UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Saulo Popov Zambiasi

UMA ARQUITETURA DE REFERÊNCIA PARA SOFTWARES ASSISTENTES PESSOAIS BASEADA NA ARQUITETURA ORIENTADA A SERVIÇOS

Florianópolis

2012

Saulo Popov Zambiasi

UMA ARQUITETURA DE REFERÊNCIA PARA SOFTWARES ASSISTENTES PESSOAIS BASEADA NA ARQUITETURA ORIENTADA A SERVIÇOS

Tese submetida ao Programa de Pós-graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas.
Orientador: Prof. Dr. Ricardo J. Rabelo

Florianópolis

2012

Catalogação na fonte pela Biblioteca Universitária
da
Universidade Federal de Santa Catarina

Z23a Zambiasi, Saulo Popov
Uma arquitetura de referência para softwares assistentes pessoais baseada na arquitetura orientada a serviços [tese] / Saulo Popov Zambiasi ; orientador, Ricardo José Rabelo. - Florianópolis, SC, 2012.
295 p.: il., grafs., tabs.

Tese (doutorado) – Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de sistemas. 2. Arquitetura de computador. 3. Software. 4. Arquitetura orientada a serviços. I. Rabelo, Ricardo José. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

CDU 621.3-231.2 (021)

Saulo Popov Zambiasi

UMA ARQUITETURA DE REFERÊNCIA PARA SOFTWARES ASSISTENTES PESSOAIS BASEADA NA ARQUITETURA ORIENTADA A SERVIÇOS

Esta Tese foi julgada adequada para obtenção do Título de "Doutor", e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 29 de março de 2012.

_____
Prof. José Eduardo Ribeiro Cury, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof. Ricardo José Rabelo, Dr.
Orientador
Universidade Federal de Santa Catarina

_____
Profª. Diana Francisca Adamatti, Dra.
Universidade Federal do Rio Grande

_____
Profª. Renata Pontin de Mattos Fortes, Dra.
Universidade de São Paulo

_____
Prof. Jomi Fred Hübner, Dr.
Universidade Federal de Santa Catarina

_____
Prof. João Bosco Mangueira Sobral, Dr.
Universidade Federal de Santa Catarina

À Patricia Leandra Barrufi Pinheiro,
minha companheira e amiga.
À minha família.

AGRADECIMENTOS

Ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas (PPGEAS).

À CAPES e ao programa de pós graduação, pelas bolsas concedidas durante o período de pesquisa.

Ao ex-coordenador do programa de pós-graduação Prof. Dr. Eugênio de Bona Castelan Neto e ao atual coordenador Prof. Dr. José Eduardo Ribeiro Cury pela força e auxílio que prestaram durante minha estadia como aluno do programa.

Ao Prof. Dr. Ricardo J. Rabelo, pelas orientações, conversas, conselhos e incentivos. Agradeço por tê-lo como orientador durante todos esses anos.

Aos colegas do grupo GSIGMA (Rui, Leandro, Gesser, Baldo, Piazza, Maiara, Drissen, Alexandra, Edmilson, Daniel, Omir, Cindy, Perin, Roque) pelo companheirismo durante todo período que pude participar do grupo.

À minha companheira Patricia, pelo incentivo e pela força durante todos os momentos difíceis. Agradeço muito por estar comigo quando mais precisei de uma amiga e companheira durante toda a fase do meu doutorado.

À minha família pela compreensão, e pelo apoio durante todos esses anos de pós-graduação e pelos momentos em que estive ausente devido aos muitos compromissos e que não pude participar de suas vidas.

À todos os meus amigos que me deram forças durante todo este período de enorme dedicação.

Viver com medo é uma grande experiência, não é? É assim que vive um escravo. Vi certas coisas que você não acreditaria. Naves de ataque ardendo ao largo de Orion. Vi raios cintilando na escuridão junto ao Portão de Tannhäuser. Todos esses momentos vão se perder no tempo, como lágrimas na chuva. Hora de morrer.

Replicante Roy Batty
(Ator Rutger Hauer)
no filme Blade Runner.

RESUMO

O conceito de softwares assistentes pessoais se firma na ideia de programas de computador que funcionam tal como um secretário humano, auxiliando as pessoas em suas tarefas diárias. Isso se dá por meio de aplicativos, interação do assistente com o seu usuário, comunicação via Internet e outros recursos. Vários esforços têm sido feitos para a criação desses assistentes. Contudo, foi observado que as propostas existentes atacam pontos isolados, além de que não há nenhuma que seja aberta, de forma a permitir a sua integração aos ambientes empresariais, i.e. a processos de negócios da empresa. Em tempo, uma outra perspectiva é a de que nenhum dos trabalhos avaliados apresenta um modelo ou arquitetura de referência para o desenvolvimento de softwares assistentes pessoais com padrões suficientes para manter a interoperabilidade com outros softwares e processos empresariais. Dessa forma, esta Tese apresenta uma arquitetura de referência aberta para softwares assistentes pessoais, que possa gerar implementações interoperáveis e customizáveis para se adequarem aos processos de negócios da empresa e que possam auxiliar os usuários em suas tarefas diárias. Tal arquitetura é baseada em um modelo de referência e na Arquitetura Orientada a Serviços, selecionada como estilo arquitetural. Com base nos resultados de testes sob uma implementação, desenvolvida baseada na arquitetura de referência, verificou-se que esta instância se comportou conforme o que foi proposto e executou corretamente as ações associadas aos comportamentos dos processos de negócios envolvidos em um exemplo estudado. Este trabalho veio no sentido de oferecer uma contribuição para a área de softwares assistentes pessoais mais flexíveis, mais interoperáveis e mais integrados ao mundo das empresas e processos de negócios.

Palavras-chave: Arquitetura de Referência, Softwares Assistentes Pessoais, Interoperabilidade, Arquitetura Orientada a Serviços.

ABSTRACT

The concept of personal assistant software is established on the idea of computer programs that work as a human secretary, helping people in their daily tasks. This comes via applications, interaction of assistant with its user, Internet communication and other resources. Several efforts have been made for the creation of these assistants. However, it was observed that the existing proposals attacks isolated points, besides that there is no proposal which is open to allow its integration into enterprise environments, i.e. the company's business processes. Another perspective is that none of the studied works presents a model or reference architecture for the development of personal assistant software with sufficient standards to maintain interoperability with other software and business processes. Thus, this thesis presents a reference architecture for personal assistants software which can generate customizable and interoperable implementations to suit the company's business processes and can assist users in their daily tasks. This architecture is based on a reference model and service-oriented architecture, selected as the architectural style. Based on the results of tests on an implementation developed based on the reference architecture, it was found that this instance has behaved according to what was proposed and performed properly the actions associated with the behavior of business processes involved in a case study. This work has come in order to provide a contribution to the area of personal assistant software more flexible, interoperable and integrated into the world of business and business processes.

Keywords: Reference Architecture, Personal Assistant Software, Interoperability, Service Oriented Architecture.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL - *Agent Communication Language*
API - *Application Programming Interface*
ARISA - *Assistant Representative: an Instance using Services Architecture*
BPEL4WS - *Business Process Execution Language for Web Services*
BPM - *Business Process Management*
BPML - *Business Process Modeling Language*
BPMN - *Business Process Modeling Notation*
CAD - *Computer Aided Design*
CALO - *Cognitive Assistant that Learns and Organizes*
CC - Cartão de Crédito
CORBA - *Common Object Request Broker Architecture*
DARPA - *Defense Advanced Research Projects Agency*
DCOM - *Distributed Component Object Model*
DM - *Direct Messages*
DME - Device Manger Entity
DTD - *Document Type Definition*
ECOLEAD - *European Collaborative Networked Organizations Leadership Initiative*
EDI - *Electronic Data Interchange*
FIFO - *First In First Out*
FIPA - *Foundation for Intelligent Physical Agents*
GAP - Gerenciador de Assistentes Pessoais
GNU - *GNU is Not Unix*
GUI - *Graphical User Interface*
HQ - Histórias em Quadrinhos
HTML - *HyperText Marckup Language*
HTTP - *Hypertext Trasnfer Protocol*
IDL - *Interface Definition Language*
IFM - Instituto Fábrica do Milênio
IM - *Instant Messaging*
IMAP - *Internet Message Access Protocol*
ISAP - Interface Social para Assistentes Pessoais
JSON – *JavaScript Object Notation*
KQML - *Knowledge Query and Manipulation Language*
LEM - *Learnable Evolution Model*
LGPL - *GNU Lesser General Public License*

MCT - Ministério da Ciência e Tecnologia
Narval - *Network Assistant Reasoning with a Validating Agent Language*
OASIS - *Advancing Open Standards for the Information Society*
OAuth - *Open Authorization*
ODE - *Orchestration Director Engine*
PAA - *Personal Assistant Agent*
PA - *Personal Assistant*
PAL - *Perceptive Assistant that Learns*
PCM - *Personal Content Manager*
PDE - *Personal Distributed Environment*
PHP - *PHP: Hypertext Preprocessor*
PLN – Processamento de Linguagem Natural
PME - Pequenas e Médias Empresas
POP3 - *Post Office Protocol*
PPGEAS - Pós-Graduação em Engenharia de Automação e Sistemas
RPC - *Remote Procedure Call*
SMTP - *Simple Mail Transfer Protocol*
SOA - *Service Oriented Architecture*
SOAP - *Simple Object Access Protocol*
SOSE - *Service-Oriented System Engineering*
QoS - *Quality of Services*
TIC - Tecnologia da Informação e Comunicação
UBL – *Universal Business Language*
UDDI - *Universal Descripition, Discovery and Integration*
UFSC - Universidade Federal de Santa Catarina
UML - *Unified Modeling Language*
WS - *Web Service*
WS-BPEL - *Web Services Business Process Execution Language*
WSCI - *Web Service Choreography Interface*
WSDL - *Web Services Definition Language*
WSIGS - *Web Service Integration Gateway Service*
XML - *eXtensible Markup Language*
XMPP - *eXtensible Messaging and Presence Protocol*

SUMÁRIO

1 INTRODUÇÃO

Com a crescente evolução das tecnologias de informação e comunicação (TIC) e, concomitantemente, com o aperfeiçoamento das ferramentas computacionais disponíveis para pessoas e organizações, o mercado tem se tornado cada vez mais competitivo. Isso tem obrigado as empresas e organizações a se aperfeiçoarem no âmbito de seus processos, em vários níveis de abstração, conceitos e tarefas (Loss, 2007), (Stojanovic e Dahanayake, 2005).

Não obstante a isso, a necessidade de antecipar novas tendências e responder a elas adequadamente, tais como a implantação e utilização de novos métodos de trabalho, tem se mostrado de vital importância para a sobrevivência, continuidade e ampliação dos negócios das organizações inseridas nesse novo contexto e de, consequentemente, tornar os processos organizacionais mais ágeis em conformidade com a conjectura de cada atividade específica (Lackenby e Seddighi, 2002), (Stojanovic e Dahanayake, 2005).

À luz de questões referentes às atividades organizacionais e seus procedimentos de supervisão, as novas ferramentas disponíveis passam a fornecer um grande suporte à inserção de inovações, que vão ao encontro de respostas mais rápidas e confiáveis aos clientes e fornecedores.

No entanto, a prática tem mostrado que essas mudanças, se por um lado têm facilitado o trabalho das pessoas, por outro as têm forçado a serem mais produtivas, levando a um aumento ainda maior das suas tarefas e responsabilidades (Maes, 1994). Além disso, observa-se empiricamente que elas têm estado cada vez mais ocupadas e imersas em diversos problemas, alguns desses consequentes da própria implantação dessas novas tecnologias, e nem sempre conseguindo cumprir com suas tarefas em tempo hábil e/ou com a qualidade esperada.

Nesse sentido, um aspecto que se deseja focar neste trabalho é o que permeia o tempo das pessoas empregado no seu trabalho e outras formas onde as TICs podem contribuir para ajudá-las. A premissa da consequência disso, no que concerne aos recursos organizacionais, é que as pessoas são o mais nobre dos recursos. Portanto, a sua participação no contexto de tarefas e negócios deveria ser direcionada para a realização de atividades tão essenciais hoje em dia quanto à operação dos processos em si, ou seja, para atividades que agregam maior valor pessoal, interpessoal e para a própria entidade da qual participam.

Dentre estes valores podem se citar a realização de negócios, o exercício da criatividade para a inovação, a aprendizagem e o aperfeiçoamento de suas funções.

Porém, a rotina diária das organizações apresenta uma ruptura nas ideias acima ilustradas. Isso acaba por não condizer com o propósito de um trabalho que proporciona maior valor às pessoas e às organizações inseridas nesta conjuntura. Consequentemente, as pessoas acabam sendo obrigadas a se dedicarem a tarefas repetitivas, desgastantes, pouco atrativas e muitas vezes sem grande valor (Hoyle e Lueg, 1997). Mesmo nas tarefas que agregam maior valor, as pessoas encontram-se tão ocupadas que não conseguem executá-las a tempo e/ou com a qualidade esperada. Por conseguinte, isso tem efeito direto na agilidade e na eficiência das organizações como um todo. Isso se dá principalmente ao se considerar que as organizações funcionam de forma cada vez mais integrada, com suas várias atividades dependentes cada vez mais umas das outras, ou seja, uma tarefa mal feita ou com atraso pode acarretar impactos em toda a cadeia de processos.

Toda essa situação acaba por culminar em uma problemática ainda maior à medida que se observa que as pessoas geralmente estão envolvidas em diversas tarefas simultaneamente, por vezes de diferentes setores de atuação, com terminologias próprias e com seus próprios contextos de execução e de negócio. Sendo assim, as pessoas têm que lidar com cada vez mais tarefas em suas rotinas diárias, assim como também é exigido que executem suas tarefas com crescente eficácia, tendo que entender e saber lidar com as particularidades de cada uma delas (Zambiasi e Rabelo, 2010).

De uma maneira geral, a referida problemática poderia ser amenizada em diversas vertentes organizacionais com a disponibilização de mais pessoas para a crescente quantidade de tarefas, assim como para suprir o aumento da complexidade dessas; ou mesmo de algum tipo de secretário ou ajudante para as pessoas em suas atividades (Bocionek, 1994). Porém, quando levada em consideração a realidade financeira da maior parte das organizações (micro, pequenas e médias), o custo final disso torna esse caminho usualmente inviável.

Mesmo que intrinsecamente alguns cargos empresariais necessitem fornecer aos seus respectivos responsáveis um ou mais secretários, ou assistentes, isso é suficiente apenas para essas pessoas e não supre a necessidade da grande maioria dos funcionários. Portanto, parece justificável o uso de uma forma alternativa de assistência para cada pessoa que necessite de auxilio em algumas de suas tarefas. Tal

assistência poderia ser fornecida, de maneira alternativa, via software. Este poderia munir as pessoas de um conjunto de ferramentas para automatização de suas tarefas e assistência na gestão das atividades relacionadas ao seu trabalho, muitas vezes tal qual um secretário humano (Bocionek, 1994).

A utilização de uma ferramenta computacional para fornecer assistência às pessoas inseridas nas empresas segue em conformidade com a evolução das TICs (Tecnologias de Informação e Comunicação), em que equipamentos cada vez mais potentes, menores, acessíveis e inteligentes permitem que a interação deixe de ser apenas entre parceiros humanos. Tal interação pode passar a ser tratada em negociações entre humanos e computador ou mesmo entre computadores (Vieira, 2000), facilitando a interação entre uma pessoa e um assistente pessoal na forma de um software computacional, por exemplo.

Conforme Markoff (2008) e Bocionek (1994), o conceito de um software com capacidade de antecipar as necessidades dos usuários e executar ações sem a necessidade de interação, tal como agendar viagens, restaurantes, gerenciar agenda e e-mails não é novo e tem influenciado a imaginação de muitos escritores de ficção científica e cientistas da computação, como o exemplo de Oliver G. Selfridge, citado por Markoff (2008), que possui o crédito pelo termo "agente inteligente".

Já para Michaell et al. (1994), pode-se imaginar um futuro com assistentes baseados em conhecimento e que funcionam através da Internet na forma de um tipo de software secretário, provendo serviços tanto para tarefas do trabalho como para casa, assim como pagamentos de contas, organização de viagens, gerenciamento do envio de ordens de compras, localização de informações em bibliotecas virtuais na internet, etc. Tal como secretários humanos, seu sucesso deve depender do quanto este tem conhecimento sobre os hábitos das pessoas que são auxiliadas por esses e do conjunto específico de operações que o software secretário pode executar. Em outras palavras, o software secretário deve possuir um conjunto de informações sobre o usuário e tudo o que envolve sua execução e seu comportamento.

De um ponto de vista organizacional, o conceito de um software assistente pessoal, apresentado por Bocionek (1994) como software secretário, pode ser visto como uma ferramenta potencialmente atraente. Isto pois este fornece uma forma de assistência em alguns níveis de auxílio no cenário organizacional atual. Concomitantemente, este

enfoque pode resolver em parte a questão da crescente quantidade de tarefas de responsabilidade das pessoas inseridas nestas organizações.

1.1 IDENTIFICAÇÃO E RELEVÂNCIA DO PROBLEMA

Durante os últimos anos, têm sido desenvolvidos diversos projetos de softwares com o intuito de fornecer assistência pessoal aos seus usuários. Entre eles, e descritos com mais detalhes no capítulo 2, pode-se citar o projeto PAL (PAL, 2011), (Markoff, 2008), financiado pela DARPA e antigo projeto CALO; o projeto de assistente virtual, estilo recepcionista, da Microsoft chamado Laura (Vance, 2011), (Mundie, 2009); Um projeto da Contemporary Web Plus, Inc. chamado *Virtual Assistant Manager*, com base na web e com a finalidade de gerenciamento de projetos e tarefas dos usuários; O projeto da ShellToys para o gerenciamento de compromissos e tarefas (SHELLTOYS, 2011); O projeto Narval (Chauvat, 2000), (Thenault, 2011), escrito em linguagem de programação Python, com a possibilidade aceitar *plugins* e registrado com licença GNU LGPL; O Siri, atualmente da Apple (SIRI, 2011); O Sandy (Mann, 2011), (SANDY, 2011), (Dornfest, 2011); e *Outlook Personal Assistant*, como *plugin* para o Microsoft Outlook; entre outros.

Apesar desses avanços na área de assistência pessoal, uma primeira perspectiva a se chamar a atenção em função da observação do autor em relação a trabalhos na área, é a utilização ou desenvolvimento de softwares assistentes de forma não integrada ao ambiente e aos processos de negócios (*Business Process*) da empresa. No primeiro aspecto, os assistentes são implementados como sistemas independentes dos demais da empresa, sem interação ou interoperação alguma, fazendo com que o usuário tenha que, explicitamente, navegar por ambientes diferentes e, pior, tenha que redigitar informações e nutrir o assistente de informações contextualizadas. No segundo aspecto, que está intimamente relacionado ao primeiro, o assistente usualmente atua como um sistema puramente reativo à interação do usuário, e não de forma ativa (ou até pró-ativa), consoante a uma lógica de negócios expressa nos processos de negócios da empresa, executando as tarefas de forma integrada com outros sistemas dela.

Apesar de estar associada a TICs, uma segunda perspectiva é a interoperação. Os assistentes pessoais costumam ser softwares fechados[1]. Isso porque foram concebidos para realizarem suas atividades sem uma preocupação, de projeto, de transacionarem com outras aplicações. Na ótica de se integrar o assistente ao ambiente geral de execução de processos da empresa, essa perspectiva é essencial uma vez que usualmente os sistemas dela são distribuídos e heterogêneos.

A terceira perspectiva a se chamar a atenção é quanto a flexibilidade das ações de um assistente. Isso envolve outros aspectos, como adaptabilidade e escalabilidade. No escopo de transações empresariais, cada negócio tem inúmeras particularidades, tanto de contexto como de ambientes computacionais envolvidos. Por exemplo, um assistente deve oferecer ajuda de forma diferente a uma mensagem referente a um processo de venda de produto (a clientes) e a uma de compra de produtos (de fornecedores). E caso o usuário esteja usando um dispositivo móvel para tal, é necessário que as funcionalidades estejam preparadas para exibirem resultados no formato adequado e, se possível, considerando os requisitos de qualidade de serviço (*Quality of Services – QoS*) adequados. Isto envolve, portanto, tanto o lado cliente como o lado servidor.

Portanto, há necessidade de adaptabilidade. Aquelas mesmas transações comerciais não são estáticas. Tanto a empresa como seus sistemas alteram-se ao longo do tempo, ou seja, seus processos de negócios. Dessa forma, há necessidade de se lidar igualmente com a escalabilidade do software assistente, em termos de que deve estar preparado para flexibilizar suas ações consoante a mudanças e/ou introdução de novos processos de negócios. O aspecto final é que as empresas, embora tradicionalmente tenham seus sistemas implantados todos localmente, começam a conviver com ambientes largamente distribuídos, fazendo uso de sistemas externos, disponibilizados por outras empresas que são, principalmente, provedores externos de serviços. Tais provedores são tipicamente formados de *software-houses* e por dispositivos computacionais distribuídos. Nesse sentido, o software assistente deve poder acessar tais serviços consoante os requisitos funcionais associados aos processos de negócios ora em execução, o que pode levar ao caso da necessidade de composição de diferentes serviços para que o comportamento desejado possa ser montado.

[1]Por softwares fechados, no contexto deste trabalho, significa que o código fonte não está disponível e não há especificações para que terceiros possam desenvolver funcionalidades para esses.

Embora sejam analisados vários trabalhos correlatos mais adiante, observa-se que nenhum dos projetos estudados sugere um modelo de referência ou uma arquitetura de referência para o desenvolvimento de softwares assistentes pessoais. Também não foi encontrado nenhum projeto que sugere sequer especificações ou padronizações para interoperabilidade com outras aplicações de forma aberta.

1.2 QUESTÕES DE PESQUISA

Referente aos assuntos abordados, inúmeras questões de pesquisa emergem frente ao que se deseja fazer, como por exemplo:

- Que padrões deve-se utilizar para manter a interoperabilidade do software assistente pessoal com outros sistemas, principalmente os sistemas da empresa? Observa-se que um Software Assistente Pessoal (SAP) é, no contexto deste trabalho, um programa (ou conjunto de programas) de computador que visa fornecer auxílio às pessoas em suas tarefas diárias ligadas a sistemas computacionais.
- Como fazer o software assistente pessoal entender os processos e negócios de empresa e agir de acordo com cada, considerando as particularidades de um negócio?
- Como tornar um software assistente pessoal adaptável às mudanças de interesses e objetivos dos usuários?
- Como tornar um software assistente pessoal flexível de forma que o usuário possa adequar as funcionalidades deste às suas necessidades?
- Como manter a interoperabilidade entre todos os componentes do modelo?
- Quais características uma arquitetura de referência deve ter para manter um padrão aberto de implementação para permitir que cada empresa possa implementar suas próprias versões de assistentes pessoais e ainda assim manter a interoperabilidade com os elementos do modelo?
- É possível modelar o sistema distribuidamente, de forma não monolítica? Que características são necessárias para tal?

Essas questões requerem solução intrinsecamente complexas, que no todo formam o problema global demasiado abrangente para ser

resolvido nesta Tese de doutorado. Assim, o que se propõe é a concepção de uma arquitetura genérica que seja aberta para poder englobar e integrar os vários aspectos e requisitos mencionados, em vários níveis. A estratégia e metodologia de desenvolvimento do trabalho foram definidas tendo isso em consideração.

Neste sentido, tenta-se garantir que o modelo seja aberto e genérico, de forma a poder mais facilmente incorporar contribuições e resultados já existentes de outras fontes/autores. Como pode ser verificado na proposta, tais resultados estão disponibilizados na forma de serviços de software, podendo, portanto, serem acoplados ao modelo com menos dificuldades. Em contrapartida, os esforços devem se concentrar, em termos de desenvolvimento, nos elementos onde os resultados disponíveis não são cobertos ou não estão em num nível de maturidade suficiente para serem acoplados/adequados ao modelo desejado.

Para a definição dos objetivos geral e específicos deste trabalho, esta tese procura responder à seguinte pergunta geral da pesquisa:

> Uma arquitetura de referência para softwares assistentes pessoais pode fornecer aos usuários instâncias implementadas abertas e interoperáveis o suficiente para se adequarem às necessidades deste com relação aos processos de negocio da empresa?

Por **arquitetura de referência** entende-se por um padrão genérico para um projeto, abordando requisitos para o desenvolvimento de soluções e que é guiado por um modelo de referência, de forma a atender as necessidades do projeto (Bass et al., 2003), (McKenzie et al., 2006).

Um **modelo de referência** é um *framework* abstrato para entendimento de como as entidades de um ambiente estão relacionadas, possibilitando o desenvolvimento de arquiteturas específicas e com padrões consistentes ou especificações suportando tal ambiente. Possui também um conjunto mínimo de conceitos unificados, axiomas e relacionamentos com a independência de padrões específicos, tecnologias, implementações, ou outro detalhe concreto (Vernadat, 1996), (Bass et al., 2003), (McKenzie, 2006).

Por **software assistente pessoal** (SAP) entende-se como um processo computacional criado para representar um usuário na execução de certas tarefas, automaticamente ou com algum grau de

intervenção/supervisão humana (Bocionek, 1994), (Hoyle e Lueg, 1997), (Huhns e Singh, 1998).

Por **instâncias implementadas abertas**, no contexto deste trabalho, entende-se por estar baseado em especificações, regras, recomendações ou definições de características declaradas como padrão por órgãos independentes de indústria ou como padrão de facto. Além disso, disponibilizados de forma gratuita para utilização e implementação. Os padrões abertos *"promovem a liberdade de escolha entre produtos e soluções de diferentes fabricantes, [...] existe uma maior interoperabilidade entre produtos de diferentes fabricantes, que implementem os mesmos padrões"* (Piazza, 2007).

Por **interoperabilidade** compreende-se a habilidade de dois ou mais sistemas trocarem múltiplas informações entre si de maneira uniforme e eficiente (IEEE, 1991), (IDABC, 2004), (Ablong et al., 2005).

1.3 OBJETIVOS

De forma a caracterizar o alcance da pesquisa e delimitação dos objetivos de forma explícita, é necessário a descrição dos objetivos gerais como uma descrição genérica e abrangente, e os objetivos específicos na forma de uma descrição mais exata de cada ponto a ser atingido (Cervo e Bervian, 2002), (da Silva, 2005).

1.3.1 Objetivo Geral

Visando responder à pergunta de pesquisa numa perspectiva tecnológica e considerando as suas várias questões de pesquisa, este trabalho tem como objetivo geral conceber uma arquitetura de referência aberta para softwares assistentes pessoais que possa servir de norteadora para implementações interoperáveis, customizáveis e que se adequem aos processos de negócios da empresa.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral do projeto, os seguintes objetivos específicos são definidos:

- Concepção de um modelo de referência para softwares assistentes pessoais que deve servir para guiar a concepção da arquitetura de referência;
- Concepção de uma arquitetura de referência para softwares assistentes pessoais;
- Concepção de uma arquitetura de implementação baseada no modelo de referência e na arquitetura de referência;
- Concepção e implementação de um protótipo computacional associado à arquitetura de referência concebida, para servir de instrumento de avaliação da proposta.

1.4 DELIMITAÇÃO DA PROPOSTA E PRESSUPOSTOS

Com relação à assistência pessoal via software de computador existem diversos subproblemas e requisitos gerais envolvidos. Esses subproblemas são identificados e apresentados a seguir. Contudo, eles são melhor detalhados no decorrer do trabalho:

- **Relevância das Informações:** Este problema está relacionado com quais informações acerca dos usuários são realmente relevantes e como elas podem ser utilizadas na execução da assistência aos usuários (Bush et al., 2006), (Huhns e Singh, 1998), (Schiaffino e Amandi, 2006).
- **Conhecimento estático:** Um assistente pessoal precisa se adaptar às mudanças de interesses e aos objetivos dos usuários, assim como ao ambiente que esses se encontram. Com isso, deve ser possível adequar os comportamentos aos diferentes contextos dos locais em que o usuário transita (Maes, 1994), (Sensoy e Yolum, 2008).
- **Interação Usuário-Assistente Pessoal:** Problema relacionado à quando e como um SAP deve se comunicar e interagir com o usuário (Bocionek, 1994), (Bush et al., 2006), (Franco et al., 2007).

- **Portabilidade:** A interação do usuário com seu assistente pessoal deve ser independente de onde ele esteja localizado e do dispositivo computacional que se está utilizando – Computador *Desktop* ou Dispositivo Móvel (Vieira, 2000), Franco et al. (2007).
- **Disponibilidade:** Com os comportamentos localizados em repositórios distribuídos, esta questão permeia em como este assistente deve proceder no caso de indisponibilidade de comportamentos quando os repositórios de comportamentos não estão acessíveis. Esse não é um problema com solução trivial (Coulouris et al., 2005), (Kshemkalyani e Singhal, 2008), (Bush et al., 2006).
- **Flexibilidade:** É necessário que um assistente pessoal possa se adaptar, em termos de funcionalidades, às necessidades do seu usuário de forma dinâmica e flexível. O problema está em como tornar possível a um usuário de um assistente pessoal reunir um conjunto de comportamentos para se adaptar às suas necessidades e, em certos casos, organizá-los de forma algorítmica para resolver um problema específico (Michaell et al., 1994).
- **Segurança:** A segurança engloba todo o sistema computacional, em vários níveis de abstração. Porém, a ênfase está principalmente no que tange a distribuição dos comportamentos do assistente pessoal na Internet e da comunicação entre os sistemas envolvidos e da utilização e do acesso do usuário ao seu assistente pessoal (Coulouris et al., 2005).
- **Conflitos:** Mesmo que os comportamentos estejam distribuídos, podem existir problemas de conflitos em relação à ordem de execução dos comportamentos no caso desses dependerem de outros para executarem seu processamento (Bocionek, 1994).
- **Interoperabilidade:** Referente à como manter a interoperabilidade de todos os componentes do modelo, tornando possível a utilização de comportamentos, vistos como serviços, já existentes e largamente utilizados na internet como fonte de funcionalidades a serem utilizadas pelos assistentes pessoais (Gesser, 2006), (MacKenzie et al., 2006).
- **Softwares Proprietários:** Muitos dos trabalhos existentes hoje são proprietários ou não se apresentam de forma aberta o

suficiente para que empresas possam desenvolver seus próprios assistentes e mesmo assim manter a interoperabilidade com assistentes de outras empresas. Ainda, sem a possibilidade da utilização explícita de outras funcionalidades (disponíveis em repositórios de terceiros) para compor os comportamentos dos assistentes pessoais (Markoff, 2008), (Vance, 2009), (Mundie, 2008).

- **Softwares Monolíticos:** Além dos itens descritos, os softwares estudados se mostraram na forma de um sistema de processamento centralizado em um único sistema computacional.

Contudo, como já mencionado, não é o propósito deste trabalho atacar todos os subproblemas envolvidos na assistência pessoal via software de computador.

O que se busca nesse trabalho é propor uma forma padronizada de se desenvolver Softwares Assistentes Pessoais, de forma que os comportamentos destes possam ser adaptáveis aos usuários e às suas necessidades, assim como o usuário possa ter seu assistente que atue conforme as tarefas nas quais quer auxílio. Outro ponto é que busca-se a liberdade de escolha dos comportamentos do assistente pessoal, com base numa ampla oferta de comportamentos, que possam ser desenvolvidos pelo usuário, pela empresa, ou por terceiros. Por fim, a busca pela interoperabilidade com outros sistemas, de forma que o assistente pessoal possa se comunicar com outros sistemas para poder cumprir com os objetivos do usuário, tal como efetuar reserva num hotel, fazer uma compra de um produto de forma automatizada, gerenciar a conta de e-mail do usuário, ou a agenda de compromissos, etc. Contando que esses recursos não necessariamente são recursos do próprio assistente, mas recursos disponibilizados por outros e que possam se comunicar com o assistente pessoal.

Em resumo, como ponto norteador tem-se a criação de uma arquitetura que possa servir de referência para a criação de assistentes pessoais que possam se adaptar às necessidades do usuário e que possam manter a interoperabilidade com os sistemas da empresa e outros.

Existem várias abordagens de solução para as várias questões de pesquisa que norteiam o que se pretende desenvolver neste trabalho. Esta seção visa, ainda que de forma preliminar, identificar alguns aspectos que, quer pela complexidade quer pelo foco, não serão cobertos na proposta, ou serão abordados de forma muito limitada.

Devido a importância da comunicação, conforme citado por Bocionek (1994), Bush et al. (2006) e Franco et al. (2007), ser um fator essencial para a interação eficiente entre usuário e software assistente pessoal, o estudo de tal disciplina para a realização do presente trabalho se torna obrigatório. Contudo, nesta proposta busca-se trabalhar sobre protocolos já estabelecidos, conhecidos e com padrões abertos, tais como HTML (*HyperText Marckap Language*), SOAP (*Simple Object Access Protocol*), e outros, sem se ater à comunicação em baixo nível ou especificidades de aplicações legadas de empresa. Além disso, a Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*), seguindo MacKenzie et al. (2006) e Gesser (2006), surge também como forma de manter a interoperabilidade e padronização de comunicação entre os elementos distribuídos dos assistentes pessoais.

SOA é um padrão arquitetural para organizações criarem competências para resolverem problemas específicos conforme suas necessidades. Essas competências são modeladas por meio de um conjunto de componentes que compõem a arquitetura e que podem ser invocados por meio da descrição de suas interfaces, que podem ser publicadas e descobertas. SOA está intrinsecamente ligada ao conceito de serviço (*SaaS – Software as a Service*) (Estefan et al., 2008), (Booth et al., 2004), (Stojanovic e Dahanayake, 2005). Os serviços podem ser implementados com várias tecnologias, mas a mais utilizada por ser de implementação padronizada é a tecnologia de serviços web (Gesser, 2006). A utilização de serviços web e SOA traz vantagens tais como baixo acoplamento, independência de implementação (linguagens de programação, ambientes de programação), configuração flexível, tempo de vida longo (serviços devem existir por tempo suficiente para serem descobertos e utilizados), granularidade (funcionalidades divididas em vários serviços) e distribuição (Singh e Huhns, 2005).

A adaptabilidade do software, conforme Vieira (2000) e Franco et al. (2007), para todos os dispositivos computacionais não é tarefa simples. Por isso, o presente trabalho não intenta abordar tal adaptabilidade para todos os dispositivos móveis ou alguma forma de que este possa ser genérico para todos os dispositivos móveis, mas apenas apresentar, por meio de um protótipo, uma forma de interação com o usuário para que o modelo possa ser validado, tal como a utilização de páginas web para os sistemas que fazem interfaceamento com o usuário, para criação e configuração de seu assistente pessoal. Tal recurso visual está disponível hoje em todos os computadores pessoais via navegadores de internet e nos atuais *smartphones*, com acesso à

internet, tanto via 3G (3ª geração de padrões e tecnologias de telefonia móvel) quanto WiFi (marca registrada da *Wi-Fi Alliance* e utilizada por produtos certificados pertencentes à classe de dispositivos de rede local sem fios – WLAN – baseados no padrão IEEE 802.11).

O aprendizado, conforme Angehrn et al. (2001), Hoyle e Lueg (1997), Schiaffino e Amandi (2006) e CALO (2009) é também um assunto muito abrangente e iria requerer um aprofundamento próprio em tal disciplina. Dessa forma, o aprendizado não é atacado especificamente neste trabalho. Ao invés disso, sugere-se uma forma de ampliação das atividades dos assistentes e adaptação a novas situações via agregação de novos comportamentos aos assistentes por meio de uma estrutura modular e flexível.

Com relação aos conflitos entre os comportamentos dos assistentes, citado por Bocionek (1994), estes podem ser parcialmente resolvidos com a paralelização dessas tarefas, contanto que elas não possuam dependências umas com as outras. Dessa forma, este trabalho procura contornar este problema com tecnologias de computação distribuída sob o paradigma da Arquitetura Orientada a Serviços.

Porém, ainda pode haver problemas para a escolha dos comportamentos a serem selecionados e executados conforme cada situação específica e contexto local do usuário. Tal característica é um requisito necessário para que o assistente possa agir corretamente para alcançar os objetivos dos usuários. Este aspecto é tratado no presente trabalho, sendo que é prevista a definição de um modelo para a gestão dos comportamentos distribuídos já citados.

A segurança também é fator de grande importância, conforme Coulouris et al. (2005). Porém, abordar tal característica estenderia demais os assuntos abordados e a implementação, além de também não ser o enfoque deste trabalho. Dessa forma, tal assunto não é contemplado no modelo proposto, assumindo que isso é resolvido em nível de *firewalls*, antivírus, subsistemas ou funcionalidades de criptografia ou mesmo serviços web.

Em termos gerais, pode-se dizer que o objetivo principal de um assistente pessoal via software de computador é fornecer auxílio aos usuários (Bocionek, 1994), (Hoyle e Lueg, 1997), (Hunhs, 1998), (Schiaffino e Amandi, 2006), (Angehrn et al., 2001), (Maes, 1994), (Sensoy e Yolum, 2008), (Franco et al., 2007), (Bush et al., 2006). Em contrapartida, o objetivo desta proposta é fornecer uma referência para o desenvolvimento desses assistentes, de forma aberta e interoperável. Os comportamentos do assistente devem estar distribuídos na Internet e

devem poder ser acessados na forma de softwares como serviços, no caso específico, padronizado com a utilização de SOA e SOAP. Assim, o auxílio ao usuário especificamente deve depender das funcionalidades dos serviços disponíveis. Dessa forma, o usuário deve customizar seu assistente pessoal para satisfazer suas necessidades e para auxiliar em suas tarefas específicas, com a utilização de serviços também específicos. Contudo, conforme Piazza (2007) os serviços web não são tão padronizados como sugere-se e ainda não há uma gama tão grande de serviços web suficientes para todas as operações necessárias para os usuários de assistentes pessoais que se baseiam nessa proposta. Dessa forma, no caso deste trabalho, uma instância de implementação é apresentada e alguns serviços são implementados especificamente para o estudo de caso de auxílio específico, a ser definido em seção mais a frente.

1.5 ORIGINALIDADE

Esta seção visa enquadrar a proposição de valor desejada, realçando os aspectos considerados inéditos cientificamente quando usados no contexto de softwares assistentes pessoais.

Com base na revisão do estado da arte efetuada (e detalhada no capítulo 2), os seguintes aspectos denotam o ineditismo desta proposta:

- Softwares assistentes pessoais como um ambiente integrado de ações de ajuda ao usuário, ações estas igualmente integradas aos processos de negócios de empresa;
- Uso de padrões abertos em todos os níveis envolvidos (processos de empresa, modelagem de assistentes pessoais, comunicação entre assistentes pessoais), potencializando a interoperação entre todas as entidades do assistente;
- Flexibilidade na composição dos comportamentos do assistente pessoal, fazendo-se uso da Arquitetura Orientada a Serviços (SOA);
- Arquitetura de referência de assistentes pessoais, permitindo que as várias instâncias de implementações de assistentes pessoais, baseadas nesta arquitetura de referência, possam trabalhar de forma integrada e interoperável.

Várias facetas da originalidade pretendida já foram mencionadas ao longo das seções anteriores, assim como serão melhor evidenciadas na revisão do estado da arte no capítulo 2 e da proposta em si.

1.6 ADEQUAÇÃO ÀS LINHAS DE PESQUISA DA PÓS-GRADUAÇÃO

Considerando-se os objetivos deste trabalho, este se enquadra nas linhas do Programa de Pós-Graduação em Engenharia de Automação e Sistemas (PPGEAS) da Universidade Federal de Santa Catarina (UFSC) nas áreas de Automação e Sistemas Mecatrônicos (na área de Organizações Virtuais), e de Sistemas Computacionais (nas áreas de Engenharia de Software e Inteligência Artificial).

1.7 PROJETOS DE AMBIENTAÇÃO

Este trabalho teve início usufruído de um *framework* de referência criado no âmbito do projeto internacional ECOLEAD, e do nacional IFM2, ambos já terminados.

O projeto de pesquisa nacional, chamado IFM – Instituto Fábrica do Milênio – teve como objetivo a formação de um *cluster* de pesquisas integradas para apoiar o desenvolvimento da competitividade da indústria nacional. Ele é uma organização em âmbito nacional, apoiada pelo MCT (Ministério da Ciência e Tecnologia), agregando 600 pesquisadores, em 31 grupos de pesquisas, alocados em 20 Instituições de Ensino Superior. Seu perfil de atuação é focado na pesquisa em manufatura voltada para as necessidades nacionais da indústria.

O projeto de cooperação internacional, chamado ECOLEAD – *European Collaborative Networked Organizations Leadership Initiative* – é um Projeto Integrado do 6º Programa Quadro da Comissão Europeia, iniciado em abril de 2004 com duração de 48 meses, que contou com a participação de 20 organizações distribuídas em 14 países, sendo 18 instituições Europeias e 2 da América Latina. O projeto visou criar fundamentos teóricos e mecanismos de tecnologia da informação para auxiliar no estabelecimento de uma avançada sociedade colaborativa entre organizações. A ideia principal é causar impacto substancial na

forma de materialização de redes colaborativas através de uma abordagem holística compreensível.

Nesses projetos, uma infraestrutura de TIC para o suporte de redes colaborativas foi desenvolvida, e uma das funcionalidades planejadas nesta era de softwares assistentes. Assim sendo, conceitualmente, o resultado desta Tese é representado como sendo um serviço de software, considerando-se que aquela infraestrutura foi desenvolvida para ser escalável e seus desenvolvimentos incrementais.

1.8 ELEMENTOS DE PESQUISA

Com base no problema abordado neste trabalho, é necessária a realização de uma pesquisa bibliográfica das áreas relativas à assistência pessoal via software, considerando os vários aspectos relacionados às questões de pesquisa explicitadas na seção 1.2:

1. Área de Softwares Assistentes Pessoais, visando refinar a visão apresentada a partir da identificação dos requisitos para execução de tarefas para pessoas sem ou com alguma intervenção humana.
2. Tecnologias, baseadas em padrões abertos, necessárias para a implementação da arquitetura a ser definida.

Para a realização da pesquisa bibliográfica, os seguintes sites de indexação de artigos científicos foram utilizados para pesquisa de material científico em formato digital:

- Cite Seer – *http://citeseer.ist.psu.edu/*
- IEEE Xplore – *http://ieeexplore.ieee.org/*
- Google Acadêmico – *http://scholar.google.com/*
- Portal Capes – *http://www.periodicos.capes.gov.br/*
- Science Direct – *http://sciencedirect.com/*

Além disso, estão disponíveis no GSIGMA livros, artigos técnicos e científicos, normas e outros materiais para fundamentar o projeto de forma apropriada.

1.9 ENQUADRAMENTO METODOLÓGICO CIENTÍFICO

De modo a fornecer uma base para definir os fundamentos para os estudos científicos do presente trabalho, é necessário enquadrá-lo em conformidade com a pesquisa e de acordo com sua classificação e abordagem do problema. A metodologia é o conjunto de processos empregados na investigação e o método é a ordem em que os processos necessários para alcançar os objetivos e resultados desejados precisam estar (Cervo e Bervian, 2002). Conforme a classificação metodológica, este trabalho se enquadra da seguinte maneira:

Quanto ao Método de Pesquisa: Hipotético-dedutivo. Através da constante busca pela comprovação da hipótese acerca da concepção de um padrão de desenvolvimento de softwares assistentes pessoais trabalhando junto aos processos de negócios da empresa, construir-se-ão deduções lógicas sobre a sua utilização, dado que não existem conhecimentos suficientes sobre esse fenômeno/impacto de softwares assistentes na ajuda ao trabalho das pessoas.

Quanto à Abordagem de Pesquisa: Preponderantemente qualitativa. A construção de uma arquitetura de referência para assistentes pessoais é tangível, mas desenvolvida a partir de uma visão/abordagem subjetiva que não pode ser plenamente traduzida em números, mas sim através de modelos. Assim, a pesquisa deve se dar qualitativamente em relação à produção de um protótipo, ou melhor, uma instância de implementação que segue tal arquitetura de referência, e que funciona em conformidade com que o sistema foi proposto.

Quanto ao Paradigma de Pesquisa: Positivista, pois é possível a detecção de uma realidade dentro da qual o problema está inserido. O problema é tangível, palpável, com algumas possibilidades de predição e controle. O pesquisador e o objeto de estudo estão bem delimitados.

Quanto à Natureza da Pesquisa: Aplicada, pois seu objetivo principal tem aplicação prática definida: a modelagem e definição da Arquitetura de Referência para Softwares Assistentes Pessoais.

Quanto aos Objetivos da Pesquisa: Exploratória, já que ela consiste basicamente em uma pesquisa bibliográfica seguida da proposta de uma arquitetura, sua implementação e avaliação.

Quanto aos Procedimentos de Pesquisa: Pesquisa Bibliográfica, elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e conferências e normas de padronização de tecnologias.

Quanto ao Tempo da Pesquisa: Estudo longitudinal, em que as informações para o desenvolvimento do modelo são coletadas em vários momentos, durante toda a fase do trabalho.

Quanto ao Tipo de Observação na Pesquisa: Sistemático, com observações feitas de forma controlada para responder aos propósitos previamente estabelecidos; e em testes sob o protótipo, feitos por mais de uma pessoa, para que o mesmo seja avaliado através da experiência dessas pessoas com a utilização da tecnologia implementada.

Este enquadramento norteou o trabalho durante todo o período de pesquisa, modelagem, implementação, testes e análise para, por fim, efetuar a avaliação da proposta sob o tema de pesquisa.

1.10 ETAPAS DE EXECUÇÃO DO TRABALHO

Esta seção descreve a análise e direcionamento dos problemas, os principais assuntos selecionados para a realização da pesquisa e a apresentação das etapas do trabalho.

De forma sucinta, a metodologia para a execução do trabalho de pesquisa teve como base as seguintes etapas:

1. Escolha e delimitação do tema, que é essencialmente o desenvolvimento de uma Arquitetura de Referência para Softwares Assistentes Pessoais.
2. Revisão bibliográfica da literatura relacionada aos assuntos abordados, definição do problema e objetivos do trabalho. A aquisição de informações foi feita por meio da leitura de livros, artigos técnicos e científicos, normas e outros materiais que pudessem melhor fundamentar este trabalho de forma apropriada. As informações adquiridas nessa etapa foram sumarizadas e incluídas no capítulo de revisão bibliográfica.
3. Definição e desenvolvimento da arquitetura de referência. De acordo com os objetivos específicos, esta etapa foi dividida em quatro passos:
    - Especificação de um Modelo de Referência para nortear o próximo passo;
    - Especificação da Arquitetura de Referência para Softwares Assistentes Pessoais;
    - Implementação da arquitetura. Esta atividade serve a dois propósitos: (i) complementar o estudo feito na

revisão bibliográfica para refinamento dos assuntos vistos de forma preliminar, tais como tecnologias para implementação e análise de ferramentas; e (ii) formar a base necessária, a partir de soluções existentes, para trabalhar nos pontos que demanda ineditismo.

4. Avaliação da proposta. Esta ocorreu em três etapas:
    - Implementação de um protótipo desenvolvido em conformidade com a arquitetura, com base em um estudo de caso e utilizando um ambiente simulado e controlado. Nessa etapa o protótipo foi testado e foi verificada sua execução e se esta se comporta como esperado.
    - Apresentação da proposta, apresentação da execução do protótipo e aplicação de questionários para um conjunto de especialistas.
    - Escrita de artigos científicos para divulgação dos resultados e avaliação perante a comunidade científica.
5. Escrita do documento final da tese, com base nos resultados obtidos.

É importante salientar que a escrita de artigos científicos se deu durante todo o período de doutorado, apresentando à comunidade científica os resultados do trabalho ao longo do seu desenvolvimento, assim servindo como forma de discussão e percepção de sugestões de melhorias na proposta.

A seguir, a Figura 1 é apresentada como uma maneira de ilustrar a etapa 3 acima citada.

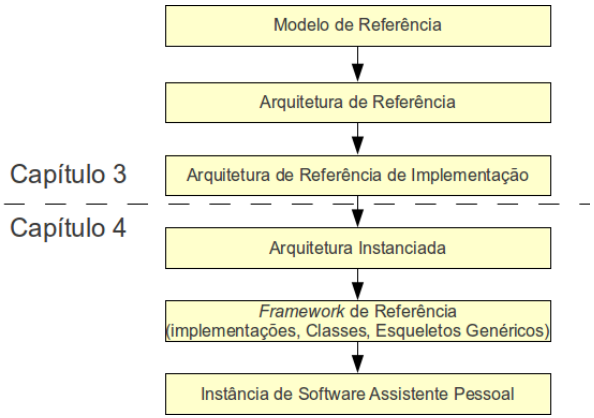

Figura 1: Estrutura do desenvolvimento da proposta.

No capítulo 3 são abordados os três primeiros níveis do desenvolvimento da proposta:

- **Modelo de Referência:** Apresentação de um *framework* abstrato contendo um conjunto de conceitos e relacionamento entre os mesmos. Essa etapa serve como ponto norteador das próximas etapas. A pesquisa do estado da arte e de requisitos apresentados por outros autores e projetos no mesmo contexto servem como base para a concepção desse modelo de referência.
- **Arquitetura de Referência:** Parte central da proposta, composto por um padrão genérico para o projeto. Nesta parte são apresentados requisitos para o desenvolvimento de Softwares Assistentes pessoais.
- **Arquitetura de Referência de Implementação:** Na forma de um modelo genérico de implementação, é apresentada uma estrutura em três camadas (camada de apresentação, camada de processos e camada de dados) como forma de guiar a passagem da Arquitetura de Referência apresentada para uma estrutura já de implementação e elementos de softwares.

No capítulo 4, por sua vez, os três níveis subsequentes são apresentados:

- **Arquitetura Instanciada:** Agregando a arquitetura de referência e a arquitetura de referência de implementação, neste ponto é efetuada uma instanciação de um caso em particular para implementação de um protótipo para avaliação da proposta.
- ***Framework* de Referência:** A partir do exemplo de caso escolhido para instanciação da proposta, são então apresentados o conjunto de classes, estrutura de *deployment*, esqueletos genéricos, bases de dados e demais elementos para a implementação de um protótipo.
- **Instância de Software Assistente Pessoal:** Por fim, o *Framework* de Referência apresentado na etapa anterior é então implementado com componentes de software, sistemas gerenciadores de banco de dados e linguagens de programação.

1.11 ORGANIZAÇÃO DO DOCUMENTO

A organização desse documento tenta refletir o objetivo essencial de uma Tese de Doutorado, que é o de identificar um problema relevante e o ponto de ineditismo, mostrar o estado em que a ciência encontra-se nesse quesito, introduzir o que se pretende atingir para se resolver o problema (ou contribuir para) e com qual visão metodológica, e descrever o como (metodologia e proposta) se pretende atingir o resultado desejado, apontando-se os pontos críticos, recursos e cronograma.

Assim sendo, este documento está dividido em 7 capítulos. O capítulo 1 apresenta o tema de pesquisa, sua delimitação, o problema a ser abordado, os objetivos do trabalho, e a metodologia a ser adotada.

O capítulo 2 traz a revisão bibliográfica referente às áreas envolvidas no problema abordado para a concepção da arquitetura e tecnologias da computação para a implementação do protótipo, assim como uma revisão do estado da arte na área de softwares assistentes pessoais.

O capítulo 3 detalha a proposta deste trabalho, apresentando um Modelo de Referência para Assistentes Pessoais e destacando a Arquitetura de Referência para Softwares Assistentes Pessoais, assim como a conexão dessa arquitetura com o Estilo Arquitetural escolhido.

No capítulo 4, é apresentada uma Arquitetura Concreta baseada na Arquitetura de Referência e é detalhada a implementação de uma instância baseada na proposta de Tese.

No capítulo 5 é apresentada a verificação e avaliação da proposta por meio de testes no estudo de caso implementado e aspectos para a avaliação da proposta, apresentando a forma como os objetivos foram alcançados e resultados da aplicação de questionários à pessoas que foram apresentadas à proposta.

O capítulo 6 apresenta as considerações finais do que foi alcançado e sugestões para trabalhos futuros.

Finalmente, no capítulo 7 são apresentadas as referências bibliográficas utilizadas e os apêndices em sequência.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo tem por objetivo apresentar o estado da arte em relação aos softwares assistentes pessoais. É importante salientar que os autores pesquisados, que abordam em seus trabalhos softwares de computador que fornecem auxilio aos usuários em suas atividades, podem uma nomenclatura diferente ou própria para definir tais softwares. Dessa forma, por escolha do autor desta Tese, neste capítulo as definições de cada um dos autores referenciados são mantidas.

Também, neste capítulo é apresentada uma revisão bibliográfica das tecnologias da informação e comunicação utilizadas para alcançar os objetivos específicos e objetivo geral do presente trabalho. É de se observar que as tecnologias escolhidas e utilizadas se enquadram em certos requisitos, já citados no capítulo anterior e outros apresentados no decorrer do trabalho.

Isso condiz com o propósito de colocar o leitor deste documento a par das bases teóricas que suportam a proposta de "Uma Arquitetura de Referência para Softwares Assistentes Pessoais" e das tecnologias que comportam a instância implementada baseada na proposta e que serve como um dos elementos para, enfim, fazer a avaliação da proposta.

2.1 SOFTWARES ASSISTENTES PESSOAIS

O conceito de um software que fornece assistência aos usuários não é novo e já vem sendo trabalhado em diversas frentes. Estes softwares objetivam auxiliar ou substituir seus usuários em certas tarefas, liberando-os para atividades mais importantes e, consequentemente, melhorando o desempenho dos mesmos perante a empresa (Bocionek, 1994).

Neste sentido, Bocionek (1994) apresenta os softwares secretários como sistemas de automação de escritórios. Ele caracteriza esses softwares como sistemas autônomos que têm a finalidade de fornecer assistência no gerenciamento do trabalho e das atividades dos seus usuários, assim como secretários humanos fariam. Este conceito começa a se mostrar cada vez mais mais atraente conforme observa-se o contexto atual das empresas. Nestas, o número de funcionários que pode

ter disponível para si um assistente, ou secretário, humano é bastante limitado.

No entanto, para que os softwares secretários possam se comportar como humanos, eles precisam ter a habilidade de negociar e de aprender. Isso se deve ao fato de que a maioria das tarefas de escritório envolve negociação. Além disso, ele deve auxiliar seus usuários em atividades de assistência em agendamentos de compromissos, seleção de correspondências importantes, processamento de ordens de serviço e outras. Não obstante, características como portabilidade e mobilidade também são requeridas, principalmente num cenário computacional de dispositivos móveis. Dessa forma, o usuário pode levar seu software secretário a qualquer lugar, servindo-lhe de auxílio não apenas no escritório, mas onde quer que o usuário vá Bocionek, 1994).

Os softwares secretários devem saber gerenciar as atividades conflitantes do usuário, efetuando negociações entre elas e, em último caso, solicitar a interferência do usuário. Ele deve gerenciar os e-mails do usuário, verificando e selecionando os que são referentes a compromissos, por exemplo, analisando e compondo os agendamentos necessários em conformidade com a data, horário, duração e local do compromisso. No entanto, essas mensagens de e-mails devem seguir uma forma padronizada de especificação para facilitar a interpretação do software secretário. Além disso, o grau de importância dos e-mails também pode ser verificado, modificando suas prioridades para que o usuário possa identificá-los com mais facilidade (Bocionek, 1994).

Hoyle e Lueg (1997) comenta em seu artigo sobre os softwares que visam auxiliar os usuários em atividades rotineiras, repetitivas, desinteressantes e que consomem muito tempo. Ele chama esses softwares de *softbots* e cita que a criação desses programas, supostamente inteligentes, é uma tendência natural para a automatização de tarefas e para a melhoria da vida e do trabalho das pessoas. O objetivo destes é fornecer assistência as pessoas, deixando-as mais livres como forma de aumentar a produtividade e a criatividade das mesmas. Pode-se dizer que a visão de Hoyle e Lueg sobre *softbots* são condizentes com a visão de SAP, no contexto desse trabalho.

Algumas atividades que podem se beneficiar da utilização dos *softbots* são, por exemplo, o gerenciamento de e-mails e o agendamento de tarefas, pois essas podem ser classificadas como repetitivas quando executadas pelas pessoas. O software pode se ocupar da exclusão de e-mails não importantes, do gerenciamento da agenda e negociação dos

compromissos conforme as preferências do usuário. Ele também pode manter uma comunicação com o servidor de e-mails para enviar e-mails e negociar automaticamente encontros e compromissos, enviando mensagens para lembrar os integrantes de uma reunião, filtrar os e-mails, etc. (Hoyle e Lueg, 1997).

As ideias e premissas que permeiam os softwares para assistência pessoal, em certos momentos, confundem-se com a tecnologia de agentes de software. Inclusive, alguns autores já se utilizam dos conceitos de agentes para suportar as teorias para a criação de softwares de assistência pessoal. Com isso, o presente trabalho se apropria de certos conceitos de agentes para apresentar premissas, requisitos e características dos softwares assistentes pessoais. Entretanto, para a apresentação da proposta de Arquitetura de Referência para Softwares Assistentes Pessoais, o autor não enquadra o assistente pessoal como um agente. Utilizar tal abordagem é de escolha do desenvolvedor no momento de modelar e implementar instâncias específicas de cada elemento da arquitetura proposta.

De forma bastante simplificada, os agentes se caracterizam como algo que, provido de sensores, recebe informações do ambiente ao seu redor e toma ações, interagindo com este ambiente por meio de mecanismos denominados de atuadores. Além disso, os agentes devem poder se comunicar com outros para poderem alcançar seus objetivos (Russel e Norvig, 2004). Seguindo este conceito, os assistentes pessoais, tal qual um agente, podem perceber seu ambiente através de recursos (como a agenda do usuário, preferências, preferências de outros usuários, reputações) e respondem com ações e assistência ao usuário.

Além disso, um agente de software pode substituir parcial ou totalmente uma pessoa em determinadas tarefas e em várias situações diferentes, assumindo o papel de um assistente. Para que o agente possa executar suas tarefas, muitas vezes ele precisa se comunicar com outros sistemas, ou outros agentes. Estes agentes podem trocar informações automaticamente, sem a intervenção direta ou indireta de um humano, por meio de negociações eletrônicas (Weiss, 1999).

Wong e Mikler (1999) afirmam que um certo nível de autonomia permite que um software possa agir sem a interferência do seu usuário para poder completar suas tarefas, fazendo as escolhas apropriadas a partir de um conjunto de estratégias possíveis. Ele apresenta essa premissa ao abordar a tecnologia de agentes.

Agentes, em ambientes virtuais, podem representar seus usuários de forma a ajudá-los a completar suas tarefas, tal como assistentes

pessoais devem funcionar. Estes agentes, quando em comunidades virtuais, devem se comunicar e cooperar com outros para alcançar seus objetivos. Isso faz emergir naturalmente comunidades de agentes com interesses parecidos. Além disso, os agentes devem refletir a mesma dinâmica que um usuário apresenta na vida real, ou seja, assim como os interesses do usuário mudam com o tempo, os agentes também devem ajustar seus objetivos para corresponder com seus usuários (Sensoy e Yolum, 2008).

Para Huhns e Singh (1998), um software assistente pessoal não pode ser visto apenas como um programa de computador personalizado. Isto porque um assistente pessoal deve ser baseado em rede, ser interativo, adaptativo, de propósito geral, de execução autônoma e que possa interagir com outros assistentes pessoais. Com a apropriação dessas especificidades, Huhns e Singh (1998) fazem um paralelo dos softwares assistentes pessoais com a tecnologia de softwares agentes. Dessa forma, eles apresentam os assistentes pessoais como softwares que podem representar as pessoas na Internet, auxiliando os usuários nas atividades do dia a dia, especialmente nas que envolvem recuperação de informação, negociação ou coordenação. Para eles, estes agentes devem ser interativos, adaptativos, de propósito geral e devem ser softwares autônomos que permanecem em execução.

Os softwares assistentes pessoais devem, e podem, lidar com vários tipos de negociações. Estes devem representar o usuário em diversas situações, devem poder fazer o gerenciamento da agenda para, por exemplo, agendar encontros e, baseado na localização, encontrar o serviço de babás mais próximo. Além disso, deve se comunicar com outros agentes para satisfazer os objetivos e necessidades do usuário e cumprir com as tarefas que lhe cabem. Os assistentes pessoais também devem poder ajudar a comprar e vender, negociar bons termos de serviços, concluir negócios, ajustar entregas, serviços e suportes. Eles podem gerenciar e filtrar chamadas de telefones e e-mails não importantes (Huhns e Singh, 1998).

Para Huhns e Singh (1998), apresentar os softwares assistentes pessoais sob a tecnologia de agentes é um caminho bastante atraente. Para somar uma abordagem mais ampla a isso, eles citam também a existência de tecnologias bastante interessantes, tais como os serviços Web, que podem servir como uma ótima fonte de investigação para se agregar à tecnologia dos assistentes e agentes. Essa tecnologia, além de possuir características parecidas com as dos agentes, possui flexibilidade

e capacidade de trabalhar em conjunto em um ambiente interoperável de empresas virtuais (Huhns, 2002).

Huhns e Singh (1998) apresentam um cenário de assistentes pessoais cujo objetivo é representar indivíduos na Web. Estes assistentes ajudam os usuários em atividades diárias como o gerenciamento de e-mails, de chamadas de celulares, e podem coordenar e negociar com certo grau de autonomia, tendo como base informações predefinidas pelo usuário. Para isso, os agentes precisam de um conjunto de informações do usuário, armazenado numa forma de perfil, e que precisa possuir certas características para estar apto a interagir e negociar, tais como:

- **Heterogeneidade** para interagir com assistentes de diferentes desenvolvedores;
- **Autonomia** para se comportar conforme os desejos dos seus usuários;
- **Idiossincrasia** para manter detalhes relevantes de usuários e classes de usuários;
- **Múltiplas perspectivas** para manter as características individuais de outros agentes para cada usuário;
- **Compartilhamento** como forma de disseminar conhecimento e opiniões dos usuários.

Além dessas características existem algumas especificidades que são fatores importantes para a realização de negócios e acordos, assim como a reputação do agente, por exemplo, que pode classificá-lo num conjunto de agentes para determinado negócio. Assim como as pessoas estão envolvidas em diferentes estruturas organizacionais e com diferentes regras de negócios, o assistente pessoal também deve se adaptar a diferentes regras e ainda, em conformidade com cada situação, localização geográfica, hora local, urgência, conveniência e disponibilidade. Em tempo, o agente precisa estar a par da situação, monitorando os dispositivos dos usuários para que possa adaptar sua atuação a cada diferente situação, tal como no computador ou em trânsito com um dispositivo móvel (Huhns e Singh, 1998).

Em suma, existem diferentes aspectos de significados e processamentos a serem invocados. Para esses diferentes tipos de contextos deve haver também diferentes tipos de comportamentos, níveis e flexibilidades (Huhns e Singh, 1998).

O conceito proposto por Huhns e Singh (1998) é um ponto de partida bastante interessante para as abordagens e objetivos propostos no

presente trabalho. Entretanto, não é apresentando qualquer modelo ou proposta de implementação por eles.

Angehrn et al. (2001) descreve um sistema baseado em agentes para auxiliar as pessoas nas organizações a aprenderem a adotar melhores práticas de compartilhamento de conhecimento. Isso se deve ao fato das empresas necessitarem de um maior compartilhamento de conhecimento para suportar o aumento da competitividade de mercado.

O agente descrito no artigo de Argehrn (2001) é tido como um assistente pessoal que analisa continuamente as ações dos usuários de forma a construir e manter um "padrão de comportamento" refletindo o nível de adoção do comportamento desejável. O agente, por sua vez, cria um guia customizado com motivação e estímulo, suportando a gradual transformação do comportamento do usuário.

Na proposta de Angehrn et al. (2001), o agente mantém um modelo de informações do usuário para adaptar suas ações, seus níveis específicos de adoção do usuário e suas preferências, possuindo três partes principais: o estado de adoção do usuário ao comportamento desejável, uma agenda com atividades propostas para o usuário do agente e suas preferências. O processo de execução se dá por meio de um relatório de ações do usuário nas interações com o agente. Com esse recurso, o agente executa um diagnóstico de como o conhecimento deve ser gerenciado e atribui um conjunto de comportamentos para o usuário. Por fim, este trabalho basicamente apresenta uma possibilidade de criação de ferramentas de aprendizado que oferecem um suporte ao processo de aprendizagem, e cobrindo um ciclo de aquisição e compartilhamento de conhecimento (Angehrn et al., 2001).

Já Carrillo-Ramos et al. (2005), apresenta um *framework* baseado em sistemas multiagentes que provê adaptações das informações dos usuários de acordo com as preferências e o histórico do sistema e em concordância com as capacidades limitadas de seus dispositivos móveis. Os autores descrevem um gerenciador de conhecimento, trocas de informações e adaptações para suportar certas capacidades de adaptação.

Esse cenário atual, de computação distribuída e com dispositivos móveis cada vez mais aperfeiçoados, trouxe um aumento significativo na complexidade dos ambientes de computação, aumento das redes e protocolos. No mesmo sentido, cada vez mais os usuários tem estado inseridos em um ambiente pessoal distribuído, com uma grande quantidade de informações altamente dinâmicas. Dentro nesse contexto, Bush et al. (2006) comenta que os assistentes pessoais, na forma de agentes, precisam interagir e reagir com estes ambientes e manter a

compatibilidade de todos os tipos possíveis de conteúdos espalhados neste meio distribuído (Bush et al., 2006).

Esta grande quantidade de informações dinâmicas, e que podem mudar conforme o local e o tempo, advêm dos mais diversos tipos de dispositivos, tanto dos ambientes quanto dos dispositivos móveis das outras pessoas. Isso acaba ocasionando para os usuários dos dispositivos móveis muitas distrações. Bush et al. (2006) apresenta um Agente Assistente Pessoal (PAA – *Personal Assistant Agent*) que pode monitorar e gerenciar as interfaces entre o usuário e o mundo e um Gerenciador de Conteúdo Pessoal (PCM - *Personal Content Manager*) (Figura 2) para administrar os mecanismos e ferramentas para fornecer ao usuário o melhor acesso ao conteúdo dos seus dispositivos (Bush et al., 2006).

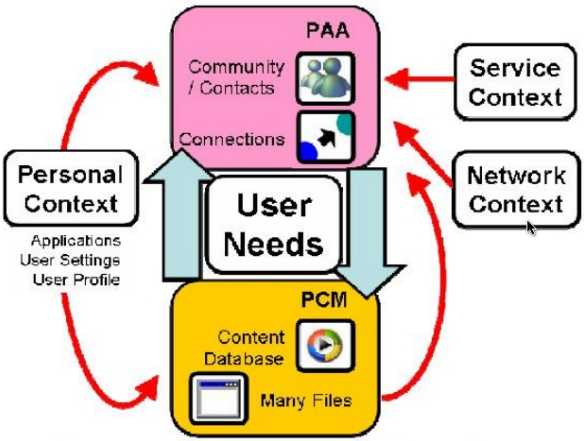

Figura 2: Interações do Assistente Pessoal de Bush et al. (2006).

A ideia de um Ambiente Pessoal Distribuído (PDE – *Personal Distributed Environment*), da Figura 2, é um conceito, ou arquitetura, em que o usuário possui todo o seu ambiente distribuído em seus dispositivos espalhados no ambiente como computador, PDA, computador do carro, dispositivos dos eletrodomésticos, informações provindas das redes ubíquas e outros. Todo este ambiente distribuído precisa ser gerenciado de forma eficiente. O Agente Assistente Pessoal precisa não apenas utilizar as informações provindas deste ambiente, mas também de um conjunto de informações e preferências de cada usuário, assim como precisa coordenar com a capacidade de funcionamento dos dispositivos como tempo de duração da bateria, capacidades de vídeos, processamento, armazenamento local, entre

outros (Bush et al., 2006). Sete planos de contextos são identificados para esse assistente pessoal:

- O **Contexto Pessoal** possui as preferências, tipos de comportamentos, histórico pessoal das aplicações utilizadas pelo usuário e características do dispositivo.
- O **Contexto Físico**, ou de Rede, engloba o contexto da representação da rede e das características de conectividade tal como *wireless,* qualidade do sinal, etc.
- O **Contexto de Serviço** avalia a disponibilidade de aplicações e serviços remotos.
- O **Contexto Social** define os diferentes tipos de regras ou tarefas a serem executadas pelo usuário em conformidade com cada contato social diferente do usuário com cada meio.
- O **Contexto de Ambiente** que gerencia as informações sobre temperatura, tempo e previsão do tempo.
- O **Contexto da Aplicação** que avalia a compatibilidade das aplicações a serem executadas pelo assistente pessoal.
- O **Contexto do Dispositivo** que avalia as capacidades e topologia.

O agente, em execução no dispositivo do usuário, precisa estar ciente do contexto local em que este se encontra. Assim, a complexidade da interação com cada outro diferente usuário e pessoas interagindo no ambiente pode ser tomada como fator crítico para a execução de diversas tarefas pelo assistente pessoal. Ele pode negociar com as redes, adquirindo informações contextuais, apresentando uma interface simples para o usuário baseada nas informações contextuais complexas e agindo como um ponto de contato com o mundo (Bush et al., 2006).

A Figura 3 apresenta a arquitetura dos assistentes pessoais definidos em um Dispositivo Gerenciador de Entidades (DME – *Device Manger Entity*) de Bush et al. (2006). Esta representação é composta de um DME raiz e um DME local com PPAs distribuídos e PCMs operacionais com funções DMEs. O Núcleo do DME representa os registros, previamente identificados, que devem ser utilizados para que as entidades pessoais possam tomar suas decisões.

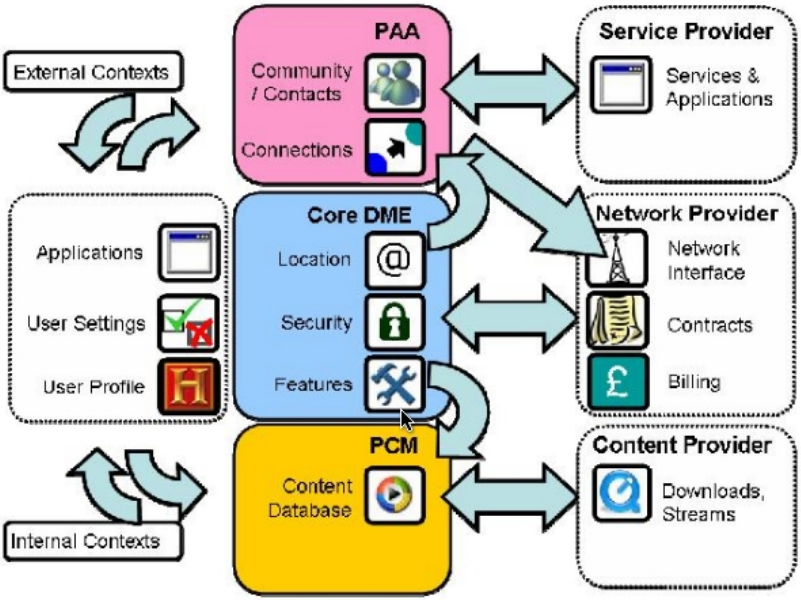

Figura 3: Arquitetura do Agente Assistente Pessoal de Bush et al. (2006).

Schiaffino e Amandi (2006) também apresentam os assistentes pessoais como softwares na forma de agentes, chamando-os de agentes pessoais. Para elas, estes agentes têm sido, e continuarão sendo, utilizados em vários domínios de aplicação, incluindo na tarefa de fornecer assistência as pessoas em suas atividades diárias. Esses assistentes pessoais devem aprender sobre os interesses dos usuários, preferências e hábitos dando uma assistência pró-ativa e personalizada, assim como agentes são análogos a assistentes pessoais no ambiente de trabalho no mundo real.

Schiaffino e Amandi (2006) se atém a um conjunto de fatores para caracterizar um contexto do usuário e uma estrutura para definição de interação do usuário com seu assistente pessoal em situações específicas.

Conforme Schiaffino e Amandi (2006), os agentes pessoais têm sido utilizados em diversos domínios de aplicações, e sua utilização trouxe consigo novos problemas. Entre esses, pode-se citar o tratamento dos objetivos dos usuários, necessidades, considerações inadequadas sobre suas ações, custos e benefícios com ações mal organizadas temporalmente e com falhas para otimizar oportunidades aos usuários. Para elas, os trabalhos de agentes pessoais atuais têm concentrado seus

esforços a obter e gerenciar as preferências dos usuários em relação às aplicações computacionais. Porém, ainda não trabalham de forma eficiente na interação dos agentes com os usuários.

Para uma eficiente forma de interação entre os usuários, é importante haver um tipo de assistência para cada diferente contexto, o agente deve saber quando interromper ou não o usuário, o tipo de assistência que o agente deve fornecer, a tolerância a erros que o usuário permite e o quanto de controle o usuário deve delegar ao agente. Além disso, os agentes devem ser simples, utilizáveis e devem ter a capacidade de dar o devido retorno aos usuários (Schiaffino e Amandi, 2006).

Sendo assim, o agente pessoal precisa descobrir que tipo de assistência o usuário necessita em cada diferente contexto e a forma de se promover essa assistência. Para isso, os agentes precisam poder observar e entender o comportamento do usuário durante as interações. No entanto, isso não é uma tarefa fácil por causa da complexidade do comportamento humano. Somando a isso, os agentes precisam estar cientes do contexto em que o usuário está inserido antes de fornecer assistência, mas o problema nesse caso é que o ambiente se encontra na maioria das vezes em constante mudança (Schiaffino e Amandi, 2006).

Um agente pessoal deve detectar a tarefa atual do usuário e sua prioridade, deve conhecer a agenda do usuário, os interesses, objetivos, preferências, hábitos, contatos pessoais, o estado emocional e deve detectar quando esses fatores mudam (Schiaffino e Amandi, 2006).

Schiaffino e Amandi (2006) analisam, em seu trabalho, diversos usuários com diferentes preferências em relação a alertas, sugestões e ações que devem ser executadas pelo agente pessoal. O agente precisa descobrir quando interromper o usuário e em que contexto ou situação. O principal problema é detectar fatores de contextos específicos dos comportamentos dos usuários e em conformidade com as diferentes preferências dos usuários com relação às ações das assistências dos agentes e modalidades.

