# 5
# Implementação

A implementação do COWS foi realizada em Flora-2, uma linguagem orientada a objeto para representação de conhecimento, além de ser, também, uma plataforma para desenvolvimento de aplicações (Flora-2, 2007). Flora-2 é implementada como um conjunto de bibliotecas de execução e um compilador, que traduz o código do programa — um dialeto estendido da linguagem de programação F-logic (Kifer et al., 1995) — em código Prolog, processado no sistema dedutivo XSB.

A arquitetura da implementação do COWS foi realizada de acordo com o ambiente operacional apresentado na seção 4.3. O banco de dados foi implementado como um conjunto de arquivos com informações sobre tipos de contratos, foros e anúncios de agentes. Essas informações são armazenadas segundo as ontologias definidas no modelo do COWS.

Neste capítulo, são apresentadas as ontologias do modelo do COWS (seção 5.1). Em seguida, é apresentado o ambiente de implementação do COWS, contendo a descrição da principal regra utilizada para a busca de soluções (seção 5.2). Finalmente, as seções 5.3 e 5.4 apresentam exemplos da aplicação de políticas, de protocolos de confiança e de encadeamento de contratos.



## 5.1. Ontologias

Nesta seção, são apresentadas as especificações das ontologias das classes do modelo do COWS, descritas no capítulo anterior. As ontologias foram especificadas em Flora-2 e suas assinaturas são apresentadas nas sub-seções seguintes, além de exemplos de instâncias.

### 5.1.1. Contrato

As informações de tipos de contrato, para efeito de testes da implementação, foram armazenadas em um arquivo Flora-2 — cows_contrato.flr —, contendo os

possíveis tipos de contratos utilizados pelos agentes que publicam seus anúncios e procuram por parceiros de negócios.

A Figura 23 apresenta as assinaturas da ontologia de contrato em Flora-2. A Figura 24 apresenta um exemplo da definição de um tipo de contrato de compra e venda envolvendo três parceiros, que desempenham os papéis de comprador, vendedor e entregador.

```
// contract
contract[version=>string
         ,contractURI=>xsd_anyURI
         ,name=> string
         ,description=> string
         ,hasRole=>>role
         ,hasChoreography=>>choreography].

// role
role[roleName=>string
     ,hasContractBind=>>contractBinding].

// contractBinding
contractBinding[contractURI=>xsd_anyURI
                ,roleName=>string].

// choreography
choreography [definition=>string].
```

Figura 23 - Ontologia de contrato do COWS em Flora-2

```
_#:contract
[version->'1.0'
,contractURI->3PARCEIROS_URI
,name->'Compra e venda entre três parceiros'
,hasRole->>
 {_#[roleName->'COMPRADOR']
  ,_#[roleName->'VENDEDOR']
  ,_#[roleName->'ENTREGADOR']}
].
```

Figura 24 - Exemplo de instância de tipo de contrato

### 5.1.2. Publicação

As informações de publicação foram armazenadas em um arquivo Flora-2 — cows_publish.flr —, contendo todos os anúncios publicados pelos agentes, que procuram por parceiros de negócios.

A Figura 25 apresenta as assinaturas da ontologia de publicação de anúncios em Flora-2. A Figura 26 apresenta três anúncios publicados por agentes que desejam participar de um negócio segundo o contrato de compra e venda entre três parceiros, apresentado na Figura 24.

```
// agentPublish
agentPublish[hasAgentInfo=>agentInfo
             ,hasContractInfo=>>contractInfo
             ,hasPolicy=>>policy].

// agentInfo
agentInfo[agentURI=>xsd_anyURI
          ,name=>string
          ,country=>string].

// contractInfo
contractInfo[contractURI=>xsd_anyURI
             ,forumURI=>>xsd_anyURI
             ,hasBindingInfo=>bindingInfo
             ,hasPolicy=>>policy].

// bindingInfo
bindingInfo[hasRoleBinding=>>roleBinding].

// roleBinding
roleBinding[contractRoleName=>string
            ,hasRelationshipBinding=>>relationshipBinding
            ,hasPolicy=>>policy
            ,hasFunctionalDescription=>functionalDescription].

// relationshipBinding
relationshipBinding[contractRoleName=>string
                    ,hasTrust=>>trust].

// functionalDescription
functionalDescription [metaModelType=>xsd_anyURI ].

// trust
trust[bindId=>integer
      ,credentialURI=>xsd_anyURI
      ,certificationURI=>>xsd_anyURI].

// policy
policy[fact=>string
       ,rule=>string].
```

Figura 25 - Ontologia de publicação do COWS em Flora-2

```
_#:agentPublish
[hasAgentInfo->
 _#[agentURI->CARLOS_URI,name->'Carlos',country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI,forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->'COMPRADOR'
             ,hasFunctionalDescription->_#[metaModelType->UNSPSC]
             ,hasRelationshipBinding->>
              {_#[contractRoleName->'VENDEDOR']
               ,_#[contractRoleName->'ENTREGADOR']}]}]]}
].

_#:agentPublish
[hasAgentInfo->
 _#[agentURI->AMERICANAS_URI,name->'Lojas Americanas',country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI,forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->'VENDEDOR'
             ,hasFunctionalDescription->_#[metaModelType->UNSPSC]
             ,hasRelationshipBinding->>
              {_#[contractRoleName->'COMPRADOR']
               ,_#[contractRoleName->'ENTREGADOR']}]}]]}
].

_#:agentPublish
[hasAgentInfo->
 _#[agentURI->SEDEX_URI, name->'Sedex', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI, forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->'ENTREGADOR'
             ,hasFunctionalDescription->_#[metaModelType->UNSPSC]
             ,hasRelationshipBinding->>
              {_#[contractRoleName->'COMPRADOR']
               ,_#[contractRoleName->'VENDEDOR']}]}]]}
].
```

Figura 26 - Exemplo de instância de anúncio de agentes

### 5.1.3.Busca

Um agente que queira fazer uma busca por parceiros de negócios no ambiente COWS, precisa especificar informações que identifiquem o que esse agente deseja. A informação de busca é especificada segundo a ontologia apresentada na Figura 27. Para efeito de testes, as informações para busca de agentes foram armazenadas em um arquivo — cows_search.flr —, contendo

exemplos de pedidos de busca realizados por agentes que procuram por parceiros de negócios.

```
// agentSearch
agentSearch[hasAgentInfo=>agentSearchInfo
            ,hasContractInfo=>>contractSearchInfo
            ,hasPolicy=>>policy].

// agentSearchInfo
agentSearchInfo[agentURI=>xsd_anyURI].

// contractSearchInfo
contractSearchInfo[contractURI=>xsd_anyURI
                   ,hasBindingInfo=>bindingInfoSearch
                   ,hasPolicy=>>policy].

// bindingInfoSearch
bindingInfoSearch[hasRoleBinding=>>roleBindingSearch].

// roleBindingSearch
roleBindingSearch[contractRoleName=>xsd_anyURI
                  ,hasPolicy=>>policy].

// policy
policy[fact=>string
       ,rule=>string].
```

Figura 27 - Ontologia de busca do COWS em Flora-2

A Figura 28 apresenta uma instância da ontologia de busca, onde o agente Carlos, que desempenha o papel de comprador em um contrato de compra e venda entre três parceiros (Figura 26), procura por parceiros que desempenhem os outros papéis daquele tipo de contrato. No caso, a busca retornaria o agente Carlos no papel de comprador, o agente Lojas Americanas no papel de vendedor, e o agente Sedex no papel de entregador (Tabela 2).

```
busca1_CARLOS:agentSearch
[hasAgentInfo->_#[agentURI->CARLOS_URI]
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI
     ,hasBindingInfo->
      _#[hasRoleBinding->>_#[contractRoleName->'COMPRADOR']]]}
].
```

