5
Everyware: Uma Arquitetura para Aplicações baseadas em serviços utilizando a Web Semântica

Everyware é uma plataforma para o desenvolvimento de aplicações peer-to-peer distribuídas que envolvam serviços utilizando a Web Semântica. O objetivo desta plataforma é oferecer aos desenvolvedores deste tipo de aplicação um conjunto de ferramentas para permitir a integração e composição de serviços disponíveis na Internet tanto sob o ponto de vista funcional quanto sob o ponto de vista de dados.

Primeiramente, é importante a adoção de padrões abertos da indústria, permitindo a integração de serviços disponíveis em diferentes plataformas. Existem dois componentes básicos nessa plataforma, um provedor de serviços leve (Everyware LW) integrado com os padrões da Web Semântica e capaz de responder a chamadas feitas através de protocolos síncronos e assíncronos e um framework de coordenação que pode ser utilizado para compor diferentes serviços no contexto de processos de negócios complexos que envolvam inúmeras interações.

5.1.
Princípio de Organização

A plataforma Everyware adota um princípio básico para o desenvolvimento de aplicações baseadas em serviços. Toda aplicação é um conjunto de estruturas de coordenação funcional e de dados de serviços distribuídos que podem ser encapsuladas em serviços de forma recursiva.

Para exemplificar este conceito, podemos tomar como exemplo uma aplicação de agendamento de compromissos em grupo. Ela pode ser definida como a coordenação de serviços individuais de calendário de forma a encontrar o melhor horário para o compromisso em grupo.

PUC-Rio - Certificação Digital Nº 0115631/CA

5.2. Componentes

Utilizando este princípio de construção de aplicações baseadas em serviços, foram desenvolvidos componentes capazes de facilitar a programação e implantação dessas aplicações levando em conta os principais requisitos levantados durante a construção dos estudos de caso. Estes componentes são: um provedor de serviços leve, um framework para a construção de serviços de coordenação e um provedor com suporte à representação semântica dos dados.

5.2.1. Provedor de Serviços leve

Através do uso de um provedor de serviços leve, os usuários de uma aplicação desenvolvida sob a plataforma Everyware podem disponibilizar serviços em dispositivos com baixa disponibilidade sem grandes necessidades de configuração ou administração. Este provedor consome menos recursos de memória e processamento, sendo mais leve do que os provedores existentes.

Peers comuns (com PCs ou telefones celulares) podem prover serviços utilizando somente protocolos assíncronos. Everyware LW implementa uma camada de transporte baseada em SMTP. Ele funciona como um Proxy para o servidor de e-mails do usuário, permitindo que requisições a serviços sejam enviados à mesma conta de e-mail utilizada para mensagens normais de texto. O provedor Everyware processa as requisições enquanto as mensagens comuns são direcionadas normalmente ao cliente de e-mail do usuário (por exemplo, Microsoft Outlook™ ou Eudora™). O suporte a protocolos síncronos também é possível através de um transporte HTTP. A figura apresentada a seguir ilustra o fluxo de informação de uma requisição de serviço simples para um peer comum na rede.

PUC-Rio - Certificação Digital Nº 0115631/CA

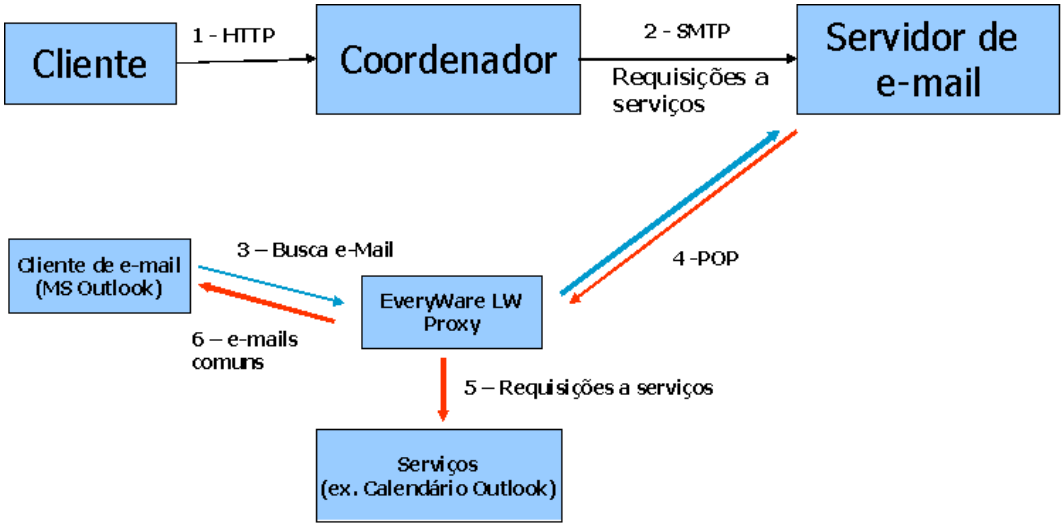

Figura 11 - Fluxo de informação do Everyware LW. Um cliente de um serviço envia um pedido para um coordenador que invoca os serviços para executar um processo. Everyware LW funciona com um Proxy para o serviço de e-mail, processando requisições de serviço toda vez que o dono do serviço checa por e-mail através de um cliente de e-mail comum

PUC-Rio - Certificação Digital Nº 0115631/CA

5.2.2. Framework de coordenação

A infra-estrutura de coordenação é responsável por gerenciar a execução de um processo através da coordenação de diversos serviços. Apesar de poder ser instalada em qualquer peer da rede, é mais interessante do ponto de vista de desempenho que ela seja instalada em um servidor com alto grau de disponibilidade de forma a garantir que a coordenação do processo sofra o mínimo de interrupções possível.

A arquitetura proposta procura separar completamente a comunicação com os componentes dos mecanismos de coordenação. Este desacoplamento permite que a infra-estrutura utilizada para coordenação seja reaproveitada e que diferentes tipos e tecnologias de componentes possam ser utilizados de forma homogênea.

Usando a taxonomia de componentes definida por (Sommerville, 00), podemos classificar os componentes da máquina de coordenação em cinco categorias: sensores, coordenadores, atuadores, comunicadores e interface.