Para resolver alguns dos problemas citados por Schiaffino e Amandi (2006), elas apresentam uma estrutura na forma de um perfil de usuário, contendo informações adequadas para os usuários conforme o contexto específico. Esses perfis são personalizados para cada usuário. O perfil é separado em duas partes: a parte que contém as informações pessoais sobre o usuário (*Application-independent*) e a parte contendo os interesses dos usuários, preferências, objetivos, hábitos de trabalho, padrões de comportamentos, conhecimento, necessidades, prioridades e

compromissos aplicados a um domínio em particular (*Application-dependent*).

Mesmo sendo definidas diversas informações, Schiaffino e Amandi (2006) afirmam que elas ainda não são suficientes para personalizar a interação com os usuários. Dessa forma são incluídas algumas informações sobre a situação ou contexto em que o usuário:

- Requer uma sugestão para lidar com o problema;
- Necessita de apenas um alerta sobre o problema;
- Aceita uma interrupção de um agente;
- Espera uma ação sobre o seu comportamento; ou
- Deseja uma notificação ao invés de uma interrupção.

Além disso, Schiaffino e Amandi (2006) definem um indicativo que informa ao agente um grau de certeza em relação a uma assistência ao usuário em uma situação em particular.

Em tempo, devem haver informações suficientes para que os agentes possam aprender com a interação com o usuário, tais como informações sobre as ações que um usuário requisita do agente, as ações de assistência que o agente realiza e o retorno do usuário para cada ação de assistência. Estas informações também devem ser armazenadas no perfil do usuário (Schiaffino e Amandi, 2006).

Schiaffino e Amandi (2006) ilustram um usuário sendo auxiliado na organização da sua agenda por seu agente pessoal (Figura 4). Nesta situação, o usuário precisa agendar uma reunião com diversos participantes para um horário do próximo sábado. Por experiências já feitas, o agente conhece um participante que não concordará com a reunião no sábado, por que ele nunca faz reuniões neste dia da semana. Dessa forma, o agente pode: (i) alertar o usuário sobre este problema, (ii) sugerir outra data para o encontro, que considera que todos os participantes poderão estar presente, por observação das preferências e prioridades ou (iii) não fazer nada.

Em um outro trabalho sobre softwares assistentes pessoais, Franco et al. (2007) apresenta uma arquitetura de assistentes baseada em serviços Web com conceitos de “Entidades Ativas”. Tais entidades trabalham com processos de negócios colaborativos em uma rede de dispositivos móveis para suportar a colaboração de atores em cenários de eventos extrenos, tais como desastres naturais como terremotos, furacões, inundações, desastres intencionais como incêndios e ataques terroristas. Por outro lado, este trabalho apenas incide no suporte a colaboradores envolvidos nesses cenários, assim como não os substitui em qualquer situação.

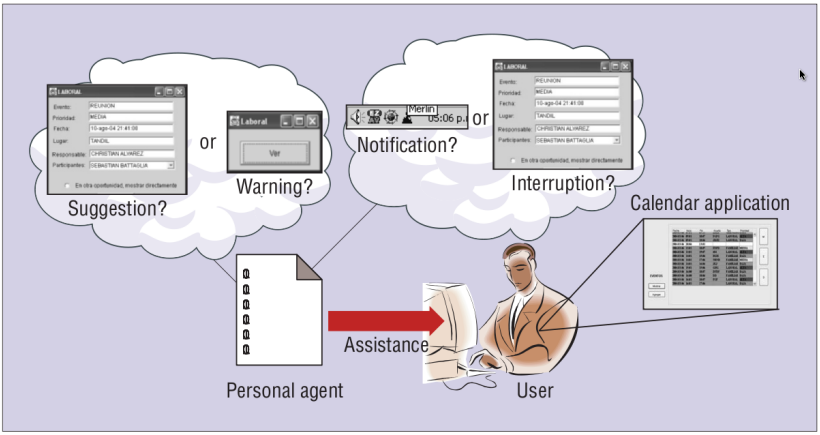

Figura 4: Opções de assistência para o gerenciamento da agenda do usuário para uma reunião com várias pessoas (Schiaffino e Amandi, 2006).

Com base nos conceitos sobre assistência pessoal via software de computador apresentados, certos requisitos gerais desejáveis para os assistentes pessoais podem ser identificados, tais como:

- Atuar com certa **autonomia** em suas tarefas de forma que estes possam verificar a situação dos seus usuários (como por exemplo novos e-mails que chegam em sua caixa postal) e que possam responder apropriadamente a cada situação que aparece (Bocionek, 1994), (Hoyle e Lueg, 1997), (Hunhs, 1998), (Schiaffino e Amandi, 2006);
- Ser **flexível** em termos de poder atuar diante de novas situações e cenários de negócios (Hunhs, 1998), (Angehrn et al., 2001);
- Ser **adaptável** ao usuário, em conformidade com as informações sobre ele, suas preferências, necessidades, situação atual, e tomando em conta a evolução dessas informações em relação ao tempo (Hunhs, 1998), (Angehrn et al., 2001), (Schiaffino e Amandi, 2006);
- **Interagir** com seu usuário de forma a tornar possível a assistência a ele (Hunhs, 1998), (Bush et al., 2006), (Schiaffino e Amandi, 2006);
- Ser **baseado em rede**, ou seja, ter a capacidade de buscar informações via redes de computador, tal como a Internet, e interagir com outros sistemas e assistentes pessoais de outros usuários de forma a alcançar os objetivos do seu usuário

(Bocionek, 1994), (Hoyle e Lueg, 1997), (Huhns e Singh, 1998);

- Ser **interoperável** com os elementos da arquitetura de referência para assistentes pessoais: comportamentos na forma de serviços e outros assistentes pessoais (Zambiasi e Rabelo, 2010);
- Ser de **propósito geral**, ou seja, não específico à apenas uma atividade ou um grupo de usuários, adaptando-se aos propósitos do seu usuário (Huhns e Singh, 1998);
- **Adaptabilidade** de contexto (Bush et al., 2006), (Huhns e Singh, 1998);

Além disso, e para se enquadrar com a proposta desse trabalho, um Software Assistente Pessoal deve ser integrado e interoperável aos processos de negócios da empresa (Huhns e Singh, 1998), (Zambiasi e Rabelo, 2010).

2.1.1 Implementações de Softwares Assistentes Pessoais

Alguns esforços têm avançado no caminho de fornecer assistência nas tarefas rotineiras das pessoas, tanto iniciativas de empresas privadas de desenvolvimento de software como iniciativas em centros de pesquisas e universidades (estudantes, professores / pesquisadores). Nesta parte, alguns desses projetos são apresentados e brevemente discutidos.

Hoyle e Lueg (1997) descrevem uma experiência com um exemplo de programa, chamado *Open Sesame!* e desenvolvido pela Macintosh[2]. O sistema foi desenvolvido com o objetivo de ser um assistente pessoal, e, Hoyle e Lueg (1997) classificaram este software como sendo o primeiro assistente pessoal disponível comercialmente. Contudo, durante a experiência de uso do sistema, diversas falhas foram verificadas nos quesitos de flexibilidade em relação a certas ações de suporte ao usuário. Também não foi possível, para o agente, gerenciar sozinho e de forma efetiva os eventos e ações do usuário no *desktop,* sendo que o próprio usuário teve que tomar decisões diferentes para cada novo momento.

O *Personal Assistant* da Shelltoys é uma espécie de software para auxílio em tarefas do usuário. Contudo, se limita a apenas gerenciar as

[2]Computadores pessoais fabricados e comercializados pela Apple Inc (APPLE, 2011).

tarefas do usuário via uma agenda de compromissos. Este assistente pessoal possui uma lista de tarefas e gera lembretes de compromissos quanto a aniversários, atividades, reuniões, com proteção de senhas, e outras pequenas funcionalidades (SHELLTOYS, 2011). É um sistema proprietário e fechado.

Um interessante projeto de Assistência Pessoal encontrado, foi o Sandy (Mann, 2011). Ele foi desenvolvido para auxiliar o usuário a lembrar de compromissos, listas, pendências, contatos, projetos. O sistema tem a capacidade de trabalhar com mensagens via SMS, Twitter e até caixa com mensagens de e-mail. Para que o usuário utilize Sandy, basta ele enviar mensagens para Sandy da forma: "Lembrar-me de pegar meu carro em 15 minutos", "O telefone do meu pai é 9999-9999", "lembrar de comprar mantimentos ovos", etc. Contudo, Sandy um projeto proprietário, fechado e foi descontinuado em 2008 (SANDY, 2011), (Dornfest, 2011). Assim, seu código não pode ser aproveitado e/ou aperfeiçoado por outros.

Um projeto de assistente pessoal da *Microsoft*[3], chamado Laura, é caracterizado como uma entidade computacional para fornecer auxílio aos usuários. Ele pode ser comparado à uma recepcionista que auxilia as pessoas a encontrarem locais específicos em um evento, ou em um centro empresarial, por exemplo. A interação de Laura é feita é por meio da visualização de uma cabeça humana representando o assistente virtual na tela de um computador. Em um exemplo de aplicação do sistema, o usuário pode falar com Laura por meio da voz para pedir coisas básicas, tais como agendar reuniões, programar voos. Além disso, Laura possui a habilidade de tomar decisões sofisticadas sobre seus usuários, julgando aspectos como roupas, estado psicológico, importância e seus horários preferidos para compromissos (Vance, 2011), (Mundie, 2008). Todavia, este é um projeto de software proprietário e qualquer adaptação ou novo comportamento que seria interessante acrescentar, precisaria ser fornecido pela própria empresa, não havendo a possibilidade de alterar o software ou acrescentar de forma flexível outras funcionalidades. Ainda, não foram encontradas referências recentes sobre tal projeto.

Outro projeto é o *Outlook Personal Assistant* (PA). Este é um *plugin* para o Microsoft Outlook que tem a finalidade de lembrar o usuário de diferentes ações. O usuário pode adicionar comentários, definir lembretes para qualquer tipo de mensagens de e-mail, lembrar

[3]Empresa de *software* dos Estados Unidos da America, conhecida principalmente pelos versões do sistema operacional Windows (MICROSOFT, 2011).

horários específicos para requisitar ações adequadas do usuário, enviar mensagens ao celular ou a outro e-mail, e também responder automaticamente a um e-mail com comentários específicos (OUTLOOKPA, 2011). Este assistente se limita ao trabalho de gerenciamento de e-mails do usuário e é, também, um sistema fechado de uma empresa específica, não permitindo acrescentar qualquer modificação ou aperfeiçoamento que se queira.

O *Virtual Assistant Manager*, da *StormSource Software*, é software de assistência pessoal virtual baseado na web. Este serve para gerenciar as tarefas de projetos do usuário, gravar o tempo gasto em cada tarefa, gerar relatório das tarefas (StormSource, 2011). Contudo, são as pessoas envolvidas que precisam efetuar as tarefas e acessar o sistema para configurar o estado das tarefas, tanto usuário do assistente como parceiros. A finalidade deste software se mantém em organizar essas informações na agenda do usuário e, via informações sobre as tarefas, enviar informações sobre solicitação de tarefas de um cliente, fechamento de uma tarefa, etc.

Há também o projeto denominado Narval (acrônimo para (*Network Assistant Reasoning with a Validating Agent Language*). Sua finalidade é auxiliar as pessoas nos trabalhos diários, em um mundo cercado de informações. Ele possui a capacidade de ser executado no próprio computador pessoal ou em um servidor remoto. A comunicação com ele é por meios normais, como e-mail, web, telnet, telefone, GUI (*Graphical User Interface*) específica, etc. O funcionamento do sistema é dado pela execução de sequências de ações descritas pelo usuário, para uma ampla gama de tarefas, como preparar o jornal da manhã, ajudar a navegar na web por meio de filtro de anúncios e lixo eletrônico, participar de leilões on-line, cuidar de tarefas repetitivas como responder e-mail, negociar data e hora de reuniões, etc. (Chauvat, 2000), (Thenault, 2011).

Chauvat (2000) o classifica como um software de assistência pessoal inteligente lançado sobre a licença GNU LGPL. O software utiliza Inteligência Artificial e tecnologia de agentes para "*mudar a maneira como a Internet e os computadores são utilizados*". O Narval é baseado em técnicas como Redes de Petri, sistemas baseados em regras, programação de contratos, planejamento, aprendizagem automática. Novos plugins para o narval podem ser desenvolvidos em Python

O Narval, desenvolvido em linguagem de programação Python, efetua um processamento em documentos XML (*eXtensible Markup Language*) em alto nível de representação de recursos externos (e-mail,

paginas web, sentenças, *bookmarks*, etc. O sistema foi utilizado para configurar um *bot* chamado Gazo, com o objetivo de traduzir a *Linux Gazette* para o francês (Chauvat, 2000), (Thenault, 2011).

O autor do projeto, Chauvat (2000), caracterizou este como o único Assistente Pessoal Inteligente lançado como software livre. Ele cita pelo menos dois projetos comerciais correlatos relevantes que são:

- Um software agregado ao vídeo cassete TiVo[4], com o objetivo de procurar programas de TV e auxiliar na gravação de algum programa, e até com a capacidade de sugerir filmes baseando-se no que a pessoa tem assistido recentemente;
- O software *Prody Parrot* que era fornecido juntamente com placas de som *Sound Blaster*. Este tentava auxiliar o usuário a classificar seu e-mail, lembrá-lo de seus compromissos, fazer resumos de páginas da web e entreter o usuário dialogando como o clássico software de simulação de psicanalista Eliza e até reconhecer fala.

Contudo, o projeto não tem tido atualizações recentes, indicando uma provável descontinuação. A última atualização do sistema, feita pelo próprio autor, foi em Novembro de 2008 e, após isso, apenas uma pequena correção feita por outro desenvolvedor no segundo semestre de 2009.

A Plaxo[5], empresa adquirida pela Comcast[6] em maio de 2008, possui um sistema chamado *Plaxo Personal Assistent*, um serviço inteligente para atualizações automáticas da agenda do usuário e suas informações de contatos. O Plaxo alerta automaticamente o usuário toda vez que ele encontra dados mais atuais sobre os contatos do usuário e requisita aprovação para atualizar as informações. Depois de um certo período de trabalho com o usuário, o sistema se torna capaz de atualizar automaticamente essas informações. O *Plaxo Personal Assistant* é um serviço pago e possui serviços de assinatura em diversos níveis, com pacotes gratuitos, *basic*, *premium* e outros. A empresa também aplicativos para iPhone, iPad, BlackBerry e Windows Mobile.

---

[4]Empresa direcionada ao entretenimento doméstico. Foi fundada em 1997 com a criação de um produto gravador de vídeo digital (DVR). (TIVO, 2011).

[5]Plaxo é uma empresa de hospedagem de agendas de contatos e uma forma de rede social para manter contato com as pessoas de forma integrada (PLAXO, 2011).

[6]A Comcast é um dos principais fornecedores de informação, entretenimento e produtos e serviços de comunicações dos Estados Unidos da America. São envolvidos na operação de conexões à cabo e no desenvolvimento, produção e distribuição de entretenimento, notícias, esportes e outros conteúdos para um público global, através NBCUniversal (COMCAST, 2011).

Um projeto bastante recente, é o Siri (2011) é um Assistente Pessoal Virtual que trabalha baseado na web e links de resultados de buscas. Sua interação com o usuário é baseada em uma forma de conversa com o usuário. O usuário diz para seu assistente o que quer fazer e este faz pesquisa em fontes de informações para poder auxiliar o usuário. Ele se utiliza de informações de preferências pessoais dos indivíduos e de um histórico de interação para ajudá-lo a resolver tarefas específicas. A intenção é que o software fique melhor com a experiência de interação com seu usuário final. Este software utiliza também como base o contexto pessoal (lugar, horário, histórico) do usuário. Ele possui algoritmos patenteados que combinam diversos atributos sobre prestadores de serviços para planejar de forma otimizada a execução de suas tarefas. Por exemplo a seleção de um restaurante para o usuário, serviços de reservas on-line, etc. O Siri foi adquirido pela Apple e se caracteriza, também, como um sistema fechado e proprietário, de um fornecedor específico.

Um projeto que pode ser considerado bastante significativo no sentido de Softwares Assistentes Pessoais, é o projeto PAL (2011), antes denominado CALO (*Cognitive Assistant that Learns and Organizes*). Este é um projeto que envolve muitos anos de pesquisa e é proveniente de um grupo de pesquisa em Menlo Park, Califórnia, SRI International. É financiado pela DARPA (*Defense Advanced Research Projects Agency*) do Pentágono e é considerado, pelas fontes, um dos maiores projetos de inteligência artificial que se tem relato. Este projeto tem por objetivo auxiliar os usuários nos trabalhos com sistemas computacionais e automatizar tarefas rotineiras de forma a liberar as pessoas para as tarefas mais importantes (Myers e Yorke-Smith, 2005), (Markoff, 2008).

O objetivo inicial do DARPA, em relação ao projeto PAL, era criar um sistema cognitivo que pode raciocinar, aprender e lidar com situações imprevistas como forma de fornecer assistência em situações militares. Além disso, o sistema deve poder se adaptar a mudanças em seu ambiente, aos objetivos dos seus usuários e as suas tarefas sem a necessidade de alterar sua programação ou de intervenção técnica. Este projeto está sendo executado por um conjunto de cientistas da computação e pesquisadores em inteligência artificial, percepção, aprendizado de máquina, processamento de linguagem natural, representação de conhecimento, diálogo multimodal, ciberconsciência, interação homem-máquina e planejamento flexível (PAL, 2011).

Segundo Markoff (2008), este é um trabalho com objetivos militares e muito pouco sobre os detalhes da pesquisa eram divulgados.

Na Tabela 1 são apresentadas algumas funcionalidades encontradas nos softwares existentes para assistência pessoal e é possível observar um comparativo com as características que fundamentam a proposta dessa Tese.

	1	2	3	4	5	6	7	8	9	10	11	12
Open Sesame! - Macintosh		X						X			L	
Personal Assistant - Shelltoys		X				X		X			P	
Software Sandy	X	X		X				X		X	X	
Laura - Microsoft				X				X		X	X	
Outlook Personal Assistant		X						X				
Virtual Assistant Manager		X						X				
Projeto Narval	P	X	X		X				X	X	X	
Plaxo Personal Assistant		X						X			L	
Siri - Apple				X	X	X		X		X	L	
PAL - DARPA		X		X	X	X		X		X	X	
Proposta	P	P	X	P	X	X	X		P	P	X	X

Tabela 1: Comparativo entre sistemas de Assistência Pessoal e a Proposta.

Marcadores:

- X – Possui
- P – Sua estrutura permite ou prevê a implementação
- L – Possui de forma limitada

Legenda:

1. Comunicação via sistemas existentes - SMS, Twitter, e-mail
2. Gerenciamento de agenda e compromissos do usuário
3. Novas funcionalidades podem ser acrescidas como *plugins*
4. Comunicação via diálogos e conversação
5. Pesquisa de informações na internet
6. Funcionamento na web
7. Utiliza padrões abertos
8. Software Proprietário e/ou fechado
9. Pode ser adaptado aos processo de negócio da Empresa
10. Inteligência Artificial ou Aprendizado
11. Autonomia
12. Sugere Arquitetura de referência

Por meio do que foi visto, pode-se notar a evolução dos assistentes pessoais no que tange aos conceitos, as tecnologias utilizadas, a atenção dada a tal campo de aplicação e recursos utilizados para tal, conforme os últimos projetos vistos. Ainda, essa evolução foi possível devido aos avanços na utilização da computação em rede, avanços na utilização e maturidade atual da Internet, melhoria nas interfaces de usuários, aperfeiçoamento dos computadores *Desktop* e dispositivos móveis (hardware) e outros.

2.1.2 Subproblemas

A complexidade para representar uma pessoa em determinadas situações, dentro dos conceitos apresentados, é um grande problema do ponto de vista computacional. Em vista disso, várias abordagens têm sido propostas para lidar com essa complexidade. Contudo, a maioria delas geralmente ataca apenas a questão de potencializar a execução de um processo da forma mais correta, e ainda assim, necessitando geralmente da presença e grande intervenção do usuário.

Tendo em vista tal problemática, é possível identificar certos subproblemas que circundam o tópico da assistência pessoal via software de computador (Figura 5).

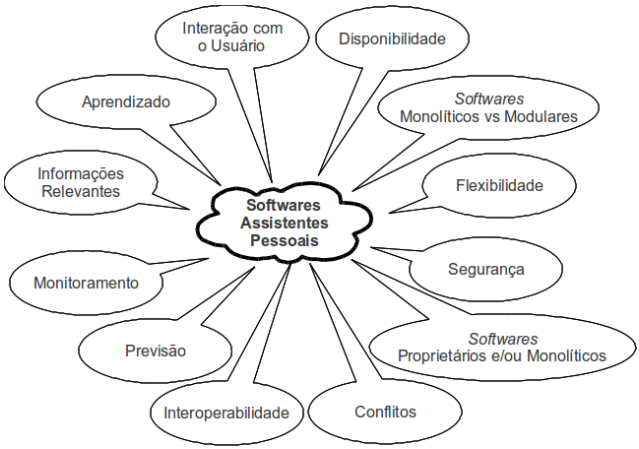

Figura 5: Subproblemas da assistência via software de computador.

Um ponto importante a se observar dos Softwares Assistentes Pessoais, é que em geral eles são proprietários e fechados, não permitindo que novas funcionalidades sejam adicionada a esses por terceiros ou pelo usuário (Zambiasi e Rabelo, 2010).

2.1.2.1 Informações Relevantes

Alguns tipos de softwares assistentes são programados para encontrar algum tipo de informação na Internet ou bases de dados, apenas provendo alguma ajuda para encontrar informações disponíveis e executar tarefas de forma mais efetiva (Huhns e Singh, 1998). Contudo, os programas de computador são cada vez mais baseados em conhecimento, com muitas informações específicas em cada diferente domínio (Michaell et al., 1994).

Para que um software possa "saber" o que fazer, quando determinada tarefa deve ser executada e como ela deve ser executada, são necessárias informações que contextualizem as preferências, necessidades e desejos dos seus usuários (Bush et al., 2006), (Huhns e Singh, 1998), (Schiaffino e Amandi, 2006). Porém o problema está em definir quais informações referentes aos usuários precisam ser armazenadas, como elas podem ser adequadamente alocadas em locais de acesso eficiente pelos sistemas. Além disso, quando e como utilizá-las é um fator de grande relevância para a execução da assistência aos usuários.

Um dos grandes problemas, citado por Blum e Langley (1997), é encontrar as informações realmente relevantes em uma grande quantidade de dados. Esse problema concomita com os vastos recursos de informações, muitas vezes de baixa qualidade, existentes hoje na Internet e na *World Wide Web*. Isso passa a requerer formas de filtragem e recuperação de informação mais eficientes. Contudo, a definição de "pertinente" pode possuir uma variedade de significados, pois deve ser levado em consideração o contexto em que a informação é relevante.

2.1.2.2 Aprendizado

Um problema que pode ser normalmente encontrado na utilização da tecnologia de agentes é que o conhecimento do agente é, na maioria

das vezes, estático e não pode ser ajustado para os hábitos e preferências de cada usuário individualmente (Maes, 1994). Contudo, para que um agente possa fornecer assistência pessoal, o ambiente desses precisa conter mecanismos para aceitar as adaptações dos seus interesses, necessidades e comportamentos (Sensoy e Yolum, 2008). A possibilidade de prover aos agentes uma forma mais dinâmica de gerência dos seus conhecimentos, segundo Maes (1994), é compreender as ações e questionamentos dos usuários e ajustar sua ação em conformidade com tais especificidades.

Em tempo, além ter consciência do contexto do usuário e de suas informações de forma adequada, um Assistente Pessoal deve ser capaz de ampliar suas atividades e aprender com as situações já executadas, adaptando-se a novas situações por meio deste aprendizado (Angehrn et al., 2001), (Hoyle e Lueg, 1997), (Schiaffino e Amandi, 2006), (PAL, 2011).

Para Monard e Baranauskas (2003), fazer com que programas de computador adquiram conhecimento de forma automática é o objetivo da área chamada de aprendizado de máquina, abordada pela disciplina de Inteligência Artificial. Para eles,

> um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem sucedida de problemas anteriores.

Para Simon (1983) o aprendizado se caracteriza por alterações que objetivam adaptar um sistema de forma a realizar uma tarefa de forma mais eficiente e eficaz da próxima vez que esta for efetuada.

Na década de 90 houve um grande aumento nas investigações sobre aprendizagem de máquina provindas de várias frentes: da Inteligência Artificial simbólica, das redes neurais artificiais, das áreas de estatística, reconhecimento de padrões e da teoria de aprendizagem computacional. Juntamente com isso, novos problemas começaram a ser tratados na aprendizagem de máquina, tais como *datamining*, controles de robôs, otimização combinatória, reconhecimento de voz e facial, análise de dados médicos jogos, entre outros (Dietterich, 1997).

O aprendizado de máquina tem sido, desde o reconhecimento da Inteligência Artificial por volta de 1950, uma área de importante investigação. Isto pois a capacidade de aprender é uma forte característica do comportamento humano e porque a aprendizagem oferece uma forma potencial de desenvolvimento de sistemas de alto desempenho. Por um lado, sistemas de aprendizagem devem monitorar

seu próprio desempenho para poder melhorá-lo, por meio de ajustes internos, por outro lado o aprendizado de máquina pode se dar pela aquisição de conhecimento estruturado na forma de conceitos, ou regras de produção, alimentados pelo conhecimento representado explicitamente, como o caso de sistemas especialistas[7] (Quinlan, 1986).

Segundo Michaell et al. (1994) objetivo da investigação da aprendizagem de máquina é desenvolver sistemas de computador que melhoram seu desempenho através da experiência e, mais recentemente, com capacidades de aprendizado mais robustas. Por outro lado, intenta produzir uma tecnologia independente de domínio para uma vasta gama de aplicações de computação. Isso teria um impacto significativo em diversas áreas: robótica, CAD (*Computer Aided Design*), bancos de dados inteligentes, etc. Em um olhar mais distante, tal investigação poderia produzir robôs que aprendem e operam em ambientes novos, compreendem a linguagem falada e se adaptam a novas condições ambientais. Estes poderiam servir, por exemplo, como sistemas consultores baseados no conhecimento humano para auxiliar especialistas a resolverem problemas difíceis.

Uma forma de se trabalhar com o aprendizado de máquina é através do aprendizado indutivo. Isso é feito através do raciocínio sobre exemplos fornecido por um processo externo ao sistema de aprendizado e pode ser classificado como supervisionado e não supervisionado. No aprendizado supervisionado um conjunto de informações de treinamento são passados para o sistema e no aprendizado não supervisionado é efetuada uma tentativa de juntar as informações passadas em agrupamentos relacionados. Este aprendizado requer uma pós análise para definir o contexto de cada agrupamento (Baranauskas, 2001).

Segundo Monard e Baranauskas (2003), os paradigmas de aprendizado de máquina podem ser classificados em cinco tipos:

- **Simbólico**: Baseado na análise de exemplos e contraexemplos de conceitos representados por símbolos. Estes podem ser representados por expressões lógicas, árvores de decisão, regras ou rede semântica.
- **Estatístico**: Consiste na ideia de utilizar modelos estatísticos para encontrar uma boa aproximação do conceito induzido. O aprendizado Bayesiano, modelo probabilístico baseado em conhecimento prévio combinado com exemplos de treinamento

[7]Sistemas baseados em regras e base de conhecimento que aprendem com o tempo e simulam o conhecimento de um especialista para solucionar problemas de domínios específicos (Nilson, 1982).

para determinar a probabilidade final de uma hipótese, é um dos métodos estatísticos utilizado.

- **Baseado em Exemplos** (*lazy*): Este método toma como base a ideia de utilizar exemplos similares para tentar resolver um novo exemplo. Este tipo de aprendizado necessita manter na memória todos os exemplos já vistos.
- **Conexionista** (Redes Neurais Artificiais - RNA): Este tipo de aprendizado é inspirado no modelo biológico do sistema nervoso. Uma RNA é modelada matematicamente por unidades altamente interconectadas que simulam de forma bastante simples o funcionamento do sistema nervoso.
- **Genético**: Derivado do modelo evolucionário de aprendizado e a teoria de Darwin, tem como base uma população de elementos em que os mais fortes possuem maiores probabilidades de proliferar com variações de si mesmos. Dessa forma, segundo de Jong (1988), a computação evolucionária pode ser utilizada para a aprendizagem de máquina, tal que também evolui conforme o tempo. Para Goldberg e Holland (1988), as propriedades da computação evolucionária são bastante atraentes para serem exploradas de forma mais aprofundada na questão da aprendizagem de máquina. Para ele, a adoção dos algoritmos genéticos no aprendizado de máquina possui uma tendência natural na medida que ambos evoluem com o tempo.

Conforme de Jong (1988), a dificuldade de se definir uma forma universal de aprendizagem é devido a complexidade dos sistemas de aprendizagem. Porém, a correlação entre estes sistemas é que devem evoluir ao longo do tempo para melhorar seu desempenho.

Uma interessante técnica de aprendizagem de máquina é o aprendizado por reforço. Esta técnica trabalha em reorganizar sua estrutura conforme o retorno do resultado de sua interação com o ambiente em que se encontra. Um dos problemas dessa técnica é definir, por meio do resultado da ação de um agente em seu ambiente, a pontuação das punições e recompensas de forma a ajustar o seu comportamento para melhorar seu desempenho. Outro problema é que um agente deve ter a capacidade de generalizar a partir de um conjunto grande de problemas. Por fim, uma questão sobre as técnicas de aprendizado por reforço é que elas geralmente convergem muito lentamente (Lin, 1992).

Michalski, (2000) apresenta o *Learnable Evolution Model* (LEM) para trabalhar com aprendizado de máquina. O LEM, ainda que seja evolutivo, se difere do modelo darwinista da computação evolucionaria. Este método cria novas populações através da geração de hipóteses sobre a alta e baixa aptidão dos indivíduos. Dessa forma, a produção de indivíduos é feita de maneira não aleatória, mas sim através de um raciocínio deliberado pela geração e instanciação de hipóteses sobre os indivíduos da população. Uma característica bastante interessante deste método é que o algoritmo leva em consideração não apenas a experiência de um único indivíduo para a geração de novos indivíduos, mas a experiência de toda a população.

Existem diversos programas que se utilizam da resolução de regras para aprender. Por exemplo, aprendem regras para diagnóstico de doenças, muitas vezes efetuando tal trabalho tão bem quanto uma pessoa. Outros adquirem a capacidade de reconhecimento de palavras faladas. Estes programas apresentam a viabilidade da aprendizagem de máquina em domínios específicos (Michaell et al., 1994).

2.1.2.3 Interação com o Usuário

Ainda, para que o SAP possa fornecer os recursos necessários para sua assistência, disponibilidade e interatividade, este precisa se comunicar de forma eficiente com seu usuário, independente de onde o usuário está localizado (Bocionek, 1994), (Bush et al., 2006), (Franco et al., 2007). Entretanto, quando o software deve se comunicar e interagir com o usuário, e em que situações específicas isso deve ser feito não é necessariamente uma tarefa trivial. É necessário que hajam definições de quando, como e em que situações o usuário deve ser interrompido para que os processos que estão em operação possam ser tratados pelo assistente.

A interação humano-computador já se encontra tão onipresente no mundo hoje em dia que as pessoas nem se dão conta do quanto ela tem afetado a automatização dos sistemas, desde abordagens baseadas em processamento de linguagem natural, utilização de Inteligência Artificial e formas mais naturais de comunicação com o sistema, tais como fala e telas sensíveis ao toque. Todas os esforços nesse sentido têm apenas um objetivo, a busca da satisfação do usuário em relação à utilização dos recursos computacionais (Levin et al., 2000).

A modelagem da interação entre o usuário e um sistema segue rumos em diversas áreas de investigações e possui o objetivo principal de produzir sistemas mais úteis e mais facilmente utilizáveis (Fischer, 2001).

2.1.2.4 Previsão e Monitoramento

Por mais que a tecnologia de agentes resolva várias questões em relação à assistência pessoal, diversos outros problemas acabam emergindo com seu uso. Dentre estes problemas, incluem-se falhas ao tentar descobrir os objetivos e necessidades dos usuários, considerações inadequadas sobre ações, custos/benefícios com ações mal organizadas temporalmente. Além desses, o baixo tempo de ação e a falha na otimização de oportunidades aos usuários são outros problemas identificados (Schiaffino e Amandi, 2006). Ainda, mesmo que a detecção de novas oportunidades de negócios e atividades seja de grande importância para o usuário, a maneira como isso pode ser feito automaticamente ou com algum nível de interação para cada tipo de oportunidade não tarefa fácil.

O monitoramento por meio de programas de computador pode ser comparado com a maneira que os humanos fazem observações. Os humanos, ao observar os processos da natureza por exemplo, criam formas de tentar compreendê-los, compondo por vezes características que facilite a compreensão do objeto em questão. O armazenamento destas características, independente de quais forem, podem seguir cinco estágios básicos: (i) memória humana; (ii) manual em papel, para manter os dados por um tempo maior; (iii) arquivo no computador, para armazenamento em forma digital; (iv) base de dados, com objetivo de acesso mais rápido; e (v) *datamining* ou *data warehousing*, quando há grandes volumes de dados para suporte à tomada de decisão (Baranauskas, 2001).

Uma forma natural para os humanos trabalharem com esses dados é na forma de exemplos. Neste caso, um subconjunto dos dados são selecionados para fazerem parte de um conjunto de treinamento para o algoritmo de aprendizado de máquina. Um dos objetivos de utilizar um classificador é para predição de novos exemplos tão precisamente quanto possível e outro é compreender a relação estrutural existente entre a classe e os atributos medidos (Baranauskas, 2001).

2.1.2.5 Flexibilidade

Para Michaell et al. (1994), a principal questão dos assistentes pessoais está em construí-los de forma que possam ser facilmente personalizados para cada usuário. Muitos programas proveem simples parâmetros que permitem aos usuários personalizarem seus comportamentos explicitamente. Porém, em geral, essa interação é bastante limitada. Uma maneira de se resolver de forma sofisticada esta limitação, é por meio da dinamicidade e da escalabilidade. Com isso, a maioria dos usuários poderia ter disponível a possibilidade de programar suas preferências de forma explícita. Ainda assim, algumas tarefas como personalizar um filtro de e-mails para selecionar mensagens urgentes, por exemplo, necessitam que a pessoa tenha certa noção detalhada e possa articular conceitos bastante sutis. Uma forma bastante interessante para resolução desse problema é deixar que especialistas desenvolvam os filtros e ao usuário fica apenas o papel de utilizá-lo e configurá-lo conforme suas necessidades.

Contudo, o problema de se lidar com a flexibilidade da forma como o agente se comporta, vai além de simplesmente alterar a configuração de um agente para cada usuário.

Brooks (1991) apresenta uma forma de trabalhar com o comportamento de agentes, no caso agentes se hardware, na forma de uma inteligência baseada na decomposição em comportamentos conectados sensorialmente à ação. Neste enfoque, os comportamentos são blocos funcionais e o comportamento geral emerge do conjunto da execução deste conjunto de blocos. Assim, um agente pode ter múltiplos comportamentos para lidar com uma variedade de circunstâncias e realizar uma variedade de tarefas.

Contudo, para Brooks (1991), deve-se observar duas questões fundamentais no mecanismo da organização dos comportamentos para sua execução: (i) o mecanismo de seleção de comportamento é centralizada ou descentralizada?; e (ii) a forma de resolução de conflitos é com prioridade fixa ou pode ser reconfigurada dinamicamente?

Segundo Pfeifer et al. (2007), a maioria dos avanços relacionados ao comportamento na robótica[8] convergem para um conjunto de princípios. Para ele, o comportamento de qualquer sistema não é apenas

[8]Embora neste ponto esteja sendo referenciado trabalhos da robótica, tanto a robótica como a tecnologia de agentes abordam assuntos sobre o comportamento. Para a teoria de agentes, um robô é classificado como um agente de hardware e um programa de computador agindo em seu ambiente computacional é um agente de software (Russel e Norvig, 2004).

o resultado de uma estrutura de controle interna, tal como um sistema nervoso central. O comportamento é afetado também pelo o ambiente em que este se encontra, pela sua morfologia e pelo material dos elementos que compõe sua morfologia.

Os sistemas biológicos, por sua vez, são uma grande fonte inspiradora para o desenvolvimento de robôs. Isto devido sua grande diversidade de espécies, com diferentes tipos de estruturas, habitats e comportamentos. Com isso, pode-se abstrair uma fonte muito rica em conhecimentos que podem ser utilizados nas disciplinas da robótica, tal como a neurociência. O intuito é abstrair os princípios dos sistemas biológicos para a concepção de tecnologia simulada. Embora essa ideia já esteja a um bom tempo sendo trabalhada, elas ainda não foram suficientemente exploradas (Pfeifer et al., 2007).

Pode-se tomar como partida, em relação a dinamicidade do comportamento de certos agentes, uma comparação com elementos em jogos de computador. Nestes, em geral existem dois desafios quando deseja-se aplicar Inteligência Artificial. O primeiro é que um jogo de computador é altamente dinâmico, complexo e imprevisível e o segundo é que os recursos de processamento para a execução de cada agente é bastante limitado. Deve-se levar em conta que muito do processamento do computador é distribuído entre os milhares de agentes e elementos que estão em execução no jogo, além do processamento gráfico, por mais que muito deste seja enviado à placas gráficas. Quando um agente faz um planejamento para agir no ambiente, deseja-se que seja um bom planejamento, mas não exatamente um ótimo planejamento, devido ao problema de processamento já explicado. O importante é que o agente possa escolher o comportamento certo para cada situação, mas sempre existe a possibilidade de falhas (Hawes, 2001).

Hawes (2000) apresenta uma forma de trabalhar com este problema com um modulo de planejamento baseado no *anytime paradigm* (Hawes, 2000 apud Dean and Boddy, 1998), em que um algoritmo *anytime* é um algoritmo que pode ser interrompido a qualquer momento e os resultados que são devolvidos após uma interrupção aumentam com o tempo de processamento. Uma vantagem deste tipo de comportamento é que, enquanto o agente estiver elaborando um plano, o agente pode interromper seu processamento para executar algum comportamento de maior prioridade e retornar a posteriori para terminar o planejamento de suas ações (Hawes, 2001).

Não obstante, o estudo das emoções tem assumido um grande papel nas pesquisas de Inteligência Artificial e tem mostrado que as

emoções são parte fundamental para o comportamento inteligente de agentes e robôs. Contudo, é importante separar o estudo das emoções como forma estética em agentes de interação humano-computador da forma de comportamento para alcançar os objetivos dos agentes. Neste enfoque, Scheutz et al. (2000) propõe uma estratégia para analisar as diferenças dos estados cognitivos e afetivos dos agentes.

Minsky (1988) discute a questão sobre as máquinas inteligentes e as emoções. Para ele, a evolução dos sistemas inteligentes está atrelado às emoções tão intrinsecamente que ele não vê a criação de máquinas realmente inteligentes que não possuam emoção. Este direcionamento se difere bastante do paradigma tradicional que impele ao questionamento se as máquinas poderão ter emoções.

Para Damásio (2000) a emoção está relacionada diretamente à sobrevivência de um indivíduo e apresenta um conjunto de reações que em geral podem ser observáveis. As emoções são encadeadas, em geral, por estímulos externos ao indivíduo que provocam alterações internas e que, por fim, é suficiente para gerar algumas reações sobre seu estado físico e responder a este estímulo. Sua função biológica, nos indivíduos, é produzir uma reação específica a um estímulo de percepção e regular o estado interno do organismo de modo que ele possa estar preparado para a reação específica. Desse modo, as emoções fornecem aos organismos comportamentos voltados para a sobrevivência e aumenta a capacidade do organismo reagir de forma adaptativa conforme suas necessidades.

Nesse sentido, diversos estudiosos têm direcionado seus esforços para compreender a forma de interação entre as emoções e o processo de cognição, trabalhando em algumas questões como que papel as emoções desempenham no processamento das informações, em que situações elas são úteis e em que situações elas são prejudiciais, como elas afetam a interação social e a comunicação dos indivíduos, etc (Scheutz et al., 2000).

Ainda, conforme Zhu (2001), um ser autônomo, pró-ativo e adaptativo, pode gerar comportamentos emergentes, ou seja, pode se comportar de formas que os próprios desenvolvedores ou usuários não esperam. Alguns estudos têm trabalhado na especificação de formalismos e lógicas para uma certa predição destes comportamentos emergentes. Porém, a solução para este problema não tem se mostrado muito fácil de se encontrar. Neste sentido, Zhu (2001) trabalham em suas pesquisas, em formas de especificar um comportamento do agente com um conjunto de regras que regem suas reações e os vários cenários do seu ambiente.

Estes esforços tem demonstrado que seguir caminhos da computação baseada no enfoque biológico são bastante atraentes para modelar comportamentos de agentes, e robôs, mais eficientes e dinâmicos de forma a alcançarem os objetivos dos agentes autônomos.

2.1.2.6 Conflitos

No que tange aos processos dos assistentes, ou comportamentos, podem existir conflitos entre a execução de determinadas tarefas. Um assistente deve poder ter a capacidade de gerenciar estes conflitos, identificando a ordem para a execução de cada tarefa por meio de configuração de prioridade e outras informações, cumprindo seus objetivos em conformidade com as necessidades dos usuários (Bocionek, 1994). De modo mais abrangente, uma solução que poderia resolver em parte alguns conflitos, seria trabalhar de forma distribuída, com comportamentos sendo executados sequencialmente ou paralelamente em dispositivos computacionais localizados remotamente.

Segundo Brooks (1991), um problema quando se lida com comportamentos divididos em blocos funcionais é decidir qual comportamento, ou comportamentos, devem ser ativados em um determinado tempo. Para resolver o conflito, de forma a decidir de forma correta comportamentos a ser executado, ele apresenta certos aspectos que devem ser considerados:

- **Coerência**: Mesmo que muitos comportamentos possam estar ativos no mesmo tempo, deve haver certa coerência da ação e dos objetivos. Não deve haver uma mudança rápida entre comportamentos incompatíveis, nem devem ser ativados simultaneamente comportamentos que conflitem, ou interfiram, uns com os outros ao ponto de não poderem funcionarem corretamente.
- **Relevância**: Os comportamentos ativos devem ser correspondentes com a situação. Por exemplo, um agente deveria responder a mensagens via *Instant Messaging* quando o usuário não está disponível, e não quando ele está em uma conversa online.
- **Adequação:** O mecanismo de seleção de comportamentos deve garantir que as metas do agente, a longo prazo, sejam cumpridas. Por exemplo, um agente, para que execute o

gerenciamento de um usuário para uma viagem a uma reunião da empresa em outra cidade, precisa executar diversos outros comportamentos para que o objetivo principal seja cumprido, tal como agendamento de passagens, hotel, recuperação de informações do local (mapas, restaurantes, locomoção, etc).

Para resolver o problema de conflitos, integrando múltiplos comportamentos em um único sistema, Brooks (1991), por exemplo, utiliza um modelo inspirado biologicamente de sistemas hormonais. Ainda, neste é aplicado um refinamento da arquitetura de *Subsuption* (Figura 6) em que os comportamentos são empilhados com prioridades conforme a necessidade de resposta a um estimulo externo (Brooks, 1989).

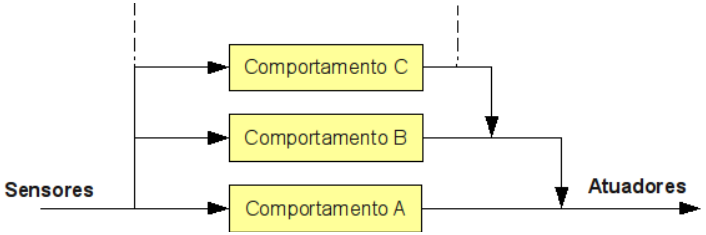

Figura 6: Arquitetura de Subsumption de Brooks (1989).

Na Arquitetura de *Subsumption* de Brooks (1989), modelada para robôs, os comportamentos são colocados sobrepostos. Conforme um conjunto de informações que são recebidas pelos sensores, um comportamento é ativado. Quando existe um comportamento de maior prioridade, este sobrepõe os de menores prioridades para entrar em execução, assumindo a execução dos atuadores do sistema robótico. Por exemplo, caso o "Comportamento A" seja "andar para frente", o robô assume este comportamento enquanto as informações dos sensores o satisfizerem. Caso algum obstáculo apareça na frente do robô, o "Comportamento B", referente a "desviar de obstáculo", é ativado e entra em ação, sobrepondo o "Comportamento A" e assumindo o controle dos atuadores.

2.1.2.7 Segurança

A integração desses sistemas computacionais, caracterizados pelo agente e cada um dos seus comportamentos combinados, findam na

composição de um sistema distribuído. Estes elementos mantêm-se interligado através da comunicação via rede em que os computadores envolvidos se comunicam e coordenam suas ações por meio de mensagens (Coulouris et al., 2005), (Kshemkalyani e Singhal, 2008).

Porém, a distribuição dos comportamentos para o software assistente gera novos problemas, tais como problemas de segurança. Este, por sua vez trabalha com medidas de garantia de privacidade, integridade e disponibilidade de recursos no sistema distribuído. Isto surge como forma de proteger os usuários do sistema contra potenciais operações maliciosas ou outros tipos de ataques, principalmente no caso de transações/negócios comerciais e/ou sigilosos (Coulouris et al., 2005).

2.1.2.8 Disponibilidade

A disponibilidade, quando se tratando de Softwares Assistentes Pessoais, refere-se ao acesso de recursos a repositórios, localizados remotamente (Bush et al., 2006). A questão está em como o software deve agir, caso esses recursos não estejam disponíveis. Por exemplo, se um assistente pessoal precisa efetuar, para seu usuário, uma reserva em um hotel, em uma viagem de negócios, mas o sistema para efetuar essa reserva não se encontra disponível. Dessa forma, é de grande importância que o assistente saiba como lidar com a situação, tal como: tentar acessar o sistema mais tarde, procurar outro hotel, tentar outra forma de reserva, contatar o usuário, etc.

2.1.2.9 Interoperabilidade

A melhor forma de se resolver o problema da interoperabilidade é adotar algum tipo de tecnologia difundida e bastante promissora, como forma de prover uma padronização. Por exemplo, os serviços Web se apresentam para tal problemática de forma bastante interessante no caso da utilização de recursos remotos por meio da Web. Estes se caracterizam como aplicações que podem ser descritas, localizadas e invocadas via Web, além de serem autocontidas e modulares (Huang, 2003). Além disso, a utilização de tal tecnologia tem se difundido muito entre grandes companhias americanas (Gesser, 2006).

Ainda, a utilização desse caminho para a utilização de comportamentos distribuídos provê também todas as vantagens e os conceitos firmados da Arquitetura Orientada a Serviços (*Service Oriented Architecture* - *SOA*). Nesta, suas funções se apresentam como serviços independentes e cada qual com interfaces de invocação bem definidas. Estas podem ser utilizadas sequencialmente ou paralelamente para compor processos de negócios (MacKenzie et al., 2006). Este conceito mostra uma forma atraente de pensar o funcionamento dos comportamentos de um Software Assistente Pessoal sob a visão de SOA.

Além disso, o SOA se apresenta como uma forma para organizar soluções que promovem o reuso, crescimento e interoperabilidade. Essas soluções são vistas na forma de serviços da qual as competências são colocadas. O SOA não é propriamente uma solução para problemas de um certo domínio, mas sim fornece um paradigma de organização da utilização de funcionalidades, com suas respectivas competências. Enquanto um provedor de serviços apresenta soluções para um determinado problema, outro provedor de serviços, que pode não ser a mesma entidade, pode apresentar outros serviços que possuem a mesma funcionalidade (MacKenzie et al., 2006). Tal característica traz a possibilidade de um usuário de um Assistente Pessoal, que utilize tal tecnologia, não necessariamente depender apenas do conjunto de funcionalidades de uma empresa específica.

Para MacKenzie et al. (2006), as empresas podem passar a oferecer funcionalidades na forma provedores de serviço. A descrição desses serviços permite que os usuários dessa tecnologia decidam se o serviço é conveniente para suas necessidades atuais. Em contrapartida, o mesmo serviço estabelece quando um consumidor satisfaz todos os requisitos do provedor de serviço.

2.1.2.10 Softwares Proprietários e/ou Monolíticos

Em relação aos assistentes pessoais, existem tanto esforços privados, como um ou outro projeto de código aberto, já citados anteriormente. Contudo, os projetos privados nem sempre podem ser adaptados às necessidades do usuário, ou mesmo, não podem ser ajustados as necessidades da empresa e das tarefas do usuário na empresa. Mesmo que alguns sejam direcionados a alguns desses pontos, eles não abordam todos os pontos suficientes para serem adaptáveis a

cada usuário em suas tarefas diárias e na empresa. Ainda, os softwares proprietários não fornecem a possibilidade de alterar o código ou acrescentar de forma flexível outras funcionalidades.

Quanto aos projetos de código aberto, eles nem sempre seguem padrões e são, em geral monolíticos. No caso o projeto Narval, este foi aparentemente descontinuado e não apresenta mais melhorias e é monolítico, apesar de aceitar *plugins*.

2.2 ARQUITETURA ORIENTADA A SERVIÇOS

Conforme definido nos objetivos, a Arquitetura Orientada a Serviços (*SOA – Service Oriented Architecture*) é utilizada como base para a criação da Arquitetura de Referência para Softwares Assistentes Pessoais. Em verdade, esta foi escolhida como Estilo Arquitetural que, juntamente com o Modelo de Referência para Assistentes Pessoais, deve servir para nortear os princípios de funcionamento e interoperabilidade dos elementos da Arquitetura de Referência. Dessa forma, se torna necessária uma explanação sobre o assunto ao leitor, situando-o de certas especificidades e apontando alguns conceitos que são citados e utilizados nos próximos capítulos dessa Tese. Também, não convém extrapolar, ou estender exaustivamente, o assunto de SOA, mas sim, colocar o leitor a par dos termos utilizados na proposta.

A Arquitetura Orientada a Serviço (SOA), conforme Estefan et al. (2008) *"é um paradigma para organização e utilização de competências distribuídas que estão sob o controle de diferentes domínios proprietários"*. Ou seja, as pessoas e organizações criam competências para resolver problemas específicos conforme suas necessidades. Estas são modeladas através de um conjunto de componentes que compõe a arquitetura e podem ser invocados por meio da descrição de suas interfaces, que podem ser publicadas e descobertas. (Booth et al., 2004).

Segundo Stojanovic e Dahanayake (2005):

> O ponto central do paradigma da Engenharia de Sistemas orientados a Objetos (*SOSE – Service-Oriented System Engineering*) é o conceito de serviço, bem como a estratégia para expor sistema de capacidades para os consumidores como os serviços através do *Service-Oriented Architecture* (SOA). Neste contexto, a tecnologia de serviços

> Web é apenas uma maneira de realizar eficientemente os conceitos de serviços e SOA. O serviço faz um acordo contratual entre fornecedor e consumidor. Além de uma interface comum que define operação assinaturas, um serviço também pode ter atributos que lhe são próprias, tais como o acordo de nível de serviço, das políticas, das dependências, e assim por diante. - (Stojanovic e Dahanayake, 2005).

A tecnologia da Arquitetura Orientada a Serviços surgiu como forma de reduzir o tempo, custos e recursos gastos na tentativa de uma integração rápida e flexível dos sistemas de Tecnologia da Informação e Comunicação envolvidos nas organizações. A utilização de SOA enfoca no aumento da agilidade na implementação de novos produtos e processos e na composição de novos processos por meio da orquestração de componentes já prontos. Além disso, abrange a possibilidade de reuso de sistemas e aplicações legadas. Esta tem como base componentes baseados em serviços, para que se possa atingir essa integração de forma rápida e flexível (Dudley, 2007).

Na arquitetura SOA, as aplicações de negócios do cotidiano são divididas em funções e processos de negócios individuais, compartilhados e reutilizáveis. Estes, chamados de serviços, são a composição básica da arquitetura SOA e podem fazer parte da composição de outros serviços pela integração/orquestração destes, independente das aplicações e plataformas computacionais para a execução (Piazza, 2007).

Para tornar estes serviços disponíveis, e viabilizados para o uso por terceiros, deve ser disponibilizada uma descrição deste serviço contendo as informações necessárias para a interação com ele e descrevendo suas entradas, saídas e semânticas associadas à eles (Estefan et al., 2008).

As interfaces de todas as funções devem estar bem definidas na forma de serviços independentes com interfaces de invocação bem definidas. Estas podem ser implementadas com várias tecnologias. Contudo, a principal tecnologia empregada hoje por diversas empresas por possuir um meio padronizado de implementação e a um custo razoável é a tecnologia de serviços web (Gesser, 2006).

Um dos principais objetivos da utilização de serviços web é a implementação de SOAs, por ser a tecnologia que melhor atende seus requisitos. Estes requisitos, segundo Singh e Huhns (2005), são:

- **Baixo acoplamento:** Os serviços da arquitetura não devem ter uma dependência forte entre si.
- **Independência de implementação:** Não se deve depender de características específicas de linguagens de programação ou ambientes de execução.
- **Configuração flexível:** Os diferentes serviços devem poder ser ligados entre si de forma dinâmica e configurável.
- **Tempo de vida longo:** Os serviços devem existir por tempo suficiente para que sejam descobertos e utilizados até se obter confiança em seu comportamento.
- **Granularidade:** As funcionalidades de um sistema devem ser divididas em vários serviços.
- **Distribuição:** Os serviços devem ficar distribuídos, para aproveitar melhor os recursos computacionais.

A arquitetura SOA combina a habilidade de invocar objetos remotos e funções com ferramentas para a descoberta dinâmica de serviços, com ênfase na interoperabilidade (Agrawal et al., 2002). Entretanto, a arquitetura SOA não é, por si só, uma solução para problemas para um dado domínio, e sim um paradigma de organização e entrega que permite obter mais valor das competências localmente disponíveis e daquelas controladas por terceiros (MacKenzie et al., 2006).

Segundo Booth et al. (2004), a arquitetura SOA é composta por três componentes básicos: o provedor, o registro e o consumidor (Figura 7).

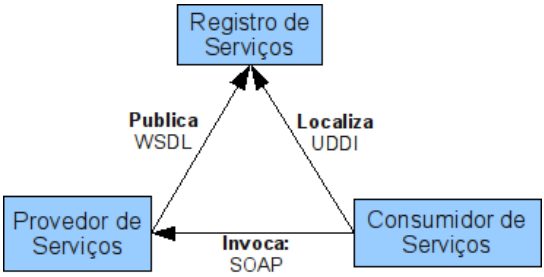

Figura 7: Modelo conceitual da Arquitetura Orientada a Serviços (Booth et al., 2004).

Nesta arquitetura, o Provedor de Serviços que provê as aplicações de softwares e processos de negócio na forma de serviços; o Consumidor de Serviços (usuário, aplicação ou outro serviço) que localiza e invoca os serviços conforme sua necessidade; e o Diretório de Serviços que fornece uma forma de localização dos serviços por meio de suas descrições e interfaces.

2.2.1 Serviços Web

Um serviço é um componente de software caracterizado como uma implementação bem definida de uma funcionalidade de negócios e com uma interface que pode ser publicada e descoberta por consumidores de serviços. Estes componentes podem ser agrupados para a criação de diferentes aplicações e processos de negócios utilizando-se de um modelo de comunicação baseado na troca de mensagens com baixo acoplamento (O'Brien et al., 2005).

Os serviços web, por sua vez, são vistos como sistemas de software, ou programas de computador, que buscam a interoperabilidade através da interação entre um computador e uma rede. Estes serviços se comunicam entre eles e os sistemas por meio de mensagens baseadas no protocolo HTTP (*HiperText Transfer Protocol*) e possuem três elementos principais: o protocolo de comunicação, a descrições dos serviços e a descoberta de serviços. (Haas e Brown, 2004). Segundo Paurobally e Jennings (2005), os serviços web trouxeram um novo paradigma suportando sistemas distribuídos fracamente acoplados com a descoberta e execução de serviços.

A base para esses três elementos é a linguagem XML (*Extensible Modeling Language*). O protocolo de comunicação entre os serviços web e seus clientes é o SOAP (*Simple Object Access Protocol*), o WSDL (*Web Services Definition Language*) é a linguagem que provê um mecanismo para descrever formalmente os serviços web, utilizando-se de *XML Schema* para especificar a estrutura e os tipos dos dados que estão contidos nas mensagens e, por fim, o UDDI (*Universal Descripition, Discovery and Integration*) é o diretório de registro de serviços, acessados por meio de mensagens SOAP (Wang et al., 2006).

A linguagem XML é uma linguagem de marcação com propósito geral que pode ser utilizada para a criação de linguagens de marcação com propósitos específicos, chamadas de dialetos. Com a linguagem XML é possível representar diferentes tipos de dados, além de fornecer

recursos para combinar textos e informações extras sobre estes. Isso significa que trechos do texto podem conter informações de como um trecho específico do texto deve ser representado. A especificação 1.0 da XML apresenta uma sintaxe que define uma forma de se criar linguagens específicas chamada DTD (*Document Type Definition*), ou Definição de Tipo de Documento. Outra forma de definição da sintaxe é a *XML Schema*, mais atual e com formas mais refinadas de representação das informações (Bray et al., 2004).

A linguagem *XML Schema*, por sua vez, se utiliza da linguagem XML para a criação de esquemas de validação para XML. Um documento *XML Schema* é constituído por declarações de elementos, atributos e tipos. A cada elemento ou atributo é associado um tipo, o que pode ser declarado juntamente com o elemento, como no caso dos elementos mensagem e destinatário ou referenciando-se um tipo declarado separadamente através de um atributo *type*. Este tipo pode ser declarado no mesmo documento ou importado de outro documento *XML Schema*, podendo ser tipos simples ou complexos (Thompson et al., 2004).

2.2.1.1 Protocolo de Comunicação SOAP

O SOAP, originalmente chamado de *Simple Object Access Protocol*, é um protocolo de comunicação leve que tem como objetivo trocar informações estruturadas em ambientes distribuídos e descentralizados. Este permite que aplicações façam interações complexas a partir da combinação de um conjunto de mensagens. Porém, ele é um paradigma de troca de mensagens em via única e que não mantém seu estado (Gudgin et al., 2007).

Inicialmente o SOAP foi projetado para prover serviços RPC (*Remote Procedure Call*) escritos em XML. A ideia era utilizar tecnologias já existentes mantendo o máximo de simplicidade possível (Swithinbak et al., 2005). Contudo este protocolo passou a ser utilizado de forma mais simples e leve para a troca de mensagens XML sobre o protocolo HTTP na web (Singh e Huhns, 2005).

É importante observar que inicialmente SOAP significava Protocolo Simples para Acesso a Objetos, mas ele passou a ser apenas um nome em versões mais recentes. Isso se deu porque ele foi considerado um acrônimo incorreto se comparado com a funcionalidade atual do protocolo (Gesser, 2006).

A estrutura hierárquica das mensagens SOAP, ou Envelopes SOAP, é constituída por dois elementos básicos, o cabeçalho *(Header)* e o corpo *(Body)*. O cabeçalho é opcional e suas informações não fazem parte da carga útil do envelope. Seu conteúdo é basicamente informações de controle que podem ser inspecionados ou alterados no percurso que a mensagem faz na rede por diferentes nós. Já o corpo é obrigatório e possui a carga útil da mensagem. Seu conteúdo pode ser um conjunto de elementos arbitrários ou pode ser a representação de uma RPC. Ainda, o corpo pode conter um elemento de exceção *(Fault)* para os casos da mensagem representar uma resposta com informações sobre um problema ocorrido (Gesser, 2006).

Os tipos de nós pelas quais as mensagens podem passar são o "Emissor", os nós "Intermediários" e o "Receptor Final" para a qual a mensagem é destinada. Por sua vez, os conteúdos do cabeçalho podem se destinar não apenas ao receptor final, mas também aos nós intermediários (Gesser, 2006).

Entretanto, a especificação do SOAP não se refere a nenhum protocolo de mensagem, se atendo apenas a sintaxe com a qual as mensagens devem ser estruturadas. Assim, o SOAP deve ser usado juntamente com algum protocolo de transporte, tal como o protocolo HTTP, um dos mais utilizados para o SOAP. Porém, outros protocolos de transferências de dados podem ser também utilizados (Gesser, 2006).

Por mais que o SOAP tenha sido amplamente adotado para a comunicação de serviços web, sendo considerado como um padrão *de facto*, as implementações de tal especificação nem sempre são desenvolvida da mesma forma e os fabricantes ainda podem inserir em seus motores SOAP detalhes e implementações proprietárias. Tal problema de interoperabilidade tem sido resolvido através das recomendações do *Basic Profile* (Piazza, 2007).

2.2.1.2 Descrição de Serviços Web via WSDL

O WSDL (*Web Service Description Language*) é uma linguagem que descreve componentes de softwares na forma de serviços web distribuídos com seus detalhes (localização, forma de invocação, etc.) definido em uma gramática XML (Christensen et al., 2001). Lindemann (2006) afirma que o conceito para a definição de interfaces não é específico do WSDL e já era utilizado por tecnologias tais como CORBA (*Common Object Request Broker Architecture*) e DCOM

(*Distributed Component Object Model*) que se utilizavam da IDL (*Interface Definition Language*). Contudo, o WSDL é construído em cima da linguagem XML, possibilitando a validação de um documento WSDL através de um processador XML.

A linguagem para descrição de serviços web, WSDL, é derivada do XML e se diferencia das outras linguagens de descrição de interface, tal como a IDL (*Interface Description Language*) da OMG (OMG, 2002), pelo seu nível de extensibilidade.

A linguagem WSDL provê uma linguagem comum para descrever os serviços e o meio para integrá-los de forma automática, caracterizando-se como uma estrutura de grande importância para o funcionamento dos serviços web (Cerami, 2002).

Christensen et al. (2001) apresenta os elementos WSDL que descreve um serviço Web em conformidade com as seguintes especificações:

- ***definitions****:* Elemento raiz de um documento WSDL.
- ***types****:* Definições de tipos de dados, conforme algum esquema.
- ***message****:* Mensagem representando uma requisição ou uma resposta.
- ***portType****:* Agrupa mensagens em operações em uma interface abstrata, que na nomenclatura WSDL se chama tipo de port.
- ***binding****:* Liga um tipo de *port* a um protocolo e um formato de dados específicos.
- ***service****:* Declara um serviço, definido por *ports*, que são combinações de uma ligação com um endereço de rede.

A WSDL permite descrever pontos de acesso e suas mensagens independente do formato da mensagem ou do protocolo de rede utilizado, permite tratar as mensagens como descrições abstratas dos dados sendo trocados e permite agrupar os conjuntos das operações na forma de um tipo abstrato, a ser mapeado posteriormente para um protocolo e tipo de dados concretos (Chappell e Jewell, 2002).

2.2.1.3 Diretório de Registro de Serviços UDDI

A UDDI (*Universal Description, Discovery and Integration*) é uma especificação OASIS que tem o objetivo de definir um conjunto de serviços para suportar a descrição da descoberta de serviços, negócios, outros provedores de serviços e as interfaces para o acesso aos serviços,

ou seja, é uma forma padrão de publicar e localizar negócios e interfaces de seus serviços em um registro (Bryan et al., 2002), (Clement et al., 2004).

O UDDI fornece um mecanismo funcional padrão interoperável para ambientes de softwares baseados em serviços web para classificação, catalogação e gerenciamento destes serviços. Por meio do UDDI, serviços podem ser descobertos e utilizados por outras aplicações, tanto para serviços de acesso público como para serviços disponíveis internamente dentro de uma organização. Um provedor de serviços pode utilizar o UDDI para publicar seus serviços de forma que um consumidor de serviços possa encontrá-lo e utilizá-lo conforme suas necessidades. Por conseguinte, o consumidor obtém os metadados necessários para a invocação do serviço (Bryan et al., 2002).

O UDDI é implementado tomando como base a linguagem XML para a descrição da estrutura do documento e o protocolo SOAP para a comunicação entre os clientes e o registro (Swithinbak et al., 2005).

Consoante Bryan et al. (2002), as informações de um registro UDDI podem proporcionar pesquisas em: "páginas brancas", retornando informações básicas tais como endereço, contatos e identificadores de uma organização e seus serviços; "páginas amarelas", com informações em conformidade com uma categoria e taxonomia; e "páginas verdes" com as informações técnicas da descrição e forma de como executar um serviço web.

Um registro UDDI pode ser público ou privado e a sua estrutura é composta de cinco tipos de dados que visa facilitar a rápida localização e entendimento dos diferentes tipos de informações que podem estar contidas no registro (Bellwood et al., 2002).

Conforme Ort e Mandava (2002) e Cerami (2002), estas estruturas de dados, citadas a seguir, têm o objetivo de serem utilizadas para a realização dos negócios e de serem acessíveis por identificadores únicos (*keys*): a estrutura chamada "*businessEntity*" é relativa as informações não técnicas conhecidas sobre o negócio (nome, descrição, endereço e informações do contato) e os serviços oferecidos; o "*businessService*" possui informações que descrevem um único serviço web (ou um conjunto de serviços relacionados); o "*businessTemplate*" possui informações técnicas sobre como acessar um serviço web específico; o "*tModel*" fornece informações sobre a categoria do serviço, do comportamento, convenções utilizadas, especificações e padrões compatíveis; e, por fim, o "*publisherAssertion*" fornece um mecanismo básico para relacionar os elementos *businessEntity*.

Para Piazza (2007), um registro UDDI não é uma parte essencial para o funcionamento dos serviços Web, uma vez que ele não descreve um serviço individualmente, mas sim o WSDL do serviço. Contudo, quando muitas organizações estão interagindo em uma ou mais oportunidades de negócios, tornar seus serviços visíveis e públicos se torna de vital importância para este cenário.

2.2.2 Composição de Serviços

A utilização de um serviço bem construído e com uma interface semanticamente bem fundamentada traz muitos benefícios aos seus usuários. Porém, segundo Singh e Huhns (2005), o maior valor ao se trabalhar com serviços está, claramente, na estrutura flexível que permite a criação de novos serviços com a composição de outros serviços. Este novo serviço passa a ser visto como um serviço, com sua própria interface.

Por vezes, ao se trabalhar da composição de serviços, é importante se observar que tal composição pode se tratar da invocação de uma série de serviços e que tais serviços podem se referir a qualquer conjunto de aplicativos na forma de serviços com as funcionalidades desejadas. Em uma perspectiva espacial, para se alcançar a composição desejada, tais serviços podem estar distribuídos geograficamente (Singh e Huhns, 2005).

Por mais que a maioria das aplicações web sejam úteis, estas não são suficientes para justificar a utilização de serviços web. Isto pois se tratam de serviços simples e com interações meramente cliente-servidor. Contudo, aplicações mais poderosas que permitem novas utilizações dos serviços podem ser desenvolvidas pela combinação de serviços mais simples (Singh e Huhns, 2005).

A composição de serviços pode ser feita utilizando qualquer linguagem de programação que trabalhe com serviços web. Porém, existem algumas formas de se efetuar isso dinamicamente, e ainda, trabalhar integrado a processos de negócios da empresa, como por exemplo, o BPEL4WS.

2.2.2.1 Processos de Negócios

Em geral, um processo de negócios pode ser caracterizado por uma execução que pode demorar um certo tempo para ser efetuada. Este pode envolver diversos participantes e possuir a obrigatoriedade de conclusão garantida, incluindo implicações legais (Singh e Huhns, 2005).

A linguagem BPEL4WS (*Business Process Execution Language for Web Services*), ou Linguagem de Execução de Processos de Negócios para Serviços Web, é uma estrutura organizada em um documento XML para descrever a execução de processos de negócios, conforme uma gramática específica para descrever a forma como ocorrem as muitas interações entre os serviços web dos parceiros, participantes do processo (Singh e Huhns, 2005). A partir da versão 2.0 do BPEL4WS, este passou a ser chamado apenas de WS-BPEL (*Web Services Business Process Execution Language*).

Não é necessário que haja uma definição completa de um parceiro em particular na modelagem de um processo de negócio utilizando WS-BPEL, mas apenas a forma de interação com este de forma consistente e independente de plataforma de execução, ou linguagem de programação utilizada pela aplicação (Singh e Huhns, 2005).

A utilização do WS-BPEL para modelar processos de negócios empresarias provê um novo serviço web composto de serviços existentes. A partir deste é gerada uma interface WSDL tal como qualquer outro serviço web, e que pode ser também registrado em uma UDDI (Singh e Huhns, 2005).

Assim como a UML (*Unified Modeling Language*) fornece uma linguagem formal para representação e modelagem de softwares, os processos de negócios também podem ser modelados por meio do mapeamento de uma especificação formal, tal qual uma espécie de UML para o WS-BPEL. Com isso a BPML (*Business Process Modeling Language*), ou Linguagem de Modelagem de Processos de Negócios, suporta a modelagem de processos de negócios do mundo real com a utilização de uma semântica avançada para trabalhar com as transações necessárias e com base na WSCI (*Web Service Choreography Interface*), ou Interface de Coreografia para Serviços Web, para expressar interfaces públicas e coreografias dos serviços web (Singh e Huhns, 2005), (Arkin et al., 2002).

Em tempo, a especificação BPEL é especificamente para processos de negócios que interagem com serviços web. Contudo, para muitos processos empresariais é necessário que pessoas participem da execução dos processos. Esta participação das pessoas agrega novos aspectos as processos tais como interação entre o processo e o usuário e a interface deste com o usuário. Para tal, a especificação WS-BPEL *Extension for People*, ou simplesmente BPEL4PEOPLE introduz os elementos necessários para que a especificação BPEL possa ser ampliada de forma a permitira a modelagem de interações humanas (Clement et al., 2009).

Uma forma de especificação da forma como os processos são gerenciados, é por meio de uma especificação chamada BPM (*Business Process Management*), ou Gerenciamento de Processos de Negócios.

Para permitir que usuários empresariais desenvolvessem de forma mais fácil e graficamente os processos de negócios, foi desenvolvida a Notação de Modelagem de Processos de Negócios (*Business Process Modeling Notation – BPMN*). Um agrande vantagem desta notação é que por meio dela podem ser gerados executáveis BPEL (White, 2005). A Figura 8 apresenta um exemplo de documento com a especificação BPM.

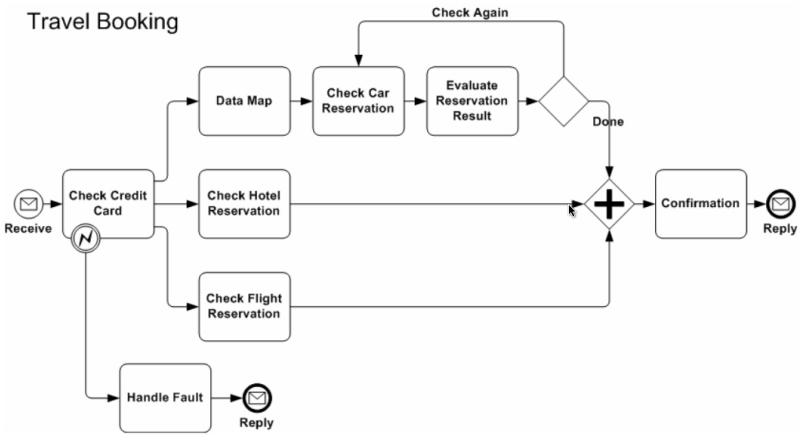

Figura 8: Exemplo de documento BPM (White, 2005).

O BPM se caracteriza como uma técnica para suportar processos de negócios de forma a especificar, controlar, executar e analisar os processos empresariais envolvendo pessoas, empresas, aplicações, documentos e outras formas de informações (Aalst et al., 2003), (Garimella et al., 2008). Nesse sentido, o BPM tem se tornado uma forte tendência (Baldam et al., 2007). Um dos motivos é porque o BPM melhor representa as ideias de gerenciamento de processos de negócios observadas atualmente (Garimella et al., 2008).

Uma combinação do BPM com SOA (Figura 9), tem sido defendida como uma melhor estratégia para aliar os processos de negócios aos recursos tecnológicos da empresa.

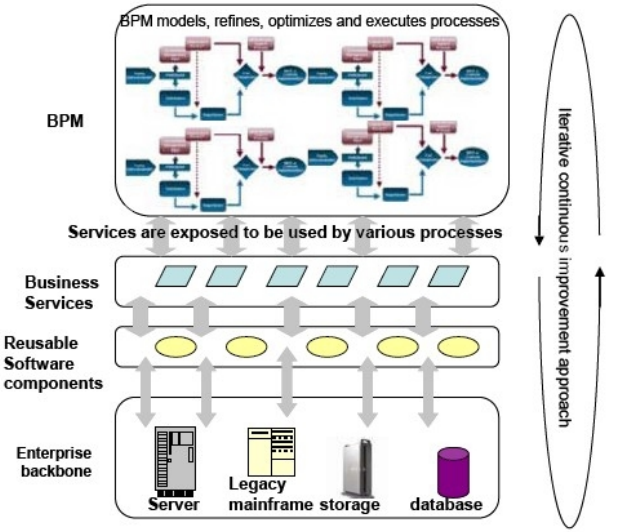

Figura 9: Combinação de BPM e SOA (Kaumon, 2007).

A união do BPM com SOA permite automatizar e otimizar os processos de negócios por meio de serviços web, que podem ser reutilizáveis (Kamoun, 2007). Além disso, pode trazer melhorias para o BPML ao fornecer as vantagens do fraco acoplamento do SOA para as aplicações de negócios e integração de sistemas (Pires, 2008).

2.2.3 Linguagem Universal de Negócios – UBL

Parceiros comerciais precisam trocar informações para que possam executar operações ou comércio eletrônico. Para tal, há a necessidade de uma padronização nos documentos a serem trocados entre essas empresas (Yarimagan e Dogac, 2008), (Figueiredo e Mayworm, 2004).

Porém, devido às limitações no EDI (*Electronic Data Interchange*), as empresas tiveram que definir e desenvolver formatos XML específicos para a troca de documentos em transações eletrônicas. Entretanto, tais documentos são difíceis de manter porque uma empresa está, em geral, envolvida em diversos contextos de negócios. Além disso, a existência de múltiplos formatos XML dificulta a integração (Bosak et al., 2006).