Figura 28 - Exemplo de instância da ontologia de busca

| Agente de Busca: Carlos<br>Contrato: Compra e venda entre três parceiros<br>Papel: COMPRADOR | | |
|---|---|---|
| Solução 1 | Nível: 1<br>Agente: Carlos<br>Contrato: Compra e venda entre três parceiros<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Carlos |
| | VENDEDOR | Lojas Americanas |
| | ENTREGADOR | Sedex |

Tabela 2 - Resultado de busca

### 5.1.4. Resultado de Busca

O resultado de uma busca, no ambiente de execução do COWS, é retornado segundo a ontologia apresentada na Figura 29. Esse resultado é composto por um conjunto de soluções, cada uma delas composta por um conjunto de parceiros agrupados em um contrato e um foro. Eventuais restrições, fruto de condições não-atendidas, especificadas em políticas, e que impossibilitariam a aproximação dos parceiros, são também devolvidas no conjunto de soluções encontrada pelo ambiente.

```
// solutionSet
solution[searchAgentURI=>xsd_anyURI
         ,hasSolution=>>solution].

// solution
solution[contractURI=>xsd_anyURI
         ,forumURI=>xsd_anyURI
         ,hasPartner=>>partner
         ,hasRestriction=>>restrictionInfo].

// partner
partner[agentURI=>xsd_anyURI
        ,roleName=>string
        ,hasRestriction=>>restrictionInfo].

// restrictionInfo
restrictionInfo[name=>string,value=>string].
```

Figura 29 - Ontologia de resultado de busca do COWS em Flora-2

### 5.1.5. Foro

As informações de foros, para efeito de testes da implementação, foram armazenadas em um arquivo Flora-2 — cows_forum.flr —, contendo todos os possíveis foros utilizados pelos agentes que publicam seus anúncios e procuram por parceiros de negócios. A Figura 30 apresenta as assinaturas da ontologia de foro em Flora-2. Exemplos de instâncias de foros serão apresentados nas seções seguintes.

```
// forum
forum[forumURI=>xsd_anyURI
      ,name=>string
      ,hasPolicy=>>policy].

// policy
policy[fact=>string
      ,rule=>string].
```

Figura 30 - Ontologia de foto do COWS em Flora-2

## 5.2. Ambiente de Testes

O ambiente de testes utilizado foi o da *shell* que acompanha a distribuição do Flora-2 (Figura 31). Essa *shell* estabelece uma interface interativa, onde é possível emitir uma série de comandos que permitem, entre outras características, a carga de módulos contendo regras programadas em Flora-2.

Assim que o ambiente Flora-2 é iniciado, a ativação do ambiente operacional do COWS é realizada pela digitação do comando que carrega o módulo de programa do COWS — "[cows]"—, com os fatos e as regras definidos no arquivo cows.flr (Figura 32). Esse módulo inclui, também, os arquivos utilizados como banco de dados — contract.flr, publish.flr, forum.flr.

A partir de então, as buscas emitidas na linha de comando farão com que o COWS procure soluções que atendam às especificações de busca do agente.

```
C:\flora\flora2\cowsTESE>runflora
[xsb_configuration loaded]
[sysinitrc loaded]

XSB Version 2.7.1 (Kinryo) of March 5, 2005
[x86-pc-windows; mode: optimal; engine: slg-wam; gc: indirection; scheduling: local]

Evaluating command line goal:
| ?-  asserta(library_directory('C:\flora\flora2')). [flora2]. flora_shell.
| ?-
yes
| ?- [flora2 loaded]
[floraInstallMode loaded]

yes
| ?- [flrversion loaded]
[p2h_config loaded]
[prolog2hilog loaded]
[flrshell loaded]
[flrundefhook loaded]
[flrutils loaded]
[flranswer loaded]
[flrload loaded]
[flrdisplay loaded]
[flrimportedcalls loaded]
[trailer$eq=none+inh=flogic+cus=none_main loaded]
[patch_main loaded]

FLORA-2 Version 0.94 (Narumigata) of May 2005

Type `flHelp.' to display help.
Type `flDemo(demoName).' to run a demo.

flora2 ?-
```

Figura 31 - Ativação da *shell* do Flora-2

```
flora2 ?- [cows].

[FLORA: Loading C:\flora\flora2\cowsTESE\cows.flr into module main]
[cows_main loaded]
[flrclause loaded]
[flraggcolbag loaded]
[flraggcolset loaded]
[flrdbop loaded]
[flrdynrule loaded]
[flrcontrol loaded]
[flraggmax loaded]
[flrnewoid loaded]
[flrtables loaded]
[flrtruthvalue loaded]
[flrmetaops loaded]
[FLORA: Dynamically loading C:\flora\flora2\cowsTESE\cows.fdb into module main]
[Preprocessing C:\flora\flora2\cowsTESE\cows.fdb]
[FLORA: Done! CPU time used: 0.0620 seconds]
[FLORA: Dynamically loading C:\flora\flora2\cowsTESE\cows.fld into module main]
[Preprocessing C:\flora\flora2\cowsTESE\cows.fld]
[FLORA: Done! CPU time used: 0.0000 seconds]

Yes

flora2 ?-
```

Figura 32 - Carga do módulo com o COWS

A busca, que dispara a procura pelo casamento de soluções, é realizada a partir da seguinte regra:

#cows_searchSet(Ok, Search, MaxLevel, MaxLevelRobot, SolutionSet)

- Ok – variável que indica a opção de apresentação dos resultados com ou sem restrição.
- Search – define um objeto do tipo *agentSearch*, com as informações da busca.
- MaxLevel – indica o número máximo de encadeamentos de contratos, caso haja soluções diretas da busca (seção 5.5).
- MaxLevelRobot – indica o número máximo de encadeamentos automáticos de contratos, caso não haja soluções diretas da busca (seção 5.5.1).
- SolutionSet – conjunto de soluções da busca.

Uma vez que uma busca tenha sido solicitada, o programa utiliza a regra apresentada na Figura 33 como base para a montagem das soluções.

```
#cows_searchSolutionSet(1, Search, AgentURI, ContractURI, RoleName, SolutionId) :-
        #cows_partner(AgentURI, ContractURI, Partners),
        #cows_partner_ForumList(AgentURI, Partners),
        #cows_partner_RoleName(AgentURI, RoleName, Partners),
        #cows_solution(AgentURI, SolutionId, Partners),
        #cows_policy(Search, SolutionId),
        #cows_trust(Search, SolutionId),
        #cows_functionalMatch(Search,SolutionId),
        #cows_solutionOK(SolutionId).
```

Figura 33 - Regra de definição de soluções de uma busca

Essa regra ativa diversas outras regras do programa, que filtram, sucessivamente, o conjunto de combinações possíveis de agentes que têm seus anúncios publicados no banco de dados:

- #cows_partner(AgentURI, ContractURI, Partners) – vasculha todos os anúncios publicados e monta as possíveis combinações de parceiros (Partners), para um determinado contrato (ContractURI), que contenham o agente que requisitou a busca (AgentURI).
- #cows_partner_ForumList(AgentURI, Partners) – filtra, do conjunto de parceiros anterior (Partners), apenas o conjunto de

parceiros que atende ao mesmo foro do agente que requisitou a busca (AgentURI).

