.2 EXPERIMENTO 61
4.3 ANÁLISE DOS RESULTADOS E DISCUSSÃO 68
5. CONCLUSÕES 72
5.1 CONTRIBUIÇÕES DO TRABALHO 72
5.2 LIMITAÇÕES 73
5.3 TRABALHOS FUTUROS 73
REFERÊNCIAS BIBLIOGRÁFICAS 75

Lista de Figuras

FIGURA 1 – PROCESSO "SEND ARTICLES TO REVIEWERS" (CAPPELLI ET AL., 2009) 3
FIGURA 2 – "TIME ASPECT" E "COMPLIANCE ASPECT" (CHARFI; MÜLLER; MEZINI, 2010) 4
FIGURA 3 – ASPECTO PARA MONITORAR O TEMPO DE EXECUÇÃO (CHARFI; MEZINI, 2007) 5
FIGURA 4 – ONTOLOGIA BASE PARA O DOMÍNIO DE ORIENTAÇÃO A ASPECTOS EM PROCESSOS DE NEGÓCIO (BASTOS; SANTORO; SIQUEIRA, 2014) 6
FIGURA 5 – VISÃO GERAL DA METODOLOGIA CIENTÍFICA 8
FIGURA 6 – CICLO DE VIDA BPM (DUMAS ET AL., 2013) 10
FIGURA 7 – PROBLEMA DE ENTRELAÇAMENTO DE CÓDIGO (JALALI, 2011) 11
FIGURA 8 – PROBLEMA DE ESPALHAMENTO DE CÓDIGO (JALALI, 2011) 12
FIGURA 9 – TRATANDO INTERESSES TRANSVERSAIS EM POA (JALALI, 2011) 13
FIGURA 10 – RELACIONAMENTO ENTRE UMA CLASSE E UM ASPECTO, ADAPTADO DE GARCIA (2010) 15
FIGURA 11 – AOPML – MODELO CONCEITUAL (CAPPELLI ET AL., 2009) 17
FIGURA 12 – EXEMPLO DE PROCESSO USANDO AO-BPMN (CAPPELLI ET AL., 2010) 18
FIGURA 13 – EXEMPLO DE PROCESSO EM AO4BPMN (CHARFI; MÜLLER; MEZINI, 2010) 19
FIGURA 14 - O PROCESSO DE PACOTE DE VIAGEM (CHARFI; MEZINI, 2007) 20
FIGURA 15 – EXEMPLO DE ASPECTO EM AO4BPEL (CHARFI; MEZINI, 2007) 21
FIGURA 16 – CONCEITOS BASE DO WSMO 22
FIGURA 17 - ONTOLOGIA DO WSMO E SEUS ELEMENTOS 24
FIGURA 18 - EXEMPLO DE DECLARAÇÃO DE PROPRIEDADES NÃO FUNCIONAIS 24
FIGURA 19 – EXEMPLO DE ONTOLOGIA DESCRITO EM WSML (FENSEL; KERRIGAN; ZAREMBA, 2008) 26
FIGURA 20 – WEB SERVICE DO WSMO E SEUS ELEMENTOS 26
FIGURA 21 - EXEMPLO DE CAPACIDADE DE *WEB SERVICE* DESCRITO EM WSML (FENSEL; KERRIGAN; ZAREMBA, 2008) 28
FIGURA 22 - GOAL DO WSMO E SEUS ELEMENTOS 29
FIGURA 23 - TIPOS DE NOÇÕES DE CORRESPONDÊNCIAS USADOS NO PROCESSO DE DESCOBERTA (KELLER ET AL., 2004). 30
FIGURA 24 - EXEMPLO DE GOAL DESCRITO EM WSML 31
FIGURA 25 - ARQUITETURA WSMX (BUSSLER ET AL., 2005) 32
FIGURA 26 - FUNCIONALIDADES SUPORTADAS POR DIFERENTES ABORDAGENS AO-BPM USANDO O *FRAMEWORK* DE AVALIAÇÃO (JALALI, 2014) 37
FIGURA 27 - REPRESENTAÇÃO DE UM INTERESSE TRANSVERSAL 40
FIGURA 28 - REPRESENTAÇÃO DO COMPORTAMENTO DO ASPECTO 41
FIGURA 29 - REPRESENTAÇÃO DA DEFINIÇÃO DE COMPORTAMENTO DO ASPECTO 42

FIGURA 30 - DEFINIÇÃO DOS OBJETIVOS DOS ASPECTOS PELO ESPECIALISTA DE NEGÓCIO . 46
FIGURA 31 - DEFINIÇÃO DO WEB SERVICE E DE SUA DESCRIÇÃO SEMÂNTICA PELO DESENVOLVEDOR 47
FIGURA 32 - ESQUEMA DE INTERAÇÃO ENTRE O BPMS E O SEE ATRAVÉS DO COMPONENTE CUSTOMIZADO 48
FIGURA 33 - ARTEFATOS IMPLEMENTADOS PARA HABILITAR *WEAVING* DINÂMICO NO AMBIENTE YAWL, OBTIDO DE JALALI ET AL. (2013) 49
FIGURA 34 - EXTENSÃO DO POINTCUT EDITOR PARA SUPORTAR OBJETIVOS OPERACIONAIS 50
FIGURA 35 - ESQUEMA DE INTERAÇÃO ENTRE O YAWL E O WSMX ATRAVÉS DO GOALSERVICE 51
FIGURA 36 - VISÃO DE EXECUÇÃO DA ARQUITETURA PROPOSTA 53
FIGURA 37 - PROCESSO "*SEND ARTICLES TO REVIEW*" E O ASPECTO "*LOG INFORMATION*", ADAPTADO DE CAPPELLI ET AL. (2009) 55
FIGURA 38 - ATRIBUIÇÃO DO *ASPECT SERVICE* A TAREFA PROCEED, E DO *GOAL SERVICE* NA TAREFA "*LOG INFORMATION*", NO EDITOR YAWL 56
FIGURA 39 - ONTOLOGIA DE LOGGING, BASEADA EM BASTOS ET AL. (2014), ESCRITO EM WSML 57
FIGURA 40 - OBJETIVO "MAILLOGGOAL", ESCRITO EM WSML 58
FIGURA 41 - DESCRIÇÃO SEMÂNTICA DO WEB SERVICE "MAILLOG", ESCRITO EM WSML59
FIGURA 42 - *POINTCUT EDITOR* ILUSTRANDO A MUDANÇA DE OBJETIVO DURANTE A EXECUÇÃO DO ASPECTO 60
FIGURA 43 - *WSMX MONITOR* ILUSTRANDO A MUDANÇA DE OBJETIVO DURANTE A EXECUÇÃO DO ASPECTO 61
FIGURA 44 - EXEMPLO DE ARQUIVO WSML USADO COMO MODELO E OWL-S CONVERTIDO PARA WSML 64
FIGURA 45 - EXEMPLO DE OBJETIVO (GOAL) CONSTRUÍDO PARA SIMULAÇÃO 66
FIGURA 46 - RESULTADO DA SELEÇÃO DE WEB SERVICES DO 1º CENÁRIO 69
FIGURA 47 - IMPACTO DA QUANTIDADE DE SERVIÇOS PARA O PROCESSO DE DESCOBERTA 70

Lista de Tabelas

TABELA 1 - ELEMENTOS CONCEITUAIS DO WSMO PRESENTES NAS ONTOLOGIAS 24
TABELA 2–ELEMENTOS DE DEFINIÇÕES DA CAPACIDADE DE UM SERVIÇO WSMO 27
TABELA 3 - DESCOBERTA E INVOCAÇÃO DO SERVIÇO BASEADO NO OBJETIVO OPERACIONAL 51
TABELA 4 - RESUMO DA DEFINIÇÃO DOS OBJETIVOS (GOALS) PARA SIMULAÇÃO 65
TABELA 5 - RESUMO DA EXECUÇÃO DA SIMULAÇÃO 67
TABELA 6 - RESUMO DAS EXECUÇÕES BEM-SUCEDIDAS PARA O 2º E 3º CENÁRIO 70
TABELA 7 - OBJETIVOS ATINGIDOS POR CADA CENÁRIO 71

Lista de Nomenclaturas

AO-BPMN	Aspect-Oriented Business Process Modeling Notation
AO4BPEL	Aspect-Oriented for BPEL
AO4BPMN	Aspect-Oriented Extension for BPMN
AOPML	Aspect Oriented Process Modeling Language
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
OCL	Object Constraint Language
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
POA	Programação Orientada a Aspectos
QVT	Query/View/Transformation
SEE	Semantic Execution Environment
SGBD	Sistema de Gerenciamento de Banco de Dados
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SWS	Semantic Web Service
URI	Universal Resource Identifier
WSMF	Web Service Modeling Framework
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Modelling eXecution environment
XML	eXtensible Markup Language
YAWL	Yet Another Workflow Language

1. Introdução

O objetivo deste capítulo é apresentar a motivação do trabalho, a caracterização do problema abordado, o enfoque de solução, uma descrição resumida da solução proposta, a metodologia de pesquisa utilizada e a organização da dissertação.

1.1 Motivação

Os processos de negócio consistem em um conjunto coordenado de atividades visando alcançar os objetivos organizacionais, sendo importantes para a eficiência e eficácia das organizações (WESKE, 2012). Os processos de negócio são representados graficamente por modelos de negócios com o objetivo de fornecer análise de negócios através de uma profunda compreensão do processo (AGUILAR-SAVÉN, 2004).

Algumas notações definem elementos de modularização, tais como processos e atividades, de modo que os modelos gerados sejam suficientemente simples para facilitar a sua compreensão por pessoas, enquanto são detalhados o suficiente para captar a complexidade dos processos de negócios (JALALI, 2011).

No entanto, esses elementos não são adequados para representar de forma modular os interesses transversais. Podemos entender como interesses transversais aqueles que podem ser separados do modelo do processo principal sem prejudicar sua compreensão e reutilizados em outras partes do mesmo processo, ou mesmo em outros processos. Esses interesses tornam-se espalhados e entrelaçados em várias partes do mesmo processo ou de diferentes processos, resultando na dificuldade de entendimento do modelo, e dificultando o reuso das partes comuns e sua própria manutenibilidade.

Em desenvolvimento de software, as funcionalidades implementadas geralmente executam uma determinada regra de negócio (função principal), mas ele também contém funções secundárias que suportam as funções principais em sua execução. O código correspondente das funções secundárias muitas vezes fica espalhado ao longo de toda aplicação. Kiczales et al. (1997) tratou este código secundário como interesses transversais, chamando-o de aspectos. Todo aspecto é modularizado, permitindo seu

desenvolvimento separado, e aumentando as chances de reuso e evolução. Assim, interesses transversais identificados na literatura de desenvolvimento de software, como segurança, auditoria, controle de acesso, autenticação, *logging*, persistência, controle transacional, controle de erro, entre outros podem ser tratados usando aspectos.

O paradigma de orientação a aspectos tem várias vantagens, como a redução do código espalhado, a transparência das responsabilidades de cada módulo, e a independência dos módulos. A evolução da aplicação é mais fácil, pois o baixo acoplamento permite mudanças na aplicação sem mudar as funcionalidades principais. Além disso, existe mais reusabilidade do código e facilidade para desenvolver e manter o sistema diminuindo o custo de integrar novas funcionalidades.

O paradigma de orientação a aspectos inspirou a comunidade científica de *Business Process Management* (BPM) com o mesmo objetivo de modularizar interesses transversais espalhados ao longo da especificação do software (CAPPELLI et al., 2009). Interesses transversais em BPM tentam resolver os problemas causados pelo espalhamento e entrelaçamento em modelos de processos de negócio. O problema do espalhamento é a repetição do interesse dentro dos modelos de processos. O problema do entrelaçamento significa que qualquer mudança no interesse deve ser refletida em todos os processos que usam aquele interesse (CAPPELLI et al., 2010; CHARFI; MÜLLER; MEZINI, 2010; COLLELL, 2012; JALALI et al., 2015; WANG; ZHU; LIANG, 2005). Assim, em BPM, interesses transversais são definidos como elementos espalhados e entrelaçados dentro dos modelos de processos. Eles podem ser parte do processo principal, contendo regras de negócio comuns a vários processos, por exemplo fórmulas contábeis e estatísticas, e não somente recursos secundários como na Programação Orientada a Aspectos (POA) (SANTOS et al., 2011).

Usando aspectos em BPM intenciona-se aumentar a modularização dos modelos de processos de negócio. Assim, a identificação, modelagem e implementação de aspectos facilita a manutenção, entendimento, a modelagem em si, e o reuso de partes do processo, consequentemente melhorando o gerenciamento do processo.

1.2 Caracterização do Problema

Na modelagem de processos, os aspectos e seus componentes são representados por elementos visuais que facilitam o entendimento do modelo pelos especialistas do negócio e pelas partes interessadas (*stakeholders*). Já na implementação de processos, os aspectos estão ligados a soluções técnicas descritas em uma linguagem de programação e componentes de software, que muitas vezes podem ser oferecidos como bibliotecas ou serviços.

Na literatura, encontramos duas diferentes técnicas que tratam orientação a aspectos em processos de negócio, de acordo com Santos et al. (2012): a primeira técnica trata aspectos na fase de implementação, como mostrado por Charfi e Mezini (2007); e a segunda técnica utiliza os conceitos de aspectos na fase de modelagem, como mostrado por Cappelli et al. (2010) e Charfi et al. (2010).

Para exemplificar as ideias de aspectos na fase de modelagem de processos, na Figura 1 mostramos o processo "*Send articles to reviewers*" (CAPPELLI et al., 2009). Neste processo, as atividades relacionadas ao conceito transversal de *logging* foram modularizadas em um aspecto. O aspecto "*Log Information"* provê no modelo as informações necessárias para representar um conjunto de atividades que registram as operações feitas ao longo do processo de revisão de artigos acadêmicos. As setas indicam as atividades onde o aspecto será inserido durante a execução do processo.

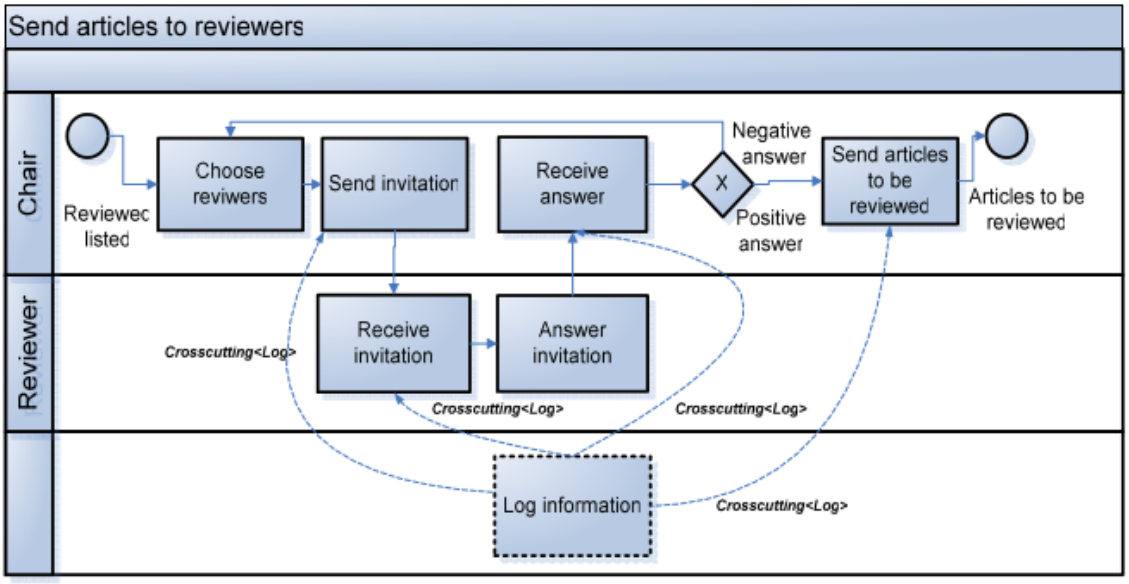


Figura 1 – Processo "*Send articles to reviewers"* (CAPPELLI et al., 2009)

Semelhantemente, em Charfi et al. (2010) temos a representação de aspectos na fase de modelagem. Na Figura 2, temos dois exemplos de aspectos: o primeiro, o "*Time Aspect*", mede o tempo de execução de uma atividade no processo principal; o segundo, o "*Compliance Aspect*" adiciona atividades de verificação de conformidade para os processos de buscas de voos e busca de vagas em hotéis, de modo a atender os critérios de viagem definidos pelo cliente. Ambos os aspectos são representados usando elementos da notação BPMN.

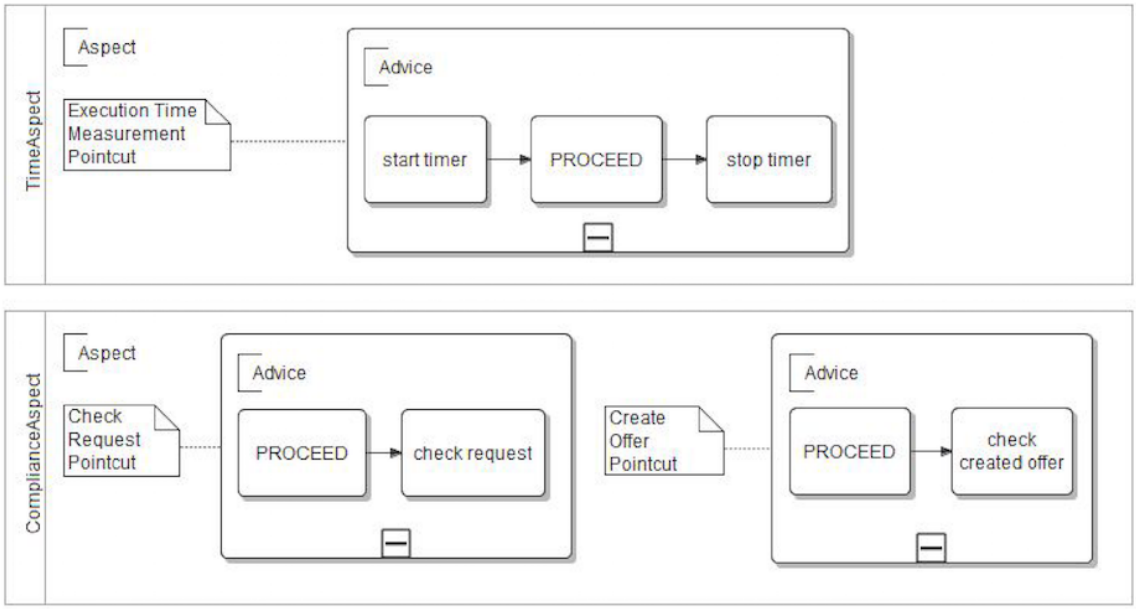


Figura 2 – "*Time Aspect*" e "*Compliance Aspect*" (CHARFI; MÜLLER; MEZINI, 2010)

Já em Charfi e Mezini (2007), temos as ideias de representação de aspectos na fase de implementação. Neste trabalho, a linguagem de composição de serviços BPEL é estendida para suportar a modularização por aspectos, de modo que um serviço composto obtém novos recursos durante sua execução sem impactar nas funcionalidades existentes. A Figura 3 mostra um aspecto de monitoramento chamado "*PerformanceMonitor*" para medir o tempo de execução de certa atividade do processo, no exemplo as atividades "*findAFlight*" e "*findARoom*", ambas definidas na *tag* "*pointcut*". Temos a invocação do *Web Service* "*AuditingWS*", definido pela *tag* "*invoke*", para registrar o início e o fim da execução da atividade. Por se tratar de um aspecto que atua ao redor da atividade, definido na *tag* "*advice*", temos duas invocações do "*AuditingWS*", uma antes e outra depois da *tag* "*proceed*".

```
<aspect name="PerformanceMonitor">
 <partnerLinks>
  <partnerLink name="AuditingWS" partnerLinkType="AuditingPLT"
            myRole="caller" partnerRole="measurer"/>
 </partnerLinks>
 <variables>
  <variable name="startTimerRequest" messageType="startTimerInput"/>
  <variable name="startTimerResponse" messageType="startTimeOutput"/>
  <variable name="stopTimerRequest" messageType="stopTimerInput"/>
  <variable name="stopTimerResponse" messageType="stopTimeOutput"/>
 </variables>
 <pointcutandadvice>
  <pointcut name="monitored activities" contextCollection="true">
    //invoke[@operation="findAFlight"] | //invoke[@operation="findARoom"]
  </pointcut>
  <advice type="around">
   <sequence>
    <assign>
     <copy>
           <from variable="ThisJPActivity" part="name"/>
           <to variable="startTimerRequest" part="activityName"/>
         </copy>
        </assign>
    <invoke partnerLink="AuditingWS" portType="AuditingPT"
          operation="startTimer" inputVariable="startTimerRequest"
          outputVariable="startTimerResponse"/>
    <proceed/>
    <assign>...</assign>
    <invoke partnerLink="AuditingWS" portType="AuditingPT"
          operation="stopTimer" inputVariable="stopTimerRequest"
          outputVariable="stopTimerResponse"/>
   </sequence>
  </advice>
 </pointcutandadvice>
</aspect>
```

Figura 3 – Aspecto para monitorar o tempo de execução (CHARFI; MEZINI, 2007)

Analisando estas abordagens, notamos que elas possuem diferenças semânticas na representação de aspectos e de seus elementos relacionados, dificultando prover uma integração entre as fases no ciclo de vida BPM. Enquanto na fase de modelagem faz-se o uso da notação BPMN e de outros recursos gráficos para representar os aspectos e seus elementos a fim de facilitar a compreensão dos analistas de negócio e *stakeholders*, na fase de implementação os aspectos são ligados a soluções técnicas descritas em uma linguagem de programação e por componentes de software, como a linguagem Java ou XML. Comparando a representação dos aspectos entre a Figura 2 e Figura 3, não encontramos dificuldades em correlacionar os elementos presentes do modelo com os elementos presentes na implementação, como *aspect*, *advice*, *pointcut* e *proceed*. Porém, ao comparar a Figura 1 e Figura 3, encontramos dificuldades em correlacionar as setas que saem do aspecto "*Log Information*" com um dos elementos do aspecto "*Performance Monitor*".

Além disso, as representações de aspectos na fase de modelagem não contêm recursos que facilitem a associação com a respectiva implementação ou que viabilizem minimamente realizar este vínculo, por exemplo um elemento de anotação padronizado para informar detalhes sobre a implementação vinculada ou que será vinculada à atividade ou ao processo. É o caso do aspecto "*Log Information*" da Figura 1, a notação utilizada não contém elementos claros que permitam vincular a atividade com uma possível implementação em virtude da diferença de objetivos, técnicas e vocabulário usados na

notação.

Com base nisso, a questão de pesquisa que norteia este trabalho é: *como diminuir a lacuna existente entre as representações de aspectos nas fases de modelagem e implementação dentro do ciclo de vida BPM?*

Bastos et al. (2014) evidenciaram o problema existente na representação de aspectos em processos de negócios, e estenderam uma ontologia de aspectos (BERG; CONEJERO; CHITCHYAN, 2005) para tratar de aspectos no domínio de processo de negócio (Figura 4) de modo a permitir a documentação e desenvolvimento de aspectos nas fases de modelagem, configuração e implementação. De acordo com a ontologia proposta, os autores estabeleceram uma representação de comportamento do aspecto: o processo (*Process*), o código base (*BaseCode*) e objetivo (*Goal*). Os autores enfatizam a classe *Goal* da ontologia, visto que o aspecto deve ter um objetivo operacional que expressa o comportamento desejado de um interesse transversal dentro do processo de negócio.

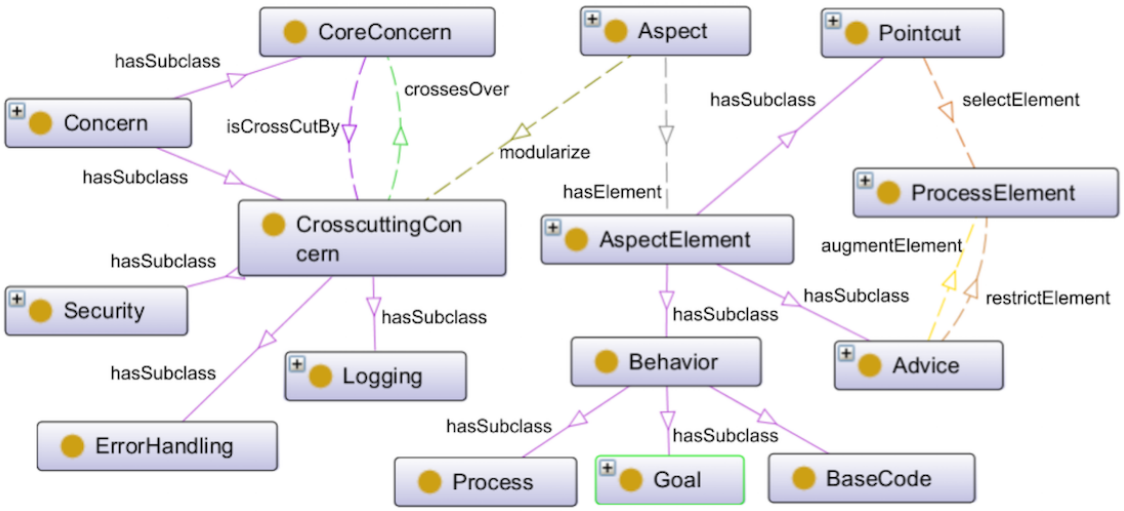


Figura 4 – Ontologia base para o domínio de orientação a aspectos em processos de negócio (BASTOS; SANTORO; SIQUEIRA, 2014)

O conceito de objetivo é o mesmo utilizado por Santos et al. (2011). O uso de objetivos operacionais como parte da descrição do aspecto durante a modelagem de processos permite que o design e código base sejam interligados durante a execução do processo. Além disso, ao definir um objetivo (*Goal*) a ser alcançado, temos a possibilidade de escolher uma implementação para o aspecto, sem ter que saber detalhes prévios da sua implementação.

Assim, nossa hipótese de pesquisa é: *se um objetivo operacional escrito usando uma linguagem formal é definido em um aspecto na fase de modelagem, então ele pode ser vinculado a uma implementação também escrita em linguagem formal, de modo a cumprir com as intenções dos especialistas de negócio.*

1.3 Enfoque da Solução

A questão dos problemas semânticos entre modelagem e implementação de aspectos fica mais evidente quando os especialistas do negócio desconhecem informações a respeito da implementação que pode ser usada em um determinado aspecto que trata requisito não funcional. Tomando como exemplo do aspecto de *logging*, os especialistas sabem que os dados trocados durante a execução do processo serão armazenados e registrados em um banco de dados, mas eles não sabem como esta operação será realizada, ou se ele estará de acordo com os objetivos estabelecidos no processo. Na Figura 1, os especialistas de negócio esperam que a implementação do aspecto de *logging* esteja alinhada com os objetivos estabelecidos no processo, mas eles não se preocupam em como esta atividade será efetivamente realizada durante a execução do processo.

Uma vez separado do processo, um requisito não funcional aspectualizado torna-se um módulo independente, e pode conter sua própria implementação encapsulada em um *Web Service*, por exemplo. Os desenvolvedores que implementam estes aspectos para atender a determinados objetivos, não tem conhecimento das intenções dos especialistas em negócios, assim a implementação dos aspectos pode não satisfazer os objetivos definidos para o processo. Além disso, a falta de subsídios para associar cada elemento do aspecto com os elementos de implementação obriga os desenvolvedores a conhecer de antemão a notação utilizada e realizar um *mapping* entre os elementos para transformar o processo modelado em um processo executável.

1.4 Metodologia de Pesquisa

A metodologia científica empregada neste trabalho baseou-se na metodologia aplicada em Brandão (2015). O trabalho foi conduzido em quatro estágios. No primeiro estágio, foi definido o problema de pesquisa e a hipótese. Além disso, foi escolhido como abordagem de solução estender o trabalho de Bastos et al. (BASTOS; SANTORO; SIQUEIRA, 2014) e formalizar o conceito de objetivo (*goal*). No segundo estágio, foi realizada uma pesquisa para obter a fundamentação teórica, buscando técnicas existentes ou inspiração para ser usado na solução criada. Na etapa seguinte, a solução foi projetada e implementada. No terceiro estágio, foi realizada uma prova de conceito para uma avaliação inicial através da comparação de resultados de um exemplo da literatura. Neste caso, usou-se o exemplo do aspecto de *logging*, o mesmo usado por Bastos et al. (2014). Na quarta etapa, foi realizado um experimento simulando a execução de um processo com alguns aspectos. Este experimento teve como objetivo avaliar o método proposto. Os resultados de ambos os estudos (prova de conceito e experimento) são apresentados no capítulo correspondente. A Figura 5 ilustra os estágios da metodologia e suas respectivas

etapas.