Para contornar tais limitações do EDI, é apresentada por Bosak et al. (2006) a *Universal Business Language* (UBL). A UBL (UBL, 2011) é uma linguagem baseada no padrão XML e que tem o propósito de descrever documentos trocados em transações eletrônicas. Esta define um conjunto de padrões voltados à negócios eletrônicos com o objetivo de aumentar a interoperabilidade através da padronização das mensagens em operações de comércio eletrônico. Dessa forma, uma vez que a UBL trafega sobre a Internet e os custos com a manutenção e configuração são baixos, isso aumenta a possibilidade da participação de Pequenas e Médias Empresas (PMEs) no comércio eletrônico.

Para a criação de um padrão internacional para comércio eletrônico, disponível gratuitamente e sem cobrança de licença, um comitê foi formado por membros de empresas como CommerceOne, Oracle, SAP, Sterling Commerce, Sun e outras, criando assim, uma biblioteca padrão de documentos XML para comércio eletrônico (Figueiredo e Mayworm, 2004).

Concomitantemente, em função de se apropriar das vantagens do *XML Schemas* com sua utilização, emergem características de reutilização, extensibilidade e modularidade. Assim, por meio da UBL é possível formatar toda e qualquer transação de comércio eletrônica de forma livre, comum e aberta. Dessa forma, se constituindo como uma forte tendência nas transações de comércio eletrônico (Bosak et al., 2006).

Contudo, as empresas operam em diferentes indústrias, localizadas geograficamente distantes. Dessa forma, existem diferentes regras e requisitos para trocas de informações, não sendo possível assim

suprir as necessidades das Pequenas e Médias Empresas (PMEs). Assim, espera-se que comunidades de usuários de UBL personalizem seus *UBL schemas* conforme suas necessidades. No entanto, utilizar esquemas sem um padrão torna mais difícil manter a interoperabilidade dentro da comunidade UBL (Yarimagan e Dogac, 2008).

Porém, existem esforços para ainda manter a compatibilidade entre os esquemas personalizados. Neste sentido, Yarimagan e Dogac (2008) propõe uma solução para este problema de interoperabilidade por meio de um mecanismo de tradução semântica que converte documentos entre esquemas personalizados para empresas de diferentes contextos. Assim, quando empresas de diferentes comunidades precisarem fazer negócios uns com os outros, eles continuam a usar os seus esquemas e a interoperabilidade é obtida pela tradução de documentos.

A especificação UBL, conforme vista com seus elementos básicos na Figura 10, possui uma representação gráfica para a identificação de cada forma de execução e interação entre as partes. O círculo pintado de preto é o início do processo e os círculos vazados com um círculo preto no meio são as finalizações dos processos. Neste exemplo, tem-se duas entidades envolvidas no processo, um provedor e um cliente. Estes estão separados por uma linha no meio da figura. As caixas centrais são os documentos (em formato XML, conforme XML *Schemas* UBL) trocados entre as partes envolvidas.

Os losangos se caracterizam como pontos de tomada de decisão e os retângulos com cantos arredondados são as tarefas a serem executadas. Por fim, as setas são o fluxo das informações. A Figura 10 é a especificação de um processo de compra UBL, com o comprador (*Buyer Party*) e o vendedor (*Seller Party*).

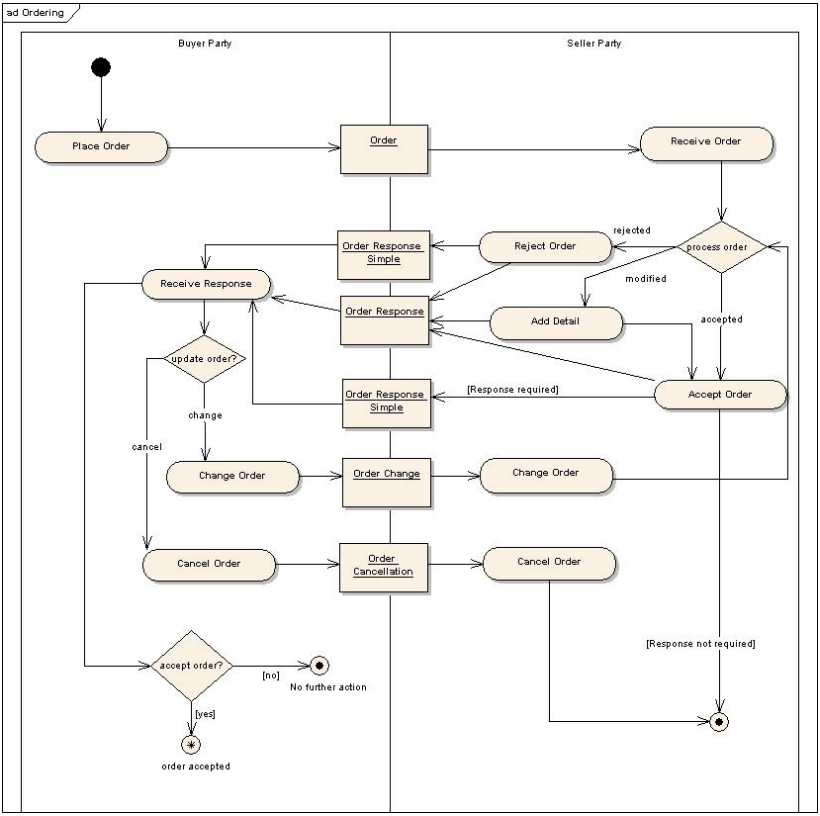

Figura 10: Especificação de um processo UBL - Processo de compra (Bosak et al., et al. 2006).

Este exemplo inicia com o comprador enviando uma ordem de compra para o vendedor. O vendedor recebe a ordem e a processa. Se a ordem foi rejeitada, envia uma resposta ao comprador avisando que a ordem foi rejeitada e termina o processo, ou tenta alterar a ordem, e assim por diante. Neste exemplo há a troca de mensagens entre o comprador e o vendedor, até que a compra/venda esteja finalizada, ou com o sucesso da venda, ou com o cancelamento da ordem.

2.3 AGENTES E SERVIÇOS WEB

Este subcapítulo se utiliza da relação de Agentes com serviços web para situar o leitor da utilização de serviços web para modelar a Arquitetura de Referência para SAPs.

Em verdade, a noção de agente, para Ricci et al. (2007), pode ser vista implicitamente no conceito da arquitetura orientada a serviço (SOA). Nesta, um serviço, que pode ser um agente, é solicitado por alguém, ou outro agente, que resulta na troca de mensagens em conformidade com a especificação da interface de serviço e um prestador de serviços encapsula a lógica empresarial no serviço, executando as atividades e interagindo via protocolo de troca de mensagens definida na descrição do serviço. Assim, são definidas entidades que atuam para a execução de determinada atividade ou para atingir algum objetivo. Por mais que se tratem de metodologias específicas, em um alto nível de conceituação, ambas as tecnologias tratam da noção de atividade, objetivo, tarefa e interação orientada a mensagens.

A tecnologia de agentes, ao ser aplicada em conjunto com a tecnologia e serviços web, permite resolver o problema de certas limitações das quais os serviços estão atrelados, tais como abstrair requisitos mais ricos de composição de serviços, descobrir serviços confiáveis, negociar entre um conjunto de outros fornecedores e fazer uma avaliação dos prestadores de serviços em relações às composições (Singh e Huhns, 2005).

Não obstante, Hunhs (2002) traz a tona a questão de que as funcionalidades de invocar, publicar e encontrar serviços web possuem certa equivalência às funcionalidades dos agentes (Figura 11). Contudo, nos agentes as interações são feita através da Linguagem de Comunicação de Agentes – ACL (*Agent Communication Language*).

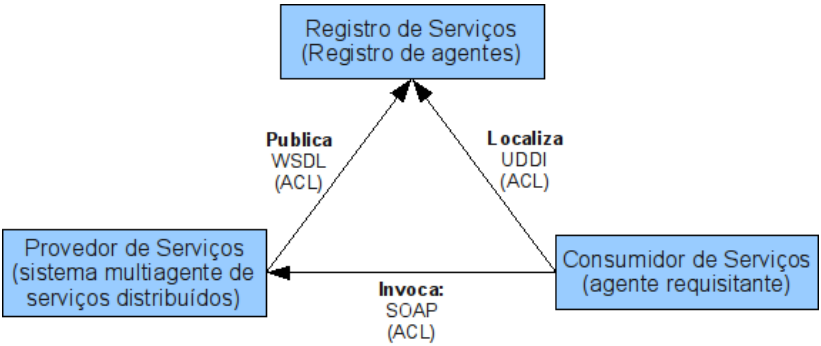

Figura 11: Equivalência do modelo conceitual de SOA com agentes (Hunhs, 2002).

Dessa forma, ambas as tecnologias, SOA e agentes, possuem certas características correlatas, tais como páginas brancas e páginas amarelas para registrar suas funcionalidades. Porém, segundo Hunhs (2002), os agentes possuem certas especificidades que vão além das características dos serviços web em vários aspectos:

- Um agente é, em geral, autoconsciente e possui metas. Por meio da aprendizagem e comportamento passa a conhecer outros agentes e suas capacidades para que as interações ocorram. Diferentemente, um serviço web apenas conhece a si mesmo, mas não seus usuários, ou clientes.
- Serviços web não são desenvolvidos em conformidade com a utilização e conciliação de ontologias, ao contrário dos agentes. Neste caso, se um cliente e um fornecedor utilizarem ontologias diferentes, o serviço a ser invocado pode ser incompreensível para o cliente.
- Os serviços web são aplicações passivas e iniciam seu funcionamento apenas quando são invocados enquanto os agentes permanecem em execução, podendo fornecer atualizações, alertas de novas informações, etc.
- Um agente é autônomo, enquanto um serviço web não. Tal autonomia em geral refere-se a autonomia social com um agente conhecendo seus parceiros, mas com atividades independentes em certas circunstâncias, de forma a alcançar seus objetivos.
- Os agentes podem formar equipes de forma a cooperar e fornecer uma forma de serviço mais abrangente em um nível mais alto, os serviços web não. Ainda, que os serviços web possam ser utilizados para compor outros serviços, estes não o

fazem autonomamente, precisam ser invocados pelo cliente/usuário.

Por outro lado, pode haver a interação dinâmica de agentes e serviços web por meio da utilização de serviços web pelos agentes. Porém, consoante Kuno e Sahai (2002), para que seja possível tal interação, devem haver três características fundamentais:

1. O agente deve ser capaz de descobrir os serviços web necessários conforme preferências e necessidades do usuário, adaptando dinamicamente seu comportamento com base nas características do serviço.
2. O serviço web deve descrever suas interfaces e resumo do protocolo de invocação para que um agente possa invocá-lo.
3. O agente deve ser capaz de realizar interações complexas com vários serviços ao mesmo tempo, agindo em nome de um usuário. Isso inclui uma monitoração contínua das interações para o caso de ocorrerem exceções.

Enquanto os serviços web se estabeleceram como uma forma bastante eficiente de conectar e executar programas remotamente via Internet, utilizando protocolos amplamente utilizados na atualidade, os agentes se fixaram na computação como uma tecnologia para resolver problemas complexos. Para Greenwood e Calisti (2004), desenvolvedores podem encontrar e utilizar agentes, por meio de invocação, como se fossem serviços web com a utilização de uma plataforma de agentes com capacidade intrínseca de invocar serviços web, tal como o JADE[9], especificamente. Tal integração traz benefícios imediatos em que torna um serviço web capaz de invocar um agente de serviço e vice-versa, permitindo assim, através dos conceitos e tecnologias de agentes, novas e avançadas funcionalidades na utilização de serviços web.

Greenwood e Calisti (2004) apresenta uma arquitetura chamada *Web Service Integration Gateway Service* (WSIGS). Conforme apresentado na Figura 12, esta arquitetura contém diversos componentes, incluindo um *FIPA Compliant Directory Facilitador* (DF), da FIPA (*Foundation for Intelligent Physical Agents*), e um conjunto de serviços web compatíveis com repositórios UDDI.

[9]O JADE (*Java Agent Development Framework*) é um *framework* implementado em linguagem Java para simplificar a implementação de sistemas multiagentes através de um *middleware* de acordo com especificações FIPA e com um um conjunto de ferramentas gráficas que suporta a depuração e implantação fases (JADE, 2011).

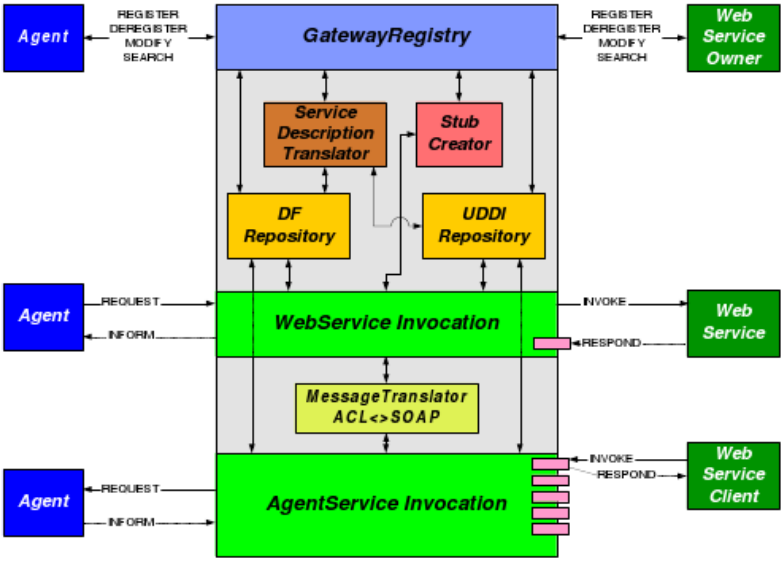

Figura 12: Visão funcional da arquitetura WSIGS de Greenwood (2004).

Os registros internos do núcleo da arquitetura WSIGS contem as descrições dos agentes na forma de serviços web, assim como as descrições para a invocação destes. Dessa forma, os agentes são vistos como serviços web com interfaces adequadas com o padrão.

2.4 ANÁLISE GERAL DO ESTADO DA ARTE

Esta seção apresentou uma revisão bibliográfica de assuntos relacionados ao fornecimento de assistência às pessoas, via software de computador, em suas atividades diárias nas empresas. Algumas pesquisas sobre o assunto, e projetos na forma de softwares prontos foram encontrados de forma a contribuir com o presente trabalho, fornecendo fundamentação da situação atual do desenvolvimento de Softwares Assistentes Pessoais. O objetivo aqui é fazer uma análise para se abstrair algumas ideias e resultados (quando pertinentes) a fim de adaptá-los ao modelo e arquitetura sendo proposta neste trabalho e não responder a uma pergunta de pesquisa.

Significantes contribuições podem ser tiradas de Bocionek (1994) e Hoyle e Lueg (1997) na parte conceitual sobre assistentes pessoais, sendo comparados secretários humanos, autonomia, atividades de assistentes pessoais citadas de forma conceitual.

Huhns e Singh (1998) fazem um paralelo de assistentes pessoais e agentes, trazendo os conceitos de agentes como forma de estender os assistentes pessoais para uma tecnologia já existente e já bastante madura tecnologicamente. Ainda, eles apresentam um cenário bastante interessante desses assistente trabalhando em prol do usuário com a utilização de recursos disponíveis na Internet. Tais conceitos se aplicam de forma bastante interessante à proposta do presente trabalho.

Outros autores também levam os assistentes pessoais o mundo dos agentes de software, ou mesmo apresentam conceitos que podem ser enquadrados aos assistentes pessoais, tais como Weiss (1999), Wong e Mikler (1999), Sensoy e Yolum (2008), Argehrn (2001), Bush et al. (2006) e Schiafino (2006). Inclusive, há a utilização de tecnologias de sistemas multiagentes por Carrillo-Ramos et al. (2005) para assistentes em dispositivos móveis.

Também pode-se abstrair algumas contribuições muito interessantes baseadas nas ideias de produtos na área de assistência pessoal, como por exemplo, o software Sandy (Mann, 2011), que trabalha com comunicação com o usuário via mensagens de Twitter, e-mail e SMS. Com isso, o usuário não precisa de uma GUI especifica para interagir com seu assistente pessoal, mas pode utilizar ambientes das quais ele já está acostumado, ou utiliza no dia a dia para se comunicar com outras pessoas.

Algumas ideias retiradas do projeto Narval (Chauvat, 2000), (Thenault, 2011) também podem ser utilizadas como fonte inspiradora na proposta. No projeto existe um núcleo central, que executa ações com base em um conjunto de informações na forma do comportamento do assistente pessoal, e novos *plug-ins*, com novas funcionalidades – a chamar aqui de comportamento – podem ser acrescidas ao software por meio da linguagem Python.

Ainda, o software Siri, da Apple, que trabalha se comunicando com o usuário por meio de mensagens via celular e que aos poucos, conforme histórico e informações que o usuário passa a ele, vai aprendendo a fazer melhores escolhas para auxiliar o usuário.

O projeto PAL (2011) do DARPA, que envolve diversos pesquisadores, teorias e técnicas da ciência da computação para produzir assistentes pessoais inteligentes e flexíveis.

Por fim, e tal como essa Tese sugere, a utilização da Arquitetura Orientada a Serviços como estilo arquitetural da Arquitetura de Referência para Softwares Assistentes Pessoais, em conjunto com um Modelo de Referência para Assistentes Pessoais baseado nas teorias de Assistentes Pessoais pesquisados, se apresenta como uma forma bastante atraente. Isso pois o SOA apresenta diversas vantagens no sentido da interoperabilidade, distribuição de processamento, baixo acoplamento, independência de implementação, configuração flexível, granularidade, entre outras (Singh e Huhns, 2005).

3 PROPOSTA

Conforme apresentado no primeiro capítulo, o objetivo deste trabalho é conceber uma arquitetura de referência para assistentes pessoais com base, mais especificamente, na arquitetura orientada a serviços. Esta seção descreve a proposta da arquitetura e cada um dos elementos que a compõe.

A concepção de uma arquitetura de referência é vista como uma forma de apresentar um padrão genérico para um projeto. Esta deve abordar os requisitos para o desenvolvimento de soluções, guiado pelo modelo de referência e por um estilo arquitetural (Figura 13) de forma a atender as necessidades do projeto (MacKenzie et al., 2009).

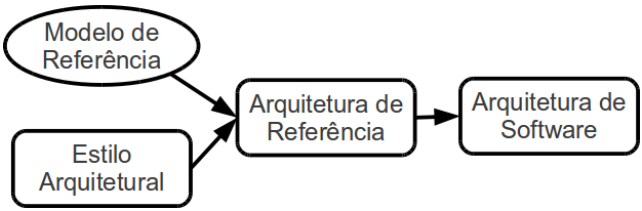

Figura 13: Relação entre modelo de referência, estilo arquitetural, arquitetura de referência e arquitetura de softwares (Bass et al., 2003).

Para compor uma arquitetura de referência é necessário apresentar os tipos dos elementos envolvidos, como eles interagem e o mapeamento das funcionalidades para estes elementos (Hofmeister et al., 2000).

O estilo arquitetural descreve os tipos de elementos e suas relações, juntamente com um conjunto de restrições sobre como eles podem ser utilizados, além de padrões de interação entre os elementos. Tais restrições, sobre a arquitetura e sobre o sistema em si, são vistas na forma de uma imagem da utilização do sistema como um todo. Por exemplo, o estilo arquitetural cliente-servidor utiliza dois tipos de elementos, o cliente e o servidor, e sua coordenação é descrita em termos de protocolos de comunicação entre eles (Bass et al., 2003).

Neste ponto, os requisitos gerais desejáveis identificados na seção 2.1 (autonomia, flexibilidade, adaptabilidade ao usuário e ao contexto, interatividade, conectividade à rede, interoperabilidade e generalidade de propósito) são rebuscados para resolvê-los em pontos específicos da proposta, do modelo de referência até a arquitetura de software (ou

arquitetura de referência para implementação). Inicialmente, alguns desses requisitos são selecionados e agrupados para serem resolvidos no modelo de referência, outros são resolvidos na arquitetura de referência, juntamente com a utilização do estilo arquitetural selecionado (SOA) e outros são resolvidos nos momentos seguintes, por meio da proposta abrangendo a possibilidade de resolução dos requisitos restantes.

Como ponto de partida para a concepção de uma Arquitetura de Referência para Softwares Assistentes Pessoais foi definido que a arquitetura deve ser baseada em padrões abertos e atuais, de modo a fornecer flexibilidade nas funcionalidades dos assistentes pessoais e interoperabilidade na comunicação entre os elementos envolvidos no auxilio à realização das tarefas dos usuários. Também não foi definida uma linguagem de programação específica, sendo que cada desenvolvedor pode ter a liberdade de escolher suas ferramentas de desenvolvimento do assistente pessoal, ou das suas funcionalidades, contanto que utilizem os padrões abertos aqui escolhidos. Para comportar tais características, o presente trabalho se apropria dos conceitos e características de SOA, apresentados por Singh e Huhns (2005), Dudley (2007), Stojanovic e Dahanayake (2005), Booth et al. (2004) e Estefan et al. (2008), e explicados no capítulo anterior. Dessa forma, o presente trabalho define o SOA como estilo arquitetural da proposta.

Ainda, para que um assistente pessoal possa fornecer assistência às pessoas em um certo nível bastante genérico é necessário que tal software possa comportar diversos tipos de funcionalidades de auxilio (Bocionek, 1994), (Hoyle e Lueg, 1997), (Huhns e Singh, 1998), e que cada usuário possa escolher as funcionalidades do seu assistente pessoal (Zambiasi e Rabelo, 2010). Dessa forma, é de interesse que as funcionalidades, ora chamadas de comportamentos, possam ser escolhidas pelo usuário e apenas as que estejam em conformidade com suas necessidades. Com isso, pode haver uma grande quantidade de funcionalidades, e o usuário pode selecionar as que convêm a ele e, adicioná-las como *plug-ins* ao seu assistente pessoal, tal como o trabalho de Chauvat (2000).

Para que o assistente pessoal possa ter essa característica de comportamentos “plugáveis”, se manter em um padrão de comunicação e ainda trabalhar com SOA, conforme já explicado acima, a tecnologia de softwares como serviço – *SaaS* – (O'Brien et al., 2005) , (Haas e Brown, 2004) , (Paurobally e Jennings , 2005), (Gesser, 2006), (Thompson et al., 2004) se enquadra de forma bastante suficiente.

Assim, os comportamentos do assistente pessoal são vistos, no presente trabalho, como serviços de software, distribuídos na Internet, na forma de uma federação de serviços (Zambiasi e Rabelo, 2010). O usuário pode então escolher os comportamentos – serviços de software – que melhor se enquadram nas necessidades das tarefas que se quer auxílio. Nesse sentido, os comportamentos que o assistente possui, devem poder ser acrescentados ou retirados de forma flexível. Ainda, isso dá ao usuário do assistente pessoal, a possibilidade de escolha entre comportamentos com a mesma funcionalidade, fornecido por diferentes empresas ou organizações ou desenvolvedores. Quando um comportamento não mais satisfizer as necessidades do usuário, este pode então trocá-lo por outro, que seja mais condizente com sua nova situação/contexto. Tais comportamentos devem poder ser utilizados gratuitamente, alugados ou vendidos por terceiros.

A utilização de SOA e dos comportamentos na forma de serviços de software, também fornece a vantagem da distribuição de processamento (Singh e Huhns, 2005), ou seja, o assistente pessoal não precisa se situar obrigatoriamente no mesmo dispositivo computacional que os seus comportamentos. Dessa forma, diversas operações podem ser efetuadas em paralelo, em computadores separados, liberando a carga de processamento do núcleo central do assistente pessoal.

É importante frisar novamente que seguir uma arquitetura de referência, juntamente com esses direcionamentos para o desenvolvimento de assistentes pessoais, trazem certas vantagens que são de grande valor para os usuários e para o mundo globalizado das empresas hoje, com modelos de execução de processos de negócios distribuídos, rede de fornecedores, compra/venda via Internet, etc. Como vantagens encontradas, pode se citar a interoperabilidade, a flexibilidade das funcionalidades do assistente pessoal, a independência de fornecedores de assistentes pessoais e funcionalidades e a possibilidade de que qualquer pessoa possa ter seu assistente pessoal, com os mais variados tipos de funcionalidades, que podem ser encontradas na internet, com o comportamento ao gosto e necessidade de cada usuário.

Antes de prosseguir com a apresentação da Arquitetura de Referência para Softwares Assistentes Pessoais, é relevante que seja mostrado o modelo de referência que serve de base, juntamente com o estilo arquitetural, para então seguir para a arquitetura de referência, propriamente dita.

Dessa forma, neste capítulo é inicialmente apresentado um modelo de referência que serve de norteador para a arquitetura de referência. Em seguida, é apresentada a arquitetura de referência e os seus elementos integrantes, de forma genérica.

3.1 MODELO DE REFERÊNCIA

Um modelo de referência é caracterizada por ser uma abstração da realidade, que pode ser expressa por um formalismo de um método de modelagem, conforme os objetivos de um usuário (Vernadat, 1996).

> Um modelo de referência é um *framework* abstrato para entendimento dos relacionamentos significantes entre as entidades de algum ambiente. Ele habilita o desenvolvimento de arquiteturas específicas usando padrões consistentes ou especificações suportando aquele ambiente. Um modelo de referência consiste de um conjunto mínimo de conceitos unificados, axiomas e relacionamentos com um domínio de um problema particular, e é independente de padrões específicos, tecnologias, implementações, ou outro detalhe concreto (MacKenzie et al., 2006, 2009).

Para Bass et al. (2003), um modelo de referência é uma divisão de funcionalidades, juntamente com o fluxo de dados entre as partes. Ele se caracteriza como um padrão de decomposição do problema. Os modelos de referência possuem características de domínios maduros, decorrente da experiência sobre este domínio.

No modelo de referência é proposto um vocabulário e um entendimento comum sobre o que são os elementos específicos do domínio trabalhado. Este contém uma normativa na forma de um modelo abstrato (MacKenzie et al., 2006, 2009).

Nesta seção é apresentado um modelo de referência para assistentes pessoais. Contudo, ainda mantendo o enfoque no conceito de assistentes pessoais firmado neste trabalho em que um assistente pessoal se caracteriza como:

> …um conjunto de processos computacionais trabalhando em conjunto e criados para

> representar um usuário na execução de certas tarefas, automaticamente ou com algum grau de intervenção/supervisão humana (Zambiasi e Rabelo, 2010).

Para apresentar o modelo de referência, primeiramente separam-se os requisitos desejáveis para assistentes pessoais que devem se enquadrar neste ponto, identificados na seção 2.1. Isso pois alguns deles são resolvidos apenas em nível de implementação ou em nível da utilização do estilo arquitetural.

A autonomia é, de certa forma, genérica e está intrinsecamente ligada às ações que um SAP deve executar. A interatividade é requisito geral e que deve ser já abstraído neste ponto. Para fazer uma ligação da autonomia com a interação, um módulo de gerenciamento também é identificado, fazendo a integração entre estes dois outros elementos.

Flexibilidade, adaptabilidade ao usuário e ao contexto, e generalidade de propósito são características que podem tomar diversos caminhos distintos neste ponto. Dessa forma, eles são deixados para se solucionar nas próximas etapas. Interoperabilidade e conectividade à rede são em nível de tecnologia e também são resolvidos mais a frente.

A Figura 14 ilustra os principais conceitos identificados acima e são definidos no modelo de referência de SAPs deste trabalho.

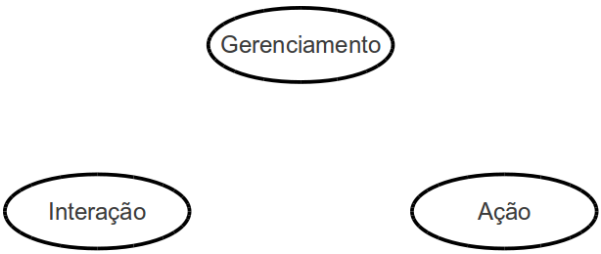

Figura 14: Modelo de Referência de SAPs.

A notação trabalhada neste modelo de referência segue o estilo da notação apresentada no modelo de referência de SOA, apresentado por MacKenzie et al. (2006). Para apresentar o modelo de referência visualmente são utilizados mapas de conceitos. Neste, uma linha entre dois conceitos representa um relacionamento, não havendo um nome, mas uma descrição de como eles estão relacionados. A seta indica uma relação assimétrica, em que um conceito é dependente de outro da qual está relacionado. Em verdade, não há uma convenção normativa para

interpretar mapas de conceitos. Estes são uma forma visual para complementar a explicação textual (MacKenzie et al., 2009).

Comparativamente, pode-se fazer um paralelo com a ideia de agentes. Um agente percebe o ambiente por meio de sensores e responde a esse ambiente por meio de atuadores. Internamente no agente, existe a deliberação, ou seja, o que fazer com as informações de entrada e com as informações que estão armazenadas na forma do "conhecimento" do Agente. Dessa forma, o agente pode agir de forma a alcançar seus objetivos. Um agente também pode interagir com outro agente ou sistemas para satisfazer seus objetivos (Russel e Norvig, 2004), (Weiss, 1999).

Assim, poder-se-ia definir um SAP na forma de um agente. Entretanto, isso limitaria a implementação a ser direcionada a apenas SAPs como agentes, e na verdade, não obrigatoriamente tal problema pode ser resolvido apenas com agentes. Portanto, na proposta abre-se essa escolha para o nível da implementação, ou da Arquitetura de Referência para Implementação". Neste ponto são definidos os itens que podem permitir a solução do resto dos requisitos identificados, e de forma que a arquitetura seja ampla o suficiente para permitir vários estilos de tecnologias de implementação, mas que possam conversar entre si pelos elementos definidos na arquitetura. Assim, abandona-se aqui a obrigatoriedade da utilização da tecnologia de agentes no nível da arquitetura de referência para SAPs.

Entre os elementos principais estão a Ação, que se refere aos comportamentos do assistente, a Interação para com o usuário ou outros assistentes ou softwares e o Gerenciamento, para gerenciar a execução do assistente, organizar as informações e o fluxo dessas entre os demais elementos.

3.1.1 Ação

A **Ação** é aquele elemento da qual se refere a forma como o assistente deve agir para auxiliar o usuário, e corresponde as atividades definidas por Bocionek (1994), Bush et al. (2006), Franco et al. (2007), Huhns e Singh (1998). Este elemento de atividade se refere ao comportamento do assistente propriamente dito (Figura 15). Contudo, a ação se subdivide em um conjunto de atividades do SAP, conforme os autores citados. Assim, o Elemento **Ação** pode se caracterizar como a composição do conjunto de *n* ações que podem ser utilizadas pelo

assistente pessoal para que este possa cumprir com seus objetivos e com os objetivos do seu usuário.

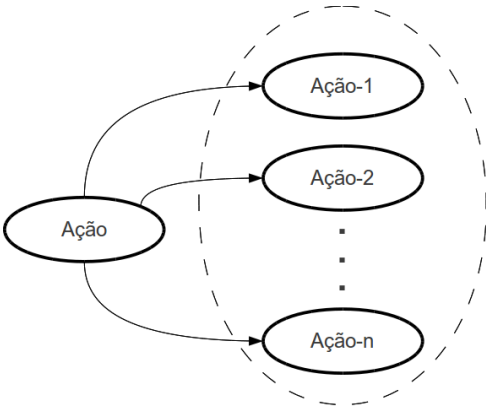

Figura 15: Conjunto de Ações que compõem a Ação do Assistente Pessoal.

Para que um SAP possa tomar iniciativa, por meio do elemento ação do modelo de referência, ou mesmo reagir a uma determinada entrada de informações, é necessário que o assistente possua um conjunto de módulos de execução. Na Figura 16, o elemento **Ação** do modelo de referência é detalhado em subelementos de grande relevância para que o comportamento do SAP possa se tornar efetivo e operacional.

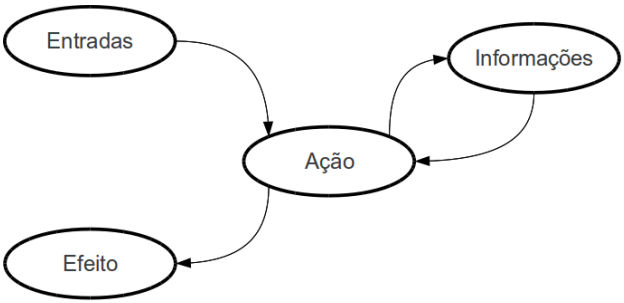

Figura 16: Visão do subelemento da Ação do Modelo de Referência para Softwares Assistentes Pessoais.

Por **Informações**, tem-se o conjunto de dados ou a base de conhecimento armazenados no SAP de suporte à execução de uma determinada ação do SAP. A relevância e importância dessas

informações, (Michaell et al., 1994), (Bush et al., 2006), (Huhns e Singh, 1998), (Schiaffino e Amandi, 2006), está diretamente ligada a sua utilização para a execução de cada ação. O item **Entradas** refere-se ao conjunto de informações que chegam de elementos externos ao assistente e que são suficientes para iniciar uma determinada ação do SAP, assim como agentes sentem o seu ambiente por sensores (Huhns e Singh, 1998), (Russel e Norvig, 2004). Essa ação, por fim, reage com um **Efeito**, que pode ser tanto interno do próprio SAP como externo, com parceiros, outros SAP, outros softwares, ou mesmo na forma de interação do usuário. Isso é correspondente com a ideia da ação dos agentes no ambiente, de Huhns e Singh (1998) e Russel e Norvig (2004).

3.1.2 Interação

A **Interação**, conforme subelementos explodidos do elemento Interação na Figura 17, é a forma como se dá a comunicação e negociação entre Usuário e módulo de Gerenciamento do assistente, ou entre o módulo de Gerenciamento do assistente e terceiros (pessoas, softwares, outros assistentes, etc.). Isso condiz com as teorias da interação com os assistentes pessoais e usuário, ou outros elementos (Bocionek, 1994), (Bush et al., 2006), (Franco et al., 2007) , de forma a alcançar os objetivos do usuário (Levin et al., 2000) .

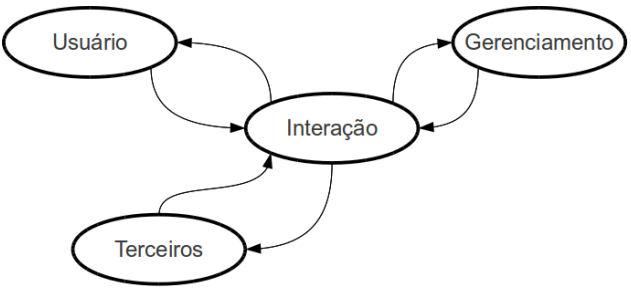

Figura 17: Explosão do elemento Interação do assistente em subelementos.

Para que o usuário possa interagir com o SAP, é necessário que esse faça uso de recursos fornecidos pelo conjunto de elementos do SAP específicos para a interação com o usuário. Esses recursos visam

fornecer facilidades para o lado do Usuário enviar comandos ao SAP como receber informações e relatórios do andamento e do resultado das ações executados pelo SAP. Esse elemento é também responsável por facilitar o acesso do assistente ao usuário, de forma a iniciar a comunicação entre o SAP e o usuário quando necessário para o cumprimento dos objetivos (Levin et al., 2000), (Huhns e Singh, 1998) . Essa interação também deve ser suficiente para resolver conflitos com terceiros, também visando o cumprimento dos objetivos do usuário do assistente ou do assistente propriamente dito.

3.1.3 Gerenciamento

O **Gerenciamento** é o mecanismo responsável pela organização das informações do usuário, das ações e do fluxo dessas informações entre os demais elementos do modelo, tal como comportamentos distribuídos (Brooks, 1991), (Brooks, 1989). Este é também responsável pela organização da chamada de execução das ações, a ordem temporal como essas ocorrem e possíveis conflitos que possam haver entre esses elementos.

A Figura 18 apresenta a forma como o módulo de Gerenciamento do assistente se relaciona com os outros elementos no modelo.

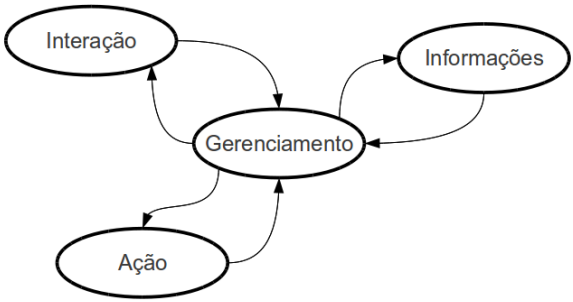

Figura 18: Explosão do elemento Gerenciamento.

Para que o gerenciamento do SAP seja efetivo, é necessário que um conjunto de informações acerca do usuário do SAP esteja armazenada e que possa ser utilizada quando necessário. Essas

informações devem evoluir com o tempo em concordância com a evolução do usuário no mundo real, com mudanças de tarefas, preferências, atividades, etc. (Michaell et al., 1994) , (Bush et al., 2006), (Huhns e Singh, 1998), (Schiaffino e Amandi, 2006).

Por meio dessas informações, o módulo de gerenciamento do SAP deve decidir quando e como iniciar uma ação. Essa também, para que possa ser executada de forma condizente, deve ser carregada com um conjunto de informações específicas e também armazenadas no SAP. Com essas informações é possível que a ação se torne operacional quando certas condições forem satisfeitas.

3.1.4 Visão Geral do Modelo de Referência

Compilando todos os conceitos em uma visão geral do modelo de referência, pode-se ter uma figura ilustrativa de todos os elementos relacionados (Figura 19).

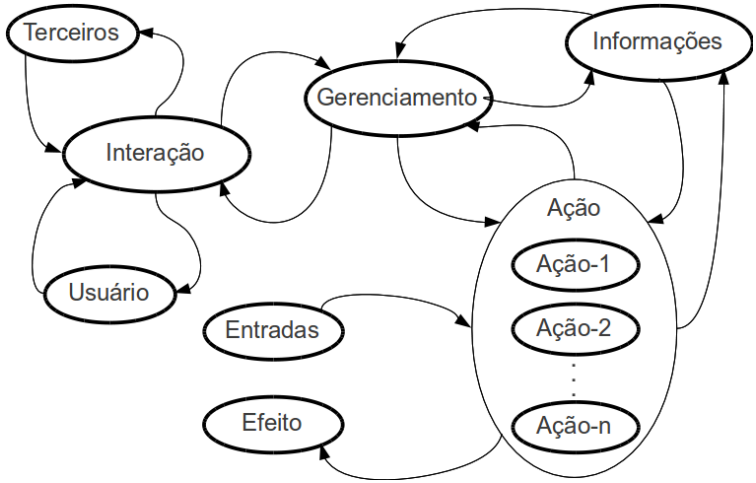

Figura 19: Visão Geral do Modelo de Referência.

Com definições abstratas e genéricas formando um modelo de referência para softwares assistentes pessoais, pode-se agora reunir o modelo, juntamente com o estilo arquitetural escolhido, SOA, para definir a Arquitetura de Referência. Contudo, é apresentado primeiramente uma visão geral da proposta.

3.2 ESTILO ARQUITETURAL

Esta seção visa apresentar e justificar a utilização do estilo arquitetural SOA na modelagem da proposta. Para tal, são apresentados os requisitos sobre SAPs identificados no capítulo 2. Requisitos estes considerados genéricos o suficiente para aportar as características desejadas.

3.2.1 Requisitos Desejáveis

De modo a manter maior conformidade com certas especificidades apresentadas no capítulo 2, alguns requisitos são aqui identificados de forma a justificar a utilização da Arquitetura Orientada a Serviços (SOA) como estilo arquitetural da proposta, quais sejam:

- O modelo deve ser baseado em padrões abertos e atuais, independente de tecnologia e que possa ser facilmente adaptável (flexível).
- O SAP deve poder ser implementado em linguagens e tecnologias tanto comerciais quanto gratuitas ou *opensource*, ou seja, não deve ser obrigatória a utilização de linguagens específicas, contanto que as escolhas possam trabalhar com os padrões aqui definidos.
- O SAP deve permitir que novos comportamentos, processos para execução das atividades do assistente, sejam agregados dinamicamente sem que se tenha que obrigatoriamente alterar sua implementação pronta.
- Deve ser possível substituir um comportamento por outro, com atividade semelhante e mesmos parâmetros, fornecido por outra entidade, sem que seja necessário reconfigurar o SAP, ou que as alterações sejam mínimas.
- Deve haver a possibilidade dos comportamentos do SAP não estarem necessariamente localizados no mesmo computador do núcleo de execução do assistente em si, ou seja, podem estar distribuídos em repositórios de comportamentos na Internet, podendo ser utilizados gratuitamente, alugados ou vendidos por terceiros.

- Um comportamento deve poder ser invocado pelo SAP apenas quando este estiver habilitado para tal e todas as suas condições para ativação forem satisfeitas.

É importante salientar que o uso de uma arquitetura de referência para o desenvolvimento Softwares Assistentes Pessoais objetiva trazer certas vantagens, tais como a interoperabilidade, a flexibilidade dos comportamentos do assistente, a independência de fornecedores de assistentes pessoais e comportamentos, a possibilidade de qualquer poder ter seu assistente pessoal, com os mais variados tipos de comportamentos, tipos de gostos e custos.

3.2.1.1 Baseado em Padrões

> "O modelo deve ser baseado em padrões abertos e atuais, independente de tecnologia e que possa ser facilmente adaptável (flexível)."

Uma das questões mais críticas quando se busca a dinamicidade de sistemas é a integração entre os elementos envolvidos. Para isso, um requisito básico é que devem existir mecanismos adequados e interfaces abertas para que seja possível tal interação (Gesser, 2006). Porém, uma das grandes barreiras para a interoperabilidade entre os sistemas é que muitos dos sistemas são legados e desenvolvidos nas organizações, lidando com os processos de forma específica e de maneira independente, contribuindo para a dificuldade de integração entre tais sistemas. Para que esses sistemas sejam capazes de interoperar de maneira transparente, automatizada e "sem costuras" (*seamlessly*), é necessário que as partes envolvidas sejam capazes de interoperar de maneira transparente (Piazza, 2007).

Nesse sentido, a utilização de tecnologias padronizadas permite que qualquer desenvolvedor que siga determinados padrões possa utilizá-los e interagir entre os demais desenvolvedores que aderem aos mesmos padrões (Gesser, 2006). Dessa forma, tecnologias baseadas em padrões apresentam uma solução bastante atraente, trazendo consigo certas vantagens, tais como a possibilidade de reuso de aplicações de software já existentes e a flexibilidade na interação entre os diversos elementos envolvidos nos processos, e que podem pertencer a diferentes organizações (Piazza, 2007).

Neste contexto, diversos fabricantes vêm adicionando cada vez mais funções e novos padrões em suas plataformas, permitindo uma comunicação mais robustas entre sistemas. Isso permite uma comunicação sem obstáculos, sem costuras, transparente e dinâmica entre sistemas computacionais de diversas organizações. Ainda, graças ao uso de padrões abertos, emerge uma maior facilidade de colaboração e interoperabilidade entre diversas plataformas (Piazza, 2007).

A utilização de SOA, da OASIS (MacKenzie et al., 2006), como padrão escolhido apresenta uma solução para abstrair as aplicações distribuídas na Internet de forma a manter a interoperabilidade entre elas, podendo assim trocar informações e efetuar negociações em uma linguagem de comunicação padrão e em um estágio de desenvolvimento/utilização já bastante maduro.

3.2.1.2 Independente de Linguagens

> "O Assistente Pessoal deve poder ser implementado em linguagens e tecnologias tanto comerciais quanto gratuitas ou *opensource*, ou seja, não deve ser obrigatória a utilização de linguagens específicas, contanto que as escolhas possam trabalhar com os padrões aqui definidos."

É importante que não haja impedimentos na escolha das linguagens de programação que a empresa/usuário deva escolher para o desenvolvimento dos elementos envolvidos na assistência pessoal via software de computador. Contudo, para que esses elementos se enquadrem na proposta, uma questão há de se citar no que se refere à interoperabilidade. As linguagens devem ser compatíveis ou permitir que sejam implementadas bibliotecas nas mesmas, ou que se possa utilizar bibliotecas já existentes que permitam a comunicação e suporte baseados no padrão SOA da OASIS (MacKenzie et al., 2006). Isto vai de encontro com o requisito da Independência de Implementação de SOA, definidos por Singh e Huhns (2005).

Além disso, contanto que a interoperabilidade e a utilização dos padrões sejam satisfeitos, deve ser de escolha do desenvolvedor, ou empresa desenvolvedora, qual a linguagem que o agente deve ser implementado, não importando o fato dela ser uma ferramenta comercial, gratuita ou *opensource*. Dessa forma, diversas empresas

podem possuir suas próprias implementações de Softwares Assistentes Pessoais, ou comportamentos na forma de serviços de software.

3.2.1.3 Comportamentos Flexíveis

> "O Assistente Pessoal deve permitir que novos comportamentos – processos para execução das atividades do assistente – sejam agregados dinamicamente sem que se tenha que obrigatoriamente alterar sua implementação pronta."

Considerando que o assistente executa comportamentos que podem estar distribuídos em repositórios locais ou remotos, o SAP deve ficar responsável apenas por gerenciar de forma efetiva a instanciação de tais comportamentos. Dessa forma, para cada situação ou evento, este deve identificar que comportamento deve ser instanciado, como ele deve ser instanciado, que informações são necessárias da base de dados para sua execução e, caso necessário, como deve ocorrer a interação com o usuário. Assim, a complexidade do comportamento deste SAP deve emergir conforme a composição de um agrupamento de comportamentos mais simples, ou complexos, que fazem parte do conjunto de possíveis comportamentos que o SAP pode instanciar.

Concomitantemente, o comportamento do Assistente Pessoal deve ser visto como uma forma macro de comportamento que é composto do subconjunto de possíveis serviços de software que podem ser encontrados e utilizados pelo SAP e que estão disponíveis em repositórios de serviços e comportamentos na Internet.

Além disso, é desejável que não seja necessário alterar a implementação do SAP a cada vez que um novo comportamento for agregado ao mesmo. Este é um requisito facilitador para a ampliação da capacidade de ação do agente e da possibilidade de customização deste para satisfazer os objetivos dos usuários.

Em SOA, conforme Singh e Huhns (2005), o Baixo Acoplamento permite que não haja uma dependência forte entre os serviços. Também, é possível trabalhar com uma forma de Configuração Flexível, uma vez que os serviços devem poder ser ligados entre si de forma dinâmica e flexível.

3.2.1.4 Comportamentos substituíveis

> "Deve ser possível substituir um comportamento por outro, com atividade semelhante e mesmos parâmetros, fornecido por outra entidade, sem que seja necessário reconfigurar o Software Assistente Pessoal, ou que as alterações sejam mínimas."

Seguindo a filosofia da Arquitetura Orientada a Serviços em que mais de um provedor de serviços pode fornecer funcionalidades iguais ou parecidas (McKenzie, 2006), deve ser possível que o usuário possa substituir um comportamento por outro que satisfaça melhor suas necessidades, sejam essas relacionadas a como o comportamento é executado, a qualidade do resultado final ou até em relação aos custos da utilização de um comportamento.

Tal especificidade afeta diretamente a qualidade do fornecimento de comportamentos para o Assistente Pessoal do usuário, sendo que acaba por incentivar a concorrência entre os fornecedores. Isso também possibilita que o usuário possa escolher por utilizar uma versão gratuita de um comportamento fornecida por entidades que não necessariamente têm foco dos lucros por meio da venda ou do aluguel de tais comportamentos diretamente.

A grande vantagem disso é a livre escolha, do usuário do Assistente Pessoal dessa proposta, em relação aos fornecedores de assistentes pessoais e comportamentos das quais ele pode agregar para compor o comportamento do seu SAP. Esse requisito também vai de encontro à especificação de Baixo Acoplamento do SOA (Singh e Huhns, 2005).

3.2.1.5 Comportamentos Distribuídos

> "Deve haver a possibilidade dos comportamentos do Software Assistente Pessoal não estarem necessariamente localizados no mesmo computador do núcleo de execução do Software Assistente Pessoal, ou seja, podem estar distribuídos em repositórios de comportamentos na Internet, podendo ser utilizados gratuitamente, alugados ou vendidos por terceiros."

A distribuição de um sistema pode ser tanto de hardware como de software, sendo que tais elementos podem estar localizados em redes de computadores e se comunicam, coordenando suas ações por passagem de mensagens. Algumas características que estes sistemas fornecem são a concorrência de componentes, independência de falhas de componentes e transparência de localização. Os usuários destes sistemas o percebem como um sistema único com funcionalidades integradas. Dentre as motivações para a utilização de sistemas distribuídos estão o compartilhamento de recursos, gerenciados por servidores e acessados por clientes e que podem ser encapsulados como objetos e acessados por outros objetos (Coulouris et al., 2005).

Outras características que são encontradas nestes sistemas são a concorrência e heterogeneidade de seus componentes. Isso permite que vários componentes, desenvolvidos em diferentes linguagens e tecnologias, possam ser executados ao mesmo tempo. Estes componentes também podem ser adicionados ou substituídos de forma dinâmica. Esta dinamicidade também traz a vantagem da tolerância a falhas, sendo que com a indisponibilidade de um elemento do sistema, outro pode assumir sua função (Coulouris et al., 2005).

Ainda, os sistemas distribuídos são paralelos e vários usuários podem invocar serviços do sistema simultaneamente, assim como vários processos servidores são executados concorrentemente, sendo que cada um atendendo várias requisições dos clientes (Kshemkalyani e Singhal, 2008).

Ao se apropriar destes conceitos e características, emerge também as vantagens inerentes aos sistemas distribuídos. Assim, grande parte do processamento do assistente pessoal pode ser executado paralelamente com o conjunto de componentes computacionais (hardware) distribuídos na Internet. Tal requisito é correspondente à especificação de SOA sobre a Granularidade, permitindo que um sistema possa ser dividido em vários serviços e com a Distribuição, distribuindo os serviços de forma a melhor aproveitar os recursos computacionais envolvidos.

3.2.1.6 Comportamentos Dinâmicos

> "Um comportamento deve poder ser invocado pelo assistente apenas quando este estiver habilitado para tal e todas as suas condições para ativação forem satisfeitas."

Deve existir um conjunto de condições para que certo comportamento seja ativado, ou seja, entre em execução. Tais condições devem estar devidamente definidas nas informações gerais do usuário do Assistente Pessoal e das especificidades de cada comportamento quando configurados ao serem agregados. Este formato de implementação segue a ideia de Brooks (1991), de que a inteligência do sistema pode ser decomposta em comportamentos conectados à ação. Assim, o comportamento geral do sistema emerge naturalmente com base em um conjunto de blocos funcionais. Neste caso, o problema fica na questão referente a qual, ou quais, comportamentos devem ser ativados a cada momento específico.

Ainda, fazendo um paralelo com o requisito de comportamentos distribuídos do assistente pessoal, emerge um grande potencial de prover uma crescente disponibilidade de recursos por causa da replicação de recursos para o sistema, bem como a possibilidade de distribuição destes geograficamente. Assim, um comportamento do Assistente pode ser invocado por este quando este estiver habilitado para tal e todas as suas condições para ativação forem satisfeitas. Porém, quando não mais estiver disponível tal recurso, este deve poder ser substituído por outro que supra a necessidade local do agente. Isso também entra em concordância com a especificidade de Baixo Acoplamento e Configuração Flexível do SOA (Singh e Huhns, 2005).

3.2.2 Um Paralelo com o Estilo Arquitetural

A abordagem SOA da OASIS (MacKenzie et al., 2006) sugere em sua especificação uma padronização na forma de comunicação entre elementos/parceiros na Internet, mais especificamente sob a web. Se tal característica for observada isoladamente, isso já fornece recursos suficientes para também Assistentes Pessoais funcionando sob essa perspectiva poderem manter a interoperabilidade de comunicação para a criação de parceiros de negócios e para a utilização de funcionalidades disponíveis na Internet na forma de serviços web.

No entanto, as características de Baixo Acoplamento, Independência de Implementação, Configuração Flexível, Granularidade e Distribuição citados por Singh e Huhns (2005), combinam de forma bastante conveniente com os requisitos apresentados nessa seção. Isso fornece ao Assistente Pessoal uma abrangência maior de possibilidades de ação no objetivo de auxiliar seu usuário. E ao usuário, uma liberdade

de escolha quanto à forma como seu Assistente deve funcionar, quanto a quem irá efetuar determinadas ações, quanto a quantia monetária que estará disposto a pagar, etc. Isso também compele à característica de SOA que, segundo MacKenzie et al. (2006), cada empresa, responsável pela solução de um domínio de problemas terá sua competência local disponível para terceiros, possibilitando assim, ao Assistente Pessoal, se utilizar de diversas funcionalidades avançadas, de diversos fornecedores de serviços de software diferentes e com as mais diferentes formas de se resolver cada problema.

3.3 ARQUITETURA DE REFERÊNCIA

Tendo como premissa inicial o modelo de referência e suas funcionalidades, uma arquitetura de referência tem por objetivo fornecer um norteador para o mapeamento das funcionalidades do modelo de referência para um modelo que deve se formar um sistema, ou seja, ela é o mapeamento do modelo de referência para elementos de de software (Bass et al., 2003). Segundo MacKenzie et al. (2006), ela é, em si, um padrão genérico para um projeto e que deve abordar requisitos para o desenvolvimento de soluções, guiadas pelo modelo de referência, de forma a atender as necessidades do projeto.

A arquitetura de referência é também guiada por um estilo arquitetural que, juntamente com o modelo de referência, formam um conjunto de requisitos e características de norteamento. Com isso, neste ponto já se torna necessário apresentar os tipos dos elementos envolvidos, como eles interagem e o mapeamento das funcionalidades destes elementos, com a finalidade de compor a arquitetura de referência (Hofmeister et al., 2000).

Para iniciar a apresentação da Arquitetura de Referencia para Softwares Assistentes Pessoais, parte-se do que já foi definido no modelo de referência apresentado anteriormente e no estilo arquitetural escolhido, que é o SOA.

No modelo de referência, alguns requisitos desejados de SAPs, identificados seção 2.1 (autonomia e interatividade). Outros pontos são resolvidos mais em nível de tecnologia utilizada. A interoperabilidade, por exemplo, pode ser resolvida com a utilização das diversas tecnologias de implementação utilizadas para se implementar o estilo

arquitetural escolhido (SOA). Este baseado em rede, também resolve o requisito de conectividade à rede identificado.

Os demais requisitos (adaptabilidade ao usuário e ao contexto, flexibilidade, e ser de propósito geral) são requisitos que a própria arquitetura de referência permite e provê estrutura para a implementação. O fato de usuários poderem conectar serviços aos comportamentos (ou ações) do SAP, provê uma forma estruturada de *plugins* de comportamentos. Esses podem prover os demais requisitos, até certo ponto, pois o usuário pode escolher os serviços que melhor se adaptam a si e ao seu contexto; é flexível, pois o usuário pode retirar ou adicionar serviços em tempo de execução/configuração; e de propósito geral, pois não há delimitação de quais serviços podem ser agregados ou não, ou seja, sendo que o serviço possua a interface necessária definida, ele pode ser utilizado no comportamento do SAP.

Partindo do elemento "Gerenciamento" do modelo, tem-se que este deve efetuar a coordenação da "Ação" do assistente pessoal e das informações necessárias para sua execução. Contudo, como também foi visto, essa "Ação" é a composição de um conjunto de possíveis "Ações" que o assistente pode realizar e um conjunto de informações necessárias para a execução de cada ação.

O comportamento do assistente pessoal é visto como o subconjunto das possíveis funcionalidades que podem ser utilizadas. Este subconjunto é escolhido pelo usuário e adicionado ao assistente pessoal como forma de satisfazer suas necessidades e auxiliar em suas tarefas. A agregação das funcionalidades se dá pela seleção de serviços de software de repositórios na Internet, juntamente com a configuração desta em um ambiente de configuração do assistente pessoal.

Conforme apresentado no modelo de referência, para que o assistente possa auxiliar o usuário em suas tarefas diárias, é necessário que haja a "interação" entre o usuário e o assistente. Aliando o modelo de referência com o estilo arquitetural SOA, pode-se modelar uma visão geral (Figura 20) com o usuário, interagindo com o seu assistente pessoal por meio de três elementos:

- **Aplicações Legadas**: Softwares de desenvolvedores específicos e softwares que podem ser desenvolvidos pela empresa onde o usuário trabalha. Tais aplicações podem possuir funcionalidades que o usuário possa querer para agregar ao seu assistente pessoal, ou pode ser um sistema da empresa que o usuário pode querer que o assistente gerencie. Como tal, deve ser possível o desenvolvimento de serviços que façam essa

"ponte" de conversa entre o assistente e essas aplicações legadas, de forma que o assistente pessoal possa ver e utilizar tais aplicações.

- **Aplicações do Usuário**: As aplicações do usuário são, no caso da presente proposta, aplicações que o usuário lida em seu dia a dia e que devem ser utilizadas para interagir com seu assistente pessoal. Pode ser uma GUI específica, ou, conforme no sistema Sandy (2011), o usuário pode se utilizar de serviços de e-mails, mensagens de Twitter, sistemas de *Instant Messaging*, ou SMS para interagir com o assistente pessoal.
- **Aplicações de Configuração e Desenvolvimento**: Essas aplicações podem ser específicas para cadastrar e configurar um assistente pessoal, ou serviços que esse assistente pessoal possa precisar e que requerem cadastro. Ambientes de desenvolvimento em linguagens específicas (Editores BPEL, IDE de Java, ou PHP, ou C/C++, etc.) que gerem elementos que integram a proposta também podem ser considerados como este tipo específico de aplicação. Essas aplicações podem aqui serem chamadas de forma simplificada apenas como "Ferramentas".

A Figura 20 apresenta uma visão geral da proposta. Esta é composta pelo usuário, que pode estar em diversos níveis de conhecimento. O usuário pode tanto ser uma pessoa que possui um assistente pessoal, agrega os comportamentos que necessita e o utiliza, como também pode ser um usuário avançado, que desenvolve funcionalidades para SAPs.

Os Serviços de Interoperabilidade são criados para servirem de interface com outros sistemas que não estão no padrão SOA de serviços web. Neste caso, adota-se uma abordagem de integração onde os aspectos de interoperabilidade são tratados de forma desacoplada dos aspectos intrínsecos funcionais e não funcionais de um serviço, como proposto no projeto COIN (2011).

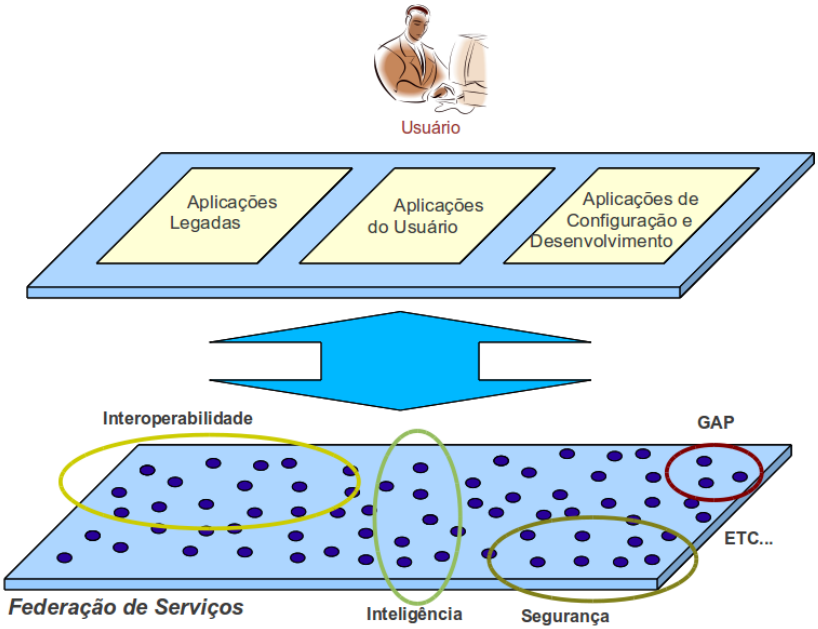

Figura 20: Visão Geral da Proposta.

A seguir, são citados alguns níveis de usuários, sendo que uma pessoa pode se enquadrar em mais de um nível:

- **Usuário Simples**: cadastra um assistente pessoal para si, procura e utiliza comportamentos (serviços de software), configura informações simples para seu assistente.
- **Usuário Avançado**: além de possuir as características de um usuário simples, pode compor comportamentos mais complexos no assistente, via uma interface de algoritmos ou fluxogramas (detalhado no próximo capítulo).
- **Desenvolvedor de serviços de software:** Desenvolve serviços de software específicos para SAPs, ou simplesmente desenvolve serviços de software, e estes podem ser utilizados pelos usuários para compor seus comportamentos. Estes serviços de software podem tanto ser serviços simples, quanto podem ser resultado da coreografia ou orquestração de serviços. Também, não é obrigatório que a implementação seja em alguma linguagem específica (Java, PHP, C/C++, BPEL), contanto que se mantenha no padrão SOA da OASIS (MacKenzie et al., 2006).

- **Desenvolvedor de serviços de interoperabilidade**: Desenvolve serviços que fazem a ponte entre um sistema baseado em SOA e SOAP (pode ser um sistema que faz orquestração ou coreografia de serviços), que necessita utilizar algum serviço que não está no mesmo padrão. Por exemplo, se um sistema da empresa precisa ser acessado/utilizado pelo SAP, que funciona conforme a presente proposta, este precisa possuir serviços web de interoperabilidade, que fornecem um conjunto de operações para acessar o sistema da empresa.
- **Desenvolvedor de Assistentes Pessoais**: Desenvolvedor avançado que pode desenvolver tanto interfaces para configuração de assistentes pessoais, como núcleos de execução dos assistentes. Porém, é necessário se manter nas linhas dessa proposta, de forma a manter a compatibilidade com os outros elementos da Arquitetura de Referência.

A "Federação de Serviços", por sua vez, é o elemento referente ao conjunto de serviços que estão distribuídos na Internet e podendo também estar disponíveis no repositório local de serviços de software da empresa onde o usuário está no momento ou trabalha. Cada serviço possui sua própria funcionalidade e sua forma de implementação em linguagem da escolha do desenvolvedor, mas deve manter o padrão SOA (MacKenzie et al., 2006).

Trabalhando no contexto de que o assistente pessoal é um conjunto de processos computacionais trabalhando em conjunto, e criados para representar um usuário na execução de certas tarefas, automaticamente ou com algum grau de intervenção/supervisão humana, tem-se uma visão dos comportamentos do assistente como a composição de um conjunto de funcionalidades distribuídas na internet, vistas como serviços, e que podem ser executadas em paralelo.

Formalizando o cenário apresentado na Figura 20, tem-se por *A* um conjunto não vazio, tal que A representa o conjunto de *n* serviços da federação de serviços *{a1, a2,..., an}* e distribuídos na Internet. Tem-se por *B* o comportamento resultante do assistente pessoal, sendo caracterizado pelo conjunto de serviços de *A*, no qual $B \subseteq A$. Seja *bi* um serviço de *B* e *{bi, b2,..., bn}* o conjunto de serviços de *B*, logo, o comportamento de *B* é caracterizado por

$$B = \sum_{i=1}^{n} bi$$

O comportamento do assistente pessoal é, portanto, o subconjunto de serviços selecionados e configurados para este, da totalidade de serviços disponíveis e distribuídos na Internet, e em conformidade com os padrões abertos de conectividade e comunicação aceitos pela proposta desse trabalho, ou que se utilizem de um serviço de interoperabilidade que servem como comunicação com os outros elementos que mantêm o padrão.

Dentre os serviços dispostos na Federação de Serviços, pode-se encontrar diversos tipos serviços, que se diferem tanto na implementação, linguagem de desenvolvimento, funcionalidade, como técnicas avançadas para resolução de problemas. Pode haver um conjunto de serviços que se utiliza de Inteligência Artificial para resolver determinados tipos de problemas, podem haver serviços que tentam resolver problemas de segurança, serviços de filtragem de informação, serviços de procura e catálogo de serviços web, serviços de aprendizagem, serviços de gestão, geradores de relatório, gerenciadores de contas de e-mail, serviços de previsão de tempo, de avaliação do trânsito, etc. Esses são apenas alguns exemplos dentre uma grande gama de aplicações que podem ser desenvolvidas/encontradas na Federação de Serviços. Além disso, uma vez que a tecnologia de serviços e SOA já se encontra em um certo estágio de maturidade, podem existir na Internet uma série de serviços já desenvolvidos para serem utilizados pelo assistente pessoal.

Uma questão importante, e isso deve estar explícito na Arquitetura de Referência, é que as aplicações legadas, aplicações do usuário e outros serviços podem não conversar com o padrão de serviços da OASIS (MacKenzie et al., 2006) escolhido, fazendo-se necessário uma ponte entre as aplicações e o assistente pessoal por meio desses "Serviços de Interoperabilidade", uma vez que o assistente pessoal deve apenas utilizar como ações, os serviços com padrão OASIS.

Por fim, é necessário que haja um módulo que coordena as atividades do assistente pessoal. Este módulo é aqui chamado de "Gerenciador de Assistentes Pessoais" e pode visto apenas na forma do "Gerenciamento" no modelo de referência. Como ele é aquele responsável pela coordenação da execução das ações e suas informações, e a proposta utiliza-se do estilo arquitetural SOA, então sua responsabilidade é fazer a coordenação e invocação dos serviços quando certas condições são satisfeitas.

Essa coordenação pode ser em três níveis básicos, definidos aqui de forma simplificada:

- **Comportamento Simples**: São necessárias informações para a chamada do serviço e o serviço é executado toda vez que o comportamento é chamado, enviando os parâmetros de entrada e retornando um resultado. Como, por exemplo, um serviço que busca a hora local. Este pode, a cada ciclo de chamada de comportamento, ser chamado para atualizar a hora do SAP, informação que pode ser utilizada na execução de outros comportamentos.
- **Comportamento Condicional:** Assim como o comportamento anterior, esta classe de comportamento também precisa de informações básicas. Contudo, não necessariamente é executado a cada ciclo de execução. Para que este comportamento seja executado é necessário que certa condição ou um conjunto de condições sejam satisfeitas. Por exemplo, um comportamento de envio de relatórios diários para o usuário é chamado apenas se a hora do sistema no assistente pessoal for igual a 23h. Essa informação da hora atualizada pode ser buscada do exemplo de comportamento anteriormente citado.
- **Comportamento Complexo:** Esta configuração de comportamento pode necessitar um certo conhecimento mais avançado do usuário. Neste ponto, além das informações, das condições, o usuário pode modelar o comportamento como na forma de um fluxograma, ou algoritmo, com laços, condicionais e outros elementos que podem ser definidos pelo desenvolvedor do SAP. Por exemplo, para gerar um relatório no final do dia com todas as atividades realizadas pelo assistente pessoal, o comportamento do SAP pode efetuar um laço, pegando cada uma das atividades registradas em um serviço de registro de *logs*, montá-las em um único arquivo e, se for 23h, ele pode enviar essas informações para o e-mail do usuário, deixando-o a par do que o assistente pessoal seu fez durante o dia.

Assim, tem-se o "Usuário" que interage com seu assistente pessoal por meio de "Aplicações do Usuário", "Aplicações Legadas" e "Ferramentas". As "Ferramentas" são compostas de ambientes/softwares de configuração, IDEs (*Integrated Development Environment*) ou outras ferramentas de desenvolvimento que servem para desenvolver ou configurar elementos da arquitetura. Contudo, o Assistente Pessoal – ou Gerenciador de Assistentes Pessoais (GAP) – pode apenas se utilizar de serviços, e para que o GAP possa "ver" os elementos de interação com o usuário ou outros serviços não

compatíveis, é necessária uma camada que é chamada aqui de “Serviços de Interoperabilidade”. Dessa forma, pode-se agora apresentar a Figura 21 como uma visão da Arquitetura de Referência para Assistentes Pessoais baseado na Arquitetura Orientada a Serviços, proposta no presente trabalho.

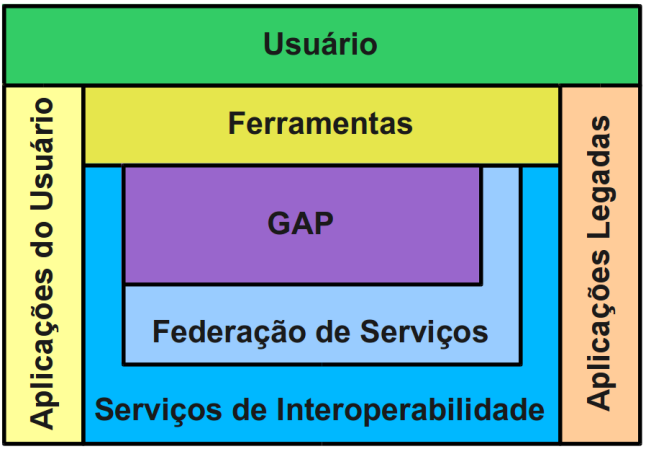

Figura 21: Arquitetura de Referência.

A área mais acima (Usuário) não é exatamente um elemento da arquitetura, mas sim a visão que o usuário possui do assistente pessoal, ou seja, os elementos da arquitetura da qual devem fazer o interfaceamento entre o usuário e seu assistente pessoal.

3.4 EXEMPLOS

Esta seção apresenta alguns exemplos como forma de apresentar uma visão mais concreta da forma de funcionamento dos Softwares Assistentes Pessoais sob o contexto da Arquitetura de Referência Proposta. Isso também se faz necessário para se ter uma melhor noção antes de se propor uma instância de implementação, de modo a se ter uma melhor visualização de certas possibilidades de se trabalhar com a presente proposta.

Resume-se aqui brevemente alguns pontos que já foram apresentados.

Primeiro, o Assistente Pessoal é uma entidade em execução autônoma que tenta auxiliar o usuário em suas tarefas no dia a dia. Segundo, a ação do Assistente Pessoal, ou comportamento, é a composição de serviços disponíveis em repositórios distribuídos na Internet e que são escolhidos pelo usuário para compor as atividades que o seu assistente pode efetuar. O terceiro ponto é a interação entre o assistente pessoal e o usuário. Neste ponto, a forma de interação deve se dar da forma menos intrusiva possível, ou seja, devem ser utilizadas interfaces e softwares já existentes e conhecidas do usuário para interagir com o assistente pessoal. Nesse contexto, e seguindo certas ideias definidas pelo sistema Sandy, apresentado por Mann (2011), o presente cenário de interação é dado via troca de mensagens via *Instant Messaging*, Twitter e e-mail.

Tendo em vista esse cenário aqui definido, a seguir são apresentados alguns exemplos – ainda num certo nível de abstração – de utilização da proposta de SAPs.

3.4.1 Gerenciamento de E-mails do Usuário

Partindo-se da premissa de que as pessoas recebem diversos e-mails diários e que o gerenciamento dessas mensagens requer um certo tempo para filtragem e análise do que realmente é importante ser analisado a cada momento, uma forma de verificação dessas mensagens, de forma automatizada, traria benefícios aos usuários de modo a liberá-los de grande parte do gerenciamento dessa tarefa. Sendo assim, como um primeiro exemplo de utilização de SAPs é apresentado um comportamento de gerenciamento de e-mails do usuário. Este exemplo já foi citado e implementado na forma de um protótipo inicial no artigo "Virtualização de Colaboradores na Manufatura: um modelo baseado em *Agent Bots*" (Zambiasi e Rabelo, 2007).

Neste exemplo, visualizado na Figura 22[10], pode-se observar que o usuário possui acesso há três tipos de aplicações do usuário: Sistema de envio de mensagens via SMS de seu dispositivo móvel, Sistema de IM do Gtalk e algum sistema que faz acesso à sua conta de e-mails em um servidor de e-mails.

[10]Nas Figuras dos exemplos, a nomenclatura SW significa "serviço web" e SWI "serviço web de interoperabilidade".

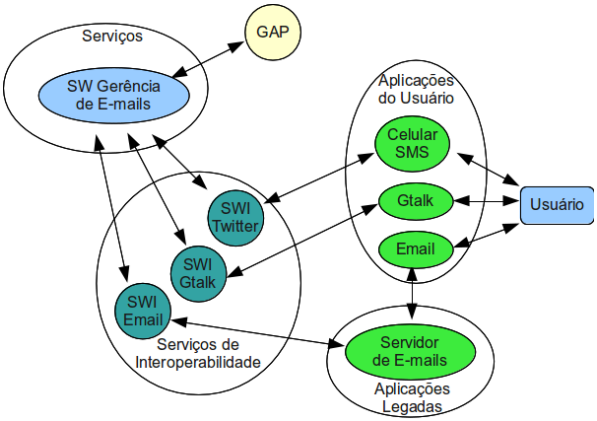

Figura 22: Gerenciamento de e-mails do usuário.

No SAP do usuário, mais especificamente no módulo GAP, há um um acesso à um serviço web de gerência de e-mails. Este, por sua vez pode se comunicar com o usuário por mensagens SMS via serviço de interoperabilidade do Twitter, pode enviar mensagens via Gtalk ou verificar se o usuário está *online* por meio do serviço de interoperabilidade do Gtalk e pode verificar a conta de e-mails do usuário no servidor de e-mails via servidor de interoperabilidade de e-mails. Este exemplo é apresentado na forma de fluxograma na Figura 23.

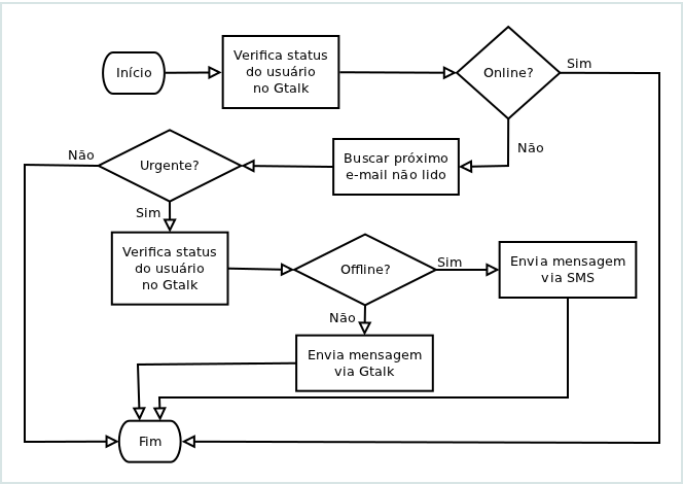

Figura 23: Fluxograma do gerenciador de e-mails.