- #cows_partner_RoleName(AgentURI, RoleName, Partners) – filtra, do conjunto de parceiros anterior (Partners), apenas o conjunto que tenha o agente (AgentURI) desempenhando o papel definido na busca (RoleName).
- #cows_solution(AgentURI, SolutionId, Partners) – gera os objetos do tipo *solution* e grava na base de dados do programa.
- #cows_policy(Search, SolutionId) – aplica as políticas definidas pelos agentes que compõe cada uma das soluções.
- #cows_trust(Search, SolutionId) – aplica as especificações de confiança dos agentes que compõe cada uma das soluções.
- #cows_functionalMatch (Search, SolutionId) – aplica o *matchmaker* associado às descrições funcionais dos agentes.
- #cows_solutionOK(SolutionId) – filtra as soluções que não apresentam restrições.

Para efeito dos testes, as soluções resultantes de uma busca são apresentadas na tela do computador, de modo a se ter uma melhor legibilidade dos resultados. Por exemplo, uma busca onde o agente Carlos, que desempenha o papel de comprador em um contrato de compra e venda entre três parceiros, procura por parceiros que desempenhem os outros papéis daquele tipo de contrato, resultaria na exibição da tela apresentada na Figura 34.



## 5.3.Política

As políticas foram implementadas como um conjunto de fatos e regras especificados em Flora-2. De forma a poder instanciar as regras e os fatos de uma política, esses dados devem ser definidos de forma reificada. Assim sendo, quando da necessidade de aplicação de uma determinada política, os fatos e as regras podem ser lidos pelo programa e anexados ao código do programa, por meio das primitivas *insert* e *insertrule*, respectivamente. A Figura 35 apresenta uma política associada a um foro, que especifica que somente parceiros que pertençam a um conjunto determinado de países podem participar do processo de negócios.

```
flora2 ?- #cows_searchSet(1,busca1_CARLOS,4,2,SolutionSet).


**************************************************************************************
**************************************************************************************
Agente da Busca: Carlos

==========================================================================
 Nivel 1
 Agente   : Carlos
 Contrato : Compra e venda entre 3 parceiros
 Foro     : Parceiros de mesmo pais

 Parceiros
  Agente  : Carlos
  Papel   : COMPRADOR

  Agente  : Lojas Americanas
  Papel   : VENDEDOR

  Agente  : Sedex
  Papel   : ENTREGADOR

==========================================================================

**************************************************************************************
**************************************************************************************



SolutionSet = solset_$_$_flora'dyn_newoid12591

1 solution(s) in 0.0310 seconds

Yes

flora2 ?-
```

Figura 34 - Exemplo de tela de resultado de busca

```
_#:forum
[forumURI->FORUM2_URI
,name->'Parceiros somente de USA, Brazil, England, France'
,hasPolicy->>
{_#[fact->
       ${ (#policy(FORUM2_URI,country,'USA')),
          (#policy(FORUM2_URI,country,'Brazil')),
          (#policy(FORUM2_URI,country,'England')),
          (#policy(FORUM2_URI,country,'France'))}
    ,rule->
      ${ (#policy(FORUM2_URI, SolutionId) :-
            SolutionId[hasPartner->->Partners],
            #policy(FORUM2_URI,SolutionId,Partners)),
          (#policy(FORUM2_URI,_,[]) :- true),
          (#policy(FORUM2_URI,SolutionId,[PartnerId|T]) :-
             PartnerId[agentURI->AgentURI],
             (_:publish).hasAgent[agentURI->AgentURI,country->Country],
             if (#policy(FORUM2_URI,country,Country))
             then (#policy(FORUM2_URI,SolutionId,T))
             else (#cows_policy_solutionForumRestriction(SolutionId,'FORUM_LEVEL')))}
   ]}
].
```

Figura 35 - Exemplo de política de foro

Existem quatro tipos diferentes de níveis onde é possível especificar políticas: foro, agente, contrato e papel. O primeiro nível se aplica a todos os agentes, de forma global, que aceitam políticas definidas para um determinado foro. Os três níveis restantes são relativos aos agentes, de forma individual.

Um agente pode especificar políticas que serão aplicadas em todos os contratos especificados em uma busca (nível agente), políticas que serão aplicadas apenas em relação a um contrato especificado em uma busca (nível contrato) e/ou políticas que serão aplicadas apenas em relação ao papel desempenhado pelo agente em um contrato especificado em uma busca (nível papel).

A aplicação da política no nível de foro é realizada pela chamada da seguinte regra:

#policy (ForumURI, SolutionId)

- ForumURI – identifica o foro.
- SolutionId – identifica a solução — objeto do tipo *solution* — à qual a política deve ser aplicada.

A aplicação da política no nível de agente, de contrato ou de papel é realizada pela chamada da seguinte regra:

#policy(AgentURI, SolutionId, MyPartnerId)

- AgentURI – identifica o agente que especificou a política.
- SolutionId – identifica a solução — objeto do tipo *solution* — à qual a política deve ser aplicada.
- MyPartnerId – identifica o parceiro — objeto do tipo *partner* — relativo ao agente que especificou a política.

Caso seja encontrada alguma restrição, existe uma chamada do COWS que permite à regra, especificada na política, definir um texto a ser armazenado na respectiva solução do resultado de busca. Uma solução é considerada não válida quando apresenta alguma restrição imposta pelas diversas políticas especificadas no ambiente.

A indicação de restrição em uma política no nível de foro é realizada pela chamada da seguinte regra:

#cows_policy_solutionForumRestriction(SolutionId, Restriction)

- SolutionId – identifica a solução — objeto do tipo *solution* — à qual a restrição se aplica.

- Restriction – especifica um texto que informa a restrição.

A indicação de restrição em uma política no nível de agente, de contrato ou de papel é realizada pela chamada da seguinte regra:

#cows_policy_agentRestriction(MyPartnerId, PartnerId, Restriction)

- MyPartnerId – identifica o parceiro — objeto do tipo *partner* — relativo ao agente que aplicou a política.
- PartnerId – identifica o parceiro — objeto do tipo *partner* — relativo ao agente ao qual se aplica a restrição.
- Restriction – especifica um texto que informa a restrição.

### 5.3.1. Exemplo

Considerando o exemplo de publicação dos agentes Carlos, Lojas Americanas e Sedex (Figura 26), vamos supor que seja acrescentado mais um anúncio de publicação: o agente FedEx desempenhando o papel de entregador no contrato de compra e venda entre três parceiros (Figura 36).

```
_#:agentPublish
[hasAgentInfo->
 _#[agentURI->FEDEX_URI, name->'FedEx', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI, forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->' ENTREGADOR '
             ,hasRelationshipBinding->>
              {_#[contractRoleName->'COMPRADOR']
               ,_#[contractRoleName->'VENDEDOR']}]}]]}
].
```

Figura 36 - Anúncio de publicação do agente FedEx

Se o agente Carlos realizasse novamente a busca, ele obteria as duas soluções, apresentadas na Tabela 3, que têm o agente Lojas Americanas como vendedor e os agentes Sedex e FedEx, como entregadores.