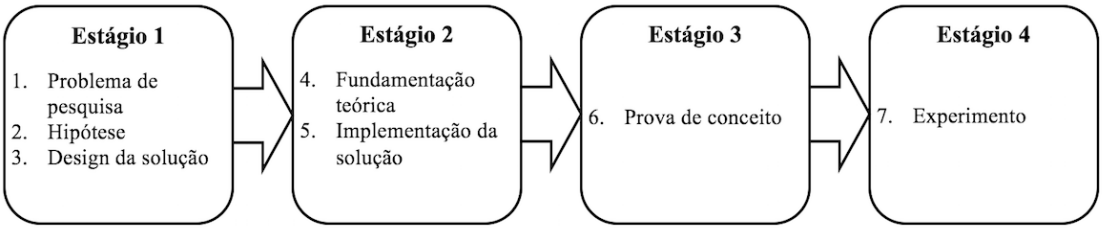


Figura 5 – Visão geral da metodologia científica

1.5 Organização do Trabalho

Este trabalho foi organizado no seguinte formato: O Capítulo 2 apresenta os conceitos teóricos abordados nesta dissertação, além de apresentar os trabalhos relacionados sobre o problema de pesquisa.

No Capítulo 3, é detalhada a proposta da solução, definindo os conceitos sobre a semântica do objetivo operacional do aspecto e apresentando uma arquitetura onde este conceito pode ser aplicado.

No Capítulo 4, avalia-se a solução proposta através de prova de conceito e um experimento, além de análise dos resultados e sua comparação com trabalhos relacionados.

No Capítulo 5 são relacionadas as conclusões, contribuições, limitações e ameaças e trabalhos futuros desta pesquisa.

2. Fundamentação teórica e trabalhos relacionados

O objetivo deste capítulo é apresentar a fundamentação teórica utilizada nesta dissertação e os trabalhos relacionados ao problema de pesquisa apresentado no Capítulo 1.

2.1 *Business Process Management* (BPM) ou Gestão de Processos de Negócios

De acordo com Davenport (1994), um processo é um conjunto simples de atividades estruturadas e medidas projetadas para resultar em um produto específico para um consumidor ou mercado em particular. Assim, um processo é uma ordenação específica de atividade de trabalho ao longo do tempo e espaço, com um início, um fim, entradas e saídas claramente identificadas (DAVENPORT, 1994).

Weske (2012) define processo de negócio como "um conjunto de atividades que são realizadas de forma coordenada em um ambiente técnico e organizacional. Essas atividades conjuntamente realizam um objetivo de negócio. Cada processo de negócio é promovido por uma única organização". Um processo de negócio é descrito por um ou mais procedimentos que juntos atingem um objetivo do negócio. A execução de um processo tem condições de início e fim bem definidos, e podem combinar procedimentos manuais e automáticos (HOLLINGSWORTH; HAMPSHIRE, 1993).

O gerenciamento de processos de negócios (do inglês *Business Process Management* – BPM) são conceitos, métodos e técnicas que suportam o projeto, administração, configuração, implementação e análise de processos de negócios. A base do gerenciamento de processos de negócio é a representação explícita dos processos com suas atividades e as restrições existentes entre eles. BPM suporta análise, melhoria e implementação dos processos (WESKE, 2012).

Pesquisas na área resultaram em vários métodos, técnicas e ferramentas para suportar o ciclo de vida BPM e suas fases, como projeto, implementação, gerenciamento e análise dos processos operacionais. Como exemplo de ciclo de vida BPM, a Figura 6 apresenta o ciclo proposto por Dumas et al. (2013), que é amplamente adotado na literatura atual de

BPM.

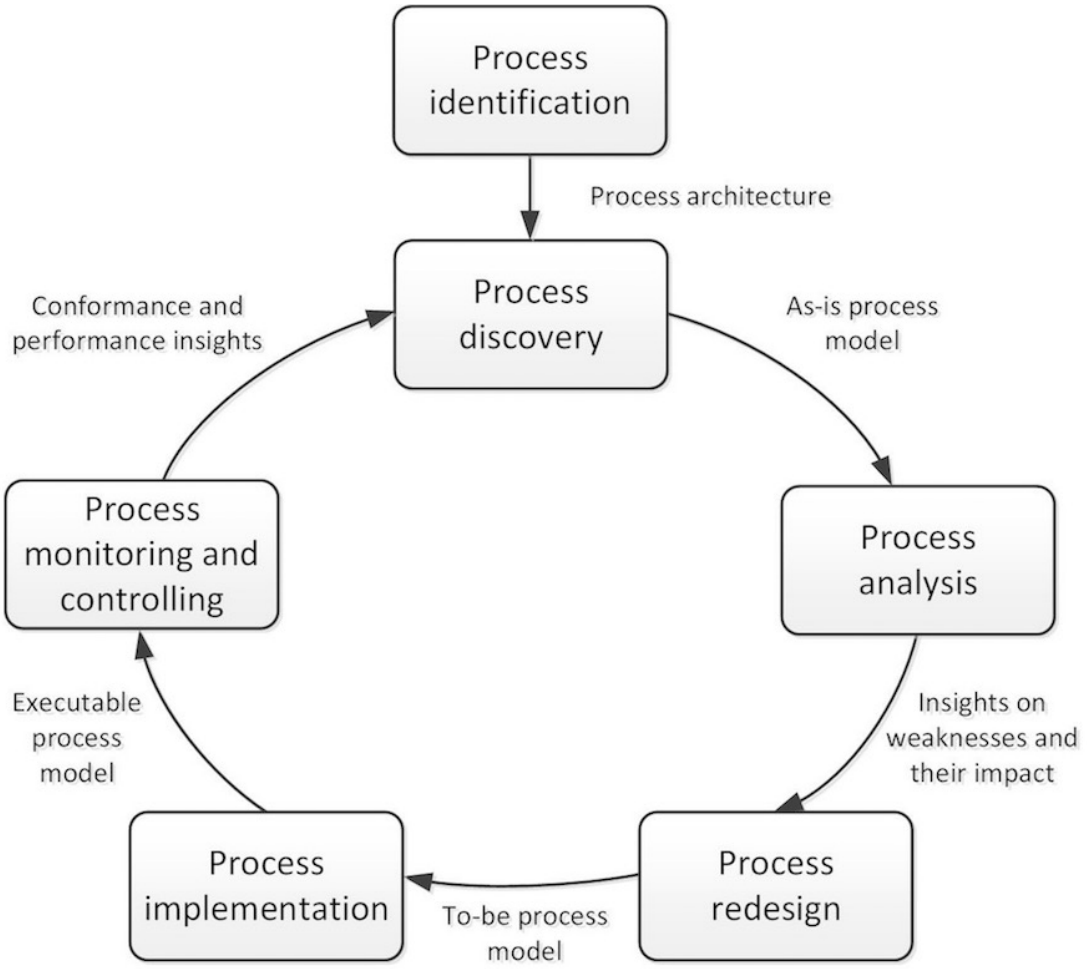


Figura 6 – Ciclo de vida BPM (DUMAS et al., 2013)

Na primeira fase, Identificação do Processo (*Process Identification*), os processos importantes são identificados, delimitados e relacionados uns com os outros. O resultado é uma nova arquitetura de processos ou a atualização da arquitetura de processos da organização. Na fase de Descoberta do Processo (*Process Discovery*), cada processo é documentado, resultando em um modelo *As-is*. Em seguida, na fase de Análise do Processo (*Process Analysis*), as questões relacionadas ao processo *As-is* são documentadas e, se possível, quantificadas por indicadores de desempenho. O resultado desta fase é a percepção das deficiências do processo e seu impacto, por exemplo, nos indicadores de desempenho. Na próxima fase, o Redesenho do Processo (*Process Redesign*), as possíveis mudanças relacionadas às questões identificadas na fase anterior são analisadas e comparadas, observando os indicadores de processo. Desta fase, obtemos o modelo *To-be* do processo. Na fase de Implementação do Processo (*Process Implementation*), as mudanças identificadas na fase de redesenho são planejadas e implementadas. A implementação do processo abrange dois aspectos: a gestão de mudança na organização e a automação de processo. A primeira preocupa-se com a forma como os participantes trabalham inseridos no processo organizacional e a segunda para o

desenvolvimento e implementação de sistemas de tecnologia da informação. Uma vez que o processo redesenhado está em execução, a fase de Monitoramento e Controle do Processo (*Process Monitoring and Controlling*) coleta dados relevantes da execução do processo para analisar o quão bem o desempenho do processo em execução está em relação às medidas de desempenho e metas. Gargalos, erros recorrentes e desvios são identificados e corrigidos. Novas questões surgem e isso pode exigir que todo o ciclo seja repetido.

2.2 O paradigma de orientação a aspectos

Separação de interesses, também conhecido como princípio "dividir e conquistar", tem sido uma estratégia para lidar com a complexidade em software. Podemos citar dois exemplos de aplicação deste princípio na área de sistemas de informação: o primeiro é a gerência separada dos dados de uma aplicação, realizada por um SGBD; o segundo é a separação da lógica principal do negócio de interesses acessórios, como segurança e privacidade. A ideia de separar certos interesses e apresentá-los como aspectos individuais ligados ao interesse principal foi aplicada pela primeira vez na programação, e foi chamada de Programação Orientada a Aspectos (KICZALES et al., 1997).

Programação Orientada a Aspectos (POA) é um paradigma que aborda o problema de complexidade no domínio da programação. Este problema é tratado separando os interesses principais de uma aplicação dos interesses transversais que normalmente ficam espalhados ao longo do código da aplicação (LADDAD, 2003). O paradigma também soluciona dois problemas comuns das abordagens tradicionais de programação: o espalhamento e o entrelaçamento de código.

Tradicionalmente, cada funcionalidade do programa é escrita em módulos. Cada módulo contém o código que trata tanto da lógica de negócio quanto de interesses acessórios, como controle de transação, segurança e controle de exceções. Entrelaçamento de código refere-se ao fenômeno onde os módulos da aplicação contém o código de diferentes interesses (Figura 7).

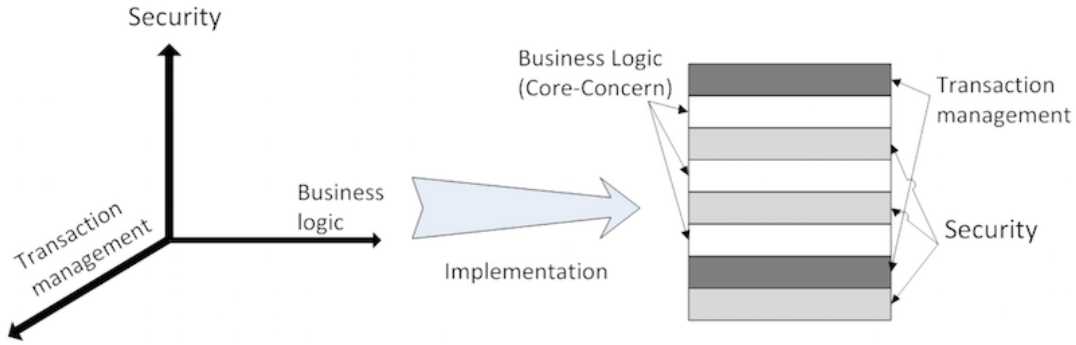


Figura 7 – Problema de entrelaçamento de código (JALALI, 2011)

Em outro método de implementação de interesses em uma aplicação, interesses transversais são implementados em módulos separados, mas cada funcionalidade da aplicação pode invocar estes módulos. Espalhamento de código refere-se ao fenômeno de implementar o mesmo interesse transversal dentro de diferentes funcionalidades da aplicação ou em outras aplicações (LADDAD, 2003). Na Figura 8, temos um exemplo de espalhamento de código. A mesma implementação do interesse de segurança (*Security Concern*) está presente nos módulos ATM, *Accounting* e *Database*.

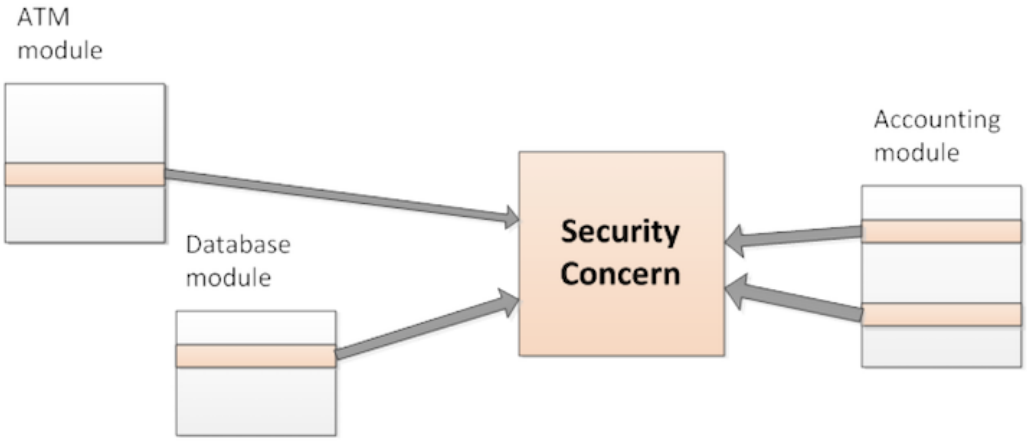


Figura 8 – Problema de espalhamento de código (JALALI, 2011)

Existem alguns problemas que surgem ao seguir estes métodos de programação. Primeiro, é difícil aplicar uma mudança em um aspecto. Usando o exemplo da Figura 7, se o gerenciamento de transação necessitar ser alterado, todo o código que trata este interesse deverá ser encontrado e alterado em todo o programa. Isto significa gastar recursos e tempo para aplicar uma mudança em um interesse. Portanto, mudanças são custosas. Segundo, se uma parte do código em um módulo é esquecida por um programador, o programa se torna inconsistente. Assim, estes métodos tornam o programa menos confiável e consistente. Terceiro, partes de código devem ser repetidas em novos módulos se eles implementarem interesses relevantes. Consequentemente, esses métodos de programação não suportam um bom grau de reusabilidade, portanto menos produtividade pode ser alcançada. Por último, mas não menos importante, é impossível rastrear um interesse em especial porque os interesses estão espalhados por todo o código.

Como consequência do código espalhado e entrelaçado, os softwares se tornam menos rastreáveis, reutilizáveis, com menos qualidade e produtividade, tornando sua evolução mais difícil (LADDAD, 2003).

POA propõe implementar interesses como módulos individuais de software e uni-los ao interesse principal para resolver os problemas de espalhamento e entrelaçamento (Figura 9). Para permitir a separação de interesses transversais, POA implementa um conjunto de construtos (CHARFI; MEZINI, 2007). A descrição destes construtos é dada

a seguir.

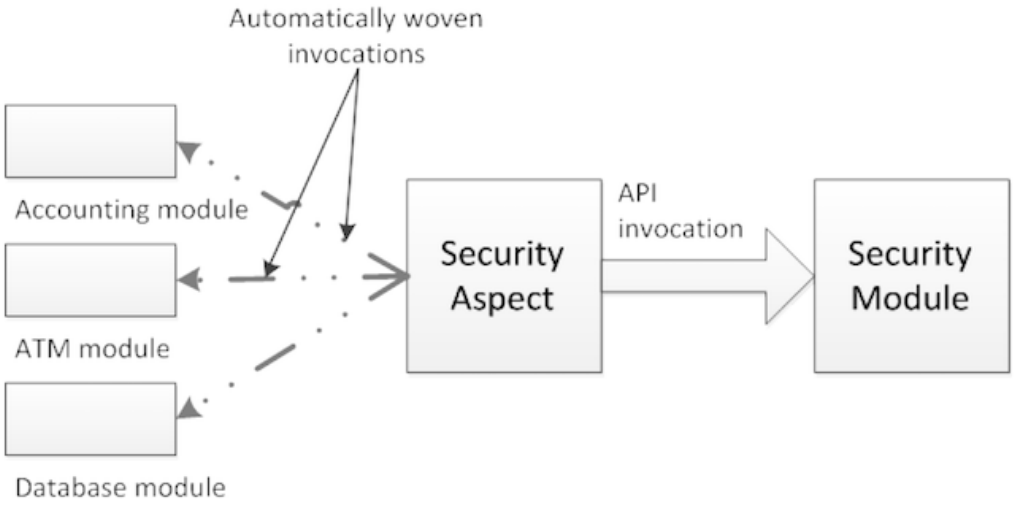


Figura 9 – Tratando interesses transversais em POA (JALALI, 2011)

- *Join points*: São definidos como pontos identificáveis na aplicação em execução (LADDAD, 2003). Esses pontos habilitam a adição de um ou de vários interesses transversais à funcionalidade principal, podendo ser vinculados a campos, métodos ou tipo.

- *Pointcut*: Uma linguagem que habilita a definição e seleção dos *join points* relevantes na aplicação em execução. Para exemplificar, um interesse que trata de *logging* pode ser necessário ser executado quando mais de 500 reais são sacados de uma conta. A linguagem de *pointcut* suporta a definição desta condição que um interesse em particular poderá ser executado.

- *Advice*: Um construto que habilita adicionar ou alterar o comportamento da funcionalidade principal; esse construto possui tarefas que representam o comportamento de um interesse transversal, como segurança. Cada *advice* pode ser executado antes, ao redor, ou depois dos *join points* monitorados. Para cada *join point* existe um *advice* que será executado.

- *Weaving*: Um mecanismo para alterar a estrutura estática da funcionalidade principal usando a execução dos interesses transversais. O mecanismo de *weaving* executa os *advices* para cada *join point* monitorado que foram definidos pela linguagem de *pointcut*.

- *Aspect*: Um módulo que contém um módulo transversal é chamado de aspecto. Aspecto contém *pointcuts* e *advices*. No exemplo da Figura 8, o interesse de segurança pode ser implementado em um aspecto. Esse aspecto contém um *pointcut* e um *advice*. O *pointcut* especifica qual *join point* que o *advice* deve

ser executado, e se o código que trata de segurança deve ser executado antes, durante ou após ao *join point*.

Ainda sobre o mecanismo de *weaving*, existem dois tipos de *weaving* (PAWLAK; SEINTURIER; RETAILLÉ, 2005):

- *Weaving* estático: Compila o código do aspecto com o código da classe principal e produz o código da aplicação. Linguagens orientadas a objetos como Java produzem *bytecodes* do código fonte em tempo de compilação. Portanto, aspectos podem ser incorporados (*weaved*) tanto no código fonte quanto no *bytecode* para produzir o *bytecode* final. A vantagem do *weaving* estático é sua rapidez e facilidade.

- *Weaving* dinâmico: É o processo de *weaving* feito em tempo de execução. O mecanismo é responsável por orquestrar a execução da classe principal e do aspecto em tempo de execução, redirecionando o fluxo de execução da classe principal para o aspecto. A vantagem do *weaving* dinâmico é que provê uma integração dinâmica do aspecto com a aplicação. Aspectos podem ser adicionados e removidos da aplicação sem a necessidade de interromper a execução do programa.

A Figura 10 ilustra uma classe Java usando os elementos de aspectos definidos pela linguagem AspectJ (LADDAD, 2003). A classe Java "*Hello*" tem dois *join points*: os métodos "*sayHello*" e "*sayMessage*". Eles serão interceptados pelo aspecto "*AspectDeclaration*", que contém o *pointcut* "*hello*" referenciando todos os métodos da classe "*Hello*". Este *pointcut* tem um *advice*, que define que o método "*hello*" deve ser executado após a execução dos *join points* encontrados (GARCIA, 2010). Um texto deve ser mostrado no console do sistema. A combinação do código Java e do código do aspecto é um exemplo do mecanismo de *weaving*.

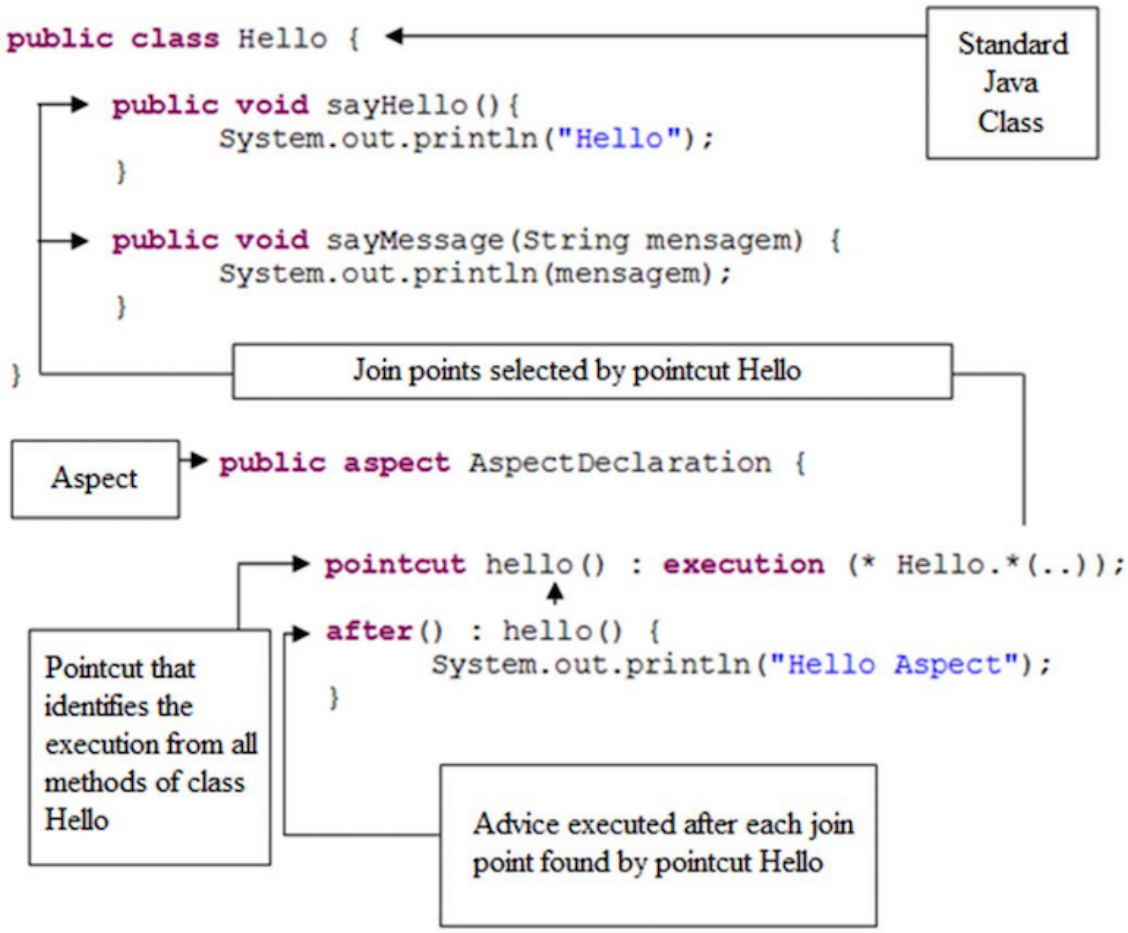


Figura 10 – Relacionamento entre uma classe e um aspecto, adaptado de Garcia (2010)

2.3 O paradigma de orientação a aspectos em BPM

Inspirado pelas ideias de POA, interesses transversais têm ganhado visibilidade na comunidade de BPM. Recentemente vários trabalhos vêm sendo realizados para implementar as ideias de POA no domínio de processos de negócio visando tratar os problemas de entrelaçamento e espalhamento nos modelos de processo (CAPPELLI et al., 2010; CHARFI; MÜLLER; MEZINI, 2010; COLLELL, 2012; JALALI et al., 2015; WANG; ZHU; LIANG, 2005). Os elementos espalhados e entrelaçados podem ser parte do processo (exemplo, regras de negócios) ou funcionalidades secundárias, como registro de *logging* e segurança.

A modularização de processos através de aspectos tem como objetivo trazer os mesmos benefícios da programação orientada a aspectos. Modularização de processo provê um melhor entendimento do processo porque agrupa as atividades pelos seus objetivos e permite o reuso destes módulos entre modelos de outros processos que compartilham os mesmos "interesses", incluindo recursos chave como regras de negócio.

Cappelli et al. (2009) propõem uma metalinguagem denominada *Aspect Oriented Process Modeling Language* (AOPML). AOPML é baseada em uma proposta semelhante

ao nível de requisitos, descrita por Da Silva (2005), e segue uma estratégia simétrica, onde os interesses transversais são representados da mesma forma que outros interesses, a diferença está no relacionamento que define a composição transversal. Uma relação entre um interesse transversal e um interesse principal é representado por uma relação "transversal". Os mecanismos usados para inserir e processar as regras depende dos construtos da AOPML.

AOPML é composto por: (i) um modelo de processo de negócio descrito usando qualquer linguagem para processos de modelagem, como BPMN (OMG, 2011), *i-star* (i*) (Yu, 2011; Ramos et al., 2007), ou EPC (Keller, 1998); (ii) um modelo central que descreve as relações transversais, e um modelo de *join point* que define como o modelo de processo principal e o modelo de componentes, no nosso caso um modelo de processo de negócio, são compostos e quais elementos deste modelo podem estar envolvidos em uma relação transversal. Estruturalmente, o AOPML consiste das seguintes partes (Figura 11):

a) Componentes da Linguagem de Modelagem de Processos - define os elementos que compõem a linguagem de modelagem de processos, por exemplo eventos, atividades, objetos de dados, entre outros.

b) Modelo Básico - descreve a relação transversal que especifica como um elemento (o elemento transversal) impacta nos outros elementos. Ambos os elementos são componentes da Linguagem de Modelagem de Processos.

c) Modelo de *Joinpoints* - define como o modelo principal e os elementos da linguagem de modelagem de processos estão relacionados e quais elementos da linguagem de processos podem estar envolvidos em uma relação cruzada. No caso de linguagens de modelagem, atividades são geralmente o alvo de relacionamentos transversais.

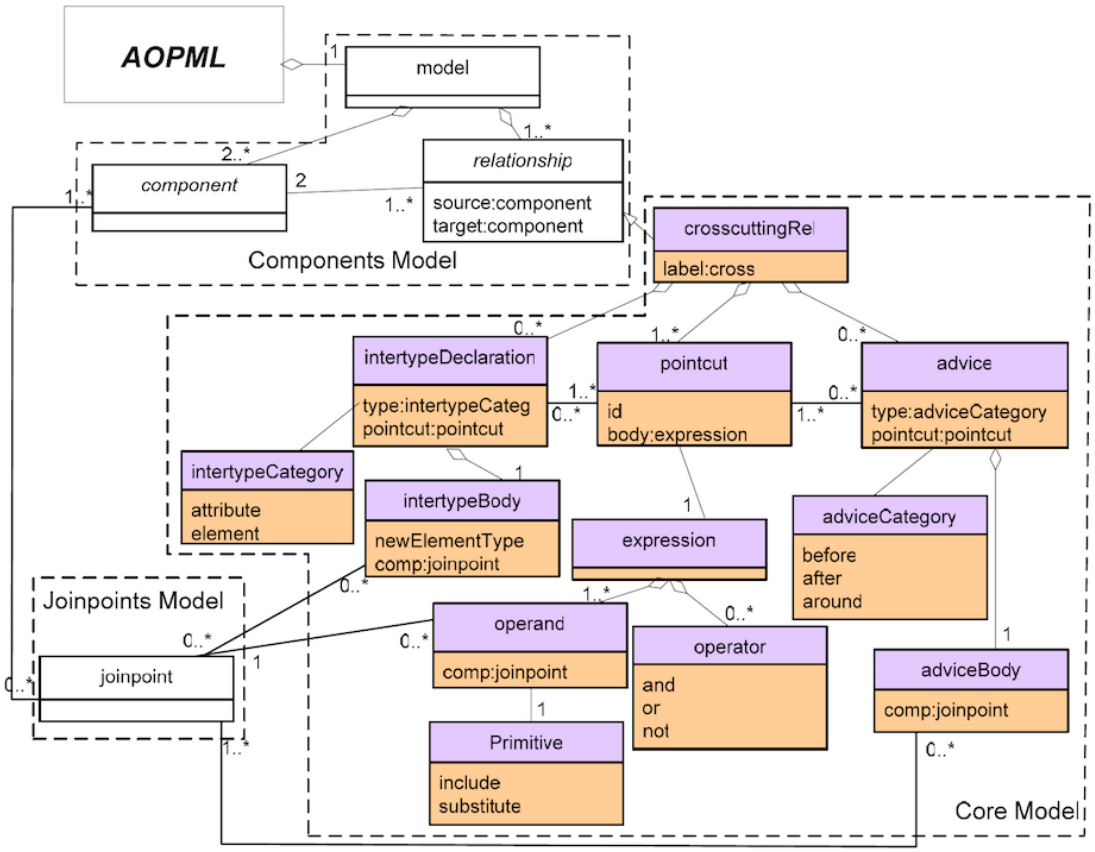


Figura 11 – AOPML – Modelo conceitual (Cappelli et al., 2009)

O AOPML reutiliza o metamodelo proposto por Da Silva (2005), substituindo apenas uma linguagem de requisitos por uma linguagem de processos. O uso da AOPML como metalinguagem permite a adoção de elementos gráficos que representem os conceitos da POA, sem especificar detalhes desses elementos, permitindo assim o uso de notações gráficas de outras linguagens de modelagem para representar interesses transversais.