O comportamento de gerenciamento verifica o *status* do usuário no Gtalk, caso ele esteja está *online,* então o usuário mesmo pode gerenciar seus e-mails. Caso contrário, o usuário pode estar *offline* ou ocupado e não pode, no momento, ficar verificando seus e-mails. Neste caso o comportamento de gerenciamento busca o próximo e-mail não lido da conta de e-mails do usuário e avalia se o e-mail é "Urgente", por meio de palavras chaves no campo "Assunto" ou e-mails procedentes de algumas contas específicas, tal como a conta de e-mail do chefe de seção, supervisor, professor orientador, etc. Caso o e-mail seja urgente, então faz uma nova verificação no *status* do usuário no Gtalk. Se o usuário estiver *offline*, então envia uma mensagem via SMS avisando o usuário sobre o e-mail urgente, caso ele esteja com o *status* setado como ocupado, então é enviada uma mensagem via Gtalk, avisando o usuário da mensagem urgente.

3.4.2 Processo de Compra Automatizado

Considere um cenário em que uma pessoa, funcionário de uma empresa, possui a função de gerenciar o estoque de produtos necessários para a fabricação dos produtos da empresa. De forma a manter o estoque sempre em dia e não haver a possibilidade da empresa ter que parar a produção dos produtos devido a falta de recursos materiais, o funcionário deve efetuar compras de matéria-prima quando o estoque estiver baixo.

Agora considere que o funcionário irá se utilizar de um SAP para auxiliá-lo na tarefa de compra de produtos para manter o estoque de matéria-prima em dia, e outras. Ou por escolha própria, ou sugestão da própria empresa. Neste caso, considere o cenário da Figura 24.

Para a utilização de um SAP, o usuário deve utilizar uma aplicação (APConf) para acessar, criar e configurar seu assistente pessoal (GAP). Para que o SAP possa auxiliá-lo na atividade citada, é necessário que este possa acessar as informações do estoque. Para isso, o sistema deve ter, ou deve ser desenvolvido, um serviço de interoperabilidade (SW-Estoque) para que o assistente pessoal (GAP) possa acessar o sistema de estoque.

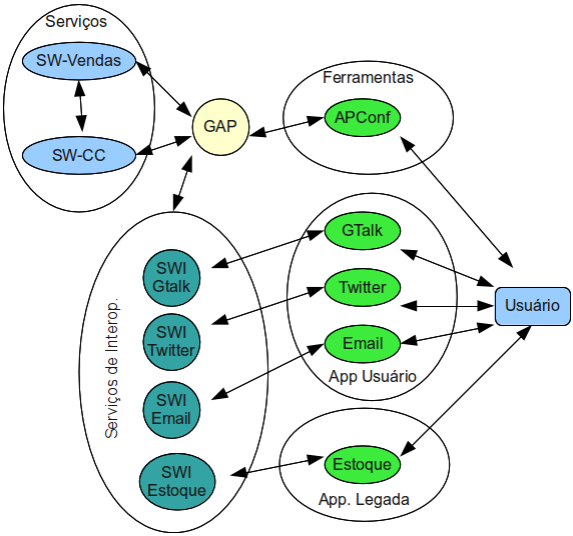

Figura 24: Estudo de caso do processo de compra automatizado.

Também, conforme determinado pelo autor do presente trabalho, o Assistente Pessoal deve acessar certas aplicações para interagir com o usuário são elas Gtalk, Twitter e E-Mail. Para acessar esses recursos, caso não existam serviços no padrão OASIS (MacKenzie et al., 2006), serviços de Interoperabilidade devem ser desenvolvidos (SW-Gtalk, SW-Twitter, SW-Email, respectivamente).

Nesse exemplo, é necessário que o SAP possa efetuar automaticamente compras para manter o estoque em dia. Assim, é necessário um serviço para acesso ao sistema de compras (ou um fornecedor específico) e um serviço para efetuar o pagamento da compra.

Com esse cenário, agora é possível ter em mente a forma de atuação do SAP. Este pode ser configurado pelo usuário para acessar o estoque, informações sobre cartão de crédito, sobre suas contas de e-mail, twitter e gtalk.

Pode-se então, ter dois caminhos distintos para a criação do comportamento automático de compra:

1. O usuário, na forma de um usuário avançado na interface de configuração do Assistente Pessoal, pode criar um comportamento na forma de um algoritmo;

2. O usuário é um usuário de nível básico na interface e faz a requisição para um desenvolvedor da empresa desenvolver um serviço que efetua o processo automático de compra. O usuário, por sua vez, utiliza este novo serviço criado na empresa para fazer a compra automática;

Neste exemplo, é considerado o tipo de usuário 1. O usuário deve entrar na interface de configuração do Assistente Pessoal e configurar as informações e um fluxograma ou algoritmo que vai fazer a compra automática.

A seguir é descrito de forma simples e em forma de algoritmo a execução do comportamento configurado:

1. Verifica se há algum produto com estoque baixo via chamada do serviço SW-Estoque;
2. Se houver um produto com estoque baixo, então:
    1. Verifica qual produto é o produto, quanto deve ser comprado e qual é o fornecedor.
    2. Envia uma requisição de compra ao Fornecedor via SW-Vendas.
3. Se a ordem de compra foi alterada pelo fornecedor, então:
    1. Se o usuário estiver online no Gtalk
        1. Envia pedido de confirmação para usuário pelo Gtalk
    2. Senão
        1. Envia pedido de confirmação para o usuário via Mensagem Direta do Twitter (que se concretizará como mensagem para o celular, devido à recursos específicos).
        2. Aguarda resposta do usuário
        3. Se usuário confirmou Continua Compra
        4. Senão cancela compra em SW-Vendas
4. Senão
    1. Pega do fornecedor SW-Vendas informações para o pagamento
    2. Envia para a empresa de negociação via cartão de crédito WS-CC informações para o pagamento
5. Termina processo de compra e atualiza Estoque via SW-Estoque informando que a compra foi efetuada.

É importante observar que, tanto no caso do usuário ser especializado e saber criar um comportamento, como no caso do usuário ser de nível básico, ele precisa configurar as informações necessárias

para o acesso aos serviços, senhas, usuários, informações do cartão de crédito, etc.

3.4.3 Geração de Relatório

Este exemplo parte também do mesmo usuário de sistema de controle de estoque acima citado. Contudo, o que o Assistente Pessoal deve fazer é gerar um relatório diário das atividades do usuário no seu trabalho no sistema de controle de estoque. Assim, um serviço de geração de relatório é necessário, assim como um comportamento para fazer o gerenciamento das informações. Esse cenário pode ser observado na Figura 25.

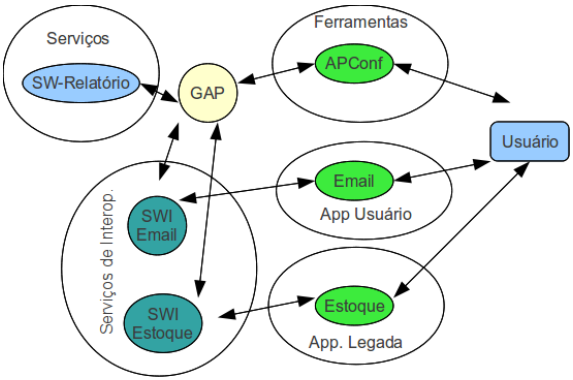

Figura 25: Comportamento de geração de relatórios.

Neste caso, o usuário deve trabalhar em seu sistema, efetuando compras, verificando os produtos com estoque baixo, etc. No GAP, por sua vez, um comportamento faz a verificação periodicamente das últimas atividades realizadas pelo usuário e as envia para o SW-Relatório, serviço referente a um gerador de relatórios. No final do expediente do usuário, o Assistente Pessoal requisita do SW-Relatório que monte um relatório completo a partir das informações enviadas. Esse relatório é então enviado para o e-mail do usuário via serviço web de interoperabilidade SW-Email. O algoritmo fica da seguinte forma:

1. Se hora não for 18h
    1. Se houveram novas atividades do usuário em SW-Estoque

    1. Busca as informações das últimas atividades em SW-Estoque
    2. Envia as informações para SW-Relatório
2. Senão
    1. Requisita relatório completo de SW-Relatório
    2. Envia relatório para o e-mail do usuário via SW-Email.

É importante salientar que, tanto nesse exemplo quanto no anterior, considera-se que o comportamento do assistente pessoal é chamado a cada ciclo de chamada de comportamentos. Isso significa que a cada certo período de tempo, por exemplo 1 minuto, o Assistente Pessoal executará todos os comportamentos ativos da lista de comportamentos configurados no Assistente Pessoal. É claro que os comportamentos que possuírem condições não satisfeitas não irão seguir sua execução.

3.4.4 Pesquisa Automatizada

Neste exemplo, um usuário de um assistente pessoal trabalha com *Design* de produtos, e seguidamente necessita pesquisar tipos de produtos semelhantes ao que ele deverá desenvolver. O processo da pesquisa poderia ser automatizado, facilitando assim o trabalho da pessoa, deixando-a com o trabalho de criação e menos com trabalho de procura e filtragem.

Assim, conforme a Figura 26, há um serviço que faz pesquisa de texto, imagem, vídeo, e outros, conforme certas palavras chaves, retornando um relatório com as informações requeridas (SW-Pesquisa).

Neste cenário, há a necessidade de interação do usuário com seu assistente pessoal. Essa interação se daria na forma de mensagens de Gtalk que o usuário enviaria para seu SAP e de e-mail que o usuário receberia do SAP com o relatório.

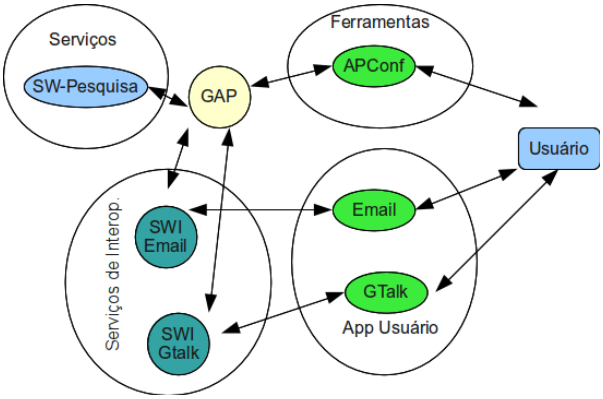

Figura 26: Caso de pesquisa automatizada

Este comportamento se dá da seguinte forma, o usuário envia uma mensagem para seu assistente pessoal via Gtalk escrito *"$Pesquisa 'Móveis estilo renascentista' $para 'amanhã 14h'"*. O SAP identifica a mensagem com a palavra "$Pesquisa" (Identificando a ação ou comportamento) e a palavra-chave "$para" (identificando o dia e horário) e envia a mensagem contendo "Móveis estilo renascentista" para o serviço de pesquisa SW-Pesquisa. No dia seguinte, as 14 horas, o assistente faz a requisição do resultado da pesquisa para o WS-Pesquisa e envia o resultado por e-mail para o usuário.

3.4.5 Gerenciamento de Viagem

Extrapolando agora a ideia dos Softwares Assistentes Pessoais sob o contexto da arquitetura proposta, e aplicando para um caso mais complexo, considera-se aqui um caso fictício de Gerenciamento de Viagem.

Um funcionário necessita viajar para uma reunião da empresa em uma filial situada em uma determinada cidade. Quando a pessoa é informada de tal atividade, envia uma mensagem via SMS de celular para seu assistente pessoal:

> "Reunião: filial código 12, dias 14 e 15, das 14 as 18 horas."

O Assistente Pessoal recebe essa mensagem e utiliza um serviço de avaliação de mensagens para identificar a atividade, o local, dias e

horários. Após identificar nos serviços de informação da empresa que a filial fica em outra cidade, o assistente pessoal avalia a quantidade de dinheiro disponibilizado pela empresa para a viagem e verifica reserva em diversos serviços.

Se há recursos suficientes para ir e voltar de avião, então efetua uma reserva para que a pessoa possa estar nos dias em questão no local. Também verifica os hotéis que a empresa utiliza regularmente quando envia funcionários naquele local e também efetua reserva.

O assistente pessoal pode ainda efetuar uma pesquisa de restaurantes, formas de transportes locais, custos, atividades recreativas que podem ser feitas nos horários livres, em concordância com as preferências da pessoal, e outros.

No caso deste exemplo, pode-se identificar alguns serviços, vistos na Figura 27, que poderiam ser necessários:

- **Serviço de pesquisa de transportes locais**: procura informações de interesse do usuário em relação a transportes locais.
- **Serviço de informações turísticas**: Retorna informações de interesse do usuário quanto a formas de laser, restaurantes, mapas e atividades recreativas na cidade.
- **Serviço de avaliação de mensagens**: utilizado para selecionar o comportamento que deve receber a mensagem enviada pelo usuário.
- **Serviço de geração de relatório**: pegando as informações geradas pelo serviço de pesquisa, envia um relatório para o usuário via e-mail.
- **Serviço de mensagem via celular**: Tal recurso está disponível no Twitter e pode ser agregado ao assistente pessoal o serviço de interoperabilidade de acesso ao Twitter.
- **Serviço de informações da empresa**: Retorna dados das filiais, recursos financeiros disponibilizados para os projetos e atividades dos usuários e outras informações pertinentes para cada caso.
- **Serviço de reserva em hotel**: Efetua a reserva e o pagamento para pessoas em um hotel específico, ou pode ser um serviço que trabalha com diversos hotéis cadastrados.
- **Serviço de reserva de passagens**: Efetua a reserva e o pagamento de passagens de avião, ou outro modo de transporte.

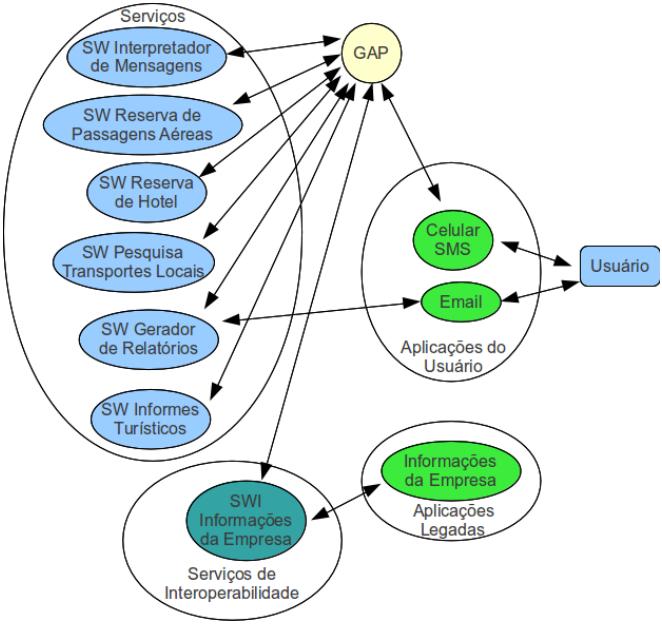

Figura 27: Caso de gerenciamento de viagens.

Este serviço de gerenciamento de viagem pode ser implementado de duas formas. Na primeira forma, o usuário é um usuário avançado e configura todas as informações, e a maneira como tudo vai ocorrer, no comportamento do seu assistente pessoal. Na segunda forma, a própria empresa, ou uma terceira empresa, já desenvolveu um comportamento para gerenciamento de viagens. Nessa segunda forma, o usuário apenas precisa configurar as informações que deverão ser utilizadas pelo comportamento específico. Este passa, então, a executar as tarefas necessárias para auxiliar o usuário.

3.5 MODELO GENÉRICO DE IMPLEMENTAÇÃO

O interesse de se apresentar um modelo genérico para a implementação do protótipo, é gerar um guia direcionando já para separação dos elementos da arquitetura. Apresentado na Figura 28, este modelo genérico utiliza os elementos conceituais da arquitetura de referência e já os direcionam para elementos implementáveis.

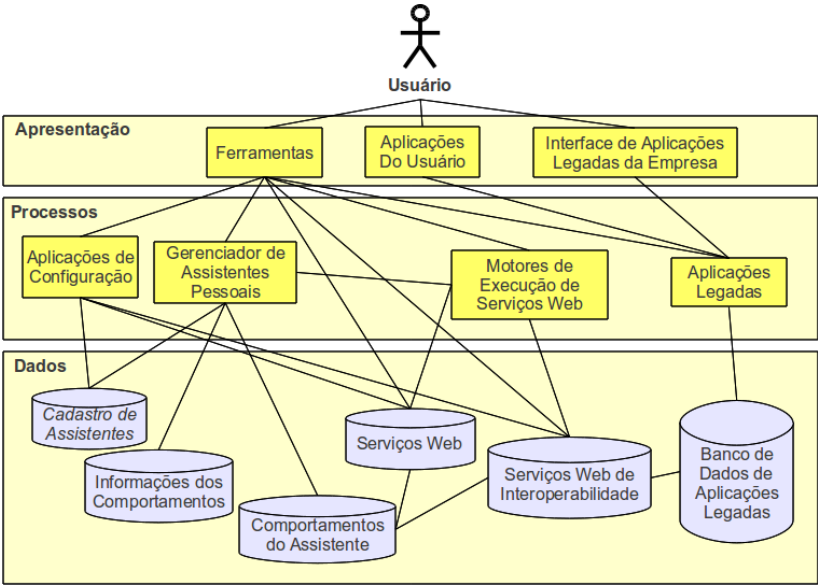

Figura 28: Modelo genérico para a implementação.

No caso dessa abstração da arquitetura de referência para implementação de módulos de sistema, e apresentado na Figura 28, há a divisão de três camadas básicas: a camada de apresentação, a camada de processos e a camada de dados.

3.5.1 Camada de Apresentação

A camada de apresentação é composta basicamente pelos elementos que fazem interface com o usuário, citados na definição da Arquitetura de Referência, no capítulo anterior. Os elementos da camada de apresentação são:

- **Ferramentas:** São softwares normalmente utilizados para efetuar a configuração de elementos dos assistentes pessoais e também para o desenvolvimento de elementos. Por exemplo, o assistente pessoal necessita de uma interface para que o usuário possa criar seu assistente pessoal e configurar seus comportamentos. Tal interface pode ser caracterizada como um software ou por uma aplicação web. Ferramentas de

desenvolvimentos podem ser IDEs para criação de serviços web, por exemplo, tal como a IDE Eclipse ou a IDE Netbeans.

- **Aplicações do Usuário**: São aqui definidas como as aplicações que o usuário utiliza no dia a dia. Podem ser aplicações de webmail, clientes de *Instant Messaging*, Twitter, cliente de e-mail, etc.
- **Interfaces de Aplicações Legadas da Empresa**: No caso da Arquitetura de Referência, esse elemento é apenas nomeado de **Aplicações Legadas**. Contudo, certas aplicações do usuário também podem ser aplicações legadas específicas de cada fornecedor, e que também deverão necessitar de serviços de interoperabilidade para que possam ser acessadas/gerenciadas. Contudo, neste elemento são definidas apenas as aplicações legadas da empresa em que o usuário trabalha e necessita do auxilio de um assistente pessoal para efetuar as tarefas no sistema. Diferenciando esse elemento da camada de processos, pode-se ver como interface alguma aplicação web e o processamento fica no servidor. Por definição, para essa implementação foi definido que a interface irá ser separada do processo.

Neste ponto já pode se definir alguns softwares a serem usados em determinados estudos de casos. Esses softwares e tecnologias são apresentadas mais a frente.

3.5.2 Camada de Processos

A camada de processos é caracterizada pelo processamento especifico para a execução de tarefas do assistente pessoal e outras aplicações que não fazem interface direta com o usuário. Os elementos dessa camada são:

- **Aplicações de configuração**: São os aplicativos responsáveis por auxilar o usuário a configurar seu assistente pessoal. Pode ser uma aplicação web que o usuário acessa (na interface) por meio de um navegador web.
- **Gerenciador de assistentes pessoais**: Este é o núcleo de execução do assistente pessoal. É o elemento que reúne as informações do usuário, as informações dos comportamentos, os serviços web que são invocados e a organização algorítmica

de como o comportamento é executado e "coloca tudo para funcionar".

- **Motores de execução de serviços web**: Aqui não considera especificamente em que linguagem o serviço web foi desenvolvido, ou como tal serviço deve ser executado. É considerado que existem repositórios de serviços web e cada repositório específico, implementado da forma específica de cada, é executado por algum motor de execução. Por exemplo, um serviço web desenvolvido em Java é colocado em um repositório de serviços e o motor de execução de serviços web Axis2, juntamente com o servidor web Apache Tomcat é chamado para executar o serviço, sendo caracterizado como o motor de execução do serviço web. Ou também, o serviço pode ser um *script* desenvolvido em linguagem PHP (*PHP: Hypertext Preprocessor*) e que é chamado pelo motor de execução do serviço web Apache. Outro exemplo é um serviço web, na forma de orquestração de serviços, desenvolvido em linguagem BPEL e chamado pelo motor de execução de serviços web Apache ODE (*Orchestration Director Engine*).
- **Aplicações Legadas**: Aqui é colocado o real processamento das aplicações legadas, com suas interfaces para o usuário na camada superior. Este elemento gerencia as informações da camada de dados, conforme requisições do usuário na camada de apresentação.

Esses quatro elementos podem, cada um especificamente, ser explodidos em diversos softwares e aplicativos, conforme é visto mais a frente do trabalho.

3.5.3 Camada de Dados

A camada de dados é caracterizada pelos elementos que são armazenados pelas aplicações e que são utilizados pelo usuário, assistentes pessoais, aplicações legadas. Esses elementos são:

- **Cadastro de Assistentes:** Cada usuário deve possuir seu próprio assistente pessoal. Assim, o usuário precisa acessar uma aplicação para criar seu assistente e para configurar suas informações básicas para a execução do mesmo.

- **Informações dos Comportamentos**: Para que cada comportamento possa ser executado para atender as necessidades de cada usuário, o comportamento precisa possuir um conjunto de informações que cada usuário deve configurar. Primeiro, cada serviço web precisa de informações que são enviadas como parâmetros e recebe um resultado da invocação. Essas informações precisam ser configuradas e deve haver uma informação que recebe o resultado. Segundo, cada comportamento pode efetuar diversas invocações de diversos serviços web diferentes, cada qual com suas próprias informações.
- **Comportamentos do Assistente**: O comportamento do assistente pode ser caracterizado por um *script* na forma de um algoritmo, ou em uma estrutura gráfica tal como um fluxograma ou qualquer outra forma de gerenciar a maneira como os serviços web são chamados. A interface para configurar tal comportamento ou a maneira como ele é visto pelo usuário é independente da Arquitetura de Referência e é de escolha de cada desenvolvedor, buscando melhor qualidade de criação de comportamentos ou buscando interfaces mais intuitivas para os usuários.
- **Serviços web**: Localizados distribuidamente na internet, são aqui considerados dados, na forma de linguagens de *scripts,* ou mesmo se forem programas, também são considerados dados que são chamados pelo motor de execução de serviços web para que eles se tornem operacionais.
- **Serviços web de interoperabilidade:** Os serviços web de interoperabilidade se diferem dos serviços web normais pois são desenvolvidos especificamente para fazerem a ponte entre o assistente pessoal, que só pode chamar serviços web, e alguma aplicação legada. Um serviço web de interoperabilidade pode invocar outros serviços, que não estão no padrão SOA OASIS (MacKenzie et al., 2006), ou acessar um banco de dados de uma aplicação legada do usuário, ou efetuar chamadas via soquetes a outra aplicação, ou outras formas.
- **Banco de dados de aplicações legadas**: São, especificamente, os dados utilizados pelas aplicações legadas do usuário, neste caso aplicações da empresa que o usuário trabalha e que necessita que o assistente pessoal o auxilie na tarefa de gerenciar tal aplicação.

3.6 CONSIDERAÇÕES SOBRE A PROPOSTA

Neste capítulo foi apresentada a proposta de uma Arquitetura de Referência para Softwares Assistentes Pessoais. Um modelo de referência foi criado para servir como base para a apresentação da Arquitetura de Referência. O estilo arquitetural escolhido foi a Arquitetura Orientada a Serviços, pois este cumpre devidamente os requisitos necessários para o tipo de Software Assistente Pessoal desejado.

A necessidade de se propor a Arquitetura de Referência em questão, surgiu da pesquisa sobre os estudos e produtos de assistentes pessoais via software de computador. Os produtos são fechados e não deixam opção ao usuário ampliar as atividades ou enquadrar as mesmas para suas próprias necessidades, ou mesmo aos processos de negócios da empresa.

Outra questão em aberto que foi encontrada é relacionada à interoperabilidade. Para que um assistente pudesse trabalhar em diversas frentes de tarefas, conforme as tarefas do usuário, seria necessário que pudesse haver diversas formas de se fornecer funcionalidades. Hoje, existem os serviços web como um padrão de fornecimento de serviços de software, com suas diversas vantagens de granularidade, distribuição de processamento, atividades especializadas fornecidas por terceiros, etc.

A utilização de tais recursos em assistentes pessoais se mostrou bastante atraente. Com esse estilo de desenvolvimento de software, o usuário do assistente pessoal pode utilizar recursos que já se encontram em um estado amadurecido de desenvolvimento.

Contudo, ainda há que se avaliar que muitos dos serviços que seriam necessários para que o assistente faça realmente o que o usuário quer, ou necessite, não estão ainda desenvolvidos. Provavelmente, muitos serviços teriam que ser desenvolvidos pela própria empresa, ou por usuários mais especializados, com o objetivo de prover ao assistente pessoal a possibilidade efetiva de ajudar o usuário. Em outro sentido, empresas podem ser incentivadas a desenvolverem serviços especificamente para assistentes pessoais que se utilizem de serviços web da especificação OASIS (MacKenzie et al., 2006) como comportamento.

De forma complementar, esse capítulo mostrou uma justificativa da utilização de SOA como estilo arquitetural da arquitetura de referência, fazendo um paralelo com requisitos, considerados desejáveis

para Softwares Assistentes Pessoais pelo autor da proposta, com as características e vantagens da especificação SOA da OASIS (MacKenzie et al., 2006).

Como continuidade deste trabalho e como uma das formas de avaliação da proposta, a seguir é apresentada a implementação de uma instância da Arquitetura de Referência para Softwares Assistentes Pessoais aqui proposta.

4 INSTANCIAÇÃO DA ARQUITETURA DE REFERÊNCIA E IMPLEMENTAÇÃO DE PROTÓTIPO

Uma das formas de verificação da proposta desta Tese é a criação de uma instância implementada baseada na Arquitetura de Referência, de modo a demonstrar a viabilidade da sua utilização para a criação de assistentes pessoais que estejam em concordância com os requisitos apresentados no presente trabalho.

Para tal, foram desenvolvidas três versões de protótipos que serviram como base para que se efetuassem avaliações em cada fase importante da maturação da proposta. As ideias e modelos foram testados com base nas definições encontradas durante o percurso.

A proposta apresentada no capítulo anterior foi utilizada para fomentar a criação do protótipo final que agora é apresentado. Para tal, um estudo de caso, modelado a partir dos dois primeiros estudos de casos apresentados no capítulo anterior foi escolhido para a elaboração de testes.

É relevante citar que aqui é implementada apenas uma instância para verificação e testes sobre a arquitetura de referência proposta. Contudo, ela é representativa porque segue todos os requisitos definidos pela proposta. Se outras instâncias forem implementadas, mesmo que utilizando tecnologias de implementações diferentes, mas seguindo as definições da proposta, elas também devem funcionar.

Este capítulo inicia com a apresentação de um modelo genérico de implementação, baseado na arquitetura de referência, e que deve servir como guia para a implementação. Depois é escolhido e descrito um estudo de caso específico de auxílio ao usuário em uma atividade na empresa. Juntamente, são apresentados os elementos e tecnologias utilizadas para a implementação e a forma organizacional de como esses elementos foram reunidos para implementar o protótipo.

4.1 CASO DE IMPLEMENTAÇÃO

Este caso é modelado a partir dos dois primeiros exemplos apresentados no capítulo anterior, que são o processo de compra automatizado e o gerador de relatórios. Contudo, neste ponto há um nível maior de detalhamento, suficiente para que o mesmo seja

direcionado para a implementação do protótipo que deve servir como um dos pontos de avaliação dessa Tese.

O cenário a ser considerado para o caso de implementação é de uma pessoa, funcionário de uma empresa, que tem como uma de suas funções, efetuar o gerenciamento do estoque de produtos de uma empresa. Quando o estoque de um determinado produto passa de um limite mínimo, fica a cargo desse funcionário efetuar nova compra desse produto, de modo a manter o estoque sempre em dia.

O funcionário possui acesso a um sistema legado de controle de estoque, com seu próprio banco de dados, por meio de um login de usuário e uma senha. Conforme os produtos vão sendo utilizados ou vendidos, sua quantidade em estoque vai diminuindo. Ao final do dia, o funcionário faz um relatório das atividades realizadas por ele durante todo o dia. É relatado o início do processo de uma compra a um determinado fornecedor, modificações em uma ordem de compra e fechamento de uma compra.

Em tempo, um sistema auxiliar para verificação e avaliação de e-mails para o usuário, quando o mesmo não está online, é criado como forma de assistência ao usuário.

Sob esse cenário, entra então a possibilidade do funcionário possuir um assistente pessoal na forma de um software de computador. Este deve auxiliá-lo em suas tarefas diárias, podendo ser desde tarefas rotineiras em casa, até tarefas na empresa. Neste caso, seu assistente pessoal deve ser direcionado a ajudar o funcionário a gerenciar o estoque, e na criação do relatório das atividades diárias das compras.

Tendo como base o cenário acima citado, a Arquitetura de Referência proposta e o modelo genérico de implementação apresentado, pode-se modelar os elementos a compor o assistente pessoal para auxiliar a pessoa que têm a função de gerenciamento do sistema de estoque (Figura 29).

É importante salientar que a escolha das tecnologias de implementação, formas de acesso ao assistente pessoal, funcionalidades dos serviços e outros, são neste ponto de escolha do desenvolvedor, no caso do autor da Tese. Contudo a implementação é efetuada com base na proposta em todos os seus níveis, conforme mostrado na Figura 31.

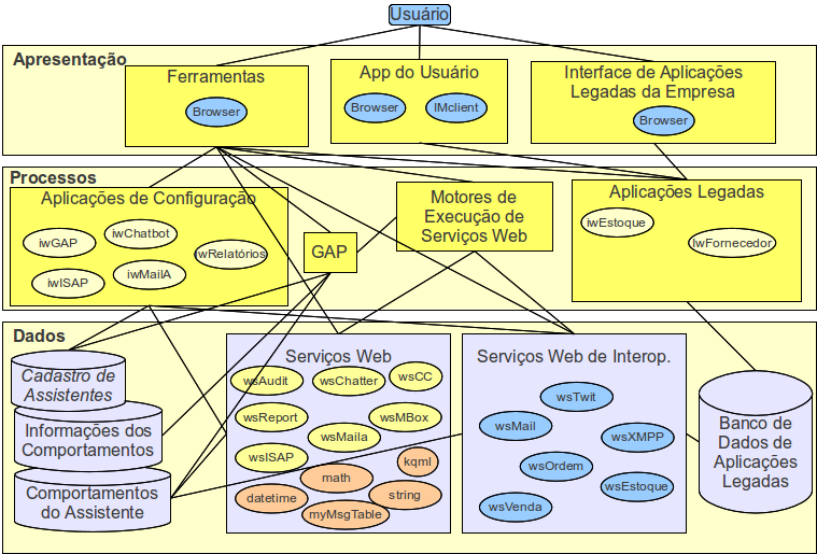

Figura 29: Elementos alocados no modelo de implementação.

Tomando como base as Figura 28 e Figura 29, que apresentam um modelo genérico de implementação e os elementos alocados no modelo de implementação, separado em três camadas, os elementos do sistema como um todo são agora apresentados.

4.1.1 Camada de Apresentação

Para essa implementação, optou-se que todo acesso aos sistemas envolvidos se dão via web, por meio de um navegador web. Os sistemas legados de controle de estoque (iwEstoque) e de controle de estoque do fornecedor (iwFornecedor) são desenvolvidos como aplicações web.

As interfaces para configuração dos serviços que o assistente pessoal utiliza (iwGap, iwISAP, iwChatbot e iwRelatório) são interfaces web. O único elemento que é um programa de computador é o cliente de *Instant Messaging*.

Assim, considera-se a apresentação por um sistema de acesso a web, que é o navegador web e um software de *Instant Messaging*, sem deixar de observar que cada programa com interface web, aqui utilizado, possui sua própria forma de apresentação de interface e interação com o usuário.

4.1.2 Camada de Processo

Na camada de processos está localizado o **Gerenciador de Assistentes Pessoais** (GAP). Este é o elemento que se utiliza das informações do usuário e de cada comportamento, juntamente com a forma de execução do comportamento e os executa, efetuando as devidas chamadas a serviços web. Também estão localizados os **Motores de Execução de Serviços Web**. Este é responsável por carregar o serviço web na memória e efetuar sua execução. Considera-se apenas que um serviço é uma informação que é carregada em um motor de software que a executa.

Das aplicações Legadas tem-se:

- **iwEstoque:** Sistema com seu próprio banco de dados e interface web da qual o usuário necessita efetuar o gerenciamento. No caso, o assistente pessoal irá auxiliar o usuário nessa tarefa.
- **iwFornecedor:** Sistema com seu próprio banco de dados e interface web que gerencia o estoque no fornecedor. Este sistema é necessário para que possa ser feita de forma efetiva a simulação o processo de compra pelo cliente. Se um produto estiver com estoque baixo na hora da venda, a ordem de venda é modificada e o cliente deve estar ciente das modificações.

Das Aplicações de Configuração tem-se:

- **iwGAP:** Ambiente web de configuração do assistente pessoal. Por meio deste sistema, o usuário efetua o cadastro do seu assistente pessoal, cadastra comportamentos e os configura, com informações gerais que todos os comportamentos podem utilizar, com suas próprias informações, com a maneira como o comportamento deve ser executado e com suas chamadas à serviços web.
- **iwRelatório:** Sistema com interface web para a criação de um usuário do sistema de geração automática de relatórios e para gerenciar as informações deste usuário.
- **iwISAP:** Sistema web para configuração do elemento "Interface Social para Assistentes Pessoais". A função desse elemento é efetuar uma interface entre o usuário e o assistente pessoal. O usuário deve se comunicar por meio de mensagens de twitter, Gtalk ou SMS de celular. As mensagens são filtradas e apenas o que é relevante ao assistente pessoal é enviado ao

GAP. Da mesma forma, o GAP envia mensagem ao ISAP, que deve processá-la antes de enviar ao usuário.

- **iwMailA:** Interface para configuração do assistente de leitura de e-mails. É possível cadastrar filtros e para cada filtro um conjunto de regras.
- **iwChatbot:** Quando um usuário entra em contato com seu assistente pessoal, enviando uma mensagem ou uma requisição de tarefa, ou mesmo quando alguma outra pessoa deseja entrar em contato com o usuário, pode enviar mensagens ao assistente pessoal. O elemento de *chatbot* serve para retornar um *feedback* ao usuário de uma forma mais amigável, como se fosse uma conversa com alguma pessoa real, tal como o Cleverbot[11]. A interface de configuração do *Chatbot* serve para configurar as conversas com o usuário e verificar os *logs* de conversas com as pessoas.

A forma como esses elementos estão dispostos na arquitetura e são implementados é apresentada mais a frente.

4.1.3 Camada de Dados

Na camada de dados encontram-se os bancos de dados das aplicações legadas, o cadastro de assistentes pessoais, com informações de cada assistente cadastrado. O sistema permite que possam haver vários usuários de assistentes pessoais, cada um com suas informações, comportamentos, etc.

Cada comportamento também necessita das informações necessárias para que cada um seja executado e de uma base de dados que é definida a forma como o comportamento é executado e os serviços web são instanciados.

Os serviços web, também considerados aqui como estando na camadas de dados, são os seguintes:

- **wsAudit:** é utilizado para auditoria geral. Neste serviço são enviadas informações que podem servir para, posteriormente avaliar como o sistema está sendo executado. Útil para a avaliação final do trabalho.

[11]O Cleverbot é uma máquina de inteligência que venceu um concurso de teste de Turing no ano de 2010 (Cleverbot, 2011).

- **wsChatter**: serviço bastante simples, que recebe uma frase e retorna uma resposta. Têm a finalidade de criar uma forma mais natural de interação com o usuário.
- **wsCC**: simulação de transação com cartão de crédito. Utilizado pelo comprador e pelo vendedor.
- **wsMaila:** avalia os e-mails do usuário e retorna mensagens que deve ser enviadas ao mesmo, caso seja de algum assunto específico e com as características definidas pelo próprio usuário.
- **wsReport**: recebe informações de tarefas realizadas e gera um relatório dessas quando requisitado. As tarefas são separadas em tarefas públicas e tarefas privadas. Isso tem consequência posteriormente na hora da forma que o relatório é apresentado ao usuário.
- **wsMBox**: gerenciador de mensagens, separa conforme o comportamento que deve receber a mensagem. As mensagens provindas do ISAP, são requisitadas pelo GAP. Essas mensagens podem ser direcionadas a comportamentos específicos. Assim, para que as mensagens não se percam, o GAP as envia para o wsMBox especificando de qual comportamento é a mensagem, e quando o comportamento requisitá-la, o wsMBox as retorna.
- **Libs**[12]: Esses são serviços que podem ser utilizados pelos comportamentos para funções especificas de manipulação de informações e outros. Neste caso, há algumas implementadas que são:
    - **kqml**: manipulação de mensagens no formato kqml sob XML;
    - **math**: funções matemáticas;
    - **datetime**: funções de data e horário;
    - **string**: funções de manipulação de strings;
    - **myMsgTable**: manipula uma estrutura própria de tabelas. Desenvolvido e utilizado pelo desenvolvedor.

O KQML (*Knowledge Query and Manipulation Language*) é uma linguagem e um protocolo para troca de informações e conhecimentos. Ela pode ser utilizada para a interação e troca de conhecimentos entre sistemas inteligentes em prol da resolução cooperativa de problemas (Hunhs, 1999). Ela é utilizada aqui por ser um padrão e já possuir uma

[12]O nome **Libs** vem de *libraries*, ou bibliotecas de funções diversas.

certa maturidade. Contudo, ainda assim, é apenas uma escolha do autor dessa implementação e que não compromete de forma alguma a proposta.

Há também os serviços de interoperabilidade, que têm a função de criar interoperabilidade de algum elemento que não se encontra no padrão de serviços web da OASIS (MacKenzie et al., 2006). Para a implementação do caso em questão, são apresentados os seguintes serviços de interoperabilidade:

- **wsTwit:** interoperabilidade com o serviço de microblog Twitter;
- **wsMail**: interoperabilidade com serviços de e-mail via protocolo POP3;
- **wsXMPP**: interoperabilidade com serviços de *Instant Messaging* que funcionam sob o protocolo XMPP (*eXtensible Messaging and Presence Protocol*). No caso do protótipo, é utilizado o Gtalk.
- **wsEstoque**: interoperabilidade com o sistema de controle de estoque, via banco de dados do sistema.
- **wsOrdem**: para gerenciamento das ordens de compra no cliente.
- **wsVenda**: para contatar um fornecedor e negociar a compra de produtos.

Todos os serviços acima citados podem ser utilizados pelo GAP e também por outros serviços.

4.2 TECNOLOGIAS DE IMPLEMENTAÇÃO

As tecnologias selecionadas (Figura 30) para as implementações apresentadas nesse capítulo são classificadas em aplicações servidoras, aplicações web para configuração, serviços web e são citados os softwares que são utilizados para acessar essas implementações.

Na implementação deste protótipo, todos os serviços web foram implementados em linguagem de programação PHP[13], e estão funcionando sob o servidor Http Apache[14]. São utilizadas as funções de

[13]PHP significa "*PHP: Hypertext Preprocessor*" e é uma linguagem de programação interpretada, direcionada para desenvolvimento para a Web e pode ser mesclada dentro do código HTML. A sintaxe da linguagem é parecida com C, Java e Perl (PHP, 2010).

[14]O projeto do servidor Apache HTTP é um esforço da *Apache Software Fundation* de desenvolvimento *opensource* para sistemas operacinoais modernos. O intuito é fornecer um

SOAP fornecidas pela própria linguagem PHP, juntamente com uma biblioteca desenvolvida pelo autor dessa Tese, para efetuar a geração automática de arquivos WSDL.

Figura 30: Elementos de utilização e implementação.

Todas as interfaces dos sistemas foram desenvolvidas para serem acessadas via navegador web e em três camadas. Elas foram desenvolvidas em linguagem PHP para os processos, HTML (*Hyper Text Markup Language*) para a apresentação e para o armazenamento das informações, ou dados, foi utilizado o banco de dados Postgresql[15]. Um simples *framework* desenvolvido pelo autor da Tese foi utilizado para efetuar a separação da camada da apresentação da camada de processos na forma da utilização de *templates* de páginas HTML. Uma vez que as interfaces não são o ponto focal para a avaliação do sistema, elas não são aqui apresentadas de forma exaustiva, a não ser a interface de configuração do GAP.

Os servidores foram desenvolvidos em linguagem de programação Java, e utilizaram bibliotecas específicas para cada tipo de necessidade que cada sistema, e são apresentadas na descrição de cada servidor.

4.3 ELEMENTOS DE IMPLEMENTAÇÃO

Como forma de separar os elementos conforme tecnologias utilizadas para o desenvolvimento, neste ponto é inicialmente feito um paralelo dos elementos do caso instanciado com a Arquitetura de Referência apresentada na Figura 21. Assim, os elementos ficam

servidor seguro, eficiente, extensível e em sincronia com os padrões HTTP (APACHE, 2010).

[15]O PostgreSQL é um banco de dados objeto-relacional *opensource.* Ele roda nos principais sistemas operacionais, incluindo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), e Windows. Possui interfaces de programação nativa para C/C++, Java,. Net, Perl, Python, Ruby, Tcl, ODBC, entre outros (POSTGRESQL, 2010).

dispostos na arquitetura separados pelas aplicações do usuário, aplicações legadas, serviços de interoperabilidade, federação de serviços e ferramentas, conforme visto na Figura 31.

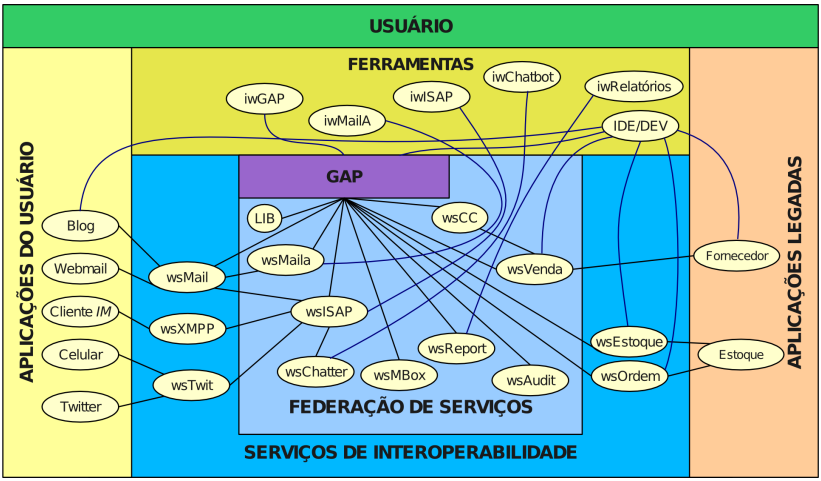

Figura 31: Elementos de implementação na arquitetura de referência.

A visão apresentada na Figura 31 relaciona de forma composicional os elementos sob a visão da arquitetura de referência proposta. Contudo, ainda é insuficiente para mostrar a estrutura tecnológica física da organização desses elementos, ou seja, como o sistema, como um todo, é executado. Para suprir a visão das necessidades de hardware do sistema, servidores físicos, estações e estrutura física distribuída, a Figura 32 é apresentada:

A Figura 32 apresenta a forma como os elementos estão distribuídos. Entre esses elementos, estão:

- **Google:** Para o exemplo de caso são utilizados alguns sistemas gratuitos da Google como servidor de *Instant Messaging* Gtalk, Servidor de e-mails Gmail e Serviço de blog Blogger.
- **Operadora de Telefonia Móvel:** Recurso para a troca de mensagens de SMS entre usuário e SAP via telefonia móvel.
- **Servidor Twitter:** Para troca de mensagens de twitter entre o usuário e seu SAP. O Twitter também tem o intuito direto de ser o elemento intermediário entre a troca de mensagens SMS por possuir tal recurso. Apenas as mensagens setadas como "privadas" ou *Direct Message* são configuradas para serem enviadas via SMS.

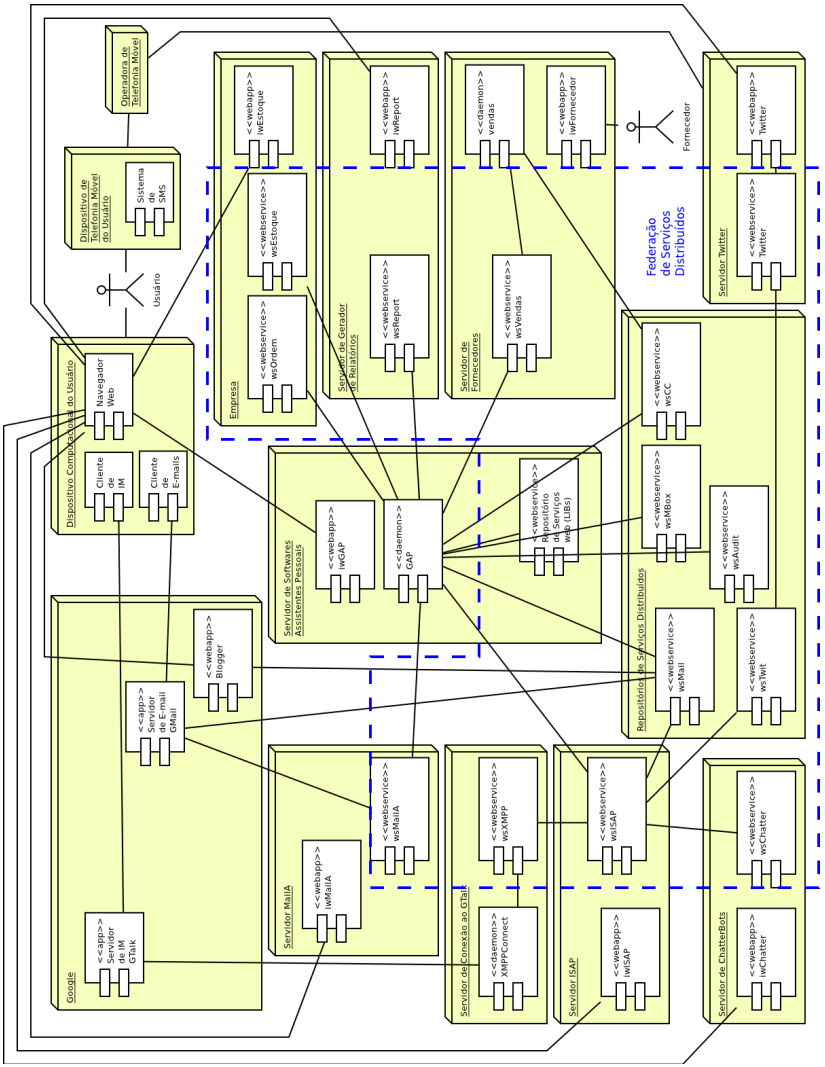

Figura 32: Diagrama de Implantação do Exemplo para Implementação.

- **Dispositivo Computacional do Usuário:** Para interagir com seu SAP, o usuário necessita de algum ambiente computacional, podendo ele ser um *smartphone* ou computador *desktop*. O cliente de *Instant Messaging* tem o propósito de servir como elemento de interação direta via troca de mensagens de texto, o navegador web tem o propósito de servir como programa de

acesso aos sistemas desenvolvidos para web utilizados no sistema e o cliente de e-mails para recuperar e enviar e-mails. Este último é opcional com a utilização de um *webmail*.

- **Dispositivo de Telefonia Móvel do Usuário:** O principal propósito do dispositivo móvel, em relação ao SAP, é a troca de mensagens entre usuário e SAP. Contudo, quando se tratando de um *smartphone*, sua utilização pode ser a mesma citada no Dispositivo Computacional do item anterior.
- **Empresa:** Refere-se aos sistemas da empresa que o usuário trabalha que quer que seu assistente pessoal utilize. Como forma de disponibilizar a utilização dos sistemas para que o assistente pessoal possa utilizar, são também implementados e implantados serviços web no servidor da empresa.
- **Servidor de Fornecedores:** Tal como os sistemas da empresa do usuário, o exemplo apresentado para testes e implementação do SAP necessita poder acessar um conjunto de fornecedores. Para tal, foi implementado um servidor capaz de suportar vários fornecedores com produtos, preços, quantidade em estoque para venda, etc. Também foi implementado um serviço web para acesso aos fornecedores, com envio e verificação de ordens de compra. Um servidor fica responsável por gerenciar as ordens de compra abertas, verificação de pagamento, fechamento e cancelamento de ordens de forma automática. Tal recurso é utilizado como forma de automatizar a interação dos usuários que gerenciam os sistemas de fornecimento de produto.
- **Servidor de Gerador de Relatórios:** Por meio desse recurso, o assistente pessoal envia atividades que estão sendo efetuadas ou foram finalizadas. Essas tarefas são compiladas em um único relatório que pode ser enviado ao usuário por e-mail ou publicadas no blog do SAP.

Em sequência, os elementos envolvidos são descritos de forma mais detalhada. Essa descrição parte de uma perspectiva *bottom-up* de descrição do sistema em partes para, por fim, emergir o sistema como um todo (Figura 33).

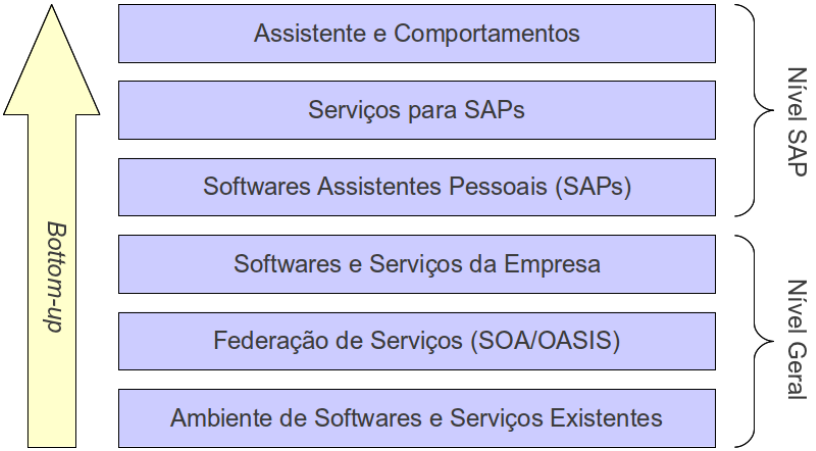

Figura 33: Visão bottom-up do sistema.

Na Figura 33 pode-se identificar inicialmente dois níveis básicos, o Nível Geral e o Nível SAP. Esses são descritos a seguir:

- **Nível Geral:** elementos que podem já estar implementados, em funcionamento e não estão diretamente atrelados aos SAPs, ou seja, não foram desenvolvidos especificamente para serem utilizados por SAPs.
    - **Elementos Existentes:** Elementos já implementados, que estão funcionando e que já são utilizados pelo usuário no dia a dia. Esses elementos não são necessários implementar;
    - **Federação de Serviços:** Elementos que são vistos pelo assistente pessoal na forma de serviço (incluindo os serviços de interoperabilidade) distribuídos em repositórios na internet. Outros elementos e implementações necessárias para a utilização desses serviços também são descritos nessa etapa;
    - **Sistemas e Serviços da Empresa:** Inclui-se aqui os sistemas legados da empresa utilizados pelos usuários e serviços necessários para que tais sistemas legados possam ser acessados externamente por outros softwares que mantém a compatibilidade com SOA/OASIS.
- **Nível SAP:** elementos que foram desenvolvidos especificamente para SAPs e a utilização de SAP especificamente.

- **Software Assistente Pessoal:** Implementação dos elementos do SAP, incluindo a interface de criação e configuração, o servidor de execução e alguns serviços básicos para sua utilização;
- **Interface Social para Assistentes Pessoais:** Implementação de um serviço específico para ser utilizado por SAPs. Este serviço define a forma de interação que se dá entre usuário e SAP no exemplo apresentado. Também é apresentada a interface de configuração desse serviço.
- **Assistente e Comportamentos:** Criação e configuração de um assistente pessoal, incluindo a utilização dos elementos de interação utilizados, serviços e criação dos comportamentos.

Com relação aos elementos existentes que são utilizados e que não precisam ser implementados, apenas uma breve descrição é apresentada, sua forma de utilização ou configuração é mostrada no momento da criação/configuração do assistente pessoal, incluindo todos os elementos necessários para a sua utilização.

4.3.1 Ambiente de Softwares e Serviços Existentes

A classificação de Elementos de Softwares e Serviços Existentes no contexto desse trabalho é referente às tecnologias já utilizadas pelo usuário no seu dia adia e que podem também ser utilizadas como meio de interação entre o SAP e o usuário. Para essa interação, algumas tecnologias foram selecionadas (Figura 34).

Figura 34: Elementos existentes.

As implementações que são utilizadas para fins de interfaceamento entre o usuário e o assistente pessoal, nesta instâncias de implementação, foram direcionadas para serem o menos intrusivas o possível, tal como é feita na primeira implementação, em Zambiasi e Rabelo (2007) e no estilo de trocas de mensagens via *Instant Messaging*[16] (IM), Mensagem de texto pelo celular e conversa via microblog Twitter, tao como no software Sandy, citado por Mann (2011).

Um ponto importante quanto a interação entre o usuário e o assistente pessoal é que, uma vez que o usuário se comunica via mensagens, comparativamente como se fosse uma comunicação online com um secretário real, então devem ser criadas contas em cada serviço específico que o usuário se comunica com seu assistente pessoal, tal como conta de e-mail (no caso o Gmail, para ser usado no Gtalk), conta de twitter, blog, etc.

O *webmail*, no caso do Gmail aqui utilizado, é uma forma de acessar uma conta de e-mails via interface web. Sua vantagem está em poder acessar seus e-mails de qualquer computador, ou *smartphone*, que possua acesso à Internet. Contudo, não é obrigatório o uso de um webmail. O usuário pode fazer uso de uma conta de e-mails, buscar e enviar mensagens por meio de um programa de e-mails específico, como por exemplo o Mozilla Thunderbird[17], utilizando os protocolos POP3 (*Post Office Protocol*) para buscar as mensagens e SMTP (*Simple Mail Transfer Protocol*) para enviar.

A conta de e-mail do usuário serve tanto para se comunicar com seu assistente pessoal via troca de e-mails, como também para receber relatórios de classificação privadas que devem ser enviadas pelo GAP.

As contas de e-mail criadas estão situadas no sistema de e-mails da Google[18] e possuem também a vantagem de já possuir também uma conta no sistema de IM no google, denominado Gtalk[19], e que trabalha sob o protocolo XMPP (*eXtensible Message and Preference Protocol*).

---

[16]Programa que permite a troca de mensagem de texto entre o usuário e pessoas cadastradas como contato. Alguns possuem recursos de troca de arquivos e comunicação via voz e/ou vídeo. Alguns exemplos são: ICQ, Windows Live Messager, Gtalk, Pidgin, etc.

[17]O Thunderbird é um program de gerenciamento de e-mails da Mozilla, desenvolvido para várias plataformas, incluindo Linux, MacOS e Windows. Ele possui filtro anti-spam, pastas de pesquisa, marcadores, proteção à privacidade e diversos outros recursos (THANDERBIRD, 2010).

[18]O Gmail é um sistema de e-mails da Google online (GMAIL, 2010).

[19]O Gtalk é um sistema de gerenciamento de Instant Messaging da Google, e trabalha em conformidade com o protocolo XMPP (GTALK, 2010).

Esta forma de troca de mensagem também é utilizada pelo usuário para se comunicar com seu assistente pessoal, de modo online e privado.

Em tempo, um sistema de Blog é um outro recurso alternativo também utilizado para o envio de relatórios para o usuário, de classificação pública, ou seja, o relatório pode estar disponível para outras pessoas visualizarem. Neste caso, o usuário possui acesso a conta no Blogger[20], assim como também seu assistente pessoal. Neste, há um recurso que habilita a um usuário do blog enviar uma postagem via e-mail.

O serviço de microblog Twitter[21] é uma espécie de rede de informações que trabalha em tempo real. Pequenas mensagens, chamadas *Tweets,* de 140 caracteres podem ser enviadas pelo usuário e compartilhada com seus contatos e, se permitido pelo usuário, visualizada por todos que acessam o serviço.

O acesso ao Twitter pode se dar via navegador web ou por algum aplicativo específico, tanto para computador quanto para celulares *smartphones*.

Uma das vantagens do Twitter é que o usuário do serviço pode enviar *tweets* via mensagem de texto pelo celular. Ou receber mensagens classificadas como *Direct Messages* (DM) também como mensagens de texto recebidas no celular. Tal característica permite que o usuário possa se comunicar com seu assistente pessoal mesmo que não haja acesso a internet, contanto que o mesmo esteja de posse do seu celular, que o tenha cadastrado no Twitter e que haja conexão com a antena de celular. Para tal, também é necessário os serviços de SMS de uma operadora de telefonia móvel.

4.3.2 Federação de Serviços

Por Federação de Serviços têm-se o conjunto de serviços (no padrão SOA/OASIS) distribuídos na internet e que podem ser utilizados para as mais diversas finalidades. No caso, aqui são descritos e implementados apenas os serviços que se fazem necessários para o caso de implementação selecionado.

---

[20]O Blogger é um sistema de publicação estilo Blog. Ele foi fundado por uma empresa de San Francisco (EUA) chamada Pyra Labs, e adquirida pela Google (BLOGGER, 2010).

[21]O Twitter é um serviço de microblog, ou seja, uma rede de informações em tempo real. Pequenos envios de informações do tamanho de 140 caracteres são chamadas de tweets e são compartilhadas com pessoas que estão conectadas à conta do usuário (TWITTER, 2010).

4.3.2.1 Serviço Geração Automática de Relatórios

O serviço web wsReport tem a função de receber informações de tarefas realizadas e, quando requisitado retorná-las juntas em um relatório único. Sua base de dados é formada por três relações, *Users*, *Text* e *Task*, apresentadas na Figura 35.

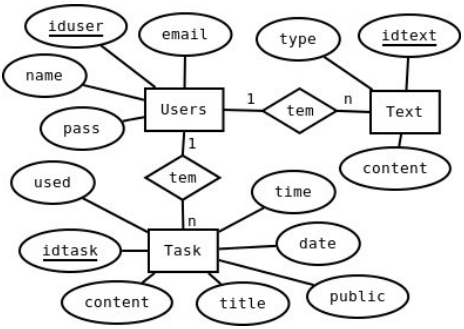

Figura 35: Diagrama ER do gerador de relatórios.

O sistema permite múltiplos usuários, sendo que cada qual pode ter suas próprias informações e gerar seus próprios relatórios. As informações dos usuários são armazenadas na relação *Users* (Tabela 2)

Users		
Atributo	**Tipo**	**Descrição**
** iduser*	*string*	Login do usuário
name	*string*	Nome do usuário
pass	*string*	Senha para autenticação do usuário
email	*string*	E-mail do usuário.

Tabela 2: ER wsReport - Relação *Users*.

A relação *Task* (Tabela 3) se refere as atividades do usuário, enviadas para que possam ser reagrupadas no relatório, quando requerido.

Task		
Atributo	**Tipo**	**Descrição**
iduser	*string*	Chave estrangeira para a relação *Users*
* *idtask*	*int*	Identificador único da tarefa, auto incrementável
title	*string*	Título da tarefa
content	*text*	Conteúdo da tarefa realizada
date	*date*	Dia que a tarefa foi enviada para o sistema de relatórios
time	*time*	Hora que a tarefa foi enviada para o sistema
used	*boolean*	*True* se a tarefa já foi utilizada para gerar um relatório
public	*boolean*	*True* se a terafa for publica, senão *false*

Tabela 3: ER wsReport - Relação *Task*.

A relação *Text* é para armazenar textos, ou frases, que são utilizadas para modelar um texto inicial para o relatório. Este recurso visa apresentar o relatório de forma mais humanizada, e torna-se mais interessante ao apresentar o relatório como público, ao enviá-lo na forma de postagem em *blog*. Seus atributos estão apresentados na Tabela 4.

Text		
Atributo	**Tipo**	**Descrição**
iduser	*string*	Chave estrangeira para a relação *Users*
* *idtext*	*int*	Identificador único do texto, auto incrementável
type	*string*	Tipo do texto (Introdução, Desenvolvimento ou Conclusão)
content	*text*	Conteúdo do texto, frase.

Tabela 4: ER wsReport - Relação *Text*.

Para que os textos sejam modelados no sistema, esses devem ser escritos conformes o tipo específico, para que ele seja modelado de forma conveniente. O texto inicial do relatório é montado com uma frase de “introdução”, um número aleatório de até 5 frases de “desenvolvimento”, pegas aleatoriamente, e uma frase de “conclusão”.

O sistema se utiliza de uma interface web para a configuração dos relatórios. Essa interface também se utiliza da base de dados mostrada no diagrama entidade-relacionamento da Figura 35. O sistema de

configuração é desenvolvido na forma de uma interface web e é dividida em três partes principais, conforme *menu* do sistema visto na Figura 36.

Figura 36: Interface web do gerador de relatórios - visualização de tarefas.

As operações em cada item do *menu* são as seguintes:

- **Configurações:** o usuário do sistema pode alterar as suas informações e senha de acesso ao sistema. Essas informações são referentes à relação *User*. É importante salientar que caso a senha seja alterada aqui, essa também deve ser atualizada no GAP.
- **Tarefas:** neste item é possível visualizar as tarefas que não foram utilizadas para a geração de um relatório e também as que já foram utilizadas.
- **Textos:** Cadastra frases/textos para serem utilizadas na montagem do parágrafo inicial do relatório, são divididos em três tipos: introdução, desenvolvimento e conclusão.
- **Sair:** efetua o *logout* do sistema.

Na tela de login do sistema, é possível fazer o cadastro da conta de um novo usuário. Como esse sistema é um protótipo, nenhum teste é feito sobre a validade das informações, a não ser se o login escolhido já existe ou se a senha confere com a senha redigitada.

4.3.2.2 Serviço de Integração ao Twitter

O serviço web wsTwit é uma interface SOA para gerenciamento da conta de Twitter de um usuário. Para que o sistema possa se conectar ao Twitter e gerenciar a conta do usuário é necessário que a implementação trabalhe com o protocolo OAuth[22] (*Open Authorization*). Este é um padrão aberto para autorização que permite que vários usuários possam compartilhar recursos privados.

Para o acesso ao serviço de microblog, está disponível uma forma de serviço web no formato JSON (*JavaScript Object Notation*). O JSON se caracteriza como um padrão para troca de dados completamente independente de linguagem de programação (JSON, 2011). Por meio de uma chamada a um site, com seus devidos parâmetros, o site faz um retorno das informações organizadas em um arquivo XML. No caso desse protótipo, foi utilizada uma classe de abstração em linguagem de programação PHP desenvolvida por Abraham Williams[23], chamada TwitterOAuth.

No caso do Twitter, é necessário que se efetue a autorização de um aplicativo na página de desenvolvedores[24], para que este possa fazer acesso ao sistema. Após o desenvolvedor ter configurado a aplicação, é são retornadas duas informações importantes, a ***Consumer key*** e a ***Consumer secret***. Essas são utilizadas na aplicação, juntamente com a autorização do usuário em duas informações também: ***oauth_token*** e ***oauth_token_secret***. Essas duas últimas são geradas automaticamente quando o usuário acessa o twitter para autorizar o sistema acessar suas informações. A forma como essa autorização é feita é apresentado na descrição da interface web do ISAP.

Todas as operações do wsTwit, com exceção de *getConsumerKey* e *getConsumerSecret* possuem como os dois primeiros parâmetros as informações *token* e *token_secret*, necessárias para identificar o usuário. Elas são descritas a seguir:

- ***getConsumerKey*:** retorna a *Consumer key* da aplicação;
- ***getConsumerSecret*:** retorna a *Consumer secret* da aplicação;

[22]OAuth é um protocolo aberto para permitir autorização segura de APIs em um simples e padronizado método para *desktop* e aplicações web (OAUTH, 2010).

[23]http://abrah.am – Abraham Williams *website*.

[24]Comunidade de desenvolvimento de aplicações para integração com o Twitter (TWITTER, 2010b).

- ***updateStatus*:** publica uma nova mensagem pública no Twitter. O terceiro parâmetro é a mensagem em si, que deve contar no máximo 140 caracteres;
- ***sendDirectMessage*:** envia uma mensagem direta a outro usuário do Twitter. Seus parâmetros subsequentes são:
  - ***user*:** id do contato no Twitter;
  - ***text*:** mensagem;
- ***getDirectMessage*:** retorna mensagens diretas;
  - ***since_id*:** cada mensagem possui um id numérico. Este parâmetro informa que devem ser retornadas as mensagens que vêm a partir da mensagem de id *since_id*;
  - ***count*:** informa a quantidade de mensagens diretas a retornar;
- ***getSentDirectMessages*:** retorna as mensagens diretas enviadas pelo próprio usuário. Seus parâmetros de entrada subsequentes são *since_id* e *count*, tal como na operação anterior;
- ***getUserTimeline*:** retorna as informações da *timeline* do usuário, ou seja, as informações que se encontram o espaço onde o usuário publica suas mensagens. Seus parâmetros de entrada são o *id* do usuário da qual se deseja retornar a *timeline*, o *since_id* e o *count*, assim como em *getDirectMessage*;
- ***getFriendsTimeline*:** retorna a *timeline* do usuário, com as informações visualizadas de todos seus contatos. Possui os parâmetros subsequentes *since_id* e *count*, conforme as operações anteriores;
- ***getMentions*:** retorna as mensagens de contatos que mencionaram o id do usuário em questão. Possui os parâmetros subsequentes de *since_id* e *count* também;
- ***getUser*:** retorna informações de um usuário específico. Passa-se como parâmetro, após *token* e *token_secret* o *id* do usuário da qual se deseja obter informações;
- ***getFollowers*:** retorna uma lista de seguidores de determinado usuário. Passa-se como parâmetro subsequente o *id* do usuário que se quer buscar os seguidores;
- ***deleteDirectMessage*:** remove a mensagem direta do usuário, especificada pelo terceiro parâmetro, ou seja, o *id* da mensagem;
- ***follow*:** habilita um usuário do Twitter como *follower*. Passa-se como parâmetro o *id* do usuário;

- ***unfollow*:** desabilita um usuário do Twitter que estava sendo seguido;

As operações que retornam informações, que são os *getters*, retornam uma estrutura de informações em formato XML.

4.3.2.3 Serviço de Armazenamento e Recuperação de Mensagens

O serviço web wsmbox é um serviço que tem a finalidade de armazenamento de mensagens, ou seja, é uma caixa de mensagens de um usuário. As mensagens são tratadas na forma de uma fila FIFO (*First In Fist Out*). As mensagens podem ser filtradas por um *receiver*, que pode se referir a quem deve ser essa mensagem ou a um tópico específico. A seguir é apresentado o diagrama entidade-relacionamento das informações do wsmbox.

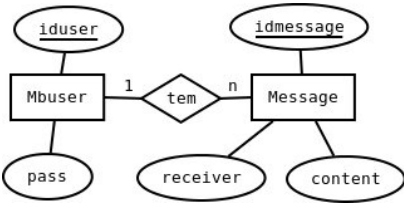

Figura 37: Diagrama ER do wsmbox.

A base de dados do wsmbox é composta de duas relações, a *Mbuser* e a *Message*. A relação *Mbuser* (Tabela 5) contem informações sobre o usuário do wsmbox. Dessa forma, o sistema pode possuir diversos usuários, e cada qual com sua própria caixa de mensagem.

Mbuser		
Atributo	**Tipo**	**Descrição**
* *iduser*	*string*	Login de usuário
pass	*string*	Senha do usuário

Tabela 5: ER wsAudit - Relação *Mbuser*.

A relação *Message* (Tabela 6) é destinada a armazenar a mensagem propriamente dita.

Message		
Atributo	**Tipo**	**Descrição**
iduser	*string*	Chave estrangeira para tabela Mbuser
* *idmessage*	*int*	Identificador único da mensagem, auto incrementável
receiver	*string*	Nome de quem deve receber a mensagem
content	*text*	Conteúdo da mensagem.

Tabela 6: ER wsAudit - Relação *Message*.

Este serviço web é composto pelas seguintes operações e devidos parâmetros de entrada e retorno:

- ***createUser***: cria um usuário para utilizar o serviço de caixa de mensagens.
  - ***User*:** login de identificação do usuário.
  - ***Pass*:** senha para autenticação do usuário.
  - **Retorno:** retorna 1 se usuário foi criado e 0 se não foi possível.
- ***put*:** armazena uma mensagem na caixa de mensagens do usuário.
  - ***User*:** login de identificação do usuário.
  - ***Pass*:** senha do usuário.
  - ***Receiver*:** a quem se destina a mensagem.
  - ***Content:*** conteúdo da mensagem.
  - **Retorno:** 0 se não foi possível e 1 se realizado com sucesso.
- ***has*:** verifica se há alguma mensagem para um *receiver* específico.
  - ***User*:** usuário.
  - ***Pass*:** senha.
  - ***Receiver*:** destinatário da mensagem.
  - **Retorno:** retorna 1 se há mensagem para *receiver*, senão retorna 0.
- ***get*:** retorna uma mensagem da caixa de mensagens de um usuário. Quando essa operação é chamada, a mensagem é retornada e retirada da caixa de mensagens. A ordem das mensagens é ao estilo *FIFO* (*First in, first out*).
  - ***User*:** usuário.
  - ***Pass*:** senha.

- ***Receiver*:** destinatário.
- **Retorno:** retorna o conteúdo da mensagem. Se não houver mensagem para retornar, então retorna 0.

No caso da sua utilização do wsmbox pelo GAP, as mensagens podem estar atrelada aos comportamentos via *Receiver*. Considerando isso aplicado ao estudo de caso apresentado, quando uma mensagem é recebida do ISAP, e se ela é direcionada para um comportamento específico, então ela é armazenada no wsmbox com o nome do comportamento como *receiver*.

4.3.2.4 Serviço web para Transações com Cartão de Crédito

O serviço web para transações com Cartão de Crédito é um serviço que simula transações para pagamento via cartão de crédito. Esse serviço está em conformidade com os serviços web SOA da OASIS (MacKenzie et al., 2006). Nesse estudo de caso e protótipo não foi desenvolvida qualquer interface humana para configuração, tal qual um software ou um interface web de configuração. Apenas as operações do serviço web é utilizada para completar as transações do estudo de caso. Na Figura 38 é apresentado o modelo entidade-relacionamento da base de dados do sistema.

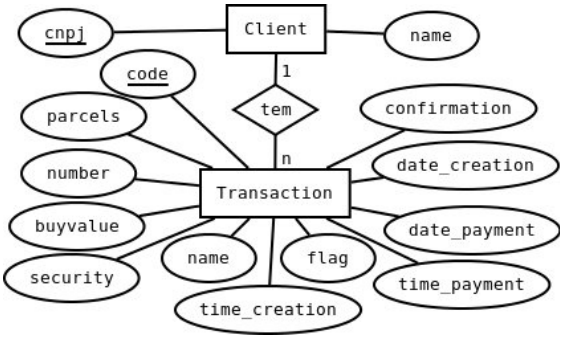

Figura 38: Diagrama ER do sistema de Cartão de Crédito.

A relação *Client* possui apenas dois atributos, o CNPJ da empresa cadastrada no sistema e o nome da empresa. Para cada transação que a empresa cliente inicia, é instanciada uma nova tupla na relação *Transaction*, descrita Tabela 7. Inicialmente, o cliente do serviço de

cartão de crédito deve criar uma nova transação, informando o valor e recebendo um código de transação. Depois disso o comprador entra em contato com o serviço de cartão de crédito, enviando as informações do cartão de crédito, o valor da compra e a quantidade de parcelas desejada. Após confirmado o pagamento (isso não é feito nesse protótipo, considera-se o pagamento válido), o atributo *confirmation* é habilitado como *true*. O cliente do serviço de cartão de crédito (no caso o Fornecedor) entra em contato com o serviço para verificar se o pagamento foi efetivado para, enfim, fechar a transação.

Transaction		
Atributo	**Tipo**	**Descrição**
* *code*	*string*	Código único da transação
cnpj	*string*	CNPJ da empresa cliente
date_creation	*date*	Dia da criação da transação
time_creation	*time*	Horário da criação da transação
date_payment	*date*	Dia do pagamento efetuado pelo comprador
time_payment	*time*	Horário do pagamento efetuado pelo comprador
confirmation	*boolean*	Confirmação do pagamento
flag	*string*	Bandeira do da empresa (Visa, Credcard, etc.)
number	*string*	Número do cartão de crédito
name	*string*	Nome do comprador, tal como escrito no cartão de crédito
security	*string*	Código de segurança, escrito atrás do cartão
buyvalue	*real*	Valor da compra
parcels	*int*	Quantidade de parcelas

Tabela 7: ER wsCC - Relação *Transaction*.

As operações do serviço web wsCC são as seguintes:

- ***create*:** Cria uma nova transação. É enviado como parâmetro o CNPJ do cliente do serviço (no caso o fornecedor), o valor e é retornado o id da transação;
- ***confirm*:** Retorna se o pagamento de uma transação foi efetuado, confirmando seu fechamento. É enviado como parâmetro o id da transação;

- ***cancel*:** É informado ao sistema de cartão de crédito que uma dada transação foi cancelada. É enviado como parâmetro o id da transação;
- ***pay*:** O comprador envia as informações necessárias para efetuar o pagamento de determinada transação. As informações enviadas são o id da transação, as informações do cartão (bandeira, número, nome do cliente e código de segurança) e as informações da compra (valor e parcelas).

Este sistema é apenas um protótipo para efetivar os testes feito em cima dos comportamentos desse estudo de caso, não sendo efetuada nenhuma operação para segurança das informações.