Porém, o agente Carlos deseja realizar compras, apenas quando as entregas são realizadas por determinados entregadores, que são de seu conhecimento e confiança. O agente Carlos poderia, então, republicar seu anúncio, com uma

restrição quanto aos possíveis entregadores de suas compras, especificada em uma política de agente (Figura 37).

| | | |
|---|---|---|
| Agente de Busca: Carlos<br>Contrato: Compra e venda entre três parceiros<br>Papel: COMPRADOR | | |
| Solução 1 | Nível: 1<br>Agente: Carlos<br>Contrato: Compra e venda entre três parceiros<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Carlos |
| | VENDEDOR | Lojas Americanas |
| | ENTREGADOR | Sedex |
| Solução 2 | Nível: 1<br>Agente:Carlos<br>Contrato: Compra e venda entre três parceiros<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Carlos |
| | VENDEDOR | Lojas Americanas |
| | ENTREGADOR | FedEx |

Tabela 3 - Resultado de busca do agente Carlos

```
_#:agentPublish
[hasAgentInfo->
 _#[agentURI->CARLOS_URI,name->'Carlos',country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI,forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->'COMPRADOR'
             ,hasRelationshipBinding->>
              {_#[contractRoleName->'VENDEDOR']
               ,_#[contractRoleName->'ENTREGADOR']}]}]]}
,hasPolicy->>
 {_#[rule->
  ${
    (#policy(CARLOS_URI, SolutionId, MyPartnerId) :-
       SolutionId[hasPartner->->Partners],
       #policy(CARLOS_URI,SolutionId,MyPartnerId,Partners)),
    (#policy(CARLOS_URI,_,_,[]) :- true),
    (#policy(CARLOS_URI, SolutionId, MyPartnerId, [PartnerId|T]) :-
      (PartnerId:partner)[roleName->RoleName,agentURI->AgentURI],
      if (RoleName = 'ENTREGADOR')
      then (
          if (AgentURI = SEDEX_URI)
          then (#policy(CARLOS_URI,SolutionId,MyPartnerId,T))
          else (#cows_policy_agentRestriction(MyPartnerId, PartnerId, 'AGENT_LEVEL'))
          )
      else (#policy(CARLOS_URI,SolutionId,MyPartnerId,T))
    )}]}
].
```

Figura 37 - Política de restrição de entregadores do agente Carlos

Se o agente Carlos realizasse novamente a busca ele obteria apenas a solução 1 da Tabela 3, com o agente Lojas Americanas, como vendedor, e o agente Sedex, como entregador.

### 5.3.2. Sobreposição de Políticas na Busca

Quando um agente emite um pedido de busca de parceiros, ele pode definir informações relacionadas às políticas, que irão sobrepor eventuais políticas que esse agente tenha definido em seu anúncio armazenado no banco de dados. Sendo assim, é possível, a um agente, alterar as políticas no momento da busca, de forma a conseguir soluções que, eventualmente, seriam descartadas por políticas definidas no anúncio armazenado no banco de dados. Um agente pode sobrepor políticas nos níveis de agente, de contrato e de papel.

Se, no exemplo anterior, o agente Carlos quisesse sobrepor a política de seu anúncio, que passou a restringir os entregadores de sua compra, estabelecendo, no momento da busca, uma nova política onde ele aceita qualquer agente no papel de entregador (Figura 38), o resultado da busca voltaria a contemplar as duas soluções da Tabela 3, com os agentes Sedex e FedEx como entregadores.

```
busca2_CARLOS:agentSearch
[hasAgentInfo->_#[agentURI->CARLOS_URI]
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI
     ,hasBindingInfo->
      _#[hasRoleBinding->>_#[contractRoleName->'COMPRADOR']]]}
,hasPolicy->>{_#[rule-> ${(#policy(CARLOS_URI,_,_) :- true)}]}
].
```

Figura 38 - Exemplo 1 de sobreposição de políticas

O agente Carlos poderia, em outra busca, especificar a preferência pelo agente FedEx (Figura 39), no papel de entregador, o que retornaria apenas o resultado 2 da Tabela 3.

## 5.4.Confiança

A desconfiança entre agentes poderia ser minimizada mediante a apresentação de credenciais digitais emitidas por órgãos de certificação. Um

determinado processo de negócios poderia exigir uma troca de credenciais, numa ordem predefinida, de forma a estabelecer a confiança entre parceiros.

```
busca3_CARLOS:agentSearch
[hasAgentInfo->_#[agentURI->CARLOS_URI]
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI
     ,hasBindingInfo->
      _#[hasRoleBinding->>_#[contractRoleName->'COMPRADOR']]]}
,hasPolicy->>
 {_#[rule->
  ${
    (#policy(CARLOS_URI, SolutionId, MyPartnerId) :-
       SolutionId[hasPartner->->Partners],
       #policy(CARLOS_URI,SolutionId,MyPartnerId,Partners)),
    (#policy(CARLOS_URI,_,_,[]) :- true),
    (#policy(CARLOS_URI, SolutionId, MyPartnerId, [PartnerId|T]) :-
      (PartnerId:partner)[roleName->RoleName,agentURI->AgentURI],
      if (RoleName = 'ENTREGADOR')
      then (
          if (AgentURI = FEDEX_URI)
          then (#policy(AGENT1_URI,SolutionId,MyPartnerId,T))
          else (#cows_policy_agentRestriction(MyPartnerId, PartnerId, 'AGENT_LEVEL'))
          )
      else (#policy(CARLOS_URI, SolutionId,MyPartnerId, T))
    )}]}
].
```

Figura 39 - Exemplo 2 de sobreposição de políticas

No ambiente COWS, esse protocolo de troca de credenciais é definido por cada um dos parceiros em seus relacionamentos com os outros parceiros. Se um dos parceiros publica um anúncio onde estabelece que existe um protocolo de confiança, uma solução, resultante de uma busca, será considerada não-válida se o outro parceiro selecionado não atender ao protocolo.

Como exemplo, vamos supor um novo tipo de contrato: contrato de compra e venda entre dois parceiros (atacado) — comprador e vendedor — que seria utilizado para a venda a atacado de produtos. Vamos supor, também, a publicação de dois novos anúncios de agentes, Loja das Fábricas e Fábrica Brastemp, que se anunciam, respectivamente, como comprador e vendedor nesse novo tipo de contrato. As Figuras 40 e 41 mostram, respectivamente, a especificação do contrato e os anúncios de publicação dos agentes.

Se o agente Fabrica Brastemp fizer uma busca por parceiros, em um contrato de compra e venda por atacado (Figura 42), o resultado será o listado na Tabela 4.

```
_#:contract
[version->'1.0'
,contractURI->2PARCEIROS_ATACADO_URI
,name->'Compra e venda entre dois parceiros (atacado)'
,hasRole->>
 {_#[roleName->'COMPRADOR']
  ,_#[roleName->'VENDEDOR']}
].
```

Figura 40 - Contrato de compra e venda por atacado entre dois parceiros

```
_#:agentPublish
[hasAgentInfo->
_#[agentURI->LOJA_URI , name->'Loja das Fabricas', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR']}]}]]}
].

_#:agentPublish
[hasAgentInfo->
_#[agentURI->FABRICA_URI, name->'Fabrica Brastemp', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->' VENDEDOR '
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'COMPRADOR']}]}]]}
].
```

Figura 41 - Anúncio dos agentes Loja das Fábricas e Fábrica Brastemp

```
busca1_FABRICA:agentSearch
[hasAgentInfo->_#[agentURI->FABRICA_URI]
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,hasBindingInfo->
    _#[hasRoleBinding->>_#[contractRoleName->'VENDEDOR']]]}
].
```

Figura 42 - Solicitação de busca do agente Fábrica Brastemp

| Agente de Busca: Fábrica Brastemp<br>Contrato: Compra e venda entre dois parceiros (atacado)<br>Papel: VENDEDOR | | |
|---|---|---|
| Solução 1 | Nível: 1<br>Agente: Loja das Fábricas<br>Contrato: Compra e venda entre dois parceiros (atacado)<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Loja das Fábricas |
| | VENDEDOR | Fábrica Brastemp |