Em outro trabalho, Cappelli et al. (2010) desenvolveram uma notação de AOPML especificamente para lidar com modelos BPMN. A notação foi denominada *Aspect-Oriented Business Process Modeling Notation* (AO-BPMN), que usa o paradigma da orientação a aspectos para estender o BPMN. Os autores propuseram a introdução de elementos gráficos na notação através de novos papéis para as raias e para as relações. Nesse caso, os interesses transversais são representados separadamente em raias verticais, ortogonais ao processo. Ele usa um novo conector chamado "*relationship cross"*, que representa a composição entre os elementos transversais do interesse e o modelo de processo (Figura 12).

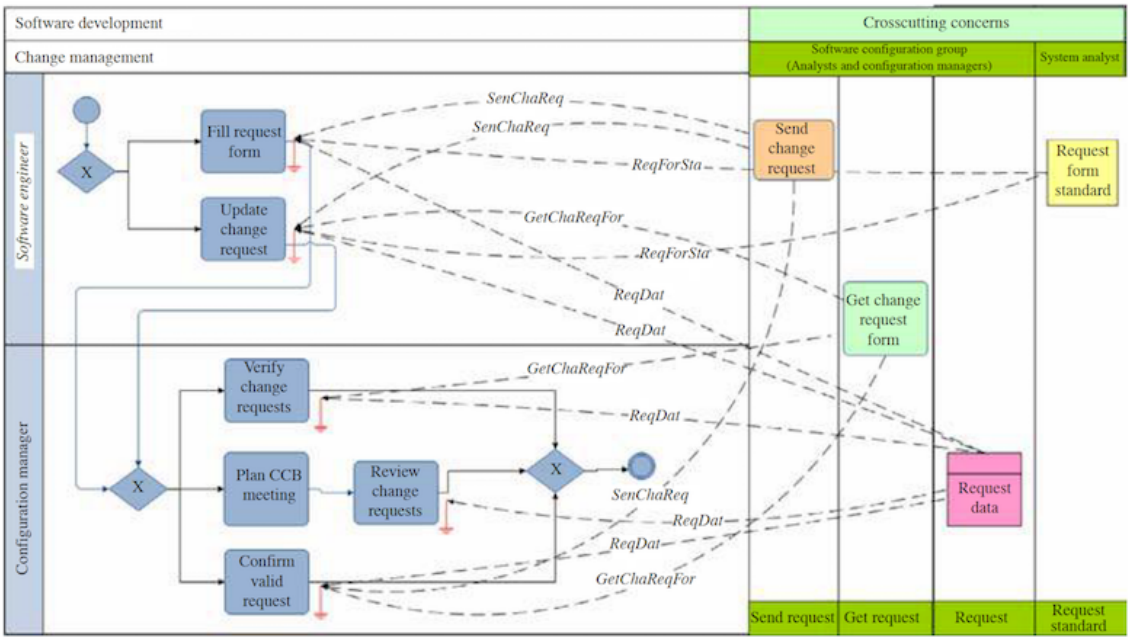


Figura 12 – Exemplo de processo usando AO-BPMN (CAPPELLI et al., 2010)

Da mesma forma, Charfi et al. (2010) sugerem uma extensão ao BPMN com conceitos para lidar com orientação a aspectos, usando a terminologia de POA. Os autores definem *join points* como atividades e eventos. Além disso, eles introduzem três tipos de linguagem de *pointcut*:

- Conectar *pointcuts* com seus *join points* correspondentes durante a fase de modelagem, que requer ter aspectos e o processo principal na mesma visão.
- Usando linguagens de consultas como OCL ou QVT para serem linguagens de *pointcuts*, e assim poder especificar a relação entre cada *pointcut* com o *join point* relevante.
- Definir anotações textuais para cada *join point* para relacioná-los aos *pointcuts* apropriados.

Um *advice* é definido com um sub-processo que contém a funcionalidade do interesse transversal. Um tipo especial de atividade é introduzido chamada *Proceed*. A atividade *Proceed* é proposta para capturar o *join point* que deve ser retomado, ou seja, indicar o ponto onde a atividade do processo principal deve ser executada. Ele também determina o tipo de *advice* (antes, depois ou ao redor) em virtude de sua posição no modelo de processo. Aspectos são definidos como módulos que contém os *pointcuts* e *advices* que incluem a modularidade transversal. Na Figura 13 temos a representação dos aspectos "*Archive*" e "*Security*", contendo seus respectivos *advices*. Os *pointcuts* são indicados através do elemento de anotação do BPMN nas respectivas atividades onde se ocorrerá a junção.

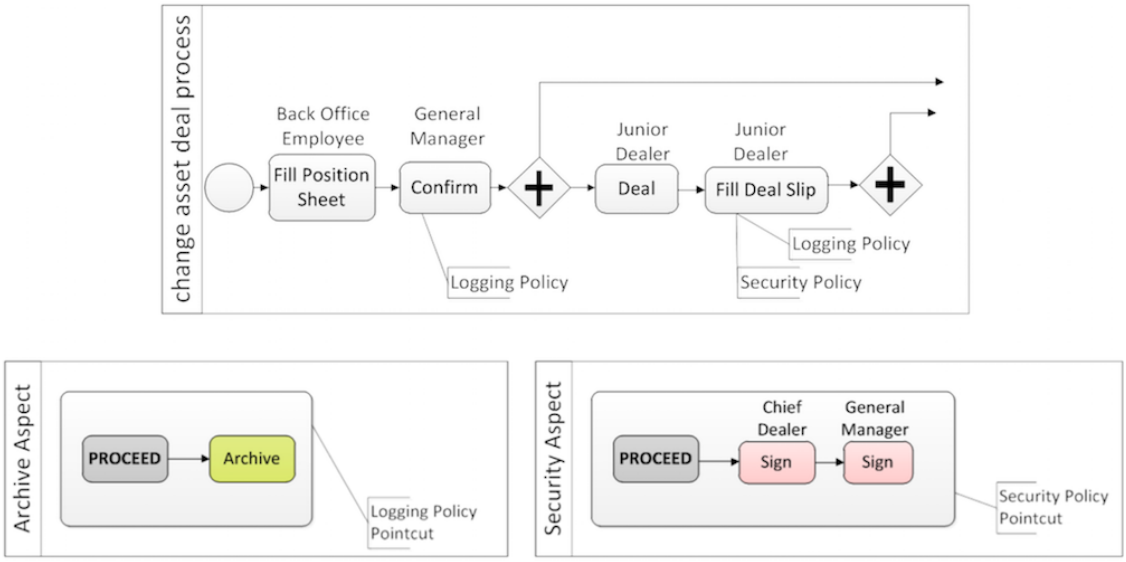


Figura 13 – Exemplo de processo em AO4BPMN (CHARFI; MÜLLER; MEZINI, 2010)

Charfi e Mezini (2007) definiram o AO4BPEL, uma extensão do *Business Process Execution Language* (BPEL) para suportar interesses transversais através da modularização orientada a aspectos. A extensão também fornece um mecanismo para ativar e desativar os aspectos, suportando *weaving* dinâmico. Atividades em BPEL são consideradas como *join points*. *Join points* relevantes são selecionados por uma linguagem de *pointcut* baseada em XPath. Elementos de *pointcut* são definidos usando o elemento "<*pointcut*>" em AO4BPEL. Os autores definem um *advice* como uma atividade BPEL que executa antes, depois, ou no lugar do *join point*. A ordem de execução é determinada usando o tipo "*property"* de cada *advice* com o valor "*before*", "*after*" ou "*around*". O *join point* monitorado pode ser substituído por um *advice* se o tipo de *advice* for "*around*". O *join point* será executado usando o elemento "<*proceed*>", que pode ser definido no *advice*.

Para ilustrar a aplicação do AO4BPEL, Charfi e Mezini (2007) usaram um exemplo de portal de viagens que agrega informações de viagens de companhias aéreas e cadeia de hotéis. O resultado da agregação é um *Web Service* composto de três métodos (Figura 14). Cada uma dessas operações é especificada por um processo BPEL (por exemplo, a operação "*getTravelPackage*" agrega os *Web Services* "*Lufthansa*" e "*hotels.com*"). Como parâmetros de entrada, é preciso informar data e cidade de origem, e data e cidade de destino. O método retorna uma descrição de um pacote de férias. A resposta do processo é mantida na variável "*clientresponse*" (linha 9). Esse processo também declara três endereços de parceiros que, respectivamente, conectam a composição com os *Web Services* cliente, "*Lufthansa*" e o "*hotels.com*" (linhas 3-5). A atividade principal deste processo é uma sequência de chamadas aos três serviços (linhas 15-38).

```
1  <process name="travelPackage" .../>
2   <partnerLinks>
3    <partnerLink name="client" partnerLinkType="clientPLT" .../>
4    <partnerLink name="flight" partnerLinkType="LufthansaPLT" .../>
5    <partnerLink name="hotel" partnerLinkType="hotelPLT" .../>
6   </partnerLinks>
7   <variables>
8    <variable name="clientrequest" messageType="tns:findPackageRequest"/>
9    <variable name="clientresponse" messageType="tns:findPackageResponse"/>
10   <variable name="flightrequest" messageType="fs:findAFlightRequest"/>
11   <variable name="flightresponse" messageType="fs:findAFlightResponse"/>
12   <variable name="hotelrequest" messageType="hs:findARoomRequest"/>
13   <variable name="hotelresponse" messageType="hs:findARoomlResponse"/>
14  </variables>
15  <sequence name="packageSequence">
16   <receive name="receiveClientRequest"
17          partnerLink="client" portType="tns:travelServicePT"
18          operation="getTravelPackage" variable="clientrequest" createInstance="true"/>
19   <assign> ...</assign>
20   <invoke name="invokeFlightService"
21         partnerLink="flight" portType="LufthansaPT" operation="findAFlight"
22         inputVariable="flightrequest" outputVariable="flightresponse"/>
23   <invoke name="invokeHotelService"
24         partnerLink="hotel" portType="HotelPT" operation="findARoom"
25         inputVariable="hotelrequest" outputVariable="hotelresponse"/>
26   <assign >
27    <copy>
28    <from variable="flightresponse" part="findAFlightReturn"></from>
29     <to variable="clientresponse" part="flightInfo"></to>
30    </copy>
31    <copy>
32    <from variable="hotelresponse" part="findARoomReturn"></from>
33     <to variable="clientresponse" part="hotelInfo"></to>
34    </copy>
35   </assign>
36   <reply name="replyToClient" partnerLink="client" portType="tns:travelServicePT"
37        operation="getTravelPackage" variable="clientresponse" />
38  </sequence>
39 </process>
```

Figura 14 - O processo de pacote de viagem (CHARFI; MEZINI, 2007)

A Figura 15 ilustra um exemplo de aspecto usando AO4BPEL, onde temos o aspecto "*Counting*" que coleta dados necessários para o faturamento. Ele conta quantas vezes a operação "*findAFlight*" do *Web Service* "*Lufthansa*" é invocado a partir de qualquer processo. Este aspecto declara um *Web Service* auxiliar, "*CounterWS*", que possui a operação "*increaseCounter*" para incrementar um contador. Esta operação tem um parâmetro inteiro como entrada, para ser adicionado ao número total de invocações. O *pointcut* "*Lufthansa Invocations*" (linhas 10-12) captura todas as invocações da operação "*findAFlight*" de qualquer processo implantado no mecanismo de orquestração. O *advice* do tipo depois (linhas 13-25) associado a este *pointcut* incrementa o total de invocações em uma unidade.

```
<aspect name="Counting">
 <partnerLinks>
  <partnerLink name="CounterWS" partnerLinkType="CounterPLT"
                  myRole="caller" partnerRole="counter"/>
 </partnerLinks>
 <variables>
  <variable name="increaseRequest" messageType="increaseCounterInput"/>
 </variables>
 <pointcutandadvice>
  <pointcut name="Lufthansa Invocations" contextCollection="true">
    //process//invoke[@portType="LufthansaPT" and @operation="findAFlight"]
  </pointcut>
  <advice type="after">
   <sequence>
    <assign>
     <copy>
      <from expression="1"/>
      <to variable="increaseRequest" part="increaseBy"/>
     </copy>
    </assign>
    <invoke partnerLink="CounterWS" portType="CounterPT"
          operation="increaseCounter"
             inputVariable="increaseRequest" outputVariable="increaseResponse"/>
   </sequence>
  </advice>
 </pointcutandadvice>
</aspect>
```

Figura 15 – Exemplo de aspecto em AO4BPEL (CHARFI; MEZINI, 2007)

2.4 Web Service Modeling Ontology (WSMO)

Uma maneira promissora de realizar interesses transversais, como controle de persistência, segurança, *logging* e auditoria em processos de negócios em tempo de execução é através da invocação de *Web Services* (ALONSO et al., 2004). Contudo, o crescente número de *Web Services* disponíveis dentro de uma organização faz emergir um desafio: localizar *Web Services* apropriados que atendam às expectativas do usuário (EL BOUHISSI; MALKI; CHERIF, 2014). Assim, existe a necessidade de usar uma abordagem que permita realizar a descoberta do *Web Service* desejado pelo usuário requisitante, e também que explore as marcações semânticas para automatizar tarefas de descoberta, composição e invocação. Tal abordagem permite uma interoperabilidade transparente entre as tarefas e mantém a intervenção do usuário no mínimo (FENSEL et al., 2005). Uma abordagem adequada ao trabalho proposto nesta dissertação, e que está alinhada ao conceito de objetivo apresentado na ontologia proposta por Bastos et al. (2014) é a *Web Service Modeling Ontology* (WSMO).

Fensel et al. (2005) propuseram a WSMO, que descreve todos os aspectos relevantes relacionados a serviços gerais que são acessíveis através de uma interface de *Web Service*, permitindo a automação total ou parcial das tarefas (descoberta, composição, mediação, execução, monitoramento etc.) envolvidas na integração intra e interempresarial de *Web Services*. O WSMO tem sua base conceitual na *Web Service Modeling Framework* (WSMF) (FENSEL; BUSSLER, 2002), refinando e estendendo este *framework* e desenvolvendo uma ontologia formal e um conjunto de linguagens.

Seguindo os principais aspectos identificados na WSMF, o WSMO identifica quatro

elementos bases como conceitos principais, abaixo listados (Figura 16):

- Ontologias fornecem a terminologia usada por outros elementos do WSMO para descrever os aspectos relevantes dos domínios, fornecendo definições formais que podem ser processadas por máquina e assim permitir que outros componentes e aplicações levem em consideração o significado dos termos no mundo rela, e não apenas seu significa sintático.

- *Web Services* representam entidades computacionais capazes de fornecer acesso a serviços que fornecem algum valor em um domínio. As descrições dos *Web Services* compreendem as capacidades, interfaces e o funcionamento interno do serviço.

- Objetivos (*goals*) descrevem aspectos relacionados aos desejos do usuário em relação à funcionalidade solicitada. Usando ontologias para definir a terminologia do domínio usado, os *goals* modelam características e funcionalidades desejadas pelo usuário ao solicitar o uso do *Web Service*.

- Mediadores descrevem elementos que tratam de problemas de interoperabilidade entre diferentes elementos do WSMO. Eles resolvem incompatibilidades no nível de dados, processos e protocolos, de forma a resolverem diferenças entre terminologias usadas (nível de dados), como se comunicar entre *Web Services* (nível de protocolo) e nível de combinação de *Web Services* (nível de processo).

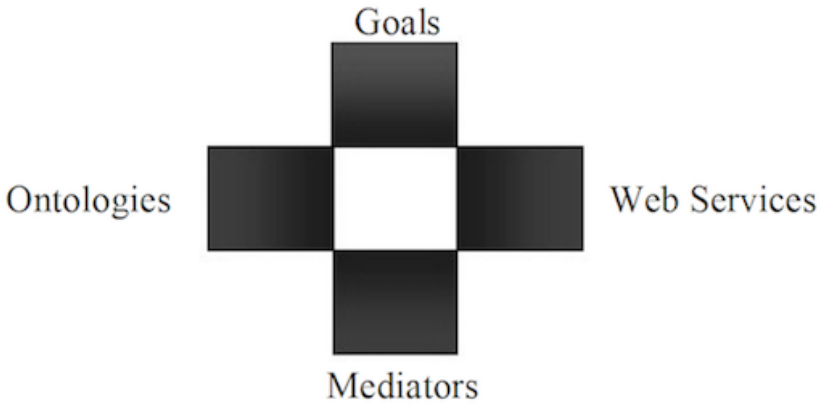


Figura 16 – Conceitos base do WSMO

Fensel et al. (2005) criaram uma linguagem formal chamada *Web Service Modeling Language* (WSML) (DE BRUIJN, 2008) para as descrições WSMO dos *Web Services*, *Goals*, ontologias e para alguns mediadores estendidos. WSML é uma família de linguagens de descrição formais (linguagens de descrição de primeira ordem) que podem ser usadas para a especificação precisa dos elementos únicos no *framework* WSMO. A linguagem provê construtos orientados a objetivos e modelos baseados em estruturas, que

são mais adequados em relação ao XML, por exemplo, para modelagem e representação de conhecimento para pessoas com algum conhecimento em ciência da computação e programação.

Outro aspecto importante no design do WSML é que a linguagem precisava suportar tecnologias e padrões Web existentes, como:

- *Universal Resource Identifier* (URI) – sequência de caracteres com escopo global que identifica um recurso físico ou abstrato (CLARK, 2013);
- *Namespace* – conjunto de símbolos que são usados para organizar objetos de vários tipos, de modo que esses objetos podem ser referenciados pelo nome (BRAY et al., 2009);
- *Simple Object Access Protocol* (SOAP) – protocolo destinado a trocar informações estruturadas em um ambiente descentralizado e distribuído. Usa tecnologias XML para definir uma estrutura de mensagens extensível fornecendo uma construção de mensagem que pode ser trocada por uma variedade de protocolos subjacentes. Foi projetado para ser independente de qualquer modelo de programação particular e de outras semânticas específicas de implementação (CÔTÉ, 2013).

A *Web Ontology Language* (OWL) (MANKOVSKII et al., 2009), que define ontologias para a *Web*, descrevendo um domínio em termo de classes, propriedades e indivíduos e pode incluir descrições ricas das características desses objetos, podendo ainda descrever recursos *Web,* não é usada diretamente pela WSML. A razão para a WSML não usar diretamente a OWL decorre do fato de que a OWL foi projetada como linguagem para *Web* Semântica (BERNERS-LEE; HENDLER; LASSILA, 2001). Ela é adequada para anotação de informações semiestruturadas com a semântica processável por máquina. Contudo, ela não foi projetada para definir semântica de processos que exigem definições ricas de suas funcionalidades (FENSEL et al., 2005).

2.5 Ontologia, WebService, Goal e Mediador do WSMO

O elemento base Ontologia do WSMO define uma terminologia comum acordada fornecendo conceitos e relações entre os conceitos. A Figura 17 mostra os elementos que constituem uma ontologia usando a notação *Meta Object Facility* (MOF) (OBJECT MANAGEMENT GROUP, 2002), onde *nonFunctionalProperty*, *importsOntology*, *usesMediator*, *concept*, *relation*, *function*, *instance* e *axiom* são palavras reservadas na WSML e usadas para descrição de uma ontologia.

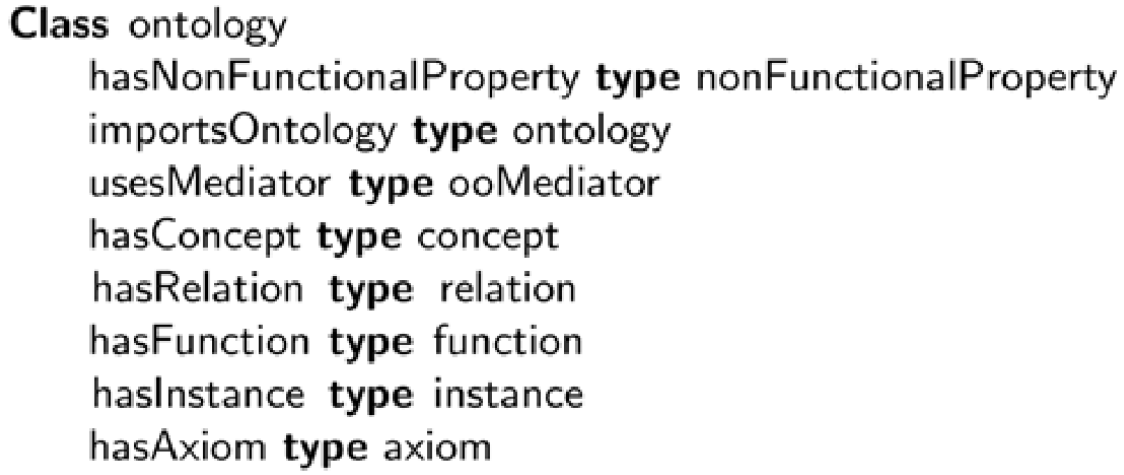


Figura 17 - Ontologia do WSMO e seus elementos

Dentre estes elementos, as propriedades não funcionais, representadas pelo *nonFuncionalProperty*, são permitidas na definição de todos os elementos do WSMO, sendo usadas para descrição de aspectos não funcionais como autor, data de criação, versão etc. A Figura 18 mostra um exemplo de ontologia com suas propriedades não funcionais. Além disso, ilustra o uso de identificadores em WSML, todos em formato de URIs e o uso do mecanismo *QName* (BRAY et al., 2009), escrito no formato `prefix:localPart`, para abreviar a escrita dos identificadores. Estes identificadores são apenas lógicos, ficando a descrição dos seus metadados nas URIs indicadas.

```
namespace <<http://example.org/wsmo#>>
  loc: <<http://wsmo.org/ontologies/location#>>
  xsd: <<http://www.w3.org/2001/XMLSchema#>>
  dc: <<http://purl.org/dc/elements/1.1#>>

ontology <<http://example.org/tripReservationOntology>>
  nonFunctionalProperties
    dc: title hasValue "An ontology describing trips and reservations"
    dc: creator hasValue "DERI Innsbruck"
    dc: publisher hasValue "DERI International"
    dc: contributor hasValues {<<www.deri.org/members/dumitrur>>,
      <<http://homepage.uibk.ac.at/~c703240/foaf.rdf>>}
    dc:date hasValue "2004-12-16"
    dc:format hasValue "text/x-wsml"
  endNonFunctionalProperties}
```

Figura 18 - Exemplo de declaração de propriedades não funcionais

Na Tabela 1 são listados os demais elementos que compõe uma ontologia acompanhada de uma breve descrição para cada um deles.

Tabela 1 - Elementos conceituais do WSMO presentes nas ontologias

Elemento	Descrição

importsOntology	Permite importar ontologias. As ontologias importadas podem ser usadas desde que nenhum conflito precise ser resolvido entre elas. Uma ontologia pode importar outras, assim todas as declarações são virtualmente incluídas.
usesMediator	São usados quando é necessário realizar um alinhamento de ontologia importada.
concept *subconcept*	Constituem os elementos básicos da terminologia acordada para algum domínio do problema dentro de uma ontologia. Eles representam classes de objetos de um mundo real ou abstrato que têm uma propriedade compartilhada específica. Conceitos podem ter subconceitos, criando uma relação é-um com o conceito declarado.
relation	São usadas a fim de modelar interdependências entre vários conceitos, mais precisamente as instâncias desses conceitos.
function	É uma relação especial, com um intervalo unário e domínio n-ário (parâmetros herdados da relação), onde o intervalo especifica o valor de retorno.
instance	Em uma ontologia, um conceito representa um conjunto de objetos em um mundo real ou abstrato com propriedades compartilhadas. Os próprios objetos são chamados instâncias.
axiom	Axioma é considerado como uma expressão lógica, juntamente com suas propriedades não funcionais.

A Figura 19 mostra um exemplo de ontologia descrita em WSML, definindo os conceitos de livro e carrinho de compras.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {
    _"http://example.org/bookOntology#",
    dc _"http://purl.org/dc/elements/1.1/"
}
ontology _"http://example.org/bookOntology"
    nonFunctionalProperties
        dc#title hasValue "Example Book ontology"
        dc#description hasValue "Example ontology about books and shopping carts"
    endNonFunctionalProperties
    concept person
        name ofType _string
    concept book
        title ofType _string
        hasAuthor ofType author
    concept author subConceptOf person
        authorOf inverseOf(hasAuthor) ofType book
    concept cart
        id ofType (1) _string
        items ofType book

    instance crimeAndPunishment memberOf book
        title hasValue "Crime and Punishment"
        hasAuthor hasValue dostoyevsky

    relation authorship(impliesType author, impliesType document)
        nonFunctionalProperties
            dc#relation hasValue authorshipFromAuthor
        endNonFunctionalProperties

    axiom authorshipFromAuthor
        definedBy
            authorship(?x,?y) :- ?x[authorOf hasValue ?y] memberOf author.
```

Figura 19 – Exemplo de ontologia descrito em WSML (FENSEL; KERRIGAN; ZAREMBA, 2008)

O elemento base *Web Services* do WSMO fornece um modelo conceitual para descrever de forma explícita e unificada todos os aspectos de um *Web Service*, incluindo suas propriedades não funcionais, sua funcionalidade e as interfaces para obtê-lo. Além disso, é uma entidade computacional que é capaz, por invocação, de atingir um objetivo de usuário. A Figura 20 mostra os elementos que constituem a definição de um Web Service usando a notação MOF.

Class webService
hasNonFunctionalProperty **type** nonFunctionalProperty
importsOntology **type** ontology
usesMediator **type** {ooMediator, wwMediator}
hasCapability **type** capability **multiplicity** = **single−valued**
hasInterface **type** interface

Figura 20 – *Web Service* do WSMO e seus elementos

Dos elementos mostrados, dois são principais na definição de um *Web Service*:

- Capacidade (*capability*) - a funcionalidade oferecida por um determinado

serviço é descrita por sua capacidade. Ela expressa o estado do mundo antes do serviço ser invocado, e o estado após a invocação bem-sucedida. A capacidade do *Web Service* destina-se principalmente para fins de descoberta e seleção, ou seja, a capacidade é usada pelo solicitante para determinar se o serviço atende às suas necessidades.

- Interface - uma interface descreve como a funcionalidade do *Web Service* pode ser alcançada, ou seja, como a capacidade do serviço pode ser cumprida. A interface do *Web Service* destina-se principalmente a fins de descrição comportamental e é apresentada de uma forma que é adequada para agentes de software determinarem o comportamento do serviço. Também pode ser útil para fins de descoberta, seleção e ligação com especificações existentes, como WSDL. Oferece uma visão dupla da competência operacional de um *Web Service*:
    - Coreografia (*choreography*) - decompõe uma capacidade em termos de interação com outros serviços;
    - Orquestração (*orchestration*) - decompõe uma capacidade em termos de funcionalidade exigida de outros serviços.

Além do *nonFuncionalProperty*, *importsOntology* e *usesMediator*, a descrição da capacidade do *Web Service* possui também as definições listadas na Tabela 2.

Tabela 2–Elementos de definições da capacidade de um serviço WSMO

Elemento	Descrição
sharedVariables	Representam as variáveis que são compartilhadas entre as *preconditions*, *postconditions, assumptions* e *effects*.
precondition	Especificam o cenário inicial válido para as informações requeridas pelo serviço, isto é, restringem as informações em um estado que satisfaça as restrições definidas e são obrigatórias para executar o serviço de maneira correta.
assumption	Descreve o cenário que é assumido antes da execução do serviço. Caso contrário, a execução bem-sucedida do serviço não é garantida. Ao contrário das *preconditions* que verificam o cenário requerido para execução do serviço, as *assumptions* servem de validação adicional do estado atual das informações antes da execução do serviço, não sendo necessariamente verificadas pelo

	Web Service. Os autores fazem esta distinção para permitir uma noção explícita de condições que existem no mundo real, mas que existem fora do espaço da informação.
postcondition	Descreve o cenário das informações garantido após a execução bem-sucedida do *Web Service*. Ela também descreve a relação entre as informações fornecidas ao serviço e seus resultados.
effect	Descreve o cenário que é garantido ser alcançado após a execução bem-sucedida do *Web Service*, ou seja, se as *preconditions* e *assumptions* do serviço foram satisfeitas.

A Figura 21 mostra um exemplo de descrição de um *Web Service* em WSML.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {
    _"http://example.org/bookOntology#",
    dc _"http://purl.org/dc/elements/1.1/"
}

webService _"http://example.org/bookService"
    nonFunctionalProperties
        dc#title hasValue "Example book buying Web service"
        dc#description hasValue "A simple example for adding items to a shopping cart"
    endNonFunctionalProperties

    importsOntology _"http://example.org/bookOntology"
    capability bookServiceCapability
        sharedVariables {?cartId, ?item}
        precondition
            definedBy
                ?cartId memberOf _string and ?item memberOf book.
        postcondition
            definedBy
                forall ?cart (?cart[id hasValue ?cartId] memberOf cart implies
                    ?cart[items hasValue ?item]).
```

Figura 21 - Exemplo de capacidade de *Web Service* descrito em WSML (FENSEL; KERRIGAN; ZAREMBA, 2008)

O elemento base *Goal* do WSMO é usado para descrever os desejos dos usuários. Ele fornece os meios para especificar os objetivos do lado requisitante ao consultar um *Web Service*, descrevendo em alto nível uma tarefa concreta a ser alcançada.

Goals são representações de objetivos para os quais sua realização deve ser alcançada através da execução de um *Web Service*. Eles podem ser descrições de serviços que potencialmente satisfariam os desejos do usuário.

O WSMO desacopla completamente os objetivos que um usuário requisitante tem dos serviços que podem realmente cumpri-los. Os objetivos devem ser atendidos selecionando dentre os *Web Services* disponíveis o que melhor satisfaz aos desejos do usuário (KELLER et al., 2004). Especificamente, a estrutura de um *Goal* possui os

seguintes elementos, conforme mostrado na Figura 22.

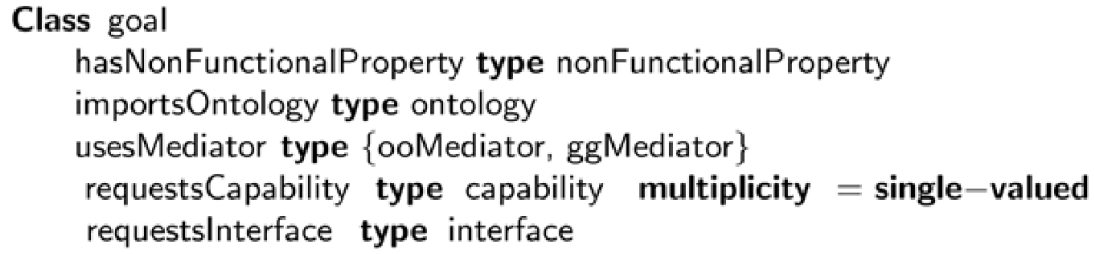


Figura 22 - *Goal* do WSMO e seus elementos

Goals são simétricos aos *Web Services* no sentido de que os objetivos descrevem a funcionalidade desejada e o serviço descreve a funcionalidade oferecida. Portanto, uma descrição de objetivo consiste nos mesmos elementos de modelagem que uma descrição de *Web Service*, ou seja, propriedades não funcionais, uma capacidade e um número de interfaces. Assim, *importsOntology* importa ontologias como uma terminologia para definir os outros elementos que fazem parte do *goal*, *usesMediator* permite fazer o uso de mediadores para tratar conflitos entre terminologias, *requestCapability* descreve a capacidade do serviço que o usuário deseja e *requestInterface* descreve a interface do serviço que o usuário deseja e quer interagir (FENSEL et al., 2007).

Além disso, uma propriedade não funcional extra, "*Type of Match"*, pode ser anexada a um *Goal*, que representa o tipo de correspondência desejada que será feita entre as descrições do *Goal* com as descrições do *Web Service* no processo de descoberta do serviço. Keller et al. (2004) identificaram cinco tipos de noções de correspondência para comparar as descrições semânticas da capacidade entre os *Goals* e *Web Services* (Figura 23). A característica importante dessas noções é que cada uma denota uma relação lógica diferente que tem de ser mantida para considerar um serviço adequado para alcançar um determinado objetivo. Por exemplo, o "*Exact Match"* mantém se e somente se, para cada instância de ontologia possível que satisfaça o *Web Service*, ele também satisfaça o *Goal* e que não exista nenhuma instância de ontologia possível que satisfaça apenas o *Goal* ou o serviço. Em contraste, a "*Intersection Match"* é válida se existir uma possível instância que possa satisfazer tanto o *Goal* quanto o *Web Service*.

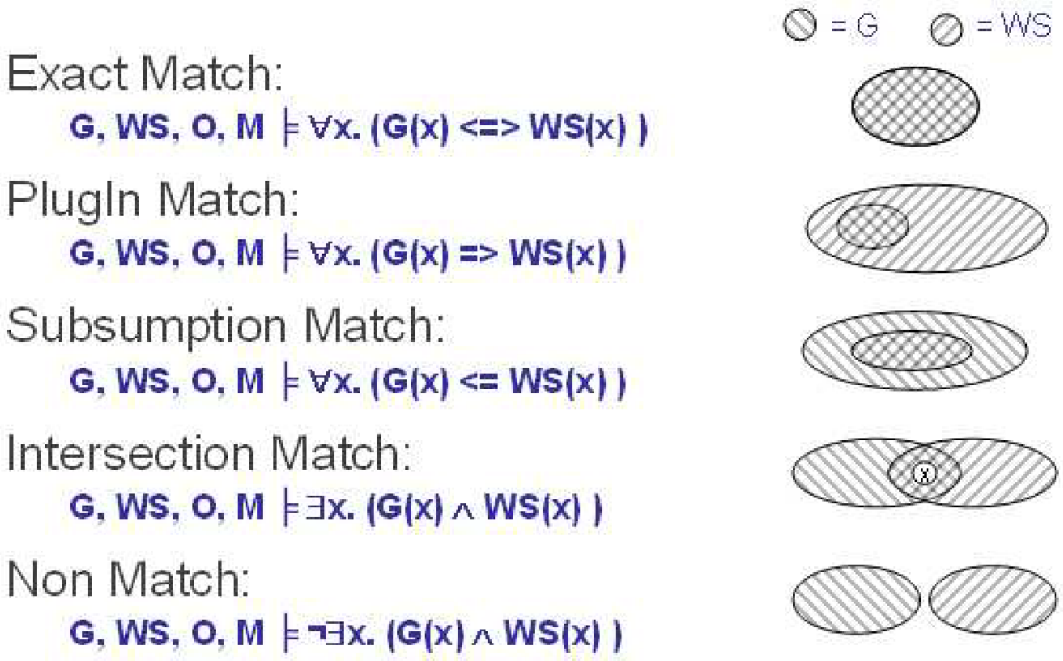


Figura 23 - Tipos de noções de correspondências usados no processo de descoberta (KELLER et al., 2004).

A Figura 24 mostra um exemplo de descrição de *Goal* em WSML. Na descrição do *goal*, definimos a capacidade desejada para o serviço a ser descoberto. Esta descrição é composta de alguns blocos, isto é, *sharedVariables*, *precondition*, *postcondition* e *effect*. O bloco *sharedVariables* é usado para indicar as variáveis que são compartilhadas por todos os demais blocos. No bloco *precondition,* são especificados os dados de entrada, onde "*person"* e "*caller"* possuem valores de entrada definidos em seus atributos. No bloco *postcondition,* são especificados os dados de saída solicitados que devem resultar do processamento do *goal*. No exemplo da Figura 24, solicitamos informações sobre um destinatário, seu "*user Part"* e um domínio para o qual uma invocação deve ser redirecionada. Similarmente, o bloco *effect* define os dados de saída requisitados correspondentes ao estado do mundo esperado após a execução do serviço, em nosso caso a ligação autorizada entre o atendente (*caller*) e o destinatário (*callee*).

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace {
    voip _"http://www.example.org/DERI/VoIPont",
    xsd _"http://www.w3.org/2001/XMLSchema#",
    dc _"http://purl.org/dc/elements/1.1#"
}

goal _"http://www.example.org/STI/SIP2SWSgoal"
    nonFunctionalProperties
        dc#title hasValue "Authorize Call"
        dc#description hasValue "Authorize a call between a caller and a person from a company"
        voip#preference hasValue "price"
    endNonFunctionalProperties