4.3.2.5 Serviço Acesso à Servidores de E-mail

O serviço de interoperabilidade wsMail é uma interface para que uma conta de e-mail do usuário possa ser acessada via uma interface SOA. Os métodos disponíveis, com seus respectivos argumentos são apresentados a seguir:

- ***countMessages*:** Verifica quantas novas mensagens o usuário possui;
  - ***host*:** servidor de e-mails do usuário;
  - ***user*:** e-mail do usuário;
  - ***pass*:** senha do usuário;
- ***getMail*:** retorna o próximo novo e-mail do usuário. Os parâmetros passados são os mesmos do *countMessages*;
- ***sendMail*:** envia um e-mail;
  - ***host*:** servidor de e-mails do usuário;
  - ***port*:** porta utilizada pelo servidor de e-mails;
  - ***user*:** e-mail do usuário;
  - ***pass*:** senha do usuário;
  - ***userName*:** Nome do usuário;
  - ***to*:** e-mail do contato para qual deve ser enviado o e-mail;
  - ***subject*:** Título da mensagem de e-mail;
  - ***body*:** corpo do e-mail;

O serviço web wsMail se utiliza do protocolo IMAP (*Internet Message Access Protocol*) para buscar mensagens e do protocolo SMTP para enviar.

4.3.2.6 Serviço *Chatbot*

O serviço web wsChatter é uma implementação simples de um *chatbot*. A finalidade desse serviço é dar um retorno à mensagens enviadas a ele. Sua funcionalidade para SAPs, particularmente, é deixar a interação com o usuário mais humanizada. O usuário deve ver seu assistente pessoal como se fosse uma pessoa real que está ali, conectada na internet para auxiliá-lo. É natural que haja uma interação na forma de conversa. Também, como o assistente pessoal aparece online no Gtalk, outras pessoas também podem conversar com ele, inclusive mandar avisos para o usuário do assistente pessoal via conversa com o assistente. As operações de interface com esse serviço web, e seus devidos parâmetros, são descritos a seguir:

- ***createUser*:** cria um usuário no sistema;
  - ***login*:** login do usuário a ser criado;
  - ***pass*:** senha do usuário para autenticação;
  - ***name*:** Nome *chatbot* do novo usuário;
  - ***owner*:** E-mail do usuário do *chatbot*;
  - ***ownerName*:** Nome do usuário do *chatbot*;
  - **Retorno:** Retorna 1 se a operação foi realizada com sucesso, senão retorna 0;
- ***updatePassword*:** Altera a senha do usuário de acesso ao sistema. São passados como parâmetros o *login* do usuário, a senha e a nova senha;
- ***updateBotName*:** Altera o nome do *chatbot*. São parâmetros o *login* do usuário, a senha e o novo nome do *chatbot*;
- ***connect*:** Efetua uma conexão ao sistema, enviando o *login* e senha e retornando 1 se conectado, "PASSWRONG" se a senha esta incorreta ou "NOUSER" se o usuário não existe no sistema;
- ***send*:** Envia uma mensagem ao *chatbot* do usuário. Os parâmetros são os seguintes:
  - ***login*:** identificador do usuário do *chatbot*;
  - ***pass*:** senha para autenticação do usuário;
  - ***friend*:** identificador da pessoa que enviou a mensagem;
  - ***friendsName*:** nome da pessoa que enviou a mensagem;
  - ***message*:** mensagem enviada;
  - ***Retorno*:** retorna uma mensagem de resposta do *chatbot*;

As mensagens de retorno são configuradas pelo próprio usuário na interface do sistema de *chatbot*, descrita na próxima seção. Essas são frases já elaboradas pelo usuário, e são selecionadas aleatoriamente a partir de um conjunto de frases que fecham com a mensagem de envio. Existem dois tipos de respostas:

- ***Threads*:** respostas encontradas conforme identificação da mensagem que a pessoa mandou para o *chatbot*.
- ***Scapes*:** quando não são encontradas respostas que fecham com as informações de entrada, então é selecionada uma resposta de escapatória, pois não foi possível responder de forma conveniente.

Conforme o que foi descrito até este ponto sobre o sistema de *chatbot*, pode-se ter então a visão uma estrutura, ou modelo entidade-relacionamento do sistema (Figura 39).

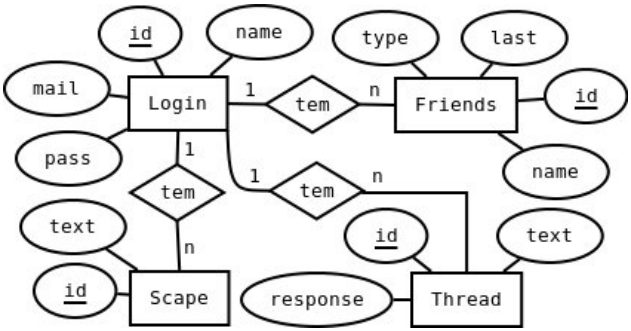

Figura 39: Diagrama ER do sistema de *chatbot*.

Um usuário pode possuir vários contatos, várias mensagens *scape* e várias *threads* de conversa distribuídas em quatro relações. A relação com informações do usuário do *chatbot* é descrita na Tabela 8:

Login		
Atributo	**Tipo**	**Descrição**
* *id*	*string*	Login de usuário
pass	*string*	Senha do usuário
name	*string*	Nome do usuário
mail	*string*	E-mail de contato

Tabela 8: ER wsChatter - Relação *Login*.

A relação referente aos contatos do usuário serve para armazenar informações dos contatos que já conversaram com o *chatbot*, servindo também como fonte de informações para a geração dos *logs* das conversas. Essa relação é a apresentada na Tabela 9:

Friends		
Atributo	**Tipo**	**Descrição**
** id*	*string*	Identificador único do contato
login	*string*	Chave estrangeira que referencia a relação *Login*
name	*string*	Nome do contato
type	*int*	Tipo de contato (0:mestre ou 1:amigo). O mestre pode programar o assistente pessoal com comandos especiais na mensagem. Isso não foi mais muito utilizado depois da criação de uma interface web para configurar o *chatbot*.
last	*datetime*	Dia e horário da última conversa com o contato

Tabela 9: ER wsChatter - Relação *Login*.

Quando chega uma mensagem de uma pessoa no *chatbot*, é avaliado se essa pessoa já se encontra na lista de contatos, por meio do atributo *id*. Se o contato não existe, então ele é acrescentado na lista de contatos. Toda conversa que acontece entre o contato e o *chatbot* é gravada em um arquivo de *logs* em um subdiretório chamado *log* e com o nome de arquivo *<id>.log*. Tanto a mensagem do contato, quanto a resposta é armazenada. Por exemplo:

(22/02/2011 14:16:39) saulopz@gmail.com: testando
(22/02/2011 14:16:39) arisa: Como assim? Que teste?
(22/02/2011 15:04:30) saulopz@gmail.com: $to compra-altera confirmar ordem 32
(22/02/2011 15:04:30) arisa: Ok! vou ver isso. Quando eu tiver alguma posição te informo.

As relações *Scape* e *Thread*, resumidamente, possuem um *id* autoincrementável inteiro para identificar a atividade. Especificamente, a relação *Scape* possui um campo texto que é a mensagem de resposta, caso não haja uma resposta que feche da relação *Thread*. A relação *Thread* possui dois atributos, além do *id*. O atributo *text* é um texto customizado que deve ser procurado na mensagem que é enviada ao *chatbot*. Por exemplo, considere que uma pessoa mandou uma mensagem para o *chatbot* com a seguinte frase:

"Olá, quem é você?"

O sistema então deve procurar por frases de *text* que fecham com essa frase, seleciona uma entre as que foram encontradas e envia como resposta. Neste caso, é encontrada uma frase em *text* que fecha:

"% quem e voce% %"

E em *response* a está declarada a seguinte resposta, e que será enviada como retorno:

> "Meu nome é Arisa. Sou um protótipo de um Assistente Pessoal baseado na Tese de Doutorado de Saulo Popov Zambiasi, do curso de Pós-Graduação em Engenharia de Automação e Sistemas da UFSC."

No sistema há um módulo que avalia o caractere "%" na forma de um coringa[25] da mensagem. Se há um % no início, significa que qualquer coisa pode vir antes, se for um coringa no final, qualquer coisa pode vir depois. Quando há um coringa entre duas palavras, então pode vir qualquer frase entre essas palavras. Assim também funciona quando o % está grudado na palavra. Neste caso, este é um coringa da palavra e não da frase. Nesta versão, o atributo *text* não deve ser acentuado. Também, no caso do voce% com o % após a palavra, significa que pode vir um ponto de interrogação junto a você, ou um ponto, ou uma vírgula, etc.

Se houverem mais tuplas da relação com *text* igual, mas *response* diferentes, então todas essas tuplas serão selecionadas e uma é será escolhida aleatoriamente. Dessa forma, pode-se enriquecer mais o vocabulário do *chatbot*.

A interface para configuração do *chatbot* é separada em três partes: (i) Amigos, para configurar os amigos, alterar nomes e visualizar os *logs*; (ii) Saídas, para configurar as frases de escapatória e (iii) Conversas (Figura 40), para configurar as respostas à frases reconhecidas.

Na interface web para configuração do *chatbot*, no *menu* Saídas, pode se acrescentar, editar ou excluir um texto. O formulário para edição possui apenas um campo, que é o texto da qual deve ser inserido ou editado. De forma diferente, no *menu* Conversas, deve ser configurado o Texto formatado com os coringas e a resposta a esse texto

[25]Um caractere coringa é um elemento utilizado para substituir algum caractere desconhecido por uma sequência de caracteres, por exemplo. Neste caso especifico, o coringa é utilizado tal como o % é utilizado no LIKE em cláusulas SQL (*Structured Query Language*).

(Figura 40). As conversas também podem ser acrescentadas, editadas ou excluídas.

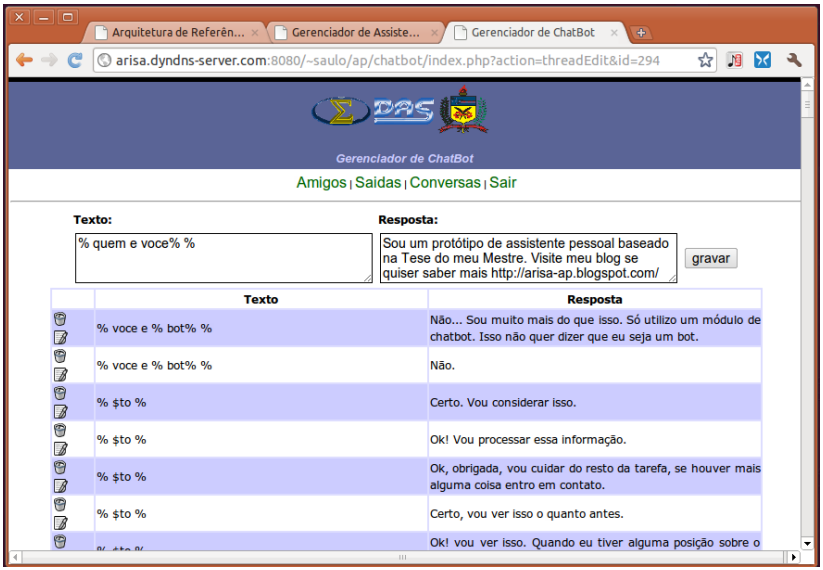

Figura 40: Interface web para configuração do chatbot - Menu Conversas.

É interessante que existam mais respostas para um mesmo texto, fazendo com que o *chatbot* do usuário tenha um vocabulário mais rico em relação à possíveis respostas.

4.3.2.7 Serviço de Assistência e Gerência de E-mails

O *Web Service for Mail Assistence* (*wsMaila*) tem a finalidade de promover ao usuário uma forma de assistência aos seus e-mails quando eles chegam em sua caixa postal e o mesmo não se encontra conectado à internet para efetuar sua verificação. Filtros podem ser criados com regras para gerenciar atividades de respostas automáticas de e-mails e de envio de mensagens para o usuário em conformidade com situações determinadas por ele. Na Figura 41 é apresentado o diagrama entidade-relacionamento das informações utilizadas pelo sistema.

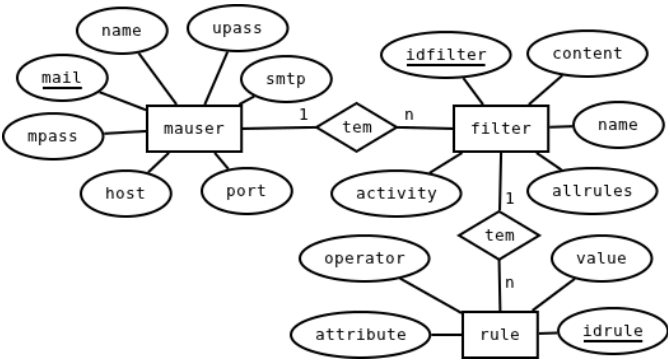

Figura 41: Diagrama ER do MailA.

Na Tabela 10 são apresentados os atributos e descrições da relação *mauser*, referente as informações do usuário do *Mail Assistence*.

mauser		
Atributo	**Tipo**	**Descrição**
** mail*	*string*	Email do usuário da qual deve ser gerenciado.
upass	*string*	Senha no sistema MailA
name	*string*	Nome do Usuário
host	*string*	Servidor de e-mails POP/IMAP do usuário
smtp	*string*	Servidor de envio de e-mails.
port	*string*	Porta do servidor SMTP
mpass	*string*	Senha da conta de e-mail do usuário.

Tabela 10: Relação mauser.

Os atributos dos filtros a serem criados pelo usuário são apresentados na Tabela 11.

filter		
Atributo	**Tipo**	**Descrição**
* *idfilter*	*integer*	Identificador único do filtro.
mail	*string*	Chave estrangeira para o e-mail do usuário.
name	*string*	Nome do Filtro.
allrules	*boolean*	Se true, todas as regras precisam ser validas, senão, apenas uma.
activity	*string*	Tipo da atividade (Responder ao remetente, mandar mensagem ao usuário).
content	*text*	Informações extras para serem utilizadas pelo tipo da atividade.

Tabela 11: Relação filter.

Cada filtro (Tabela 12) pode ser composto de diversas regras. Essas se referenciam aos campos referente aos e-mails de forma a efetuar testes de validação.

rule		
Atributo	**Tipo**	**Descrição**
* *idrule*	*string*	Identificador único da regra
idfilter	*string*	Chave estrangeira para filtro
attribute	*string*	Tipo do atributo a ser verificado (*From, To, Date, Subject, Content*)
operator	*string*	Tipo de operação de comparação (*==, !=, <, <=, >, >=, contem*).
value	*text*	A informação com a qual se quer comparar

Tabela 12: Relação rule.

Um exemplo de regra pode ser da seguinte forma. Se o título (*subject*) da mensagem contiver a tag “orientação” e for de um determinado e-mail específico, então retorna uma mensagem que pode ser enviada ao usuário.

4.3.2.8 Interface web para configuração do *Mail Assistence (MailA)*

A Interface para configuração do *MailA* (Figura 42) tem a propriedade de servir ao usuário como um meio de configurar uma conta de e-mail, os filtros, regras e atividades a serem efetuadas pelo sistema de assistência na leitura dos e-mails.

Figura 42: Página principal do MailA

Na Figura 43 é apresentado um exemplo de configuração de um filtro de e-mail.

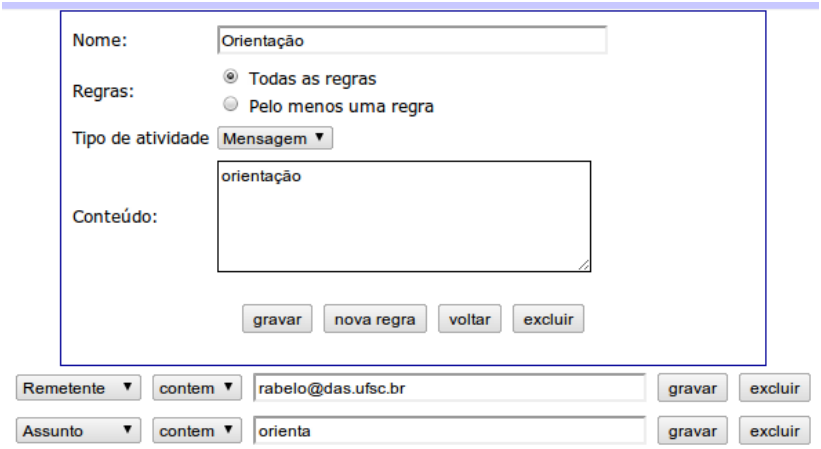

Figura 43: Configuração de Filtros e Regras no MailA.

No caso apresentado, no filtro de nome **Orientação** é definido que todas as regras devem ser satisfeitas para que a atividade **mensagem** possa ser efetuada. Essa deve enviar ao usuário uma mensagem montada com a informação contida em **conteúdo**. As regras criadas são: (i) caso o remetente contenha o e-mail ***rabelo@das.ufsc.br*** e o assunto contenha ***orienta***, então o filtro é satisfeito e a atividade de envio de mensagem de alerta ao usuário deve ser efetuada.

4.3.2.9 Serviço de Acesso a Servidor de *Instant Messaging*

Uma particularidade de um servidor de *Instant Messaging* (IM) é que o usuário precisa estar conectado ao servidor de forma síncrona e um serviço web funciona de forma assíncrona, i.e., o usuário requisita uma operação de um serviço web e então se desconecta. Dessa forma, para que um usuário possa acessar e utilizar um serviço de IM via serviço web, é necessário haver uma espécie de software cliente de IM/XMPP, no caso um software servidor, que mantém o usuário conectado de forma síncrona ao servidor. O usuário então se utiliza de chamadas a um serviço web para receber e enviar mensagens. Contudo, também é necessário que o usuário envie um *ALIVE* ao software servidor a cada período de tempo, caso contrário, se o usuário não estiver mais utilizando o IM, ele deve ser desconectado.

Assim, inicialmente é descrito o desenvolvimento de um servidor cliente de IM. Este software permanece em execução, aguardando que algum usuário se conecte. O usuário precisa ser um usuário de um sistema sob o protocolo XMPP, no caso o Gtalk. Para fazer o acesso ao Gtalk via linguagem de programação Java, é utilizada a API Smack[26]. A seguir, são apresentadas as principais classes do software (Figura 44) e é descrito brevemente seu funcionamento.

A classe **XMPPConnector** possui o método *run*, que permanece em um laço de execução. Neste laço, é gerenciada uma conexão via soquetes, aguardando alguma conexão. Se alguma conexão é feita e uma mensagem é entregue, então ela precisa ser avaliada para ser entregue ao ciente específico.

[26]Smack é uma biblioteca opensource cliente de mensagens instantâneas de protocolo Jabber, XMPP (SMACK, 2010).

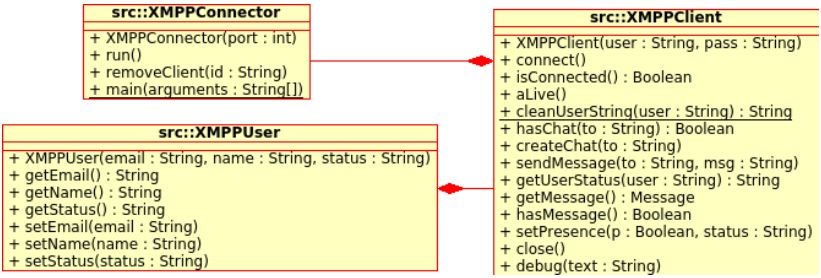

Figura 44: Diagrama de classes do XMPPConnector.

O formato dessas mensagens são uma estrutura específica com informações separadas por ":" e seguem a seguinte estrutura:

- **usuário:** e-mail do usuário a conectar no Gtalk. Todas as mensagens que chegam ao servidor são específicas ao cliente deste usuário.
- A partir da segunda informação, a estrutura é modificada. Cada tipo específico é agora mostrado:
  - ***connect*:** informa ao servidor que deve ser feita uma nova conexão. A informação seguinte deve ser a senha do usuário no Gtalk;
  - ***alive*:** avisa ao servidor que o usuário específico está ainda online;
  - ***sendMessage*:** uma nova mensagem deve ser entregue. As seguintes informações são:
    - ***to*:** para quem a mensagem se destina, ou seja, algum contato do usuário;
    - ***message*:** conteúdo da mensagem a enviar.
  - ***hasMessage*:** verifica se alguma mensagem foi enviada para o usuário;
  - ***getMessage*:** busca a última mensagem enviada para o usuário;
  - ***getUserStatus*:** retorna o estado de um contato específico;
  - ***setStatus*:** altera o próprio estado do usuário (*online, offline, away,* etc.);
  - ***close*:** fecha uma conexão.

Para cada novo usuário que se conecta, é criado uma *thread* de um cliente, **XMPPClient.** A cada certo período de tempo, cada cliente é avaliado para verificar se houve alguma atividade ou se recebeu alguma mensagem do usuário. A mensagem do tipo *alive* serve especificamente

para o usuário avisar o servidor que ainda está conectado. Caso não seja informado nada, depois de um tempo o servidor desconecta o usuário, acreditando que este não está mais utilizando o software.

A classe **XMPPClient** possui uma lista de contatos, e cada contato do usuário é armazenado na classe **XMPPUser.** Esta é regularmente atualizada de forma a saber se o usuário está online, se há uma conversa ativa com esse usuário, etc.

Quando um contato inicia uma conversa com o usuário, uma nova seção de conversa é aberta. A cada mensagem que o contato envia para o cliente, ela é colocada em uma fila de mensagens para envio ao usuário. Nela contém informações de que contato está enviando a mensagem e o conteúdo da mensagem. Esta pode ser recuperada pelo usuário com a mensagem *getMessage*.

Tendo o servidor funcionando, um serviço de interoperabilidade é desenvolvido, chamado wsXMPP, que se destina a oferecer uma interface SOA para acesso ao servidor **XMPPConnector.** A forma de comunicação entre o wsXMPP e XMPPConnector se dá via soquetes e uma estrutura de mensagens proprietária. Com esse serviço, o Assistente Pessoal, que pode apenas ver serviços web, pode agora se utilizar deste servidor, mesmo que indiretamente. Os métodos para acesso a esse serviço web são apresentados a seguir, com seus respectivos argumentos:

- ***connect*:** avisa ao servidor XMPPConnector que uma conexão está sendo requisitada.
  - ***id*:** e-mail do usuário para conexão ao Gtalk;
  - ***pass*:** senha do usuário para autenticação;
  - **Retorno:** 1 se bem sucedido, senão retorna 0;
- ***alive*:** informa ao servidor que o usuário ainda se encontra conectado/ativo. Como parâmetro envia apenas o *id*;
- ***sendMessage*:** envia uma mensagem para um contato;
  - ***id*:** e-mail do usuário;
  - ***to*:** e-mail do contato da qual a mensagem se destina;
  - ***msg*:** conteúdo da mensagem;
  - **Retorno:** 1 se bem sucedido, senão retorna 0;
- ***hasMessage*:** Retorna 1 se houver alguma mensagem para ser recebida, senão 0. Envia como parâmetro apenas o *id*;
- ***getMessage*:** Envia o *id* do usuário e retorna uma mensagem do XMPPConnector. As mensagens são buscadas na forma FIFO (*First In First Out)*;

- ***getUserStatus*:** Retorna o estado de algum contato do usuário;
  - ***id*:** e-mail do usuário;
  - ***user*:** e-mail do contato da qual quer se saber o estado;
  - **Retorno:** retorna o estado do usuário;
- ***close*:** fecha uma conexão;

Enquanto um usuário estiver ativo, enviando *alive* ou mensagens ao **XMPPConnector**, o usuário vai aparecer online para seus contatos no Gtalk.

4.3.3 Softwares e Serviços da Empresa

Por Softwares e Serviços da Empresa tem-se o conjunto de softwares produzidos para serem utilizados para o gerenciamento das informações e processos de uma determinada empresa, gerenciados pelos funcionários e com interfaces para serem acessados por sistemas externos na forma de serviços.

As implementações específicas para serem utilizadas pelo processo automático de compras, no caso aqui estudado, são referentes ao acesso ao sistema gerenciador de Estoque, do usuário, ao sistema gerenciador do fornecedor (do lado do fornecedor) e sistema de transações via cartão de crédito, no caso é apenas uma simulação baseada em operações reais.

4.3.3.1 Sistema de Controle de Estoque

O sistema de controle de estoque é, conforme o estudo de caso escolhido para implementação, um sistema legado da qual uma pessoa, que deve se utilizar de um Assistente Pessoal para auxiliá-lo, deve gerenciar. Sua função é manter o estoque de produtos (matéria-prima para a produção) sempre em dia, efetuando compras quando necessário para que a produção não pare.

Na Figura 45 é apresentado o diagrama entidade-relacionamento do sistema de controle de estoque.

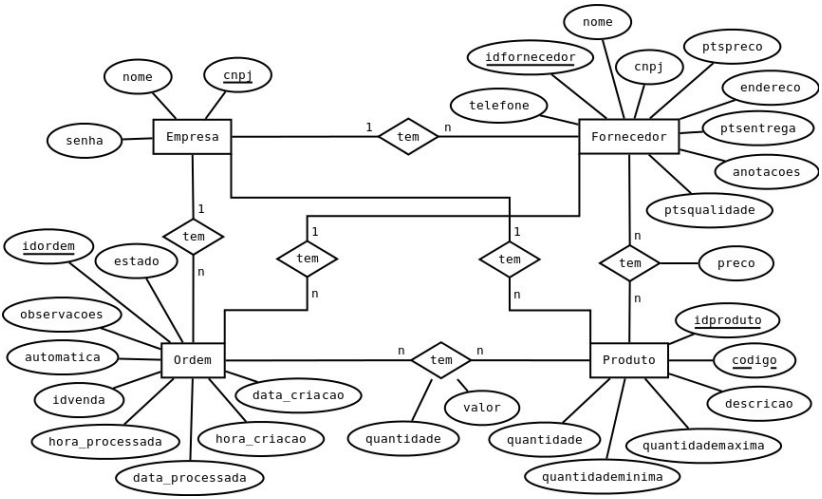

Figura 45: Diagrama ER do sistema de controle de estoque.

As informações do diagrama ER da Figura 45 não são apresentadas aqui na forma de relação, mas são descritos os elementos de maior importância. Observa-se que há uma relação chamada *Empresa*, com informações bastante resumidas (cnpj, nome e senha). Isso identifica o usuário de qual empresa está logando no sistema, uma vez que o sistema é multiusuário. Cada empresa pode ter sua lista de produtos e de fornecedores. Há também uma ligação entre produto e fornecedor, pois cada produto pode ter vários fornecedores, assim como também um fornecedor pode ter vários produtos, e cada fornecedor fornece o produto com um preço diferente.

Os atributos *quantidademinima* e *quantidademaxima* da relação Produto são utilizados na hora de selecionar quais produtos devem ser repostos no estoque e o quanto deve ser comprado. Se um produto possui uma quantidade menor de produtos do que *quantidademinima*, então o produto deve ser reposto de forma a chegar a uma quantidade equivalente à *quantidademaxima*.

Outro ponto a se observar na relação Produtos, é que há o *idproduto* e o *codigo*. O *idproduto* é um identificador único no banco de dados, chave primária, e o código do produto é um identificador escolhido pelo usuário e que é único apenas para cada empresa. Se a empresa A tiver um produto de código 001, a empresa B também pode ter, mas na mesma empresa não pode haver mais de um produto com o mesmo código. Este código serve para adequar a empresa a um padrão

de códigos de produtos na hora de criar e enviar uma ordem de compra para outra empresa.

A Figura 46 mostra uma tela do sistema de controle de estoque, mais especificamente a parte de edição das informações de um produto. Na parte mais acima, logo abaixo do *menu*, estão as informações dos produtos. Logo a baixo um espaço para ligar um novo fornecedor ao produto, a partir dos fornecedores cadastrados, e informando o preço que tal fornecedor vende o produto. Em seguida a lista de fornecedores do produto, com preços e telefones.

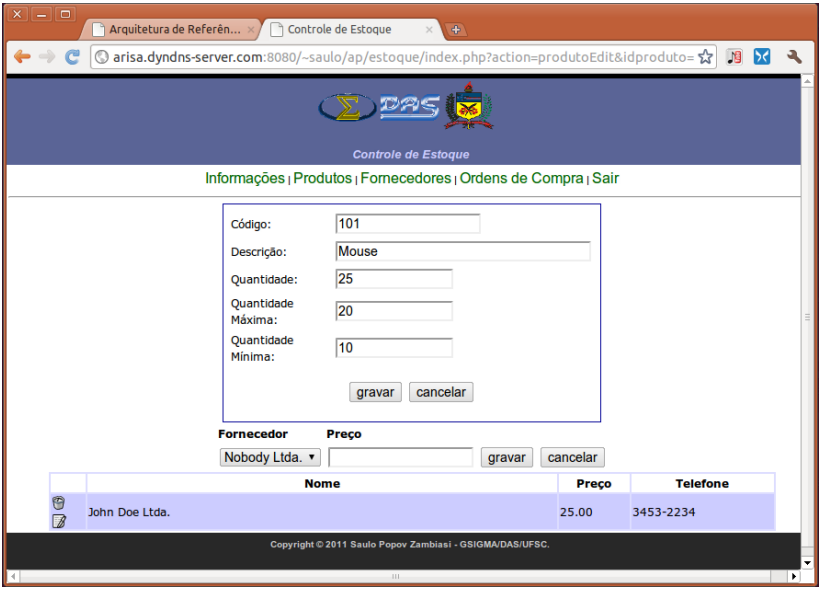

Figura 46: Interface web do sistema de controle de estoque.

As opções do *menu* são:

- **Informações:** para gerenciar os dados da empresa;
- **Produtos:** Lista os produtos em estoque, com opção de adicionar novo produto, excluir e editar um produto existente;
- **Fornecedores:** Listagem dos fornecedores, com opção de adicionar novo fornecedor, excluir e editar um existente;
- **Ordens de Compra:** Listagem das ordens de compra e estado atual. As ordens podem ser do tipo automática (gerenciada pelo assistente pessoal) ou não automática, que o assistente não deve gerenciar.

A listagem das ordens de compra é apresentada na Figura 47. Se o usuário quiser saber detalhes da ordem, deve selecionar **Visualizar** na linha da ordem.

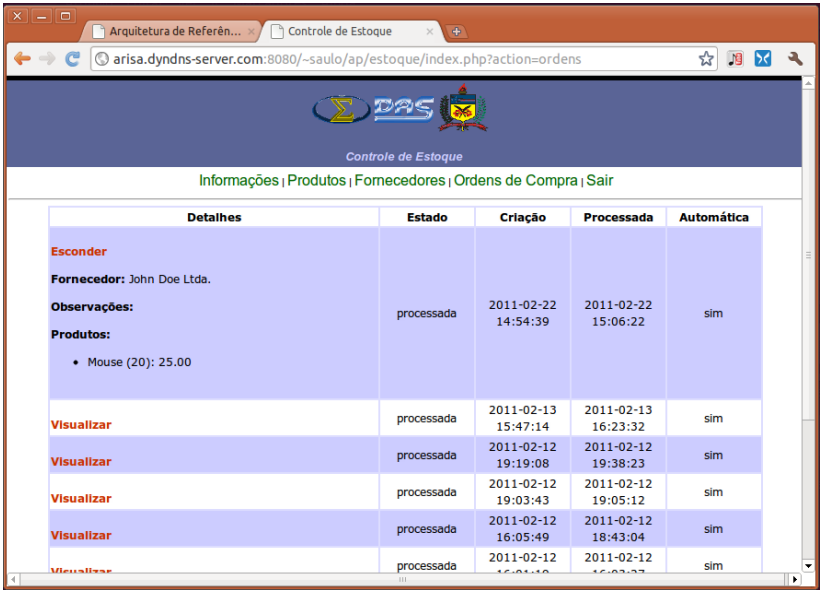

Figura 47: Interface visual do sistema de controle de estoque - Ordens de compra.

Observa-se os detalhes na última ordem processada (a primeira na lista da Figura 47). Pode-se ver os produtos da ordem, no caso apenas um, o estado, dia e horário de criação e de processamento e se a ordem é automática. Como o objetivo desse estudo de caso é fazer com que o assistente pessoal efetue todas as compras de forma automática ou semiautomática, como é mostrado mais a frente, nessa versão da implementação o usuário pode apenas visualizar as ordens, mas não pode editá-las. Logo, todas as ordens são do tipo automática.

A. Serviço web de Interoperabilidade wsEstoque

O serviço web de interoperabilidade wsEstoque, fornece um conjunto de operações para acessar informações referentes à situação do sistema de controle de estoque. Essa interface SOA torna o sistema compatível com a arquitetura, para ser utilizada como serviço web por

outros serviços ou pelo Assistente Pessoal. Suas operações são descritas a seguir com seus parâmetros, observando-se que todas as operações possuem como primeiro parâmetro o CNPJ da empresa e a senha de acesso:

- ***getProdutos*:** Sem outros parâmetros, retorna uma lista com os produtos e suas informações. A estrutura do resultado está na forma descrita pelo serviço web *myMessageTable* (seção 8.12);
- ***getFornecedores*:** Funciona tal como a operação anterior, mas retornando a lista de fornecedores cadastrados no sistema;
- ***getProduto*:** É passado como parâmetro subsequente o id do produto, retornando as informações do mesmo na mesma estrutura de tabela;
- ***getProdutoCodigo*:** É passado como parâmetro o id do produto, retornando o código correspondente;
- ***getProdutoValor*:** Retorna o valor de um produto passando como parâmetro o id do produto;
- ***getProdutoQuantidadeParaCompra*:** Passando o id do produto como parâmetro, retorna a quantidade de produtos que se quer comprar, ou seja, a operação *quantidademaxima* - *quantidade* (atributos encontrados no modelo ER da base de dados do sistema de controle de estoque na Figura 45);
- ***nextFornecedor*:** Essa operação serve para pegar o próximo fornecedor de uma lista de fornecedores de um produto. Isso é conveniente para o caso de necessitar procurar outro fornecedor caso o escolhido não possua estoque suficiente do produto para venda, ou não foi encontrado, etc. São passados como parâmetro o id do produto e o id do fornecedor atualmente selecionado;
- ***getFornecedorCNPJ*:** Retorna o CNPJ de um fornecedor por meio do id do fornecedor, que é passado por parâmetro;
- ***getFornecedor*:** Retorna as informações do fornecedor no formato descrito em *myMessageTable*, passando-se como parâmetro o id do fornecedor;
- ***getCodigosProdutosEstoqueBaixo*:** Retorna uma lista dos produtos com o estoque que se encontram com valor menor que o atributo *quantidademinima*;
- ***nextProdutoEstoqueBaixo*:** Retorna o próximo produto de uma lista de produtos com estoque baixo no sistema de controle de estoque. Passa-se como parâmetro o id do último produto verificado;

- ***getFornecedoresDeProduto*:** Passando como parâmetro o id de um produto, a operação retorna uma lista de fornecedores e suas informações, também no formato descrito em *myMessageTable*.

É importante observar que o sistema é multiusuário e diversas empresas podem possuir seus próprios dados. O identificador CNPJ da empresa é a chave primária de identificação única. A estrutura de mensagem descrita em *myMessageTable* foi utilizada por já ter sido desenvolvida pelo autor do trabalho em outra ocasião e se mostrou útil no momento. Inicialmente ela foi utilizada para esse serviço web, um dos primeiros desenvolvidos, e posteriormente começou-se a utilizar XML. O serviço web *myMessageTable* é utilizado por outros serviços web ou pelo GAP para separar devidamente as informações dessa estrutura. Em verdade, não houve a intenção, nem a necessidade, de se modificar esse módulo nessa fase de desenvolvimento para adequá-lo à nova forma de estrutura na forma XML, pois é de se considerar que cada sistema pode ter sua maneira de manipular as informações, não invalidando o fato de que trabalha-se aqui com serviços web SOA da OASIS (MacKenzie et al., 2006).

B. Serviço web de Interoperabilidade wsOrdem

O serviço de interoperabilidade wsOrdem é um serviço que serve de complemento ao sistema de gerenciamento de estoque e serve para gerenciar as ordens de compra. Todas as operações possuem como primeiro e segundo parâmetros o CNPJ da empresa e a senha de acesso. Suas operações são descritas a seguir:

- ***criaOrdem*:** Cria uma nova ordem de compra, são parâmetro o CNPJ do fornecedor da qual a ordem se destina;
- ***muda*:** Altera o fornecedor selecionado de um produto;
- ***getProdutoValor*:** Retorna o valor de um produto em uma ordem. Os parâmetros são o *id* da ordem e o *id* do produto;
- ***getFornecedorCNPJ*:** Retorna o CNPJ do fornecedor selecionado de uma ordem de compra. Passa o *id* da ordem como parâmetro;
- ***situacaoOrdem*:** Passando-se como parâmetro o *id* da ordem, é retornada a situação da ordem na forma de um texto;
- ***proximoProduto*:** Retorna o id do próximo produto em uma ordem. É passado como parâmetro o id da ordem e o id do

último produto visto. Se o id do produto passado for 0, então pega o primeiro;

- ***setIdVenda*:** Informa o id da venda para uma ordem. Este id é utilizado para identificar a compra para o sistema de Cartão de Crédito e para identificar a ordem de venda no fornecedor;
- ***próxima*:** Retorna a próxima ordem ainda ativa, passando como o id da última ordem visualizada;
- ***cancela*:** Cancela uma ordem de compra por meio do *id* da ordem como parâmetro;
- ***setEstado*:** Altera o estado de uma ordem, passando como parâmetro o id da ordem e o novo estado (aberta, rejeitada, modificada, aceita, cancelada ou processada);
- ***processa*:** Processa uma ordem, atualizando-as informações dela e, no caso específico dessa implementação, altera também a quantidade de produtos no estoque;
- ***getEstado*:** Passando como parâmetro o id de uma ordem de compra, retorna o estado atual da ordem;
- ***getIdVenda*:** Passa o id da ordem de compra e retorna o id relacionado com a ordem de venda no fornecedor;
- ***getIdOrdem*:** Passando como parâmetro o id de uma ordem de venda, busca o código da ordem de compra que está relacionado com esse id;
- ***alteraOrdem*:** Altera as informações de uma ordem de compra, passando como parâmetro o id da ordem e um XML com as informações atualizadas dos produtos.

A ordem está descrita de forma simplificada em XML e está convenientemente enquadrada no estudo de caso tratado nesse trabalho. Um exemplo de ordem é apresentada no código XML no Quadro 1:

```
<ordem>
      <empresa>01.000.000/000-01</empresa>
      <observacoes>none</observacoes>
      <produto>
            <codigo>001</codigo>
            <quantidade>20</quantidade>
            <valor>25.00</valor>
      </produto>
</ordem>
```

Quadro 1: XML de ordem de compra/venda.

A estrutura permite que vários produtos sejam adicionados. No exemplo acima, apenas um produto se encontra na ordem. Outros poderiam ser adicionados.

4.3.3.2 Sistema de Controle de Estoque do Fornecedor

O sistema de controle do estoque do fornecedor é bastante parecido com o sistema de controle de estoque. Contudo, existem algumas diferenças. O sistema de fornecedor é um protótipo e serve apenas para que as informações sejam convenientemente visualizadas e alteradas, se for o caso. Não há a necessidade de haverem fornecedores de produtos para o sistema. O modelo entidade-relacionamento da base de dados para o sistema do fornecedor é apresentado na Figura 48.

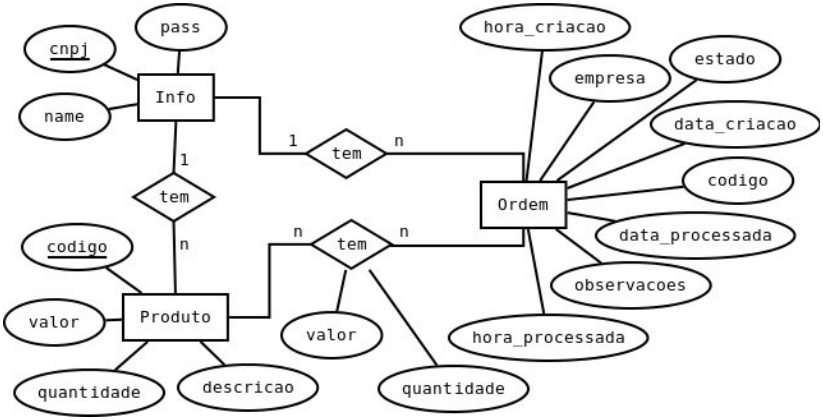

Figura 48: Diagrama ER do sistema de fornecedores.

No modelo é apresentada uma relação chamada *Info*, que corresponde a empresa fornecedora. O sistema também é multiusuário e permite que várias empresas fornecedoras sejam cadastradas. Há também a relação *Produto*, que possui as informações de preço e quantidade disponível, por exemplo, de cada produto. Também é necessária a relação *Ordem*, de forma a conter informações sobre as ordens que estão sendo processadas. Quando uma ordem é processada, os produtos que estão na ordem são atualizados no estoque, decrementando-os conforme a quantidade que está na ordem.

A. Servidor para processo automático de Venda

Este sistema efetua um controle automático das ordens de venda. Este faz acesso ao banco de dados do sistema de fornecedores, buscando as informações das ordens que estão ativas, ou seja, que ainda não foram processadas e que não estão no estado rejeitada ou cancelada. Um diagrama de classes desse sistema é apresentado na Figura 49.

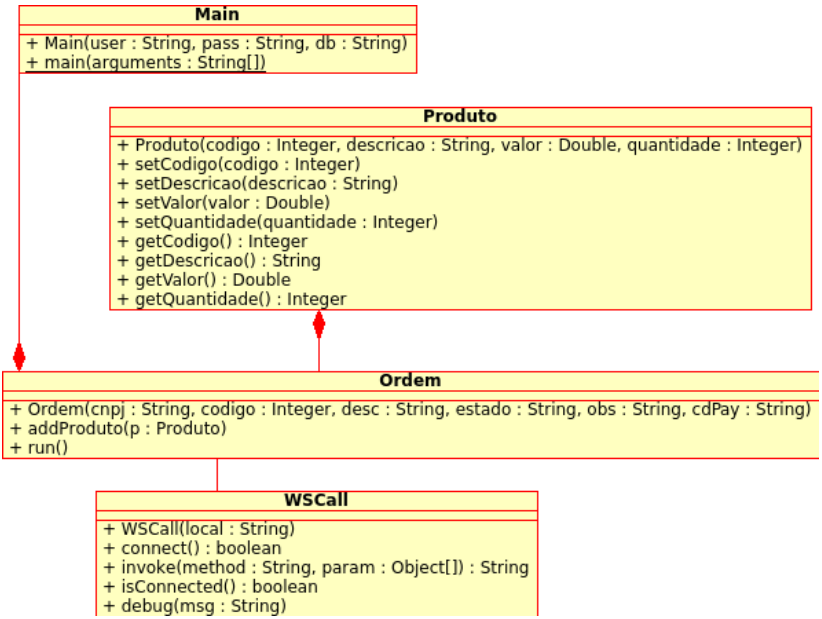

Figura 49: Diagrama de classes do processo automático de venda.

O sistema permanece em execução, e a classe *Main* avalia periodicamente as ordens de venda que estão ativas. Quando algo é alterado, o sistema avalia a atua sobre as alterações.

Quando é encontrada uma ordem de venda ativa, é criada uma instância da classe *Ordem* e as informações da ordem são carregadas nela, inclusive os produtos contidos na ordem. A execução da avaliação da ordem se dá da seguinte forma:

- Se o estado da ordem é **aberta**, então
  - Verifica se os preços batem, ou se há produtos suficientes no estoque para serem vendidos

        - Se algo não estiver ok, então altera a ordem e a coloca como **modificada**
        - Senão a coloca como **aceita**
- Se o estado da ordem é **aceita**, então
    - Se ainda não existe um código de pagamento, então
        - Contacta o sistema de Cartão de Crédito via serviço web e cria uma nova transação
    - Senão, se já foi pago, então
        - Atualiza estoque e seta ordem como **processada**
    - Senão, se foi cancelado o pagamento, então
        - Atualiza ordem como **cancelada**

O sistema permanece em execução, chamando esse algoritmo sempre que uma ordem estiver ativa.

B. Serviço web de Interoperabilidade wsVenda

O serviço web de interoperabilidade wsVendas é uma interface compatível com a especificação SOA da OASIS (MacKenzie et al., 2006) para efetuar a venda de produtos do sistema protótipo de controle de fornecedor. Suas operações são descritas a seguir:

- ***codigoDePagamento*:** Envia como parâmetro o id da ordem de compra e retorna o código de pagamento a ser efetuado no sistema de transações via cartão de crédito. Se não há código de pagamento, então retorna 0;
- ***alteraOrdem*:** Altera uma ordem de pagamento. Envia como parâmetros o id da ordem e um XML com as informações da ordem, no formato apresentado na seção A;
- ***novaOrdemXML*:** Cria uma nova ordem de venda, passando como parâmetro um XML no formato também da seção A, já com as informações do cliente, do fornecedor e dos produtos;
- ***novaOrdem*:** Cria uma nova ordem de venda. Os parâmetros enviados são o CNPJ do fornecedor da qual se deseja efetuar o negócio, observações (opcional) e o CNPJ do cliente que intenta efetuar a compra. A operação retorna o id da ordem criada;

- ***setProduto*:** Insere um novo produto em uma ordem de venda. Os parâmetros enviados são o id da ordem de venda, o código, a quantidade e o preço do produto;
- ***cancelaOrdem*:** Envia um pedido para que o sistema cancele uma ordem, conforme o id passado por parâmetro;
- ***confirmaModificar*:** Avisa o sistema de ordens que o cliente confirmou a modificação de uma ordem feita pelo sistema de venda automatizado. É passado como parâmetro da operação apenas o id da ordem;
- ***estadoOrdem*:** Retorna o estado de uma ordem de venda, passando como parâmetro o id da ordem;
- ***buscaOrdem*:** Passando-se como parâmetro o id da ordem, retorna as informações da ordem em um XML, conforme especificado na seção A;
- ***listaDeProdutos*:** Retorna em formato XML a lista dos produtos de determinado fornecedor com suas informações de código, descrição, preço e quantidade em estoque. É passado como parâmetro o CNPJ do fornecedor da qual se quer pesquisar;
- ***preco*:** Retorna o preço de um produto específico de um fornecedor através do seu código. Os parâmetros passados são o CNPJ do fornecedor e o código do produto;
- ***valorOrdem*:** Retorna o valor total de uma ordem, ou seja, o somatório dos preços dos produtos multiplicados pelas quantidades definidas na ordem de compra. O parâmetro passado é o id da ordem de venda.

O XML retornado na operação listaDeProdutos é no mesmo formato apresentado na seção A, mas sem as informações da ordem.

4.3.4 Softwares Assistentes Pessoais

O Sistema que efetua o gerenciamento do assistente pessoal (GAP) é separado em duas partes, a primeira parte é o software servidor que mantêm os assistentes pessoais funcionando, e a segunda parte é a interface para configuração do assistente pessoal do usuário. O sistema é multiusuário e permite que vários assistentes pessoais possam estar em execução ao mesmo tempo. Em tempo, também podem ser implementados serviços que podem ser utilizados como funcionalidades

auxiliares na construção dos comportamentos. Alguns desses serviços são apresentados no APÊNDICE A.

Na Figura 50 é apresentado o diagrama entidade-relacionamento da base de dados dos assistentes pessoais.

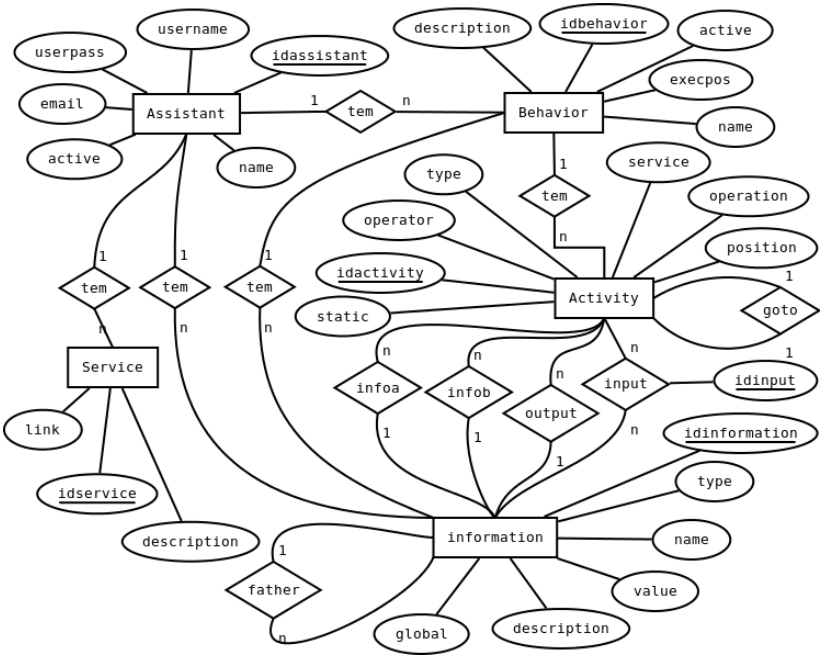

Figura 50: Diagrama ER do GAP.

A entidade *Service* é utilizada para que o usuário possa cadastrar os serviços que ele utiliza normalmente em seu assistente pessoal. A vantagem em mantê-los cadastrado é que o sistema busca o WSDL do serviço web e apresenta para o usuário as operações do serviço web e seus parâmetros. Essas informações são úteis no momento de montar um comportamento do assistente. A relação é composta apenas de um identificador inteiro auto incrementável, de uma chave estrangeira referenciando a relação *Assistant*, de uma descrição textual do serviço web e do link do serviço.

Cada assistente pessoal cadastrado se torna uma instância de uma tupla da relação *Assistant* (Tabela 13).

Assistant		
Atributo	**Tipo**	**Descrição**
* *idassistant*	*string*	Identificador do assistente – login
name	*string*	Nome do assistente pessoal
username	*string*	Nome do usuário
userpass	*string*	Senha para autenticação
email	*string*	E-mail do usuário
active	*boolean*	*True* se o assistente está em funcionamento

Tabela 13: ER GAP - Relação *Assistant*.

O assistente pessoal possui uma estrutura de informações na forma de árvore, armazenadas na relação *Information* (Tabela 14). Uma informação pode ter vários filhos e cada filho pode ter outros *n* filhos. Assim, existe um atributo chamado *father*, em que é definida a informação pai da informação. Caso *father* seja igual a *null*, então ela é uma informação raiz.

Information		
Atributo	**Tipo**	**Descrição**
* *idinformation*	*string*	Identificador único da informação, auto incrementável
idassistant	*string*	Chave estrangeira referenciando a relação *Assistant*
idbehavior	*integer*	Chave estrangeira referenciando a relação *Behavior*
father	*integer*	Chave estrangeira referenciando a própria relação.
global	*boolean*	*True* se for uma informação for global, *false* para local.
description	*string*	Descrição da informação
name	*string*	Nome da informação
value	*string*	Valor da Informação
type	*string*	Tipo da informação

Tabela 14: ER GAP - Relação *Information*.

Uma informação pode ser de tipos diferentes, definidos no atributo *type*. Os tipos podem ser básicos como *string, integer, float, boolean, text*, ou um tipo complexo chamado *struct*. Se o tipo for *struct,* então é caracterizada como uma informação pai e ela pode conter *n* informações filhas.

O atributo *global* identifica se uma informação pode ser visualizada pelos outros comportamentos do assistente pessoal. Se ela for valor *false*, então ela só pode ser visualizada no comportamento da qual a informação pertence, identificado em *idbehavior*.

Cada assistente pessoal pode possuir diversos comportamentos (Tabela 15). O comportamento pode ou não estar funcionando, e deve ser informado o estado deste no atributo *active*.

Behavior		
Atributo	**Tipo**	**Descrição**
** idbehavior*	*integer*	Identificador único da informação, auto incrementável
idassitant	*string*	Chave estrangeira referenciando a relação *Assistant*
description	*text*	Um texto para descrever o comportamento
name	*string*	Nome do comportamento
execpos	*integer*	Posição em que se encontra a execução do comportamento
active	*boolean*	*True* se o comportamento está ativo, funcionando.

Tabela 15: ER GAP - Relação *Behavior*.

Pela maneira como foi implementado o GAP dessa instância de implementação, a forma como o comportamento é executado é seguindo uma estrutura algorítmica, a ser mostrada no próximo subcapítulo.

Cada linha de execução é uma atividade e está colocada em sua devida ordem. O atributo *execpos* se referencia à atividade em que a execução do comportamento está, ou linha de execução do algoritmo. Se o usuário requisitar que o comportamento pare seu funcionamento, a posição de execução é salva, para que o comportamento possa voltar a se executar a partir da posição que parou.

Os comportamentos são executados paralelamente. A cada ciclo de execução do Assistente pessoal, uma linha de execução de cada comportamento é executada.

As linhas de execução do comportamento são descritas no GAP como atividades, armazenadas no banco de dados na relação *Activity* (Tabela 16).

Activity		
Atributo	**Tipo**	**Descrição**
* *idactivity*	*integer*	Identificador único da informação, auto incrementável
idbehavior	*integer*	Chave estrangeira referenciando o comportamento
position	*integer*	Posição da atividade no algoritmo do comportamento
type	*string*	Tipo de atividade
operator	*string*	Indica o tipo de operação utilizado pelo comportamento
service	*string*	Se chamada a serviço, indica o link do serviço web
operation	*string*	Operação a ser chamada no serviço web
goto	*integer*	Se é um Bloco, referencia o início ou o fim do bloco.
static	*string*	Armazena o valor de uma informação, caso estática
infoa	*integer*	Referencia uma *Information*
infob	*integer*	Referencia uma *Information*
output	*integer*	Referencia uma *Information*
inputs	*multival*	Referencia uma lista de *Information*. É utilizado para os parâmetros de uma invocação a um serviço web.

Tabela 16: ER GAP - Relação *Activity*.

A seguir são descritos os tipos de atividades que um comportamento pode ter:

- ***assign call*:** Atividade referente à invocação de um serviço web. No atributo *service* é armazenado o link para o serviço, em *operation* a operação. O atributo *output* é ligado à uma *Information*, que deve receber o retorno da invocação do serviço web e os *inputs* são os parâmetros que são passados à operação do serviço web. Esses parâmetros são também valores referentes à informações da relação *Information*;
- ***assign call var*:** Funcional tal como a atividade *assign call*, mas se difere por utilizar o link para o serviço web e a operação como variáveis, vindas de valores da relação *Information*, e não informações estáticas passadas na atividade;
- ***assign static*:** É uma atividade de atribuição estática. Uma variável *Information* recebe um valor passado de forma estática, e armazenado no atributo *static* da relação *Activity*;
- ***assign var*:** Atividade de atribuição variável, ou seja, uma variável *Information* (atributo *infoa*) recebe o valor de outra variável *Information* (atributo *infob*);

- ***conditional static*:** Equivale ao Se-Então de um algoritmo. Se a condição for satisfeita, então o bloco de atividades interno a essa atividade será executado. Nesse tipo de condicional, os atributos utilizados para efetuar a validação são *infoa*, referente à uma informação do comportamento, *operator*, que indicará o tipo de operação de comparação, e *static* sendo o valor estático da qual se quer comparar com a informação *infoa*. Os tipos de operações de comparação são:
  - ***a == b* :** retorna *true* se *a* é igual a *b*;
  - ***a != b* :** retorna *true* se *a* é diferente de *b;*
  - ***a < b* :** retorna *true* se *a* é menor que *b*;
  - ***a > b* :** retorna *true* se *a* é maior que *b*;
  - ***a <= b* :** retorna *true* se *a* é menor ou igual a *b*;
  - ***a >= b* :** retorna *true* se *a* é maior ou igual a *b*;
  - ***a contem b* :** retorna *true* se a cadeia de caracteres *a* contem a cadeia de caracteres *b;*
- ***conditional var*:** Funciona tal como a atividade anterior, contudo, a segunda informação também é uma variável *Information*, e não uma informação estática;
- ***conditional end*:** Indica onde o bloco condicional começa, por meio do atributo *goto*. As atividades *conditional static* e *conditional var* precisam ligar o atributo *goto* ao seu *conditional end* também.
- ***loop static*:** funciona tal como o *conditional static*, contudo, o bloco continua sendo executado enquanto a condição é satisfeita.
- ***loop var*:** Funciona tal como o *conditional var*, mas também continua executando enquanto a condição é satisfeita.
- ***loop end*:** Indica fim de bloco de um laço de execução e deve ser ligado com seu início de bloco de laço também, através dos atributo *goto*.

Essa base de dados é utilizada pelo servidor, para executar os comportamentos, e o usuário pode criar, alterar e excluir comportamentos por meio da interface web do GAP.

4.3.4.1 Servidor Gerenciador de Assistentes Pessoais

Os assistentes pessoais dos usuários precisam estar em execução para poder auxiliar nas suas atividades. Isso caracteriza a necessidade de ser um software que se mantém em execução, gerenciando os assistentes pessoais ativos. O sistema também é multiusuário e permite que diversos assistentes pessoais estejam sendo executados ao mesmo tempo.

A inspiração para o desenvolvimento do servidor GAP vem do gerenciamento de processos de Sistemas Operacionais (SO) e utilização de *threads* para paralelização de tarefas. Os computadores modernos podem efetuar várias coisas ao mesmo tempo que executam as aplicações dos usuários. Os sistemas operacionais que gerenciam esses computadores são sistemas de multiprogramação. No caso do computador possuir uma única CPU, os processos são alternados nessa. Isso é uma forma de pseudo-paralelismo em que a CPU executa um processo por dezenas ou centenas de milissegundos, retirando-o de execução e passando a executar outro processo na memória. Um SO mantém uma tabela com informações sobre os processos, seu ponteiro de execução do programa, alocação de memória, arquivos abertos, e outras (Tanenbaum, 2000).

Para Jandi Jr. (2004), o processo pode ser *"entendido como uma atividade que ocorre em um meio computacional"*. Esses também podem ser paralelizados, ocorrendo simultaneamente em durante um certo tempo. Esses concorrem aos recursos do computador na qual estão executando.

O protótipo aqui foi desenvolvido para suportar múltiplos usuários. Cada usuário pode possuir seu próprio Assistente Pessoal. Em vias de implementação, para cada SAP, uma *thread* é criada no processo servidor.

A classe *Server*, mostrada no diagrama de classes simplificado do servidor do GAP (Figura 51) se mantém em um laço de execução, avaliando a cada período de tempo se o assistente pessoal está ativo ou não. Se um assistente pessoal estava ativo, e agora o atributo *active*, se encontra com o valor *false*, significa que por algum motivo, ou por opção do usuário, o assistente foi desligado. A classe *Server* chama um método da classe *Assistant* avisando que o assistente foi desligado. Esta classe, por sua vez, toma as devidas medidas para salvar as informações na memória para que a classe *Server* possa tirar essa instância da classe *Assistant* da memória.

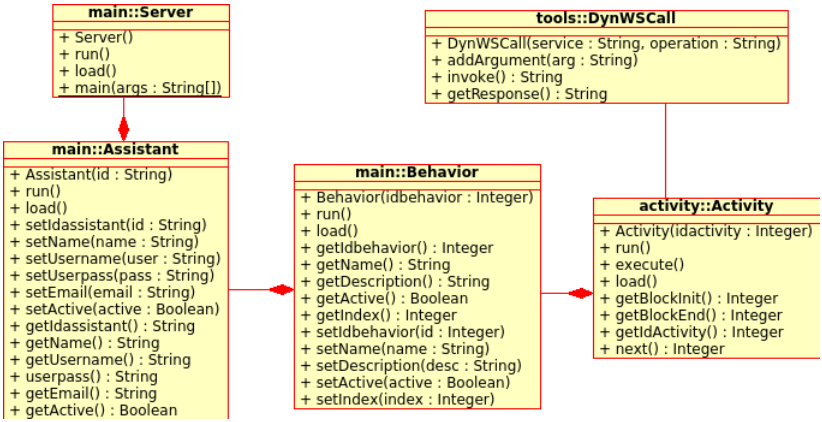

Figura 51: Diagrama de classes do GAP.

A classe *Assistant*, quando possui uma instância de uma assistente pessoal na memória, funciona na forma de *thread*. Ela é composta de uma lista de comportamentos e, sua execução se dá na forma de um laço em que cada comportamento, se ativo, é executado. Após isso, volta para o início do laço.

As *threads*, diferentemente de um processo tradicional em que há uma única linha de controle para cada processo, possuem suporte há múltiplas linhas de controle dentro de um processos. Elas ocupam o mesmo espaço de endereçamento, mas cada uma possui seu próprio ponteiro de controle de execução (Tanenbaum, 2000). Com a utilização de *threads* em um programa, é possível, por exemplo, abrir uma *thread* para enviar uma mensagem para um servidor e esperar o recebimento de um retorno ao mesmo tempo que o programa principal executa outras funções. O programa não precisa ficar esperando o retorno para continuar trabalhando em outras tarefas.

Cada *thread* de SAP pode possuir vários comportamentos. Esses comportamentos são compostos de várias linhas de execução (tal como um algoritmo) e precisam ser executados em paralelo. Para que a execução dos comportamentos possa ser feita paralelamente nesse protótipo foi utilizada a ideia do pseudo-paralelismo dos Sistemas Operacionais. Um comportamento é visto tal como um processo em um sistema operacional. Este é executado por uma fatia de tempo e seu processamento passa a outro processo, dando a impressão que todos estão sendo executados em paralelo (Tanenbaun, 2000).

Contudo, no caso aqui em questão, não é fornecida uma fatia de tempo ao processo (comportamento), mas sim, a cada ciclo de execução do SAP, cada comportamento executa apenas uma linha (chamada aqui de atividade, ou *activity*). Uma outra opção seria que cada comportamento também fosse uma *thread* de um SAP.

A execução dos comportamentos, instâncias da classe *Behavior*, é feita de forma bastante simples. Toda vez que o método *run* do comportamento é chamado, a atividade na posição *execpos* (ver Tabela 15) é executada. A próxima atividade a ser depende do *execpos* que retorna da execução de uma atividade. Por exemplo, se uma atividade simples de atribuição estática *assign static* é chamada, ela retorna a posição dela acrescida de um, ou seja, a atividade seguinte. Se for um tipo condicional, a posição retornada é, ou a primeira atividade do bloco (se válida a condição), ou a próxima posição após seu final de bloco (se inválida). No caso do laço, retorna ou a primeira posição do bloco, ou a próxima posição do final de bloco e no final do bloco do laço, a posição é direcionada para o início do bloco.

Quando *execpos* possui um valor maior do que a quantidade de atividades que o comportamento possui, então o valor de *execpos* retorna para 1, e a execução do algoritmo recomeça.

A classe *Activity* é, na verdade, uma classe pai, contendo uma classe filha que corresponde as atividades já descritas.

A classe DynWSCall é uma classe que facilita montar dinamicamente uma chamada à uma operação em um serviço web. Esta é utilizada pelas atividades *assign call* e *assign call var*.

4.3.4.2 Interface web para o Gerenciador de Assistentes Pessoais

A interface web para configuração e gerenciamento do assistente pessoal serve para cadastrar um novo assistente pessoal, adicionar informações, serviços e comportamentos. Na Figura 52 é mostrada a página inicial do assistente pessoal. O *menu* possui as opções de voltar para a página principal em *início*, editar as informações dos comportamentos e gerais em *Informações*, adicionar novos serviços web do usuário em *Serviços*, e efetuar logout do sistema em *Sair*.

Na esquerda da página aparece o avatar do assistente pessoal, a escolha do usuário e logo a baixo um botão para desligar/ligar o assistente pessoal. A direita o nome do assistente e um ícone que leva

para uma página destinada a alterar as informações do assistente pessoal e selecionar o avatar (Figura 53).

Figura 52: Pagina principal da interface web do Assistente Pessoal.

A direita, mais abaixo, são apresentadas algumas poucas informações sobre o usuário e em seguida os comportamentos. No caso da Figura 52, pode se visualizar a presença de sete comportamentos, que são descritos mais a frente. Cada um possui um botão de desligar/ligar, o nome do comportamento que é o link para a página de edição de comportamentos e a descrição do mesmo.

Novos comportamentos podem ser criados ao clicar no ícone (sinal de soma) ao lado do título *Comportamentos (X)*.

Figura 53: Configuração das informações do assistente pessoal.

As informações são acrescentadas ou alteradas por meio de um formulário (Figura 54), configurando o nome, o tipo (já descrito), se é global, uma descrição e o valor da informação.

Figura 54: Edição de Informações.

A opção *Serviços* (Figura 55) no *menu*, leva o usuário a uma página para adicionar serviços que lhe são úteis. Sua finalidade é facilitar o reconhecimento das operações do serviço, sem a necessidade de precisar interpretar um arquivo WSDL. A funcionalidade é tal qual comparada à um *bookmarks*, ou favoritos, de um navegador web.

Na opção *serviços*, o usuário pode cadastrar um novo serviço, alterar os existentes, ou visualizá-los. Ao clicar em Detalhes, suas operações são mostradas na tela e, se o usuário preferir, pode clicar no link WSDL ao lado do link do serviço, levando o usuário até o documento WSDL do serviço web.

Figura 55: Serviços web cadastrados.

A página de edição dos comportamentos (Figura 56) é separada em várias partes. A esquerda estão as informações específicas do comportamento, tal como o nome e a descrição. Também existem dois botões, o botão de ligar/desligar comportamento e o botão de recarregar informações na página. Mais a baixo logotipos com links para dois esforços para fornecer serviços web. O primeiro, *seekda*[27] funciona como uma espécie de *search engine* para pesquisar serviços web, e o segundo, é *Takever.eu*[28] uma comunidade de de desenvolvedores de serviços web que podem disponibilizar funcionalidades na forma de serviços web para outros desenvolvedores.

A direita, se o usuário clicar no link *Detalhes*, irá aparecer a lista de informações relativas ao comportamento em questão e também as informações que são globais. O usuário pode acrescentar novas informações, editar ou excluir. Observa-se que informações que estão sendo referenciadas por alguma atividade não podem ser excluídas, de forma não afetar a integridade referencial da base de dados do sistema.

Mais abaixo na figura é mostrada a maneira como o comportamento é executado, que se dá como uma forma de um algoritmo. É importante observar que, tanto o servidor do GAP quanto sua interface de configuração, são apenas protótipos e podem se apresentar como uma maneira não tão trivial de criação/edição de comportamentos.

[27]Seekda é um motor de buscas para APIs web no formato de serviços web e de provedores de serviços web (SEEKDA, 2010).

[28]O Tekever.eu é uma comunidade online de desenvolvimento de serviços web (TEKEVER, 2011).

Figura 56: Configuração de comportamento.

A opção de configurar o comportamento do assistente pessoal de forma algorítmica, e da forma como é aqui apresentado foi de escolha do autor da Tese, que também é o desenvolvedor da Instância implementada (protótipo). Não necessariamente o comportamento deve ser dessa forma. Cada desenvolvedor pode implementar a sua própria forma de configurar, criar, editar comportamentos. Podem desenvolver maneiras mais fáceis e intuitivas, ou mais aperfeiçoadas. Entretanto, a escolha da forma de execução dos comportamentos não compromete de forma alguma a proposta da Tese, de uma Arquitetura de Referência para Assistentes Pessoais sob o contexto de SOA.

Em essência, o comportamento é composto de uma estrutura que é executada em determinadas condições e suas atividades são chamadas à serviços web.

Um comportamento, quando criado, possui apenas a primeira linha de atividade, que é definida como *INÍCIO*, e a última linha, *FIM*. Para acrescentar uma nova atividade, na linha seguinte, o usuário deve clicar no um ícone em formato de sinal de soma, que fica nas linhas de atividades. O ícone de sinal de diminuição serve para excluir uma

atividade. Caso uma atividade seja um bloco, apenas o início do bloco e o final são excluídos, restando os elementos internos.

As setas apontando para cima e para baixo servem para alterar a ordem de uma atividade. Ao clicar a seta para cima, uma atividade é trocada pela atividade anterior, se clicado na eta para baixo, a atividade é trocada pela posterior. Quando um início ou fim de bloco é movido, todo o bloco vai junto. Se ao mover um bloco para cima, for encontrado o final de outro bloco, então o bloco que foi movido é movido para dentro do bloco superior. Também acontece isso quando movido para baixo. Quando um bloco é movido para cima, e a atividade superior é outro bloco, então o bloco movido é retirado todo do bloco mais externo. Da mesma forma se dá ao mover para baixo e ao se encontrar o fim de um bloco externo.

Quando o usuário passa com o mouse por cima de informações e outros itens da atividade, outras informações são apresentadas, como o valor de uma informação, o link de um serviço web, etc.

A atividade que se encontra em execução momento em que a página é carregada, é apresentada com uma cor azul claro. Para que o usuário possa visualizar o andamento da execução, é necessário clicar no botão **recarregar**, na parte esquerda da página. Neste protótipo, as informações não são carregadas automaticamente.

O ícone de edição, entre o sinal de diminuição e a seta para cima, serve para editar um comportamento existente. Na Figura 57 é apresentado um exemplo de edição de uma atividade condicional estática. No primeiro campo escolhe-se a informação que se quer comprar, no segundo o tipo de comparação e no terceiro o valor estático. O terceiro campo, quando na edição de uma atividade variável, é também um campo de seleção de outra informação.

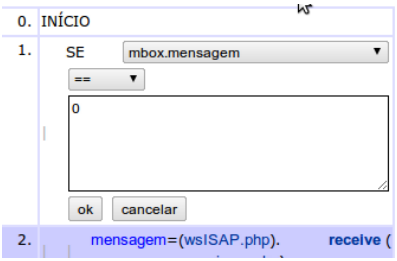

Figura 57: Edição de atividade condicional.

Na Figura 58 é apresentado um outro exemplo de edição de atividade. Para cada tipo de atividade diferente, um formulário diferente é apresentado. No caso, é a edição de uma atividade de chamada de serviço web (Linha 2). O primeiro campo do formulário é a informação que deve receber o retorno da invocação do método, o segundo campo é o link do serviço e o terceiro a operação. Ao colocar ou alterar essas informações, deve ser selecionado o botão "*ok*" para salvar os dados, antes mesmo de selecionar os parâmetros. Nos parâmetros, há uma lista de parâmetros que já foram inseridos, na ordem que deve estar na operação do serviço web. Também à um botão de excluir, para retirar um parâmetro já inserido. Para adicionar um novo parâmetro, basta selecionar selecionar da lista de informações do comportamento e informações globais e clicar no botão "adicionar".

Figura 58: Edição de uma atividade Assign Call.

Quando uma atividade do tipo condicional ou laço é criada, automaticamente, e logo na sequencia, uma atividade de fechamento de bloco é criado (FIM SE ou FIM ENQUANTO).

4.3.5 Serviços Para Softwares Assistentes Pessoais

Com o contexto dos Softwares Assistentes Pessoais surge a possibilidade da criação de serviços específicos para serem utilizados pelos SAPs. Para o caso de implementação selecionado, foi implementado um serviço específico chamado de Interface Social para Assistentes Pessoais (ISAP). Este tem a finalidade de fazer o gerenciamento dos recursos de interação entre o SAP e o usuário via outros serviços como Twitter, E-mail, Blog, IM, etc.

O ISAP é separado em três partes principais, um servidor que fica gerenciando os eventos que são lidos dos recursos de interação, uma interface web para configuração e um serviço web que serve de interface para ser acessada pelo SAP.

4.3.5.1 Servidor ISAP

Primeiramente, a justificativa para a criação do ISAP na forma de um servidor é liberar o GAP (Gerenciador de Assistentes Pessoais) do máximo de trabalho possível em relação ao interfaceamento com o usuário. Dessa forma, o ISAP fica responsável por receber mensagens do usuário, analisá-la, filtrar as informações e apenas enviar para o GAP o que é realmente relevante. Contudo, essa implementação requer que o GAP envie regularmente para o ISAP um *alive*, caso o assistente pessoal esteja em funcionamento. De outra forma, o ISAP vai "achar" que o assistente pessoal não está mais funcionando, por qualquer motivo que seja, e então desativa a conta de ISAP do usuário. Isso previne que o usuário do assistente pessoal continue enviando comandos ao GAP, via ISAP, sem que o GAP possa receber o comando.

Na Figura 59 é apresentado o modelo entidade-relacionamento do ISAP, suas relações, relacionamentos e atributos.

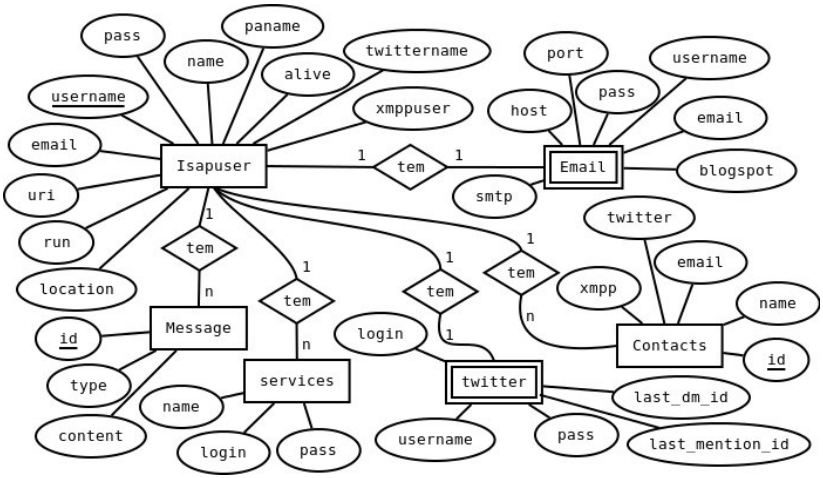

Figura 59: Diagrama ER do ISAP.

A relação Isapuser é referente às informações dos usuários do ISAP, conforme Tabela 17:

Isapuser		
Atributo	**Tipo**	**Descrição**
* *username*	*string*	Login de usuário
name	*string*	Nome do usuário
pass	*string*	Senha para autenticação
email	*string*	E-mail de contato
paname	*string*	Nome do assistente pessoal do usuário
twittername	*string*	Id do usuário no Twitter
xmppuser	*string*	Id do usuário no Gtalk – e-mail do Gmail
location	*string*	Local do serviço web do GAP, se houver
uri	*string*	URI do serviço web do GAP, se houver
Alive	*boolean*	*True* se recebeu um *alive* do GAP.
run	*boolean*	*True* se a conta do usuário no ISAP está ativa, rodando.

Tabela 17: ER ISAP - Relação *Isapuser*.