Tabela 4 - Resultado de busca do agente Fábrica Brastemp

Se o agente Carlos republicar seu anúncio, se candidatando também a comprador no contrato acima (Figura 43), a busca do agente Fabrica Brastemp retornaria mais uma solução (Tabela 5).

```
_#:agentPublish
[hasAgentInfo->
 _#[agentURI->CARLOS_URI, name->'Carlos', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR']
         ,_#[contractRoleName->'ENTREGADOR']}]}]]
 ,_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR']}]}]]}
].
```

Figura 43 - Anúncio do agente Carlos com o contrato de compra e venda por atacado

| Agente de Busca: Fábrica Brastemp<br>Contrato: Compra e venda entre dois parceiros (atacado)<br>Papel: VENDEDOR | | |
|---|---|---|
| Solução 1 | Nível: 1<br>Agente: Fábrica Brastemp<br>Contrato: Compra e venda entre dois parceiros (atacado)<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Loja das Fábricas |
| | VENDEDOR | Fábrica Brastemp |
| Solução 2 | Nível: 1<br>Agente: Fábrica Brastemp<br>Contrato: Compra e venda entre dois parceiros (atacado)<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Carlos |
| | VENDEDOR | Fabrica Brastemp |

Tabela 5 - Resultado de busca do agente Fábrica Brastemp (2)

Porém, o agente Fábrica Brastemp só deseja vender para comerciantes no atacado e não diretamente no varejo. Para tal, durante o processo de negócios, esse agente exige que o agente que desempenha o papel de comprador apresente uma credencial de comerciante de atacado. Essa exigência pode ser definida como uma restrição e testada pelo ambiente COWS, durante o processo de aproximação dos parceiros. Aqueles que não atenderem a esse protocolo de confiança serão descartados das soluções. A Figura 44 apresenta os anúncios dos agentes Loja das Fábricas e Fábrica Brastemp publicados com a restrição de confiança.

```
_#:agentPublish
[hasAgentInfo->
_#[agentURI->LOJA_URI , name->'Loja das Fabricas', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR'
             ,hasTrust->>
              {_#[trustURI->TRUST1_URI
                  ,hasBind->>{_#[bindId->1
                                 ,credentialURI->CREDENTIAL1_URI
                                 ,certificationURI->>DIGICERT2_URI]}]}]}]}]}]}
].

_#:agentPublish
[hasAgentInfo->
_#[agentURI->FABRICA_URI, name->'Fabrica Brastemp', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->' VENDEDOR '
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'COMPRADOR'
             ,hasTrust->>
              {_#[trustURI->TRUST1_URI
                  ,hasBind->>{_#[bindId->1
                                 ,credentialURI->CREDENTIAL1_URI
                                 ,certificationURI->>DIGICERT2_URI]}]}]}]}]}]}
].
```

Figura 44 - Anúncios de agentes com restrição de confiança

Se o agente Fábrica Brastemp refizer a busca, o resultado será, novamente, o listado na Tabela 4, pois o agente Carlos não atende à restrição de confiança.

## 5.5. Cadeia de Contrato

Uma cadeia de contratos é obtida quando, para uma determinada solução, não existem contratos diretos que possam satisfazer os requisitos do agente que solicitou a busca. Para que exista uma cadeia de contratos é necessário que um agente específico participe como intermediário, uma espécie de ponte, entre os contratos que estão sendo encadeados.

Se considerarmos o exemplo anterior, o agente Carlos não tem como realizar um negócio diretamente com o agente Fábrica Brastemp, pois esse agente requisita uma credencial que o primeiro não possui.

Vamos supor, então, um terceiro contrato de compra e venda — compra e venda entre dois parceiros (varejo) —, que permite o relacionamento entre dois parceiros desempenhando os papéis de comprador e vendedor (Figura 45). Vamos supor, também, que esse contrato tenha um encadeamento com o contrato de compra e venda entre dois parceiros (atacado), ou seja, um agente que desempenha o papel de vendedor no contrato de varejo e desempenha o papel de comprador no contrato de atacado.

```
_#:contract
[version->'1.0'
,contractURI->2PARCEIROS_VAREJO_URI
,name->'Compra e venda entre dois parceiros (varejo)'
,hasRole->>
 {_#[roleName->'COMPRADOR']
  ,_#[roleName->'VENDEDOR'
      ,hasContractBind->>
       _#[contractURI->2PARCEIROS_ATACADO_URI
          ,roleName->'COMPRADOR']]}
].
```

Figura 45 - Contrato de compra e venda no varejo entre dois parceiros

O agente Loja das Fábricas poderia republicar seu anúncio para poder atuar como intermediário entre o varejo e o atacado. Para tal, ele precisaria fazer uma vinculação do papel de vendedor no contrato de varejo com o papel de comprador no contrato de atacado (Figura 46).

```
_#:agentPublish
[hasAgentInfo->
_#[agentURI->LOJA_URI, name->'Loja das Fabricas', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR'
             ,hasTrust->>
              {_#[trustURI->TRUST1_URI
                  ,hasBind->>{_#[bindId->1
                                 ,credentialURI->CREDENTIAL1_URI
                                 ,certificationURI->>DIGICERT2_URI]}]}]}]}]]
 ,_#[contractURI->2PARCEIROS_VAREJO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'VENDEDOR'
          ,hasContractBind->>
           _#[contractURI->2PARCEIROS_ATACADO_URI
              ,roleName->'COMPRADOR']
         ,hasRelationshipBinding->>
          {_#[contractRoleName->'COMPRADOR']}]}]]}
].
```

Figura 46 - Anúncio do agente Loja das Fábricas

```
_#:agentPublish
[hasAgentInfo->
 _#[agentURI->CARLOS_URI, name->'Carlos', country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->3PARCEIROS_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR']
          ,_#[contractRoleName->'ENTREGADOR']}]}]]
 ,_#[contractURI->2PARCEIROS_ATACADO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR']}]}]]
 ,_#[contractURI->2PARCEIROS_VAREJO_URI
   ,forumURI->>{FORUM1_URI}
   ,hasBindingInfo->
    _#[hasRoleBinding->>
      {_#[contractRoleName->'COMPRADOR'
        ,hasRelationshipBinding->>
         {_#[contractRoleName->'VENDEDOR']}]}]]}
].
```

Figura 47 - Anúncio do agente Carlos

Se o agente Carlos republicar seu anúncio (Figura 47) para aceitar o novo tipo de contrato (Figura 45) e requisitar uma busca de parceiros para o contrato de varejo (Figura 48), ele obterá a solução listada na Tabela 6, que tem o contrato de varejo encadeado com o contrato de atacado. O agente Loja das Fábricas atua como intermediário — comprador no contrato de atacado e vendedor no contrato de varejo — entre os agentes Carlos — comprador no varejo — e o agente Fábrica Brastemp — vendedor no atacado.

```
busca4_CARLOS:agentSearch
[hasAgentInfo->_#[agentURI->CARLOS_URI]
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_VAREJO_URI
   ,hasBindingInfo->
    _#[hasRoleBinding->>_#[contractRoleName->'COMPRADOR']]]}
].
```

Figura 48 - Solicitação de busca do agente Carlos



| Agente de Busca: Carlos<br>Contrato: Compra e venda entre dois parceiros (varejo)<br>Papel: COMPRADOR | | |
|---|---|---|
| Solução 1 | Nível: 1<br>Agente: Carlos<br>Contrato: Compra e venda entre dois parceiros (varejo)<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Carlos |
| | VENDEDOR | Loja das Fábricas |
| | Nível:2<br>Agente: Loja das Fabricas<br>Contrato: Compra e venda entre dois parceiros (atacado)<br>Foro: Parceiros de mesmo país | |
| | COMPRADOR | Loja das Fábricas |
| | VENDEDOR | Fábrica Brastemp |

Tabela 6 - Resultado de busca do agente Carlos (2)

### 5.5.1. Cadeia de Contrato Automática

O ambiente do COWS constrói, automaticamente, soluções de encadeamento de contratos de forma reversa, no caso de buscas sem sucesso. Caso um agente solicite uma busca que não retorne uma solução válida, o ambiente procura por contratos que possam ser encadeados de forma a que se obtenha uma solução válida.

No exemplo anterior, se o agente Carlos solicitasse uma busca em um contrato por atacado (Figura 49), ele deveria receber um conjunto vazio de

soluções, pois ele não atende ao protocolo de confiança do agente Fábrica Brastemp.

```
busca5_CARLOS:agentSearch
[hasAgentInfo->_#[agentURI->CARLOS_URI]
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_ATACADO_URI
   ,hasBindingInfo->
    _#[hasRoleBinding->>_#[contractRoleName->'COMPRADOR']]]}
].
```

Figura 49 - Ontologia de contrato do COWS em Flora-2

Porém, o ambiente COWS vasculha os contratos publicados e consegue vincular os contratos de atacado e varejo, iniciando automaticamente uma nova busca a partir do contrato encadeado, de forma reversa. Dessa forma, o resultado da busca seria o mesmo apresentado na Tabela 6.

## 5.6. Casamento Funcional

Uma vez que o *matchmaker* do COWS tenha selecionado um conjunto de soluções envolvendo parceiros em um processo de negócios, esse conjunto é submetido ao *matchmaker* associado à parte funcional do processo, responsável pelo casamento dos parceiros a partir das descrições funcionais, que podem estar especificadas de acordo com os diversos meta-modelos existentes.

Não faz parte do escopo deste trabalho o desenvolvimento de *matchmakers* funcionais. A partir do conhecimento do meta-modelo, que descreve um determinado serviço de um agente, o programa chama um regra que está associada a esse meta-modelo.

Para efeito de testes, foi desenvolvido um meta-modelo baseado na categorização de produtos e serviços do UNSPSC, uma das taxonomias utilizadas para a publicação e casamento de serviços utilizando UDDI.

Rusbiz.com (Rusbiz, 2007) é um *site* que cria um ambiente de mercado virtual, onde pessoas e organizações podem cadastrar produtos e serviços, indexados pela taxonomia UNSPSC. A partir de um produto/serviço do catálogo, um usuário pode criar um anúncio de compra ou de venda. Aquele que coloca algo para vender é definido como um fornecedor. Aquele que coloca algo para comprar é definido como um comprador. A busca de um parceiro é feita

pesquisando-se a lista de anúncios. Pode-se também procurar por um determinado produto e, então, verificar-se a existência de algum anúncio relativo ao produto. Essa busca pode ser feita a partir dos códigos UNSPSC. Alguém que se anuncia como fornecedor está procurando por alguém que se anuncia como comprador. Alguém que se anuncia como comprador está procurando por alguém que se anuncia como fornecedor.

As Figura 50 e 51 apresentam, respectivamente, um exemplo de publicação e de busca em um registro UDDI.

```
<businessEntity businessKey="uddi:fabrica_brastemp.example">
   <categoryBag>
      <keyedReference
         tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
         keyName="UNSPSC: Domestic kitchen appliances"
         keyValue="52.14.15.00.00"/>
      <keyedReference
         tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
         keyName="UNSPSC: Domestic laundry appliances and supplies"
         keyValue="52.14.16.00.00"/>
    </categoryBag>
</businessEntity>
```

Figura 50 - Exemplo de uma publicação em um registro UDDI

```
<?xml version = "1.0" encoding = "UTF-8"?>
<find_business xmlns = "urn:uddi-org:api_v3"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">
  <findQualifiers>
    <findQualifier>
      uddi:uddi.org:findqualifier:exactmatch
    </findQualifier>
  </findQualifiers>
  <categoryBag>
    <keyedReference
      keyValue = "52.14.15.00.00"
      tModelKey = "uddi:uddi.org:ubr:categorization:unspsc"/>
  </categoryBag>
</find_business>
```

Figura 51 - Exemplo de uma busca em um registro UDDI

A classe FunctionalDescription do COWS possui um atributo — metaModelType —, que define o meta-modelo da descrição funcional do agente. A descrição funcional de um *Web Service* poderia ser apontada por uma URI especificada em um atributo, por exemplo, webServiceURI. Para facilitar a implementação, a classe FunctionalDescription foi estendida com um novo atributo — unspsc —, que contém uma lista de códigos da taxonomia UNSPSC, associados à parte funcional do agente. Cada um dos parceiros pode definir uma lista de códigos UNSPSC que identifiquem o seu negócio. A Figura 52 apresenta

os anúncios dos agentes Carlos e Loja das Fábricas, com a especificação do atributo unspsc.

```
_#:agentPublish
[hasAgentInfo->
 _#[agentURI->CARLOS_URI,name->'Carlos',country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_VAREJO_URI,forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->'COMPRADOR'
             ,hasFunctionalDescription->_#[metaModelType->UNSPSC
                                           ,unspsc->{'52.14.15.00.00'}]
             ,hasRelationshipBinding->>
              {_#[contractRoleName->'VENDEDOR']}]}]]}
].

_#:agentPublish
[hasAgentInfo->
 _#[agentURI->LOJA_URI,name->'Loja das Fabricas',country->'Brasil']
,hasContractInfo->>
 {_#[contractURI->2PARCEIROS_VAREJO _URI,forumURI->>{FORUM1_URI}
     ,hasBindingInfo->
      _#[hasRoleBinding->>
         {_#[contractRoleName->'VENDEDOR'
             ,hasFunctionalDescription->_#[metaModelType->UNSPSC
                                           ,unspsc->{'52.14.15.00.00','52.14.16.00.00'}]
            ,hasRelationshipBinding->>
             {_#[contractRoleName->'COMPRADOR']}]}]]}
].
```

Figura 52 - Exemplo de instância de anúncio de agentes com o atributo unspsc

O *matchmaker* do meta-modelo UNSPSC, construído nessa implementação, considera que existe um resultado de sucesso quando os parceiros têm, pelo menos, um código UNSPSC em comum.

A Figura 53 apresenta a regra que testa se existe um casamento entre dois parceiros que tenham a descrição funcional especificada segundo o modelo definido acima.

```
#cows_matchMaker(MyFuncDesc,UNSPSC,PartnerFuncDesc,UNSPSC,Restriction) :-
        MyFuncDesc[unspsc->->MyUnspsc],
        PartnerFuncDesc[unspsc->->PartnerUnspsc],
        if (oneMemberList(PartnerUnspsc,MyUnspsc))
        then #cows_matchMakerReturnCode(UNSPSC,UNSPSC,success,Restriction)
        else #cows_matchMakerReturnCode(UNSPSC,UNSPSC,error,Restriction).

#cows_matchMakerReturnCode(UNSPSC,UNSPSC,error,'FUNCTIONAL MISMATCH').
#cows_matchMakerReturnCode(UNSPSC,UNSPSC,success,'').
```

Figura 53 - *MatchMaker* UNSPSC do COWS

Essa regra recebe um ponteiro para os objetos com as informações de duas descrições funcionais e testa se existe uma interseção entre a lista de códigos UNSPSC dessas duas descrições.

Essa regra poderia ser modificada para introduzir uma busca mais sofisticada, considerando que os códigos da taxonomia UNSPSC são hierárquicos. O código 52.14.00.00.00 engloba toda a família de códigos que têm 52.14 como os quatro primeiros dígitos. Portanto, um vendedor que comercializasse produtos que estão associados ao código 52.14.00.00.00 aceitaria compradores que estivessem procurando, por exemplo, um produto associado ao código 52.14.15.00.00.

Para realizar o casamento de serviços descritos de acordo com o vocabulário de outros meta-modelos, bastaria que se agregasse a regra que faz o casamento atualmente desses serviços.

Por exemplo, um *Web Service* em WSMO é descrito pela sua *capability*, que descreve o que um serviço oferece. Essa informação é fornecida por meio de precondições, condições posteriores e efeitos do serviço. A forma de se especificar as informações do comprador da passagem é por meio da descrição do seu objetivo, do seu gol. Esse gol é especificado em termos de condições posteriores e efeitos esperados pelo solicitante da busca.

As Figuras 54 a 57 apresentam a descrição funcional de um serviço de venda de passagens de trem com origem e destino na Áustria ou Alemanha. A capacidade do serviço seria descrita por:

- precondição: os dados de entrada contêm informações sobre o comprador do bilhete, o bilhete e o passageiro. Para que os dados de entrada sejam considerados válidos, o ponto de partida e o ponto de chegada da viagem devem estar localizados na Áustria ou na Alemanha, a data de compra tem que ser posterior à data corrente e o pagamento deve ser efetuado com cartão de crédito.
- condição posterior: o sistema retorna uma ordem de compra de um bilhete para pagamento com cartão de crédito.
- efeito: o bilhete emitido é entregue ao comprador, em um endereço informado ou por e-mail.

As Figuras 58 a 60 apresentam uma requisição de busca para a compra de uma passagem definida por:

- condição posterior: a emissão de uma ordem de compra de um bilhete para uma viagem de Innsbruck (Áustria) para Frankfurt (Alemanha), no dia 17 de julho de 2007, tendo com comprador e passageiro Tim Berners-Lee.
- efeito: a entrega por e-mail do bilhete

```
namespace <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041119/resources/VTAService#>>
    dc:      <<http://purl.org/dc/elements/1.1#>>
    dt:      <<http://www.wsmo.org/ontologies/dateTime#>>
    tc:      <<http://www.wsmo.org/ontologies/trainConnection#>>
    po:      <<http://www.wsmo.org/ontologies/purchase#>>
    loc:     <<http://www.wsmo.org/ontologies/location#>>
    ucase:<<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041105/resources/>>
    targetnamespace: <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041119/resources/ws#>>

webservice <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041119/resources/ws.wsml>>

    nonFunctionalProperties
        dc:title hasValue "OEBB Online Ticket Booking Web Service"
        dc:creator hasValue "DERI International"
        dc:description hasValue "web service for booking online train tickets for Austria and Germany"
        dc:publisher hasValue "DERI International"
        dc:contributor hasValues {"Michael Stollberg", "Ruben Lara", "Holger Lausen"}
        dc:date hasValue "2004-10-04"
        dc:type hasValue <<http://www.wsmo.org/2004/d2/#webservice>>
        dc:format hasValue "text/html"
        dc:language hasValue "en-us"
        dc:relation hasValues {<<http://www.wsmo.org/ontologies/dateTime>>,
            <<http://www.wsmo.org/ontologies/trainConnection>>,
            <<http://www.wsmo.org/ontologies/purchase>>,
            <<http://www.wsmo.org/ontologies/location>>}
        dc:coverage hasValues {tc:austria, tc:germany}
        dc:rights hasValue <<http://deri.at/privacy.html>>
        version hasValue "$Revision: 1.5 $"
    endNonFunctionalProperties

    importedOntologies {<<http://www.wsmo.org/ontologies/dateTime>>,
        <<http://www.wsmo.org/ontologies/trainConnection>>,
        <<http://www.wsmo.org/ontologies/purchase>>,
        <<http://www.wsmo.org/ontologies/location>>}
```

Figura 54 - *Web Service* (WSMO) - propriedades não-funcionais

```
capability oebbWSCapability
        precondition
            axiom oebbWSprecondition
                nonFunctionalProperties
                    dc:description hasValue "The oebbWSprecondition puts the following conditions ont
                        the input: it has to include a buyer with a billTo and a shipTo
                        address, and credit card as a paymentMethod, and trip with the start- and
                        endlocation have to be in Austria or in Germany, and the departure date
                        has to be later than the current date. "
                endNonFunctionalProperties
            definedBy
                forAll ?Buyer, ?BuyerBilltoAddress, ?BuyerShiptoAddress, ?BuyerPaymentMethod,
                            ?Trip, ?Start, ?End, ?Departure (
                ?Buyer memberOf po:buyer[
                    po:billToAddress hasValue ?BuyerBilltoAddress,
                    po:shipToAddress hasValue ?BuyerShiptoAddress,
                    po:hasPaymentMethod hasValues {?BuyerPaymentMethod}] and
                ?BuyerBilltoAddress memberOf loc:address and
                ?BuyerShiptoAddress memberOf loc:address and
                ?BuyerPaymentMethod memberOf po:creditCard and
                ?Trip memberOf tc:trainTrip[tc:start hasValue ?Start,tc:end hasValue ?End,
                    tc:departure hasValue ?Departure] and
                (?Start.locatedIn = austria or ?Start.locatedIn = germany) and
                (?End.locatedIn = austria or ?End.locatedIn = germany) and
                dt:after(?Departure,currentDate)).
```

Figura 55 - *Web Service* (WSMO) - precondições

```
postcondition
  axiom oebbWSpostcondition
    nonFunctionalProperties
      dc:description hasValue "the output of the service is a purchase for a ticket for train
      trips wherefore  the start- and endlocation have to be in Austria or in Germany and
      the departure date has to be later than the current Date."
    endNonFunctionalProperties
    definedBy
      forAll ?Purchase, ?Seller, ?OEBBContactInformation, ?Purchaseorder,
              ?PaymentMethod, ?Product, ?Ticket, ?Itinerary, ?Trip, ?Start, ?End
        (?Purchase memberOf po:purchase[ purchaseorder hasValue ?Purchaseorder,
          seller hasValue ?Seller] and
        ?Seller memberOf po:seller[contactInformation hasValue ?OEBBContactInformation,
                  acceptsPaymentMethod hasValues {?PaymentMethod}] and
        ?OEBBContactInformation memberOf po:contactInformation[
                    name hasValue "Oesterreichische Bundesbahn",
                    emailaddress hasValue "office@oebb.at",
                    physicalAddress hasValue ?OEBBAddress] and
        ?OEBBAddress memberOf loc:address[
                    streetAddress hasValue "Hauptfrachtenbahnhof 4",
                    city hasValue innsbruck,
                    country hasValue austria] and
        ?Purchaseorder memberOf po:purchaseOrder[
                    product hasValues {?Product},
                    payment hasValue ?PaymentMethod] and
        ?PaymentMethod memberOf po:creditCard and
        ?Product memberOf po:product[item hasValues {?Ticket}] and
        ?Ticket memberOf tc:ticket[itinerary hasValue ?Itinerary] and
        ?Itinerary memberOf tc:itinerary[trip hasValue ?Trip] and
        ?Trip memberOf tc:trainTrip[start hasValue ?Start,end hasValue ?End,
          departure hasValue ?Departure] and
          (?Start.locatedIn = austria or ?Start.locatedIn = germany) and
          (?End.locatedIn = austria or ?End.locatedIn = germany) and
            dt:after(?Departure,currentDate)).
```