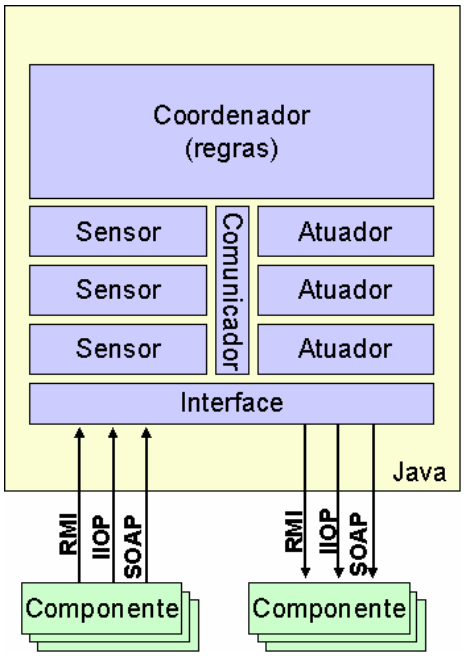

Figura 12 - Arquitetura geral do sistema em função dos componentes

PUC-Rio - Certificação Digital Nº 0115631/CA

- Os coordenadores são responsáveis por executar a lógica de coordenação seguindo um conjunto de regras de negócio, especificadas em uma linguagem de descrição de serviços. Estes componentes são estimulados a partir de atividades dos sensores
- Os sensores são responsáveis por captar e tratar estímulos feitos ao sistema por meio da interface. A partir dos componentes comunicadores, estes são passados ao coordenador
- Os atuadores são responsáveis por executar as ações determinadas pelos coordenadores, utilizando a interface como meio de comunicação
- Os comunicadores são utilizados para a troca de dados entre os componentes, permitindo a modulariação do framework e independência dos componentes
- A interface é responsável pela troca de dados com componentes externos ao sistema

Em resumo, os sensores são responsáveis por capturar respostas de componentes e requisições de processos, enquanto os atuadores são responsáveis por invocar os serviços disponibilizados por diferentes componentes uma vez que a máquina de coordenação determinou a execução de mais um passo de um determinado processo. Uma camada única de comunicação permite a comunicação entre todos os componentes de forma homogênea.

5.2.2.1. Design

O design do sistema tem como princípio a facilidade de compilação de outras linguagens para a arquitetura e a flexibilidade para criação de interfaces com diferentes tecnologias de componentes.

Estes princípios tornaram necessária a criação da entidade *CoordinationFactory*, que simplifica o mapeamento de outras linguagens de coordenação para a arquitetura. É possível desenvolver aplicações que geren código para a arquitetura. Neste caso, deve ser criada uma entidade *Factory* (p.ex.: *BasicFactory*) que será responsável por criar as máquinas de estados e tratadores de acordo com o processo definido.

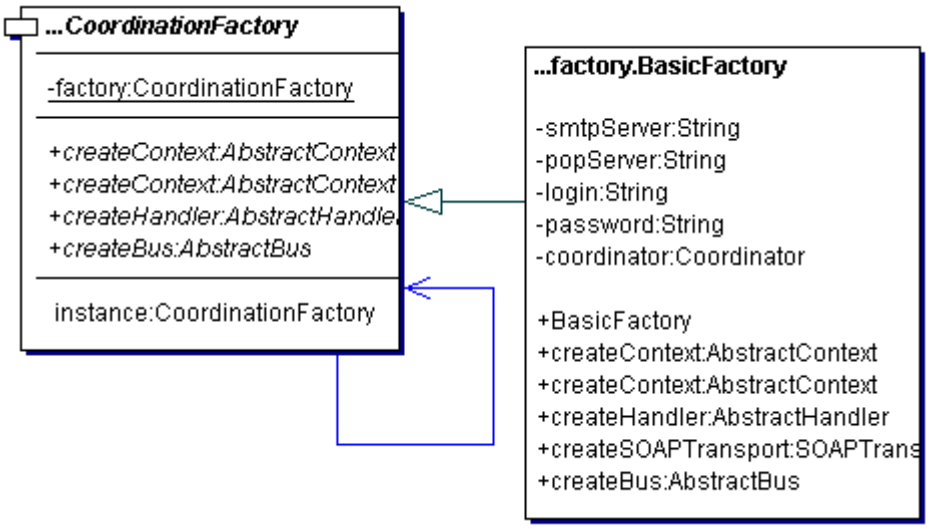

Figura 13 - A classe Coordination Factory

A classe *AbstractBus* fica responsável pela comunicação dos componentes de coordenação com entradas e saídas do sistema geradas pelos sensores e atuadores. Estes sensores tipicamente são implementações para algum protocolo de comunicação específico. A classe *SMTPIOManager* é capaz de receber requisições através do protocolo SMTP.

PUC-Rio - Certificação Digital Nº 0115631/CA

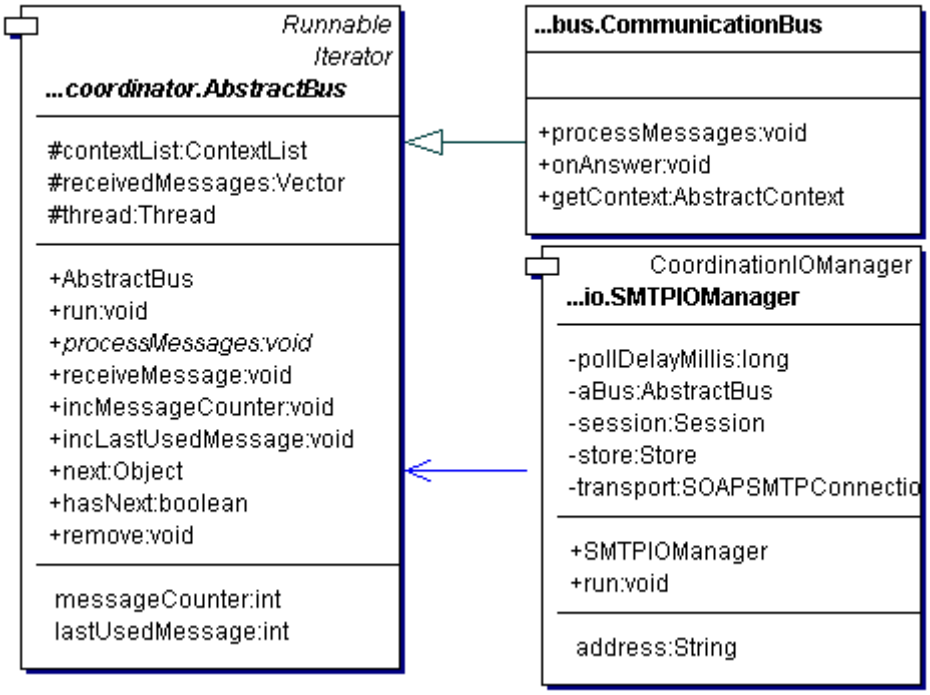

Figura 14 - A classe AbstractBus

Por outro lado, os atuadores são proxies dinâmicos capazes de invocar qualquer componente distribuído em tempo de execução. Para o caso específico de Web Services, a classe *ServiceClient* implementa um Proxy dinâmico para Web services distribuídos. A interface *ServiceClient* permite que sejam implementados componentes para as diferentes tecnologias de componentes.