Algumas informações apresentadas na Tabela 17, são necessárias para que o assistente pessoal reconheça seu usuário por meio de certos serviços, como e-mail, Twitter e Gtalk. A relação *Message*, por sua vez, possui os atributos apresentados na Tabela 18:

Message		
Atributo	**Tipo**	**Descrição**
* *id*	*string*	Login de usuário
username	*string*	Chave estrangeira referenciando a relação *Isapuser*
type	*string*	Tipo da mensagem (in : entrada, out : saída)
content	*string*	Conteúdo da mensagem

Tabela 18: ER ISAP - Relação *Message*.

A finalidade da relação *Message* é armazenar mensagens que são trocadas entre o GAP e o ISAP. As mensagens que chegam *(type in)* são avaliadas pelo ISAP, que encontra a melhor maneira de enviar para o usuário do Assistente Pessoal. As mensagens que são para o GAP (*type out*) ficam armazenadas ali, até que o GAP as busque via serviço web do ISAP (wsISAP).

A relação *Twitter* possui apenas informações sobre o usuário do Assistente Pessoal representado por um usuário no Twitter. Essas informações são atualizadas de forma automática quando o usuário autoriza a utilização do aplicativo via protocolo OAuth. Também são armazenadas as informações *last_dm_id* e *last_mention_id*, que são respectivas ao *id* da última mensagem direta analisada e ao *id* da última menção feita ao assistente pessoal analisada. Isso previne que o assistente pessoal se mantenha analisando sempre as primeiras informações.

A relação *Services* é destinada a aplicações que precisam ser configuradas apenas por um login e uma senha. No caso, apenas o serviço de Gtalk está configurado.

A relação *Email* (Tabela 19) é destinada a configurar não apenas a conta de e-mail do Assistente Pessoal, mas também a forma de acesso ao blog (*blogspot*).

Email		
Atributo	**Tipo**	**Descrição**
**username*	*string*	Chave estrangeira que referencia a relação Isapuser
host	*string*	Endereço do *host* POP3
port	*string*	Porta do servidor
pass	*string*	Senha do usuário
smtp	*string*	Endereço do *host* SMTP
username	*string*	Nome do usuário
email	*string*	E-mail do assistente pessoal
blogspot	*string*	E-mail para enviar postagens para o Blog

Tabela 19: ER ISAP - Relação *Message*.

A relação *Contacts* (Tabela 20), por sua vez, serve para contatar usuários especificamente. Quando chega uma mensagem do GAP, se a mensagem estiver no formato KQML sobre XML, essa é avaliada para verificar o destinatário. Este é procurado na lista de contatos do usuário, tanto pelo usuário no Twitter, como por e-mail, como por nome. O primeiro que fechar é utilizado.

Contacts		
Atributo	**Tipo**	**Descrição**
* *id*	*int*	Identificador único e auto incrementável
username	*string*	Chave estrangeira referenciando a relação *Isapuser*
name	*string*	Nome do contato
email	*string*	E-mail do contato
twitter	*string*	Id do contato no Twitter
xmpp	*string*	Id do contato no Gtalk

Tabela 20: ER ISAP - Relação *Message*.

Na Figura 60 é apresentado o diagrama de classes das principais classes do sistema servidor do ISAP.

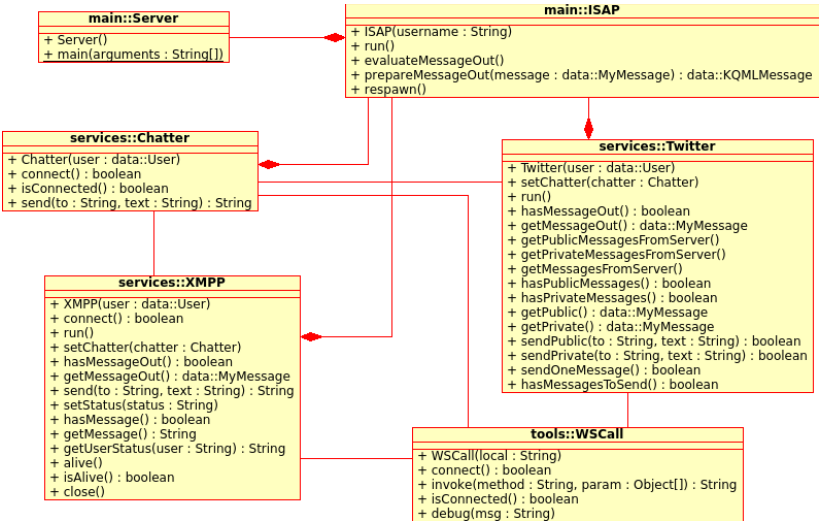

Figura 60: Diagrama de classes do ISAP.

A classe *Server* a cada certo tempo avalia se algum novo usuário de ISAP foi criado e então cria uma instância na memória referente a esse usuário, caso o sistema esteja ativo, que é a classe *ISAP*. Também, se uma usuário não mais está ativo, então o *Server* se encarrega de retirar as informações da lista de usuários de ISAP em funcionamento.

As classes *Twitter* e *XMPP*, quando chamado o método *run*, avaliam se alguma mensagem chegou. Quando uma mensagem chega, é

chamada a classe *Chatter*, que efetua uma invocação ao serviço web *wsChatter*, retornando uma resposta. Essa resposta é já enviada de volta a quem enviou a mensagem. Se a mensagem possui a palavra-chave *$to*, então é uma mensagem específica para ser enviada ao GAP, e deve ser armazenada em uma lista de mensagens de saída de cada um dos serviços.

A execução da classe *ISAP* executa, periodicamente cada um dos serviços (XMPP e Gtalk, por exemplo). Se uma mensagem de saída é detectada, então ela é organizada como uma mensagem KQML/XML e é colocada na lista de mensagens de envio.

Se o *ISAP* detecta na lista de mensagens de entrada que alguma mensagem chegou, então é avaliado para quem a mesma é destinada, e verificado na lista de contatos. Após isso a mensagem é enviada para o contato específico. Aqui se torna importante o atributo *status* adicionado ao padrão KQML.

Se uma mensagem é de estado pública, então ela é enviada para o usuário no *timeline* do Twitter. Se for privada, então é enviada ao contato via Gtalk. Contudo, se o contato estiver desconectado no Gtalk, a mensagem é então enviada via *Direct Message* no Twitter. Se o usuário cadastrou seu celular no Twitter, então essa será direcionada como mensagem de texto SMS ao seu celular.

As classes *XMPP*, *Twitter* e *Chatter* se utilizam de invocações aos serviços web *wsXMPP*, *wsTwit* e *wsChatter* respectivamente. Dessa forma, se utilizam de uma classe facilitadora chamada *WSCall*.

4.3.5.2 Serviço web *wsISAP*

O serviço web wsISAP servem não apenas para gerenciar o usuário do ISAP de um Assistente Pessoal específico, mas como também fornece ao utilizador do serviço web acesso à lista de contatos do usuário no ISAP. Todas as operações possuem como primeiro parâmetro o *id* do usuário no ISAP e como segundo parâmetro a senha. As operações referentes à gerência do ISAP via serviço web são as seguintes:

- ***start*:** não possui outros parâmetros e envia um comando para iniciar o gerenciador da interface de um usuário;
- ***stop*:** envia um comando para o ISAP para terminar a execução do gerenciador da interface do usuário;

- ***alive*:** envia um *alive* para o ISAP, avisando que o usuário em questão está funcionando e necessita que a interface continue ativa;
- ***respawn*:** envia um comando para o ISAP recarregar as informações da base de dados referentes ao usuário e todos os dados para o funcionamento da interface deste;
- ***send*:** envia uma menagem ao ISAP. O terceiro parâmetro é a mensagem, normalmente em formato KQML/XML;
- ***receive*:** busca a próxima mensagem do ISAP que se destina ao GAP. Retorna 0 se não houverem mensagens;
- ***sendKQML*:** envia uma mensagem ao ISAP. Esse método pega os parâmetros *(id, type, sender, receiver, in_replay_to, language, ontology, status, content),* os monta no formato KQML/XML e coloca a mensagem na lista de mensagens de entrada.

As operações referentes ao gerenciamento da lista de contatos do usuário no ISAP são as seguintes:

- ***contactsAdd*:** Adiciona um contato à lista de contatos do usuário no ISAP. Os parâmetros subsequentes são as informações do contato, ou seja, o nome, o e-mail, o id no Gtalk e o id no Twitter;
- ***contactsGetName*:** Retorna o nome de um contato. É passado como parâmetro uma palavra-chave que é pesquisada na lista dos contatos nos atributos de Nome, e-mail, id no Twitter e id no Gtalk. Se fechar algum desses atributos, então o nome do contato é retornado;
- ***contactsGetTwitter*:** Funciona tal como na operação anterior, mas retorna o id no Twitter do contato;
- ***contactsGetEmail*:** Tal como as operações anteriores, mas retorna o e-mail;
- ***contactsGetXMPP*:** Retorna o id do Gtalk do contato.

É importante observar que o próprio usuário do Assistente Pessoal também deve estar na lista de contatos no ISAP.

4.3.5.3 Interface web de configuração do ISAP

A interface web do ISAP foi customizada de forma a ser melhor vista em *smartphones*. Essa é composta por diversos ícones, cada qual com função específica (Figura 61).

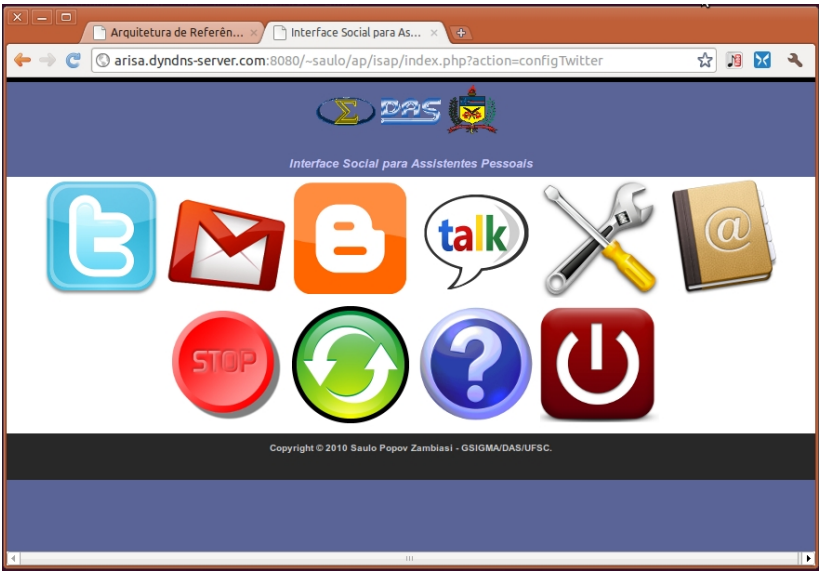

Figura 61: Interface web do ISAP.

A operações são realizadas ao clicar nos ícones, que estão dispostos conforme visto na Figura 61, conforme observados da esquerda para a direita e de cima para baixo:

- Configuração do Twitter para habilitar a utilização da aplicação no Twitter. É necessário que o usuário esteja logado no navegador pela conta do seu Assistente Pessoal no Twitter;
- Configuração do e-mail do Assistente Pessoal;
- Configuração das informações para postagem no Blog;
- Configurações do Gtalk;
- Configurações gerais do usuário do ISAP;
- Lista de contatos do usuário no ISAP;
- Botão de *STOP*. Aparece quando o sistema está funcionando. Quando não, aparece o botão *START*;

- Botão *RELOAD*, para recarregar as informações e serviços do usuário no ISAP. Esse botão só aparece se o sistema para o usuário específico estiver funcionando;
- Botão de *HELP*, ainda não implementado;
- Botão de *EXIT*, para efetuar o *logout* o usuário.

Ao clicar no botão do Twitter, se o usuário ainda não tiver cadastrado as informações de OAuth, então ele levará até uma página do próprio Twitter, para que a aplicação seja autorizada pelo usuário. Após isso, a tela retorna à interface web do ISAP, cadastrando então as informações do usuário para acesso ao Twitter automaticamente. Se as informações já estiverem cadastradas, então o sistema pede confirmação se o usuário quer efetuar nova configuração.

4.3.6 Assistente e Comportamentos

De modo a apresentar o assistente pessoal mais humanizado, nessa implementação (critério de escolha do desenvolvedor), cada usuário pode criar uma caracterização personificada para seu Assistente Pessoal. Sendo que o sistema implementado é multiusuário. Cada usuário de assistente pode escolher um nome e uma imagem para representar visualmente seu assistente pessoal. No caso, o assistente pessoal utilizado para os testes e avaliação da implementação é um personagem feminino de nome Arisa (*Assistant Representative: an Instance using Services Architecture*) – Lê-se "arissá". Este possui um nome (inclusive pronúncia) e uma imagem ilustrativa (Figura 62) no estilo de mangas e animes japoneses[29].

[29]Manga é um estilo literário japonês equivalente as histórias em quadrinhos (HQ) do ocidente, e anime é equivalente aos desenhos animados ocidentais. Seus traços são bastante característicos, com os personagens possuindo olhos grandes, nariz e boca pequena. Normalmente o estilo do cabelo e roupa é o que diferencia os personagens.

Figura 62: Avatar da assistente pessoal Arisa.

Para criar a imagem ilustrativa do avatar da Arisa, na Figura 62, foi utilizado um software chamado "*Anime Character Generator 2.05*".

Antes da criação do Assistente Pessoal no GAP, é necessário que o usuário já possua e crie contas em alguns serviços já existentes, descritos na seção 4.3.1. Algumas dessas contas devem ser criadas especificamente para o assistente pessoal.

4.3.6.1 Conta de E-mail e IM

Uma das formas de comunicação entre o SAP e o usuário é via troca de e-mails e troca de mensagens via IM. Para isso, é necessário que ambos possuam uma conta no mesmo serviço. Neste caso de implementação é utilizado o serviço da Google. Por meio da criação de uma conta de e-mail no Gmail, o usuário também tem acesso ao serviço de IM Gtalk. As contas utilizadas para os testes são:

- Conta de e-mail e IM do usuário: **saulopz@gmail.com**;
- Conta de e-mail e IM do assistente pessoal: **personal.assistant.arisa@gmail.com**.

Para os testes realizados no sistema implementado é utilizado o programa de IM chamado Pidgin[30].

[30]Pidgin é um programa de chat que permite que o usuário possa se logar em várias contas de Instant Messaging (PIDGIN, 2010).

4.3.6.2 Serviço de Microblog Twitter

O usuário do assistente pessoal deve possuir uma conta no twitter, assim como também seu assistente pessoal. Para isso, é necessário acessar o site do twitter e efetuar a criação das contas. As contas utilizadas nos testes do sistema são:

- Conta do usuário no twitter: **noctislupus**.
- Conta do assistente pessoal no twitter: **arisa_ap**.

Entretanto, para que possa se criar uma conta no twitter, é necessário que tanto o usuário quanto o Assistente Pessoal já possuam as conta de e-mail registradas em algum servidor de e-mails.

4.3.6.3 Serviço de Blog

O blog tem a finalidade de publicar relatórios do Assistente Pessoal classificados como públicos. Para os testes, é utilizado pelo assistente pessoal o blog:

- ***http://arisa-ap.blogspot.com/*** - blog do assistente pessoal. Utilizado para testes de envio de relatórios de atividades classificadas como públicas. Entretanto, foi criado o seguinte domínio nacional para o blog: ***http://www.projetoarisa.com/***.

Para que o assistente pessoal possa efetuar a postagem de um texto no blog, basta apenas o assistente pessoal enviar um e-mail para: ***personal.assistant.arisa.SENHA@blogger.com***. É necessário que essa configuração seja efetuada no serviço de *blog*, e uma palavra chave (**SENHA**) seja configurada.

4.3.6.4 Configuração dos Serviços

Após a criação das contas de e-mail, twitter e blog, é necessário que o usuário acesse as interfaces web para configuração dos serviços de *Chatbot*, ISAP, Report, e MailA. Também é necessário a criação de uma empresa no sistema de Estoque e algumas empresas no sistema de fornecedores.

4.3.7 Comportamentos do SAP

Por escolha do autor da Tese, e de forma a mostrar a maneira de implementar um comportamento mais complexo no GAP, o processo de compra automático é aqui implementado como comportamentos feitos por um usuário mais especializado. Uma outra escolha, seria que tal comportamento poderia ser desenvolvido por um desenvolvedor na forma de um serviço web e o usuário poderia agregá-lo ao seu assistente pessoal apenas na forma de chamada de serviço web, configurando as informações necessárias para a invocação do serviço.

Neste exemplo, os comportamentos que desenvolvidos são: *AtualizaHora, ISAPalive, mbox, Report, Compra-ordem, Compra-altera* e *compra*. Cada um desses comportamentos é apresentado e descrito em sequência. O comportamento da compra, especificamente, foi separado em três partes. O *Compra-ordem* procura os produtos com estoque baixo e cria uma ordem de compra no sistema de estoque e uma ordem de venda no fornecedor, o *Compra-altera* aguarda confirmação ou cancelamento do usuário caso a compra tenha tido alterações pelo fornecedor, e o *Compra* efetua a compra, efetivamente.

4.3.7.1 Comportamento *AtualizaHora*

O comportamento *AtualizaHora* (Figura 63) é um comportamento bastante simples e efetua apenas invocação de operações do serviço web *datetime*, descrito na seção 8.11.

```
0. INÍCIO
1.   dia=(datetime.php). getLocalDate ()
2.   hora=(datetime.php). getLocalTime ()
     FIM
```

Figura 63: Comportamento AtualizaHora.

Sua função é manter as informações sobre o dia e horário atual sempre atualizados, pois caso algum comportamento necessite dessa informação, ela deve estar atualizada. No comportamento existe apenas duas variáveis globais, dia e hora. Na linha 1 do comportamento a operação *getLocalDate* do serviço web é chamado, retornando a data, e na linha 2, a operação *getLocalTime* retorna a hora. Nenhuma dessas

operações necessita da passagem de qualquer parâmetro. Quando o comportamento for todo executado, sua execução retorna para a primeira linha e assim continua enquanto o comportamento e o assistente pessoal estiverem ligados.

4.3.7.2 Comportamento *ISAPalive*

O comportamentos *ISAPalive* (Figura 64) possui a única funcionalidade de informar, de tempos em tempos, que o assistente pessoal ainda se encontra em funcionamento. Logo, a interface com o usuário também deve se encontrar ativa.

```
0. INÍCIO
1. |   retorno=(wsISAP.php). alive ( usuario, senha)
2. |   timer = 1
3. |   ENQUANTO (timer < max) FAZER
4. |   |   timer=(math.php). increment ( timer, max)
5. |   FIM ENQUANTO
   FIM
```

Figura 64: Comportamento ISAPalive.

Seu funcionamento se dá da seguinte forma:

- **Linha 1:** a operação *alive* do serviço web *wsISAP* é chamado, enviando as informações do login do usuário e a senha no ISAP. A informação *retorno* é uma variável auxiliar apenas para receber o resultado da invocação do método. Na implementação é sempre necessário que uma informação receba o resultado;
- **Linha 2:** a informação *timer* recebe valor 1;
- **Linha 3:** inicia um laço. Enquanto *timer* for menor que *max*, o laço deve continuar. A informação *max* possui o valor 4. A intenção desse laço é que não seja enviado *alive* ao ISAP para cada execução do comportamento, pois não é necessário um tempo tão curto de *alive*;
- **Linha 4:** a operação *increment* do serviço web *math* é chamado, incrementando a informação *timer* em um;
- **Linha 5:** fecha o bloco de ENQUANTO.

Basicamente o comportamento chama o *alive* no ISAP e entra em um laço, esperando um certo tempo para então enviar um novo *alive*.

4.3.7.3 Comportamento *mbox*

O comportamento *mbox* (Figura 65) é responsável por receber periodicamente mensagens vindas do ISAP e armazenar no serviço web *wsmbox*.

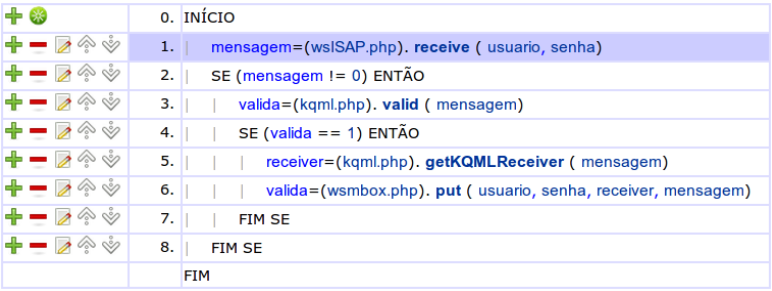

Figura 65: Comportamento mbox.

O funcionamento deste comportamento se dá da seguinte forma:

- **Linha 1:** invoca o método *receive* do serviço web *wsISAP*, retornando uma mensagem, se houver alguma mensagem de saída, ou 0, caso contrário;
- **Linha 2:** se a informação *mensagem* for diferente de 0, significa que uma nova mensagem foi recebida. O fim desse condicional termina na **Linha 8**;
- **Linha 3:** invoca o método *valid* do serviço web *kqml*, de forma a verificar se é uma mensagem no estilo KQML/XML e armazena o resultado na informação *valida*;
- **Linha 4:** se *valida* for igual a 1, significa que a mensagem está no estilo KQML/XML. O fim do condicional termina na **Linha 7**;
- **Linha 5:** invoca o método *getKQMLReceiver* do serviço web *kqml* para saber qual é o destinatário da mensagem. A informação *receiver* recebe o resultado da invocação;
- **Linha 6:** invoca o método *put* do serviço web *wsmbox*, armazenando uma mensagem. Os parâmetros de entrada

passados é o login do usuário no *wsmbox*, a senha, o identificador para saber quem é o destinatário da mensagem e a mensagem em si. A informação *válida* é utilizada apenas como uma variável auxiliar porque uma invocação sempre precisa, nessa implementação, passar o resultado para uma informação;

Essas mensagens são depois utilizadas pelo comportamento de *Compra-altera*, para verificar se o usuário confirmou as alterações ou não. O destinatário, nesse caso, é o comportamento *Compra-altera*, que é informado pelo usuário quando enviada uma menagem ao assistente pessoal, na forma *"$to Compra-altera confirmar ordem 34"*.

4.3.7.4 Comportamento *GerenciaEmail*

O comportamento *GerenciaEmail* (Figura 66) serve para efetuar a verificação dos e-mails do usuário, filtrar e-mails importantes para serem enviados como mensagem ao usuário e responder e-mails, se configurado, para os remetentes.

```
0.  INÍCIO
1.  |  conectado=(wsISAP.php). isContactOnline ( usuario, senha, contato)
2.  |  SE (conectado == 0) ENTÃO
3.  |  |  retorno=(wsMaila.php). nextMail ( Usuário, Senha)
4.  |  |  SE (retorno != 0) ENTÃO
5.  |  |  |  SE (retorno != 1) ENTÃO
6.  |  |  |  |  mensagem=(kqml.php). toXML ( null, null, comportamento, mestre, null, null, null, tipo, retorno)
7.  |  |  |  |  aux=(wsISAP.php). send ( usuario, senha, mensagem)
8.  |  |  |  FIM SE
9.  |  |  FIM SE
10. |  FIM SE
    FIM
```

Figura 66: Comportamento de Gerência de Emails.

Sua funcionalidade se dá da seguinte forma:

- **Linha 1:** Invoca a operação *isConcactOnline* do serviço web *wsISAP* passando como parâmetro o usuário e a senha no ISAP, além do contato da qual se deseja a informação se está, ou não, conectado. No caso, o próprio usuário do assistente pessoal
- **Linha 2:** Se o usuário estiver conectado, então segue.
- **Linha 3:** Executa a operação *nextMail* do serviço web *wsMaila*. Este retorna o próximo e-mail marcado como não lido.

- **Linha 4 e Linha 5:** Se não retornar nem 0, nem 1, significa que há alguma mensagem para enviar ao usuário. No caso de retorno 1, implica que houve alguma atividade que foi efetuada no serviço *Mail Assistance*.
- **Linha 6:** Utiliza a operação *toXML* do serviço web *kqml* para criar uma mensagem no estilo KQML/XML para enviar ao usuário, via ISAP.
- **Linha 7:** Por meio da operação *send* do serviço *wsISAP*, envia uma mensagem ao usuário sobre uma operação específica ocorrida no *Mail Assistence*.

Este comportamento só entra efetivamente em execução quando o usuário não se encontra conectado no seu GTalk no momento em questão.

4.3.7.5 Comportamento *Report*

A finalidade do comportamento *Report* (Figura 67) é enviar um relatório para o usuário todo dia, neste caso, as **22:30** horas.

```
0.  INÍCIO
1.  |  SE (hora contem 22:30:) ENTÃO
2.  |  |  relatorioPrivado=(wsReport.php). getPrivate ( usuario, senha)
3.  |  |  relatorioPublico=(wsReport.php). getPublic ( usuario, senha)
4.  |  |  subject = Relatorio
5.  |  |  subject=(string.php). concat ( subject, dia)
6.  |  |  SE (relatorioPrivado != 0) ENTÃO
7.  |  |  |  retorno=(wsMail.php). sendMail ( host, port, user, pass, username, to, subject, relatorioPrivado)
8.  |  |  |  relatorioPrivado = 0
9.  |  |  FIM SE
10. |  |  SE (relatorioPublico != 0) ENTÃO
11. |  |  |  retorno=(wsMail.php). sendMail ( host, port, user, pass, username, toBlog, subject, relatorioPublico)
12. |  |  |  relatorioPublico = 0
13. |  |  FIM SE
14. |  FIM SE
    FIM
```

Figura 67: Comportamento Report.

O seu funcionamento se dá da seguinte forma:

- **Linha 1:** Se a hora contiver a cadeia de caracteres "**22:30:**", então entra no bloco. Este bloco termina na **Linha 14**. O motivo de haver os dois pontos no final da cadeia de caracteres é para que a hora não seja confundida com 22 minutos e 30 segundos.
- **Linha 2:** é invocada a operação *getPrivate* do serviço web *wsReport*, passando como parâmetros o usuário e a senha do serviço, e retornando o resultado na informação *relatorioPrivado*.
- **Linha 3:** faz a mesma coisa que a linha 3, mas retorna em *relatorioPublico* o resultado da invocação da operação *getPublic*.
- **Linha 4:** a informação *subject* recebe a cadeia de caracteres "**Relatório**".
- **Linha 5:** a informação *subject* recebe o resultado da invocação *concat* do serviço web *string*. Os parâmetros são o próprio *subject* e o dia atual, atualizado pelo *AtualizaHora*. Em *subject* deve estar algo como, por exemplo, "**Relatório 2011-02-26**".
- **Linha 6:** se *relatorioPrivado* for diferente de 0, significa que existe um relatório privado para enviar ao usuário. Então entra no bloco de execução.
- **Linha 7:** utiliza a invocação da operação *sendMail* do serviço web *wsMail* para enviar um e-mail para o usuário, contendo o relatório.
- **Linha 10:** se *relatorioPublico* for diferente de 0, então significa que retornou um relatório público de *wsReport*.
- **Linha 11:** invoca a operação *sendMail* para enviar um e-mail para o serviço de *Blog* enviando o relatório como postagem.

Na Figura 68 é apresentado um *screenshot* do programa *Mozilla Thunderbird*, com um e-mail recebido do assistente pessoal do usuário com um relatório privado, ou seja, das atividades com *status* setado como *private*.

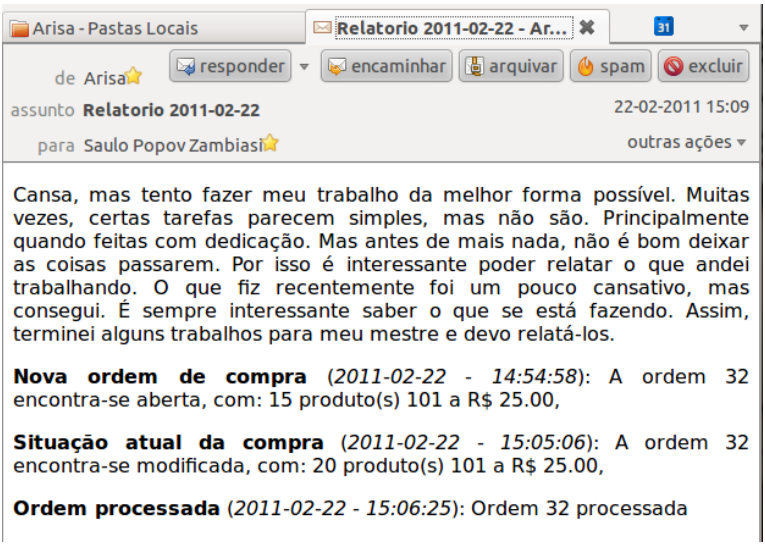

Figura 68: Exemplo de relatório privado, por e-mail.

Como descrito no sistema de geração de relatório, inicialmente há um texto de introdutório, depois são apresentadas as tarefas realizadas. A hora do recebimento do e-mail, na Figura 68 está como "**15:09**", diferente do que está configurado no comportamento, pois foi o horário em que foi realizado o teste no dia.

4.3.7.6 Comportamento *Compra-ordem*

A funcionalidade do comportamento *Compra-ordem* (Figura 69) é efetuar periodicamente se algum produto se encontra com o estoque baixo. Em caso positivo, uma nova ordem de compra é então criada no sistema de estoque, que o usuário do assistente pessoal gerencia, e uma ordem de venda é criada do lado do fornecedor.

```
0.  INÍCIO
1.  | produtoID=(wsEstoque.php). nextProdutoEstoqueBaixo ( estoqueCNPJ, estoqueSenha, produtoID)
2.  | SE (produtoID != 0) ENTÃO
3.  | | produtoQuantidade=(wsEstoque.php). getProdutoQuantidadeParaCompra ( estoqueCNPJ, estoqueSenha, produtoID)
4.  | | ordemID=(wsOrdem.php). criaOrdem ( estoqueCNPJ, estoqueSenha, produtoID, produtoQuantidade)
5.  | | SE (ordemID != 0) ENTÃO
6.  | | | fornecedorCNPJ=(wsOrdem.php). getFornecedorCNPJ ( estoqueCNPJ, estoqueSenha, ordemID)
7.  | | | vendaCodigo=(wsVenda.php). novaOrdem ( fornecedorCNPJ, aux, estoqueCNPJ)
8.  | | | SE (vendaCodigo != 0) ENTÃO
9.  | | | | aux=(wsOrdem.php). setIdVenda ( estoqueCNPJ, estoqueSenha, ordemID, vendaCodigo)
10. | | | | produtoCodigo=(wsEstoque.php). getProdutoCodigo ( estoqueCNPJ, estoqueSenha, produtoID)
11. | | | | produtoValor=(wsOrdem.php). getProdutoValor ( estoqueCNPJ, estoqueSenha, ordemID, produtoID)
12. | | | | aux=(wsVenda.php). setProduto ( vendaCodigo, produtoCodigo, produtoQuantidade, produtoValor)
13. | | | | ordemSituacao=(wsOrdem.php). situacaoOrdem ( estoqueCNPJ, estoqueSenha, ordemID)
14. | | | | SE (ordemSituacao != 0) ENTÃO
15. | | | | | aux=(wsReport.php). send ( usuario, senha, reportTitulo, ordemSituacao, mensagemEstado)
16. | | | | | aux=(kqml.php). toXML ( null, null, comportamentoNome, mestre, null, null, null, mensagemEstado, ordemSituacao)
17. | | | | | aux=(wsISAP.php). send ( usuario, senha, aux)
18. | | | | FIM SE
19. | | | FIM SE
20. | | FIM SE
21. | FIM SE
    FIM
```

Figura 69: Comportamento Compra-ordem.

O comportamento *Compra-ordem* funciona da seguinte forma:

- **Linha 1:** a operação *nextProdutoEstoqueBaixo* do serviço web *wsEstoque* é invocado, retornando 0 se todos os produtos estiverem com seu estoque em dia, ou o *id* do próximo produto da lista de produtos com estoque baixo. Como parâmetros de entrada são enviados o CNPJ da empresa e a senha para acessar o sistema de estoque, e o último *id* de produto verificado. Se for 0, então pega o primeiro da lista. O resultado da invocação é armazenado na informação *produtoID*.
- **Linha 2:** Se *produtoID* é diferente de zero, então segue.
- **Linha 3:** invoca *getProdutoQuantidadeParaCompra* do serviço web *wsEstoque* para criar a ordem de compra.
- **Linha 4:** Invoca a operação *criaOrdem*, para gerar uma nova ordem de compra para um produto e sua quantidade.
- **Linha 5:** Se a ordem foi criada com sucesso, então segue.
- **Linha 6:** Retorna o CNPJ do fornecedor escolhido pelo sistema de estoque para efetuar a compra. Se uma ordem com esse fornecedor foi cancelada, então um novo fornecedor é selecionado.

- **Linha 7:** Invoca a operação *novaOrdem* no serviço web *wsVenda* para gerar uma nova ordem de venda referente ao fornecedor *fornecedorCNPJ*. O resultado é o código da venda gerado no sistema de venda automatizado (Servidor Venda).
- **Linha 8:** Se foi criada a ordem de venda com sucesso, então.
- **Linha 9:** Invoca a operação *setIdVenda* de *wsEstoque*, para fazer a conexão de uma ordem de compra no estoque com uma ordem de venda no fornecedor.
- **Linha 10:** invoca a operação *getProdutoCodigo* de *wsEstoque*. Isso pois para efetuar uma compra em *wsVenda*, é necessário que a requisição seja por meio de uma especificação de códigos que o fornecedor conheça.
- **Linha 11:** Invoca a operação *getProdutoValor* de *wsOrdem*, para a criação da ordem de compra. O valor que o fornecedor, cadastrado no sistema de estoque, vende.
- **Linha 12:** adiciona o produto na ordem de venda, invocando o serviço web *setProduto* em *wsVenda*.
- **Linha 13:** Invoca o método *situacaoOrdem* para montar uma mensagem para o usuário da situação da ordem atual.
- **Linha 14:** Se retornou um resultado na **Linha 13**, então:
- **Linha 15:** Invoca o método *send* de *wsReport*, para enviar uma nova tarefa executada ao sistema de geração de relatório. Neste caso, a mensagem é do tipo *private*, informado em *mensagemEstado*.
- **Linha 16:** Invoca *toXML* do serviço web *kqml*, para criar uma mensagem estilo KQML/XML. Armazena o resultado em *aux*.
- **Linha 17:** Utiliza a operação *send* de *wsISAP* para enviar uma mensagem para o usuário, informando-o da operação realizada. A mensagem enviada está armazenada em *aux* e no formato KQML/XML.

O envio de mensagens ao usuário é utilizado aqui para que os testes sejam avaliados passo a passo do que acontece. Em uma situação real, muitas mensagens não são interessantes que o usuário receba. Provavelmente apenas o resultado da operação, e quando necessário, uma mensagem para confirmação/cancelamento de certa operação que foi modificada ou que requer interação. De qualquer forma, as mensagens que o usuário deve receber são opção dele.

4.3.7.7 Comportamento *Compra*

O comportamento *Compra* (Figura 70) efetua a finalização de uma compra. As informações utilizadas para este comportamento são as informações provindas da ordem de compra e da ordem de venda. Inclusive, se uma ordem de venda for modificada, é enviada uma mensagem para o usuário, requisitando que o mesmo confirme, caso concorde com a modificação feita pelo fornecedor, ou cancele uma ordem, caso contrário.

```
0.  INÍCIO
1.      ordemID=(wsOrdem.php). proxima ( estoqueCNPJ, estoqueSenha, ordemID)
2.  |   SE (ordemID != 0) ENTÃO
3.  |   |   ordemIDVenda=(wsOrdem.php). getIdVenda ( estoqueCNPJ, estoqueSenha, ordemID)
4.  |   |   ordemEstado=(wsVenda.php). estadoOrdem ( ordemIDVenda)
5.  |   |   SE (ordemEstado contem aceita) ENTÃO
6.  |   |   |   ordemValor=(wsVenda.php). valorOrdem ( ordemIDVenda)
7.  |   |   |   ccCodigoPagamento=(wsVenda.php). codigoDePagamento ( ordemIDVenda)
8.  |   |   |   ccResultado=(wsCC.php). pay ( ccCodigoPagamento, ccBandeira, ccNumero, ccName, ccVerificacao, ordemValor, ccParcelas)
9.  |   |   |   SE (ccResultado == 1) ENTÃO
10. |   |   |   |   ordemResultado=(wsOrdem.php). processa ( estoqueCNPJ, estoqueSenha, ordemID)
11. |   |   |   |   SE (ordemResultado != 0) ENTÃO
12. |   |   |   |   |   aux=(wsReport.php). send ( usuario, senha, mensagemTitulo, ordemResultado, mensagemEstado)
13. |   |   |   |   |   aux=(kqml.php). toXML ( null, null, comportamentoNome, mestre, null, null, null, mensagemEstado, ordemResultado)
14. |   |   |   |   |   aux=(wsISAP.php). send ( usuario, senha, aux)
15. |   |   |   |   FIM SE
16. |   |   |   FIM SE
17. |   |   FIM SE
18. |   |   SE (ordemEstado contem modificada) ENTÃO
19. |   |   |   ordemEstoqueEstado=(wsOrdem.php). getEstado ( estoqueCNPJ, estoqueSenha, ordemID)
20. |   |   |   SE (ordemEstoqueEstado contem aberta) ENTÃO
21. |   |   |   |   ordemXML=(wsVenda.php). buscaOrdem ( ordemIDVenda)
22. |   |   |   |   ordemResultado=(wsOrdem.php). alteraOrdem ( estoqueCNPJ, estoqueSenha, ordemID, ordemXML)
23. |   |   |   |   SE (ordemResultado != 0) ENTÃO
24. |   |   |   |   |   aux = confirmar ou cancelar?
25. |   |   |   |   |   ordemResultado=(string.php). concat ( ordemResultado, aux)
26. |   |   |   |   |   aux=(kqml.php). toXML ( null, null, comportamentoNome, mestre, null, null, null, mensagemEstado, ordemResultado)
27. |   |   |   |   |   aux=(wsISAP.php). send ( usuario, senha, aux)
28. |   |   |   |   FIM SE
29. |   |   |   FIM SE
30. |   |   FIM SE
31. |   FIM SE
    FIM
```

Figura 70: Comportamento Compra.

A maneira como o comportamento funciona segue conforme descrito abaixo:

- **Linha 1:** A informação *ordemID* recebe o resultado da invocação da operação *próxima* do serviço web *wsOrdem*. Se

existir alguma ordem em aberto, então retorna o *id* da ordem, caso contrário, retorna 0.

- **Linha 2:** Se *ordemID* for diferente de 0, então continua.
- **Linha 3:** Invoca a operação *getIdVenda* do serviço web *wsOrdem*, para retornar o *id* da ordem de venda no fornecedor, correspondente com essa ordem de compra.
- **Linha 4:** Retorna o estado da ordem no fornecedor, invocando a operação *estadoOrdem* no serviço web *wsVenda*.
- **Linha 5:** Se o estado da ordem (informação *ordemEstado*) contiver a cadeia de caracteres "**aceita**", então segue.
- **Linha 6:** Invoca a operação *valorOrdem* de *wsVenda*, para saber o valor total a ser pago.
- **Linha 7:** Invoca a operação *codigoDePagamento* de *wsVenda* para saber qual o código de pagamento no sistema de cartão de crédito *wsCC*.
- **Linha 8:** Efetua o pagamento no sistema de cartão de crédito invocando a operação *pay* do serviço web *wsCC*.
- **Linha 9:** Se o pagamento foi efetuado com sucesso, ou seja, a informação *ccResultado* contiver o valor 1, então segue.
- **Linha 10:** Invoca a operação *processa* em *wsOrdem*. Atualizando a ordem e o banco de dados no sistema de Estoque.
- **Linha 11:** Se a ordem foi processada com sucesso, então segue.
- **Linha 12:** A operação *send* do serviço web *wsReport* é invocada para enviar informações sobre a atividade executada. Essas informações são provindas da operação *processa* do serviço web *wsOrdem*.
- **Linha 13:** Gera uma mensagem no formato KQML/XML com o resultado da atividade.
- **Linha 14:** Envia para o ISAP, que deve contatar o usuário, para repassar as informações da atividade realizada.
- **Linha 18:** Se *ordemEstado* contiver a *string* "**modificada**", então segue esse bloco. Neste caso, o bloco da **Linha 5**, até a **Linha 17**, não foi executado.
- **Linha 19:** Verifica na ordem de compra, no estoque, se a ordem está aberta. Se não estiver aberta, essa verificação já foi feita, uma mensagem já foi enviada ao usuário e deve ser aguardado o seu retorno.
- **Linha 20:** Se o estado contiver "**aberta**", então segue.

- **Linha 21:** Invoca a operação *buscaOrdem* no serviço web *wsVenda*. Isso armazena em *ordemXML* uma estrutura XML contendo as informações da ordem de venda.
- **Linha 22:** Invoca a operação *alteraOrdem* em *wsOrdem*, informando as alterações feitas no sistema de estoque do usuário, nas informações da ordem de compra. A informação *ordemResultado* recebe um texto sobre o que foi alterado.
- **Linha 23:** Se retornou algo da operação anterior, então segue.
- **Linha 24:** Concatena as informações de *ordemResultado* com a pergunta "**confirmar ou cancelar?**". Armazena o texto em *aux*.
- **Linha 25:** Gera uma mensagem KQML/XML com as informações da alteração.
- **Linha 26:** Envia para o usuário, via operação *send* do serviço web *wsISAP*, as informações da alteração, requisitando que o usuário confirme ou altere as alterações efetuadas pelo fornecedor.

Essa confirmação de alteração, também é opcional. Um usuário especializado, pode modificar essa funcionalidade de forma a deixar o sistema totalmente automatizado, sem a necessidade da intervenção do usuário no comportamento.

4.3.7.8 Comportamento *Compra-altera*

O comportamento *Compra-altera* (Figura 71) é utilizado apenas quando uma ordem foi alterada pelo fornecedor, e o assistente pessoal requisita, para o seu usuário, a confirmação ou cancelamento da operação. A requisição de confirmação/cancelamento é feita no comportamento *Compra*, mais especificamente quando o serviço web *wsVenda* informa, por meio da operação *ordemEstado*, retornando a informação "**modificada**".

Neste caso, este comportamento considera que o usuário já recebeu a requisição de confirmação e se encontra aguardando o resultado. O comportamento considera o caso de que várias ordens podem estar nesse estado. Dessa forma, o usuário deve enformar o código da ordem da qual ele deseja responder ao comportamento.

```
0.  INÍCIO
1.  |  modificou = 0
2.  |  aux=(wsmbox.php). has ( usuario, senha, comportamentoNome)
3.  |  SE (aux != 0) ENTÃO
4.  |  |  mensagemKQML=(wsmbox.php). get ( usuario, senha, comportamentoNome)
5.  |  |  mensagemConteudo=(kqml.php). getKQMLContent ( mensagemKQML)
6.  |  |  aux = ordem
7.  |  |  ordemID=(string.php). nextWord ( mensagemConteudo, aux)
8.  |  |  vendaID=(wsOrdem.php). getIdVenda ( estoqueCNPJ, estoqueSenha, ordemID)
9.  |  |  SE (mensagemConteudo contem confirma) ENTÃO
10. |  |  |  aux=(wsVenda.php). confirmaModificar ( vendaID)
11. |  |  |  aux = aberta
12. |  |  |  aux=(wsOrdem.php). setEstado ( estoqueCNPJ, estoqueSenha, ordemID, aux)
13. |  |  |  SE (aux == 1) ENTÃO
14. |  |  |  |  modificou = 1
15. |  |  |  FIM SE
16. |  |  FIM SE
17. |  |  SE (mensagemConteudo contem cancela) ENTÃO
18. |  |  |  aux=(wsVenda.php). cancelaOrdem ( vendaID)
19. |  |  |  aux=(wsOrdem.php). cancela ( estoqueCNPJ, estoqueSenha, ordemID)
20. |  |  |  SE (aux == 1) ENTÃO
21. |  |  |  |  modificou = 1
22. |  |  |  FIM SE
23. |  |  FIM SE
24. |  |  SE (modificou == 1) ENTÃO
25. |  |  |  mensagemConteudo=(wsOrdem.php). situacaoOrdem ( estoqueCNPJ, estoqueSenha, ordemID)
26. |  |  |  aux=(wsReport.php). send ( usuario, senha, mensagemTitulo, mensagemConteudo, mensagemEstado)
27. |  |  |  aux=(kqml.php). toXML ( null, null, comportamentoNome, mestre, null, null, null, mensagemEstado, mensagemConteudo)
28. |  |  |  aux=(wsISAP.php). send ( usuario, senha, aux)
29. |  |  |  modificou = 0
30. |  |  FIM SE
31. |  FIM SE
    FIM
```

Figura 71: Comportamento Compra-altera.

A forma como o comportamento é executado é descrito a seguir:

- **Linha 1:** A informação *modificou* recebe valor 0.
- **Linha 2:** A operação *has* do serviço web *wsmbox* é invocada. O nome do comportamento, no caso *Compra-altera*, é passado como parâmetro, juntamente com o login e a senha do usuário no serviço de caixa de mensagens. Isso pois é necessário identificar o destinatário da mensagem, ou seja, o usuário deve enviar uma mensagem exatamente para esse comportamento, com a tag **$to** seguido do nome do comportamento.
- **Linha 3:** Se há alguma mensagem para o comportamento em questão, então *aux* terá valor 1, senão 0. No caso, se for 1, então continua a execução do bloco.

- **Linha 4:** A informação *mensagemKQML* recebe o resultado da invocação da operação *get* do serviço web *wsmbox*, ou seja, a mensagem no formato KQML/XML.
- **Linha 5:** A informação *mensagemConteudo* recebe o conteúdo da mensagem, ou seja, o atributo *content* da estrutura KQML.
- **Linha 6:** Para a informação *aux* é atribuído o valor "**ordem**".
- **Linha 7:** A operação *nextWord* do serviço web *string* é invocado de forma a pegar a próxima palavra que vêm depois de *aux*, no caso "**ordem**". Supõe-se aqui que o usuário vai informar o código da ordem enviando a mensagem com a *tag* **ordem** seguido do código. Por exemplo: "... **ordem 54** ...". O resultado, é armazenado em *ordemID*. No caso desse exemplo, o seria o valor **54**.
- **Linha 8:** A informação *vendaID* recebe o *id* da venda configurado na ordem de compra do sistema de estoque. No caso, a ordem de compra da qual o usuário informou o *id* via mensagem para seu assistente pessoal.
- **Linha 9:** Se *mensagemConteudo* contiver a cadeia de caracteres "**confirma**", então segue.
- **Linha 10:** Invoca a operação *confirmaModificar* do serviço web *wsVenda*. Isso faz com que o sistema automático de venda retorne a analisar a ordem em questão, colocando-a novamente com o estado "**aberta**".
- **Linha 11:** Atribui o valor "**aberta**" para *aux.*
- **Linha 12:** Altera o estado da ordem de compra invocando a operação *setEstado* do serviço web *wsOrdem*, do lado do sistema de controle de estoque. No caso, o novo estado é "**aberta**", voltando a estar na lista de ordens a serem analisadas no comportamento *Compra*.
- **Linha 13:** Se a operação da **Linha 12** retornou sucesso, então segue.
- **Linha 14:** Configura a informação *modificou* como sendo **1**.
- **Linha 17:** Se *mensagemConteudo* possui a cadeia de caracteres "**cancela**", então segue. Neste caso, o bloco entre as linhas 9 e 16 não foi executado.
- **Linha 18:** Invoca a operação *cancelaOrdem* do serviço web *wsVenda*, informando ao fornecedor que a ordem foi cancelada pelo cliente.

- **Linha 19:** Invoca a operação *cancela* do serviço web *wsOrdem*, alterando o estado da ordem no estoque, informando que a ordem agora se encontra cancelada.
- **Linha 20:** Se a operação realizada na **Linha 20** foi realizada com sucesso, então executa a **Linha 22**, ou seja, informa que *modificou* é igual a **1**.
- **Linha 24:** Se *modificou* é igual a **1**, então segue.
- **Linha 25:** Retorna em *mensagemConteudo* a informação sobre a situação da ordem, por meio da operação *situacaoOrdem* do serviço web *wsOrdem*.
- **Linha 26:** Envia a tarefa realizada, ou seja, a situação da ordem, para o sistema de gerenciamento de relatórios, invocando a operação *send* de *wsReport*.
- **Linha 27:** Monta uma mensagem KQML/XML com as informações, para o usuário, invocando a operação *toXML* do serviço web *kqml*.
- **Linha 28:** Envia para o usuário, via ISAP, as informações da atividade realizada, via invocação da operação *send* do serviço web *wsISAP*.
- **Linha 29:** Armazena na informação *modificou* o valor 0. Preparando o comportamento para uma nova execução.

Um exemplo de mensagem de confirmação/cancelamento para este comportamento pode ser da seguinte forma: "*$to compra-ordem confirmar ordem 54*". Nesta mensagem é identificado o comportamento *Compra-ordem* como destinatário da mensagem, a palavra **confirmar** também é identificada como informação para confirmar a atividade e **ordem 54** é identificado o *id* da ordem da qual se quer confirmar as alterações.

Um produto pode possuir vários fornecedores. Quando uma ordem é cancelada, o fornecedor selecionado para enviar a ordem é modificado, pegando o próximo fornecedor.

5 ANÁLISE GERAL DA PROPOSTA E DA SUA INSTANCIAÇÃO

Com o objetivo de analisar a proposta de Arquitetura de Referência para Softwares Assistentes pessoais, este capítulo apresenta os três procedimentos gerais metodológicos aplicados, cada qual com sua importância específica:

1. A partir da execução de uma instância de implementação criada a partir do modelo de referência e a Arquitetura de Referência, foram efetuados testes baseados em um estudo de caso selecionado e foram extraídos dados da sua execução com o intuito de apresentar sua aplicabilidade e cumprimento dos requisitos destacados nos capítulos anteriores.
2. Uma fase de entrevistas realizada com pessoas sobre os resultados. Para isso foram efetuadas duas apresentações da proposta, com dois enfoques e para dois grupos distintos de pessoas.
    - Uma apresentação foi direcionada a pessoas que se enquadrassem como usuários, desde pessoas não especializadas em TI como pessoas especializadas e desenvolvedores.
    - Uma outra apresentação foi específica a pessoas que poderiam se enquadrar como desenvolvedores e fornecedores de serviços que se enquadrassem na proposta. Este último grupo foi de especialistas em TI e desenvolvedores de software.
3. Publicação de artigos em eventos científicos para avaliação da proposta durante todo o percurso do trabalho. Este processo foi de essencial importância para a evolução do trabalho, com sugestões, questões não bem definidas e críticas para redefinições dos rumos do trabalho durante todo o seu ciclo de definição, modelagem e projeto.

5.1 VERIFICAÇÃO

Tomando como base o estudo de caso para implementação definido na seção 4.1, nessa seção é feita uma verificação de corretude da execução do protótipo e das trocas de mensagens entre os módulos.

5.1.1 Estado inicial

No sistema de controle de estoque é cadastrado um produto (Figura 72), de código “101”, descrição “mouse”, quantidade de 20 elementos. São definidas também as informações sobre a quantidade mínima deste produto no estoque, ou seja, quando a quantidade do produto em questão estiver a baixo dessa quantidade, um processo de compra deve ser iniciado. A informação sobre a quantidade máxima refere-se a quantidade de produtos máxima que deve estar no estoque. Também são cadastrados os fornecedores, e o preço que cada um fornece determinado produto.

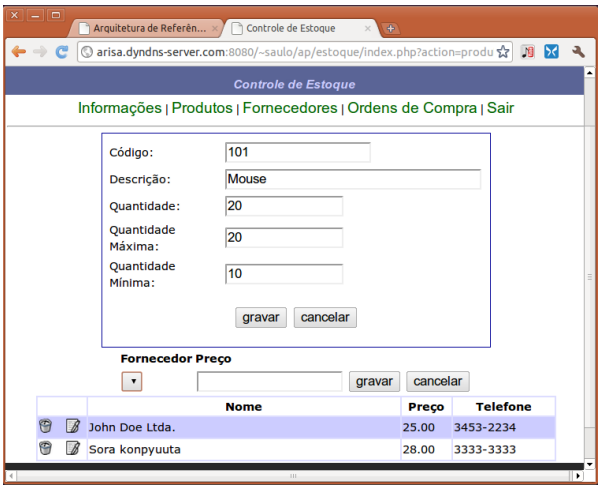

Figura 72: Produto cadastrado no Controle de Estoque.

Os fornecedores cadastrados que fornecem o produto são: (i) John Doe Ltda que fornece o produto a 25 reais a unidade e (ii) Sora Konpyuuta com o produto a 28 reais a unidade. Os nomes “John Doe Ltda” e “Sora Konpyuuta” aqui citados são nomes fictícios utilizados apenas para exemplificação dos comportamentos.

5.1.2 Criação da Ordem de Compra

Para fins de testes e verificação do funcionamento do sistema, é considerado o caso de que o sistema possuía o cadastro de 20 unidades do produto 101 e 17 foram vendidas. Neste caso, como a quantidade mínima definida é 10, e a quantidade máxima 20, o estoque do produto passa a conter apenas 3 unidades do produto 101. Neste caso o sistema deve efetuar compra do produto para que este fique na quantidade máxima, ou pelo menos fique com mais produtos do que está informado em "quantidade mínima".

O primeiro comportamento para o processo de compra a entrar em ação é o "Compra-ordem" (Figura 69). Conforme a seção 9.1 dos Apêndices, no momento "21:25:52", a linha de execução 1 do comportamento é executada e o produto 5 é retornado como sendo um produto com estoque baixo. Este código 5 é um código para controle interno do sistema de controle de estoque. Este é independente do código do produto, já visto aqui como "101". O sistema busca a quantidade de produtos necessários para a compra, ou seja, o que falta para completar a quantidade máxima.

Uma nova ordem é gerada no sistema de estoque e um fornecedor automaticamente selecionado para a compra pelo próprio sistema de estoque. Com o CNPJ do fornecedor, o sistema então efetua uma requisição de criação de uma ordem de venda no sistema de fornecedores.

Paralelamente a isso no sistema de vendas, no momento 21:26:19 dos *logs* do sistema de vendas (seção 9.8), o sistema detecta uma nova ordem no sistema, com sua quantidade e preço. Esta nova ordem, no sistema gerenciador de vendas e fornecedores, possui o código identificador 62. No momento 21:26:20 da seção 9.8, é verificado que o fornecedor não possui a quantidade requisitada na ordem e efetua uma modificação de 17 para 10 produtos.

A requisição é executada com sucesso e o código da ordem de venda 62 é retornado. Informações de código e preço do produto são retornados do sistema e o produto é então acrescentado na ordem de venda no fornecedor. Informações sobre a situação da ordem de compra são buscadas e enviadas como mensagem para o sistema gerenciador de relatórios e para o usuário nas linhas 15 (21:26:14h) e 17 (21:26:18) respectivamente.

Inicialmente, na linha 6, momento 21:26:00 do log da seção 9.1,o fornecedor selecionado foi o fornecedor de CNPJ 000.000/000-01 (John Doe Ltda.).

Conforme os *logs* da seção 9.9 do Apêndice, no momento 21:26:18 o servidor *XMPPConnector* recebe uma mensagem do ISAP para enviar para o contato *saulopz@gmail.com*. Essa mensagem é enviada 4 linhas de *debug* depois.

No momento 21:26:19 o usuário recebe a mensagem do seu assistente pessoal avisando do início do processo da compra, ou seja, que a ordem se encontra em aberto (Figura 73).

Figura 73: Troca de mensagens entre o assistente pessoal e o usuário.

5.1.3 Inicio da Compra

Com uma ordem de compra em aberto, o comportamento "Compra" entra em execução. Ao invocar a operação *próxima* do serviço web *wsOrdem.php*, o código dessa nova ordem com estado "aberta" é retornado e o comportamento entra em execução. Isso está descrito nos *logs* na seção 9.2 dos Apêndices, no momento 21:25:57, linha de execução 1 da Figura 70 do comportamento. O código 80 da respectiva ordem de compra é retornado.

Na linha 4, momento 21:26:30, é detectado que a ordem se encontra no estado “modificada”. Ou seja, o fornecedor modificou algo na ordem e para que ela seja completada, é necessário que o comprador aceite tais modificações. Isso leva a execução do comportamento a aceitar a linha 18, momento 21:26:33 da seção 9.2 dos Apêndices. Ou seja, se a ordem encontra-se no estado “modificada” executa o bloco de instruções subsequente.

O comportamento altera as informações da ordem de compra no sistema de controle de estoque em conformidade com as modificações efetuadas pelo fornecedor e envia uma mensagem ao usuário requisitando confirmação ou cancelamento dessa ordem de compra na linha 27, momento 21:26:47 dos *logs* da seção 9.2 dos Apêndices. O XMPPConnector envia a mensagem ao contato *saulopz@gmail.com* as 21:26:49 (seção 9.9), que recebe a mensagem do seu assistente pessoal (Figura 73) com informações da ordem alterada.

Como não há nenhuma ordem com estado “aberta”, a linha 1 do comportamento “Compra” passa a retornar falso, não seguindo a execução do resto do comportamento.

5.1.4 Cancelamento da Ordem de Compra

No momento 21:27:17 o usuário “Saulo” envia uma mensagem ao assistente pessoal, mais especificamente ao comportamento “Compra-Altera”, cancelando a ordem de compra.

As 21:27:19, o XMPPConnector (seção 9.9) recebe a mensagem do contrato *saulopz@gmail.com*. As 21:27:19 (seção 9.7) o ISAP recebe a mensagem e envia para a operação *send* do serviço web *chatter.php*, retornando a mensagem “*Ok! Vou processar essa informação.*” e enviando ao contato, via XMPPConnector, que a recebe as 21:27:19 (Figura 73).

No momento 21:27:20 o comportamento *mbox* (*log* da seção 9.4) recebe a mensagem vinda do ISAP, identificando para qual comportamento a mesma se destina e a armazena no serviço web *wsmbox.php*.

O comportamento *Compra-altera* verifica que há uma mensagem de *wsmbox.php* no momento 21:27:32 (*logs,* seção 9.3 dos apêndices), retornando seu conteúdo as 21:27:35. Como a mensagem contém em seu conteúdo a palavra “cancela”, a condição da linha 17 é satisfeita as 21:27:45. Ambas as ordens, de venda no sistema do fornecedor (Figura

74) e de compra no sistema de estoque, são canceladas as 21:27:46 e 21:27:48 respectivamente e uma mensagem é enviada ao sistema de gerenciamento de relatórios e ao usuário. No fornecedor, o código da ordem é próprio e diferente da ordem de compra no cliente.

Figura 74: Ordem cancelada no fornecedor John Doe Ltda.

Uma mensagem é enviada ao sistema de controle de relatórios e ao usuário 21:28:04, Figura 73. Esse comportamento passa então a não ser mais executado até que outra mensagem seja recebida do serviço web *wsmbox.php*.

5.1.5 Criação de Nova Ordem de Compra

Como a ordem de compra de código 80, no cliente foi cancelada, o produto de código 101 ainda se encontra com estoque baixo. Uma nova ordem de compra é assim instanciada. Contudo, quando uma ordem de compra é cancelada no *wsOrdem.php*, o sistema se encarrega de tornar ativo outro fornecedor. No caso, o fornecedor a se tornar ativo agora é o Sora Konpyuuta (momento 21:27:57 da seção 9.1). Entre os momentos 21:28:12 e 21:28:15 da mesma seção, uma mensagem é enviada ao sistema gerenciador de relatórios e uma ao usuário, informando que a nova ordem de compra se encontra aberta.

Paralelamente, no sistema de vendas, momento 21:28:20 da seção 9.8, uma nova ordem, de código 62 no fornecedor, se encontra em aberto e inicia sua verificação. Como o novo fornecedor possui todas as unidades requisitadas, nada é modificado e o estado da ordem é modificado para “aceita”.

5.1.6 Fechamento da Compra

Agora, com uma nova ordem na lista de ordens sendo processada (linha 1 da Figura 70, momento 21:29:09 da seção 9.2), o comportamento de compra pode entrar novamente em atividade para processar a ordem de código 81. Nesse caso, o fornecedor aceitou a ordem sem alterações, retornando como “aceita” no momento 21:29:14 da seção 9.2, linha 4 da Figura 70.

Dessa forma, a linha 5 é satisfeita e o processo de pagamento para fechamento da compra inicia. Na linha 6, momento 21:29:17, o valor R$ 476,00 do valor total da ordem é retornado. Na linha 7, momento 21:29:19, o código de pagamento (código específico no sistema de pagamento via cartão de crédito) é retornado do sistema de vendas.

Como não existe ainda um código de pagamento no sistema de vendas, o comportamento termina e esse comportamento deve tentar novamente finalizar o processo.

O código de pagamento, buscado do serviço web *wscc.php* é apenas criado no momento 21:29:20 da seção 9.8, com código 71. No momento 21:29:41 da seção 9.2, o código de pagamento 71 é então retornado para o comportamento de compra no assistente pessoal, podendo agora o sistema efetuar o pagamento.

Na linha 8, momento 21:29:43, o pagamento é realizado. Na linha 10, momento 21:29:46, a ordem de compra é processada (Figura 75).

Nas linhas 12 à 14, entre os momentos 21:29:49 e 21:29:52, uma mensagem é enviada ao sistema gerenciador de relatórios e outra enviada ao usuário, informando que a ordem 81 foi processada (Figura 73).

Figura 75: Ordens de compra do sistema de controle de estoque.

Ainda, no sistema de vendas, no momento 21:30:21 (seção 9.8) é verificado que o pagamento foi confirmado via serviço web *wscc.php*, e a ordem de venda é finalizada no fornecedor Sora Konpyuuta (Figura 76).

Figura 76: Ordens de venda no fornecedor Sora Konpyuuta.

5.1.7 Geração de Relatórios

Para a geração de relatório, segundo o comportamento *Report* (Figura 67), é necessário que a hora do sistema esteja atualizada. A responsabilidade disso é via comportamento *AtualizaHora* (Figura 63) e verificado nos logs do comportamento do *AtualizaHora* na seção 9.6.

Para este teste, o horário do relatório foi configurado para 21:55h. Na linha 2 (Figura 67), momento 21:55:02 da seção 9.5, é detectado um relatório privado. Este é enviado ao e-mail do usuário do assistente pessoal no momento 21:55:14 da seção 9.5, linha 7 da Figura 67.

O e-mail recebido pelo usuário (Figura 77) contém um parágrafo introdutório gerado automaticamente pelo sistema de geração de relatórios e a descrição das atividades realizadas pelo assistente pessoal. É importante salientar que essas atividades devem ser devidamente registradas durante os comportamentos para que possam aparecer no relatório.

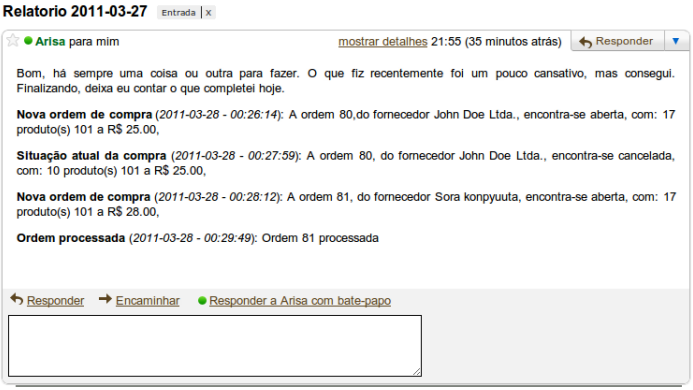

Figura 77: Relatório por e-mail.

Outra forma de relatório, é via envio de *post* em um blog, conforme o exemplo visualizado na Figura 78. Para que um relatório via blog seja gerado, é necessário que o sistema de relatórios seja informado de atividades do tipo *public*.

Para um relatório público, via blog, essas atividades públicas são reunidas e gerado um relatório que pode então ser enviado para o blog do assistente pessoal.

No caso, do exemplo da Figura 78, como as atividades necessitam ser públicas para que o relatório também seja público, então a interação com o usuário também foi pública.

Figura 78: Relatório de atividades públicas via blog.

Neste caso, o ISAP interpretou que as trocas de mensagens com o usuário são públicas e selecionou a *timeline* do Twitter como meio de interação, conforme Figura 79 e Figura 80.

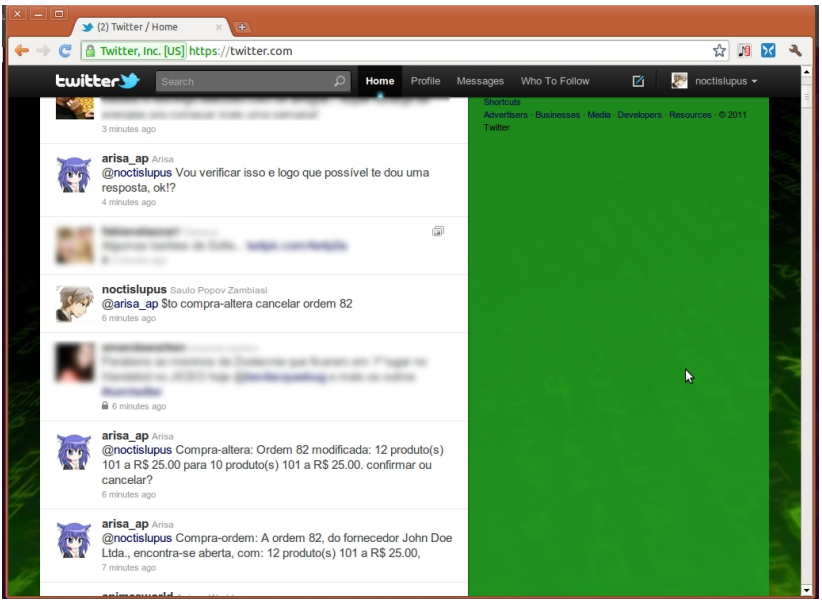

Figura 79: Interação pública com o usuário via Twitter (A).

Na Figura 79, o assistente pessoal informa da nova ordem criada e logo após informa o usuário da modificação. O usuário responde para cancelar a ordem e o assistente pessoal retorna com um *feedback*. Todo esse processo se dá da mesma forma que a conversa com o usuário via chat na Figura 73, contudo agora em um meio público de troca de mensagens.

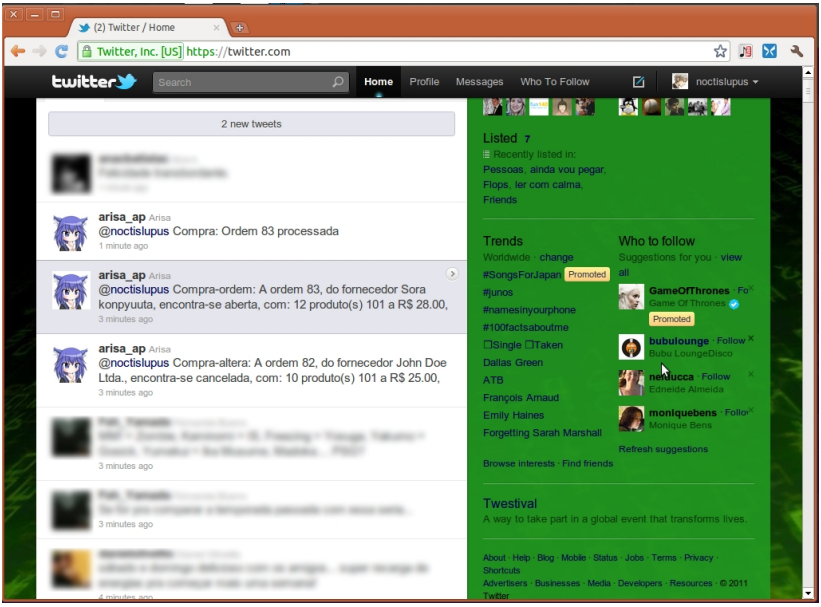

Figura 80: Interação pública com o usuário via Twitter (B).

Na sequência (Figura 80), por fim, o assistente pessoal informa do cancelamento da ordem de código 82, da abertura de uma nova ordem, de código 83 e de sua conclusão.

Outra forma de interação com o usuário, é via celular. Quando a atividade é privada e o usuário não se encontra conectado no gtalk, o ISAP se encarrega de entregar a mensagem na forma de mensagem de texto no celular (Figura 81). Isso é possível devido ao envio de *DirectMessages* via Twitter.

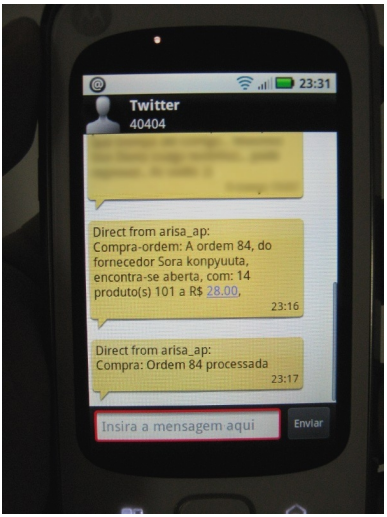

Figura 81: Comunicação com o assistente pessoal via celular.

Ainda quanto a essa execução, é importante que alguns pontos sejam observados. Primeiramente, pode não ser interessante que o usuário fique recebendo informações do assistente pessoal a todo momento, ou mesmo, o usuário pode querer que esse cancelamento de uma ordem de compra, para a geração de uma nova para outro fornecedor, seja automático, sem a necessidade da sua intervenção. Contudo, neste exemplo várias mensagens são trocadas com o usuário para fins de testes e verificação do funcionamento do comportamento.

5.2 APLICAÇÃO DE QUESTIONÁRIOS

Segundo Günther (2003), efetuar perguntas às pessoas sobre o que pensam sobre determinado assunto é uma forma de conduzir estudos empíricos de observação, experimento e *survey* (avaliação). A qualidade dos dados extraídos desse método de estudo pode trazer vantagens para o objeto de estudo de uma pesquisa.

O questionário, que mede a opinião de pessoas sobre determinado tópico, é o instrumento utilizado. Para sua elaboração, é importante analisar o objetivo da pesquisa e qual é a população-alvo (Günther, 2003). Por meio de perguntas, entrelaçadas aos objetivos específicos do

trabalho, este meio se caracteriza como sendo uma das mais rápidas e eficientes formas de coletar dados (Gil, 2010).

Para este trabalho foi selecionado um tipo de resposta para os questionários baseado na escala de Likert (1932). Nessa escala, mede-se o nível de concordância ou não à afirmação. Usualmente são utilizados cinco níveis de respostas (Não concordo fortemente, não concordo parcialmente, indiferente, concordo parcialmente e não concordo fortemente). No presente trabalho ainda foi acrescentado a resposta "Não tenho condições de responder / não sei opinar" para o caso da questão não estar no nível de conhecimento do usuário.

Nos questionários ainda há uma última pergunta, não baseada nessas respostas, mas na forma de um espaço para críticas, sugestões, e comentários, em especial para as respostas “discordo” às perguntas.

A aplicação do questionário possui como objetivo comprovar que os objetivos gerais e específicos do trabalho foram atingidos e, assim, a comprovação da hipótese.

Antes de se apresentar cada um dos questionários aplicados, é importante ressaltar que um total de 61 pessoas que responderam os questionários. Para título de interpretação da última questão do questionário, na forma de resposta discursiva, os usuários estão identificados por numéricos com o sinal “#” precedendo cada número. Também é importante salientar que o perfil de usuários sem conhecimento de TI são os usuários identificados de #01 até #23, os usuários com conhecimento de TI são os usuários de #24 até #42, e as pessoas com conhecimento de TI e/ou desenvolvedores estão identificados com as *tags* #43 à #61.

Também é importante salientar que que não foi viável selecionar pessoas para utilizarem o software (ou conjunto de softwares quem compõe o assistente pessoal) diretamente para testes. Dessa forma, os questionários não servem diretamente como instrumento de comprovação científica / instrumento concreto, mas serve de auxílio de análise subjetiva do assunto.

5.2.1 Aplicação do Questionário a Usuários

A aplicação do questionário para usuários foi efetuada após uma apresentação da proposta, direcionada para pessoas que utilizariam implementações geradas a partir da proposta de arquitetura de referência

para softwares assistentes pessoais. Essas pessoas poderiam possuir assistentes pessoais e utilizar-se de suas funcionalidades.

Entre as pessoas que assistiram a essa primeira apresentação e responderam o questionário com as questões mostradas no Quadro 2, se encontravam 23 pessoas não especialistas na área de TI (que não trabalham na área e possuem apenas conhecimentos como usuários de computador e softwares) e 19 pessoas com conhecimentos de TI e/ou desenvolvedores (mas no papel de usuários de assistentes pessoais), somando 42 pessoas. As perguntas dos questionários são de caráter geral, ao aspecto de SAP, de caráter específico ao que foi desenvolvido e de caráter mais prospectivo, uma vez que também se trata de um trabalho potencialmente exploratório.

1. Na sua opinião, a proposta de arquitetura de referência tem relevância para a criação de implementações de Softwares Assistentes Pessoais?
2. Apesar de ser um protótipo, você acha que a interface de configuração de comportamentos é adequada para ser utilizada?
3. Você acredita que a utilização de Softwares Assistentes Pessoais pode auxiliar efetivamente os usuários nas tarefas diárias na empresa/trabalho?
4. Você acredita que a utilização de Softwares Assistentes Pessoais pode diminuir efetivamente o tempo de uma pessoa dispendido em suas tarefas rotineiras na empresa?
5. Você acredita que a distribuição/terceirização de comportamentos do SAP, podendo ser fornecido por terceiros, pode trazer benefícios para a criação de Softwares Assistentes Pessoais mais adequados para diversos perfis de usuários e empresas?
6. Se não existisse um Software Assistente Pessoal gratuito para auxiliar em suas atividades, você cogitaria pagar pelas funcionalidades de um Software Assistente Pessoal adequado às suas necessidades?
7. Você acredita que é viável um Software Assistente Pessoal dessa natureza ser utilizado em cenários reais?
8. Na sua opinião, a proposta apresentada pode ser considerada inovadora?
9. Você acredita que com a proposta apresentada, é flexível para adaptar Softwares Assistentes Pessoais às mudanças de interesses e objetivos dos usuários?
10. Você acredita que a proposta apresenta uma forma suficiente para a criação de Softwares Assistentes Pessoais que possam fornecer aos usuários recursos para a sua utilização e customização em conformidade com as necessidades dos usuários e com os processos de negócios na empresa em que estão inseridos?
11. Por favor, utilize o espaço a seguir para fazer comentários, sugestões, críticas, e eventualmente propostas de trabalhos futuros. Em especial para casos de respostas "discordo" às perguntas.