Figura 56 - *Web Service* (WSMO) - condições posteriores

```
effect
  axiom oebbWSeffect
    nonFunctionalProperties
      dc:description hasValue "the sold ticket is delivered to the buyer via a drop ship carrier or via email."
    endNonFunctionalProperties
    definedBy
      forAll ?Delivery, ?Product, ?Buyer, ?BuyerShipToAddress, ?Seller,
             ?OEBBContactInformation, ?OEBBAddress
        (((?Delivery memberOf po:dropShipDelivery[deliveryItem hasValues {?Product},
             receiver hasValue ?Buyer, sender hasValue ?Seller,carrier hasValue PostAt ]) or
          (?Delivery memberOf po:onlineDelivery[deliveryItem hasValues {?Product},
             receiver hasValue ?Buyer, onlineDeliveryMethod hasValue "email"])) and
         ?Product memberOf po:product[item hasValues {?Ticket}] and
         ?Buyer memberOf po:buyer[shipToAddress hasValue ?BuyerShipToAddress] and
         ?BuyerShipToAddress memberOf loc:address and
         ?Seller memberOf po:seller[contactInformation hasValue ?OEBBContactInformation] and
         ?OEBBContactInformation memberOf po:contactInformation[
           name hasValue "Oesterreichische Bundesbahn",emailaddress hasValue "office@oebb.at",
           physicalAddress hasValue ?OEBBAddress                    ] and
          ?OEBBAddress memberOf loc:address[streetAddress hasValue "Hauptfrachtenbahnhof 4",
            city hasValue innsbruck, country hasValue austria]).
```

Figura 57 - *Web Service* (WSMO) - efeitos



```
namespace
      <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041119/resources/SpecificTrainTripInnbsruckFrankfurt#>>
            dc:<<http://purl.org/dc/elements/1.1#>> dt:<<http://www.wsmo.org/ontologies/dateTime#>>
            tc:<<http://www.wsmo.org/ontologies/trainConnection#>>
            po:<<http://www.wsmo.org/ontologies/purchase#>>
            loc:<<http://www.wsmo.org/ontologies/location#>>
            kb:<<http://www.wsmo.org/ontologies/kb#>>
            wsml:<<http://www.wsmo.org/2004/d2/#>>
            xsd:<<http://www.w3.org/2001/XMLSchema#>>
goal <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041119/resources/goal.wsml>>
   nonFunctionalProperties
            dc:title hasValue "Buying a train ticket from Innsbruck to Frankfurt on..."
            dc:creator hasValue "DERI International"
            dc:subject hasValues {"Train Tickets", "Online Ticket Booking", "Train trip"}
            dc:description hasValue "Express the goal of buying a concrete ticket for a train trip"
            dc:publisher hasValue "DERI International"
            dc:contributor hasValues{"Michael Stollberg",
                 <<http://www.deri.org/foaf#lara>>,<<http://homepage.uibk.ac.at/~c703240/foaf.rdf>>,
                 <<http://homepage.uibk.ac.at/~c703262/foaf.rdf>>, <<http://www.deri.org/foaf#haller>>}
            dc:date hasValue "2004-10-04"
            dc:type hasValue <<http://www.wsmo.org/2004/d2#goals>>
            dc:format hasValue "text/html"
            dc:identifier hasValue
     <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041008/resources/goal.wsml>>
            dc:language hasValue "en-US"
            dc:relation hasValues {<<http://www.wsmo.org/ontologies/dateTime>>,
                 <<http://www.wsmo.org/ontologies/trainConnection>>,
                 <<http://www.wsmo.org/ontologies/purchase>>,
                 <<http://www.wsmo.org/ontologies/location>>}
            dc:coverage hasValue "ID:7029392 Name:World"
            dc:rights hasValue <<http://deri.at/privacy.html>>
            version hasValue "$Revision: 1.4 $"
   endNonFunctionalProperties
   usedMediators
            ggMediator <<http://www.wsmo.org/2004/d3/d3.3/v0.1/20041105/resources/ggm1.wsml>>
   importedOntologies {<<http://www.wsmo.org/ontologies/dateTime>>,
                       <<http://www.wsmo.org/ontologies/kb>>}
```

Figura 58 - Busca (WSMO) - propriedades não-funcionais

```
postcondition
  axiom purchasingTicketForSpecificTraintrip
    nonFunctionalProperties
      dc:description hasValue "The goal postcondition specifies that Tim Berners-Lee wants
      to go buy a train ticket from Innsbruck to Frankfurt, departing from innsbruckHbf
      on 17th July 2004 at 6 p.m."
    endNonFunctionalProperties
    definedBy
      exists ?Purchase, ?Purchaseorder, ?Buyer, ?Product, ?PaymentMethod, ?Ticket, ?Itinerary,
            ?Passenger, ?Trip, ?DepartureTime, ?DepartureDate, ?Departure,
            ?TBLAddress, ?TBLContactInformation
    (?Purchase memberOf po:purchase[
        po:purchaseorder hasValue ?Purchaseorder, po:buyer hasValue ?Buyer] and
     ?Buyer memberOf po:buyer[po:contactInformation hasValue ?TBLContactInformation,
        po:billToAddress hasValue ?TBLAddress, po:hasPayment hasValues {?PaymentMethod}] and
     ?TBLContactInformation memberOf po:contactInformation[po:name hasValue "Tim Berners-Lee",
        po:emailaddress hasValue "tbl@w3c.org"] and
     ?TBLAddress memberOf loc:address[po:streetAddress hasValue "32 Vassar Street",
        po:city hasValue boston, po:state hasValue massachusetts, po:country hasValue usa] and
     ?Purchaseorder memberOf po:purchaseOrder[po:product hasValues {?Product},
        po:payment hasValue ?PaymentMethod] and
     ?PaymentMethod memberOf po:creditCard[po:type hasValue masterCard,
        po:creditCardNumber hasValue 5535446466867747, po:holder hasValue "Tim Berners-Lee",
        po:expMonth hasValue 09, po:expYear hasValue 2007] and
     ?Product memberOf po:product[po:item hasValues {?Ticket}] and
     ?Ticket memberOf tc:ticket[ po:itinerary hasValue ?Itinerary] and
     ?Itinerary memberOf tc:itinerary[po:passenger hasValue ?Passenger, po:trip hasValue ?Trip] and
     ?Passenger memberOf tc:person[po:name hasValue "Tim Berners-Lee"] and
     ?Trip memberOf tc:trainTrip[po:start hasValue kb:innsbruckHbf, po:end hasValue kb:frankfurtHbf,
        po:departure hasValue ?Departure] and
     ?Departure memberOf dt:dateAndTime[dt:date hasValue ?DepartureDate,
        dt:time hasValue ?DepartureTime] and
     ?DepartureDate memberOf dt:date[dt:dayOfMonth hasValue 17, dt:monthOfYear hasValue 07,
        dt:year hasValue 2007] and
     ?DepartureTime memberOf dt:time[dt:hourOfDay hasValue 18] ).
```

Figura 59 - Busca (WSMO) - condições posteriores

```
effect
 axiom getTicketDelivererd
   nonFunctionalProperties
     dc:description hasValue "The goal effect that a ticket is delivered via emial."
   endNonFunctionalProperties
   definedBy
     exists ?Delivery, ?Product, ?Buyer(
       ?Delivery memberOf po:onlineDelivery[po:deliveryItem hasValues {?Product},
           po:receiver hasValue ?Buyer, po:onlineDeliveryMethod hasValue "email"] and
       ?Product memberOf po:product[po:item hasValues {?Ticket}]and
       ?Ticket memberOf tc:ticket).
```

Figura 60 - Busca (WSMO) - efeitos