    capability
        sharedVariables {?caller, ?callee}
        precondition
            definedBy
                exists {?person, ?caller} (
                    ?person[
                        name hasValue "Tomas",
                        company hasValue"STI"
                    ] memberOf voip#Person and
                    ?caller[
                        userPart hasValue "Jana",
                        domain hasValue "voip-hub.ie"
                    ] memberOf voip#CallParticipant
                ).
        postcondition
            definedBy
                exists {?callee, ?calleeUserPart, ?calleeDomain} (
                    ?callee[
                        userPart hasValue ?calleeUserPart,
                        domain hasValue ?calleeDomain
                    ] memberOf voip#CallParticipant
                ).
        effect
            definedBy
                exists {?caller, ?callee } (
                    CallAuthorized(?caller, ?callee)
                ).
```

Figura 24 - Exemplo de *Goal* descrito em WSML

O elemento base Mediador do WSMO fornece elementos processuais adicionais para especificar mapeamentos adicionais que não podem ser capturados diretamente pelo uso de ontologias. O conceito de mediação no WSMO aborda o tratamento de heterogeneidades que ocorrem entre elementos que devem interoperar, resolvendo diferenças entre terminologias usadas (nível de dados), a comunicação entre serviços (nível de protocolo) e a combinação de serviços (nível de processo). Um mediador conecta os elementos do WSMO de uma forma totalmente desacoplada e fornece facilidades de mediação para resolver os desajustes que podem surgir no processo de conexão de diferentes elementos. Mais especificamente, o WSMO define quatro tipos de mediadores: *OOMediators* conectam e mediam ontologias heterogêneas; *GGMediators* conectam *Goals*; *WGMediators* ligam *Web Services* a *Goals*; e *WWMediators* conectam serviços interoperacionais resolvendo os conflitos entre eles.

Não abordamos o uso de mediadores nesta dissertação em virtude de um estudo mais aprofundado dos quatro tipos de mediadores definidos pelo WSMO e seu impacto no processo de descoberta, seleção e invocação de serviços. Partimos do princípio que tanto os *Web Services* criados quanto os *Goals* compartilham as definições constantes em uma

única ontologia. Esta ontologia descreve todas as características pertinentes do domínio de negócio que se está trabalhando, e assim dispensa a necessidade de resolver possíveis conflitos entre os conceitos.

2.6 Web Service Execution Environment

Os *Web Services* semânticos requerem ambientes especiais de execução. Além de executar *Web Services* - o que significa controle e monitoramento na troca de informações entre os serviços e seus clientes enquanto as implementações internas do serviço são executadas em sua respectiva localização física - o processo de uso do *Web Service* semântico precisa ser gerenciado e controlado por um ambiente de execução semântica (*Semantic Execution Environment* - SEE). O *Web Service Execution Environment* (WSMX) (Haller et al., 2005) desenvolve esse ambiente como uma implementação para o modelo conceitual provido pelo WSMO, tornando uma implementação de referência. Dado um WSMO *Goal* de um solicitante, o WSMX invoca componentes funcionais para descoberta, seleção, mediação, composição e invocação de serviços a fim de resolver o *Goal* automaticamente.

A principal vantagem do ambiente WSMX (Figura 25) é ser uma arquitetura de componentes baseada em eventos, juntamente com a semântica de execução formal, que permite a invocação dinâmica dos componentes do sistema conforme necessário para processar um objetivo específico do cliente.

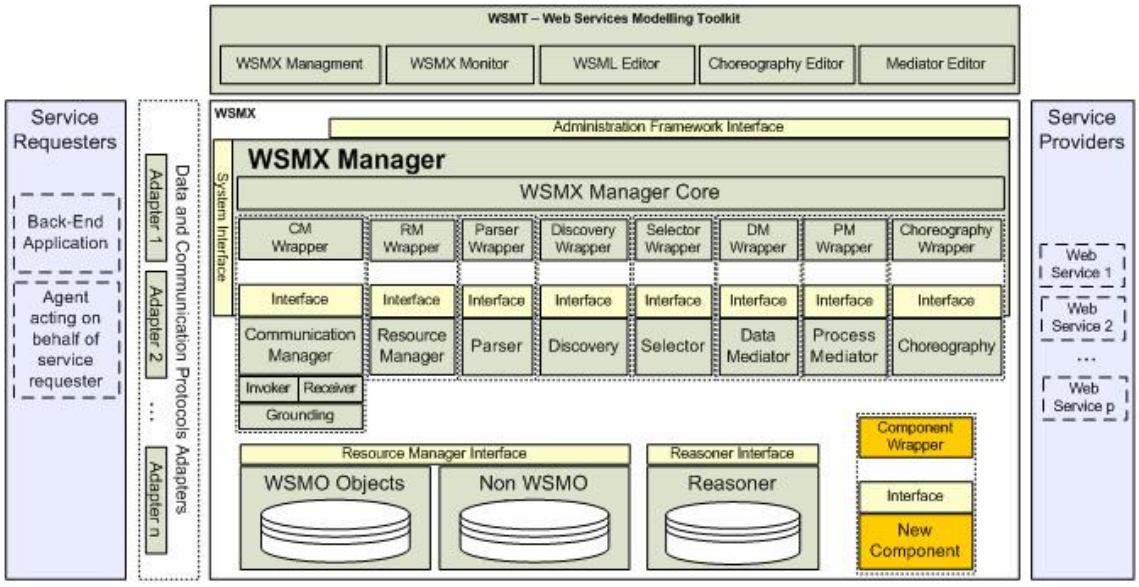


Figura 25 - Arquitetura WSMX (BUSSLER et al., 2005)

As funcionalidades do WSMX podem ser classificadas em duas principais categorias: a primeira é a funcionalidade necessária para suportar as operações (por exemplo, descoberta ou invocação) no *Web Service* semântico; e a segunda, a funcionalidade adicional proveniente dos recursos do framework. No primeiro caso, a funcionalidade

global do WSMX pode ser vista como uma agregação das funcionalidades dos componentes, que fazem parte da arquitetura. No segundo caso, o WSMX oferece recursos como um mecanismo de conexão que permite a integração de vários componentes distribuídos, um mecanismo de fluxo de trabalho interno capaz de executar descrições formais do comportamento dos componentes ou um gerenciador de recursos que permite a persistência de dados WSMO e não-WSMO produzidos durante o tempo de execução.

O comportamento externo do WSMX é descrito em termos dos chamados *entry points*, que representam interfaces padrão que permite a comunicação com entidades externas. Existem quatro pontos de entrada obrigatórios que têm de estar disponíveis em cada instância de trabalho do sistema. Cada um desses pontos de entrada aciona uma semântica de execução particular, que, por sua vez, seleciona o conjunto de componentes a ser usado para esse cenário específico:

- Execução unidirecional de *Goals* - Este *entry point* permite a realização de um objetivo sem qualquer interação de ida e volta. Neste cenário simplista, o solicitante deve fornecer uma descrição formal de seu objetivo em WSML e os dados necessário para invocação, e o sistema selecionará e executará o serviço em nome do solicitante. O solicitante pode receber uma confirmação final, mas este passo é opcional.

- Descoberta de *Web Service* - Um cenário mais complexo (e realista) é consultar apenas o WSMX sobre o conjunto de *Web Services* que satisfazem um determinado objetivo (a seleção pode ocorrer mais tarde, por exemplo, no lado do solicitante). Este *entry point* implica uma invocação síncrona, o solicitante fornece um objetivo e o WSMX retorna um conjunto de serviços correspondentes.

- Envio de mensagem - Após a decisão sobre o serviço a ser usado, uma conversação ida e volta envolvendo mensagens entre o solicitante e o WSMX pode ser iniciada. O parâmetro de entrada é uma mensagem WSML que contém um conjunto de instâncias de ontologia e referências ao *Web Service* a ser invocado e a coreografia de destino (se disponível).

- Armazenar entidades no registro - Esse *entry point* fornece uma interface para armazenar entidades WSMO (descritas em WSML) no repositório do WSMX.

É importante observar que todas as mensagens recebidas e enviadas são representadas em WSML, e são fragmentos de ontologias WSMO ou entidades WSMO (*Web Service, Goals*, mediadores ou ontologias). Ou seja, apenas o WSML é usado como representação

de dados internos do WSMX e todas as operações de adaptação necessárias de e para outros formatos de representação são tratadas por um *framework* de adaptadores.

A arquitetura WSMX segue os princípios fundamentais do *Service-Oriented Architecture* (SOA) (JOSUTTIS, 2007), que consistem na integração de componentes heterogêneos por funcionalidades desacopladas providas por componentes de software distribuídos, que juntos conseguem executar uma tarefa. Mesmo que o WSMX ofereça implementações padrão para todos os componentes da arquitetura, componentes autônomos com funcionalidades bem definidas podem ser facilmente conectados e desconectados a qualquer momento.

2.7 Trabalhos relacionados

Poucos trabalhos têm sido conduzidos para investigar esta lacuna existente nas representações de aspectos dentro do ciclo de vida BPM. Os trabalhos mais recentes são direcionados a identificar novas técnicas e soluções para aplicação de aspectos de BPM, sejam na modelagem de processos, sejam na composição de serviços.

Em Charfi e Zhi (2015), os autores evidenciaram a dificuldade de expressar interesses transversais (ou não funcionais) como segurança e qualidade de serviço nas linguagens de modelagem de processos, como BPMN, e muitos trabalhos na literatura focam a representação dos interesses nas fases de modelagem, não focando sua realização na fase de implementação. O trabalho apresenta uma abordagem genérica para realização de interesses não funcionais, onde os interesses transversais são identificados no modelo usando o elemento anotação do BPMN. Depois cada interesse é mapeado para perfis não funcionais, estendidos de um trabalho anterior do primeiro autor, e depois são convertidos para AO4BPEL. O processo principal é mapeado para WS-BPEL. Além disso, foi desenvolvido uma ferramenta baseada em Eclipse para suportar a abordagem apresentada e também uma *engine* de execução para WS-BPEL e AO4BPEL. Analisando a proposta, ela aproxima os interesses não funcionais de suas respectivas implementações, mas não é baseada em uma das representações de AO-BPM, conforme apresentado na Seção 2.3. A proposta não provê mais detalhes sobre o interesse transversal aspectualizado, seus *advices* e *pointcuts*, se limitando a somente indicar qual atividade fará uso de um interesse transversal. Assim, resolve parcialmente a lacuna entre as fases de modelagem e implementação, pois não realiza os interesses representados graficamente como aspectos.

Shankardass (2009) propôs uma extensão ao BPMN, a fim de encapsular interesses transversais em módulos denominados "*Aspect wrapper*". Para relacionar esses módulos ao modelo de processo principal, propôs um novo elemento denominado "*Aspect dot*", que pode ser colocado antes e/ou após as atividades do processo principal. Esses "*aspect dots"* podem estar relacionados aos "*aspect wrappers"* por meio de novos elementos

chamados "*Aspect flow*". A posição dos "*aspect dots"* determina o tipo e a ordem dos aspectos que estão relacionados ao modelo de processo. Isto significa que se um "*advice dot"* é colocado antes ou depois de uma atividade, o "*aspect wrapper"* relacionado será invocado antes ou depois, respectivamente. No entanto, se um "*aspect wrapper"* estiver relacionado (através de um fluxo chamado "*forked aspect flow*") com "*aspect dots"* que são colocados antes e depois de uma atividade, o "*aspect wrapper"* será considerado do tipo ao redor. Cada atividade pode ter diferentes "*aspect dots"*, de modo que a ordem que aparece no modelo determina sua precedência. No entanto, esta abordagem não propõe nenhuma ferramenta gráfica para apoiar a definição de modelos de processos orientados a aspectos, diferente da nossa abordagem que já faz uso do ambiente YAWL (YAWL FOUNDATION, 2014), que possui um editor gráfico. O autor propõe a nível conceitual ferramenta AMAP, não havendo indicação no trabalho que a ferramenta foi efetivamente implementada. Nosso trabalho além de definir o conceito de objetivo operacional, propõe o esquema conceitual e artefatos, indo do nível conceitual até a implementação e aplicação prática. O trabalho proposto procura também unir modelagem e implementação de aspectos quando propõe uma ferramenta para mapear os novos elementos para linguagem BPEL, além de sugerir uma *engine* onde o processo principal e seus aspectos serão executados. Nossa proposta não estende nenhuma linguagem de modelagem de processos em particular, como o autor faz com a linguagem BPMN, e também une modelagem e implementação através de um objetivo operacional semanticamente descritos, evitando assim o mapeamento de elementos gráficos de uma linguagem de modelagem.

Em Jalali et al. (2015), os autores propõem uma abordagem sistemática para apoiar orientação a aspectos indo da modelagem de processos de negócios para a implementação do processo. Os autores definiram uma sintaxe formal para a modelagem de processos de negócios orientados a aspectos e semântica operacional formalmente especificada usando *Coloured Petri Nets* para a execução desses processos baseados no princípio do *weaving* dinâmico de POA. A abordagem foi implementada em artefatos de software, dois quais um permite a um BPMS suportar a técnica de *weaving* dinâmico, e outro permite a definição de aspectos, seus *advices*, os *pointcuts* e suas condições. A solução proposta foi implementada no ambiente YAWL (YAWL FOUNDATION, 2014) servindo de prova de conceito que oferecem suporte prático ao projeto e implementação de processos de negócio. A abordagem foi demonstrada usando um estudo de caso de processos bancários, onde os processos escolhidos foram modelados usando a notação *Petri-Nets*, e executada no ambiente YAWL onde os artefatos produzidos foram implantados. Além disso, é feito uma análise comparativa dos demais trabalhos na área de AO-BPM com a proposta apresentada usando o framework de avaliação proposto por Jalali (2014), onde os resultados mostraram as diferenças entre as abordagens, seus pontos positivos e

negativos, e apontou lacunas na área que podem ajudar a direcionar as pesquisas futuras. Diferente da proposta apresentada pelos autores, nossa solução foca na definição de um objetivo descrito semanticamente que auxilie na união entre modelagem e implementação, a fim de atender os objetivos dos especialistas de negócio e na descoberta de uma implementação para os aspectos.

Bastos et al. (2014) também tentaram resolver a lacuna existente entre modelagem e implementação. Em seu trabalho, propõem uma ontologia para o domínio de processos de negócio orientado a aspectos (ver Figura 4, Capítulo 1) para possibilitar a documentação e desenvolvimento de aspectos em todas as fases da modelagem, configuração e implementação do gerenciamento de processos de negócio. Para isso, realizou uma prova de conceitos explorando a modelagem e implementação do interesse transversal de *logging*, no processo "*Send articles to reviewers*" (ver Figura 1, Capítulo 1). Uma ontologia para o domínio de *logging* foi estendida da ontologia base apresentada para ajudar na prova de conceitos. A ontologia de *logging* foi implementada usando a WSML como entidades WSMO. Foi usado o WSMX para execução das entidades e serviços criados. Conceitualmente é possível relacionar os *goals* definidos durante a fase de modelagem com a implementação de serviços descritos semanticamente usando WSML. Nosso trabalho é uma extensão da abordagem de Bastos et al. (2014) pois usa o conceito de objetivo operacional, o mesmo usado por Santos et al. (2011), visto como uma abordagem que pode unir modelo e implementação sem necessidade de criação de novas anotações, de acoplamento com linguagens de modelagem existentes, e o uso de técnicas e ferramentas para mapeamento entre os elementos. Além disso, foram usados as definições e artefatos de softwares produzidos por Jalali et al. (2015) para apoiar a pesquisa.

Além dos trabalhos citados, outras abordagens foram comparadas e avaliadas em termos de suporte a orientação a aspecto em BPM. O *framework* de avaliação proposto por Jalali (2014) compara as diferentes abordagens em quatro dimensões, abaixo descritas:

- Transversalidade (*Crosscutting*) - investiga se uma abordagem resolve o problema de transversalidade, investigando se a técnica proposta resolve os problemas de espalhamento e entrelaçamento.

- Quantificação (*Quantification*) - investiga se uma abordagem apoia quantificação. Para isso, é necessário a existência de uma ferramenta de suporte. A quantificação pode ser baseada em diferentes perspectivas envolvidas na definição de um modelo de processo de negócios: controle de fluxo, tarefa, dados e recursos. Além disso, a quantificação pode ser baseada

em combinações dessas perspectivas, e não deve impor uma perspectiva dominante.

- Relação com *advices* (*Advice Relations*) - investiga o suporte da abordagem para definição de diferentes relações entre *advices* e o modelo de processo principal. *Advices* podem ser definidos antes, depois ou ao redor de um *advised join point*. Além disso, *advices* podem ser definidos para um *advised join point* em paralelo. É possível que um *advice* esteja relacionado a outro *advice*, ou seja, alinhado. Finalmente, *advices* podem ter diferentes precedências quando definidos para um *advised join point*.

- Suporte a fases (*Phase Support*) - investiga se uma abordagem apoia a separação de interesses transversais em diferentes fases do ciclo de vida BPM, suportando na fase de modelagem com um design que permita projetar modelos de acordo com a abordagem, e se os aspectos são incorporados na própria fase de modelagem (*Weaving* estático) ou na fase de execução (*Weaving* dinâmico). Outra medida chamada promoção, verifica se a abordagem suporta a execução de modelos aspectualizados. Além disso, são investigados três ajustes a serem considerados ao executar um modelo de processo orientado a aspectos: *Backward*, que requer ajustar as instâncias do processo principal e *advices* em execução; *Forward*, que requer ajustar novas instância do processo; e *Backward-Forward*, que requer ajustar novos *advices* para instâncias do processo em execução.

A Figura 26 mostra o resultado desta avaliação, onde as colunas de cada linha representam diferentes funcionalidades de uma abordagem de modelagem de processo orientada a aspecto. Os símbolos "+" e "!" em uma célula indica que aquela funcionalidade é suportada diretamente ou indiretamente pela abordagem.

	Crosscutting		Quantification							Advice Relations						Phases Support						
	Scattering	Tangling	Tool Support	Control-flow	Task	Data	Resource	Combinations	Domination	Before	After	Around	Parallel	Nested	Precendece	Designer	Static Weaving	Dynamic Weaving	Enactment	fwd adj	bckwrd adj	fw-bck
Wang et al.	+									+	+	+	!	!	!	+						
Shankardass (1)	+									+	+	+	!		+							
Charfi et al. (AO4BPMN 1.0)	+	+								+	+	+				+						
Cappelli et al. (AO-BPMN)	+	+	+	+	+	+		+	+	+	+					+						
Jabeen et al.	+	+	+	+	+					+	+	+										
Jalali et al. (AOBPMN)	+	+								+	+	+	+		+							
Patiniotakis et al. (2)	+	+	+	+	+					+	+	+	!		!	+		+	+			
Witteborg et al.	+	+	+	+	+		+	+	+	+	+	+				+	+		+			
Our Approach	+	+	+	+	+	+		+		+	+	+	+			+		+	+	+		+

+ : Directly Supported
! : Indirectly Supported

(1) It is not clear if the tool (called AMAP) is implemented or only proposed.
(2) The details are not given, so it is not possible to judge about different properties.

Figura 26 - Funcionalidades suportadas por diferentes abordagens AO-BPM usando o *framework* de avaliação (JALALI, 2014)

A linha "*Our Approach*" refere-se a abordagem apresentada por Jalali et al. (2015). Como nossa abordagem faz uso das definições e artefatos propostos pelos autores, assim podemos afirmar que contemplamos os mesmos itens marcados com o símbolo "+" ("*Scattering*", "*Tangling*", "*Tool Support*", "*Control-flow*", "*Task*", "*Data*", "*Combinations*", "*Before*", "*After*", "*Around*", "*Parallel*", "*Design*", "*Dynamic Weaving*", "*Enactment*", "*Backward-Forward*" e "*Forward*"). Além dos itens mostrados na Figura 26, o uso de *Web* Semântica abre uma nova frente para orientação a aspectos em BPM, favorecendo novas definições de regras para as perspectivas de controle de fluxo, tarefas, dados e recursos.

3. Integração semântica em AO-BPM

O objetivo deste capítulo é apresentar a proposta de solução para o problema indicado no Capítulo 1, descrevendo a semântica do objetivo operacional para criação da relação entre a modelagem do aspecto e sua respectiva implementação. Também será apresentada a arquitetura conceitual onde os conceitos definidos podem ser aplicados.

3.1 Comportamento do aspecto

De acordo com a ontologia proposta por Bastos et al. (2014) o comportamento de um interesse transversal aspectualizado pode ser representado por um dos seguintes elementos:

- Processo - uma descrição baseada no modelo de processo;
- Código fonte - implementação do comportamento em uma linguagem de programação;
- Objetivo - definição abstrata de um objetivo a ser atingido.

Na Figura 27, é apresentado o exemplo conceitual de um interesse transversal de segurança aspectualizado baseado nas definições propostas na ontologia. Foi usado a AO4BPMN (CHARFI; MÜLLER; MEZINI, 2010) para representar o aspecto de segurança com seu *advice* contendo a tarefa especial `PROCEED` e o *pointcut* representado pelo elemento anotação do BPMN. O elemento "*Behavior*" foi adicionado para indicar qual comportamento foi incluído ao aspecto, no caso um objetivo (*goal*).

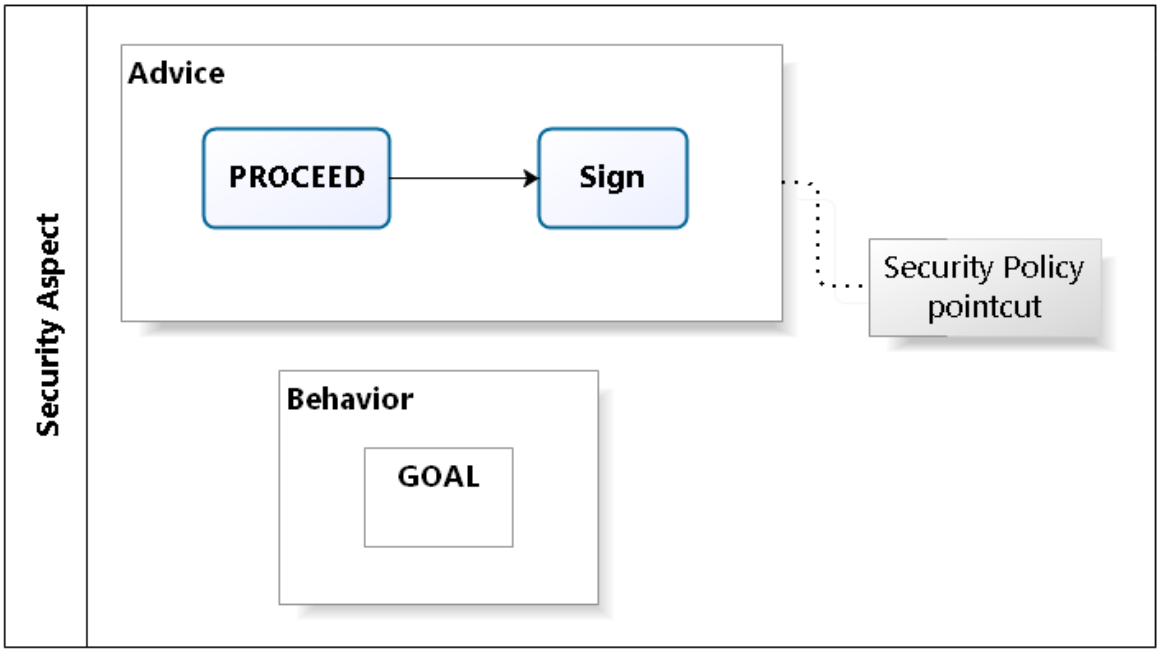


Figura 27 - Representação de um interesse transversal

Ao analisar os conceitos propostos na ontologia de Bastos et al. (2014), obtemos um *template* para documentar as características pertinentes ao interesse transversal que se encontra aspectualizado, definindo assim qual modelo de processo, *Web Service* ou objetivo abstrato que representa o comportamento desejado para o interesse, dependendo da fase dentro do ciclo de vida BPM que se está trabalhando. Por exemplo, na fase de modelagem, definimos qual modelo de processo ou objetivo abstrato representa o comportamento do aspecto, e na fase de implementação temos a definição da rotina, componente ou serviço que representa o comportamento do aspecto naquela fase.

Para se obter um maior alinhamento entre as fases de modelagem e implementação, as três representações definidas por Bastos et al. (2014) devem compor a definição do comportamento do aspecto, de modo a permitir a definição da representação gráfica do aspecto na fase de modelagem, estabelecer qual seu objetivo operacional a ser atingido, e, a partir da descrição semântica do objetivo operacional do aspecto, fazer a realização do interesse transversal ao descobrir uma implementação que atenda ao objetivo definido.

Tal entendimento é necessário em virtude das características dos interesses transversais que representam requisitos não funcionais. No modelo de processos, esses interesses geralmente são indicados contendo poucas atividades descrevendo suas tarefas, pois os detalhes destas etapas ficam encapsuladas em rotinas, módulos ou serviços, sejam eles fornecidos pelo próprio BPMS ou disponibilizados via serviços e componentes remotos. Desta maneira, o comportamento do aspecto em si fica atrelado a própria implementação que é atribuída ao aspecto na fase de implementação.

Assim, o aspecto (*Aspect*) possui um comportamento (*Behavior*), e este comportamento

é composto por processo (*Process*), código base (*Base Code*) e objetivo (*Goal*), conforme mostrado na Figura 28. Definimos os elementos do comportamento do aspecto da seguinte forma:

- Processo - uma notação gráfica de modelagem de processos (por exemplo, BPMN, Petri-Nets, entre outras);
- Código base - funcionalidade (componente ou serviço remoto) a ser executada na fase de execução do processo, escrita em uma linguagem de programação (por exemplo, Java, C#, entre outras);
- Objetivo - definição abstrata de um objetivo expressando o comportamento desejado do aspecto dentro do processo de negócio, escrita usando uma linguagem semântica.

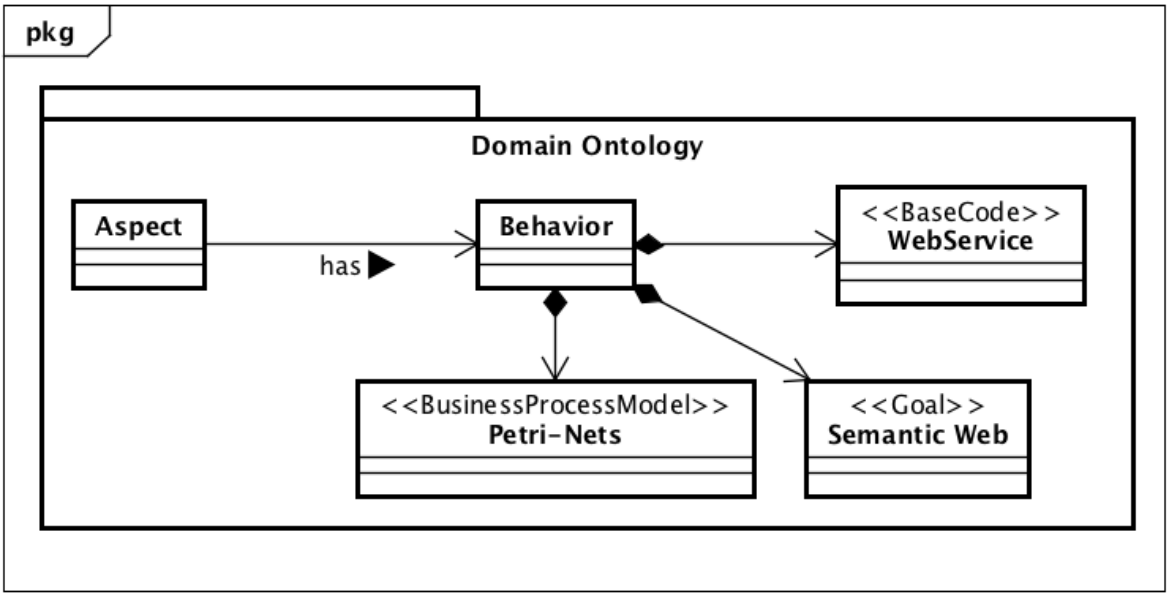


Figura 28 - Representação do comportamento do aspecto

Na Figura 29 temos o aspecto de segurança representando os três elementos. A própria atividade do *advice* faz o papel de representar o elemento "*processo*" do comportamento. Temos ainda o objetivo (*goal*) operacional fazendo o elo entre a atividade e o *Web Service* que será invocado.

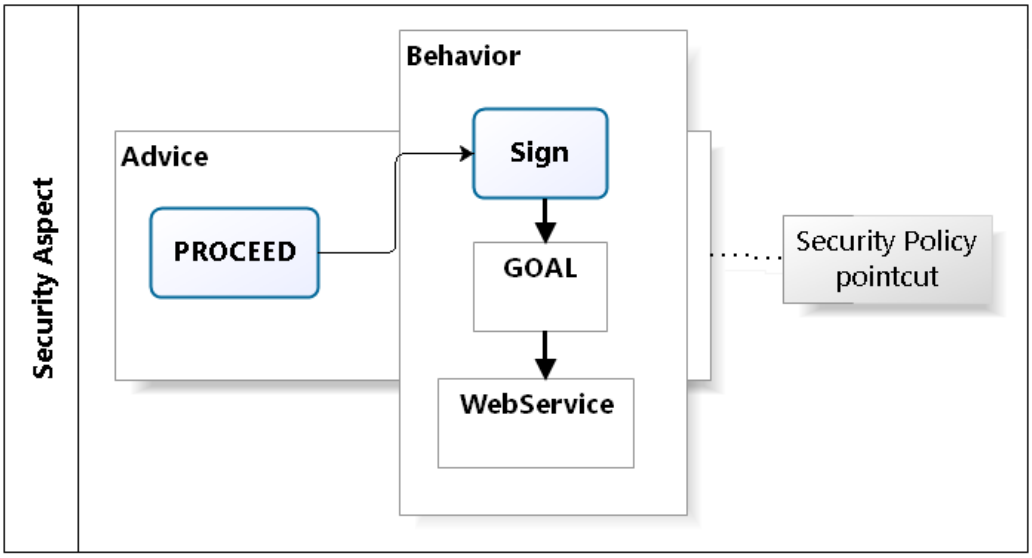


Figura 29 - Representação da definição de comportamento do aspecto

3.2 Objetivo operacional

Bastos et al. (2014) utilizaram o conceito de *Goal* em sua ontologia representando um objetivo operacional que expressa o comportamento desejado para o interesse transversal dentro do processo de negócio. A adoção deste conceito foi inspirada no trabalho de Santos et al. (2011) onde objetivos operacionais foram usados para identificação de aspectos.

A ideia de ter objetivos associados a processos de negócio vem do fato de que os processos existem para satisfazer um objetivo de uma organização. Segundo Soffer e Wand (2004), a associação de objetivos de negócios com modelos de processos de negócios tem recebido pouca atenção. Como consequência, os objetivos não são integrados à modelagem do processo. Alguns esforços na representação de objetivos vêm da engenharia de requisitos, onde os objetivos estão associados com os requisitos (SOFFER; WAND, 2005).

Santos et al. (2011) usam em seu trabalho os conceitos do *framework* GPM (*Generic Process Model*). O *framework* é baseado em conceitos da ontologia de Bunge (BUNGE, 1977, 1979), adaptados para modelagem de sistemas de informação e modelagem de processos de negócio. Além dos conceitos ontológicos básicos, Soffer e Wand (2004) definiram outros para representar interesses de processos em termos ontológicos: domínio, estado, subdomínio, estado estável, estado instável, lei, lei de transição, condição de estabilidade e processo. Também são apresentadas algumas definições para relacionar o conceito de objetivos operacionais com modelos de processo. Tanto Santos et al. (2011) quanto Soffer e Wand (2004) ressaltam que a ideia de "*goal*" está relacionada a um objetivo operacional do processo somente, em oposição aos objetivos de negócio da organização. Em outras palavras, uma meta a ser atingida pelo processo.

Para relacionar o conceito de objetivos a processos, Soffer e Wand (2004) apresentam

as seguintes definições: objetivo (*goal*) é um conjunto de estados estáveis que fazem parte do subconjunto apropriado do conjunto de estados estáveis. O objetivo do processo é quando cada execução do processo permite que a meta seja alcançada, ou seja, alcançar um destes estados estáveis. Para relacionar o objetivo estabelecido com design do processo para que o mesmo seja alcançado, o conceito é operacionalizado através da função de critério, que mapeia os valores das variáveis de estado do domínio, permitindo concluir se o processo atingiu seu objetivo ou não.

Quando temos interesses transversais modularizados em aspectos, temos um processo aspectualizado que terá um objetivo operacional a ser cumprido. Podemos dizer que aquele processo foi concluído ao atingir com sucesso ou não o objetivo atribuído a ele. De acordo com a nossa proposta, atingir o objetivo significa encontrar uma implementação que execute a tarefa que foi designada para o aspecto pelo especialista de negócio através do seu objetivo operacional.

3.3 Formalização do objetivo no AO-BPM

O propósito de definir uma sintaxe formal para aspectos em processos de negócio tem as seguintes vantagens. Primeiramente, fornece uma base sólida para o desenvolvimento de uma sintaxe concreta para a modelagem de aspectos em BPM, que pode ser uma extensão de uma notação de processos existentes como BPMN (OMG, 2011) ou YAWL (VAN DER AALST; TER HOFSTEDE, 2005). Em segundo lugar, serve como referência padrão durante o desenvolvimento de uma semântica operacional para processos de negócios orientados a aspectos. E por fim, é um passo essencial para a definição de uma semântica formal para suportar análise em tempo de design, a fim de verificar sua integridade.

A partir do trabalho de Jalali et al. (2015), estendemos as definições da sintaxe abstrata de AO-BPM para incorporar os conceitos de objetivo operacional aos aspectos, de modo a permitir sua extensão em outras linguagens de modelagem de processos.

Definição 1 (Modelo de processo de negócio) - Um modelo de processo de negócio P pode ser definido como uma tupla (T, C, C_I, C_E, X, F, L, D, R, *Name*, *Dat*, *Act*) onde:

- T é um conjunto de tarefas.
- C é um conjunto de condições.
- $C_I \subset C$ é um conjunto de condições iniciais.
- $C_E \subset C$ é um conjunto de condições finais.
- X é um conjunto de construtos de rotas.

– $F \subseteq (C \setminus C_E(T \cup X)) \cup ((T \cup X) \times C \setminus C_I) \cup ((T \cup X) \times (T \cup X))$ é uma relação de fluxo, tal que cada nó no gráfico $(T \cup C \cup X, F)$ está no caminho direto de uma condição inicial $i \in C_I$ até uma condição final $i \in C_E$.

– L é um conjunto de rótulos de tarefas.

– D é um conjunto de dados de objetos.

– R é um conjunto de regras (recursos humanos).

– $Name : T \rightarrow L$ atribui um rótulo a uma tarefa.

– $Dat : T \nrightarrow D$ associa objetos de dados com uma tarefa.

– $Act: T \nrightarrow 2^R$ designa um ou múltiplas regras para uma tarefa (que requer interação com usuário).

Um *advice* é um processo de negócio especial que contém pelo menos uma tarefa `PROCEED`. Existem três tipos de *advice* que são inseridos: antes (*before*), depois (*after*) e durante (*around*).

Definição 2 (*Advices*) – Um advice P^a pode ser especificado por um modelo de processo $(T, C, C_I, C_E, X, F, L, D, R, Name, Dat, Act)$ em que $\exists_{t \in T} Name(t) =$ 'PROCEED' (onde 'PROCEED' $\in L$ é um rótulo reservado para tarefas `PROCEED`).

Seja P^A um conjunto de *advices* e AN um conjunto de nomes de *advices*, $Advice : P^A \rightarrow AN$ atribui para cada processo um nome de *advice* e $Type : P^A \rightarrow \{$'*before*', '*after*', '*around*'$\}$ especifica quando um *advice* pode ocorrer no *join point* associado.

De acordo com nossa proposta, objetivos operacionais são associados a *advices*, indicando qual o comportamento desejado para aqueles *advices* em tempo de execução.

Definição 3 (*Goal*) - Seja G um conjunto de objetivos operacionais que podem ser atribuídos a um *advice* P^a $(G \rightarrow P^a)$ tal que $\exists_{g \in G} \nrightarrow P^a$.

Uma especificação de aspecto compreende uma série de *advices* pertencentes ao mesmo aspecto, um conjunto de objetivos onde cada objetivo é associado a um *advice*, e um conjunto de regras (*pointcut*) especifica quando estes *advices* são esperados para serem utilizados pelo processo de negócio principal.

Definição 4 (Especificação de aspecto) – Seja P^A um conjunto de processos, Q um conjunto de processos de negócio, e T_P o conjunto de tarefas em cada P $\in$ Q, uma especificação de aspecto pode ser definida como uma tupla (*A*, *Pointcut, G*) onde:

– $A \in 2^G$ representa um ou mais objetivos operacionais.

– $A \in 2^{(P^A)}$ é o número de *advices*.

– Pointcut : $(\bigcup_{P \in Q} T_P) \times BoolExpr \nrightarrow A$ define um conjunto de predicados que se relaciona a uma tarefa específica (que é chamada *advised join point*), sob uma dada condição (de um conjunto de expressões booleanas *BoolExpr*), para o correspondente *advice* em *A*.

Seja $\mathbb{S}$ um conjunto de especificações de aspectos e *SN* um conjunto de nomes de aspectos, $Aspect : \mathbb{S} \rightarrow SN$ atribui para cada especificação de aspecto um nome de aspecto.

Um modelo de processo de negócio orientado a aspecto é composto de um processo principal e um número de aspectos relevantes. O processo principal contém as tarefas com os *advised join points*, que são associados com um *advice* relevante de acordo com as regras específicas do *pointcut* e não possui nenhuma tarefa `PROCEED`.

Definição 5 (Modelo de processo de negócio orientado a aspecto) – Um processo onde:

– *P* é um modelo de processo de negócio que captura o interesse principal, onde dado um conjunto de tarefas *T* do processo *P*, $\neg \exists t \in T$ tal que *Name(t)* = 'PROCEED'.

– $T_A \subseteq T$ é o conjunto de *advised join points*.

– $\mathbb{S}$ é o conjunto de especificações de aspectos.

– $\forall\ t \in T_A, \exists S \in \mathbb{S}$ e $\exists c \in BoolExpr$ tal que $(t,c) \in dom(Pointcut)$.

3.4 Esquema conceitual

Esta seção apresenta em detalhes a arquitetura onde os conceitos apresentados neste capítulo se aplicam. A arquitetura leva em consideração que tanto a linguagem de modelagem de processos (por exemplo, BPMN) quanto o ambiente BPMS, por exemplo YAWL (YAWL FOUNDATION, 2014), já tenham suporte a orientação a aspectos, e que os interesses transversais existentes no processo principal foram identificados pelos especialistas de negócios, ou através de técnicas automatizadas de identificação de aspectos, como apresentado em Brandão et al. (2015). Como a proposta faz uso dos conceitos semânticos, como a WSMO, é importante a definição prévia de ontologias que contenham detalhes do domínio do negócio. A definição destas ontologias servirá de base para os conceitos utilizados na descrição semântica dos *Web Services* e objetivos (*goal*) elaborados, e estarão acessíveis aos especialistas e desenvolvedores através de um repositório de ontologias.

Um objetivo operacional permite indicar qual será o comportamento que um determinado interesse transversal possuirá. O especialista define este objetivo usando uma linguagem de descrição semântica, como a WSML, onde irá indicar qual ontologia será usada, descrever os detalhes da capacidade desejada do *Web Service* a ser invocado (seus recursos, quais dados são válidos na mensagem de envio e retorno, quais condições devem

ser atendidas), e descrever a interface desejada do *Web Service* que executará aquele interesse transversal (item 1, Figura 30). A ontologia definida no objetivo já foi previamente elaborada, ou está disponível em um repositório de ontologias, e que ela representa o domínio do negócio que está sendo modelado.

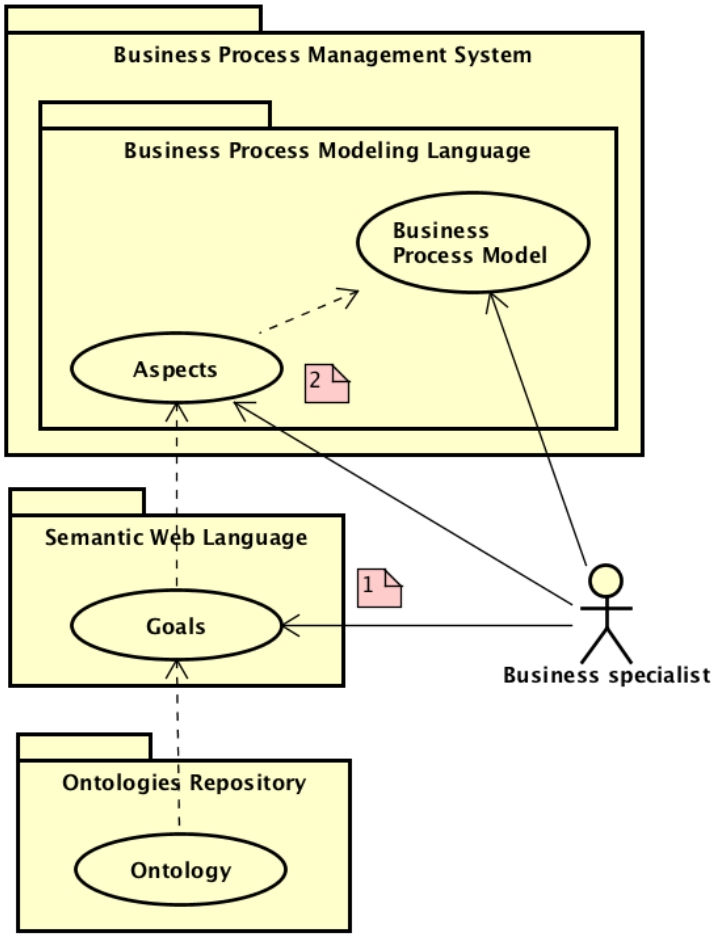


Figura 30 - Definição dos objetivos dos aspectos pelo especialista de negócio

Uma vez definido o objetivo, o especialista terá uma opção para atribuir este objetivo ao aspecto que deseja que sua implementação seja descoberta (item 2, Figura 30). A atribuição do objetivo operacional ao aspecto pode ser feita através de um artefato de software a parte ou ser feito através do próprio BPMS, desde que este ofereça uma funcionalidade para fazer esta atribuição. As associações feitas entre os aspectos e seus objetivos são armazenadas em um repositório de regras acessível ao BPMS, e que é consultado toda vez que um *advice* entra em execução (vide item 6, Figura 32). Os objetivos operacionais descritos são armazenados em um repositório específico, acessível por um ambiente de execução semântica (SEE) (vide item 1, Figura 32).

O desenvolvedor do *Web Service* implementa as funcionalidades providas pelo serviço em uma linguagem de programação, como Java ou C# (item 1, Figura 31). Para cada serviço criado, o desenvolvedor descreve semanticamente a capacidade e interface do serviço, de modo que aquele serviço possa ser descoberto e invocado pelo SEE (item 2, Figura 31). Assim como no objetivo operacional, a descrição do serviço faz uso de

ontologias, então o desenvolvedor faz uso do repositório de ontologias do domínio que o serviço trata (item 3, Figura 31). Uma vez criado, o *Web Service* será publicado em um repositório de serviços onde ele poderá ser acessado por qualquer aplicação, e a descrição semântica do serviço ficará em um repositório específico que será usado pelo SEE.

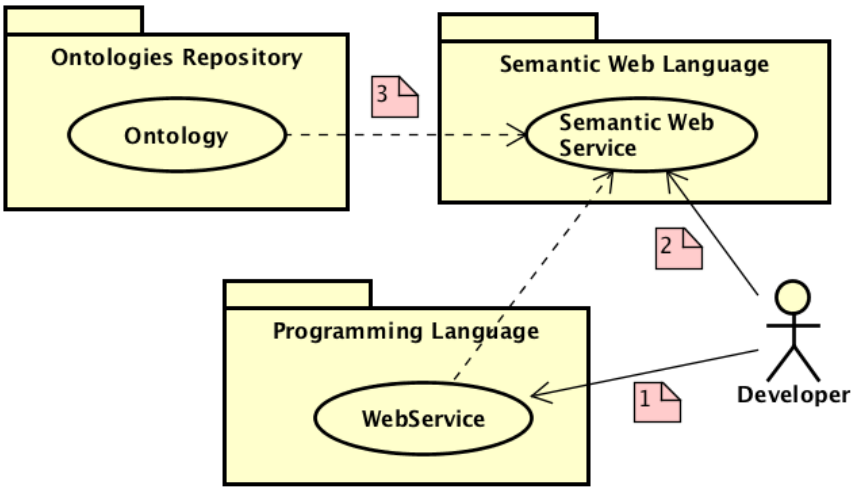


Figura 31 - Definição do *Web Service* e de sua descrição semântica pelo desenvolvedor

O SEE é o responsável por realizar o processo de descoberta e invocação do *Web Service* a partir dos objetivos definidos pelos especialistas e disponibilizados no repositório de objetivos semânticos (item 1, Figura 32). A comunicação com o SEE é feita a partir de duas interfaces: a primeira interface é responsável por interagir com o componente que efetuará o processo de descoberta (item 2, Figura 32); e a segunda interface será responsável por interagir com o componente que efetuará a invocação do serviço escolhido (item 3, Figura 32).

Para viabilizar a comunicação entre o BPMS e o SEE, um componente customizado é incluído no BPMS (item 4, Figura 32). Este componente interage dentro do ciclo de execução dos processos a partir de uma API fornecida pelo próprio ambiente (item 5, Figura 32), de modo a ter acesso aos detalhes da tarefa que está sendo executada no momento. Este nível de acesso é necessário para capturar os dados recebidos pela tarefa no momento da execução, e enviar ao *Web Service* selecionado. O componente customizado tem acesso ao repositório de regras contendo as associações entre os *advices* e seus objetivos (item 6, Figura 32). A partir das associações contidas no repositório de regras, o componente customizado pode invocar a interface de descoberta do SEE passando o identificador do objetivo. O SEE faz o trabalho de descoberta dos serviços, verificando a similaridade das definições (pré e pós condições, parâmetros de interface do serviço) contidas no objetivo com as definições semânticas dos *Web Services* contidas no repositório de serviços. A lista de serviços candidatos é ordenada de acordo com o grau de similaridade das definições, isto é, os *Web Services* que contém o maior número de definições correspondentes aparecem em primeiro lugar. O componente customizado recebe do SEE a lista serviços que atendem aquele objetivo, invocando o primeiro serviço

da lista através da interface de invocação.

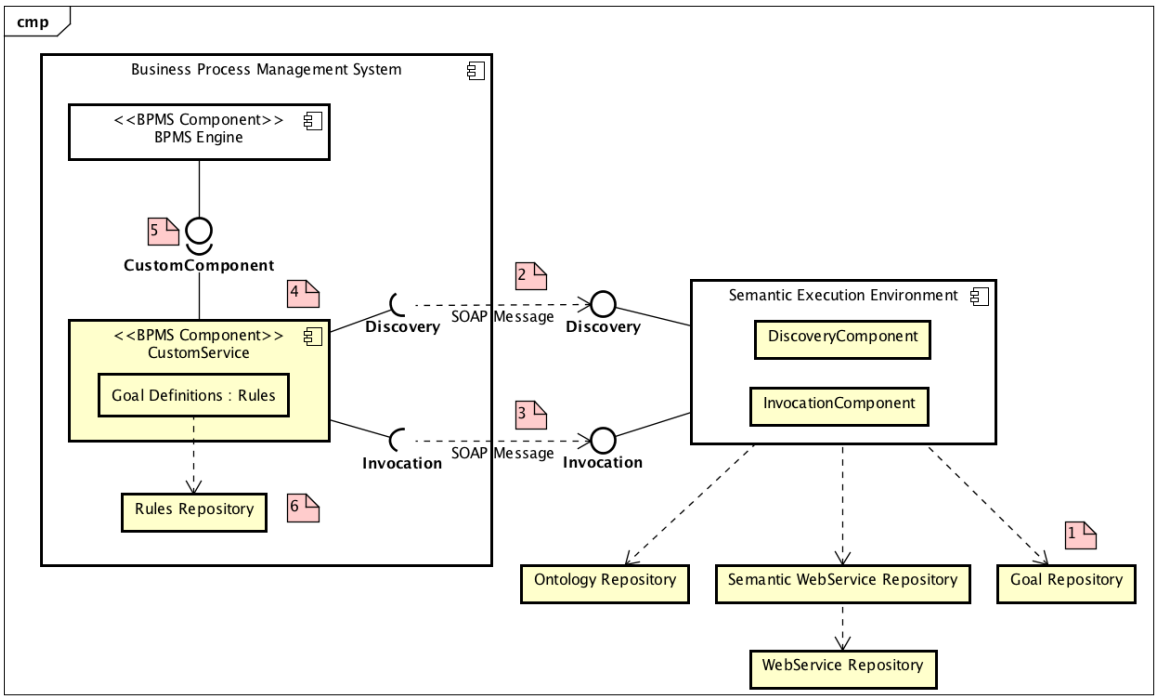


Figura 32 - Esquema de interação entre o BPMS e o SEE através do componente customizado

3.5 Arquitetura e artefatos

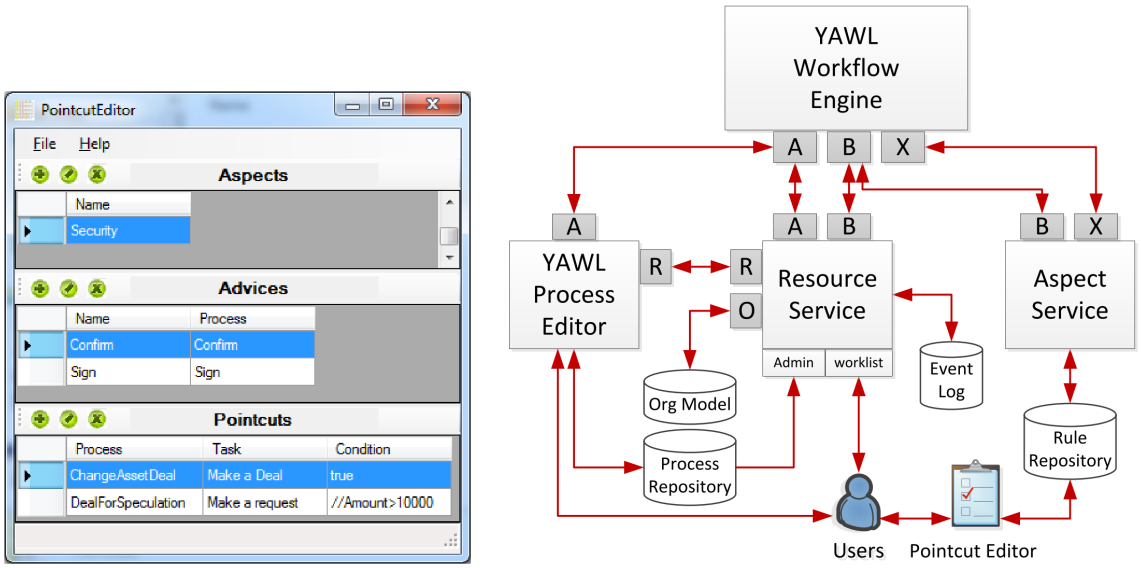
Na seção anterior, foi apresentada a arquitetura a nível conceitual proposta neste trabalho para realizar o elo entre modelagem e implementação de aspectos. Nesta seção serão detalhadas as tecnologias utilizadas e artefatos produzidos a partir da arquitetura apresentada, parte delas sugeridas por Bastos et al. (2014).

A arquitetura leva em consideração que tanto a linguagem de modelagem de processos quanto o ambiente BPMS já tenham suporte a orientação a aspectos. Para atender a este requisito foram usadas as contribuições constantes no trabalho de Jalali et al. (2015), onde os autores definiram a formalização de modelagem de processos orientada a aspectos e a interação entre interesses transversais e o processo principal durante a execução do processo (*weaving* dinâmico), assim habilitando o ambiente YAWL (YAWL FOUNDATION, 2014) a suportar tanto modelagem quanto implementação de aspectos.

O especialista de negócio irá interagir com dois artefatos de software produzidos de acordo com Jalali et al. (2015): o *Pointcut Editor* e o *Aspect Service*. O primeiro artefato (item a, Figura 33) habilita a definição dos aspectos, *advices* e *pointcuts*. Na ferramenta elaborada, cada aspecto pode ter vários *advices*, e cada *advice* pode ter vários *pointcuts*. Estes *pointcuts* consistem no nome do processo e atividade para que o *advice* deve ser inserido. O *pointcut* também inclui uma condição, que é usada para definir restrições de dados, e controlar o ponto exato que o *advice* será invocado. Se a condição for atendida, o

advice é invocado. A condição pode ser escrita usando a linguagem XPath, suportada pelo ambiente YAWL. As definições dos aspectos, *advices* e *pointcuts* ficam registradas em formato XML em um repositório próprio de regras (um diretório no servidor da aplicação acessível pelo *Aspect Service*).

O segundo artefato (item b, Figura 33) mostra o *Aspect Service* e sua relação com o motor de *workflow* do YAWL. O componente customizado está ligado ao motor do YAWL através das interfaces B e X. As interfaces B e X são utilizadas para capturar eventos a nível de instância de processo (*case instance*) e instância de atividade (*workitem instance*), respectivamente. O componente também lê as regras definidas pelo *Pointcut Editor* do repositório de regras. As regras especificam quais eventos devem ser capturados.



(a) Pointcut Editor (b) Aspect Service Architecture

Figura 33 - Artefatos implementados para habilitar *weaving* dinâmico no ambiente YAWL, obtido de Jalali et al. (2013)

Para definir o interesse transversal, as funcionalidades do Editor YAWL são estendidas ao registrar o *Aspect Service* no motor do YAWL, desse modo habilitando o suporte a definição de tarefas `PROCEED` para os *advices*. Essa extensão é possível graças a comunicação entre o editor e o motor de workflow através da interface A (item b, Figura 33).

Os objetivos operacionais, ontologias e *Web Services* são definidos a partir de uma linguagem de descrição semântica. Seguindo a proposta de Bastos et al. (2014), a WSML é usada como linguagem semântica para definir a ontologia dos dados do domínio do processo de negócio modelado, descrever a capacidade e interface dos *Web Services* a serem utilizados, e descrever os objetivos (*Goals*) contendo a capacidade e interface dos serviços desejados pelo usuário. O especialista é responsável por estabelecer o objetivo

com as informações desejadas do serviço a ser descoberto, e o desenvolvedor provê a descrição semântica dos *Web Services* que poderão ser invocados.

Para estabelecer a relação entre os objetivos e seus respectivos *advices*, o artefato *Pointcut Editor* foi estendido para suportar a associação entre os *advices* e os objetivos WSMO. Para isso, uma coluna extra "*Goal*" foi incluída na seção "*Advices*" no *Pointcut Editor* (Figura 34). Esta coluna recebe o nome de identificação do arquivo WSML do objetivo elaborado. A partir desse nome o SEE buscará o objetivo incluído no repositório de objetivos.

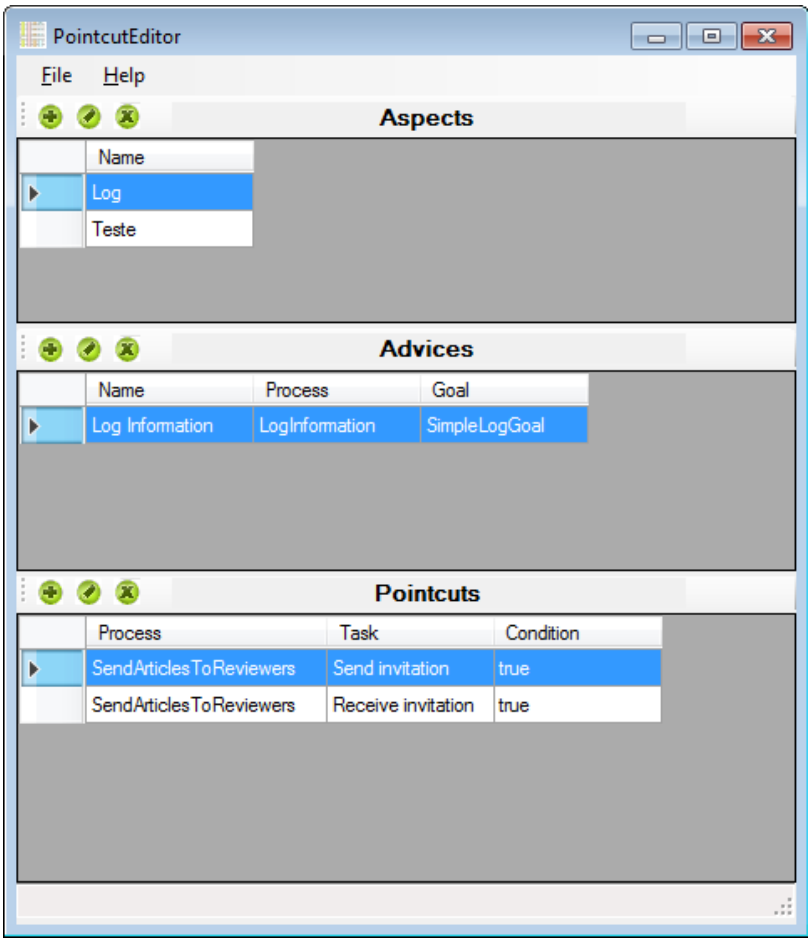


Figura 34 - Extensão do *Pointcut Editor* para suportar objetivos operacionais

Seguindo a proposta de Bastos et al. (2014), o WSMX foi o ambiente de execução semântica escolhido para oferecer os recursos necessários para descoberta e invocação dos serviços. Ontologias, *Web Services* e *Goals* WSMO são armazenados em repositórios específicos acessíveis ao WSMX. Podemos entender estes repositórios como diretórios de arquivos WSML acessíveis ao WSMX.

Para permitir uma interação entre o ambiente YAWL e o WSMX, o artefato *GoalService* foi desenvolvido (Figura 35). Ele é responsável por permitir que o YAWL interaja com o WSMX de modo a invocar o *Web Service* a partir dos objetivos operacionais

estabelecidos durante a execução do aspecto. O *GoalService* foi implementado como um componente personalizado do ambiente YAWL onde a comunicação com o motor de *workflow* é feita via interface B. Já a comunicação com o WSMX é feita através de duas interfaces de serviços: uma para fazer a descoberta dos serviços candidatos para cumprir o objetivo definido no *advice*; a outra interface invoca o serviço escolhido entre os serviços candidatos.

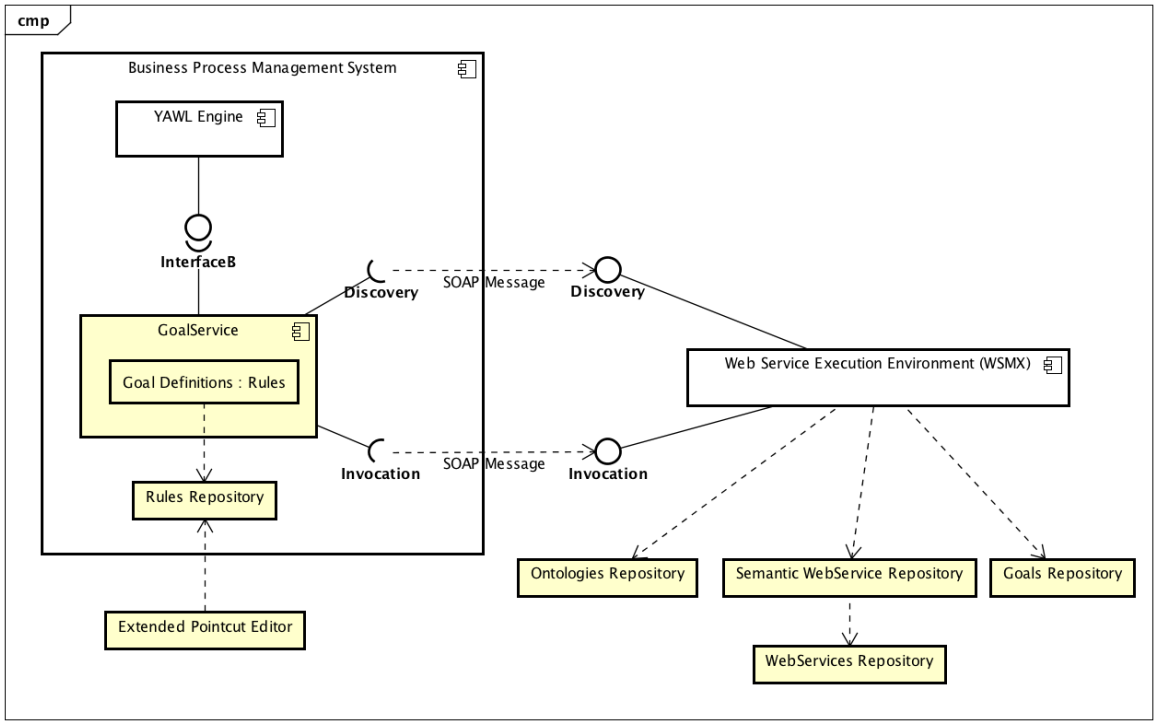


Figura 35 - Esquema de interação entre o YAWL e o WSMX através do *GoalService*

O *GoalService* faz uso do repositório de regras usado pela versão estendida do *Pointcut Editor* para identificar os objetivos atribuídos ao aspecto. O uso de um repositório de regras é útil porque permite que uma possível mudança possa ser atendida em tempo de execução sem causar interrupção do processo, uma vez que a cada execução do *advice* o *GoalService* faz uma nova consulta às regras contidas no repositório. A Tabela 3 ilustra a interação do *GoalService* com o WSMX para descobrir e invocar o *Web Service* baseado no identificador do objetivo armazenado no repositório de regras.

Tabela 3 - Descoberta e invocação do serviço baseado no objetivo operacional

```
1. connect_to(WfMS_Engine);
2. actual_Aspect = WfMS_Engine.get_Actual_Actitivy();
3. WSMX_Connection = WSMX_Server.get_Active_Connection();
4. if (WSMX_Connection not exists) then
5.    WSMX_Server.start_New_Connection();
6.    WSMX_Server.load_Ontologies();
7.    WSMX_Server.load_Semantic_Web_Services();
```

```
8.    WSMX_Server.load_Goals();
9.    WSMX_Connection = WSMX_Server.get_Active_Connection();
10. end if
11. aspect_Goal =
find_operational_goal_from_rule_repository(actual_Aspect);
12. if (aspect_Goal not exists) then
13.   return_Control_to_Engine(WfMS_Engine, actual_Aspect);
14. end if
15. candidate_Web_Service =
WSMX_Connection.run_Discovery_Service(aspect_Goal);
16. if (candidate_Web_Service exists) then
17.   WSMX_Connection.invoke_Web_Service(candidate_Web_Service);
18. else
19.   return_Control_to_Engine(WfMS_Engine, actual_Aspect);
20. end if
21. return_Control_to_Engine(WfMS_Engine, actual_Aspect);
22. end_of_method;
```

Durante a execução do processo principal, o aspecto é incorporado ao processo pela técnica de *weaving*. Quando o aspecto está sendo executado, o *GoalService* intercepta a execução do aspecto e pode ter acesso aos dados recebidos. Inicialmente, é verificado se o WSMX está disponível para interação. Quando o WSMX é iniciado, as ontologias, os *Web Services* semânticos e os objetivos definidos durante a modelagem ficam disponíveis em uma base interna para o procedimento de descoberta e invocação. Uma vez verificado, o artefato acessa o repositório de regras procurando o objetivo definido para o *advice*. O artefato invoca a interface de descoberta do WSMX, enviando o identificador do objetivo e é retornado o *Web Service* semântico disponível que melhor atenda aos critérios definidos no objetivo. Depois disso, o artefato invoca a interface de invocação do WSMX, através da qual o *Web Service* deve ser invocado para executar a tarefa que lhe é cabida. A cada novo processo de *weaving* e execução do *advice*, o *GoalService* pesquisa o objetivo operacional e executa o procedimento de descoberta e invocação do serviço. A Figura 36 ilustra uma visão da execução da arquitetura proposta envolvendo os principais conceitos apresentados.

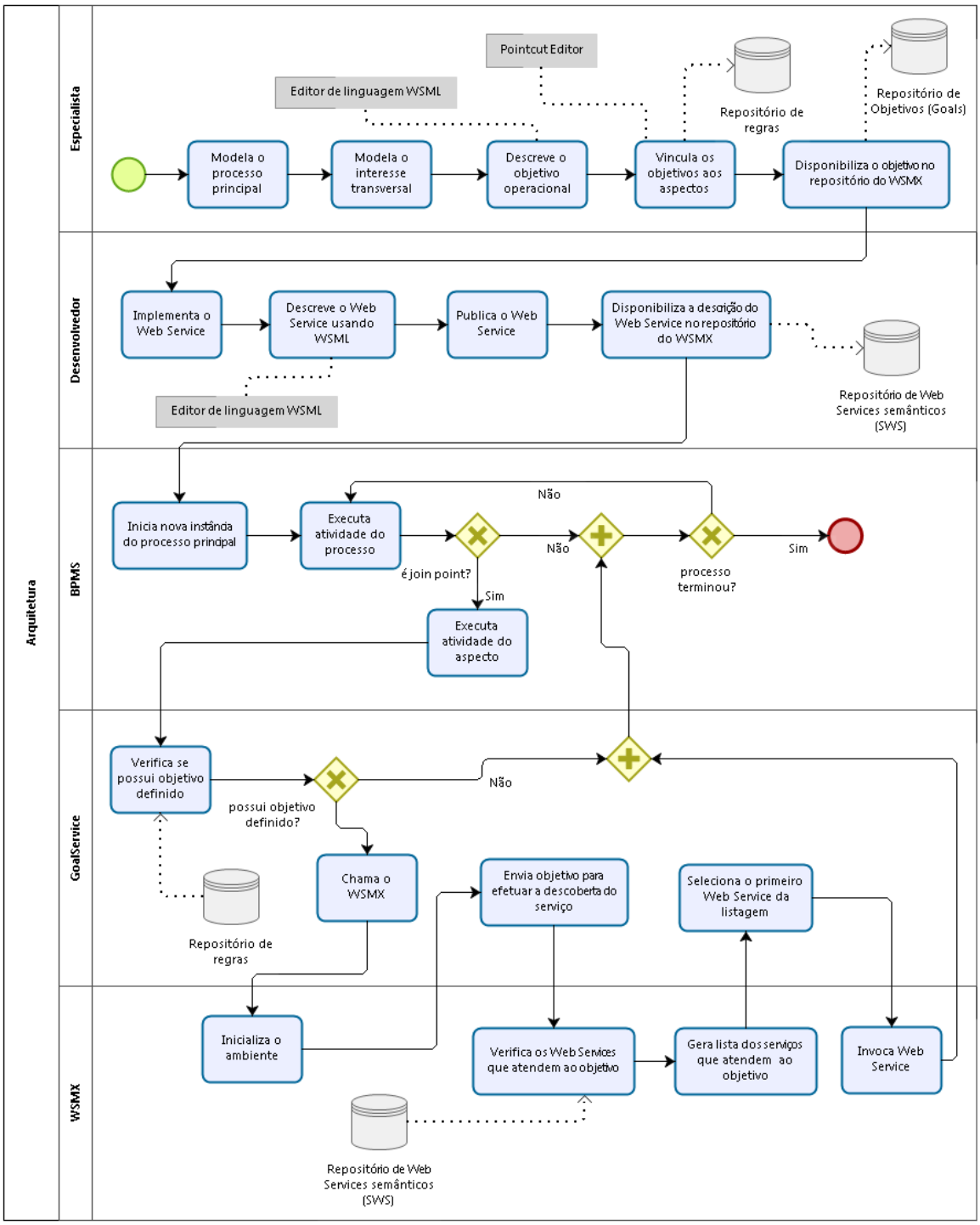


Figura 36 - Visão de execução da arquitetura proposta

4. Avaliação da proposta

Este capítulo tem como objetivo apresentar a avaliação da solução proposta. A partir dos conceitos e arquitetura apresentados no capítulo anterior, os artefatos de software criados foram utilizados para realizar uma prova de conceito. Em seguida foi realizado um experimento como forma de validação da proposta e os resultados foram analisados e comparados com outros trabalhos da literatura.

4.1 Prova de conceito

A prova de conceito é a realização de um certo método ou ideia para demonstrar sua viabilidade, cujo objetivo é verificar se algum conceito ou teoria tem o potencial de ser usado, permitindo demonstrar na prática a metodologia, os conceitos e as tecnologias envolvidas na elaboração do projeto (PINHEIRO, 2010). O termo "*prova de conceito*" foi previamente definido pelo engenheiro eletrônico Bruce Carsten (CARSTEN, 2006) publicado originalmente em sua coluna "*Carsten's Corner*", na revista norte-americana "*Power Conversion and Intelligent Motion*", onde um protótipo foi concebido apenas para verificar as possibilidades do novo produto a fim de comprovar sua utilidade e demonstrar do quanto ele é capaz. A prova de conceito é aplicável ao nosso estudo porque ajuda a validar a viabilidade técnica da arquitetura proposta e dos artefatos produzidos, identificar possíveis obstáculos técnicos e oportunidades ao utilizar o ambiente YAWL em conjunto ao ambiente de execução semântica, e demonstrar de forma prática as evidências apontadas na prova de conceito elaborado por Bastos et al. (2014).

A proposta apresentada no capítulo anterior foi aplicada em um cenário de exemplo com os artefatos produzidos. O cenário usado foi adaptado para o processo "*Send Articles to Review*" descrito por Cappelli et al. (2009) (item a, Figura 37), compreendendo o aspecto "*Log Information"* (item b, Figura 37). O processo escolhido trata da seleção e convite de revisores de artigos pelo responsável de uma conferência. O objetivo desta prova de conceito foi investigar se a abordagem proposta é capaz de selecionar um serviço adequado para implementar o aspecto de acordo com o objetivo definido na fase de modelagem. Além disso, usando o exemplo já fornecido e discutido por Cappelli et al.

(2009), podemos comparar ambas as abordagens e destacar as diferenças e contribuições.

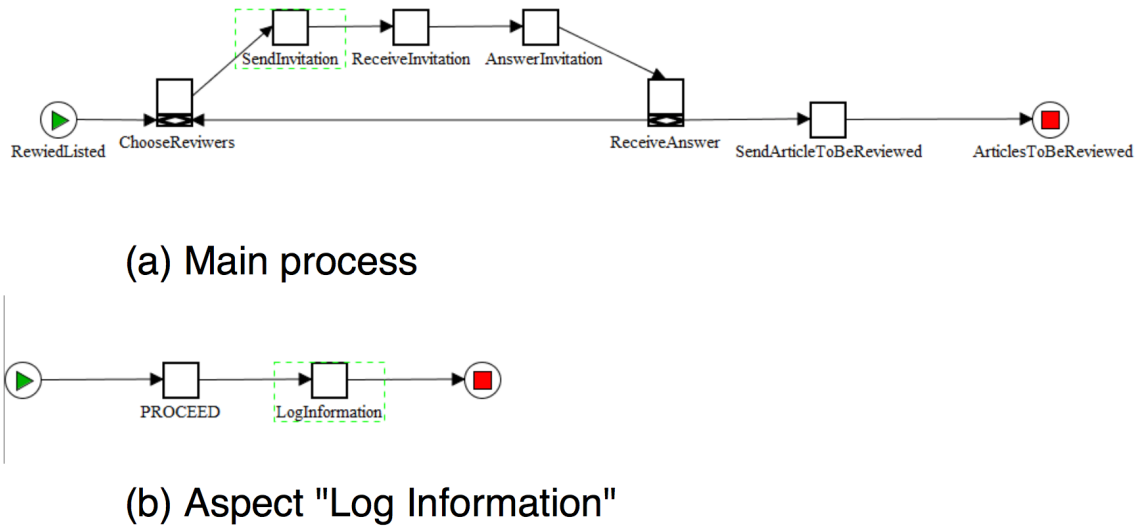


Figura 37 - Processo "*Send Articles to Review*" e o aspecto "*Log Information*", adaptado de Cappelli et al. (2009)

O processo principal e o aspecto foram modelados no editor YAWL. Para permitir que o aspecto fosse incorporado ao processo principal, configuramos o artefato *Aspect Service*[1] (JALALI et al., 2015) no ambiente YAWL, habilitando assim o suporte a *weaving* dinâmico. Usamos os conceitos explicados por Jalali et al. (2015), onde uma tarefa especial chamada `PROCEED` deve ser inserida no *advice*, indicando o *join point* no processo principal. O item A da Figura 38 mostra a configuração da tarefa `PROCEED` para o *advice*. A posição onde a tarefa especial se encontra no modelo indica um *join point* do tipo *after*. Configuramos o item "*Custom Service*" na janela lateral de propriedades do Editor YAWL, indicando assim que aquela atividade deve ser executada pelo *Aspect Service*. Além disso, configuramos na atividade "*Log Information*" o componente *GoalService*[2] (item B, Figura 38), previamente configurado no ambiente YAWL, indicando assim que a execução da atividade será realizada pelo componente customizado desenvolvido.

[1]A versão do *Aspect Service* usada na prova de conceito está disponível em https://github.com/herculeshssj/unirio-ppgi-aspectservice
[2]A versão do *GoalService* usada na prova de conceito está disponível em https://github.com/herculeshssj/unirio-ppgi-goalservice

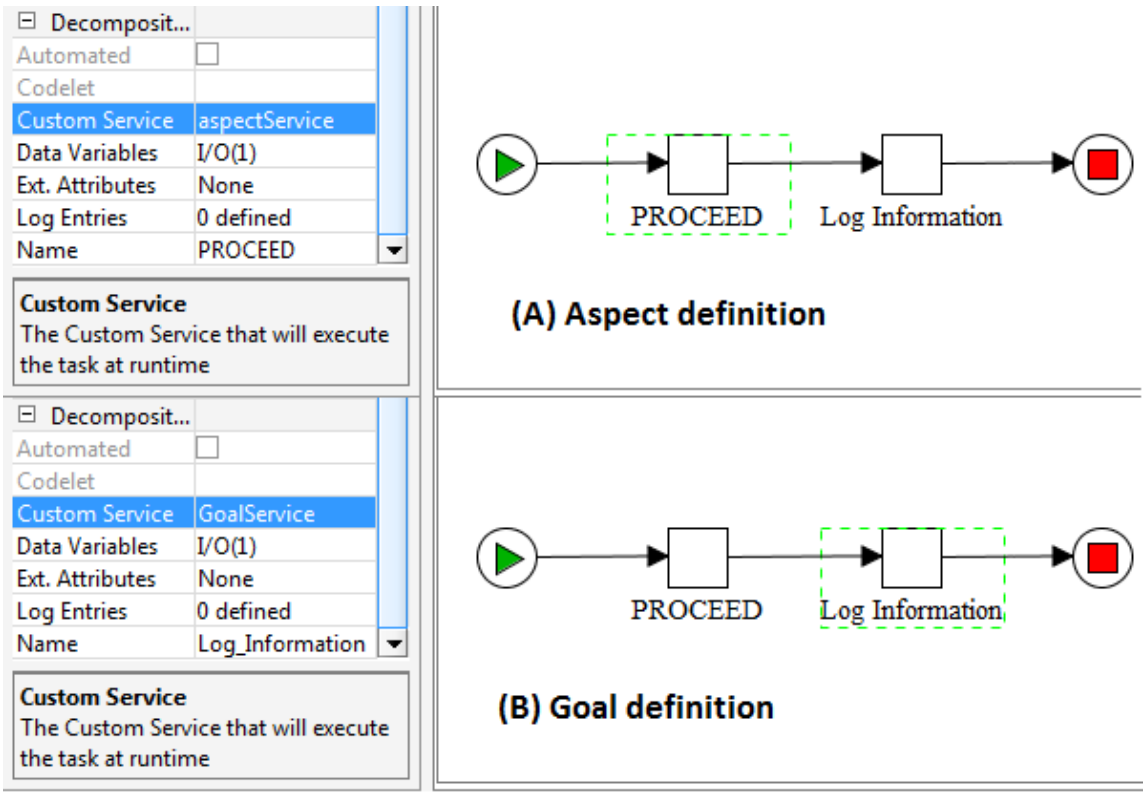


Figura 38 - Atribuição do *Aspect Service* a tarefa `PROCEED`, e do *Goal Service* na tarefa "*Log Information*", no Editor YAWL

A definição dos aspectos, seus *advices* e *pointcuts* para o *Aspect Service* foi feita usando a versão estendida do *Pointcut Editor*[3]. Junto com a definição dos *advices*, foi atribuído o objetivo (*goal*) que deve ser atingido, onde é informado o nome do objetivo. O nome do objetivo é o mesmo que está disponível no WSMX durante o processo de descoberta.

A ontologia, os *Web Services* e os objetivos foram escritos e validados usando a ferramenta *Web Service Modeling Toolkit* (WSMT)[4], que possui suporte nativo para a sintaxe WSML e integração com o ambiente WSMX, permitindo o carregamento das ontologias, *Web Services* e objetivos, e ainda executar o processo de descoberta, seleção e invocação do WSMX através da opção "*Achieve Goal*".

Foram definidos três objetivos, cada um indicando uma estratégia desejável de *logging*: *log* no console ou terminal ("*SimpleLogGoal"*), *log* por e-mail ("*MailLogGoal"*) e *log* em base de dados ("*DBLogGoal"*). As definições das três estratégias de *logging* foram criadas baseadas na ontologia proposta por Bastos et al. (2014), que também foi escrita em WSML usando o WSMT. Na Figura 39 temos a definição da ontologia com os

[3]A versão estendida do Pointcut Editor está disponível em https://github.com/herculeshssj/unirio-ppgi-pointcut-editor

[4] Web Service Modeling Toolkit, disponível em https://sourceforge.net/projects/wsmt/ .

conceitos de severidade (“*Severity*”), requisição (“*LogRequest*”) e resposta (“*LogEffect*”), com seus respectivos atributos. Temos também a definição de quatro instâncias do conceito “*Severity*”, indicando o nível de severidade a ser registrado junto com a mensagem enviada.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.uniriotec.br/aspect#",
    dc _"http://purl.org/dc/elements/1.1#"
}
ontology LogOntology

concept Severity
    name ofType _string
    level ofType _integer

concept LogRequest
    severity ofType Severity
    message ofType _string

concept LogEffect
    messageLogged ofType _boolean
    mailSent ofType _boolean
    logPersisted ofType _boolean
    messageShowed ofType _boolean

instance DEBUG memberOf Severity
    name hasValue "Debug"
    level hasValue 1

instance INFO memberOf Severity
    name hasValue "Information"
    level hasValue 2

instance WARN memberOf Severity
    name hasValue "Warning"
    level hasValue 3

instance ERROR memberOf Severity
    name hasValue "Error"
    level hasValue 4
```

Figura 39 - Ontologia de *logging*, baseada em Bastos et al. (2014), escrito em WSML

Cada objetivo espera que a instância retornada do conceito “*LogEffect*” venha com dois atributos com valores verdadeiros: o primeiro atributo é o “*messageLogged*”; o segundo atributo varia de acordo com o objetivo. Em “*MailLogGoal*” por exemplo (Figura 40), ele espera que o atributo “*mailSent*” seja verdadeiro, não levando em consideração os demais atributos do conceito “*LogEffect*”.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { _"http://www.uniriotec.br/aspect#",
    onto _"http://www.uniriotec.br/aspect#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
    dc _"http://purl.org/dc/elements/1.1#",
    discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#"
 }

goal MailLogGoal
    nfp
        dc#title hasValue "Mail log goal"
        dc#contributor hasValue "Hercules S. S. Jose"
        dc#description hasValue "Achieve the goal of send the log by e-mail"
    endnfp
    importsOntology {
        onto#LogOntology
    }
    capability MailLogGoalCapability
        nfp
            discovery#discoveryStrategy hasValue discovery#LightweightDiscovery
            discovery#discoveryStrategy hasValue discovery#NoPreFilter
        endnfp

        postcondition
            definedBy
                ?logEffect memberOf onto#LogEffect
                    and ?logEffect[messageLogged hasValue _boolean("true")]
                    and ?logEffect[mailSent hasValue _boolean("true")].
```

Figura 40 - Objetivo "*MailLogGoal*", escrito em WSML

Foram criadas também três descrições de *Web Services* em WSML ("*SimpleLog"*, "*MailLog"* e "*DBLog"*) cada uma tratando uma estratégia de *logging*, onde os detalhes da funcionalidade do serviço (sua capacidade) atende a um dos objetivos criados (Figura 38). Foram declarados também a interface do serviço que será invocado, com as respectivas regras para invocação do *Web Service*.

Para verificar os recursos de invocação foi desenvolvido um *Web Service* ("*LogService"*) contendo três métodos remotos, cada método indicando uma estratégia de *logging*. O *Web Service* foi criado usando a IDE Eclipse Mars for Java EE[5], configurada com o Apache Axis2[6]. Cada *Web Service* semântico foi definido com a interface de invocação de cada um dos métodos remotos, de modo a permitir que o WSMX invocasse o método remoto correspondente ao final do processo de descoberta e seleção do *Web Service* semântico. Por exemplo, ao se invocar o "*MailLog"* (Figura 41), o método correspondente no "*LogService"* é invocado.

[5] Eclipse IDE, disponível em https://eclipse.org/downloads/ .
[6] Apache Axis2, disponível em http://axis.apache.org/axis2/java/core/ .

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { _"http://www.uniriotec.br/aspect#",
    onto _"http://www.uniriotec.br/aspect#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
    dc _"http://purl.org/dc/elements/1.1#",
    discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#"
 }

webService MailLog
    nfp
        dc#title hasValue "Mail Log service"
        dc#contributor hasValue "Hercules S. S. Jose"
        dc#description hasValue "Send the information logged by e-mail"
        _"http://owner" hasValue _"http://www.uniriotec.br/aspect/logging"
    endnfp
    importsOntology {
        onto#LogOntology
    }
    capability MailLogCapability
        nfp
            discovery#discoveryStrategy hasValue discovery#LightweightDiscovery
            discovery#discoveryStrategy hasValue discovery#NoPreFilter
        endnfp

        postcondition
            definedBy
                ?logEffect memberOf onto#LogEffect
                    and ?logEffect[messageLogged hasValue _boolean("true")]
                    and ?logEffect[mailSent hasValue _boolean("true")].

interface MailLogGoalInterface
    choreography MailLogGoalChoreography
    stateSignature MailLogGoalStateSignature
        importsOntology onto#LogOntology
        in
            onto#LogRequest withGrounding _"http://localhost:8080/LogApp/service
        out
            onto#LogEffect withGrounding _"http://localhost:8080/LogApp/services

    transitionRules MailLogGoalTransitionRules
        forall {?request} with (?request memberOf onto#LogRequest) do
            add (_# memberOf onto#LogEffect)
        endForall
```

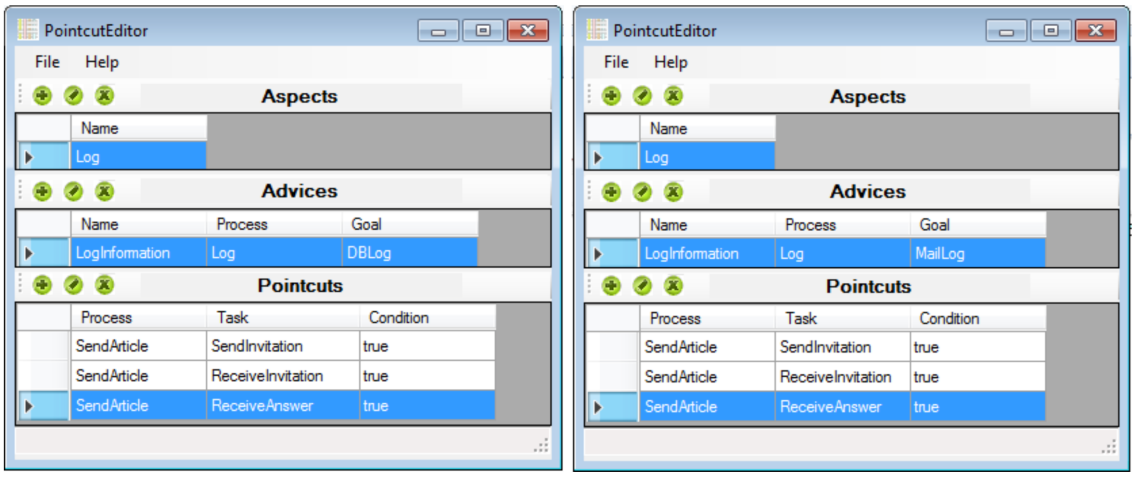
Figura 41 - Descrição semântica do *Web Service* "*MailLog*", escrito em WSML

No início da execução do processo "*Send Articles to Review*", o objetivo "*MailLog*" foi definido para a atividade "*Log Information*" usando o *Pointcut Editor*. Após a execução da atividade "*Send Invitation*", o *Aspect Service* foi invocado, fazendo o *weaving* do aspecto no processo principal. Assim que a atividade "*Log Information*" foi habilitada para execução, o *GoalService* entra em operação. Ao ser iniciado, o *GoalService* fez a invocação do WSMX e passa para a interface de descoberta o objetivo definido.

O processo de descoberta identifica os três serviços semânticos previamente elaborados comparando ontologias usadas, o que o serviço provê e modelagem dos objetos declarados com o que foi descrito no objetivo, no caso as regras declaradas na

seção de pós-condição. De acordo com as regras contidas em "*MailLogGoal*", onde o valor do atributo "*mailSent*" é definido para verdadeiro, o serviço escolhido foi o "*MailLog*". Uma vez selecionado o serviço, o WSMX retorna uma resposta para o *GoalService* contendo o identificador do serviço encontrado. Assim, o *GoalService* prossegue invocando a interface de invocação de serviços do WSMX, passando o identificador de "*MailLog*" como parâmetro. Finalizado esse processo, o controle da execução do processo volta para o YAWL dar prosseguimento.

Além disso, foi realizado durante a execução do processo a mudança do objetivo, de "*MailLogGoal*" para "*DBLogGoal*" (Figura 42). Quando o *GoalService* foi invocado na execução da atividade "*Receive Answer*", um novo objetivo foi enviado para o WSMX, e como resultado o serviço "*DBLog*" foi retornado para posterior invocação. Este procedimento em nenhum momento interrompeu a execução do processo principal, sendo esta mudança de objetivo ocorrido no momento em que a atividade "*Receive Answer*" aguardava uma entrada de usuário. A Figura 42 ilustra a mudança feita ao se alterar o objetivo através do *Pointcut Editor*.



(a) Goal "DBLog" set in the advice (b) Goal "MailLog" set in the advice

Figura 42 - *Pointcut Editor* ilustrando a mudança de objetivo durante a execução do aspecto

O WSMX disponibiliza uma interface gráfica, o *WSMX Monitor* onde é possível acompanhar o processo de descoberta, seleção e invocação dos serviços. A Figura 43 mostra a janela do *WSMX Monitor* exibindo o processo de descoberta e seleção, onde na tela da esquerda mostra a seleção do serviço "*MailLog*" definido no início da execução do processo. Na tela da direita mostra a seleção do serviço "*DBLog*" após a mudança de objetivo.

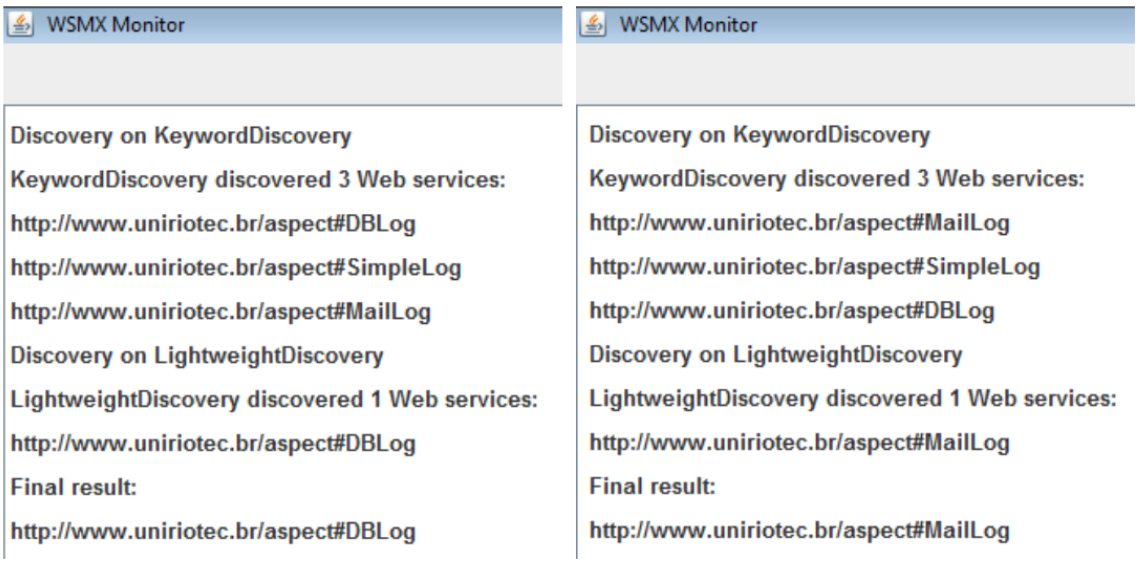


Figura 43 - *WSMX Monitor* ilustrando a mudança de objetivo durante a execução do aspecto

Os resultados da prova de conceito mostraram que é possível realizar com sucesso o processo de *weaving* dinâmico usando os artefatos desenvolvidos por Jalali et al. (2015), uma vez que o aspecto "*Log Information*" foi incorporado ao processo principal em tempo de execução. Foi possível também verificar o elo criado entre o ambiente YAWL e o WSMX através do componente customizado *GoalService*. Além disso, foi possível observar o processo de descoberta, seleção e invocação de um *Web Service* através de um objetivo semântico, escrito em WSML, e, com o apoio da versão estendida do *Pointcut Editor*, defini-lo no aspecto. Por fim, foi possível verificar a mudança no comportamento do aspecto quando outro *Web Service* foi selecionado para atender ao novo objetivo definido.

A partir da prova de conceito pudemos ver na prática que a abordagem proposta por Jalali et al. (2015) apoiou adequadamente a pesquisa, dando o suporte necessário ao ambiente YAWL para realizar o *weaving* dinâmico, e confirmamos também as evidências apontadas por Bastos et al. (2014) quanto a aplicação de um objetivo operacional para diminuir a lacuna entre as representações de aspectos nas fases de modelagem e implementação, além de validar a arquitetura proposta no Capítulo 3 desta dissertação.

4.2 Experimento

Segundo Recker (2013), estudos experimentais são aqueles que são especificamente destinados a examinar as relações de causa e efeito. A principal força da pesquisa experimental sobre outros métodos é a ênfase na validade interna devido à disponibilidade de meios para isolar, controlar e examinar variáveis específicas (a causa) e a consequência que elas provocam em outras variáveis (o efeito). Aplicando este conceito no presente

estudo, temos os objetivos operacionais definidos pelos especialistas (a causa) que serão submetidos ao processo de descoberta e identificar quantos serviços são selecionados para aquele objetivo (o efeito). O experimento é aplicável ao nosso estudo porque podemos construir cenários possíveis para descoberta de serviços e avaliar o quanto mudanças na definição dos objetivos e na quantidade de serviços no repositório podem afetar o resultado do processo de descoberta do serviço adequado para implementar o aspecto.

Assim, com o intuito de avaliar a proposta e confirmar a hipótese de pesquisa formulada ("*se um objetivo operacional é definido em um aspecto na fase de modelagem, então ele pode ser vinculado a uma implementação que cumpra com as intenções dos especialistas de negócio")*, foi conduzido um experimento simulando a definição de objetivos operacionais para aspectos pelos especialistas de negócio na fase de modelagem, e a construção de repositórios de serviços pelos desenvolvedores na fase de implementação para atender aos aspectos definidos.

O experimento teve como foco simular os seguintes cenários para a definição dos objetivos pelos especialistas:

- 1º cenário - Especialista define o objetivo em cima das descrições semânticas de um determinado serviço existente, que pode estar presente ou não no repositório construído. Objetivos a serem atingidos:
    - a) identificar o serviço que foi utilizado de base para definição do objetivo;
    - b) selecionar o serviço que foi utilizado de base para definição do objetivo;
    - c) confirmar se o serviço que foi utilizado de base para o objetivo é o primeiro da lista de serviços selecionados no processo de descoberta.
- 2º cenário - Especialista desconhece o repositório construído pelos desenvolvedores, e define o objetivo esperando que haja um serviço que implemente adequadamente o aspecto modelado.
- 3º cenário - Especialista conhece os conceitos e definições presentes na ontologia e que foram aplicados na definição semântica dos serviços presentes no repositório construído, e estabelece o objetivo do aspecto em cima desses conceitos e definições.

O 2º e 3º cenários compartilham os mesmos objetivos a serem atingidos, abaixo listados:

- a) verificar a quantidade de serviços identificados;
- b) verificar a quantidade de serviços selecionados;
- c) verificar se para cada objetivo definido tem pelo menos um serviço selecionado para realizar a invocação.

Já para analisar a influência do tamanho do repositório de serviços construído pelos desenvolvedores para o processo de descoberta, foram criados alguns repositórios de tamanhos distintos a partir do conjunto de serviços encontrados para o experimento, e cada cenário foi executado em cima dos repositórios criados.

Inicialmente procurou-se um repositório de *Web Services*, onde deveria conter pelo menos uma ontologia para um domínio de negócio e exemplos de *goals* que poderiam ser utilizados na simulação. Nas referências encontradas sobre WSMO foram encontrados somente exemplos prontos, todos direcionados para um estudo em particular, com uma quantidade pequena de *Web Services*. Usar estes exemplos iria impactar no estudo em virtude da alta possibilidade de sucesso das correspondências entre *Web Services* e *goals*.

Por conta disso decidiu-se usar o repositório SWS-TC[7] contendo uma coleção de 241 serviços descritos em OWL-S (MARTIN et al., 2004) para montar a coleção de serviços WSMO[8]. O repositório SWS-TC (convertido para OWL-S usando a ferramenta *Protégé*[9], conforme comentário encontrado nos arquivos) contém uma ontologia que reúne todos os conceitos presentes nos serviços da coleção. Os conceitos presentes na ontologia não possuem atributos definidos, contendo unicamente a definição de "*Class"* e "*subClassOf"*. Através do serviço remoto disponibilizado pelo site OWL-WSML Translator[10], foi possível realizar a conversão de OWL-S para WSML da ontologia e dos serviços, e assim construir nossa coleção de *Web Services* WSMO necessária para a simulação.

A partir dos 241 serviços WSMO gerados foram construídos 12 repositórios de *Web Services*, identificados de 1 a 12, acrescentando 20 *Web Services* a mais de um repositório a outro (o 1º com 20, o 2º com 40, o 3º com 60, e assim por diante). No 12º repositório criado, um *Web Service* foi propositalmente deixado de fora para avaliar o impacto de sua ausência. A seleção dos *Web Service* para cada repositório foi aleatória, e feita a partir dos 241 serviços, sendo realizada no momento da conversão dos arquivo OWL-S. Os arquivos foram numerados de 0 a 240, para facilitar na identificação, preservando o nome interno do serviço. A numeração do *Web Service* foi incluída nas propriedades não funcionais. Para manter a padronização na construção dos *Web Service* foi usado um dos arquivos de exemplo que acompanha a ferramenta WSMT (item a, Figura 44). Foi desenvolvido um utilitário em Java para realizar todo o processo de conversão dos serviços para WSML usando como apoio a biblioteca WSMO4J[11]. O item b da Figura 44

[7] http://projects.semwebcentral.org/projects/sws-tc/
[8] Os serviços OWL e SWS estão disponíveis em https://github.com/herculeshssj/unirio-ppgi-webservices
[9] http://protege.stanford.edu/
[10] http://tools.deri.org/wsml/owl2wsml-translator/v0.1/
[11] http://wsmo4j.sourceforge.net/

mostra um exemplo de *Web Service* após o processo de conversão.

BookStore.wsml | *WebService0.wsml

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-dl"
namespace { _"http://www.sti-innsbruck.at/services#",
    media _"http://www.sti-innsbruck.at/ontologies#",
        discovery _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#"
}

webService BookStore

importsOntology {media#MediaOntology}

capability BookStoreCapability

nonFunctionalProperties
  discovery#discoveryStrategy hasValue discovery#LightweightDLDiscovery
  discovery#discoveryStrategy hasValue discovery#NoPreFilter
endNonFunctionalProperties

sharedVariables ?x

postcondition bookPost
     definedBy
          ?x memberOf media#Book.
```

(A) WSML file from WSMT

BookStore.wsml | *WebService0.wsml

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://www.uniriotec.br/wsmo/services#"
,
     dc _"http://purl.org/dc/elements/1.1/",
     wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
     xsd _"http://www.w3.org/2001/XMLSchema#",
     targetnamespace _"http://www.uniriotec.br/wsmo/services#" }


webService _"http://www.uniriotec.br/wsmo/services/ABCHeadlines"
     nonFunctionalProperties
          _"http://purl.org/dc/elements/1.1#description" hasValue "Headline "
          _"http://purl.org/dc/elements/1.1#title" hasValue "Web Service 0"
     endNonFunctionalProperties

     importsOntology
            _"http://www.uniriotec.br/wsmo/ontology/Concepts.owl"

capability targetnamespace#ABCHeadlinesCapability

postcondition targetnamespace#ABCHeadlinesAxiom
     definedBy
          ?a memberOf _"http://www.uniriotec.br/wsmo/ontology/Concepts.owl#Headline".
```

(B) OWL-S converted to WSML

Figura 44 - Exemplo de arquivo WSML usado como modelo e OWL-S convertido para WSML

Foram definidos 70 *goals*, numericamente identificados, distintos entre si pelas definições dos conceitos constantes na seção *postcondition*. Os *goals* foram separados em três grupos distintos, cada um representando um cenário:

- 1º grupo (1 ao 10) – as definições eram baseadas em definições de *Web Services* existentes;

- 2º grupo (11 ao 40) – as definições foram definidas aleatoriamente a partir dos conceitos presentes na ontologia;
- 3º grupo (41 ao 70) – as definições foram definidas aleatoriamente a partir dos conceitos presentes no repositório contendo 60 *Web Services*.

Os grupos 2 e 3 ainda foram divididos em três subgrupos, cada subgrupo contendo 10 *goals* com uma variável, 10 contendo 2 variáveis, e 10 contendo 3 variáveis. A Tabela 4 traz um resumo das definições dos *goals*.

Tabela 4 - Resumo da definição dos objetivos (*goals*) para simulação

Grupo	Cenário	Quantidade de variáveis	Base para definição das variáveis
Grupo 1 (1 a 10)	1º cenário	Depende do *Web Service* escolhido	*Web Service* existente
Grupo 2 (11 a 20)	2º cenário	1	Aleatório, a partir da ontologia
Grupo 3 (21 a 30)	2º cenário	2	Aleatório, a partir da ontologia
Grupo 4 (31 a 40)	2º cenário	3	Aleatório, a partir da ontologia
Grupo 5 (41 a 50)	3º cenário	1	Aleatório, baseado em conceitos presentes em repositório criado
Grupo 6 (51 a 60)	3º cenário	2	Aleatório, baseado em conceitos presentes em repositório criado
Grupo 7 (61 a 70)	3º cenário	3	Aleatório, baseado em conceitos presentes em repositório criado

A Figura 45 mostra um exemplo de *goal* construído. Ele segue a mesma padronização adotada nos *Web Services*. Usou-se a classe *java.util.Random*[12] para seleção dos *Web Services* e dos conceitos usados nas variáveis.

[12] https://docs.oracle.com/javase/7/docs/api/java/util/Random.html

Figura 45 - Exemplo de objetivo (*goal*) construído para simulação

Por fim, para simular a execução do aspecto no ambiente YAWL, foi usado o componente customizado *GoalService* para realizar o processo de descoberta do serviço a partir dos *goals* definidos. O utilitário em linha de comando chamado *YSimulation*[13] foi adaptado para permitir a execução sequencial das instâncias do aspecto "*Log Information*". A lista de *Web Services* identificados e selecionados foram registrados em uma base de dados PostgreSQL[14], onde foi possível fazer a análise do processo de descoberta.

O experimento foi realizado em uma máquina virtual Windows 7 Home Premium 64 bits com 8 GB de RAM, rodando sob o VirtualBox versão 5.1.14 em um MacBook Pro com CPU Intel Core i7 2.3 GHz, com 16 GB de RAM e com o sistema operacional OS X 10.11.6 El Capitan. A máquina virtual estava utilizando a versão do Java JDK 1.7_80 32 bits e banco de dados PostreSQL 9.4.

Todos os 70 *goals* foram executados nos 12 repositórios criados, totalizando 840 execuções de identificação e seleção. Foram registrados quantos e quais *Web Services* foram identificados, e destes quantos e quais foram selecionados. Foi utilizado a configuração padrão do WSMX de descoberta, que consiste em verificar a correspondência entre *goal* e *Web Service* por palavra-chave ("*Keyword Discovery"*) e depois por similaridade nas declarações constantes nas seções *Capability* e *Interface* ("*Lightweight Discovery"*). Na correspondência por palavra-chave é comparada as

[13] *YSimulation* acompanha o código fonte do YAWL, disponível em https://github.com/yawlfoundation/yawl
[14] https://www.postgresql.org/

palavras existentes nas definições das propriedades não funcionais, e retornado até 6 serviços candidatos para a próxima etapa. Na correspondência por similaridade nas declarações, três tipos de correspondências (*Exact, PlugIn e Subsumption*) são aplicados nas declarações para retornar o(s) *Web Service(s)* que atendem ao *goal* enviado. A Tabela 5 resume o resultado da execução.

Tabela 5 - Resumo da execução da simulação

	1º Cenário		**2º Cenário**						**3º Cenário**					
Grupo	**1º**		**2º**		**3º**		**4º**		**5º**		**6º**		**7º**	
Repositório	I	S	I	S	I	S	I	S	I	S	I	S	I	S
1	44	1	60	1	60	-	60	-	55	5	56	-	57	-
2	56	3	58	6	59	-	58	-	60	6	60	-	57	2
3	59	2	55	3	54	-	56	-	59	10	56	1	55	2
4	59	6	60	6	53	-	60	-	56	1	55	1	55	3
5	59	7	57	7	56	1	60	-	59	6	59	3	55	1
6	55	12	60	4	60	-	58	-	54	8	58	1	56	2
7	59	13	60	8	56	1	58	-	56	10	55	2	56	2
8	57	13	60	5	58	1	55	-	52	8	54	-	60	4
9	60	13	60	9	60	1	52	-	50	7	53	2	60	4
10	59	16	60	10	60	-	53	-	57	9	58	3	60	3
11	58	15	60	9	60	-	55	-	53	9	59	2	60	3
12	59	21	60	10	60	-	58	-	54	9	59	2	60	3

Legenda: I – Total de serviços identificados; S – Total de serviços selecionados

No 1º cenário, onde os *goals* possuem sua descrição semântica idêntica a de *Web Services* existentes, podemos observar um crescente aumento da quantidade de serviços selecionados, indicando que quanto mais serviços estiverem presentes no repositório, maiores as chances de obter uma correspondência. Contudo, ao analisar a listagem de *Web Services* selecionados com a listagem que originou os *goals* do 1º grupo, temos muitos falsos positivos nos primeiros 6 repositórios, ou seja, seleção de serviços que não cumprem com aquele *goal*. Este comportamento era esperado em virtude das características da coleção de serviços selecionada, pois temos serviços como "*FindClosestFirehouse*" que faz uso dos mesmos conceitos ("*Latitude*" e "*Longitude*") que "*FindLatLong*", e também serviços como "*EMailValidator*" que faz uso de conceitos como "*EMailAddress*" que estão presentes em diversos serviços da coleção. Além disso, à medida que a quantidade de *Web Services* aumenta, obtemos mais de uma seleção para o mesmo *goal*, dificultando determinar qual dos serviços escolhidos realmente atende ao que foi especificado no *goal*.

No 2º cenário identificamos um baixo nível de correspondência, pois nem todos os *Web Services* identificados terão as definições presentes no *goal*. Para o 2º grupo de *goals* temos uma quantidade significativa de serviços selecionados, para o 3º grupo temos apenas 4 seleções, e no 4º grupo não temos serviços selecionados. Tal fato ocorre em

virtude da definição aleatória dos *goals*, onde nenhum critério foi aplicado para especificá-los. No 2º grupo, temos a vantagem de a comparação dos serviços identificados ser feita com a primeira variável definida na seção de pós-condições, mesmo quando o conceito usado pela variável foi escolhido aleatoriamente. Já para o 3º grupo, esta correspondência não é tão simples, em virtude de que as duas variáveis são comparadas, dificultando obter uma correspondência exata mesmo com todos os serviços. Era esperado um baixo nível de correspondência, mas não tanto quanto foi observado. Ao analisarmos os *goals* gerados para o 3º e 4º grupo, temos conceitos como "*AreaCode*", "*Group*" e "*CheckResult*" que estão presentes em pelo menos um serviço, mas não iremos encontra-los juntos em um *Web Service* da coleção.

No 3º cenário já temos resultados mais promissores, em virtude dos *goals* usarem definições semânticas obtidas de um repositório existente, e não geradas ao acaso como ocorreu no 2º cenário. Os conceitos usados foram extraídos do repositório 3, onde contém 60 *Web Services.* Conceitos como "*Address*" e "*DateTime*" são encontrados com mais frequência nos *Web Services* do que "*Yellowpage*", por exemplo. Ao montar a definição dos *goals* aleatoriamente, são grandes as chances de aparecer conceitos não presentes nos *Web Services*. Podemos observar que até para *goals* contendo 3 variáveis houve alguma correspondência. Tal fato ocorreu em virtude de pelo menos um dos serviços identificados conseguiu uma intersecção entre *goal* e *Web Service*.

4.3 Análise dos Resultados e Discussão

Na seção anterior foram criados três cenários para simular a definição de objetivos operacionais para aspectos pelos especialistas de negócio, com o intuito de averiguar três situações em particular: especialista conhece o serviço que implementa o aspecto, mas não sabe se o serviço está presente no repositório construído pelo desenvolvedor (1º cenário); especialista desconhece qualquer característica dos serviços presentes no repositório (2º cenário); e especialista define o objetivo em cima de conceitos presentes em um repositório construído pelo desenvolvedor (3º cenário). Todos os cenários criados foram executados sob 12 repositórios criados a partir de uma coleção de serviços OWL convertidos para WSMO. A criação desses repositórios teve o intuito de averiguar se a quantidade de serviços existentes impacta no processo de descoberta, e nos resultados obtidos em cada cenário.

Analisando os dados apresentados ao 1º cenário na Tabela 5, tivemos um aumento crescente no número de serviços identificados e serviços selecionados. Este aumento ocorreu devido ao aumento da quantidade de serviços presentes no repositório, aumentando assim a probabilidade do *Web Service* que foi usado para criar o *goal* estar presente no repositório. Igualmente, também elevamos a probabilidade de termos mais

serviços que podem atender àquele *goal*, contudo não necessariamente atender a intenção do especialista.

A Figura 46 mostra especificamente o resultado da seleção de *Web Services* para cada um dos 10 *goals* do Grupo 1, onde indicamos se dentre os 10 *goals*, quantos tiveram *Web Services* selecionados (em azul) para invocação. Dentre os *goals* com *Web Services* selecionados, foi verificado se o *Web Service* que deu origem ao *goal* estava presente na seleção (em vermelho), e se o mesmo era o primeiro da lista (em amarelo). Pelo gráfico podemos constatar que o fato de um *goal* ter sido criado a partir de um *Web Service* não garante necessariamente que ele será o primeiro da seleção, mas há grande probabilidade de ele ser um dos serviços selecionados.

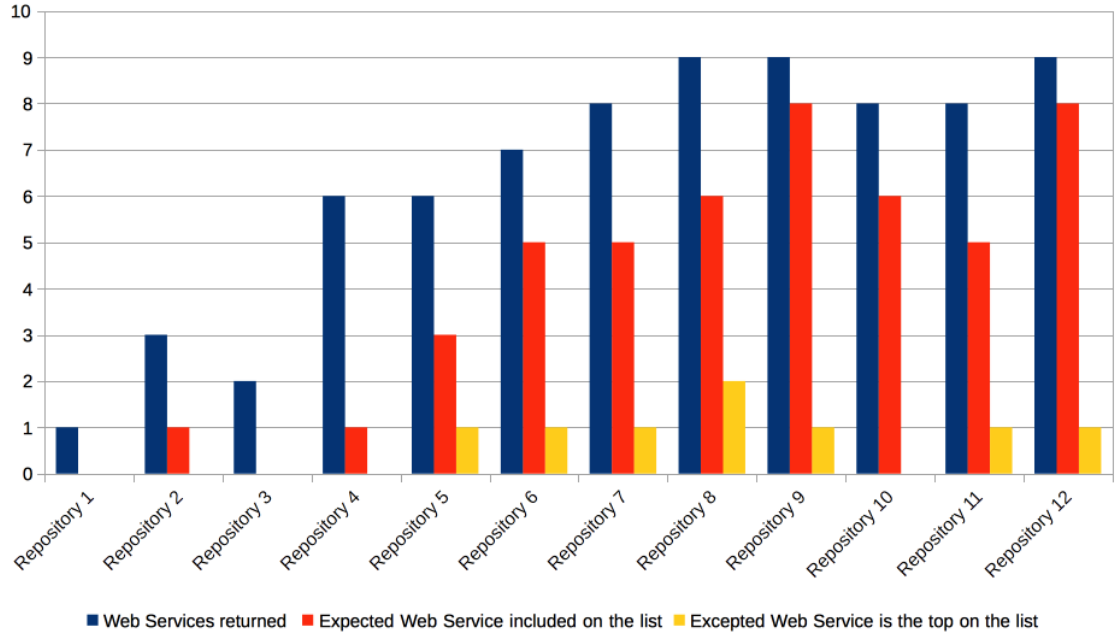


Figura 46 - Resultado da seleção de *Web Services* do 1º cenário

Outro ponto que o gráfico mostra é que mesmo com toda a coleção de *Web Services*, ainda temos um *goal* que teve nenhuma seleção. O que pode justificar é o fato da ausência de um *Web Service* no repositório 12, contudo confirmamos após a simulação que todos os *Web Services* usados para criar os *goals* do Grupo 1 estavam presentes, só que não foi o primeiro a ser selecionado na listagem retornada de serviços, indicando assim que a presença de muitos serviços com definições semelhantes afeta o processo de descoberta.

Analisando o 2º cenário, onde os especialistas desconhecem os conceitos presentes no repositório, a definição arbitrária dos *goals* afetou o processo de descoberta, pois não foi possível em grande parte coincidir as definições criadas aleatoriamente com as definições presentes na coleção de serviços. Mas, ao comparar com o 3º cenário, quando o especialista conhece as definições do repositório, as chances são maiores de obter um serviço que atende ao especialista.

A expectativa para o 2º e 3º cenário era a proporção de um serviço selecionado para

cada *goal* criado. No total, 30 *goals* foram criados para cada cenário, onde cada *goal* foi submetido aos 12 repositórios, totalizando 360 execuções por cenário, das quais era esperado um terço delas (120 execuções). A Tabela 6 resume os dados da execução, onde temos uma diferença de 25,25% entre o resultado esperado e o 2º cenário, e de 2,75% entre o resultado esperado e o 3º cenário.

Tabela 6 - Resumo das execuções bem-sucedidas para o 2º e 3º cenário

	Execuções	Percentagem
Total	360	100%
Esperado	120	33,3%
2º cenário	29	8,05%
3º cenário	110	30,55%

Os dados da execução comprovam o impacto causado pelo desconhecimento das características do repositório de serviços pelos especialistas. Quanto mais alinhado o especialista estiver dos conceitos presentes no repositório, maiores as chances do *goal* ser atingido. Comparando com os resultados apresentados na Tabela 5, a quantidade de serviços no repositório impacta diretamente no processo de descoberta. Ao calcular a variância dos resultados do 2º e 3º cenário (Figura 47), obtemos uma dispersão maior nos repositórios contendo quase todos os serviços da coleção (repositório de 10 a 12), e picos na dispersão em virtude do sucesso na descoberta de serviços onde para cada *goal* houve um serviço selecionado.

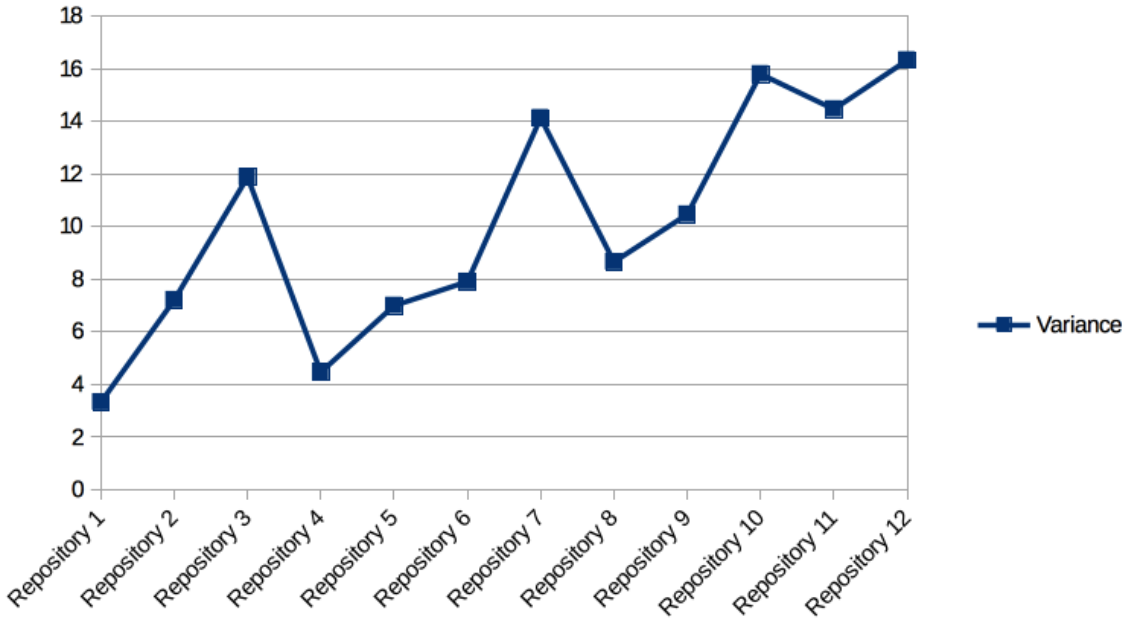


Figura 47 - Impacto da quantidade de serviços para o processo de descoberta

Seria possível obter melhores resultados com a descoberta de serviços, principalmente para o 2º cenário, contudo a configuração padrão do ambiente WSMX retorna apenas 6

Web Services, e a versão do ambiente usada não possui todos os tipos de correspondências (o *Intersection Match* não foi implementado, informação obtida ao analisar o código fonte). Outro ponto limitador é a influência da identificação por palavra-chave ("*Keyword Discovery*"), onde se leva em consideração as informações contidas nas propriedades não funcionais do *goal* e do *Web Service*. Procurou-se definir as propriedades não funcionais de forma a impactar o mínimo possível na identificação dos serviços, sem direcionar para um grupo particular de serviços. E por fim, as características da própria coleção de serviços, onde temos na ontologia apenas o nome os conceitos, sem seus respectivos atributos, e a predominância de um conjunto de conceitos em partes específicas da coleção. Por exemplo, o conceito "*Book*" aparece mais nos serviços iniciados com letra A, B e C, e não aparece muito nos serviços iniciados com letra G.

A definição de objetivo operacional permite criar um elo entre as fases, desacoplando tanto linguagem de modelagem de processos quanto de linguagem de programação. Ao compararmos os resultados obtidos com os objetivos definidos nos cenários da simulação, constatamos a criação desse elo ao permitir o aspecto encontrar uma implementação adequada para ele a partir de um objetivo operacional definido na fase de modelagem. Os resultados obtidos também constatam que o especialista pode se concentrar na modelagem do aspecto sem necessariamente se preocupar com sua implementação, uma vez que a implementação será descoberta e invocada durante a execução do processo, desde que o objetivo definido ao aspecto seja baseado nos conceitos presentes no repositório de serviços que será usado. Para o sucesso da abordagem é necessário o uso dos conceitos presentes na ontologia, tanto pelo especialista para descrição dos objetivos, quanto pelo desenvolvedor para descrição semântica dos serviços disponibilizados no repositório. A Tabela 7 mostra os objetivos de cada cenário e quais foram atendidos e atendidos parcialmente.

Tabela 7 - Objetivos atingidos por cada cenário

1º cenário			2º cenário			3º cenário		
a)	b)	c)	a)	b)	c)	a)	b)	c)
✓	✓	!	✓	!	!	✓	✓	✓

Legenda: (✓) atendido; (!) atendido parcialmente

5. Conclusões

A orientação a aspectos em processos de negócio visa reduzir a complexidade dos modelos e possibilitar sua reutilização. A literatura demonstrou as vantagens desta abordagem em diferentes fases do ciclo de vida do BPM. No entanto, a fim de adotar uma abordagem única para ambos os estágios de modelagem e implementação, é necessário que as representações de aspectos compartilhem um vocabulário comum.

Este trabalho estendeu a pesquisa de Bastos et al. (2014) para definir características comportamentais para os aspectos a fim de estabelecer uma ligação entre modelagem e implementação. Esta ligação pretende diminuir a lacuna existente entre as fases. Os artefatos propostos nesta dissertação permitem a um BPMS selecionar uma implementação para o aspecto de acordo com o objetivo operacional definido, de modo que, durante a execução do processo, esta implementação possa ser descoberta e invocada. Assim, obtemos flexibilidade na realização dos objetivos, e tornamos os aspectos mais adaptáveis durante a fase de implementação.

5.1 Contribuições do Trabalho

Este trabalho evidenciou que o uso da abordagem proposta, suportada por um artefato sendo executado em um BPMS, permite criar uma ligação semântica entre a modelagem de um aspecto e sua respectiva implementação a partir de um objetivo operacional definido. Quanto maior for a riqueza de detalhes sobre o serviço desejado, maiores as chances de ele ser selecionado em um repositório de serviços.

A pesquisa de Jalali et al (2015) procurou habilitar um BPMS a incorporar aspectos no processo principal na prática. Nossa abordagem avança nessa questão trazendo o conceito de objetivo operacional para o aspecto, então diferentes implementações podem ser usadas para realizar as atividades de cada aspecto.

Outras questões foram levantadas por Jalali et al. (2015). Uma delas é que algumas vezes diferentes opções de modelagem são possíveis, por exemplo, uma atividade chamada “Arquivar” em um *advice* pode representar vários tipos de aspectos, como

logging. Nosso estudo avança nesta questão permitindo diferentes comportamentos para um mesmo *advice* bastando a definição de um objetivo operacional a ser atingido, obtendo assim flexibilidade e adaptabilidade para o aspecto.

5.2 Limitações

O uso do WSMO como linguagem semântica trouxe seus desafios. O fato de serem ferramentas de código aberto permitiu avançar com a pesquisa. A falta de um repositório de serviços foi uma limitação para a pesquisa, pois parte dos endereços indicados nos artigos sobre o WSMO e WSMX não estão mais disponíveis, e todos os exemplos encontrados foram desenvolvidos para uma realidade específica do projeto de pesquisa que os confeccionou.

Outra limitação identificada no uso do WSMO foi que os conceitos aqui expostos são aplicáveis somente para *Web Services* SOAP. Esta limitação restringe o uso da nossa solução no quesito de invocação de serviços. Não identificamos trabalhos recentes que visem permitir o WSMO a trabalhar, por exemplo, com *Representational State Transfer* (REST), um estilo arquitetural que utiliza tecnologias e protocolos comumente encontrados na Web, como HTTP (RICHARDS, 2006).

A solução proposta neste trabalho faz uso dos princípios SOA para auxiliar a descoberta de serviços. Assim, a solução é projetada para BPMS com capacidade de orquestração e coreografia de *Web Services*. Ambientes de modelagem de processo sem este suporte podem ser habilitados para composição de serviços, e consequentemente suportar nossa solução, através de implementação nativa ou através de componentes customizados.

Parte-se do princípio que o repositório criado contém implementação de requisitos não funcionais. A inclusão de outros tipos de serviços pode impactar a modelagem do aspecto, principalmente se os objetivos estão definidos a partir das especificações dos serviços disponíveis no repositório. Além disso, a implementação atual dos artefatos não foi validada em um contexto onde existe a necessidade de resolver conflitos entre terminologias das ontologias usadas pelo *Web Service* e o *Goal*, através do uso de mediadores. Diferenças entre terminologias impacta diretamente no processo de descoberta e seleção, uma vez que acusa falha na correspondência entre *Web Service* e *Goal*.

5.3 Trabalhos Futuros

Como trabalho futuro, pretendemos investigar mais a questão de descoberta e seleção de serviços retornados pelo WSMX ao componente *GoalService*. Com suporte de ferramentas de simulações e com uso de metadados para descrição dos serviços,

determinar a acurácia na busca e seleção nos casos de dois ou mais *Web Services* serem retornados para o mesmo objetivo, ou de nenhum *Web Service* ser retornado. Somado a isso, verificar o impacto do uso de mediadores e de realizar a compatibilidade entre ontologias distintas.

Além disso, investigaremos uma questão específica sobre como a nossa abordagem poderia lidar com conflitos entre diferentes aspectos (por exemplo, entre a privacidade de informações pessoais e as estratégias de coleta de informações, ou entre privacidade de informações pessoais e políticas de registro). A condução de um estudo de caso poderá ajudar a apontar novos horizontes para a pesquisa, indicando quais pontos ainda precisam ser tratados.

Também pretendemos estender a solução para torna-la a mais abstrata possível, para não ficar presa a tecnologias e BPMS específicos, assim como generalizar a descrição do objetivo para facilitar o trabalho do especialista de negócio ao modelar o objetivo. Trabalhar com outros tipos de serviços, como REST, usando *wrappers* por exemplo, avaliar outras linguagens semânticas como OWL e WSMO-Lite e simplificar a modelagem dos objetivos através de ferramentas que gerem o código WSML e permita enviar o objetivo direto ao WSMX. E por fim realizar estudos de casos envolvendo especialistas de modelagem de processos focando no aperfeiçoamento da metodologia, da arquitetura e dos artefatos.

Referências Bibliográficas

AGUILAR-SAVÉN, R. S. Business process modelling: Review and framework. **International Journal of Production Economics**, v. 90, n. 2, p. 129–149, 2004.

ALONSO, G.; CASATI, F.; KUNO, H.; MACHIRAJU, V. **Web Services - Concepts, Architectures and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

BASTOS, A.; SANTORO, F. M.; SIQUEIRA, S. W. M. Bringing Semantics to Aspect-Oriented Business Process Management. In: **Business Process Management Workshops**. Beijing, China: Springer International Publishing, 2014. v. 171p. 291–302.

BERG, K. VAN DEN; CONEJERO, J. M.; CHITCHYAN, R. **AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation.** EnschedeAOSD-Europe, maio 2005. Disponível em: <http://www.aosd-europe.net/deliverables/d9.pdf>. Acesso em: 18 jul. 2015.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. **Scientific american**, v. 284, n. 5, p. 28–37, 2001.

BRANDÃO, B. C. P. **Towards Aspects Identification in Business Process Through Process Mining**. Universidade Federal do Estado do Rio de Janeiro, 2015.

BRANDÃO, B. C. P.; SANTORO, F. M.; AZEVEDO, L. G. **Towards Aspects Identification in Business Process Through Process Mining**. (S. W. M. Siqueira, S. T. de Carvalho, Eds.). XI Brazilian Symposium on Information System. Goiânia, GO, Brazil: Instituto de Informática, Universidade Federal de Goiás, 2015.

BRAY, T.; HOLLANDER, D.; LAYMAN, A.; TOBIN, R.; THOMPSON, H. S. **Namespaces in XML 1.0**. W3C Recommendation, 2009. Disponível em: <https://www.w3.org/TR/REC-xml-names/>. Acesso em: 23 jan. 2017.

BUNGE, M. **Treatise on basic philosophy - Ontology I: the furniture of the world**. Springer Science & Business Media, 1977. v. 3

BUNGE, M. **Treatise on basic philosophy - Ontology II: A World of Systems**. Dordrecht, Holland. Springer Science & Business Media, 1979. v. 4

BUSSLER, C.; CIMPIAN, E.; FENSEL, D.; GOMEZ, J. M.; HALLER, A.; HASELWANTER, T.; KERRIGAN, M.; MOCAN, A.; MORAN, M.; OREN, E.; SAPKOTA, B.; TOMA, I.; VISKOVA, J.; VITVAR, T.; ZAREMBA, M.; ZAREMBA, M. **Web Service Execution Environment (WSMX). W3C Member Submission 3 June 2005**. Disponível em: <https://www.w3.org/Submission/WSMX/>. Acesso em: 21 dez. 2015.

CAPPELLI, C.; LEITE, J. C. S. P.; BATISTA, T.; SILVA, L. **An aspect-oriented approach to business process modeling**. Proceedings of the 15th workshop on Early aspects - EA '09. Charlottesville, VA, USA: 2009.

CAPPELLI, C.; SANTORO, F. M.; LEITE, J. C. S. P.; BATISTA, T.; MEDEIROS, A. L.; ROMEIRO, C. S. C. Reflections on the modularity of business process models: The case for introducing the aspect-oriented paradigm. **Business Process Management Journal**, v. 16, n. 4, p. 662–687, 2010.

CARSTEN, B. **Bruce Carsten: Lifetime Achievement Award Winner**. Power Electronics Technology, 2006. Disponível em: <http://powerelectronics.com/site-files/powerelectronics.com/files/archive/powerelectronics.com/power_systems/switch_mode_power_supplies/609PETOnline Lifetime Achievement.pdf>. Acesso em: 28 fev. 2017.

CHARFI, A.; MEZINI, M. AO4BPEL: An aspect-oriented extension to BPEL. **World Wide Web**, v. 10, n. 3, p. 309–344, 2007.

CHARFI, A.; MÜLLER, H.; MEZINI, M. Aspect-Oriented Business Process Modeling with AO4BPMN. In: **ECFMA 2010. Paris, France, 2010**. LNCS v. 6138p. 48–61.

CHARFI, A.; ZHI, H. Aspect-Based Realization of Non-functional Concerns in Business Processes. In: BOUAJJANI, A.; FAUCONNIER, H. (Eds.). **Networked Systems : Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers**. Cham: Springer International Publishing, 2015. p. 140–154.

CLARK, T. Uniform Resource Identifier (URI). In: DUBITZKY, W. et al. (Eds.). **Encyclopedia of Systems Biology**. New York, NY: Springer New York, 2013. p. 2319–2320.

COLLELL, D. C. **Aspect-Oriented Modeling of Business Processes**. Technischen Universitat Darmstadt, 2012.

CÔTÉ, R. G. Simple Object Access Protocol (SOAP). In: DUBITZKY, W. et al. (Eds.). . **Encyclopedia of Systems Biology**. New York, NY: Springer New York, 2013. p. 1941.

DA SILVA, Lyrene Fernandes. An aspect-oriented approach to model requirements. In: **Engineering Conference**. 2005. p. 24.

DAVENPORT, T. H. Reengenharia de processos. In: **Rio de Janeiro: Campus** , 1994. p. 6–8.

DE BRUIJN, J. **The WSML Specification**. WSML Working Draft, 2008. Disponível em: <http://www.wsmo.org/TR/d16/>. Acesso em: 17 fev. 2017.

DUMAS, M.; LA ROSA, M.; MENDLING, J.; REIJERS, H. A. **Fundamentals of Business Process Management**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

EL BOUHISSI, H.; MALKI, M.; CHERIF, M. A. S. A. From User's Goal to Semantic Web Services Discovery: Approach Based on Traceability. **International Journal of Information Technology and Web Engineering**, v. 9, n. 3, p. 15–39, jan. 2014.

FENSEL, D.; FACCA, F. M.; SIMPERL, E.; TOMA, I. Web Service Modeling Ontology. In: **Semantic Web Services**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. v. 1p. 107–129.

FENSEL, D.; LAUSEN, H.; DE BRUIJN, J.; STOLLBERG, M.; ROMAN, D.; POLLERES, A.; DOMINGUE, J. **Enabling Semantic Web Services**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. v. 1

FENSEL, D.; BUSSLER, C. The Web Service Modeling Framework WSMF. **Electronic Commerce Research and Applications**, v. 1, n. 2, p. 113–137, jun. 2002.

FENSEL, D.; KERRIGAN, M.; ZAREMBA, M. **Implementing Semantic Web Services**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

GARCIA, R. **O que é Programação Orientada a Aspectos?** Java Framework Portal, 2010. Disponível em: <http://www.javaframework.org/portal/2010/04/14/o-que-programao-orientada-a-aspectos/>. Acesso em: 20 ago. 2016.

HOLLINGSWORTH, D.; HAMPSHIRE, U. K. Workflow management coalition the workflow reference model. **Workflow Management Coalition**, v. 68, p. 26, 1993.

JALALI, A. **Foundation of Aspect Oriented Business Process Management**. Stockholm University, 2011.

JALALI, A.; WOHED, P.; OUYANG, C.; JOHANNESSON, P. Dynamic Weaving in Aspect Oriented Business Process Management. **On the Move to Meaningful Internet Systems: OTM 2013 Conferences**, v. 8185, p. 2–20, 2013.

JALALI, A. Assessing Aspect Oriented Approaches in Business Process Management. **Lecture Notes in Business Information Processing**, v. 194, p. 231–245, 2014.

JALALI, A.; OUYANG, C.; WOHED, P.; JOHANNESSON, P. Supporting aspect orientation in business process management. **Software & Systems Modeling**, p. 1–23, 24 set. 2015.

JOSUTTIS, N. M. **SOA in practice: the art of distributed system design**. O'Reilly Media, Inc., 2007.

KELLER, G.; TEUFEL, T. **SAP R/3 process oriented implementation**. Addison-Wesley Longman Publishing Co., Inc., 1998.

KELLER, U.; LARA, R.; POLLERES, A.; TOMA, I.; KIFER, M.; FENSEL, D. **WSMO web service discovery**. WSML Working Draft, 2004. Disponível em: <http://wsmo.org/2004/d5/d5.1/v0.1/20041112/d5.1v0.1_20041112.pdf>. Acesso em: 17 jul. 2016.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J.; IRWIN, J. Aspect-oriented programming. **Computer Science**, v. 1241/1997, p. 220–242, 1997.

LADDAD, R. **AspectJ in action: practical aspect-oriented programming**. Dreamtech Press, 2003.

MANKOVSKII, S.; GOGOLLA, M.; URBAN, S. D.; DIETRICH, S, W.; YANG, M.; DOBBIE, G.; LING, T. W.; HALPIN, T.; KEMME, B.; SCHWEIKARDT, N.; ABELLÓ, A.; ROMERO, O.; JIMENEZ-PERIS, R.; STEVENS, R.; LORD, P.; GRUBER, T.; LEENHEER, P. D.; GAL, A.; BECHHOFER, S.; PATON, N. W.; LI, C.; BUCHMANN, A.; HARDAVELLAS, N.; PANDIS, I.; LIU, B.; SHAPIRO, M.; BELLATRECHE, L.; GRAY, P. M. D.; AALST, W. M. P.; PALMER, N.; RISCH, T.; GALUBA, W.; GIRDZIJAUSKAS, S; BECHHOFER, S . OWL: Web Ontology Language. In: LIU, L.;

ÖZSU, M. T. (Eds.). . **Encyclopedia of Database Systems**. Boston, MA: Springer US, 2009. p. 2008–2009.

MARTIN, D.; BURSTEIN, M.; HOBBS, J.; LASSILA, O.; MCDERMOTT, D.; MCILRAITH, S.; NARAYANAN, S.; PAOLUCCI, M.; PARSIA, B.; PAYNE, T. OWL-S: Semantic markup for web services. **W3C member submission**, v. 22, p. 2004–2007, 2004.

OBJECT MANAGEMENT GROUP, I. **Meta object facility (MOF) specification v1.4.** 2002.

OMG. **Business Process Model and Notation**. 2011.

PAWLAK, R.; SEINTURIER, L.; RETAILLÉ, J.-P. The Concepts of AOP. **Foundations of AOP for J2EE Development**, p. 7–21, 2005.

PINHEIRO, J. M. S. **Prova de Conceito no Projeto de Redes de Computadores**. Projeto de Redes, 2010. Disponível em: <http://www.projetoderedes.com.br/artigos/artigo_prova_de_conceito_no_projeto_de_redes.php>. Acesso em: 28 fev. 2017.

RAMOS, Ricardo A.; ALENCAR, F; ARAÚJO, J; MOREIRA, A; CASTRO, J. i* with Aspects: Evaluating Understandability. In: **WER**. 2007. p. 171-178.

RECKER, J. **Scientific Research in Information Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

RICHARDS, R. Representational State Transfer (REST). In: RICHARDS, R. (Ed.). . **Pro PHP XML and Web Services**. Berkeley, CA: Apress, 2006. p. 633–672.

SANTOS, F. J. N.; CAPPELLI, C.; SANTORO, F. M.; LEITE, J. C. S. P.; BATISTA, T. V. **Using goals to identify aspects in business process models**. Proceedings of the 2011 international workshop on Early aspects - EA '11. New York, New York, USA: ACM Press, 2011.

SANTOS, F. J. N.; CAPPELLI, C.; SANTORO, F. M.; LEITE, J. C. S. P.; BATISTA, T. V. Aspect-oriented business process modeling: analyzing open issues. **Business Process Management Journal**, v. 18, n. 6, p. 964–991, 2 nov. 2012.

SHANKARDASS, A. **The dynamic adaptation of an aspect oriented business process in a service oriented architecture platform**. Edmonton, AB: Cambridge University Press, 2009.

SOFFER, P.; WAND, Y. Goal-Driven Analysis of Process Model Validity. In: **Advanced Information Systems Engineering**. CAiSE 2004. Springer Berlin Heidelberg, 2013. LNCS v.3084, p. 229–319.

SOFFER, P.; WAND, Y. On the notion of soft-goals in business process modeling. **Business Process Management Journal**, v. 11, n. 6, p. 663–679, 2005.

VAN DER AALST, W. M. P.; TER HOFSTEDE, A. H. M. YAWL: yet another workflow language. **Information Systems**, v. 30, n. 4, p. 245–275, jun. 2005.

WANG, J.; ZHU, J.; LIANG, H. **Aspect-Oriented Business Process Modeling**. IBM Research. Beijing, China, 2005.

WESKE, M. **Business Process Management: Concepts, Languages, Architectures**. New York: Springer Science & Business Media, 2012.

YAWL FOUNDATION, T. **YAWL: Yet Another Workflow Language**. 2014. Disponível em: <http://yawlfoundation.org/>. Acesso em: 22 abr. 2015.

YU, Eric. Modelling strategic relationships for process reengineering. **Social Modeling for Requirements Engineering**, v. 11, p. 2011, 2011.