Quadro 2: Perguntas do questionário aplicado à pessoas com perfil de usuários.

Na apresentação foram apresentados outras propostas/produtos, vistos na seção 2.1.1, que fornecem assistência pessoal as pessoas como forma de fazer um paralelo com a proposta desse trabalho; foi apresentada a proposta e suas possibilidades, o protótipo implementado, suas possibilidades e a execução do estudo de caso também implementado.

Após concluída a apresentação foi aplicado o questionário às pessoas presentes. Os resultados foram analisados e são mostrados a seguir.

Conforme Gráfico 1, as pessoas responderam, em sua maioria, como concordando com a relevância de assistentes pessoais. Este é um considerável resultado positivo quanto a aceitação da proposta para a criação de implementações de SAPs.

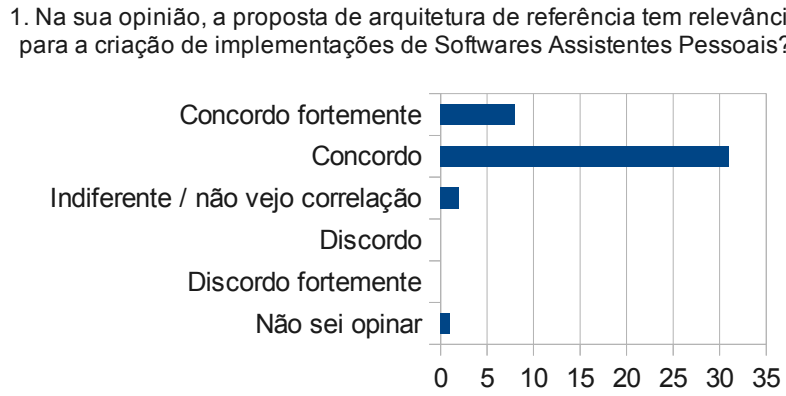

Gráfico 1: Relevância do problema.

O Gráfico 2 apresenta as respostas dos entrevistados quanto a interface de configuração de comportamentos implementada.

Ainda que a interface de configuração sendo caracterizada como um protótipo, ela conseguiu um bom grau de aceitação das pessoas com perfil de usuário. Contudo, segundo o entrevistado #32 "*A formulação das regras (algoritmos) pode ser algo muito complexo para usuários que mal sabem utilizar a internet*".

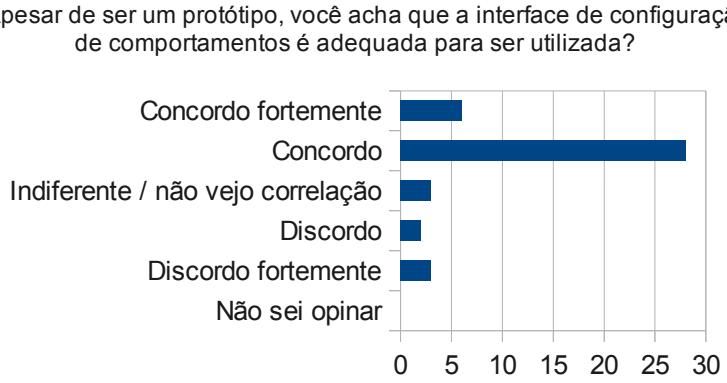

Gráfico 2: Interface de configuração de comportamentos.

Em tempo, com relação à interface, foi citado sobre a maneira de formulação de comportamentos. Para o entrevistado #40:

> "a personalização e design deve ser mais facilitado, retirando a parte lógica para o usuário final, já que muitos mal sabem mandar e-mail." – Entrevistado #40.

Como comentado anteriormente na proposta, a área de configuração de comportamentos baseados em serviços é uma área ainda em franca pesquisa, com diversas tecnologias, e até tratada em diferentes níveis (por exemplo, no nível de modelagem de processos, em ambientes de BPM). Porém, e como igualmente já foi discutido, observa-se que variadas situações de configuração de sistemas ainda requerem certos graus de conhecimento ou familiaridade com TI.

Por outro lado, o entrevistado #38 enfatizou que "*...a interface de usuário poderia ser um pouco mais elaborada, mas se tratando de um protótipo está bom"*, entendendo que tal recurso de interação humano-computador pode ser melhorada para um produto que venha a se tornar apresentável à um ambiente real de mercado, e que como se trata de um protótipo, para se analisar o comportamento/funcionamento em um nível mais geral, não seria realmente o enfoque a melhor forma de interação com o usuário.

É importante salientar que as respostas discursivas respondidas referente a interface foram respondidas apenas por pessoas com conhecimento em TI.

Um entrevistado com conhecimentos básicos de TI, o entrevistado #01, comentou sobre a parte de interface, mas mais especificamente da interação entre o usuário e seu assistente pessoal por meio de gtalk, twitter, e-mail, etc. Em suas palavras ele afirma:

> "gostei da possibilidade de ter um assistente pessoal que possua uma interface 'humanizada'. Isso certamente atrai mais o usuário e facilita a relação com o software". – Entrevistado #1.

Tais recursos de interação com o usuário podem ser caracterizados como formas mais atrativas de relacionamento usuário-assistente. Isso pois o usuário já está acostumado a se relacionar com outras pessoas via esses meios sociais e de interação eletrônica.

O Gráfico 3 está relacionado com o reconhecimento de que assistentes pessoais podem realmente auxiliar as pessoas nas atividades diárias.

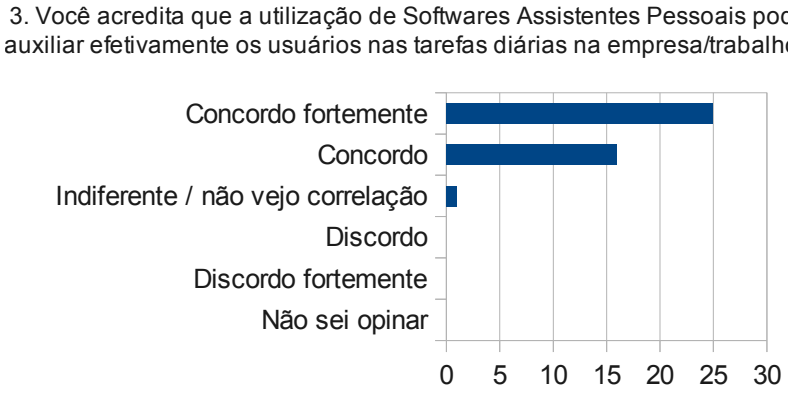

Gráfico 3: Auxílio nas atividades.

Nessa questão, as pessoas mantiveram suas respostas entre "Concordo Fortemente" e "Concordo", identificando como sendo uma grande aceitação por parte de pessoas que se enquadrariam em um perfil de usuário. Para o entrevistado #03, "*tudo o que for feito para nos ajudar é válido*", contudo ele frisa que:

> "Um ponto contra seria que diminuiria o trabalho do funcionário, assim como o número de funcionários, dessa forma gerando um aumento no desemprego. Porém, toda a mudança gera consequências. - Entrevistado #03

Tal premissa é válida, mas não é um ponto a se analisar no presente trabalho. Esta é uma questão bastante polêmica que sempre foi levantada diante da automatização de tarefas que antes eram efetuadas por pessoas, ou animais. No momento atual, é uma tarefa difícil analisar os impactos que a utilização de assistentes pessoais por funcionários de empresas e organizações e pela população em geral pode gerar, tanto a curto, médio e principalmente a logo prazo.

O entrevistado #10 também comenta sobre essa questão, enfatizando que "*um serviço desses seria muito bom para a empresa, mas muito ruim para essas pessoas que vão perder o emprego*". Isso considerando no caso de atividades que possam realmente substituir o funcionário.

Outro ponto a se tomar como base é a resposta dada pelo Entrevistado #05. Para ele, que assinalou "concordo fortemente", a solução de assistência pessoal via software de computador pode ajudar "*tanto quanto pode trazer problemas, se o software parar depois do funcionário já ter 'viciado' nas facilidades que o assistente traria*". Ou seja, as pessoas acabam acostumando-se com as facilidades que a automatização de processos e com recursos de TI e, de forma também problemática, acabam dependendo diretamente desses recursos. Isso acaba por gerar problemas sérios quando na indisponibilidade dessas tecnologias. Uma pessoa cujo as tarefas automatizadas são efetuadas por um assistente não mais tem para si seu assistente disponível, pode acabar por levar muito mais tempo para realizar a tarefa do que uma pessoa que já a realiza costumeiramente, ou até não conseguindo realizar a tarefa sozinho.

Já para o Entrevistado #41, "*para o usuário a proposta é boa, mas não acredito que possa mudar imediatamente os hábitos pessoais*". Isso está relacionado com mudanças. Estas, em geral, levam um tempo para serem abstraídas e por fim se tornarem rotineiras. Para o entrevistado #41, a mudança de hábitos das pessoas, em relação à utilização de um software para auxiliá-las nas atividades diárias, poderia levar um certo tempo para a adaptação.

O Gráfico 4, por sua vez, está relacionado diretamente com a premissa de que com a existência de um software assistente pessoal, o tempo gasto dos usuários desses SAPs na realização das atividades diárias seria diminuído. Com isso, as pessoas poderiam utilizar seu tempo em atividades de maior valor para a empresa e retorno intelectual para a pessoa.

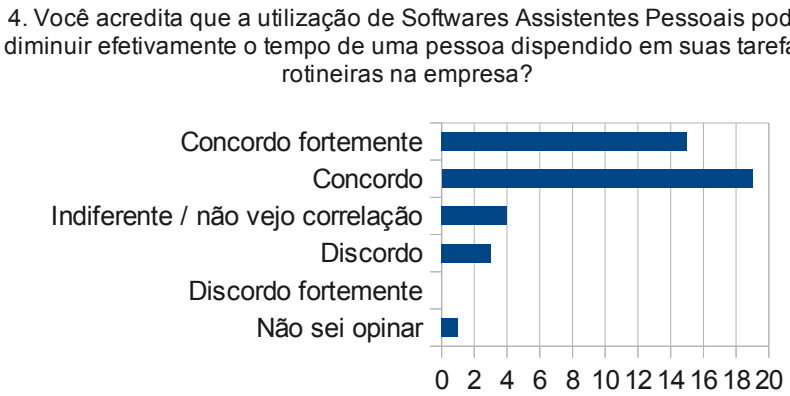

Gráfico 4: Diminuição do tempo gasto nas atividades diárias.

A grande maioria dos entrevistados concorda que tal recurso traria a diminuição do tempo gasto com as atividades diárias, ficando entre "concordo fortemente" e "concordo". Apenas três entrevistados discordaram, mas não justificaram o porque. Já o entrevistado #1 afirma:

> "... gostei da proposta de otimizar o meu tempo, mandando o assistente pessoal realizar tarefas chatas e enfadonhas, que muitas fezes nos fazem perder horas do dia, como: pesquisas na internet, orçamentos, etc." – Entrevistado #1.

O Gráfico 5 está ligado mais a questões arquiteturais da distribuição dos comportamentos na forma de serviços web. Tal questão também pode ser avaliada do ponto de vista da liberdade do usuário poder escolher os comportamentos que melhor se enquadram com suas

atividades/necessidades, independente de desenvolvedor ou "fornecedor de comportamentos".

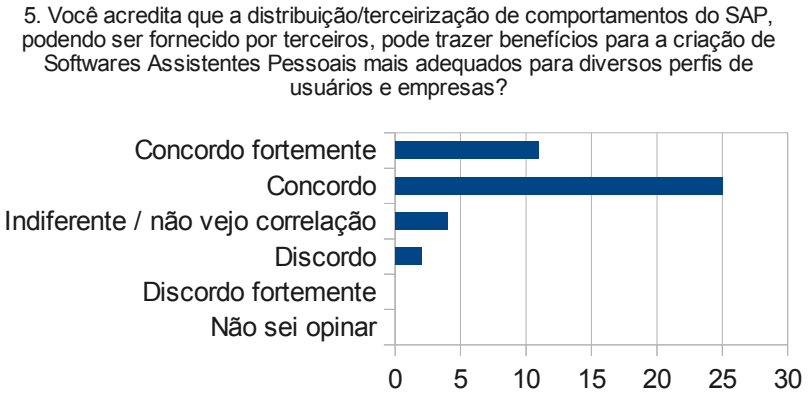

Gráfico 5: Benefícios da terceirização de comportamentos de SAPs.

Ainda que essa questão seja caracterizada mais como uma questão de proposta de distribuição, e talvez até um pouco técnica, apenas dois entrevistados discordaram, sendo entre eles uma pessoa com pouco conhecimento em TI e outra com conhecimentos. Também, nenhuma justificou sua discordância quanto a questão. Sem as justificativas escritas das discordâncias, não dá para se concluir muito sobre isso.

O Gráfico 6 se relaciona a questão de pagamento pela utilização de Softwares Assistentes Pessoais. Nessa premissa, houve um número maior de pessoas discordantes, 13 pessoas, sendo que 3 delas afirmaram que discordam fortemente do pagamento de funcionalidades de softwares assistentes pessoais.

O entrevistado #30 afirma: "*não pagaria por um software de assistente pessoal pois não sou o cliente-alvo do produto*". Nesse mesmo sentido, o entrevistado #32 afirma: "*discordo, pois atualmente não sinto falta de um assistente*".

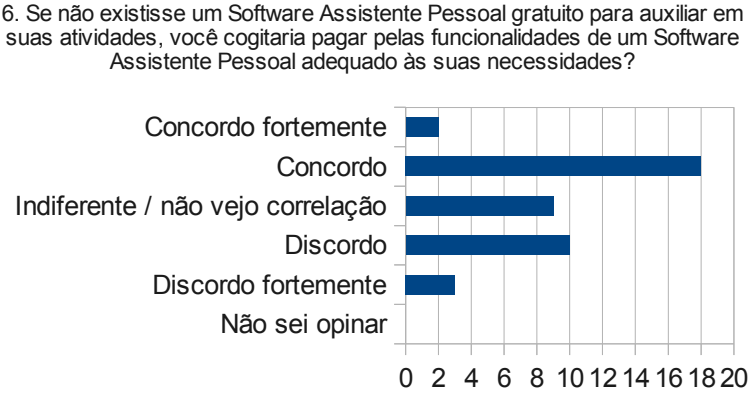

Gráfico 6: Pagamento pela utilização de SAPs.

Já o entrevistado #42 explana um pouco mais sobre tal necessidade e pagamento por essa necessidade:

> "Eu discordo com a questão 6, pois a resposta é muito relativa a necessidade do usuário. Por exemplo, hoje a minha necessidade é mínima, jamais pagaria por um assistente pessoal. Mas se amanhã a minha necessidade for alta, por eu não conseguir equilibrar as minhas atividades, certamente pagaria por um SAP." - Entrevistado #42.

Todas as respostas dadas estão relacionadas diretamente com a necessidade da utilização de um SAP. Para esses entrevistados, tal recurso seria realmente interessante ser pago caso a necessidade fosse presente.

Na verdade, esta questão é um pouco mais ampla, pois depende do modelo de negócios e de disponibilização dos serviços por parte de cada provedor. Conceitualmente, por exemplo, o modelo SaaS (*Software as a Service*) assume o potencial de pagamento por uso e acesso sob demanda. Porém, há inúmeros casos em que serviços são acessados numa ótica SaaS mas sem custo. Há inclusive variações, híbridas, em que uma certa versão do serviço de software é disponibilizada gratuitamente, e uma outra versão, mais sofisticada, é paga.

A questão 7, com suas respostas apresentadas no Gráfico 7, refere-se à aplicação de Softwares Assistentes Pessoais baseados na proposta em cenários reais.

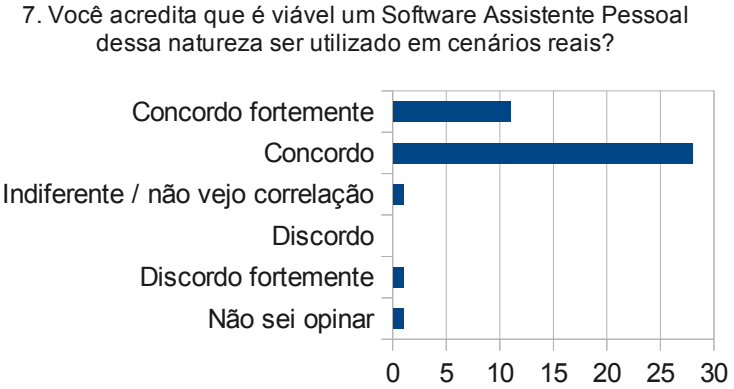

Gráfico 7: Utilização de SAPs em cenários reais.

Dos entrevistados, apenas um assinalou a resposta "Discordo fortemente", mas mesmo assim não a justificou. A maioria colocou como "Concordo" e um número de 11 pessoas, selecionaram "Concordo fortemente". Isso representa uma boa aceitação da ideia.

A questão referente a premissa da proposta ser inovadora ofereceu as respostas apresentadas no Gráfico 8.

Nesta questão, 3 pessoas discordaram, 2 não souberam opinar e 8 consideraram-se indiferente com a questão. Mesmo tendo sido apresentados outros assistentes pessoais e suas formas de trabalho, assim como tendo sido enfatizado na apresentação o diferencial da proposta, tal conjunto de respostas negativas ou indiferentes podem estar ligadas ao desconhecimento do assunto em geral.

Para o entrevistado #7, essa é uma "*ideia inovadora e que deve ser tendência para os próximos anos*". Contudo, o entrevistado #8 discorda, e afirma em suas palavras: "*discordo porque durante a apresentação foram mostrados alguns SAPs. Não ficou claro para mim a parte inovadora*". Contudo, foi dedicado um tempo na apresentação justamente para mostrar a diferença entre a proposta e os outros SAPs apresentados, ou seja, a parte inovadora.

Gráfico 8: Proposta inovadora.

Ainda, uma porcentagem de 69% de concordância, sendo que desses 9 pessoas assinalaram "concordo fortemente" leva a crer que para a maioria das pessoas presentes na palestra, foi entendido o ponto de inovação do trabalho.

O Gráfico 9 se refere às respostas dos entrevistados em relação à flexibilidade e adaptabilidade dos SAPs aos usuários e mudanças de interesses e objetivos dos mesmos.

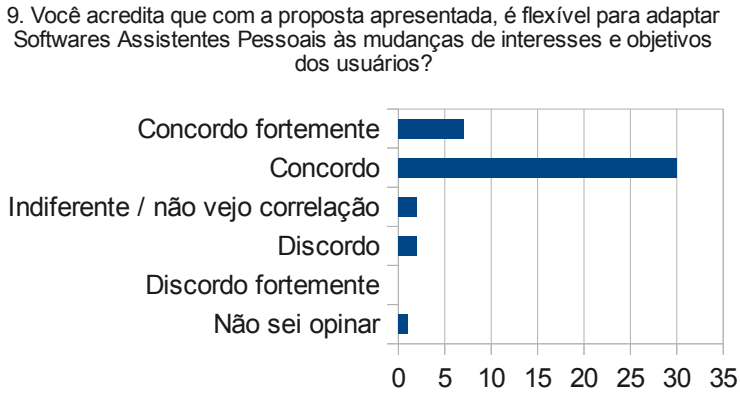

Gráfico 9: Flexibilidade e adaptação dos SAPs aos usuários.

Nesta questão, houve um grande grau de concordância. Entretanto, o entrevistado #42, também classificado como conhecedor de TI, colocou em pauta a seguinte questão:

> "Vejo que não é muito viável reconfiguração de comportamentos, uma vez que é possível incluir comportamentos que possam 'contradizer' outros já existentes. Como identificar isso?" - Entrevistado #42.

Em verdade, tal questão é bastante válida e relevante, também citada nos capítulos anteriores da Tese. Para responder a essa questão, é importante considerar primeiramente que a proposta da arquitetura de referência não é direcionada a resolver conflitos de comportamentos, mas na distribuição e utilização destes por meio de padrões.

A segunda questão a ser tratada é que o gerenciamento de conflitos de comportamentos do presente protótipo é de responsabilidade do usuário, uma vez que foi desenvolvido da forma mais simples possível, com relação à proposta.

Algoritmos de identificação de conflitos são de responsabilidade dos desenvolvedores dos núcleos dos assistentes pessoais, e neste protótipo tal recurso não foi implementado por não ser foco da Tese.

No Gráfico 10 estão as respostas referentes a questão da conformidade das atividades dos SAPs em relação aos processos de negócios das empresas das quais os usuários estão inseridos.

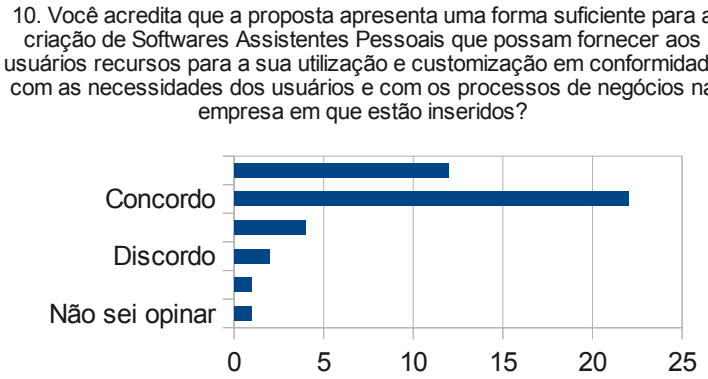

Gráfico 10: Conformidades entre o SAP e os processos de negócios da empresa.

Nesta questão houve um significativo número de entrevistados que assinalaram “concordo fortemente” e um maior número de entrevistados assinalando “concordo”. Isso caracteriza como um ótimo nível de concordância. Nenhum usuário comentou sobre a questão e nem os que discordaram justificaram.

5.2.2 Aplicação do Questionário a Desenvolvedores

O questionário direcionado à pessoas conhecedoras da área de TI e com perfil de desenvolvedores objetivou apresentar a proposta à pessoas que poderiam se utilizar da mesma com o intuito de gerar produtos ou desenvolver soluções que se enquadrassem na arquitetura de assistência pessoal apresentada. Para estes entrevistados foi efetuada a mesma apresentação que mostrada aos outros usuários; contudo com um enfoque mais técnico e direcionada para o desenvolvimento de serviços de software, fazendo com que algumas perguntas fossem diferentes das do anterior. Nessa apresentação, 19 pessoas estiveram presentes e responderam o questionário com as questões apresentadas no Quadro 3.

1. Na sua opinião, a proposta de arquitetura de referência tem relevância para a criação de implementações de Softwares Assistentes Pessoais?
2. Você considera que o protótipo apresentado atende aos requisitos apresentados no modelo conceitual?
3. Você acha aplicável a utilização de um modelo de Softwares Assistentes Pessoais com comportamentos distribuídos/terceirizados, conforme apresentado na proposta, como forma de adequar os Softwares Assistentes Pessoais aos mais variados perfis de usuários?
4. Apesar de ser um protótipo, você acha que a interface de criação/configuração de comportamentos provê as condições para tal?
5. Você acredita que a criação de Softwares Assistentes Pessoais nos moldes como o proposto tem o potencial de se tornar um um foco para um modelo de negócios?
6. Você acha que a implementação de um Software Assistente Pessoal é suficientemente simples para um desenvolvedor de software com certa experiência?
7. Você acha que a implementação de comportamentos para Softwares Assistentes Pessoais é suficientemente simples para um desenvolvedor de software com certa experiência?
8. Você acha que um Software Assistente Pessoal dessa natureza tem o potencial de ser utilizado em cenários reais empresariais?
9. Você acha que o modelo conceitual proposto de Software Assistente Pessoal pode ser considerado inovador?

10. Você acha que, os padrões de Tecnologias de Informação e Comunicação (TICs) utilizados são adequados para criar e manter a interoperabilidade de Softwares Assistentes Pessoais com outros sistemas, principalmente os sistemas das empresas?
11. Você acha que a proposta de arquitetura de referência para softwares assistentes pessoais é aberta o suficiente para que as instâncias de implementações baseadas nessa (gerenciadores, programas, comportamentos, clientes, interfaces, etc.) possam interoperar e ainda fornecer ao usuário um recursos para a utilização e a customização do assistente pessoal em conformidade com as necessidades do usuário e com os processos de negócio da empresa?
12. Por favor, utilize o espaço a seguir para fazer comentários, sugestões, críticas, e eventualmente propostas de trabalhos futuros. Em especial para casos de respostas "discordo" às perguntas.

Quadro 3: Perguntas do questionário aplicado à pessoas com perfil de usuários.

O Gráfico 11 é referente as respostas em relação a relevância da proposta, tal como a mesma questão apresentada às pessoas com perfil de usuários.

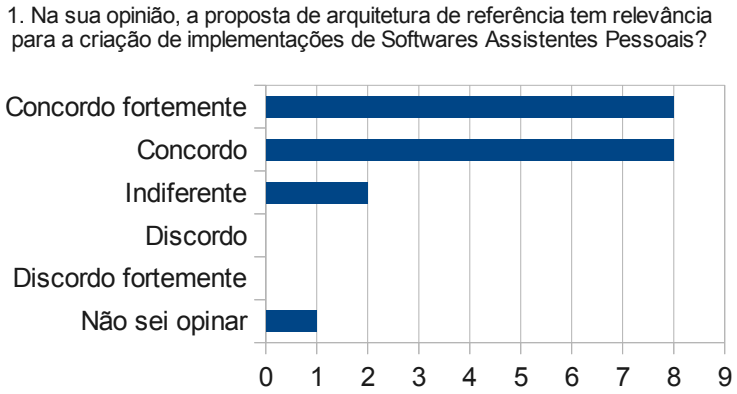

Gráfico 11: Relevância da proposta na visão de desenvolvedores.

Quanto a essa questão, não houve nenhum entrevistado que tenha discordado, e um bom nível de aceitação, 84%, foi obtido quanto a relevância da proposta.

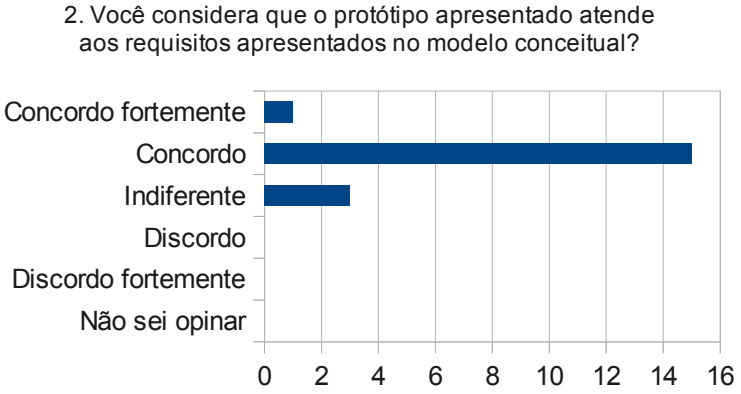

Gráfico 12: O protótipo e requisitos do modelo conceitual.

Com um grande grau de aceitação, considera-se aqui que os entrevistados entenderam a proposta conforme modelo conceitual e implementação baseada no modelo.

O Gráfico 13 contém as respostas ligadas a proposta de distribuição dos comportamentos na forma de funcionalidades (ou serviços web) que pode ser terceirizadas.

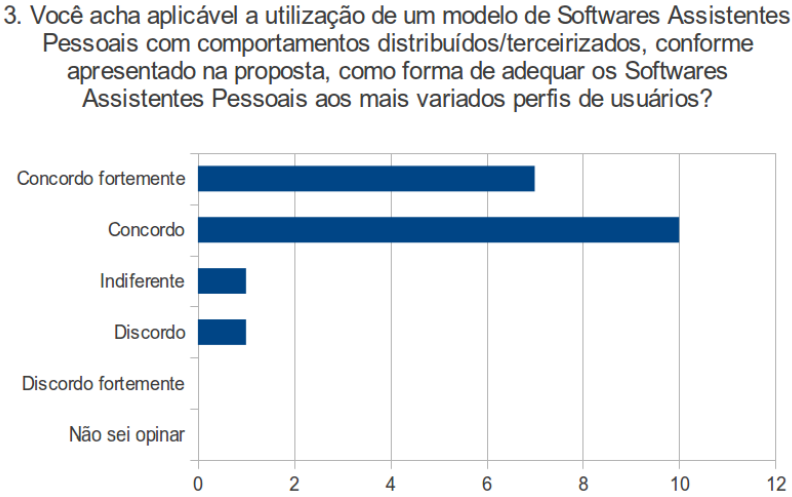

Gráfico 13: Comportamentos distribuídos/terceirizados como forma de adequar os SAPs aos usuários.

Nessa questão, apenas um entrevistado discordou (mas não justificou) e um assinalou como indiferente. Houve um auto grau de concordância com tal especificidade da proposta apresentada.

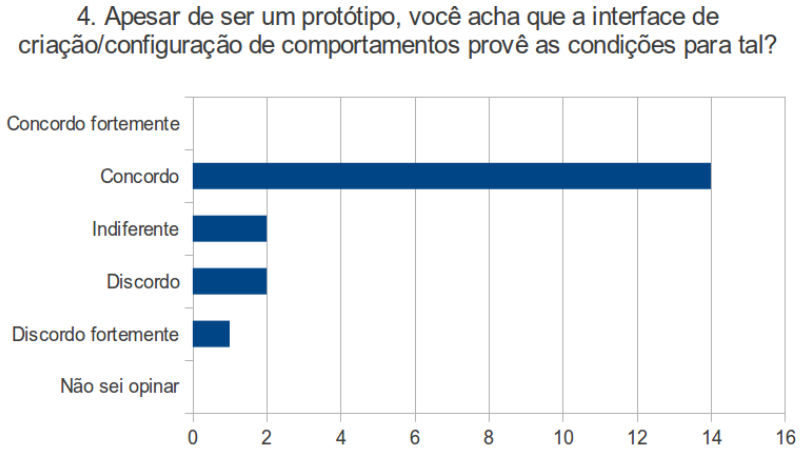

Gráfico 14: Interface criação/configuração de comportamentos na visão de desenvolvedores.

Uma grande quantidade de entrevistados assinalou que concordam que a interface de criação e configuração de comportamentos provê condições para o que foi proposto. Contudo, não houve nenhum que assinalou "concordo fortemente". Isso já era esperado, devido à forma não tão intuitiva que o protótipo oferece para tal. Segundo o Entrevistado #45:

> "Ao meu ver, mesmo que seja um protótipo, é necessário que exista uma interface mais amigável, pois agregará mais valor, principalmente na experiência do usuário". - Entrevistado #45.

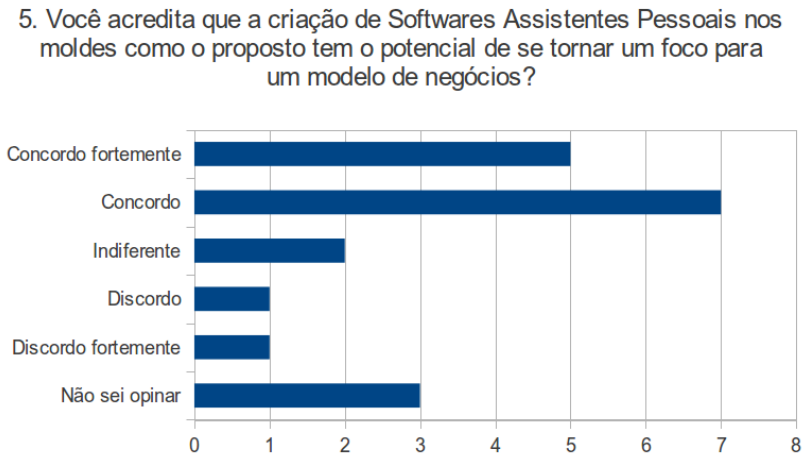

Gráfico 15: Proposta como modelo de negócios.

Um ponto de grande relevância a ser verificado com os desenvolvedores se refere a esta pergunta (Gráfico 15). Neste ponto, é colocada em questão se a ideia de assistência pessoal via software de computador e em conformidade com as especificidades apresentadas na proposta pode se tornar um foco para um modelo de negócio.

Quanto a essa questão, duas pessoas discordaram que a proposta possa se tornar um modelo de negócios, 5 não souberam opinar ou foram indiferentes e 12 assinalaram como “Concordo” e “Concordo fortemente”.

O Gráfico 16 e o Gráfico 17 apresentam as respostas dos desenvolvedores em relação a simplicidade de desenvolvimento de SAPs e comportamentos (serviços web) em conformidade com a proposta apresentada. Estes estão intrinsecamente ligados e são comentados em conjunto.

Gráfico 16: Implementação de SAP simples para um desenvolvedor experiente.

Em relação às dificuldades de desenvolvimento dos comportamentos, segundo usuários com perfil de desenvolvedores, o Gráfico 17 apresenta suas opiniões.

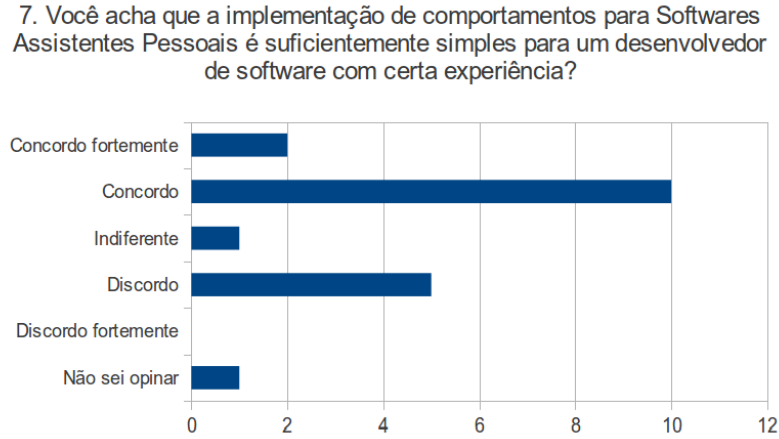

Gráfico 17: Implementação de comportamentos simples para um desenvolvedor experiente.

No caso da questão 6, as respostas apresentadas no Gráfico 16 se mostram com um grau maior de discordância do que concordância. Oito entrevistados responderam de forma negativa e sete positivamente. Há uma afirmação do entrevistado #61 em relação a essa questão:

> “Discordo da questão 6 pois acredito ser complexo o desenvolvimento de um SAP uma vez que este envolve regras de negócios e uma série de técnicas, conceitos e recursos tecnológicos”. - Entrevistado #61.

No mesmo sentido, o Entrevistado #44 explana:

> “Discordo quanto ao fato de que desenvolver seja uma tarefa fácil. Eu vejo nesse tipo de software uma inteligência por trás que não é fácil de implementar. Requer conhecimento avançado na área”. - Entrevistado #44.

Visto que não haveria tempo suficiente na apresentação para apresentar todas as técnicas e algoritmos utilizados para implementar o protótipo como um todo, é esperado que muitos desenvolvedores possam não ter a noção de muita coisa que está envolvida. Entretanto, as respostas apresentadas podem condizer com uma verdade devido ao fato de estar envolvido na proposta de arquitetura de referência e na implementação de um protótipo, muitas tecnologias que podem não ser de conhecimento do desenvolvedor, e para uma atualização nessas tecnologias, o desenvolvedor necessitaria de um período de aprendizado e aperfeiçoamento para entrar em contato com o que precisa conhecer para, por fim, implementar tais recursos.

O Gráfico 18 refere-se a questão de aplicação dos SAPs conforme a proposta em cenários reais empresariais.

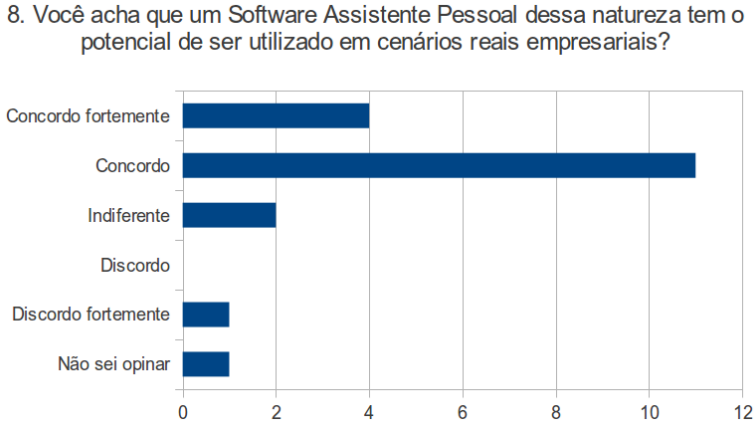

Gráfico 18: Utilização em cenários reais na visão dos desenvolvedores.

Com apenas um entrevistado assinalando “Discordo fortemente” e a grande maioria concordando com a questão, é visto que os desenvolvedores conseguem visualizar a assistência pessoal via software de computador como um recurso que pode ser aplicado em cenários reais empresariais.

A questão 9 trata da inovação do modelo e as respostas são apresentadas graficamente no Gráfico 19.

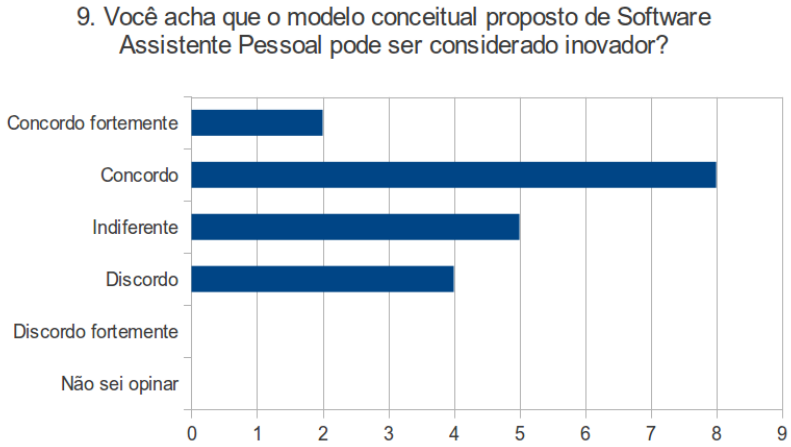

Gráfico 19: Proposta inovadora na visão dos desenvolvedores.

Nessa questão, houveram quatro entrevistados que discordaram. Para o entrevistado #60, “*A aplicação é interessante, mas não traz um conceito inovador, pois já existem outros projetos com os mesmos objetivos*”. Essa é uma premissa verdadeira se visualizar que assistência pessoal via software de computador não é um conceito novo e já existem realmente projetos que trabalham nesse sentido. Contudo, a inovação da proposta não está na criação de softwares assistentes pessoais, mas em uma arquitetura de referência para assistentes pessoais, com comportamentos distribuídos na forma de serviços web e que mantenham um padrão para a interoperabilidade com outros programas, principalmente com os sistemas e processos de negócios da empresa.

A premissa de que o entrevistado #60 pode não ter entendido o diferencial da proposta, ou seja a inovação pode estar no fato de que ele não citou qualquer outro projeto que se enquadre no perfil citado como inovação no presente projeto/proposta. Ainda assim, mais de 50% dos entrevistados concordaram que o conceito é inovador para assistentes pessoais.

A questão 10 visa analisar os padrões escolhidos e se os mesmos são adequados para manter a interoperabilidade dos Softwares Assistentes Pessoais com outros sistemas, e sistemas das empresas. As respostas são apresentadas no Gráfico 20.

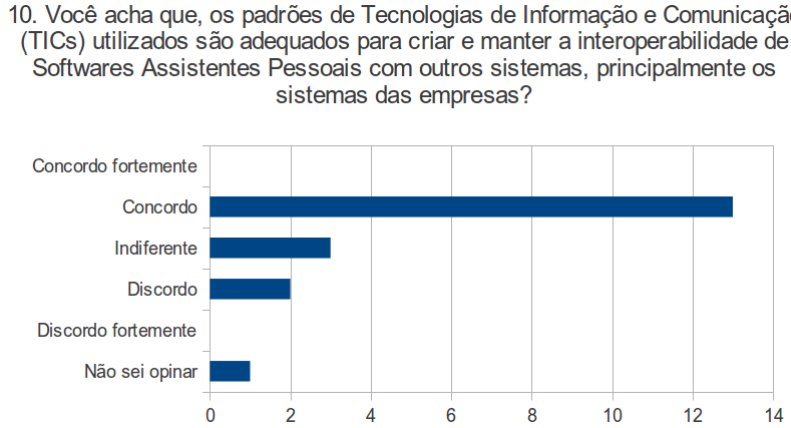

Gráfico 20: Padrões de TIC adequados para a interoperabilidade.

Nessa questão houve uma grande aceitação quanto aos padrões utilizados e apenas dois entrevistados discordaram. Contudo, não houve comentário sobre o porque discordaram.

E por fim, na questão 11, os usuários responderam (Gráfico 21) quanto a proposta ser aberta o suficiente para implementação de instâncias interoperáveis para fornecer assistentes pessoais customizados aos usuários e adequados às suas necessidades e com os processos de negócio da empresa.

Essa questão também obteve um alto grau de aceitação. A formulação dessa questão está diretamente ligada ao entendimento do entrevistado em relação a proposta geral apresentada. Isso qualifica a apresentação como tendo sido compreendida pelas pessoas presentes na apresentação, em termos gerais.

Por fim, algumas sugestões apresentadas pelos entrevistados, mas que não se enquadram em nenhuma questão acima apresentadas, são em relação ao autoaprendizado e a interface do Assistente Pessoal com o usuário.

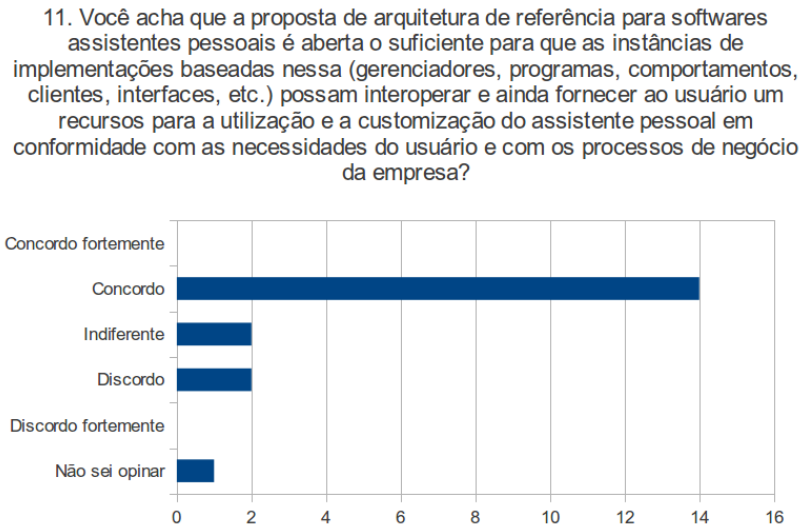

Gráfico 21: Proposta suficiente para adequar as necessidades do usuário aos processo de negócios da empresa na visão dos desenvolvedores.

O Entrevistado #57 afirma que "*acrescentar características de aprendizado seria interessante para evoluir o chatbot"*, e o entrevistado #56 também comenta sobre o mesmo assunto:

> "Pela demonstração não foi possível identificar até que nível o assistente consegue interpretar as frases digitadas na janela de *chat*... Seria interessante integrar a capacidade de aprendizado." - Entrevistado #56.

Ambas as respostas (referente ao Entrevistado #56 e ao #57) foram baseadas na forma de apresentação. No momento em que o caso foi apresentado e executado para demonstração da execução do assistente pessoal, houve o momento em que ocorre de interação entre o SAP e o usuário via IM pelo Gtalk. Como a parte de interação foi na forma de uma conversa com o módulo de *chatbot*, essa interação foi algo que chamou a atenção dos Entrevistados. Isso levou à alguns Entrevistados dar muita relevância à conversa entre o usuário e o SAP. Por outro lado, isso demonstra a importância e relevância do módulo de interação para usuários de SAPs.

5.3 PUBLICAÇÃO DE ARTIGOS CIENTÍFICOS

A publicação de artigos para a comunidade científica é um fator de grande importância para a evolução do trabalho em todo seu ciclo de definição, modelagem e projeto. Assim, procurou-se publicar regularmente em eventos relevantes ao assunto como uma forma de complementos de avaliação da proposta.

Foram publicados e apresentados quatro artigos em eventos nacionais e três em eventos internacionais. Os artigos são apresentados no Quadro 4 e ordenados conforme data de publicação:

Título	Evento	Local	Ano	Tipo
Virtualização de Colaboradores na Manufatura: um modelo baseado em *Agent Bots*	SBAI'2007: Simpósio Brasileiro de Automação Inteligente, Florianópolis. Simpósio Brasileiro de Automação Inteligente	Florianópolis (SC) Brasil	2007	Evento Nacional
Um Arcabouço para Assistentes de Software Virtuais em Ambientes Colaborativos	WESAAC´2008: II *Workshop* - Escola de Sistemas de Agentes para Ambientes Colaborativos	Santa Cruz do Sul (RS) Brasil	2008	Evento Nacional
Uma Proposta de Arcabouço para Agentes Assistentes Pessoais.	I2TS'2008: *7th International Information and Telecommunication Technologies Symposium*	Foz do Iguaçu (PR) Brasil	2008	Evento Internacional
Uma arquitetura de referência para assistentes pessoais: uma abordagem baseada em agentes e serviços de software	WESAAC'2010: VI *Workshop* - Escola de Sistemas de Agentes, seus Ambientes e Aplicações	Rio Grande (RS) Brasil	2010	Evento Nacional
Uma Arquitetura de Referência para Softwares Assistentes Pessoais Baseada na Arquitetura Orientada a Serviços	I2TS'2010: *9th International Information and Telecommunication Technologies Symposium*	Rio de Janeiro (RJ) Brasil	2010	Evento Internacional
Uma Arquitetura de Referência para Softwares Assistentes Pessoais Baseada em Agentes e SOA	WESAAC'2011: VII *Workshop* - Escola de Sistemas de Agentes, seus Ambientes e Aplicações	Curitiba (PR) Brasil	2011	Evento Nacional
A Proposal for Reference Architecture for Personal Assistant Software Based on SOA	I2TS'2011: *10th International Information and Telecommunication Technologies Symposium*	Florianópolis (SC) Brasil	2011	Evento Internacional

Título	Evento	Local	Ano	Tipo
A Proposal for Reference Architecture for Personal Assistant Software Based on SOA	*IEEE Latin America Transactions. vol.10, no.1. - pg.1227-1234*	Revista eletrônica IEEE da América Latina	2012	Revista Internacional
Uma Arquitetura Aberta e Orientada a Serviços para Softwares Assistentes Pessoais	RITA: Revista de Informática Teórica e Aplicada	Revista eletrônica da UFRGS (RS) Brasil	2012	Revista Nacional

Quadro 4: Lista de publicações em eventos científicos e revistas.

Também foi publicado um artigo em uma revista indexada internacional e outro em uma revista nacional.

5.4 CONSIDERAÇÕES

Este capítulo objetivou apresentar uma verificação e avaliação da proposta de três formas: (i) testes sob um protótipo e aplicação de um estudo de caso, (ii) Apresentação da proposta e aplicação de questionário e (iii) publicação de artigos científicos.

Os testes sob o protótipo foram avaliados quanto a cumprir com o que era esperado em conformidade com o estudo de caso selecionado. Dessa forma alguns comportamentos foram criados e utilizados pelo assistente pessoal e, por meio da execução do sistema e um conjunto de informações de *logs* (Seção 9 - Apêndice B – Logs dos Comportamentos), foi verificado se o sistema funcionou conforme esperado.

É importante observar que os critérios de escolha de tecnologias de implementação do protótipo são definidas pelo desenvolvedor. Para fins de avaliação, neste caso, pelo próprio autor desta Tese. Portanto, desde que sigam as definições apresentadas e requisitos associados à arquitetura de referência proposta, outras instâncias de softwares assistentes pessoais podem ser implementadas mesmo que com outras tecnologias.

Ainda, por ser um trabalho qualitativo, foi efetuada uma apresentação para dois grupos de pessoas. O primeiro grupo classificado com perfil usuários de assistentes pessoais, contendo pessoas com conhecimento em TI e pessoas com conhecimento limitado na área de TI. A apresentação para o segundo grupo foi focada para

desenvolvedores, com conhecimentos em TI e teve a perspectiva de apresentar a proposta como um modelo de negócios.

Para resumir a avaliar do trabalho, é aqui feito um paralelo com as perguntas de pesquisa, questão de pesquisa, hipótese, objetivo geral e objetivos específicos especificados na Tese .

5.4.1 Questões de pesquisa

As questões de pesquisa são aqui respondidas em conformidade com todo o processo de geração da arquitetura de referência e implementação da instância implementada. A seguir elas são apresentadas e respondidas.

> Que padrões deve-se utilizar para manter a interoperabilidade do software assistente pessoal com outros sistemas, principalmente os sistemas da empresa?

Foi escolhido como padrão para interoperabilidade entre os SAPs e outros sistemas a utilização da Arquitetura Orientada a Serviços (SOA) como estilo arquitetural para modelar a arquitetura de referência.

> Como fazer o software assistente pessoal entender os processos e negócios de empresa e agir de acordo com cada, considerando as particularidades de um negócio?

Considerando que se está trabalhando sobre o padrão SOA, para que o assistente pessoal possa entender os processos de negócio da empresa, é necessário que a empresa esteja adaptada ou se adapte ao padrão SOA. Ou seja, softwares da empresa devem passar a ter interface de serviços web para que eles possam se integrar aos assistentes pessoais.

> Como tornar um software assistente pessoal adaptável às mudanças de interesses e objetivos dos usuários?

Essa questão está ligada diretamente à adaptabilidade do assistente pessoal ao seu usuário.

> Como tornar um software assistente pessoal flexível de forma que o usuário possa adequar as funcionalidades deste às suas necessidades?

Em relação à flexibilidade, sendo que as funcionalidades a serem utilizadas nos comportamentos do assistente pessoal são serviços web, o usuário pode selecionar os serviços que melhor se enquadram as suas necessidades e tarefas. Além disso, pode configurar internamente no próprio SAP, nos seus comportamentos, a forma como essas atividades, na forma de serviços web, devem ocorrer (Orquestração).

> Como manter a interoperabilidade entre todos os componentes do modelo?

Com a utilização de um conjunto de tecnologias de base para a implementação do modelo SOA, de serviços web, outras TICs/padrões associados, e do Modelo de Referência e Arquitetura de Referência aqui propostos, resolvem-se diversos problemas de interoperabilidades em relação aos componentes do modelo. Contudo, isso não significa que os problemas de interoperabilidade são completamente resolvidos. Ainda existem muitos problemas e esforços no sentido de se resolver problemas interoperabilidade na comunidade científica e de desenvolvedoras de solução de software.

> Quais características a arquitetura de referência deve ter para manter um padrão aberto de implementação para permitir que cada empresa possa implementar suas próprias versões de assistentes pessoais e ainda assim manter a interoperabilidade com os elementos do modelo?

Considerando que o estilo arquitetural escolhido é o SOA e um conjunto de tecnologias envolvidas, já citado no parágrafo anterior, um conjunto de especificações e padronização são utilizadas para permitir que assistentes pessoais baseados na arquitetura de referência proposta sejam interoperáveis. Contudo, seguindo as especificações feitas nessa Tese.

> É possível modelar o sistema distribuidamente, de forma não monolítica? Que características são necessárias para tal?

Sim, na medida em que SOA está implicitamente ligada a distribuição de funcionalidades de forma não monolítica, essa

característica, utilizada pela arquitetura de referência e pelas implementações de assistentes pessoais baseadas na mesma.

5.4.2 Pergunta de pesquisa

> Uma arquitetura de referências para softwares assistentes pessoais pode fornecer aos usuários instâncias implementadas abertas e interoperáveis o suficiente para se adequarem às necessidades deste com relação aos processos de negocio da empresa?

Essa pergunta foi respondida com a implementação de um protótipo baseado na proposta de arquitetura de referência. A implementação seguiu o estilo arquitetural e o modelo de referência e se utilizou de um estudo de caso para a criação dos comportamentos do assistente pessoal. Conforme as necessidades do usuário nesse estudo de caso, o assistente pessoal se adequou de forma efetiva, efetuando as tarefas enquadradas nas atividades do usuário na empresa. A execução foi então verificada neste capítulo, demonstrando a aplicabilidade da mesma.

5.4.3 Objetivo geral

> Conceber uma arquitetura de referência para softwares assistentes pessoais que possa gerar implementações interoperáveis e customizáveis para se adequarem aos processos de negócios da empresa e que auxiliem os usuários em suas tarefas diárias.

No capítulo 3 foi apresentada a arquitetura de referência para softwares assistentes pessoais. Essa foi baseada em um modelo de referência criado com base nas pesquisas sobre o estado da arte de assistentes pessoais e no estilo arquitetural escolhido, que foi o SOA. Por meio dessa arquitetura de referência foi possível criar uma instância de implementação em que o usuário pode customizar seu assistente pessoal para trabalhar com suas necessidades e com os processos de

negócio da empresa nas quais ele está envolvido. Isso, contanto que a empresa também esteja enquadrada com esse padrão.

O assistente pessoal implementado forneceu auxilio ao usuário conforme o estudo de caso escolhido, assumindo certas atividades do usuário na empresa.

5.4.4 Objetivos específicos

> Concepção de um modelo de referência para softwares assistentes pessoais que deve servir para guiar a concepção da arquitetura de referência.

O modelo de referência para SAPs foi concebido no capítulo 3. Este foi baseado na pesquisa de referências do estado-da-arte sobre assistência pessoal via software de computador e serviu como base para a criação da arquitetura de referência.

> Concepção de uma arquitetura de referência para softwares assistentes pessoais;

Ainda no capítulo 3, foi apresentada a arquitetura de referência para SAPS, que se utilizou do modelo de referência criado e de SOA como estilo arquitetural.

> Concepção de uma arquitetura de implementação baseada no modelo de referência e na arquitetura de referência;

No capítulo 4 foi apresentada a modelagem da arquitetura de implementação baseado na arquitetura de referência e servindo de base para a implementação de um protótipo.

> Concepção e implementação de um protótipo computacional associado à arquitetura de referência concebida, para servir de instrumento de avaliação da proposta.

Por fim, um protótipo foi implementado com base nos objetivos acima descritos e alcançados. Este protótipo foi testado, sendo que sua verificação foi efetuada na seção 5, comprovando sua aplicabilidade.

6 CONSIDERAÇÕES FINAIS

Em um cenário de softwares que fornecem assistência às pessoas em suas atividades diárias, esse trabalho apresentou uma proposta de uma Arquitetura de Referência para Softwares Assistentes Pessoais. Para tal, foram efetuadas pesquisas do estado-da-arte na área de assistência pessoal via software de computador. Essas pesquisas forneceram bases suficientes para a formulação de características necessárias para se trabalhar neste enfoque. Isso foi motivador para a criação de um modelo de referência genérico para softwares assistentes pessoais.

Juntamente com esse modelo de referência, a proposta se apropriou da Arquitetura Orientada a Serviços, que veio servir de estilo arquitetural para que a arquitetura de referência pudesse resolver, ou minimizar, uma série de aspectos relacionados à assistência.

Entretanto, constata-se que a área de assistentes pessoais tem ainda enormes desafios a serem enfrentados, incluindo ao que se refere a limitações das TIs atuais para certos propósitos (por exemplo, o problema da interoperabilidade semântica), os impactos organizacionais e a complexidade do problema de composição de comportamentos dentro da abordagem SOA, etc.

Em termos do uso de Softwares Assistentes Pessoais em ambientes de empresa, verificou-se que SAPs não têm sido desenvolvidos para serem integrados aos processos de negócios empresariais. Isso cria pouca agilidade em processos de decisão e potencializa inúmeros tipos de erros de interpretação, digitação, não adequada ou útil execução de ações, etc. Na prática, faz com que o usuário final tenha que migrar de ambientes ao longo do seu trabalho e, ele, usuário, tenha que realizar as ações de interoperabilidade gerais, dentro de um dado processo e contexto de negócio. Por processos de negócios empresariais quer-se dizer ações e transações usuais em empresas, como compras, vendas, negociações, consulta de preços, resposta a cotações, distribuição, gestão da produção, etc. Apesar de ainda a maioria das empresas utilizar suas próprias representações de processos, observa-se o uso crescente de padrões de processos negócios empresariais, tais como UBL (UBL, 2011), ebXML (ebXML, 2011) e RosettaNet (RosettaNet, 2011).

A solução para todos os problemas relacionados com os vários aspectos de um SAP adaptativo, flexível, interoperável, interagente e integrado aos ambientes empresariais é das mais complexas e, sob certas

perspectivas, ainda não plenamente solucionável com o estado das tecnologias atuais e abordagens conceituais.

Esse trabalhou trouxe uma contribuição conceitual e de modelo de implementação que vai na direção daqueles requisitos. A premissa básica associada a esta pesquisa é de que as empresas precisam cada vez mais tomar decisões confiáveis e de forma ágil, mas seus colaboradores estão cada vez mais imersos em inúmeros processos de negócios, simultâneos, quer repetitivos quer complexos, e cada vez mais sem o devido tempo para geri-los. Portanto, SAPs tem o potencial de atuar como um importante mecanismo / ferramenta de trabalho para auxiliar seus usuários a executar suas atividades melhor ou mesmo automaticamente, substituindo-os em certas ações.

Este trabalho deve ser visto como uma contribuição para a área de SAPs, indo na direção dos inúmeros requisitos, em vários níveis, identificados na literatura.

A primeira contribuição a ser citada, é que não existe qualquer modelo ou arquitetura de referência para a criação de assistentes pessoais. Apenas existem implementações prontas ou direcionadas aos seus próprios requisitos. Não existe a flexibilidade de criação de novas funcionalidades para esses SAPs. Assim, esse trabalho traz justamente tal proposta de uma Arquitetura de Referência que possa servir de base para a criação de soluções que possam ser integradas, adaptadas aos usuários e flexíveis o suficiente para que ela possa servir ao usuário também em conformidade com as mudanças das atividades do usuário durante o tempo e ainda ser interoperável com os ambientes e sistemas empresariais.

A segunda contribuição é que tal proposta fornece um modelo de negócios a ser explorado, tanto na criação de assistentes pessoais, como na criação de serviços web para compor os comportamentos dos assistentes pessoais. Cada empresa pode possuir sua própria implementação de elementos da arquitetura e pode integrá-los à outros elementos produzidos por outras. Esses serviços poderiam ser vendidos, alugados ou fornecidos gratuitamente por terceiros, dependendo do seu próprio modelo de negócios.

Essa forma de funcionalidades distribuídas e com protocolos padrões de chamada também possibilita que um usuário de um assistente pessoal possa utilizá-lo para gestão de suas atividades na empresa, se essa fornecer interfaces de serviços web de seus sistemas, para que esses possam ser vistos pelo assistente pessoal.

Para título de avaliação dos resultados, três procedimentos metodológicos foram executados. O primeiro foi a implementação de uma instância baseada na arquitetura de referência em cima de um estudo de caso específico de um funcionário com a necessidade de auxílio em suas tarefas. O segundo caminho foi a apresentação da proposta e aplicação de questionários para dois grupos: um grupo de potenciais usuários de Softwares Assistentes Pessoais e outro de potenciais desenvolvedores de soluções baseadas na proposta. O terceiro foi a publicação em apresentação de artigos científicos para avaliação da proposta durante todo o seu ciclo de vida, de modo a verificá-la frente a um crivo de cunho científico.

Em relação ao desenvolvimento de uma instância implementada baseada na arquitetura de referência proposta, foi gerado um protótipo de Software Assistente Pessoal envolvendo todo o conjunto de funcionalidades necessárias para a execução de apenas um exemplo de caso para uso de assistência. Mesmo assim, toda a complexidade compreendida para atender outros exemplos já está presente. Faz-se apenas necessário a existência de serviços web prontos para servirem como funcionalidades para os outros exemplos.

Com base nos resultados de testes de implementação final, verificou-se que esta instância se comportou conforme o que foi proposto na arquitetura de referência e executou corretamente as ações associadas aos comportamentos do processo de negócio envolvido no exemplo estudado. Embora esse não tenha sido o principal foco do trabalho, o sistema apresentou boa estabilidade perante as atividades realizadas em uma arquitetura distribuída, mas em rede local.

A instância derivada mostrou-se coerente com o modelo e arquitetura de referência, o que garantiu a coerência do SAP derivado. Com a implementação realizada, mostrou-se igualmente que é possível realizar aquela derivação assim como ter um SAP integrado ao ambiente de negócios da empresa, interoperando com subsistemas, que trabalhem dentro de uma arquitetura aberta e de intenso uso de padrões como descrito nesse trabalho.

Com base nas respostas dos questionários, baseados na forma de pesquisa qualitativa desse trabalho, pode-se observar uma boa aceitação da proposta, tanto em nível de pessoas com perfil de usuário, quanto de pessoas com perfil de desenvolvedores de aplicações e soluções baseadas na arquitetura proposta.

A proposta aqui apresentada, além de apresentar uma inovadora arquitetura de referencia para a criação de SAPs, fornece indicativos da

criação de comportamentos baseados em SOA. Este estilo arquitetural aberto permite que funcionalidades do SAP não precisem ser criadas especificamente para o SAP do usuário em si, mas podendo ser utilizadas para a composição de comportamentos de forma inclusive dinâmica. Isso também permite que a própria empresa do usuário forneça funcionalidades que façam a integração do SAP do usuário com os processos de negócio da empresa. Outra vantagem é que funcionalidades podem ser desenvolvidas por terceiros, e utilizadas pelo usuário de forma gratuita ou paga. Isso pode levar a um foco específico de negócios, que é a criação de comportamentos e funcionalidades específicas para SAPs, apenas criando serviços web e disponibilizando-os aos usuários para a composição dos seus comportamentos. Como os serviços web vêm se firmando como um padrão de mercado, isso provê, consequentemente, um leque já existente de opções de funcionalidades e comunicação para quem queira desenvolver SAPs que sigam a proposta desse trabalho.

6.1 LIMITAÇÕES

Baseado em uma análise conjunta dos resultados das etapas de avaliação da proposta, algumas limitações foram identificadas, não em nível da arquitetura de referência especificamente, mas a nível do protótipo implementado, a saber:

Para a avaliação do protótipo, a seleção de um grupo de usuários para criar e utilizar assistentes pessoais adaptados às suas necessidades não pôde ser plenamente realizada. Para cada caso, seria necessário a avaliação de um estudo de casos diferentes e a criação de serviços web para compor os comportamentos dos assistentes pessoais desses usuários, no caso dos serviços web não existirem. Isso levaria a uma nova fase de desenvolvimento de serviços web específicos para cada usuário. Isso porque, por mais que os serviços web já tenham se tornado um padrão *de facto*, nem todas empresas aderiram a esse padrão e nem todas as funcionalidades necessárias para compor os comportamentos dos assistentes pessoais estão disponíveis para que o assistente possa efetivamente fornecer assistência. Em tempo, as empresas dos usuários podem não estar adaptadas com essa interface de serviços web para seus sistemas internos.

As interfaces não ficaram tão intuitivas aos usuários, principalmente quanto a criação e configuração de comportamentos na

interface web de configuração do GAP. O fato da importância da interface foi constatado nas respostas dos questionários. Seguindo a premissa do parágrafo anterior, para que os usuários pudessem utilizar o sistema, seria necessário um período de treinamento para que os usuários pudessem criar seus próprios comportamentos na forma de algoritmos. Isso pois o modo de criação de comportamentos na forma de algoritmos é limitadora para usuários comuns, sem o conhecimento de lógica de programação. Esse formato manteria afastados do sistema diversos potenciais usuários por necessitar de um certo conhecimento básico de programação para poder criar qualquer comportamento que fosse, mesmo que bastante simples.

Também, alguns dos serviços foram implementados na forma de protótipo e sem diversas funcionalidades que poderiam vir a ser interessantes para funcionarem de uma maneira mais genérica, que não no estudo de caso apresentados. Pode-se citar o caso do módulo de *chatbot*, implementado de maneira bem simples e que apenas encontra respostas baseadas em palavras chaves contidas nas frases que o usuário digita e envia ao assistente pessoal. Tal módulo poderia trabalhar com tecnologias mais atuais para *chatbots*, tal como Processamento de Linguagem Natural.

6.2 PONTOS DE REFLEXÃO

O que se pôde observar da parte do estado-da-arte de assistência pessoal via software de computador, tanto a nível de pesquisa quanto a nível comercial, é que existem esforços bastante significativos. Contudo, esses esforços ainda requerem alcançar uma maior maturidade, tanto ao que se refere às possibilidades de ação desses assistentes pessoais, mundo virtual interconectado e interoperável, o interfaceamento do assistente com o mundo real, inteligência artificial do assistente, quanto as questões filosóficas, éticas e psicológicas de aceitação.

Por mundo virtual interconectado e interoperável compreende-se a utilização de padrões de comunicação e intercomunicação entre softwares, sistemas das empresas e comércio eletrônico. Um assistente pessoal precisa poder acessar diferentes meios para conseguir realizar até as que poderiam ser consideradas mais simples tarefas realizadas por um assistente pessoal real, tais como leitura, seleção e respostas de e-mail, agendamento de reuniões, reservas de passagens de avião ou estadia em hotéis, pesquisas de informações na internet. Esses recursos

precisam ser acessados por SAPs de forma padronizada (i.e. uma interface padrão). A proposta dessa tese sugere tal padronização, mas ainda há muitos caminhos a percorrer até que uma quantidade suficiente de provedores de produtos/serviços, empresas dos usuários de SAPs, sistemas de compra/venda, sistemas verificação de cartão de crédito e outros, estejam em concordância com essa padronização. Isso se houver uma migração em massa para esses padrões.

Por interfaceamento do assistente com o mundo real tem se a forma como os assistentes interagem com seus usuários. Esses podem ser vistos tanto como softwares em que o usuário apenas gerencia informações e consegue respostas na forma de atividades automatizadas, como agentes robóticos, humanoides ou não, que podem interagir no mundo real para auxiliar o usuário em diversas atividades, tanto virtuais como reais. Uma tendência observada é que busca-se uma interface mais amigável com o usuário. É mais intuitivo e natural o usuário "conversar" com seu assistente sem a necessidade de teclado, mouse ou monitor. A exemplo disso pode-se citar o Siri da Apple como um forte representante a nível comercial e o Projeto PAL como um significativo representante da área de pesquisas. Inclusive, mesmo o assistente do Projeto PAL sendo para fins militares, a interação entre os usuários (militares estrategistas) é via conversação. Em tal situação é possível imaginar que uma interação via conversação poderia ser considerada uma forma de interação com desperdícios de informações inúteis, ou seja, era de se imaginar que uma linguagem formal padronizada agilizaria o processo de comunicação.

Neste caminho, muitos trabalhos estão sendo feitos na área da robótica, na forma de brinquedos para comercialização, na linha de robôs para a automatização industrial, robôs para entretenimento (performances, teatro, shows, cinema), robôs para exploração marítima, vulcões, espacial, entre outros. Mesmo softwares que interagem com os usuários via dispositivo computacional já têm tido certa aceitação em produtos de massa, novamente citando o Siri da Apple no iPhone.

Por Inteligência Artificial do Assistente tem-se a utilização de diversas tecnologias da Ciência da Computação para tornar o comportamento do assistente "inteligente" ou simuladamente inteligente. A questão toda circunda na forma como os assistentes conseguem entender o que os usuários necessitam, possam obter informações diversificadas e distribuídas dos usuários e detectar relações entre essas informações para poder formar uma base de dados de preferências dos usuários, prioridades de atividades, listas de

contatos, tarefas, etc. Além de elementos para poder fazer o interfaceamento com o usuário, identificação do usuário e outras pessoas via imagem ou voz, processamento de linguagem natural para a compreensão das mensagens e outros.

Quanto à questões éticas, filosóficas e psicológicas, o assunto gira em torno de novas formas de trabalho ou da aceitação dessas novas tecnologias pelas pessoas. Muito se tem falado, desde o início da era industrial e da padronização dos processos industriais, sobre que a automatização viria como um esforço para que as pessoas pudessem trabalhar de forma mais eficiente, conseguindo produzir mais em menos tempo. Isso acarretaria que as pessoas poderiam trabalhar menos tempo nessas atividades, sobrando mais tempo para atividades de laser e para o crescimento intelectual pessoal. Contudo, o que se tem observado ao longo dos anos é que as pessoas têm produzido mais, mas ainda assim continuam trabalhando a mesma quantidade de horas, gerando mais lucros para seus empregadores. As pessoas têm cada vez mais se inserido a ambientes de trabalho estressantes e desgastantes pelos novos processos em busca de maior produtividade. No mesmo sentido, a questão de que assistentes pessoais levarão as pessoas à um novo patamar, podendo ficar liberadas para outras atividades, é também, de certa forma, uma utopia.

Também, existe o problema da aceitação das pessoas à esses softwares que diz-se que irão fazer as coisas por ela. Muitas pessoas se sentem ameaçadas por acreditarem que esses recursos irão substituí-las, restando para ela ou uma atividade de trabalho inferior ou a demissão. E isso, de certa forma, não está errado. É só observar a automatização nas indústrias, por exemplo, as indústrias de produção de carros. Essas passaram por um período de robotização de várias atividades que antes eram efetuadas por humanos. Isso gerou a execução de movimentos rápidos, efetivos e padronizados, gerando produtos de boa qualidade, em grande quantidade e em menos tempo. Porém, gerou também consequências drásticas para os trabalhadores, agravando o desemprego. Consequentemente a históricos como este, as pessoas tendem a se sentirem ameaçadas por qualquer nova tecnologia que tenha como promessa “auxiliá-la” em suas tarefas diárias e até “substituí-la” em outras.

Além desses pontos de reflexão, ainda haveria muito a se discutir sobre a criação, ou não, de software para o auxílio (ou substituição) do usuário em suas atividades diárias. Apesar de poderem existir muitos pontos de vista divergentes e/ou convergentes sobre esse assunto, ainda

é de objetivo de pesquisadores das áreas de automação e computação a busca por aquelas utopias supracitadas. É de objetivo, e deve continuar sendo, a busca pela melhor qualidade de vida dos usuários e formas de trabalhos mais dignas.

6.3 TRABALHOS FUTUROS

Tendo como base as limitações e certos pontos apresentados durante toda a extensão da tese, alguns trabalhos futuros podem ser sugeridos como forma de gerar novas pesquisas subsequentes ou derivados a este trabalho em questão.

A proposta da tese se manteve em uma linha de padronização quanto ao SOA, mantendo sob uma perspectiva OASIS. Entretanto, alguns outros padrões foram constatados durante algumas pesquisas. Inclusive, para criar o serviço de interoperabilidade de acesso ao Twitter, foi utilizado o acesso ao site por meio de um padrão chamado JSON. Assim sendo, um outro direcionamento poderia ser a utilização de serviços web em outros padrões, ficando estes fora da classificação de serviços de interoperabilidade. Quanto ao acesso a esses serviços, como esses já possuem seu próprio padrão, ficaria em uma camada de implementação no próprio GAP. Para o usuário o processo de orquestração ou criação de comportamento ainda deveria ficar transparente, mas o GAP em si estaria preparado para acessar outros tipos de serviços web, em outros padrões. Isso ficaria então limitado à programação do GAP, e não na arquitetura de referência. Por um lado isso ampliaria a quantidade de serviços web disponíveis que poderiam ser utilizados para compor os comportamentos do SAP. A Google, por exemplo, também possui um padrão próprio, com APIs, para acessar seus serviços na forma de serviços web. Um estudo mais apurado, com testes e validação de tal argumento pode se enquadrar como um interessante trabalho futuro derivado desta Tese.

Um outro trabalho que pode ser feito é quanto ao módulo de *chatbot* do SAP. Este foi desenvolvido nesta Tese em uma perspectiva de simplicidade, apenas trabalhando com palavras chaves da mensagem de entrada e por meio de similaridades quanto às frases de saída. Também não há aprendizado efetivo e o reconhecimento da pessoa da qual o *chatbot* está interagindo também é limitado. Contudo, constatou-se por meio das entrevistas que essa funcionalidade é uma das que mais chamou a atenção. Isso mostra a importância de uma boa interação entre

SAP e usuário. Um aperfeiçoamento de tal serviço pode ser bastante interessante, seguindo caminhos de Processamento de Linguagem Natural (PLN), aprendizado durante a interação e melhor reconhecimento da pessoa da qual interage com o assistente.

Um problema constatado e já descrito é quanto a interface para criação e configuração de SAPs e comportamentos. No protótipo aqui apresentado, seguiu-se um caminho de conveniência quanto a modelagem dos comportamentos, por meio da modelagem de algoritmos. Contudo, tal caminho não deixou tal tarefa simples de se trabalhar, principalmente para usuários leigos em programação. Formas de modelagem de comportamentos mais intuitivas e gráficas podem ser pesquisadas, estudadas e testadas como forma de ampliar o leque de possíveis usuários que podem configurar os comportamentos no SAP.

O serviço ISAP ficou limitado à alguns elementos. Poderia se ampliar as possibilidades de acesso desse serviço, alcançando outras redes sociais como Google+, Faceboook, Google Agenda e outros. Isso ampliaria as possibilidades de elementos de pesquisa do SAP para auxiliar seu usuário.

É previsto também, conforme requisitos básicos de SAPs, que um SAP precisa aprender e se adaptar para poder dar uma melhor assistência ao seu usuário. Na visão da proposta, tal recurso pode ser feito por meio da criação de um serviço, ou serviços, de aprendizado. Poderiam haver serviços para gerenciar bases de conhecimento para os assistentes pessoais, serviços para – no caso de uma implementação de SAP na forma de Agente BDI (*Beliefs-Desires-Intentions*) – gerenciar informações de Crenças, Desejos e Intenções do SAP, etc.. Estudar, pesquisar, modelar, implementar e validar recursos de aprendizado na forma de serviços (*softwar-as-a-service*) e que podem ser utilizados por SAPs e pelos comportamentos dos SAPs, em conformidade com essa proposta, é um outro caminho que pode ser bastante atraente como trabalhos futuros.

Para o usuário modelar os comportamentos do SAP, ele precisa saber os serviços das quais ele pode utilizar. Formas de indexação e pesquisa de serviços web na internet, tais como o seekda ou Takever.eu, mais eficientes, intuitivas e facilitadas também pode ser visto como um trabalho interessante. Um modelo interessante a se analisar é o sistema de pesquisa de aplicativos do Android[31] Market (autal Google Play), ou Central de Programas do Ubuntu. O usuário poderia pesquisar serviços

[31]Sistema Operacional para Dispositivos Móveis da Google, baseado no Sistema Operacional GNU/Linux.

web e utilizá-los apenas arrastando, soltando e configurando o serviço no comportamento sendo editado. Contudo, esse estudo ainda deve analisar possibilidades disso e impacto que teria sobre a atual proposta.