Finalmente, outro aspecto importante da implementação é a manutenção de estados e execução das ações. Através das classes AbstractContext e AbstractHandler, é possível manter o estado de execução dos processos e associar tratadores às respostas esperadas respectivamente.

PUC-Rio - Certificação Digital Nº 0115631/CA

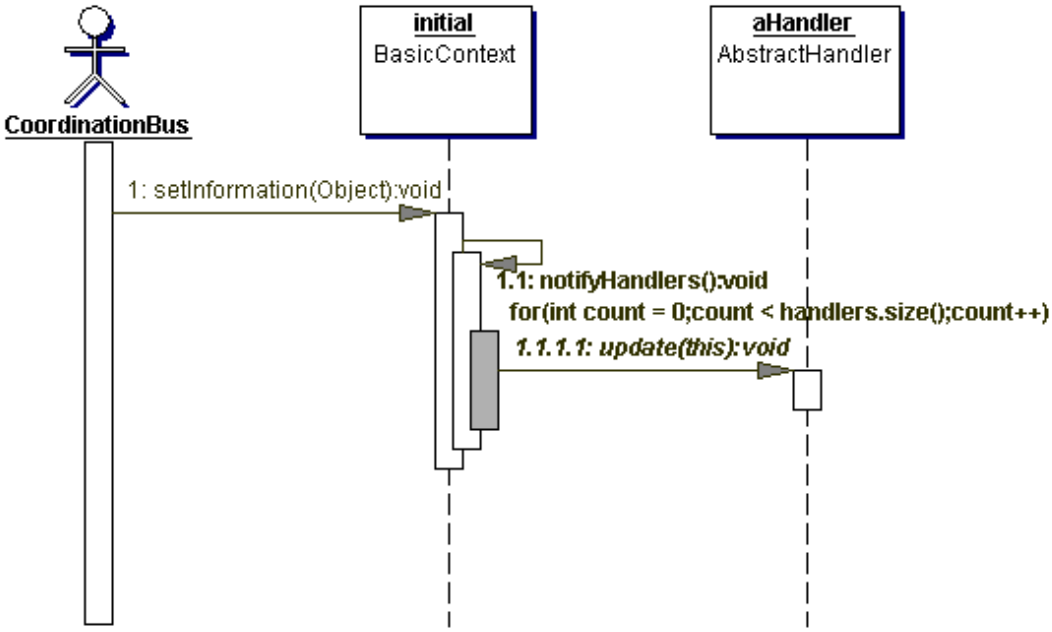

Figura 15 - A seqüência para tratamento de uma resposta de um componente

O uso de classes abstratas foi o artifício de design que permitiu a criação de uma infra-estrutura comum reutilizável independente dos diversos protocolos de comunicação implementados.

A classe Coordinator é responsável pela execução da máquina como um todo, armazenando todas as referências para os canais de comunicação, contextos e tratadores correspondentes.

5.2.2.2. Cenário de uso

A arquitetura proposta pode ser utilizada para diversos fins diferentes. É possível instanciar aplicações a partir do framework diretamente, aproveitando a infra-estrutura de coordenação e os componentes para comunicação com diferentes tecnologias de componentes. Outra possibilidade é realizar mapeamentos entre a estrutura do framework e linguagens de especificação de processos, permitindo sua instanciação automática dado um conjunto de regras de coordenação para um processo específico.

PUC-Rio - Certificação Digital Nº 0115631/CA

A partir do framework, é possível realizar a composição de componentes utilizando diferentes protocolos de transporte. Isto permite que componentes possam ser utilizados independente da maneira como estão disponibilizados.

Outro ponto importante é a independência de protocolo de comunicação com os componentes. Cada tecnologia define um protocolo de acesso diferente, o que torna necessária a criação de invocadores dinâmicos para cada tipo de componente diferente.

Além disso, é possível flexibilizar também o formato do conteúdo das mensagens, permitindo que qualquer dado comunicado com algum componente seja processado de forma homogênea pelo framework.

Através das instanciações destes diferentes hot-spots, é possível utilizar o framework para coordenar componentes em diversos contextos diferentes. É possível coordenar processos que envolvam tecnologias de componentes diferentes, utilizando protocolos de transporte diferentes e mecanismos de trocas de mensagens diferentes.

PUC-Rio - Certificação Digital Nº 0115631/CA

5.2.3. Extensão para suporte à semântica nos serviços

Com o intuito de prover semântica aos serviços, o provedor de serviços Apache SOAP foi estendido de forma a permitir o uso de tipos definidos em RDFSchema e DAML+ OIL. A especificação WSDL permite que sejam utilizados diferentes sistemas de tipos na definição de um Web Service. Aproveitando esta possibildade, é possível definir conceitos RDF ou DAML como dados que serão passados a serviços como parâmetros.

O uso de RDF ou DAML como sistema de tipos em descrições de serviços baseadas na linguagem WSDL ainda não é muito difundido. Normalmente, os parâmetros fornecidos aos serviços e os resultados dos mesmos são definidos em tipos complexos XML Schema. No entanto, XML Schema apenas consegue definir estruturas de dados complexas formadas por agregações de tipos básicos, enquanto que RDFSchema e DAML possuem mais poder para expressar restrições e relações entre tipos.

A descrição de dados em RDF ou DAML traz semântica às informações, permitindo dedução e inferência. Uma descrição em RDF é um conjunto de triplas, que sempre representam um recurso, uma propriedade e seu valor associado. A

representação por triplas é a forma mais simples de representar informações, permitindo um tratamento simplificado e a criação de mecanismos de consulta poderosos. Cada tripla é considerada uma sentença ou afirmação. Um conjunto de afirmações, associadas a um esquema, pode ser validado e seu significado pode ser inferido a partir dos significados contidos neste esquema. Regras de inferência podem ser associadas a essas estruturas da informação, permitindo o raciocínio em cima dos dados processados. Todos estes componentes juntos formam uma ontologia, que pode então ser compartilhada por diversas aplicações com o intuito de permitir o processamento automático das informações por máquinas.

Por estas razões, é possível definir um serviço semântico como um serviço que troca informações com outras entidades utilizando somente ontologias para representar os dados relativos à computação realizada. A diferença entre um serviço considerado semântico e um serviço comum está representada na figura a seguir:

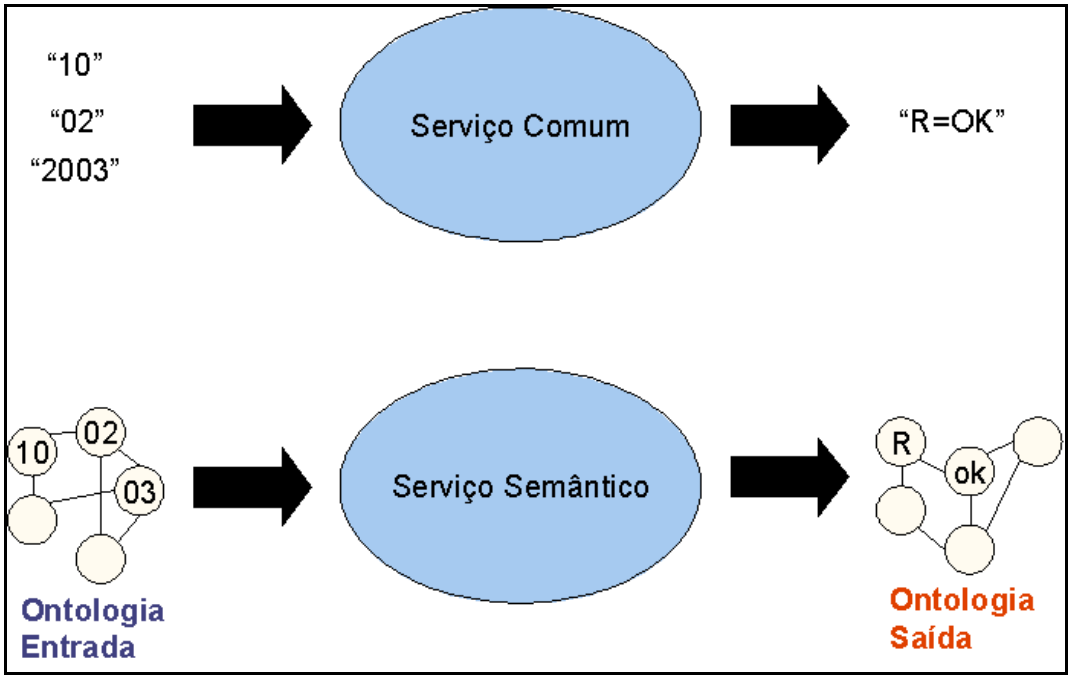

Figura 16 – A diferença entre um serviço semântico e um serviço comum. Enquanto o serviço comum recebe e envia informações sem nenhum tipo de representação que indique seu significado, os serviços semânticos utilizam estruturas e ontologias para permitir inferência sobre a informação por máquinas.

A introdução de informações semânticas nos dados compartilhados pelos serviços leva a algumas modificações na infra-estrutura de software para implantação dos mesmos. Por esta razão, a plataforma Everyware oferece uma extensão a um

PUC-Rio - Certificação Digital Nº 0115631/CA

provedor de serviços comum que permite a inclusão deste tipo de informação. Esta extensão permite que requisições a serviços que envolvam dados descritos em RDF e DAML sejam seriadas corretamente. Utilizando um processador RDF/DAML chamado (JENA), o provedor de serviços é capaz de verificar a chamada ao serviço, identificar os esquemas RDF ou DAML correspondentes, validar as instâncias contra esses esquemas e instanciar objetos correspondentes a estas instâncias na linguagem de programação na qual o serviço está implementado. No caso do Everyware, foi implementado suporte à linguagem (JAVA).

Para incluir um tipo RDF ou DAML em uma descrição de serviço feita em WSDL, é necessário primeiramente incluir o *namespace* correspondente ao esquema em RDF ou DAML que representa os dados que serão trocados com o serviço. Esta inclusão deve ser feita ainda no cabeçalho da descrição WSDL, no tag *definitions* (no exemplo, a inclusão é feita para o namespace iCalendar):

```
<definitions name="CalendarServiceInterface"
   targetNamespace="http://sw.inf.puc-rio.br/CalendarService"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:tns="http://sw.inf.puc-rio.br/CalendarService"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:iCalendar="http://ilrt.org/discovery/2001/06/schemas/ical-
full/hybrid.rdf#"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

Figura 17 – Cabeçalho de arquivo WSDL incluindo o namespace de um esquema RDF.

Uma vez feita esta inclusão, o passo seguinte é importar as definições dos esquemas que serão utilizados pelo serviço. Estes esquemas permitirão a validação das informações pelo processador de ontologias, garantindo a consistência dos dados tratados pelo serviço.

```
<import
   namespace="http://ilrt.org/discovery/2001/06/schemas/ical-
full/hybrid.rdf#"
   location="http://ilrt.org/discovery/2001/06/schemas/ical-
full/hybrid.rdf"/>
```

Figura 18 – Importação dos esquemas utilizados pelo serviço.

PUC-Rio - Certificação Digital Nº 0115631/CA

Depois da importação dos esquemas que serão utilizados, é necessário definir o formato das mensagens que serão trocadas com os serviços. Este passo é importante pois um esquema pode definir diversas classes e conceitos, tornando imperativo definir quais desses conceitos estarão sendo enviados e recebidos pelos serviços envolvidos.

Para cada mensagem de entrada e saída, é necessário definir uma série de partes, onde cada uma representa uma classe definida no esquema importado. É importante ressaltar que vários esquemas podem ser utilizados e que os conceitos definidos em cada um podem ser misturados em uma mesma mensagem.

```
<message name="SetupUserAppointmentRequest">
      <part name="event" type="iCalendar:VEVENT"/>
</message>
<message name="SetupUserAppointmentResponse">
      <part name="confirmation" type="iCalendar:VEVENT"/>
</message>
```

Figura 19 – Definição do conteúdo de uma mensagem para um serviço semântico

Em seguida, devem ser definidos o ponto de entrada do serviço e todas as suas operações, que consistem em mensagens de entrada e saída. As informações de seriação relativas ao protocolo de transporte utilizado para a conexão com o serviço são fornecidas a seguir. Estas informações não são influenciadas pelo uso de marcação semântica para as mensagens. Porém, são necessárias para que provedores e clientes de serviço saibam como os mesmos devem ser disponibilizados e acessados via rede.

```
<portType name="CalendarService">
        <operation name="insert">
            <input message="tns:SetupUserAppointmentRequest"
name="SetupUserAppointmentRequest"/>
            <output message="tns:SetupUserAppointmentResponse"
name="SetupUserAppointmentResponse"/>
        </operation>
</portType>


<binding name="CalendarServiceBinding" type="tns:CalendarService">
   <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="insert">
```

PUC-Rio - Certificação Digital Nº 0115631/CA

```
       <soap:operation soapAction="" style="rpc"/>
           <input name="SetupUserAppointmentRequest">
             <soap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://sw.inf.puc-rio.br/CalendarService"
use="encoded"/>
           </input>
           <output name="SetupUserAppointmentResponse">
              <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="http://sw.inf.puc-rio.br/CalendarService"
use="encoded"/>
           </output>
  </operation>
</binding>
<service name="CalendarService">
   <port binding="binding:CalendarServiceBinding"
name="CalendarServicePort">
      <soap:address location="http://sw.inf.puc-rio.br/services/router"/>
   </port>
</service>
```

Figura 20 – Informações de implantação de serviço

Como forma de comparação, um arquivo de descrição de um serviço semelhante ao anterior, porém sem marcação semântica, é apresentado a seguir. A diferença fundamental entre as duas especificações está na ausência de esquemas importados e de seus respectivos namespaces.

```
<definitions name="CalendarService"
    targetNamespace="http://www.puc-rio.br/service/CalendarService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:binding="http://www.puc-rio.com/definitions/Calendar Interface"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
>
```

Figura 21 – Importação de namespaces. Não há necessidade de namespaces adicionais para os esquemas de marcação semântica

Uma vez definidos os namespaces, é necessário novamente definir o formato das mensagens. Como XML Schema é o sistema de tipos padrão em WSDL, não há necessidade de importação de esquemas e de definição de tipos quando não há uso de tipos complexos.

PUC-Rio - Certificação Digital Nº 0115631/CA

```
<message name="insertRequest">
   <part name="sDay" type="xsd:int"/>
   <part name="sMonth" type="xsd:int"/>
   <part name="sYear" type="xsd:int"/>
   <part name="sHour" type="xsd:int"/>
   <part name="sMinute" type="xsd:int"/>
   <part name="duration" type="xsd:int"/>
   <part name="subject" type="xsd:string"/>
</message>
<message name="insertResponse">
   <part name="result" type="xsd:string"/>
</message>
```

Figura 22 – Definição das mensagens trocadas por um serviço comum em função de tipos básicos presentes em XML Schema

Após a definição das mensagens, é necessário definir as operações presentes em cada serviço. Estas operações precisam ser obrigatoriamente definidas em função das mensagens definidas previamente. É importante ressaltar que a ordem dos parâmetros é um fator crucial e precisa ser incluída na especificação dos serviços comuns.

```
<portType name="InsertSingleAppointmentService">
   <operation name="insert" parameterOrder="sDay sMonth sYear sHour
sMinute duration subject">
      <input message="tns:insertRequest" name="insertRequest"/>
      <output message="tns:insertResponse" name="insertResponse"/>
    </operation>
</portType>
```

Figura 23 – Especificação de operação de um serviço comum. A ordem dos parâmetros faz parte da especificação da operação.

O conteúdo restante da descrição WSDL é equivalente a de um serviço com marcação semântica. Todas as questões relativas à ligação com o protocolo de transporte e aos mecanismos de acesso ao serviço permanecem inalteradas.

PUC-Rio - Certificação Digital Nº 0115631/CA

5.3. Arquitetura

Everyware utiliza o conceito de coordenador central para executar um dado processo. Como cada processo também é um serviço, é possível compor diversos processos de forma a prover um serviço de funcionalidade mais complexa. É também importante ressaltar que a presença de diversos coordenadores em uma rede permite que diversos processos simultâneos possam ser atribuídos a diferentes coordenadores, permitindo uma maior distribuição do processamento.

Neste contexto, toda aplicação baseada no Everyware pode ser vista como um processo de composição de um ou mais serviços por um ou mais coordenadores. Cada aplicação baseada no Everyware deverá implementar o serviço de coordenação que deve então ser invocado pela aplicação ou usuário cliente visando a execução de uma lógica de negócios complexa.

O principal objetivo desta abordagem é prover um componente centralizado que pode atuar como "gerente" do processo inteiro, recebendo as informações provenientes dos serviços distribuídos e realizando uma computação sobre as mesmas. É importante notar que qualquer peer na rede pode atuar como coordenador nesta arquitetura, visto que qualquer processo de coordenação pode ser disponibilizado com um serviço no provedor Everyware. Considerando a hipótese onde todos são coordenadores, chega-se a uma abordagem peer-to-peer dita pura, ou seja, sem nenhum tipo de coordenação central. Coordenadores podem ser vistos como uma variação de sincronizadores (Frolund, 1996). Experiências práticas mostraram que o uso deste tipo de mecanismos diminui o esforço de desenvolvimento em uma ordem de grandeza (Agha, 1996).

No entanto, a experiência prática adquirida durante a implementação do segundo estudo de caso (calendário) mostrou que o uso de um coordenador central em máquinas com alta disponibilidade aumenta a tolerância a falhas e desempenho do sistema. Apesar de contrária às visões mais tradicionais p2p, essa abordagem se mostrou mais razoável tendo em vista que existem diferenças consideráveis entre os dispositivos envolvidos.

PUC-Rio - Certificação Digital Nº 0115631/CA

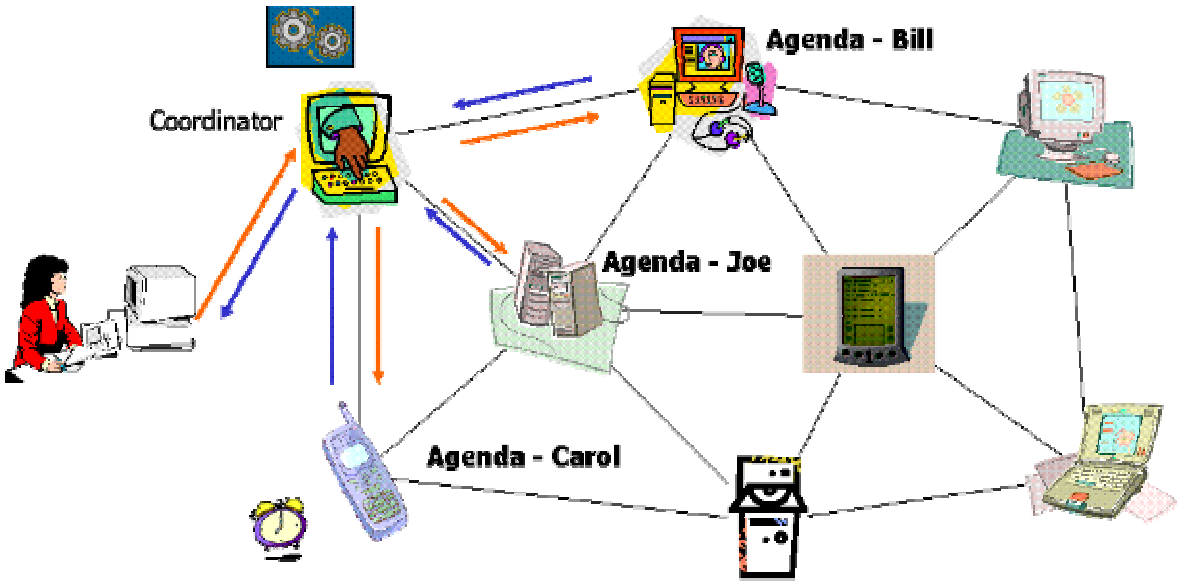

Figura 24 – Uma simples execução de um serviço de calendário. Depois de uma requisição, cada calendário individual é chamado e recupera informações relativas aos compromissos. O relógio próximo ao telefone celular da Carol ilustra o comportamento assíncrono desse serviço durante todo o processo de coordenação.

Cada requisição é recebida por meio de algum protocolo de transporte. Uma camada de transporte processa requisição, extraindo a chamada codificada em algum protocolo de acesso (Everyware opera com SOAP) de acordo com os diferentes protocolos de transporte. A chamada é então passada para a camada de gerenciamento do serviço, que processa o conteúdo e então cria um contexto para a requisição que será utilizado para coordenar o processo. A informação de contexto é passada em seguida para a camada de coordenação, que estabelece um identificador de transação que será utilizado para o acompanhamento das requisições subseqüentes no contexto do mesmo processo. As diferentes camadas e os dados processados respectivamente são apresentados a seguir.

PUC-Rio - Certificação Digital Nº 0115631/CA

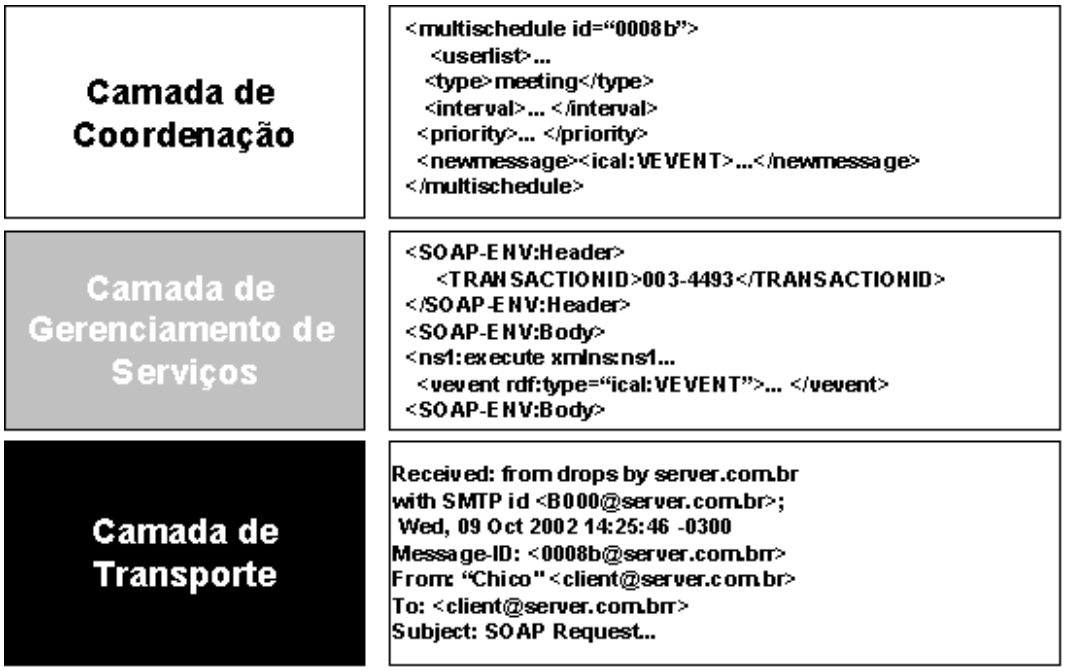

Figura 25 – As camadas do provedor Everyware e o conteúdo associado.

PUC-Rio - Certificação Digital Nº 0115631/CA

5.4. Questões de implementação e design

Everyware foi totalmente desenvolvido em Java. Entretanto, cada componente da arquitetura poderia ter sido desenvolvido em uma linguagem de programação diferente, uma vez que a comunicação é feita utilizando protocolos padrão de Web services.

Diversas questões referentes ao desenvolvimento e ao design da arquitetura foram levantadas durante o desenvolvimento da Everyware. O mais importante desafio para a coordenação de serviços peer-to-peer foi a heterogeneidade dos serviços.

Além da integração de diferentes plataformas de software utilizando protocolos padrão (integração funcional), há uma necessidade de integração de dados também. Por exemplo, cada fonte pode representar seus dados de uma forma diferente. Por essa razão, é importante definir uma maneira (ou um conjunto de maneiras) de representar a informação.

Podemos citar novamente o segundo estudo caso, onde algumas aplicações definem um compromisso com uma tupla (início-fim) enquanto outras usam uma tupla início duração para definir o intervalo de tempo de um compromisso. Outro problema comum é a granularidade de um compromisso. Alguns sistemas de calendário permitem somente compromissos que começam e terminam no mesmo

dia, enquanto outras permitem compromissos que duram diversos dias. Nesse caso, uma requisição para um compromisso de longa duração precisa ser quebrada em uma série de requisições para compromissos de um dia que correspondem ao mesmo período.

Essas diferenças têm de ser tratadas na implementação de cada serviço. Como muitos provedores de serviços não oferecem suporte à representação semântica dos dados, estes problemas aparecem de forma recorrente. Por esta razão, o provedor de serviços Everyware LW teve que incorporar suporte às linguagens de representação de ontologias, como RDF/RDFSchema e DAML. Dessa forma, o conteúdo dos serviços pode ser descrito de forma semântica, facilitando o seu processamento pelos serviços correspondentes.

O suporte à segurança na comunicação de dados não foi desenvolvido na primeira versão da Everyware. A confiabilidade das informações e de cada peer na rede também não foi atacada. Todo peer é considerado confiável sem nenhum tipo de restrição.

Limitar processamento desnecessário nos peers também era um requisito importante de design, visto que muitos dispositivos não podem buscar requisições indiscriminadamente (por exemplo, usuários de telefones celulares são taxados cada vez que se conectam na Internet utilizando WAP). Por esta razão, requisições são processadas utilizando mensagens de mail normais, sem a necessidade de contas de e-mail extras ou checagens adicionais por mensagens. Como peers comuns tendem a ter um tráfego menor do que servidores da aplicação convencionais, a arquitetura do Everyware é bem mais simples do que outras arquiteturas de provedores de Web Services.

Como exemplo dessas diferenças, é possível verificar que provedores de Web Services comuns baseados em transportes http precisam ser instalados configurados e gerenciados. Os baseados em SMTP checam por mensagens em intervalos de tempo regulares. Everyware LW processa requisições quando o usuário checa suas mensagens. Isto normalmente acontece quando o usuário acessa a Internet para realizar alguma tarefa. Se um serviço precisa de interação humana para ser realizado, existe uma possibilidade de que o usuário seja capaz de responder a chamada assim que a chamada seja processada.

PUC-Rio - Certificação Digital Nº 0115631/CA

5.5. Os estudos de caso utilizando Everyware

Como forma de contornar os problemas encontrados durante o desenvolvimento inicial dos estudos de caso e de testar os componentes e a arquitetura da Everyware, as aplicações foram convertidas para a nova plataforma. De forma a ilustrar como todos os componentes e a arquitetura ficam estruturados em uma aplicação real, o sistema de calendários e a busca semântica serão discutidos sob a ótica da plataforma Everyware, apresentando como seu design foi remodelado com os componentes da plataforma.

5.5.1. Calendário

A maior mudança na aplicação de calendários inclui a integração de marcação semântica aos dados consumidos e gerados pelos serviços. Foram criados três serviços novos que recebem modelos RDF como entrada e retornam uma resposta também em RDF. Estes serviços são:

- InsertSingleUserAppointment: Serviço capaz de inserir um compromisso com a participação de um único usuário. Este serviço recebe como entrada uma entidade VEVENT da ontolgia Hybrid iCal. A definição do conceito VEVENT pode ser vista no anexo I. Este serviço retorna também uma instância de VEVENT confirmando a marcação do evento solicitado. Em caso de falha, um exceção é levantada e nenhum VEVENT é retornado.
- InsertGroupAppointment: Serviço capaz de inserir um compromisso com a participação de um grupo de usuários. Este serviço realiza internamente chamadas ao serviço InsertSingleUserAppointment para a marcação de compromisso em cada um dos usuários. Para a descrição dos participantes do curos é utilizada a ontologia FOAF (friend of a friend) (Brickley & Miller, 2003).
- ListAppointments: Serviço que enumera os compromissos de um determinado usuário em um determinado espaço de tempo. Este serviço recebe um espaço de tempo representado por instâncias do conceito VDATE e retorna uma lista de instâncias de VEVENT contendo todos os compromissos presentes naquele serviço de calendário no período solicitado.

PUC-Rio - Certificação Digital Nº 0115631/CA

Existem ainda um conjunto de serviços necessários para a administração do sistema, como RemoveAppointment, CreateUser e CreateGroup.

Além destes serviços, foi desenvolvido um serviço de tradução capaz de traduzir conteúdo representado segunda a ontologia DAML Agenda para a ontologia Hybrid iCal. Este serviço permite a marcação e compromisso entre serviços de marcação individual que utilizem diferentes ontologias para representar a informação. A implementação do serviço possui a lógica de mapeamento entre os conceitos de cada uma das ontologias, identificando as correspondências e gerando representações correspondentes para ontologias dos dois tipos. Por exemplo, a chamada ao serviço de tradução com a seguinte ontologia no padrão DAML Agenda:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.daml.org/2001/10/agenda/agenda-ont#"
>
   <Meeting rdf:ID="http://www.inf.puc-rio.br/chicao/events/115">
     <day>
       <Day>
         <start>2003-05-25T09:00:00</start>
         <items rdf:parseType="daml:collection">
            <Block>
              <theme>Reunião Brainstorm - Projeto Serviços</theme>
              <items rdf:parseType="daml:collection"/>
              <duration>PT45M</duration>
            </Block>
          </items>
       </Day>
     </day>
   </Meeting>
</rdf:RDF>
```

Figura 26 – Representação original – DAML Agenda

Será convertida para a seguinte representação na ontologia Hybrid iCal:

PUC-Rio - Certificação Digital Nº 0115631/CA

```
<rdf:RDF xmlns:ical="http://ilrt.org/discovery/2001/06/schemas/ical-full/hybrid.rdf#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<ical:VEVENT rdf:about="http://www.inf.puc-rio.br/chicao/events/115">
   <ical:DTSTART>
      <ical:DATE-TIME>
         <rdf:value>20030525T090000</rdf:value>
      </ical:DATE-TIME>
  </ical:DTSTART>
  <ical:DTEND>
     <ical:DATE-TIME>
         <rdf:value>20030525T094500</rdf:value>
     </ical:DATE-TIME>
  </ical:DTEND>
  <ical:DESCRIPTION>Reunião Brainstorm - Projeto Serviços</ical:DESCRIPTION>
</ical:VEVENT>
</rdf:RDF>
```

Figura 27 – Representação final – Hybrid iCal

O serviço de coordenação, responsável por agendar os compromissos, foi implementado utilizando o framework de coordenação da plataforma. Para o tratamento dos processos, foram implementadas sub-classes de AbstractHandler, *hot-spot* responsável pelo tratamento das mensagens enviadas pelo coordenador durante o processo de agendamento de compromissos. Os tratadores básicos podem ser utilizados em diversos pontos do processo de coordenação. Alguns dos tratadores básicos são listados a seguir:

- **TranslationHandler**: É o tratador responsável por recuperar uma resposta a um pedido de tradução a um serviço e enviar ao serviço final correspondente. Este tratador encapsula a funcionalidade de tradução entre diferentes ontologias
- **RequestResponseHandler**: Responsável por encapsular a chamada a um serviço de calendário em particular. Este tratador encapsula a chamada a cada serviço, tratando as exceções e eventualmente procurando por serviços alternativos.
- **RDFHandler:** Este tratador é responsável por processar as informações representadas em RDF para determinação das próximas ações no escopo do processo de marcação de compromissos.

A arquitetura da aplicação de calendários utiliza o serviço de inserção de compromissos em grupo como serviço de coordenação que internamente resolve os serviços individuais, obtém as datas disponíveis para cada participante, propõe uma data para todos os participantes e confirma a data em cada serviço individual de calendário.

PUC-Rio - Certificação Digital Nº 0115631/CA

Para os serviços de calendário localizados em PCs (por exemplo, armazenados no Microsoft Outlook), o uso do provedor de serviços Everyware LW permite que os serviços baseados em MS Outlook sejam disponibilizados através da interface SOAP. Os usuários podem optar por transportes http e SMTP. O uso do transporte assíncrono é recomendável a usuários que mantém as máquinas desligadas durante boa parte do dia. Neste caso, o uso de transporte http pode inviabilizar a marcação de compromissos enquanto a máquina estiver desligada.

As chamadas via SMTP ficam armazenadas na caixa postal do usuário e são processadas toda vez que ele verifica se existem e-mails, junto com as mensagens comuns de e-mail. Todas as requisições aos serviços de calendário são processadas e enviadas de volta também via SMTP. Por exemplo, se um usuário desliga a máquina durante um fim de semana e recebe um pedido de agendamento de compromisso, este pedido será processado assim que ele ligar a máquina.

O uso dos tratadores desenvolvidos e da API de coordenação Everyware permite que o coordenador realize chamadas a serviços independente do protocolo de transporte utilizado por cada um. Para os desenvolvedores da aplicação, basta informar a url de acesso ao serviço que todo o processo de chamada e recebimento das respostas é encapsulado pelo framework.

O uso de múltiplos protocolos de transporte é necessário para a aplicação visto que serviços de calendário podem ser disponibilizados em servidores Web em transportes http ou em serviços implementados em cima de programas de e-mail utilizando o protocolo SMTP.

Para garantir tolerância a falhas no coordenador, um serviço de backup de coordenação também pode ser utilizado. Neste caso, todas as ações do coordenador são enviadas para um serviço de backup, responsável por assumir o processo de coordenação em caso de falha do coordenador. Este serviço de backup deve ser ativado pelos serviços participantes do processo em caso de falha do coordenador.

Além disso, ele é responsável por informar ao serviço original sobre seu novo status de coordenador, para evitar que dois serviços diferentes tentem realizar a coordenação de um mesmo processo.

5.5.2.Busca Semântica

A aplicação de busca semântica possui processos bem mais simples, que não necessitam da API de coordenação para seu gerenciamento. Os serviços de

PUC-Rio - Certificação Digital Nº 0115631/CA

informação da aplicação de busca semântica utilizam protocolos síncronos para acesso, o que elimina a necessidade de tratadores para as respostas às requisições.

No entanto, a aplicação de busca semântica segue a arquitetura proposta na plataforma Everyware para desenvolvimento de aplicações. Toda consulta realizada por um usuário final é processada por um serviço de coordenação. Este serviço, em um primeiro momento, invoca os diversos serviços de ontologia para identificar os conceitos que possuem relação com os termos procurados.

Ao receber as respostas dos serviços de ontologias, o mecanismo de coordenação utiliza a ontologia de mapeamento interna para transforma todos os conceitos para uma única representação, eliminando ambigüidades e inconsistências. Uma vez obtido o conjunto de conceitos, um serviço de listagem de fontes de informação é consultado, retornando uma lista de serviços disponíveis (por exemplo, Google, Amazon, etc). Finalmente, cada serviço possui um tratador associado, que realiza as chamadas e processa as respostas, que podem então ser apresentadas na tela para o usuário final.

Como o processo de busca requer somente recuperação de informações, eventuais falhas nos serviços de ontologia ou nos serviços de informação são ignoradas. O processo implementado no coordenador da aplicação de busca semântica é ilustrado na figura abaixo:

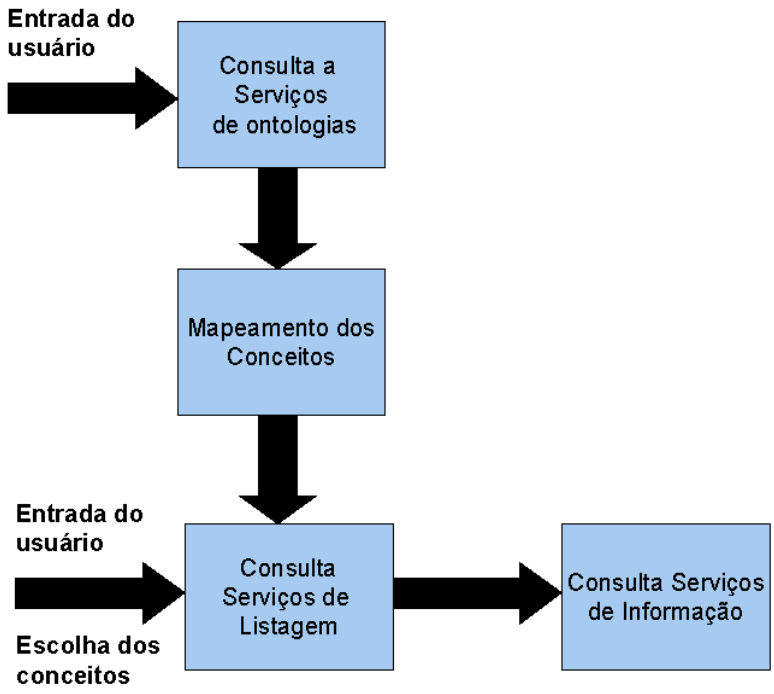

Figura 28 – Processo da aplicação de busca semântica

PUC-Rio - Certificação Digital Nº 0115631/CA

O desenvolvimento das aplicações dos estudos de caso mostrou-se interessante por permitir a reutilização dos componentes além da integração dos diversos aspectos de aplicações baseadas em serviços em uma arquitetura única. A adequação das aplicações à plataforma permitiu verificar seu uso e sua capacidade de prover infra-estrutura para aplicações de diferentes domínios.

PUC-Rio - Certificação Digital Nº 0115631/CA