7 REFERÊNCIAS

AALST, V.D.W.M.P.; HOFSTEDE, A.H.M.; WESKE, M. ***Business process management: a survey. In: Computer Science***. *Springer*. 2003.

ABLONG, T.; BARTLETT, J.; BARTLEY, D.; BUSBY, J.; et al.. ***Interoperability Technical Framework for the Australian Government. Technical Report*** - *National Office for the Information Economy. Canberra, Australia.* 2005.

AGRAWAL, R., J. BAYARDO, Jr., R., GRUHL, D., PAPADIMITRIOU, S. ***Vinci: a service-oriented architecture for rapid development of Web applications.*** *Computer Networks, vol.39, issue 5,* pg.523–539. 2002.

ANGEHRN, A., NABETH, T., RAZMERITA, L. et al. ***K-InCA: Using Artificial Agents for Helping People to Learn New Behaviours.*** *Proc. IEEE International Conference on Advanced Learning Technologies.* pg.225-226. 2001.

BALDAM, R.L., VALLE, R., PEREIRA, H., HILST, S., ABREU, M., SOBRAL, V; et al. **Gerenciamento de processos de negócios BPM - *business process management***. 2ª ed. São Paulo: Érica, 2007.

BARANAUSKAS, J.A. **Extração automática de conhecimento por múltiplos indutores. Tese de Doutorado**. ICMC-UFSC. <*http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08102001-112806*> acessado em Jun/2009. 2001.

BASS, L.; CLEMENTS, P. and KAZMAN, R. ***Software architecture in practice****. 2nd ed*. Addison-Wesley, 2003.

BLUM, A.L.; LANGLEY, P. ***Selection of relevant features and examples in machine learning***. *In: Artificial intelligence.* Vol 97. Num 1-2. pg.245-271. Elsevier. 1997.

BOCIONEK, S. ***Software secretaries: learning and negotiating personal assistants for the daily office work In Systems, Man, and Cybernetics.*** *'Humans, Information and Technology'. 1994 IEEE International Conference on*, 12 vol.1. 2-5 Oct. 1994.

BROOKS, R.A. ***A robot that walks: emergent behaviours from a carefully envolved network***. *Neural Computing*, vol.1, num.2, pg.253-262. MIT Press. 1989.

BROOKS, R.A. ***Integrated systems based on behaviors***. *ACM SIGART Bulletin*, vol.2, num.4, pg.46-50. ACM New York, NY, USA. 1991.

BUSH, J.; IRVINE, J. and DUNLOP, J. ***Personal Assistant Agent and Content Manager for Ubiquitous Services***. *Wireless Communication Systems, 2006. ISWCS'06. 3rd International Symposium on.* pg.169-173. 2006.

CARRILLO-RAMOS, A., VILLANOVA-OLIVERr, M., GENSEL, J. et al.. ***Knowledge Management for Adapted Information Retrieval in Ubiquitous Environments***. *Proceedings of the 2nd International Conference on Web Information Systems and Technologies (WEBIST 2006) Insticc Press*, Portugal , 21-29. 2006.

CERAMI, E. ***Web Services Essentials***. *O'Reilly & Associates, Sebastopol,* CA, EUA, 1a. edição. 2002.

CERVO, A. L. e BERVIAN, P. A. **Metodologia Científica**. São Paulo: Prentice Hall., 5a. Edição. 2002.

CHAPPELL, D. A. e JEWELL, T. ***Java Web Services***. *O'Reilly & Associates, Sebastopol*, CA, EUA. 1a. Edição. 2002.

CHAUVAT, Nicolas. ***Narval, the intelligent personal assistant or how the french Linux Gazette is built.*** *Linux Gazette, issue 59.* Nov. Disponível em <*http://linuxgazette.net/issue59/chauvat.html*>, acessado em Novembro/2011. 2000.

COULOURIS, G.F.; DOLLIMORE, J.; KINDBERG, T. ***Distributed systems: concepts and design***. *Addison Wesley Longman*, 2005.

DAMÁSIO, A. **O mistério da consciência: do corpo e das emoções ao conhecimento de si**. Tradução Laura Teixeira Motta. São Paulo. Companhia das letras. 2000.

DE JONG, K. ***Learning with genetic algorithms: an overview***. pg. 121-138. *In: Machine Learning*, vol.3, num.2. Springer. 1988.

DIETTERICH, T.G. ***Machine learning research: for current directions***. In: AI Magazine. 1997.

FIGUEIREDO, G.; MAYWORM, M. **Reutilização de software em sistemas de comércio eletrônico**. COPPE Sistemas. 2004.

FISCHER, G. ***User modeling in human-computer interaction. In: User modeling and user-adapted interaction***. vol.11, num.1, pg.65-86. Springer. 2001.

FRANCO, R.D.; NEYEM, A.; OCHOA, S. et al. ***Supporting mobile virtual team's coordination with soa-based active entities. Establishing The Foundation of Collaborative Networks****, IFIP TC 5 Working Group 5.5 Eighth IFIP Working Conference on Virtual Enterprises*, September 10-12, 2007, Guimarães, Portugal. IFIP 243 Springer 2007.

GARIMELLA, K.; LEES, M.; WILIAMS, B. ***BPM basic for dummies***. Indiana: Wiley Publishing. 2008.

GESSER, Carlos E. **Uma abordagem para a integração dinâmica de serviços web em portais**. Eng. Elétrica - UFSC. Dissertação. 2006.

GIL, A.C.. **Como elaborar projetos de pesquisa**. 5ª edição. São Paulo: Atlas. 2010.

GOLDBERG, D.E.; HOLLAND, J.H. ***Genetic algorithms and machine learning***. pg. 95-99. *In: Machine Learning*, vol.3, num.2. Springer. 1998.

GREENWOOD, D.; CALISTI, M.; at al. ***Engineering Web Service - Agent Integration***. *IEEE International Conference on Systems, Man and Cybernetics.* Vol 2. 2004.

GÜNTHER, H. **Como elaborar um questionário.** Série Planejamento de pesquisa nas ciências sociais, nº 1. Brasília, DF. Laboratório de Pesquisa Ambiental, 2003.

HOFMEISTER, C.; NORD, R.; SONI, D. ***Applied Software Architecture***. Addison Wesley, 2000.

HOYLE, M.A. and LUEG, C. ***Open Sesame!: A Look at Personal Assistants***. *Proceedings of the International Conference on the Practical Application of Intelligent Agents* (PAAM97), London, 1997, 51 60, 1997.

HAWES, N. ***Real-time goal-oriented behaviour for computer game agents***. *In: Game-On 2000, 1st International Conference on Intelligent Games and Simulation*, pg.71-75. 2000.

HUHNS, Michael N. SINGH, M.P., ***Personal assistants***. *IEEE Internet Computing*, Vol. 2, Issue 5, pp. 90-92. Sep./Oct. 1998.

HUHNS, Michael N., STEPHENS, Larry M. ***Multiagent systems and societies of agents***. *In: WEISS,* ***Gerhard, Multiagent systems: a modern approach to distributed artificial inteligence***. Weiss, Gerhard, MIT, 1999.

HUHNS, Michael N. ***Agents as Web services***. Internet Computing, IEEE 6, 93-95. 2002.

IDABC, C. S.. ***European Interoperability Framework for pan-European eGovernment Services*** *- Interoperable Delivery of European eGovernment Services*. 2004.

IEEE. ***Standard Computer Dictionary*** *- A Compilation of IEEE Standard Computer Glossaries-Institute of Electrical and Electronics Engineers* (IEEE). 1991

KAMOUN, F. ***A roadmap towards the convergence of business process management and service oriented architecture***. *In: Ubiquity Journal*, V.8, n.14, p.1-8. 2007.

KSHEMKALYANI, A.D.; SINGHAL, M. ***Distributed Computing: Principles, Algorithms, and Systems***. *Cambridge University Press*. 2008.

KUNO, H.; SAHAI, A. ***My agent wants to talk to your service: personalizing web services through agents***. *Proceedings of the First International Workshop on Challenges in Open Agent Systems*. Pg 25-31. 2002.

LACKENBY, C. e SEDDIGHI, H. ***A dynamic model of virtual organisations: formation and development***. *In: Collaborative Business Ecosystems and Virtual Enterprises.* Camarinha-Matos, páginas 37–44. 2002.

LEVIN, E. PIERACCINI, R. ECKERT, W. ***A stochastic model of human-machine interaction for learning dialogstrategies***. *In: IEEE Transactions on Speech and Audio Processing*. vol.8, num.1, pg.11-23. 2000.

LIKERT, R. ***A technique for the measurement of attitudes***. New York. 1932.

LIN, L.J. ***Self-improving reactive agents based on reinforcement learning, planning and teaching***. pg. 293-321. In: Machine Learning, vol.8, num.3. Springer. 1992.

LOSS, Leandro. **Aprendizado de redes colaborativas de organizações: um arcabouço para dar suporte à criação e uso de conhecimento**. 2007. Tese (Doutorado) - Curso de Engenharia Elétrica, UFSC, Florianópolis, 2007.

MAES, P. ***Agents that reduce work and information overload***. *Communications of the ACM*, nº.7, vol37, NY, USA, 1994

MARKOFF, John. ***A Software Secretary That Takes Charge***. *New York Times.* Disponível em: <*http://www.nytimes.com/2008/12/14/business/14stream.html?_r=1&scp=7&sq=personal%20assistant&st=cse*>. Acessado em Março/2009. 2008

MICHAELL, T., CARUANA, R., FREITAG, D., MCDERMOTT, J., ZABOWSKI, D. ***Experience With a Learning Personal Assistant***. *Communications of the ACM, July*, 1994.

MICHALSKY, R.S. ***Learnable evolution model: evolutionary processes guided by machine learning***. *pg. 9-40. In: Machine Learning*, vol.38, num.1. Springer. 2000.

MINSKY, M. ***The society of mind***. Simon & Schuster. 1988.

MONARD, M.C.; BARANAUSKAS, J.A. **Conceitos de aprendizado de máquina**. Em: Rezende, S.O. Sistemas inteligentes: fundamentos e aplicações. Barueri, SP: Manole. pgs 89-114. 2003.

MYERS, Karen L., YORKE-SMITH, Neil. ***A Cognitive Framework for Delegation to an Assistive User Agent***. *AAAI 2005 Fall Symposium on Mixed-Initiative Problem Solving Assistants, Arlington, VA, November* 2005.

NILSON, N.S. ***Principles of Artificial Intelligence*****.** Springer Verlag, Berlin, 1982.

O'BRIEN, L.; BASS, L.; MERSON, P. ***Quality Attributes and Service-Oriented Architectures***. *Technical Note - Software Engineering Institute, Carnegie Mellon University*. 2005.

PAUROBALLY, S.; JENNINGS, N.R. ***Developing Agent Web Service Agreements***. *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence* (WI'05). 2005.

PFEIFER, R. LUNGARELLA, M. LIDA, F. ***Self-organization, embodiment, and biologically inspired robotics***. *Sciencer*, vol.318, num.5853, pg.1088. AAAS. 2007.

PIAZZA, André P. **Uma abordagem para interoperabilidade entre plataformas heterogêneas de serviços web para redes colaborativas de organizações**. Eng. Elétrica - UFSC. Dissertação. 2007.

PIRES, P.F. **Ambiente para especificação e execução ad-hoc de processos de negócios baseados em serviços web: projeto webflow**. UFRN. Disponível em: <*http://www.dimap.ufrn.br/~paulo.pires/Projetos/WF/index.htm*> acessado em junho/2009. 2004.

QUINLAN, J.R. ***Induction of decision trees***. *pg.81-106. In: Machine Learning, vol.1, num.1. Springer*. 1986.

RICCI, A.; BUDA, C.; ZAGHINI, N. ***An Agent-oriented programming model for SOA & Web Services***. *Industrial Informatics, 5th IEEE International Conference on.* Vol 2. 2007.

RUSSEL, S. e NORVIG, P., **Inteligência Artificial**. 2ªEd, Tradução da segunda edição. Rio de Janeiro: Elsevier, 2004.

SCHEUTZ, M., SLOMAN, A., LOGAN, B.. ***Emotional states of realistic agent behaviour***. *In: Proceedings of GameOn*, pg.81-88. Citeseer. 2000.

SENSOY, M. and YOLUM, P. ***Evolving service semantics cooperatively: a consumer-driven approach***. *Springer Science+Business Media, LLC*, Nov 9, 2008.

SCHIAFFINO, S., AMANDI A.; ***Polite Personal Agents***. *in IEEE Intelligent Systems*, p12-18. Jan-Fev 2006.

SIMON, H.A. ***Why should machines learn. In Machine learning: an artificial intelligence approach***. Vol 1. Morgan Kaufmann Pub. 1983.

SINGH, M. P. and HUHNS, M. N. ***Service-Oriented Computing: Semantics, Processes, Agents***. John Wiley & Sons, New York, NY, EUA, 1a. Edição. 2005.

STOJANOVIC, Z.; DAHANAYAKE, A. ***Service-oriented software system engineering: challenges and pratices***. *Idea Group Inc. Hershely, PA*. 2005.

SWITHINBAK, P.; GIGANTE, F.; PROCTOR, H.; RATHORE, M.; et al. ***WebSphere and .Net Interoperability using Web Services***. *IBM RedBooks*: IBM. 2005.

VERNADAT, F.B. ***Enterprise modeling and integration: principles and applications***. *London, Chapman & Hall.* 1996.

VIEIRA, W. ***Agentes Móveis Adaptáveis para Operação Remota***. *PhD Thesis,* Universidade Nova de Lisboa, Lisboa, 2000.

WANG, H., HUANG, J. Z., QU, Y., e XIE, J. ***Web services: problems and future directions***. *Journal of Web Semantics*, 1(3):241–320. 2005.

WEISS, G. ***Multiagent systems: A Modern Approach to Distributed Artificial Intelligence***. *MIT Press,* 1999.

WHITE, S. ***Using BPMN to model BPEL process***. In: BPTrends, vol.3, num.3, pg.1-18. 2005.

WONG, J. S. K. and MIKLER, A. R., ***Intelligent mobilet agents in large distributed a*****utonomous cooperative systems**. The Journal of Systems and Software 47, pag.75-87, 1999.

YARIMAGAN, Y.; DOGAC, A. ***A semantic based solution for the interoperability of UBL schemas***. *submitted for publication*. 2008.

ZAMBIASI, Saulo P. e RABELO, Ricardo J. **Virtualização de Colaboradores na Manufatura: um modelo baseado em Agent Bots**.

In: Simpósio Brasileiro de Automação Inteligente, Florianópolis. Simpósio Brasileiro de Automação Inteligente, 2007.

ZAMBIASI, S.P.; RABELO R.J.. **Uma arquitetura de referência para softwares assistentes pessoais baseada na arquitetura orientada a serviços**. In I2TS'2010: *9th International Information and Telecommunication Technologies Symposium*, Rio de Janeiro (RJ), Brasil, 2010.

ZAMBIASI, S.P; RABELO, R.J. **Uma Arquitetura de Referência para Softwares Assistentes Pessoais Baseada em Agentes e SOA**. In WESAAC'2011: VII Workshop - Escola de Sistemas de Agentes, seus Ambientes e Aplicações, Curituba (PR), Brasil. 2011.

Zhu, H. ***Formal specification of agent behaviour through environment scenarios***. *In: Lecture notes in computer science*. pg.263-277. Springer. 2001.

Especificações, Padrões e Documentos Técnicos

ARKIN, A.; ASKARY, S.; FORDIN, S.; at al. ***Web Service Choreography Interface (WSCI) 1.0***. Disponível em: <*http://www.w3.org/TF/wsci/*>. Acessado em Abril/2009. 2002.

BELLWOOD, T.; EHNEBUSKE, D.; HUSBAND, Y. L.; KARP, A.; et al. ***UDDI Version 2.03 Data Structure Reference*****.** Disponível em: <*http://uddi.org/pubs/DataStructure_v2.htm*>. Acessado em Maio/2009. 2002.

BOOTH, D.; HAAS, H.; MCCABE, F.; NEWCOMER, E.; et al. ***Web Services Architecture***. Disponível em: <http://www.w3.org/TR/ws-arch/> Acessado em Fevereiro/2009. 2004.

BOSAK, J.; MCGRATH, T.; HOLMAN, G.K. ***Universal business language v2.0.*** Disponível em: <*http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.html*>. Acessado em Maio/2009. 2006.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C.M.; at al. ***Extensible Marckup Language (XML)***. *W3C Recomendation.* Disponível em: <*http://www.w3.org/TR/xml*>. Acessado em Março/2009. 2008.

BRYAN, D.; DRALUK, V.; EHNEBUSKE, D.; GLOVER, T.; et al. ***UDDI Version 2.04 API Specification.*** Dipsonível em: <*http://uddi.org/pubs/ProgrammersAPI_v2.htm*>. Acessado em Março/2009. 2002.

CLEMENT, L., HATELY, A., VON RIEGEN, C., e ROGERS, T. **UDDI Version 3.0.2**. Disponível em: <*http://uddi.org/pubs/uddi_v3.htm*>. Acessado em Abri/2009. 2004.

CLEMENT, L.; KONING, D.; MEHTA, V.; at al. ***WS-BPEL extension for people (BPEL4PEOPLE) Specification 1.1.*** Disponível em: <*http://www.oasis-open.org/committees/download.php/32776/rev130.zip*>. Acessado em Julho/2009. 2009.

CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEERAWARANA, S. ***Web Services Description Language (WSDL) 1.1.*** <*http://www.w3.org/TR/wsdl*>. Accessed in Mar/2009. 2001.

ESTEFAN, J. A.; LASKEY, K.; MCCABE, F.; THORNTON, D. ***Reference Architecture for Services Oriented Architecture Version 1.0***. *OASIS.* Disponível em: <*http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf*>. Acessado em: 10 dez. 2008.

ebXML, ***Electronic Business using eXtensible Markup Language***. Disponível em <http://www.ebxml.org/> Acessado em Junho/2011.

GUDGIN, M., HADLEY, M., MENDELSOHN, N., MOREAU, J., e NIELSEN, H. F.; 2003. ***SOAP Version 1.2.*** <*http://www.w3.org/TR/soap/*> acessado em Nov/2008. 2007.

HAAS, H; e BROWN, A.; 2004. **Web Services Glossary**. Disponível em: <*http://www.w3.org/TR/ws-gloss/*>, Acessado em Dez/2008.

MACKENZIE, C.; LASKEY, K.; MCCABE, F. at all. ***Reference Model for Service Oriented Architecture 1.0***. *OASIS Standard, 12 October 2006*. Disponível em: <*http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf*>. Acessado em Fevereiro/2009.

OMG. ***OMG IDL Syntax and Semantics*****.** Disponível em: <*http://www.omg.org/cgi-bin/doc?formal/02-06-39*>. Acessado em Janeiro/2009. 2002.

ORT, E.; MANDAVA, R. ***Web Services Developer Pack Part 1: Registration and the JAXR API***. *The Java Web Services Developer Pack.* <*http://java.sun.com/developer/technicalArticles/WebServices/WSPack*> . Accessed in Mai/2009. 2002.

THOMPSON, H. S., BEECH, D., MALONEY, M., e MENDELSOHN, N.; 2004. ***XML Schema (2nd Edition)***. Disponível em: <*http://www.w3.org/TR/xmlschema-1/*>. Acessado em Abril/2006. 2004.

UBL, **Universal Business Language.** Disponível em <*http://www.oasis-open.org/committees/ubl/*> Acessado em Junho/2011.

Web Sites de Empresas, Projetos, Tecnologias e outros

APACHE. **Apache http server project**. Available in: <*http://httpd.apache.org*>. Accessed in Ago 2010.

APPLE, **Apple Inc**. Available in: <*http://www.apple.com*>. Accessed in Mar 2011.

BLOGGER. **Blogger**. Available in: <*http://www.blogger.com/*>. Acessado em Outubro/2010.

CALO, *CALO Project*, Disponível em: <*http://caloproject.sri.com/darpa/*>. Acessado em Março/2009.

CLEVERBOT. **Cleverbot wins Machine Intelligence Prize**. Campbridge, Dec 15, 2010. Disponível em: <*http://www.cleverbot.com/machine*>. Acessado em Março/2011.

COIN. ***Coin: enterprise collaboration & interoperability***, Disponível em: <http://www.coin-ip.eu>. 2009. Acessado em Junho/2011.

COMCAST. **Concast Corporation**. Available in: <*http://www.comcast.com/*>. Accessed in Mar 2011.

DORNFEST, Rael. ***A fork in the road (an important announcement about i want sandy)***. Disponível em: <*http://getsatisfaction.com/iwantsandy/topics/a_fork_in_the_road_an_important_announcement_about_i_want_sandy*>, acessado em Fevereiro/2011.

EMPSON, R. ***Plaxo goes back to begin a smart address book, launches virtual assistant. TechCrunch***. Disponível em: <*http://techcrunch.com/2011/03/16/plaxo-goes-back-to-being-a-smart-address-book-launches-virtual-assistant/*>. Acessado em Março/2011.

GMAIL, **Google mail**. Disponível em: <*http://mail.google.com*>. Acessado em Agosto/2010.

GTALK, **Google Talk**. Disponível em: <*http://www.google.com/talk/*>. Acessado em Agosto/2010.

JADE. ***Java Agent Development Framework***. Disponível em: <*http://jade.tilab.com/*>. Acessado em Abril/2011.

JSON. ***Introducing JSON***. Disponível em: <*http://www.json.org*>. Acessado em Abril/2011.

MANN, Holly. ***Free Personal Assistant - Meet Sandy***. Disponível em <*http://honestholly.com/free-personal-assistant-meet-sandy/*>. Acessado em Fevereiro/2011.

MICROSOFT. **Microsoft**. Dipsonível em: <*http://www.microsoft.com*>. Acessado em Março/2011.

MUNDIE, Criag. ***Microsoft to Make Computers "Humanistic" ...meet Laura...: Interview on Beet.TV***. October, 2008. Disponível em: <*http://www.youtube.com/watch?v=6GOrNU6e_og*>, acessado em Março/2009.

OAUTH. **Oauth**. Disponível em: <*http://oauth.net/*>. Acessado em Novembro/2010.

OUTLOOKPA, ***Outlook personal assistant - email manager***. Disponível em: <*http://outlookpa.com/*>. Acessado em Fevereiro/ 2011.

PAL, **PAL Project**, Disponível em: <*http://pal.sri.com/*>. Acessado em 14 de Fevereiro de 2011.

PHP. ***PHP: Hypertext Preprocessor***. Disponível em: <*http://br.php.net*>. Acessado em Agosto/2010.

PIDGIN. **Pidgin**. Disponível em: <*http://www.pidgin.im/*>. Acessado em Setembro/2010.

PLAXO, **Plaxo Inc**. Disponível em: <*http://www.plaxo.com*>. Acessado em Março/2011.

POSTGRESQL, **Postgresql project**. Disponível em: <*http://www.postgresql.org*>. Acessado em Agosto/2010.

RosettaNet, Disponível em <http://www.rosettanet.org/>. Acessado em Junho/2011.

SANDY. ***I Want Sandy***. Disponível em: <*http://www.iwantsandy.com/*>. Acessado em Fevereiro/2011.

SEEKDA. ***Seekda: web services***. Disponível em: <*http://webservices.seekda.com/*>. Acessado em Novembro/2010.

SIRI, ***SIRI application***. Disponível em: <*http://siri.com/*>. Acessado em Fevereiro/2011.

SHELLTOYS. ***Personal assistant - day planner and personal information manager***. Disponível em: <*http://www.shelltoys.com/personal_assistant/*>. Acessado em Fevereiro/2011.

SMACK. ***Smack API***. Disponível em: <*http://www.igniterealtime.org/projects/smack/index.jsp*>. Acessado em Setembro/2010.

STORMSOURCE, Inc. ***Virtual assistant manager***. Disponível em: <*http://www.virtualassistantmanager.com/*>. Acessado em Fevereiro/2011.

TEKEVER, ***Web Services Tekever.eu: Serviços de informação disponibilizados via Web Services***. Disponível em: <*http://webservices.tekever.eu/*>. Acessado em Mar de 2011.

THENAULT, Sylvain. **Narval-moved**. Logilab Project. Disponível em: <*http://www.logilab.org/908/*>, Acessado em 14 de Fevereiro de 2011.

THUNDERBIRD, **Thunderbird: Mozilla website**. Disponível em: <*http://br.mozdev.org/thunderbird/*>. Acessado em Setembro/2010.

TIVO, **TiVo Inc**. Disponível em: <*http://www.tivo.com/*>. Acessado em Março/2011.

TWITTER. **Twitter**. Disponível em: <*http://twitter.com*>. Acessado em Outubro/2010.

TWITTER. ***Twitter Development***. Disponível em: <*http://dev.twitter.com/*>. Acessado em Outubro/2010b.

VANCE, Ashlee. ***Microsoft Mapping Course to a Jetsons-Style Future*.** New York Times, March 1, 2009. Disponível em: <*http://www.nytimes.com/2009/03/02/technology/business-computing/02compute.html?scp=1&sq=jetsons&st=cse*>. Acessado em Fevereiro/2011.

8 APÊNDICE A – OUTROS SERVIÇOS WEB DE UTILIDADE PARA OS ASSISTENTES PESSOAIS

Neste ponto são descritas algumas classes criadas pelo autor da Tese que fornecem facilidades para o desenvolvimento dos aplicativos. Também são descritos alguns serviços que são de utilidade para se compor comportamentos no assistente pessoa, e para outros comportamentos. É importante se observar que, conforme foi definido pelo autor do presente trabalho e desenvolvedor da instância de implementação, as operações efetuadas nos comportamentos são chamadas à serviços web, extrapolando a ideia de que tudo é serviço web. Dessa forma, mesmo operações simples como operações matemáticas e manipulação de *strings*, por exemplo, são efetuadas por chamadas à serviços web.

8.1 CLASSE *KQMLMESSAGE*

A classe *KQMLMessage* manipula uma estrutura de mensagem KQML sob a sintaxe XML (Quadro 5), definida aqui como KQML/XML, e serve para criar mensagens nessa estrutura, extrair informações da mensagem e manipulá-las.

Contudo, a mensagem KQML foi modificada, acrescentando dois atributos, *id* para que o sistema possa ter a possibilidade de identificar uma mensagem – pode ser utilizada opcionalmente por algumas aplicações – e *status*, que se refere ao tipo de mensagem quanto a ser pública ou privada. Esse atributo se torna útil no ISAP, no momento em que o ISAP precisa decidir qual o meio onde a mensagem deve ser enviada para o usuário.

```
<message>
        <id>343</id>
        <type>request</type>
        <sender>Compra-altera</sender>
        <receiver>saulopz@gmail.com</receiver>
        <in_replay_to></in_replay_to>
        <language></language>
        <ontology></ontology>
        <status>public</status>
        <content>
        Ordem 31 modificada: 10 produto(s) 101 a R$ 25.00
        para 5 produto(s) 101 a R$ 28.00.
        confirmar ou cancelar?
        </content>
</message>
```

Quadro 5: Exemplo de mensagem KQML/XML.

O estilo de mensagem KQML/XML definido para ser utilizado em algumas aplicações neste protótipo possui os seguintes atributos:

- ***id*:** Opcional. Valor numérico para que algumas aplicações possam identificar a mensagem;
- ***type*:** Segundo RusselRussel e Norvig e Norvig (2004), os tipos de mensagens trocadas entre agentes são: *information, ask, response, request, promess, acknoledgement* e *share*. Contudo na implementação foram apenas utilizados os tipos: *information, ask* e *response*. Isso não quer dizer que os outros tipos de mensagens não possam ser utilizados por aplicações que necessite;
- ***sender*:** Remetente da mensagem. Em alguns casos, na implementação, quando a mensagem parte do GAP, o remetente é o nome do comportamento;
- ***receiver*:** Destinatário da mensagem. Em alguns caso, o usuário pode enviar uma mensagem diretamente para um comportamento, sendo ele o *receiver* da mensagem;
- ***in_replay_to*:** Opcional. Identifica que essa mensagem é resposta a uma outra mensagem. Neste atributo pode ser utilizado o *id* da mensagem que foi a mensagem de *request*, ou *ask*;
- ***language*:** Opcional. Identifica o tipo da linguagem utilizada pelo conteúdo da mensagem;
- ***ontology*:** Opcional. Identifica a ontologia utilizada pelo conteúdo da mensagem;

- ***status*:** Opcional. Identifica se a mensagem é pública ou privada, no caso do GAP. Este atributo pode ser utilizado para outros fins, em outras aplicações;
- ***content*:** Conteúdo da mensagem.

Esta classe foi desenvolvida tanto em PHP como em Java, de modo que os programas desenvolvidos nessas linguagens, nesse protótipo, pudessem trocar as mensagens nessa estrutura. A classe possui os atributos referentes a cada um dos atributos da mensagem, incluindo os ***getters*** e ***setters*.** Possui dois construtores, um que não é passado nenhum parâmetro, para caso da criação de uma nova mensagem, e outro que o usuário passa como parâmetro o XML de uma mensagem KQML/XML. Há um método chamado ***isValid()*** que verifica se o XML passado é uma mensagem válida nessa estrutura e um método chamado ***toXML()***, que pega as informações da classe e as transforma em um XML no formato KQML/XML aqui definido.

8.2 CASE *MYMESSAGE*

A classe *MyMessage* também é implementada em PHP e Java. Essa classe é utilizada para a criação da mensagem estilo tabela *MyMessageTable*. Sua função é pegar um conjunto de informações e montá-las sequencialmente com o uma tabela, separando as informações por dois pontos (:). Quando esse carácter especial é encontrado, ele é substituído por ***&dts;*** para não haver problemas nas informações. Seus métodos são:

- ***Constructor* com parâmetro:** cria uma instância da classe na memória com uma mensagem nessa estrutura sendo passada como parâmetro. As informações podem então serem extraídas com os outros métodos;
- ***Constructor* sem parâmetro:** uma nova instância é criada com uma mensagem vazia, sem informação nenhuma;
- ***add*:** Adiciona uma informação na mensagem;
- ***next*:** pega uma informação da mensagem da mensagem e move o ponteiro de indexação para a próxima informação;
- ***reset*:** move o ponteiro de indexação para a primeira informação da mensagem;
- ***get*:** recupera uma informação na posição específica, passada por parâmetro;

- ***size*:** retorna a quantidade de informações na mensagem;
- ***toString*:** retorna a mensagem no formato especificado, separando as informações por dois pontos (**:**).

Tanto a classe, quanto o formato da mensagem é bastante simples, mas útil para algumas aplicações que desejam enviar uma lista de informações como resultado, e não apenas uma informação única. Essa classe é utilizada principalmente pelo serviço web de interoperabilidade *wsEstoque*, pela classe *myMessageTable* e para a comunicação via soquetes do *XMPPConnector*. A classe foi inicialmente desenvolvida para ser utilizada na troca de mensagens via soquetes neste último elemento. Contudo, se mostrou útil para outras aplicações, das quais acabou sendo utilizada.

8.3 CLASSE *MYSOAP*

A classe *MySoap* foi desenvolvida pelo autor da Tese de modo a facilitar a utilização de SOA/SOAP na linguagem PHP. Sua principal função é criar um arquivo WSDL com as operações e seus respectivos parâmetros quando, e retornar ao usuário quando requisitado. O ***constructor*** da classe apenas inicializa as variáveis utilizadas. O método ***register*** registra uma operação no WSDL. Seus parâmetros são o nome da operação, uma lista com os parâmetros de entrada, o parâmetro de saída o estilo de codificação e uma descrição da operação. O método ***execute*** monta tudo numa *string* com o conteúdo do WSDL e o ***getWSDL()*** retorna o resultado como uma *string*. Um exemplo de utilização da classe é apresentada no Quadro 6:

```
<?php
require_once('MySoap.php');
$mySoap = new MySoap();

function concat ($a, $b) {
       return $a.$b;
}

$mySoap->register ('concat',
       array('a' => 'xsd:string', 'b' => 'xsd:string'),
       array ('return' => 'xsd:string'),
       'encoded', 'Concat two strings');

$mySoap->execute();
?>
```

Quadro 6: Exemplo de PHP utilizando classe MySoap.

Quando uma chamada a uma operação em um serviço web é feita, a operação é chamada e o resultado enviado como resposta. Quando o usuário digita o link em um navegador, com **"?wsdl"** depois do link, então é apresentado para o usuário o WSDL do serviço web.

8.4 CLASSES *REQUEST*

A classe *Request* é utilizada pela linguagem PHP, para recuperar as informações enviadas de uma página a outra via POST e GET. Essa classe é chamada no início do *script*, recuperando as informações e as colocando como atributos da sua própria classe. Quando o *constructor* é chamado, todas as informações passadas como POST ou GET, se transformam em atributos públicos da classe, e podem ser acessados como tal.

8.5 CLASSE *SESSION*

A classe *Session* também foi criada para trabalhar com a linguagem PHP, armazenados informações que devem existir durante toda a seção de acesso do usuário. Quando o *constructor* é chamado, uma nova seção é criada, ou a seção atual é recuperada. O método ***done*** é chamado para destruir uma seção, quando a mesma não é mais

necessária. O método ***set*** é chamado para criar um novo atributo na seção, passando como parâmetros o nome da seção na forma de uma *string*, e o valor do atributo. As informações da seção também podem ser recuperadas, tal como na classe *Request*, como atributos públicos da classe.

8.6 CLASSE *TEMPLATE*

A classe *Template* é de implementação bastante simples, mas de grande utilidade para o desenvolvimento de aplicações em PHP, separando por quase completo a interface HTML da linguagem de programação PHP. Foi desenvolvida pelo próprio autor da Tese em 2005 e utilizada nas implementações das interfaces web da instância de implementação devido sua praticidade e vantagens em tornar o desenvolvimento mais rápido.

Para a utilização desse recurso, existem dois elementos que devem ser criados em um documento HTML, o elemento ***template***, e o elemento ***variable***. Para criar um elemento *template*, é necessário a criação de um bloco, na forma de um comentário HTML, que inicia com:

<!--tpl:NomeDoTemplate-->

e termina com:

<!--/tpl:NomeDoTemplate-->.

O código HTML entre essas tags de comentários são os *templates* utilizados pela classe. As variáveis, são elementos definidos por colchetes, tal como ***{nomeDaVariavel}***, e que são substituídas por outra informação quando requisitada.

Os métodos da classe são os seguintes:

- ***Constructor*:** passando como parâmetro um texto que é referente à um arquivo HTML, normalmente é requisitado da classe *FileTemplate*. O segundo parâmetro é o nome do template;

- ***assignVar*:** Substitui a variável com o nome passado no primeiro parâmetro, pelas informações contidas no segundo parâmetro;
- ***assignTemplate*:** substitui todo o *template* de nome passado no primeiro parâmetro, pelas informações passadas no segundo parâmetro. Normalmente, busca-se um *template*, altera as variáveis, e então substitui o *template* não trabalhado, pelo atualizado;
- ***get*:** retorna em forma de uma *string* todo o conteúdo que corresponde ao *template*.

A classe *FileTemplate* apenas pega o parâmetro que é passado, com o nome do arquivo HTML, e cria uma *string* para conter o conteúdo do arquivo. O método ***get*** retorna o documento no formato de *string*. Em um *script* HTML, diversos *templates* podem precisar serem utilizados. A intenção do *FileTemplate* é que não seja necessário ler e recuperar as informações de um arquivo toda a vez que um novo *template* necessitar ser trabalhado. Apenas recuperam-se as informações originais de *FileTemplate*.

Como exemplo da utilização da classe, pega-se como exemplo o arquivo HTML com o conteúdo apresentado no Quadro 7.

```
<!--tpl:Teste-->
<html><head><title>teste</title></head>
<body>
        <h1>{titulo}</h1>
        <p>{conteudo}</p>
</body>
</html>
<!--/tpl:Teste-->
```

Quadro 7: Documento HTML com informações para o Template.

A utilização desse *template* pela linguagem PHP é apresentada no Quadro 8.

```
<?php
        $ft = new FileTemplate('arquivo_template.html');
        $tpl = new Template($ft->get(), 'Teste');
        $tpl->assignVar('titulo', 'Testando <i>Templates</i>');
        $tpl->assignVar('conteudo', 'Alo mundo!');
        echo $tpl->get();
?>
```

Quadro 8: Exemplo de linguagem PHP utilizando um template.

Também é interessante observar que o documento pode possuir *templates* aninhados. Além disso, a grande vantagem da utilização dos *templates* é a separação do código da apresentação. Pode-se assim modificar toda a apresentação da aplicação sem a necessidade de se modificar o código PHP, e vice-versa.

8.7 CLASSE *SEARCHENGINE*

A classe *SearchEngine* foi desenvolvida em PHP e utilizada para o *chatbot* desenvolvido para o protótipo. A sua função é verificar se uma expressão, composta com coringas definidos pelo sinal de porcentagem (**%**) fecha com um determinado texto. Os métodos da classe são.

- ***Constructor*:** Passa-se como parâmetro o texto da qual quer se encontrar uma expressão que fecha com o texto;
- ***setText*:** Altera o texto que será efetuada a pesquisa;
- ***match*:** Passa-se como parâmetro a expressão. Retorna verdadeiro se foi encontrada no texto e falso se não.

A forma como é definida a expressão é melhor explicada quando apresentado o *chatbot*.

8.8 SERVIÇO WEB KQML

No caso da implementação em questão, por opção do desenvolvedor e autor do presente trabalho, a comunicação com o ISAP se dá via mensagens no formato KQML. Dessa forma, foi necessária a criação/utilização de um serviço web que trabalhe com tal formato. Ainda, lembrando que o formato KQML aqui utilizado é modificado com o acréscimo de dois atributos: *id* e *status*. Ainda, nessa implementação, a mensagem KQML é colocada sob o formato XML. Todas as operações, com exceção da última, do serviço web em questão precisam que seja enviada a mensagem como parâmetro de entrada, de forma a extrair a informação requerida. São elas:

- ***valid*:** retorna 0 se for uma mensagem KQML sob XML bem formatada, conforme definida nessa implementação;
- ***getKQMLId*:** retorna o atributo id;
- ***getKQMLType*:** retorna o tipo da mensagem;

- ***getKQMLSender*:** retorna o remetente;
- ***getKQMLReceiver*:** retorna o destinatário;
- ***getKQMLInReplayTo*:** retorna qual mensagem se está respondendo;
- ***getKQMLLanguage*:** retorna a linguagem específica do conteúdo da mensagem;
- ***getKQMLOntology*:** retorna a ontologia do conteúdo da mensagem;
- ***getKQMLStatus*:** retorna se o estado da mensagem, ou seja, se é *public* ou *private*. Esse atributo foi acrescentado pelo desenvolvedor, devido a necessidades específicas;
- ***getKQMLContent*:** retorna o conteúdo da mensagem;
- ***toXML*:** por meio do envio dos parâmetros correspondentes aos atributos de uma mensagem KQML definida aqui, é montado e retornado uma *string* como um documento XML correspondente. Os parâmetros sevem ser passados na seguinte ordem: *id, type, sender, receiver, in_replay_to, language, ontology, status, content*.

O atributo *status* desse KQML modificado, é necessário para o ISAP "saber" se uma mensagem deve ser enviada de forma privada apenas para seu usuário, ou se é algo que pode ou deve ser divulgado publicamente pelos serviços que o ISAP se utiliza.

8.9 SERVIÇO WEB *MATH*

O serviço web *math* tem o objetivo de efetuar operações matemáticas a partir de parâmetros enviados na chamada das operações. Elas são, nessa implementação, todas operações com dois parâmetros de entrada, descritas a seguir:

- ***sum*:** efetua a soma de dois valores passados como parâmetros de entrada;
- ***minus*:** retorna a diminuição do primeiro pelo segundo;
- ***multiply*:** multiplica os dois parâmetros de entrada;
- ***divide*:** divide o primeiro parâmetro pelo segundo;
- ***rest*:** divide o primeiro parâmetro pelo segundo e retorna o resto da operação da divisão;

- ***increment*:** pega o primeiro parâmetro e incrementa em um. Quando o valor máximo (segundo parâmetro) for alcançado, então o valor retornado é 0 (zero);
- ***decrement*:** pega o primeiro parâmetro e decrementa em um. Quando o valor for menor que zero, então retorna o valor máximo (segundo parâmetro).

Essas foram apenas operações básicas que possivelmente seriam utilizadas para a implementação do estudo de caso apresentado. Novas e mais complexas operações podem ser acrescentadas se necessário, ou pode-se utilizar algum serviço web de operações matemáticas na Internet, se existente.

8.10 SERVIÇO WEB *STRING*

O serviço web *string* é utilizado para gerenciar informações na forma de texto e operações que podem ser úteis para serem utilizadas na modelagem de comportamentos no GAP. As operações atualmente implementadas são:

- ***upper*:** transforma uma cadeia de caracteres em caracteres caixa alta.
  - ***Input*:** cadeia de caracteres que se quer modificar.
  - **Retorno:** Retorna a cadeia de caracteres transformada em caixa alta.
  - **Exemplo:** se for enviada a *string* "Assistente Pessoal", então será retornado "ASSISTENTE PESSOAL".
- ***lower*:** tem o efeito contrário da operação *upper*, retornando os caracteres em caixa baixa.
  - ***Input*:** cadeia de caracteres a efetuar a operação.
  - **Retorno:** retorna a cadeia de caracteres em caixa baixa.
  - **Exemplo:** ao enviar "Assistente Pessoal", retorna "assistente pessoal".
- ***size*:** informa a quantidade de caracteres que uma cadeia de caracteres possui.
  - ***Input*:** cadeia de caracteres.
  - **Retorno:** quantidade de caracteres.
  - **Exemplo:** ao enviar "Assistente Pessoal", retorna o valor 18. O espaço também é contado como carácter.

- ***concat*:** concatena duas cadeias de caracteres.
  - **A:** primeira cadeia de caracteres.
  - **B:** segunda cadeia de caracteres.
  - **Retorno:** retorna uma cadeia de caracteres que é a **A** seguida da **B**.
  - **Exemplo:** Se enviado no primeiro parâmetro "Assistente" e no segundo " Pessoal", retorna "Assistente Pessoal". Observar o espaço no início da segunda cadeia de caracteres.
- ***replace*:** substitui uma cadeia de caracteres por outra em um texto. Toda *string* que fechar com *Search* deve ser trocada por *Replace*.
  - ***Search*:** *string* da a procurar.
  - ***Replace*:** *string* que se quer substituir.
  - ***Text*:** texto em que as *strings* devem ser substituídas.
  - **Exemplo:** Ao enviar os parâmetros "assistente pessoal", "assistente" e "meu assistente pessoal me auxilia", então é retornado "meu assistente me auxilia".
- ***nextWord*:** procura por uma palavra em uma *string*, retornando a palavra que vem a seguir.
  - ***Text*:** texto a procurar a palavra.
  - ***Search*:** palavra a procurar.
  - **Exemplo:** Ao enviar o texto "confirmar ordem codigo 45 hoje" e a palavra "codigo" para procurar, então será retornada a *string* "45".
- ***nextString*:** faz o mesmo que *nextWord*. Porém, retorna a *string* toda que vem após *Search*.
  - ***Text*:** texto a procurar a palavra.
  - ***Search*:** palavra a procurar.
  - **Exemplo:** Ao enviar o texto "confirmar ordem codigo 45 hoje" e a palavra "codigo" para procurar, então será retornada a *string* "45 hoje".

Essas operações de manipulação de *strings* são necessárias nessa instância de implementação do GAP, pois na mesma é extrapolada a utilização de serviços web e nenhuma operação é vista senão como serviço web.

8.11 SERVIÇO WEB *DATETIME*

O serviço web *datetime* possui apenas duas operações muito simples, mas que podem ser muito úteis:

- ***getLocalDate*:** retorna a data atual no formato ano-mês-dia.
  - **Exemplo:** 2011-02-23.
- ***getLocalTime*:** retorna a hora atual no formato hora:minuto:segundo.
  - **Exemplo:** 19:23:34.

Nenhuma das operações requerem parâmetros de entrada.

8.12 SERVIÇO WEB MYMESSAGETABLE

Este serviço web foi planejado e desenvolvido pelo próprio autor do presente trabalho e desenvolvedor da implementação. O objetivo desse serviço é tratar mensagens no formato de uma tabela simplificada. As informações são separadas pelo carácter "**:**" e possui a seguinte estrutura:

- A primeira informação é numérica e se refere a quantas colunas a tabela possui;
- As seguintes informações são as *x* colunas definidas pela primeira informação com seus respectivos nomes;
- As linhas da tabela são as seguintes *x* informações e assim por diante;

Se, por exemplo, pegar a *string* no formato específico:

3:Código:Atributo:Valor:1:id:45:2:nome:Arisa:3:função:Assistência Pessoal

e transformá-la em tabela, conforme definido aqui, então fica conforme a Tabela 21:

Código	Atributo	Valor
1	id	45
2	nome	Arisa
3	função	Assistência Pessoal

Tabela 21: Mensagem MyMessageTable formatada como tabela.

As operações do serviço web para extração de informações da mensagem em forma de tabela são as seguintes:

- ***getColumnSize*:** a partir da mensagem no formato específico passado por parâmetro, retorna a quantidade de colunas da tabela;
- ***getLineSize*:** a partir da mensagem no formato específico passado por parâmetro, retorna a quantidade de linhas da tabela;
- ***getColumnName*:** a partir da mensagem no formato específico passado por parâmetro e da posição da coluna, retorna o nome da coluna;
- ***getElement*:** a partir da mensagem no formato específico passado por parâmetro, da linha e da coluna, retorna o valor da célula requerida;

Caso um carácter ":" seja encontrado, por ser um carácter especial dessa estrutura de mensagem de tabela, ele é substituído por um carácter especial, de forma a não haver problemas com a estrutura da mensagem.

9 APÊNDICE B – *LOGS* DOS COMPORTAMENTOS

9.1 *LOGS* DO COMPORTAMENTO COMPRA-ORDEM

(Sun Mar 27 21:25:49 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:25:50 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 21
(Sun Mar 27 21:25:52 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 5
(Sun Mar 27 21:25:53 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:25:55 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 3]: return 17
(Sun Mar 27 21:25:57 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 4]: return 80
(Sun Mar 27 21:25:58 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 5]: goto 5
(Sun Mar 27 21:26:00 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 6]: return 01.000.000/000-01
(Sun Mar 27 21:26:02 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 7]: return 62
(Sun Mar 27 21:26:03 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 8]: goto 8
(Sun Mar 27 21:26:05 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 9]: return 1
(Sun Mar 27 21:26:06 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 10]: return 101
(Sun Mar 27 21:26:08 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 11]: return 25.00
(Sun Mar 27 21:26:10 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 12]: return 1
(Sun Mar 27 21:26:11 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 13]: return A ordem 80,do fornecedor John Doe Ltda., encontra-se aberta, com: 17 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:26:13 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 14]: goto 14
(Sun Mar 27 21:26:14 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 15]: return 1
(Sun Mar 27 21:26:16 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 16]: return <?xml version="1.0"?>
(Sun Mar 27 21:26:18 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 17]: return 1
(Sun Mar 27 21:26:25 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:26:27 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 21
(Sun Mar 27 21:26:28 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:26:30 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 21
(...)
(Sun Mar 27 21:27:46 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:48 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 21
(Sun Mar 27 21:27:49 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 5
(Sun Mar 27 21:27:51 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:27:52 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 3]: return 17
(Sun Mar 27 21:27:54 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 4]: return 81
(Sun Mar 27 21:27:56 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 5]: goto 5
(Sun Mar 27 21:27:57 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 6]: return 01.000.000/000-02
(Sun Mar 27 21:27:59 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 7]: return 63
(Sun Mar 27 21:28:01 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 8]: goto 8
(Sun Mar 27 21:28:02 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 9]: return 1
(Sun Mar 27 21:28:04 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 10]: return 101
(Sun Mar 27 21:28:05 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 11]: return 28.00
(Sun Mar 27 21:28:07 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 12]: return 1
(Sun Mar 27 21:28:08 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 13]: return A ordem 81, do fornecedor Sora konpyuuta, encontra-se aberta, com: 17 produto(s) 101 a R$ 28.00,
(Sun Mar 27 21:28:10 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 14]: goto 14
(Sun Mar 27 21:28:12 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 15]: return 1
(Sun Mar 27 21:28:13 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 16]: return <?xml version="1.0"?>
(Sun Mar 27 21:28:15 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 17]: return 1
(Sun Mar 27 21:28:22 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:28:24 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 21
(Sun Mar 27 21:28:25 BRT 2011) DEBUG [Behavior(9) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:28:27 BRT 2011) DEBUG [Behavior(9) activity.ConditionalStatic(conditional static) 2]: goto 21

9.2 *LOGS* DO COMPORTAMENTO COMPRA

(Sun Mar 27 21:25:53 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:25:55 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:25:57 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 80
(Sun Mar 27 21:25:58 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2

(Sun Mar 27 21:26:00 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return null
(Sun Mar 27 21:26:02 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return inexistente
(Sun Mar 27 21:26:03 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:26:05 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:26:08 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:26:10 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:26:11 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 80
(Sun Mar 27 21:26:13 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:26:15 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 62
(Sun Mar 27 21:26:16 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return aberta
(Sun Mar 27 21:26:18 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:26:19 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:26:22 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:26:24 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:26:25 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 80
(Sun Mar 27 21:26:27 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:26:29 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 62
(Sun Mar 27 21:26:30 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return modificada
(Sun Mar 27 21:26:32 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:26:33 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 18
(Sun Mar 27 21:26:35 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 19]: return aberta
(Sun Mar 27 21:26:36 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 20]: goto 20
(Sun Mar 27 21:26:38 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 21]: return <ordem>
(Sun Mar 27 21:26:39 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 22]: return Ordem 80 modificada: 17 produto(s) 101 a R$ 25.00 para 10 produto(s) 101 a R$ 25.00.
(Sun Mar 27 21:26:41 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 23]: goto 23
(Sun Mar 27 21:26:42 BRT 2011) DEBUG [Behavior(12) activity.AssignStatic(assign static) 24]: aux = confirmar ou cancelar?
(Sun Mar 27 21:26:44 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 25]: return Ordem 80 modificada: 17 produto(s) 101 a R$ 25.00 para 10 produto(s) 101 a R$ 25.00. confirmar ou cancelar?
(Sun Mar 27 21:26:45 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 26]: return <?xml version="1.0"?>
(Sun Mar 27 21:26:47 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 27]: return 1
(Sun Mar 27 21:26:58 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:00 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:27:01 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 80
(Sun Mar 27 21:27:03 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:27:04 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 62
(Sun Mar 27 21:27:06 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return modificada
(Sun Mar 27 21:27:07 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:27:09 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 18
(Sun Mar 27 21:27:10 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 19]: return modificada
(Sun Mar 27 21:27:12 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 20]: goto 29
(Sun Mar 27 21:27:16 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:18 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:27:20 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 80
(Sun Mar 27 21:27:21 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:27:23 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 62
(Sun Mar 27 21:27:25 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return modificada
(Sun Mar 27 21:27:26 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:27:28 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 18
(Sun Mar 27 21:27:29 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 19]: return modificada
(Sun Mar 27 21:27:30 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 20]: goto 29
(Sun Mar 27 21:27:35 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:37 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:27:38 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 80
(Sun Mar 27 21:27:40 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:27:42 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 62
(Sun Mar 27 21:27:43 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return modificada
(Sun Mar 27 21:27:45 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:27:46 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 18
(Sun Mar 27 21:27:48 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 19]: return cancelada
(Sun Mar 27 21:27:49 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 20]: goto 29
(Sun Mar 27 21:27:54 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 81
(Sun Mar 27 21:27:56 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:27:57 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return null
(Sun Mar 27 21:27:59 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return inexistente
(Sun Mar 27 21:28:01 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:28:02 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:28:06 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:28:07 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:28:09 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 81

(Sun Mar 27 21:28:10 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:28:12 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 63
(Sun Mar 27 21:28:13 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return aberta
(Sun Mar 27 21:28:15 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 17
(Sun Mar 27 21:28:16 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:28:19 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:28:21 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:28:22 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 81
(Sun Mar 27 21:28:24 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:28:25 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 63
(Sun Mar 27 21:28:27 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return aceita
(Sun Mar 27 21:28:29 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 5
(Sun Mar 27 21:28:30 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 6]: return 476.0
(Sun Mar 27 21:28:32 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 7]: return null
(Sun Mar 27 21:28:34 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 8]: return 0
(Sun Mar 27 21:28:35 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 9]: goto 16
(Sun Mar 27 21:28:38 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:28:41 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:28:43 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:28:44 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 81
(Sun Mar 27 21:28:46 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:28:47 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 63
(Sun Mar 27 21:28:49 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return aceita
(Sun Mar 27 21:28:50 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 5
(Sun Mar 27 21:28:55 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 6]: return 476.0
(Sun Mar 27 21:28:57 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 7]: return null
(Sun Mar 27 21:28:58 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 8]: return 0
(Sun Mar 27 21:29:00 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 9]: goto 16
(Sun Mar 27 21:29:03 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:29:06 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:29:07 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:29:09 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 81
(Sun Mar 27 21:29:11 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:29:12 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 63
(Sun Mar 27 21:29:14 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return aceita
(Sun Mar 27 21:29:16 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 5
(Sun Mar 27 21:29:17 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 6]: return 476.0
(Sun Mar 27 21:29:19 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 7]: return null
(Sun Mar 27 21:29:21 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 8]: return 0
(Sun Mar 27 21:29:22 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 9]: goto 16
(Sun Mar 27 21:29:25 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:29:28 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:29:30 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31
(Sun Mar 27 21:29:31 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 81
(Sun Mar 27 21:29:33 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:29:34 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 3]: return 63
(Sun Mar 27 21:29:36 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 4]: return aceita
(Sun Mar 27 21:29:38 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 5]: goto 5
(Sun Mar 27 21:29:39 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 6]: return 476.0
(Sun Mar 27 21:29:41 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 7]: return 71
(Sun Mar 27 21:29:43 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 8]: return 1
(Sun Mar 27 21:29:44 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 9]: goto 9
(Sun Mar 27 21:29:46 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 10]: return Ordem 81 processada
(Sun Mar 27 21:29:48 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 11]: goto 11
(Sun Mar 27 21:29:49 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 12]: return 1
(Sun Mar 27 21:29:50 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 13]: return <?xml version="1.0"?>
(Sun Mar 27 21:29:52 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 14]: return 1
(Sun Mar 27 21:29:58 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 18]: goto 30
(Sun Mar 27 21:30:01 BRT 2011) DEBUG [Behavior(12) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:30:03 BRT 2011) DEBUG [Behavior(12) activity.ConditionalStatic(conditional static) 2]: goto 31

9.3 *LOGS* DO COMPORTAMENTO COMPRA-ALTERA

(Sun Mar 27 21:27:26 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 1]: modificou = 0
(Sun Mar 27 21:27:28 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 2]: return 0
(Sun Mar 27 21:27:29 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 3]: goto 31

(Sun Mar 27 21:27:30 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 1]: modificou = 0
(Sun Mar 27 21:27:32 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 2]: return 1
(Sun Mar 27 21:27:34 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 3]: goto 3
(Sun Mar 27 21:27:35 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 4]: return <?xml version="1.0" encoding="UTF-8"?>
(Sun Mar 27 21:27:37 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 5]: return $to compra-altera cancelar ordem 80
(Sun Mar 27 21:27:38 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 6]: aux = ordem
(Sun Mar 27 21:27:40 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 7]: return 80
(Sun Mar 27 21:27:41 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 8]: return 62
(Sun Mar 27 21:27:43 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 9]: goto 16
(Sun Mar 27 21:27:45 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 17]: goto 17
(Sun Mar 27 21:27:46 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 18]: return 1
(Sun Mar 27 21:27:48 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 19]: return 1
(Sun Mar 27 21:27:49 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 20]: goto 20
(Sun Mar 27 21:27:51 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 21]: modificou = 1
(Sun Mar 27 21:27:56 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 24]: goto 24
(Sun Mar 27 21:27:57 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 25]: return A ordem 80, do fornecedor John Doe Ltda., encontra-se cancelada, com: 10 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:27:59 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 26]: return 1
(Sun Mar 27 21:28:01 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 27]: return <?xml version="1.0"?>
(Sun Mar 27 21:28:02 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 28]: return 1
(Sun Mar 27 21:28:04 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 29]: modificou = 0
(Sun Mar 27 21:28:08 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 1]: modificou = 0
(Sun Mar 27 21:28:10 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 2]: return 0
(Sun Mar 27 21:28:12 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 3]: goto 31
(Sun Mar 27 21:28:13 BRT 2011) DEBUG [Behavior(13) activity.AssignStatic(assign static) 1]: modificou = 0
(Sun Mar 27 21:28:15 BRT 2011) DEBUG [Behavior(13) activity.AssignCall(assign call) 2]: return 0
(Sun Mar 27 21:28:16 BRT 2011) DEBUG [Behavior(13) activity.ConditionalStatic(conditional static) 3]: goto 31

9.4 *LOGS* DO COMPORTAMENTO MBOX

(Sun Mar 27 21:27:17 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:18 BRT 2011) DEBUG [Behavior(4) activity.ConditionalStatic(conditional static) 2]: goto 8
(Sun Mar 27 21:27:20 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 1]: return <?xml version="1.0" encoding="UTF-8"?>
(Sun Mar 27 21:27:21 BRT 2011) DEBUG [Behavior(4) activity.ConditionalStatic(conditional static) 2]: goto 2
(Sun Mar 27 21:27:23 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 3]: return 1
(Sun Mar 27 21:27:25 BRT 2011) DEBUG [Behavior(4) activity.ConditionalStatic(conditional static) 4]: goto 4
(Sun Mar 27 21:27:26 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 5]: return compra-altera
(Sun Mar 27 21:27:28 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 6]: return 1
(Sun Mar 27 21:27:32 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:34 BRT 2011) DEBUG [Behavior(4) activity.ConditionalStatic(conditional static) 2]: goto 8
(Sun Mar 27 21:27:35 BRT 2011) DEBUG [Behavior(4) activity.AssignCall(assign call) 1]: return 0
(Sun Mar 27 21:27:37 BRT 2011) DEBUG [Behavior(4) activity.ConditionalStatic(conditional static) 2]: goto 8

9.5 *LOGS* DO COMPORTAMENTO REPORT

(Sun Mar 27 21:54:58 BRT 2011) DEBUG [Behavior(7) activity.ConditionalStatic(conditional static) 1]: goto 14
(Sun Mar 27 21:55:01 BRT 2011) DEBUG [Behavior(7) activity.ConditionalStatic(conditional static) 1]: goto 1
(Sun Mar 27 21:55:02 BRT 2011) DEBUG [Behavior(7) activity.AssignCall(assign call) 2]: return Bom, há sempre uma coisa ou outra para fazer. O que fiz recentemente foi um pouco cansativo, mas consegui. Finalizando, deixa eu contar o que completei hoje.
(Sun Mar 27 21:55:04 BRT 2011) DEBUG [Behavior(7) activity.AssignCall(assign call) 3]: return 0
(Sun Mar 27 21:55:05 BRT 2011) DEBUG [Behavior(7) activity.AssignStatic(assign static) 4]: subject = Relatorio
(Sun Mar 27 21:55:07 BRT 2011) DEBUG [Behavior(7) activity.AssignCall(assign call) 5]: return Relatorio 2011-03-27
(Sun Mar 27 21:55:09 BRT 2011) DEBUG [Behavior(7) activity.ConditionalStatic(conditional static) 6]: goto 6
(Sun Mar 27 21:55:14 BRT 2011) DEBUG [Behavior(7) activity.AssignCall(assign call) 7]: return true
(Sun Mar 27 21:55:16 BRT 2011) DEBUG [Behavior(7) activity.AssignStatic(assign static) 8]: relatorioPrivado = 0
(Sun Mar 27 21:55:19 BRT 2011) DEBUG [Behavior(7) activity.ConditionalStatic(conditional static) 10]: goto 13

(Sun Mar 27 21:55:22 BRT 2011) DEBUG [Behavior(7) activity.ConditionalStatic(conditional static) 1]: goto 1

9.6 *LOGS* DO COMPORTAMENTO ATUALIZAHORA

(Sun Mar 27 21:24:22 BRT 2011) DEBUG [Behavior(8) activity.AssignCall(assign call) 1]: return 2011-03-27
(Sun Mar 27 21:24:23 BRT 2011) DEBUG [Behavior(8) activity.AssignCall(assign call) 2]: return 21:24:23
(Sun Mar 27 21:24:25 BRT 2011) DEBUG [Behavior(8) activity.AssignCall(assign call) 1]: return 2011-03-27
(Sun Mar 27 21:24:26 BRT 2011) DEBUG [Behavior(8) activity.AssignCall(assign call) 2]: return 21:24:26

9.7 *LOGS* DO SERVIDOR ISAP

(Sun Mar 27 21:24:19 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [main.ISAP-arisa]: User config loadded: arisa
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [main.ISAP-arisa]: Email config loadded: arisa
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [main.ISAP-arisa]: Twitter last_mention_id 52127970210492418
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [main.ISAP-arisa]: Twitter data loadded: noctislupus
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [main.ISAP-arisa]: Service loadded: gtalk
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [main.ISAP-arisa]: All data loadded...
(Sun Mar 27 21:24:20 BRT 2011) DEBUG [services.Chatter-arisa]: http://arisa.dyndns-server.com:8080/~saulo/ws/wsChatter.php
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [services.Chatter-arisa]: started...
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Webservice: http://arisa.dyndns-server.com:8080/~saulo/ws/wsXMPP.php
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Connected...
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [main.Server]: User loaded: arisa
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [main.Server]: Running...
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:24:22 BRT 2011) DEBUG [main.ISAP-arisa]: Running...
(Sun Mar 27 21:25:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:25:22 BRT 2011) DEBUG [main.ISAP-arisa]: Alive operations...
(Sun Mar 27 21:25:22 BRT 2011) DEBUG [main.ISAP-arisa]: Alive information received...
(Sun Mar 27 21:25:22 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Not connected...
(Sun Mar 27 21:25:22 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Login: personal.assistant.arisa@gmail.com Pass: XXXXX
(Sun Mar 27 21:25:25 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Login return: true

(Sun Mar 27 21:25:25 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Conected...
(Sun Mar 27 21:25:28 BRT 2011) DEBUG [services.Twitter-arisa]: get public messages from server - the last was 52127970210492418
(Sun Mar 27 21:25:30 BRT 2011) DEBUG [services.Twitter-arisa]: get private messages from server
(Sun Mar 27 21:25:31 BRT 2011) DEBUG [services.Twitter-arisa]: time out
(Sun Mar 27 21:25:31 BRT 2011) DEBUG [services.Twitter-arisa]: get public messages from server - the last was 52127970210492418
(Sun Mar 27 21:25:32 BRT 2011) DEBUG [services.Twitter-arisa]: get private messages from server
(Sun Mar 27 21:26:18 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Return user status (saulopz@gmail.com): available&dts; away ()

(Sun Mar 27 21:26:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:26:24 BRT 2011) DEBUG [main.ISAP-arisa]: Alive operations...
(Sun Mar 27 21:26:24 BRT 2011) DEBUG [main.ISAP-arisa]: Alive information received...
(Sun Mar 27 21:26:49 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Return user status (saulopz@gmail.com): available&dts; away ()

(Sun Mar 27 21:27:19 BRT 2011) DEBUG [services.Chatter-arisa]: Sending a chat to saulopz@gmail.com - $to compra-altera cancelar ordem 80

(Sun Mar 27 21:27:19 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: New message to saulopz@gmail.com
(Sun Mar 27 21:27:19 BRT 2011) DEBUG [main.ISAP-arisa]: Evio para GAP: saulopz@gmail.com(compra-altera)$to compra-altera cancelar ordem 80

(Sun Mar 27 21:27:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:27:25 BRT 2011) DEBUG [main.ISAP-arisa]: Alive operations...
(Sun Mar 27 21:27:25 BRT 2011) DEBUG [main.ISAP-arisa]: Alive information received...
(Sun Mar 27 21:27:34 BRT 2011) DEBUG [services.Twitter-arisa]: get public messages from server - the last was 52127970210492418
(Sun Mar 27 21:27:36 BRT 2011) DEBUG [services.Twitter-arisa]: get private messages from server
(Sun Mar 27 21:27:37 BRT 2011) DEBUG [services.Twitter-arisa]: time out
(Sun Mar 27 21:27:37 BRT 2011) DEBUG [services.Twitter-arisa]: get public messages from server - the last was 52127970210492418
(Sun Mar 27 21:27:38 BRT 2011) DEBUG [services.Twitter-arisa]: get private messages from server
(Sun Mar 27 21:28:03 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Return user status (saulopz@gmail.com): available&dts; away ()

(Sun Mar 27 21:28:16 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Return user status (saulopz@gmail.com): available&dts; away ()

(Sun Mar 27 21:28:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:28:28 BRT 2011) DEBUG [main.ISAP-arisa]: Alive operations...
(Sun Mar 27 21:28:28 BRT 2011) DEBUG [main.ISAP-arisa]: Alive information received...
(Sun Mar 27 21:29:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:29:29 BRT 2011) DEBUG [main.ISAP-arisa]: Alive operations...
(Sun Mar 27 21:29:29 BRT 2011) DEBUG [main.ISAP-arisa]: Alive information received...
(Sun Mar 27 21:29:41 BRT 2011) DEBUG [services.Twitter-arisa]: get public messages from server - the last was 52127970210492418
(Sun Mar 27 21:29:42 BRT 2011) DEBUG [services.Twitter-arisa]: get private messages from server
(Sun Mar 27 21:29:43 BRT 2011) DEBUG [services.Twitter-arisa]: time out
(Sun Mar 27 21:29:43 BRT 2011) DEBUG [services.Twitter-arisa]: get public messages from server - the last was 52127970210492418
(Sun Mar 27 21:29:44 BRT 2011) DEBUG [services.Twitter-arisa]: get private messages from server
(Sun Mar 27 21:29:54 BRT 2011) DEBUG [services.XMPP-Saulo Popov Zambiasi]: Return user status (saulopz@gmail.com): available&dts; away ()

(Sun Mar 27 21:30:22 BRT 2011) DEBUG [main.Server]: Loading users...
(Sun Mar 27 21:30:30 BRT 2011) DEBUG [main.ISAP-arisa]: Alive operations...
(Sun Mar 27 21:30:30 BRT 2011) DEBUG [main.ISAP-arisa]: Alive information received...

9.8 *LOGS* DO SERVIDOR VENDAS

(Sun Mar 27 21:24:19 BRT 2011) DEBUG [Main]: VENDAS starting at user saulo and database fornecedor
(Sun Mar 27 21:26:19 BRT 2011) DEBUG [Ordem 62]: Verifiyng order(62) status(aberta)
(Sun Mar 27 21:26:19 BRT 2011) DEBUG [Ordem 62]: qtd_Fornecedor[10] qtd_Ordem[17]
(Sun Mar 27 21:26:19 BRT 2011) DEBUG [Ordem 62]: entrou...
(Sun Mar 27 21:26:20 BRT 2011) DEBUG [Ordem 62]: quantity of product 101 is changed
(Sun Mar 27 21:26:20 BRT 2011) DEBUG [Ordem 62]: Changing order status to: modificada
(Sun Mar 27 21:26:20 BRT 2011) DEBUG [Ordem 62]: Stopping order verification 62
(Sun Mar 27 21:27:20 BRT 2011) DEBUG [Ordem 62]: Verifiyng order(62) status(modificada)
(Sun Mar 27 21:27:20 BRT 2011) DEBUG [Ordem 62]: Stopping order verification 62
(Sun Mar 27 21:28:20 BRT 2011) DEBUG [Ordem 63]: Verifiyng order(63) status(aberta)
(Sun Mar 27 21:28:20 BRT 2011) DEBUG [Ordem 63]: qtd_Fornecedor[100] qtd_Ordem[17]
(Sun Mar 27 21:28:20 BRT 2011) DEBUG [Ordem 63]: Changing order status to: aceita
(Sun Mar 27 21:28:20 BRT 2011) DEBUG [Ordem 63]: Stopping order verification 63
(Sun Mar 27 21:29:20 BRT 2011) DEBUG [Ordem 63]: Verifiyng order(63) status(aceita)
(Sun Mar 27 21:29:20 BRT 2011) DEBUG [Ordem 63]: Creating a new pay order of order 63
(Sun Mar 27 21:29:21 BRT 2011) DEBUG [Ordem 63]: order: 63 new pay order = 71
(Sun Mar 27 21:29:21 BRT 2011) DEBUG [Ordem 63]: Creating a new pay order from cnpj = 01.000.000/000-02 and value = 476.0
(Sun Mar 27 21:29:21 BRT 2011) DEBUG [Ordem 63]: Stopping order verification 63
(Sun Mar 27 21:30:21 BRT 2011) DEBUG [Ordem 63]: Verifiyng order(63) status(aceita)
(Sun Mar 27 21:30:21 BRT 2011) DEBUG [Ordem 63]: Payment confirmed 71
(Sun Mar 27 21:30:21 BRT 2011) DEBUG [Ordem 63]: payment of order 63 is ok
(Sun Mar 27 21:30:21 BRT 2011) DEBUG [Ordem 63]: Stopping order verification 63

9.9 *LOGS* DO SERVIDOR XMPPCONNECTOR

(Sun Mar 27 21:24:19 BRT 2011) DEBUG: XMPPConnector running at port 4000
(Sun Mar 27 21:24:19 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:25:22 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:connect:XXXX
(Sun Mar 27 21:25:22 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:connect:******
(Sun Mar 27 21:25:22 BRT 2011) DEBUG: personal.assistant.arisa@gmail.com:/127.0.0.1 - Command: connect
(Sun Mar 27 21:25:22 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Thread run... 9
(Sun Mar 27 21:25:25 BRT 2011) DEBUG: Message out: true
(...)
(Sun Mar 27 21:26:12 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:15 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:26:15 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:26:15 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:15 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:26:15 BRT 2011) DEBUG: Message out: false
(Sun Mar 27 21:26:15 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] < User status
(saulopz@gmail.com): available: away ()
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Message out: available&dts; away ()
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-ordem&dts; A ordem 80,do fornecedor John Doe Ltda., encontra-se aberta, com&dts; 17 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-ordem&dts; A ordem 80,do fornecedor John Doe Ltda., encontra-se aberta, com&dts; 17 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] New chat created with
saulopz@gmail.com
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Message sent:
saulopz@gmail.com - Compra-ordem: A ordem 80,do fornecedor John Doe Ltda., encontra-se aberta, com: 17 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Message out:
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:26:18 BRT 2011) DEBUG: Message out: false
(...)
(Sun Mar 27 21:26:46 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] < User status
(saulopz@gmail.com): available: away ()
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Message out: available&dts; away ()
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-altera&dts; Ordem 80 modificada&dts; 17 produto(s) 101 a R$ 25.00 para 10 produto(s) 101 a R$ 25.00. confirmar ou cancelar?
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-altera&dts; Ordem 80 modificada&dts; 17 produto(s) 101 a R$ 25.00 para 10 produto(s) 101 a R$ 25.00. confirmar ou cancelar?
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive

(Sun Mar 27 21:26:49 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Message sent:
saulopz@gmail.com - Compra-altera: Ordem 80 modificada: 17 produto(s) 101 a R$ 25.00 para 10 produto(s) 101 a R$ 25.00.
confirmar ou cancelar?
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Message out:
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:26:49 BRT 2011) DEBUG: Message out: false
(...)
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:getMessage
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:getMessage
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] got message:
saulopz@gmail.com/Notebook DC795974F
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Message out: saulopz@gmail.com:$to compra-altera cancelar ordem 80
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Ok! Vou processar essa informação.
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Ok! Vou processar essa informação.
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Message sent:
saulopz@gmail.com - Ok! Vou processar essa informação.
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Message out:
(Sun Mar 27 21:27:19 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:27:22 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:27:22 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:27:22 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:27:22 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:27:22 BRT 2011) DEBUG: Message out: false
(...)
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: Message out: available&dts; away ()
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-altera&dts; A ordem 80, do fornecedor John Doe Ltda., encontra-se cancelada, com&dts; 10 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-altera&dts; A ordem 80, do fornecedor John Doe Ltda., encontra-se cancelada, com&dts; 10 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Message sent:
saulopz@gmail.com - Compra-altera: A ordem 80, do fornecedor John Doe Ltda., encontra-se cancelada, com: 10 produto(s) 101 a R$ 25.00,
(Sun Mar 27 21:28:03 BRT 2011) DEBUG: Message out:
(...)
(Sun Mar 27 21:28:13 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] < User status
(saulopz@gmail.com): available: away ()
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Message out: available&dts; away ()
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Message in:
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-ordem&dts; A ordem 81, do fornecedor Sora konpyuuta, encontra-se aberta, com&dts; 17 produto(s) 101 a R$ 28.00,
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Execute: /127.0.0.1 -
personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra-ordem&dts; A ordem 81, do fornecedor Sora konpyuuta, encontra-se aberta, com&dts; 17 produto(s) 101 a R$ 28.00,
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive

(Sun Mar 27 21:28:16 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Message sent: saulopz@gmail.com - Compra-ordem: A ordem 81, do fornecedor Sora konpyuuta, encontra-se aberta, com: 17 produto(s) 101 a R$ 28.00,
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Message out:
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:hasMessage
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:28:16 BRT 2011) DEBUG: Message out: false
(...)
(Sun Mar 27 21:29:51 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:getUserStatus:saulopz@gmail.com
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] < User status (saulopz@gmail.com): available: away ()
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Message out: available&dts; away ()
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Wait new connection...
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Message in: personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra&dts; Ordem 81 processada
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Execute: /127.0.0.1 - personal.assistant.arisa@gmail.com:sendMessage:saulopz@gmail.com:Compra&dts; Ordem 81 processada
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Client: personal.assistant.arisa@gmail.com:/127.0.0.1
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] alive
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: [CLIENT - personal.assistant.arisa@gmail.com] Message sent: saulopz@gmail.com - Compra: Ordem 81 processada
(Sun Mar 27 21:29:54 BRT 2011) DEBUG: Message out: