UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE APLICAÇÕES USANDO COMPOSIÇÃO DE SERVIÇOS DIRIGIDA PELO USUÁRIO

ALEX ROBERTO GUIDO

ORIENTADOR: PROF. DR. ANTONIO FRANCISCO DO PRADO

São Carlos – SP

2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE APLICAÇÕES USANDO COMPOSIÇÃO DE SERVIÇOS DIRIGIDA PELO USUÁRIO

ALEX ROBERTO GUIDO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software

Orientador: Prof. Dr. Antonio Francisco do Prado

São Carlos – SP

2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Guido, Alex Roberto
G948d Desenvolvimento de aplicações usando composição de serviços dirigida pelo usuário / Alex Roberto Guido. -- São Carlos : UFSCar, 2016.
81 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2016.

1. Engenharia de software. 2. Desenvolvimento de aplicações. 3. Composição de serviços dirigida pelo usuário. 4. REST. 5. HATEOAS. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Alex Roberto Guido, realizada em 14/09/2016.

Prof. Dr. Antonio Francisco do Prado
(UFSCar)

Prof.Dr. Wanderley Lopes de Souza
(UFSCar)

****

Prof. Dr. Vinicius Cardoso Garcia
(UTFPR)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Prof. Dr. Vinicius Cardoso Garcia . Depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Alex Roberto Guido.

Prof. Dr. Antonio Francisco do Prado
Coordenador da Comissão Examinadora
(UFSCar)

Dedico este trabalho aos meus pais Antonio e Silvana, que sempre batalharam para dar condições para que eu pudesse alcançar meus objetivos.

O mais importante da vida não é a situação em que estamos,
mas a direção para a qual nos movemos.

Oliver Wendell Holmes

RESUMO

Um dos desafios da Engenharia de Software é o desenvolvimento de aplicações capazes de adaptar-se às diferentes necessidades dos usuários. A técnica de Composição Dinâmica de Serviços Dirigida pelo Usuário é uma solução para desenvolver aplicações capazes de superar esses desafios. Esse tipo de aplicação, a qual denominaremos *User-Driven Service Composition Application (UDSCA),* permite compor serviços durante sua execução, satisfazendo assim às necessidades dos usuários. Porém a falta de orientação em como desenvolver UDSCAs pode dificultar ou mesmo impossibilitar a construção dessas aplicações, pelo fato de agregar soluções que possam ser desconhecidas pelos desenvolvedores. Procurando suprir essa falta, este trabalho apresenta uma abordagem capaz de orientar os desenvolvedores durante o desenvolvimento deste tipo de aplicação. Na abordagem foram definidas quais as atividades devem ser realizadas durante o desenvolvimento deste tipo de aplicação, assim como os conceitos, técnicas, artefatos, tecnologias e ferramentas necessárias para realizar essas atividades. Para avaliar a abordagem foi realizado um estudo de caso que compreende o desenvolvimento de uma UDSCA no domínio de serviços urbanos. A aplicação resultante da abordagem mostrou-se capaz de adaptar-se às necessidades heterogêneas dos usuários. Além disso, a abordagem forneceu artefatos que promoveram o reuso. Concluiu-se assim que a abordagem orienta os desenvolvedores durante o desenvolvimento de UDSCAs e fornece artefatos que reduzem esforços durante esse desenvolvimento.

Palavras-chave: Engenharia de Software; Desenvolvimento de Aplicações; Composição de Serviços Dirigida pelo Usuário; REST; HATEOAS.

ABSTRACT

One of the Software Engineering challenges is the development of applications that can adapt to the heterogeneous needs of users. Technical Dynamic Composition of Services Driven by User is a solution for developing applications capable of overcoming these challenges. This type of application which will call *User-Driven Service Composition Application (UDSCA)* allows to compose services during its execution, thus meeting the needs of users. But the lack of guidance on how to develop UDCAs can make it difficult or even impossible to build these applications, because it may aggregate unknown solutions by developers. Looking supply this lack, this work presents an approach to guide developers during the development of this kind of application. To develop the approach, it has been defined which activities should be undertaken during the development as well as the concepts, techniques, artifacts, technologies and tools needed to perform these activities. To evaluate the approach one conducted a case study in which a UDSCA was developed in the field of building maintenance services. The resulting application of the approach was shown to be able to adapt to the heterogeneous needs of the user, also the approach provided artifacts that promoted reuse. In conclusion, the approach guides the developer during the UDSCAs development and provides artifacts that reduce efforts for development.

Keywords: Software Engineering; Application Development; User-Driven Service Composition; REST; HATEOAS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo é apresentada a introdução desta dissertação de mestrado. Na Seção 1.1 é abordado o contexto desta pesquisa; na Seção 1.2 são discutidas as motivações e os objetivos, e na Seção 1.3 é apresentada a estrutura deste documento.

1.1 Contexto

Com a evolução dos dispositivos móveis e a internet tornando-se onipresente, é cada vez mais comum as pessoas utilizarem o apoio computacional para realizar tarefas cotidianas. Atualmente é comum a realização de tarefas tais como acessar uma rede social, ler um *e-mail*, controlar suas finanças, conversar com familiares e amigos, planejar uma viagem e desfrutar de uma abundância de outras aplicações a partir de dispositivos móveis. Esse cenário mostra que a visão de ubiquidade computacional introduzida por Weiser há mais de duas décadas, tem se concretizado (WEISER, 1993).

Um dos desafios da Engenharia de Software é produzir aplicações capazes de adaptar-se às diferentes necessidades dos usuários (BERTOLINO, 2015). Essas aplicações podem ser utilizadas por usuários com diferentes preferências, bem como podem ser utilizadas em diferentes situações. (SILVA; PIRES; SINDEREN, 2012). O uso em diferentes situações exige que a aplicação seja capaz de adaptar-se às mudanças que ocorrem no ambiente, nos dispositivos e nas necessidades que podem variar dependendo do usuário (CIRILO, 2010). Uma situação que pode ilustrar esse cenário é um planejamento de viagem. Pelo fato dos usuários possuírem diferentes preferências, esse planejamento pode ser realizado de diversas maneiras, como por exemplo: enquanto uma pessoa está interessada em visitar monumentos históricos, outra pessoa pode estar apenas interessada em conhecer pessoas da região; enquanto uma pessoa está efetuando o planejamento de sua viagem utilizando um dispositivo estacionário, outra pessoa pode estar utilizando um

dispositivo móvel.

Essa realidade tem exigido cada vez mais aplicações capazes de se adaptar para suprir as diferentes necessidades dos usuários (GARTNER, 2010). Na prática, ao implementar esse tipo de aplicação, os desenvolvedores encontraram algumas dificuldades, como por exemplo: inviabilidade em desenvolver algumas funcionalidades, pelo fato dessas funcionalidades mudarem conforme as necessidades dos usuários (SILVA; PIRES; SINDEREN, 2012). Portanto, construir aplicações para satisfazer as diferentes necessidades dos usuários é uma desafio que tem incentivado várias pesquisas na comunidade científica (SILVA; PIRES; SINDEREN, 2012; SANTOS, 2010; VLIEGS, 2010).

Uma das soluções que tem se mostrado bem sucedida é o reuso de serviços (OREIZY; MEDVIDOVIC; TAYLOR, 2008), bastante explorado na *Service-Oriented Computing* (SOC). SOC permite a criação de aplicações de maneira rápida e com baixo custo, pelo fato de reutilizar ou combinar serviços próprios ou de terceiros de maneira interoperável em ambientes heterogêneos (PAPAZOGLOU, 2008). Seguindo essa ideia, utilizar a técnica de combinar serviços (também conhecida como composição de serviços), é uma interessante solução para suprir a necessidade de desenvolver aplicações se adaptam para acompanhar as mudanças que podem ocorrer no ambiente e nas preferências dos usuários (SILVA; PIRES; SINDEREN, 2012; CHEN; CLARKE, 2014). Essa técnica quando realizada de forma dinâmica e centrada nas necessidades do usuário (*Composição Dinâmica de Serviços Dirigida pelo Usuário*) torna viável o desenvolvimento de User-Driven Service Composition Applications (UDSCA), as quais são capazes de suprir as necessidades apresentadas anteriormente. O uso dessa técnica aumenta as chances de encontrar serviços e composições que atendam as preferências dos usuários, pelo fato dos numerosos serviços e composições disponibilizadas por diversos provedores.

Para compreender melhor esse tipo de composição, suponha o seguinte exemplo: um turista está fazendo uma viagem por um país desconhecido e deseja obter informações de hospedagem a partir de seu dispositivo móvel. Para que isto seja possível, deve haver uma aplicação nesse dispositivo que irá buscar serviços de hospedagem na internet, bem como serviços sobre a localização do usuário. Após realizada a busca desses serviços, a composição desses serviços é então realizada durante a execução da aplicação. As preferências individuais do usuário também são consideradas, como por exemplo a quantidade de quartos desejável na hospedagem ou a busca de outro serviço responsável por verificar se o hotel é próximo a algum restaurante com suas preferências gastronômicas.

No desenvolvimento de UDSCAs, a falta de orientação pode tornar essa tarefa complexa, pelo fato de agregar soluções que possam ser desconhecidas pelos desenvolvedores, podendo

inclusive inviabilizar a construção dessas aplicações.

1.2 Motivação e Objetivo

O Institute of Electrical and Electronics Engineers (IEEE[1]) lançou uma edição especial de uma revista (BERTOLINO, 2015) apresentando pesquisas realizadas na Engenharia de Software, sendo que um dos desafios é o desenvolvimento de aplicações para ambientes heterogêneos. A Sociedade Brasileira de Computação (SBC[2]) tem buscado sistematicamente definir os futuros desafios em Ciência da Computação, dentre os quais tem se destacado o desenvolvimento de aplicações capazes de serem acessadas por qualquer dispositivo em qualquer ambiente e capazes de atender às preferências dos usuários.

Outro desafio é que essas aplicações possam atender aos requisitos específicos de cada usuário, principalmente em tarefas cotidianas, como por exemplo na busca de imóveis, no qual o usuário deve utilizar serviços rotineiros de sua preferência. Um exemplo desses serviços rotineiros é a localização, devido ao fato dos usuários possuírem preferências geográficas ao buscar imóveis.

Motivados por esses desafios, diversas soluções têm sido propostas na literatura para o desenvolvimento de aplicações adaptáveis: para estabelecer uma Programação Orientada ao Contexto (SALVANESCHI; GHEZZI; PRADELLA, 2012; HIRSCHFELD; COSTANZA; NIERSTRASZ, 2008; HAN, 2013); para definir um processo de desenvolvimento (VIEIRA; TEDESCO; SALGADO, 2009; AUGUSTO, 2014); para a utilização de serviços e suas composições (SILVA; PIRES; SINDEREN, 2012; SHENG, 2014); e para a definição de métodos e construção de ferramentas e *frameworks* (BRHEL, 2015; OREIZY; MEDVIDOVIC; TAYLOR, 2008).

Uma das soluções que tem se mostrado bem sucedida é a utilização de serviços e suas composições (OREIZY; MEDVIDOVIC; TAYLOR, 2008; SILVA; PIRES; SINDEREN, 2012; SHENG, 2014). Devido a isto, atualmente é comum encontrar aplicações que fazem uso de serviços ou disponibilizam conteúdo através de serviços.

Conforme apresentado na Seção 1.1, a composição dinâmica de serviços dirigida pelo usuário é uma solução para desenvolver UDSCAs. Porém, a falta de orientação em como desenvolver esse tipo de aplicação ainda é um tema pouco explorado, tornando o desenvolvimento desse tipo de aplicação uma tarefa complexa. De acordo com (ALI; PETERSEN; WOHLIN, 2014) uma abordagem que oriente os desenvolvedores durante o desenvolvimento é essencial para a pro-

[1] https://www.ieee.org/ acessado em 13/07/2016
[2] http://www.sbc.org.br/ acessado 13/07/2016

dução de aplicações de qualidade com redução de recursos e tempo.

Portanto, o objetivo deste trabalho é apresentar uma abordagem capaz de orientar os desenvolvedores durante o processo de desenvolvimento de UDSCAs e fornecer artefatos que promovam o reuso, reduzindo custos e tempo dos desenvolvedores.

1.3 Estrutura da Dissertação

Conforme ilustrado na Figura 1.1, esta dissertação está organizada em seis capítulos, incluindo este capítulo introdutório. O conteúdo de cada capítulo é brevemente descrito a seguir:

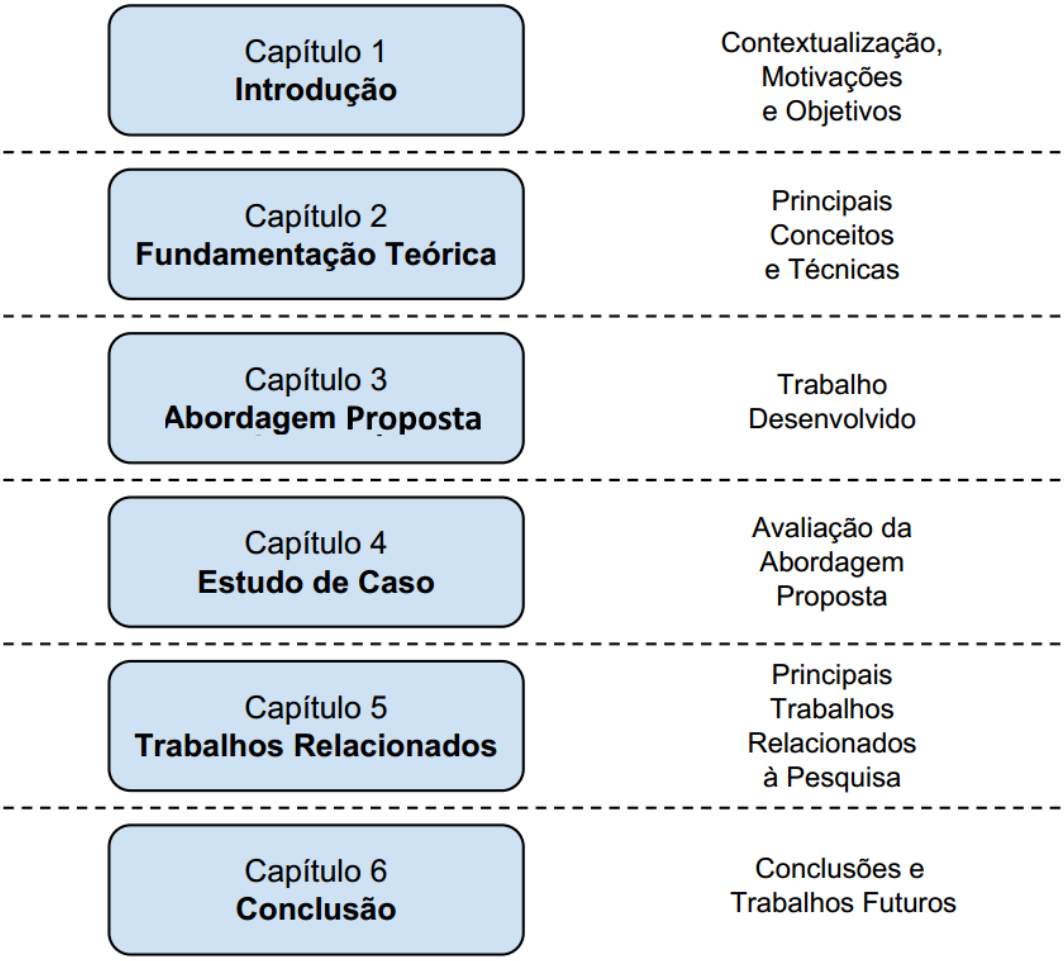

Figura 1.1: Estrutura da Dissertação.

- No **Capítulo 2** é apresentada a fundamentação teórica relacionada aos principais conceitos, técnicas e ferramentas abordados nesta pesquisa: Desenvolvimento de Software, Computação Orientada a Serviços e Framework A-DynamiCoS.

- No **Capítulo 3** é apresentada a abordagem desenvolvida neste trabalho, que teve como objetivo orientar os desenvolvedores durante o desenvolvimento de UDSCAs. Para cada atividade definada na abordagem são apresentadas suas respectivas entradas, saídas e ferramentas necessárias para executá-las.

- No **Capítulo 4** é apresentado um estudo de caso que compreende o desenvolvimento de uma UDSCA no domínio de serviços urbanos e realizada uma avaliação dos resultados obtidos para fins de comparação;

- No **Capítulo 5** são apresentados os trabalhos relacionados à presente pesquisa, apresentando uma análise comparativa desses trabalhos com a abordagem apresentada neste trabalho; e

- No **Capítulo 6** são discutidas as conclusões deste trabalho, apresentando as contribuições e trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Foi fundamental para o desenvolvimento deste trabalho a obtenção de conhecimento referente às áreas de Desenvolvimento de Aplicações, Serviços e Composição Dinâmica de Serviços Dirigida pelo Usuário, as quais são descritas respectivamente na Seção 2.1, Seção 2.2 e Seção 2.3 deste capítulo.

2.1 Desenvolvimento de Aplicações

Com o crescimento dos dispositivos móveis, redes sociais e os avanços da computação, o uso de aplicativos vêm se tornando comum na solução de problemas e no apoio nas tarefas cotidianas.

Atualmente é comum usuários buscarem soluções para seus problemas através do apoio de aplicativos disponíveis nos diferentes domínios do conhecimento. Essa demanda tem incentivado o desenvolvimento de aplicativos de forma a melhor atender as necessidades dos usuários proporcionando ganho de tempo e redução de custos.

Outro aspecto importante a ressaltar nesse contexto é que o usuário muitas vezes tem necessidades específicas que podem não ser atendidas pelos aplicativos existentes. Assim, torna-se importante, a disponibilidade de recursos computacionais com os quais o próprio usuário possa customizar os aplicativos conforme suas necessidades mais específicas. Por exemplo, num domínio de aplicações na área de viagens e turismo, vários aplicativos têm sido construídos, e mesmo assim podem não atender os objetivos de cada usuário. Muitas vezes os serviços disponíveis em um aplicativo atende parcialmente e precisam ser complementados com serviços de outros aplicativos, como no caso de uma viagem que requisita passagens, transportes, acomodações, alimentação e passeios.

Aplicações dirigidas pelo próprio usuário podem resolver grande parte dessas suas necessidades específicas. A construção e disponibilização de recursos que possibilita ao usuário customizar aplicações conforme suas demandas específicas têm sido pesquisadas e exploradas como solução para melhor atender esses objetivos específicos (SILVA; PIRES; SINDEREN, 2012).

Com foco em Engenharia de Domínio(PRESSMAN, 2010), numa primeira etapa a abordagem orienta o desenvolvimento de aplicações que atendam aos requisitos definidos no escopo do domínio do problema. Nessa etapa são utilizadas ontologias para especificar os requisitos do domínio do problema, e dessa forma obter um conjunto maior de serviços no escopo desse domínio. A adoção de ontologias possibilita um levantamento mais amplo dos requisitos e a construção de aplicações que atendem uma maior gama de serviços dentro do escopo do domínio do problema. Essas ontologias definem os objetivos que podem ser alcançados em um determinado domínio para serem implementados por um conjunto de serviços. Numa segunda etapa da abordagem, o usuário, reutiliza os serviços disponíveis para atender suas necessidades específicas. Essa aplicação, dirigida pelo usuário, facilita o atendimento dos seus objetivos usando composição dinâmica de serviços. O usuário mesmo com pouco conhecimento sobre computação, será capaz de personalizar a aplicação durante sua execução para alcançar seus objetivos considerando suas necessidades e preferências específicas.

2.2 Serviços

De acordo com Kotler e Bloom (2002),serviço é qualquer atividade ou benefício que uma parte possa oferecer a outra. Seja essa parte um indivíduo, sociedade ou aplicação. Inúmeras dessas atividades ocorrem no cotidiano, por exemplo: o fornecimento de energia elétrica por parte de uma empresa para uma cidade; o recolhimento de lixo urbano das ruas; e serviços de telefonia.

Na Figura 2.1 é apresentado um cenário onde cada indivíduo está apto a fornecer um tipo diferente de serviço.

O termo serviço pode ser utilizado em diversas áreas, pois sua definição genérica não será alterada. Entretanto, ao especificarmos o âmbito em que este se encontra, seu significado pode ser estendido para abranger o escopo em questão.

Na Ciência da Computação por exemplo, serviços tornaram-se conhecidos devido a Computação Orientada a Serviços, sendo esse um paradigma computacional. De acordo com Papazoglou (2007), SOC prega o uso de serviços como o principal meio para desenvolver aplicações. Esses serviços são independentes de plataforma e podem colaborar entre si, viabilizando sua

Figura 2.1: Indíviduos e serviços prestados (ERL, 2009).

combinação e utilização em aplicações. Essa utilização de serviços em aplicações torna mais ágil o desenvolvimento, pelo fato do desenvolvedor reutilizar serviços já existentes. Portanto, serviços agem como elementos fundamentais neste tipo de desenvolvimento.

Conforme ilustrado na Figura 2.2, um serviço no contexto de aplicações pode oferecer várias ações para realizar uma determinada tarefa, assim como é feito por pessoas em tarefas cotidianas. Por exemplo, da mesma maneira que uma pessoa pode preencher uma lista de mercadorias e disponibilizar de maneira impressa, uma aplicação pode utilizar seu método *add* para preencher uma lista semelhante e disponibilizar no formato digital.

Figura 2.2: Comparação entre serviços prestados por pessoas e aplicações (ERL, 2009).

2.2.1 Arquitetura Orientada a Serviços (SOA)

SOC fundamenta-se na Arquitetura Orientada a Serviços *(Service-Oriented Architecture - SOA)* para introduzir o uso de serviços interoperáveis como elementos fundamentais no desenvolvimento de aplicações (PAPAZOGLOU, 2007). Esse tipo de serviço possui a capacidade de se comunicar com outros sistemas de maneira transparente e indepentende de plataforma.

SOA é um modelo arquitetônico, o qual utiliza serviços como o principal meio para realização dos objetivos estratégicos associados à computação orientada a serviços (ERL, 2009), como por exemplo prover as funcionalidades na forma de serviços reutilizáveis. Essas funcionalidades são desenvolvidas a partir da combinação (ou também denominada composição) desses serviços. A funcionalidade resultante dessa composição é fornecida também como um serviço (PAPAZOGLOU; HEUVEL, 2007). Diferente das arquiteturas convencionais, as quais geralmente possuem alto grau de acoplamento entre seus componentes internos, SOA tem foco no projeto de aplicações que disponibilizam suas funcionalidades por meio de interfaces públicas, tornando possível sua reutilização de maneira interoperável (PAPAZOGLOU, 2007; PAPAZOGLOU; HEUVEL, 2007).

O aumento de interoperabilidade e redução do acoplamento proporcionados por SOA tem como finalidade aprimorar a eficiência, agilidade e a produtividade no desenvolvimento e manutenção de aplicações (ERL, 2009).

2.2.2 Web Services

De acordo com Sheng (2014), nos últimos anos a tecnologia de *Web Services (WS)* tem se destacado como a maneira mais promissora de implementar serviços em aplicações e aplicar o paradigma SOC na Web. Um WS é um tipo específico de serviço, sendo identificado por um *Uniform Resource Identifier (URI)* e utilizado através de padrões abertos e protocolos da Internet (PAPAZOGLOU, 2003). Sheng (2014) dá uma visão mais abstrata, definindo WS como uma aplicação acessível por outras aplicações, utilizando a Internet como forma de comunicação.

A definição padrão utilizada pela *World Wide Web Consortium* (W3C[1]), define WS como uma aplicação identificada por uma URI, descrita utilizando *eXtensible Markup Language (XML)* e descoberta por outras aplicações, com o intuito dessas aplicações reutilizarem a funcionalidade disponibilizada pelo WS. Essas aplicações podem interagir com o WS utilizando protocolos da Internet.

[1] http://www.w3.org/ acessado em 15/05/2016

De acordo com (PAUTASSO; ZIMMERMANN; LEYMANN, 2008), existem dois tipos de WS que são mais populares e amplamente utilizados:

- ***WS-*:*** Este tipo de WS depende de três importantes tipos de normalização: *Simple Object Access Protocol* (SOAP[2]), *Web Services Description Language* (WSDL[3]), e *Universal Description, Discovery, and Integration* (UDDI[4]). Geralmente esse tipo de WS necessita de mais recursos computacionais, pelo fato de incorporar protocolos de segurança e transferência de dados. Portanto são geralmente utilizados em aplicações complexas.
- ***RESTful*:** Este tipo de WS utiliza a arquitetura *REpresentational State Transfer* (REST), sendo essa uma arquitetura para construção de sistemas hipermídia distribuídos em grande escala (FIELDING, 2000). A comunicação com um WS RESTful é realizada através do protocolo *Hypertext Transfer Protocol* (HTTP), por meio da troca de mensagens. Como necessita de menos recursos computacionais, esse tipo de WS costuma ser mais simples.

Pelo fato do WS RESTful utilizar menos recursos computacionais e ser mais simples de implementar, esse tipo de WS será adotado para disponibilizar artefatos resultante da abordagem proposta neste trabalho.

2.2.3 *Web Services RESTful*

Apesar da W3C ter definido a comunicação de WS apenas utilizando o protocolo SOAP, é possível efetuar essa comunicação de maneira interoperável utilizando outros padrões encontrados na Web (SHI, 2006). Uma solução alternativa para efetuar a comunicação de WS bastante utilizada recentemente é fornecida por *REpresentational State Transfer (REST)* (FIELDING, 2000). REST é um estilo arquitetural para sistemas hipermídia distribuídos de larga escala, onde seus princípios são fundamentais para que haja a alta escalabilidade da Web. Sendo assim, REST não é apenas uma alternativa para WS, e sim uma abstração da arquitetura utilizada pela Web (PAUTASSO; ZIMMERMANN; LEYMANN, 2008; BELQASMI; GLITHO; FU, 2011).

WS que se fundamentam nos princípios de REST são chamados de WS RESTful. Esse tipo de WS tem por objetivo simplificar o desenvolvimento, publicação e utilização de WS, e ao mesmo tempo garantir uma alta escalabilidade. Essa simplicidade ocorre pelo fato de REST utilizar padrões disponibilizados pela própria Web, como por exemplo, o uso do HTTP como o seu principal protocolo de comunicação (BELQASMI; GLITHO; FU, 2011). Diferente dos WS

[2]http://www.w3.org/TR/SOAP acessado em 15/05/2016
[3]http://www.w3.org/TR/wsdl acessado em 15/05/2016
[4]http://www.uddi.org acessado em 15/05/2016

RESTful, WS-* necessitam de outros protocolos e normalizações, tornando assim esse tipo de WS mais complexo.

WS RESTful expõe suas capacidades na forma de *recursos* que podem ser acessíveis pela Web. Para acessar esses recursos são utilizados os métodos de comunicação do protocolo HTTP. Esse tipo de exposição faz com que WS RESTful utilize uma arquitetura similar à da Web, a qual é estruturada como uma rede de recursos interligados (FIELDING; TAYLOR, 2002). De acordo com Pautasso, Zimmermann e Leymann (2008), Belqasmi, Glitho e Fu (2011), Richardson e Ruby (2007) a construção de WS RESTful baseia-se nos seguintes princípios:

- *Recursos identificados por URIs*[5]: No contexto de REST, um recurso é qualquer tipo de informação que possa ser identificada e endereçado através de uma URI, por exemplo: documentos textuais, resultados de uma busca, imagens e registros no banco de dados (BERNERS-LEE; FIELDING; MASINTER, 1998). Seguindo essa ideia, um WS RESTful expõe suas capacidades como recursos que podem ser acessados por URIs através do protocolo HTTP. A requisição a um recurso resulta no retorno de uma ou mais *representações* para o requisitante, sendo que esses retornos podem estar representados na forma de uma página HTML ou uma imagem. Uma situação que pode ilustrar esse cenário é um WS que fornece o recurso *lista de contas bancárias*. A requisição a esse recurso resultará no retorno de todas as contas cadastradas no banco de dados da instituição bancária. Esse retorno pode ser representado como um documento no formato *Portable Document Format (PDF)*, uma sequência de caracteres;

- *Interface uniforme*: A manipulação de recursos em REST se dá por um conjunto fixo de operações CRUD, que permitem criar *(Create)*, ler *(Read)*, atualizar *(Update)* e excluir *(Delete)*. Essas operações são realizadas pelos métodos do HTTP: PUT, GET, POST e DELETE, respectivamente e na Figura 2.3 é ilustrado como um WS RESTful disponibiliza seus recursos e operações de manipulação. Aplicações que desejem utilizar um WS RESTful apenas devem compreender como utilizar as operações do HTTP, trazendo assim vantagens como facilidade de uso e interoperabilidade; e

- *Interações Stateless*: Uma interação é *Stateless* pelo fato de uma requisição a um recurso conter todos os dados necessários para que o servidor possa processá-lo, sem a necessidade de informações da sessão do cliente ou do estado de outros recursos. Esse servidor irá sempre retornar a representação atual do recurso, sendo que essa pode estar representada, por exemplo em formato PDF, HTML e texto.

[5] *Uniform Resource Locator (URL)* é o tipo de URI mais utilizado para endereçamento na Web usando o HTTP

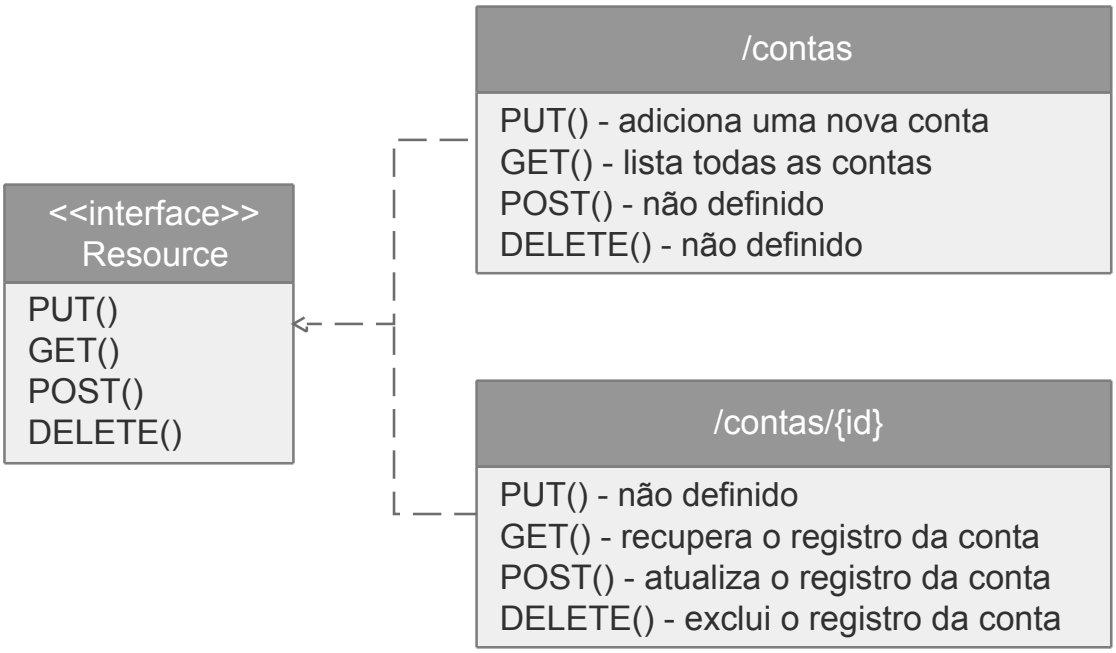

Figura 2.3: Exemplo de recursos fornecidos por um Web Service RESTful de uma instituição bancária.

A aplicação dos princípios de REST simplifica a disponibilização e utilização de WS. Isso ocorre porquê presume-se que apenas utilizando os mecanismos fornecidos pela Web já é suficiente para interagir com um WS. Isto descarta a necessidade de protocolos adicionais para a comunicação, evitando assim a adição de uma carga adicional desnecessária na interação com o WS, tornando asim o WS mais simples. Nas Figuras 2.4 e 2.5 são apresentadas duas requisições HTTP à um WS. Nessas requisições é solicitado uma operação (ou recurso) que retorne uma listagem das contas cadastradas no banco de dados de uma determinada instituição bancária. Na Figura 2.4 a requisição é feita a um WS que utiliza o protocolo SOAP para troca de mensagens, enquanto que na Figura 2.5 a requisição é realizada utilizando os princípios de REST. Pode-se observar que a requisição que utiliza os princípios de REST reduz significativamente a quantidade de informação da requisição HTTP, pelo fato da requisição ser realizada utilizando apenas o protocolo HTTP, tornando esse tipo de requisição mais simples. Em contrapartida, a requisição realizada para um WS que utilize SOAP necessita ser sobrecarregada com conteúdo XML do envelope SOAP, tornando esse tipo de requisição mais complexa. O REST tem por objetivo fazer com que os WS utilizem a Web para alcançar seus objetivos, tornando-se assim parte da Web e não apenas disponibilizados na Web (RICHARDSON; RUBY, 2007).

Um recurso REST pode representar seu estado em diferentes formatos (PAUTASSO; ZIM-

```
POST /conta
SOAPAction: “http://www.bancoacademico.com/contalist”
Content-Type: text/xml; charset=”utf-8”
Content-Length: …
…

<SOAP:Envelope xmlns:SOAP=”http://schemas.xmlsoap.org/soap/envelope/”>
    <SOAP:Body>
        <pt:conta xmlns:pt=”http://www.bancoacademico.com/conta/schema”>
            <pt:contaNumero numero=”*” />
        </pt:conta>
    </SOAP:Body>
</SOAP:Envelope>
```

Figura 2.4: Requisão a WS utilizando SOAP.

```
GET /contas HTTP/1.1
Host: bancoacademico.com
Accept: text/xml
...
```

Figura 2.5: Requisão a WS utilizando REST.

MERMANN; LEYMANN, 2008). Alguns desses formatos são: HTML, XML, JSON e texto puro (BELQASMI; GLITHO; FU, 2011). Na Figura 2.6 é ilustrada a representação do recurso lista de contas bancárias nos formatos JSON e XML, ambos comumente usados para representar recursos em WS RESTful. O formato em que o recurso será representado é negociado através do campo *Accept* do cabeçalho da requisição HTTP.

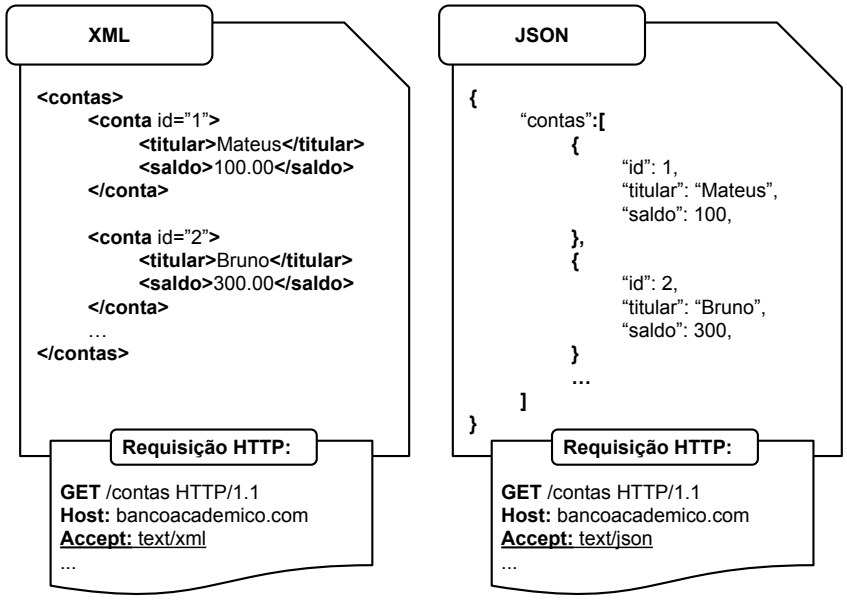

Figura 2.6: Representações do recurso *contas* em XML e JSON.

2.2.4 *Hypermedia as the Engine of Application State*

HATEOAS é a abreviação de *Hypermedia as the Engine of Application State*, sendo essa uma restrição da arquitetura REST apresentada por Fielding e Taylor (2002). O princípio utilizado por HATEOAS é que um cliente não precisa de nenhum conhecimento de como se comunicar com uma aplicação ou servidor na rede, apenas deve ter um entendimento de como efetuar a comunicação utilizando hipermídia. Um bom exemplo de utilização desse princípio é a Web, quando solicitamos uma página Web não precisamos ter conhecimento da lógica implementada no servidor, apenas efetuamos uma requisição através do protocolo HTTP passando a URL da página Web e em seguida o servidor retorna o conteúdo e os links com as próximas possíveis ações que podem ser realizadas.

Assim como as páginas Web, serviços podem ser implantados na Web. Com isto, empresas têm fornecido seus dados publicamente por meio de *Application Programming Interfaces (APIs)* implementadas como serviços Webs. Pesquisas mostram que 74% dessas APIs são implementadas utilizando a arquitetura REST, sendo denominadas RESTful APIs (LISKIN; SINGER; SCHNEIDER, 2011).

A reutilização dessas APIs pode ser realizada por aplicações em diferentes plataformas, através de requisições HTTP ao servidor onde a RESTful API está localizada. No exemplo do Código 2.1 uma aplicação faz uma requisição para uma RESTful API de uma instituição bancária, essa requisição é realizada utilizando o método GET do protocolo HTTP. Nessa requisição é solicitado o recurso conta para um determinado host, e se espera como resposta os dados no formato *JavaScript Object Notation (JSON)*.

```
1 GET para o recurso de conta bancaria.
2 GET /conta/1 HTTP/1.1
3 Host: banco_academico.org
4 Accept: app/json
5 ...
```

Código 2.1: Exemplo de Requisição GET para o recurso de conta bancária.

A resposta à requisição GET para o recurso conta bancária pode ser observada no Código 2.2. A resposta está no formato JSON e é composta pelo cabeçalho, conteúdo (número e saldo), as possíveis ações que podem ser realizadas posteriormente (depositar, retirar, transferir e encerrar) e a localização de cada recurso responsável por efetuar essas ações (/conta/1/depositar, /conta/1/retirar, /conta/1/transferir, /conta/1/encerrar).

```
1 HTTP/1.1 200 OK
2 Content-Type: app/json
```

```
3  Content-Length: ...
4
5  {"conta": {
6    "numero": 1,
7    "saldo": 100,
8    "links": [
9     {"rel": "depositar", "href": "/conta/1/depositar"}
10    {"rel": "retirar", "href": "/conta/1/retirar"}
11    {"rel": "transferir","href": "/conta/1/transferir"}
12    {"rel": "encerrar", "href": "/conta/1/encerrar"}
13   ]
14   }
15 }
```

Código 2.2: Resposta para a requisição do recurso conta bancária com características de HATEOAS.

Como pode ser observado no Código 2.2, a resposta utiliza o princípio fornecido por HATEOAS. Após realizada a requisição, o cliente deve receber o conteúdo solicitado e as possíveis próximas ações que podem ser realizadas. Uma RESTful API que adote essas características em suas respostas pode também ser chamada de Hipermídia API.

A vantagem em se utilizar o princípio fornecido por HATEOAS está no desacoplamento entre cliente e servidor. Esse desacoplamento torna viável a evolução da aplicação cliente e servidor de forma paralela e independente, reduzindo assim esforços para ambas as equipes de desenvolvimento.

2.3 Composição Dinâmica de Serviços Dirigida pelo Usuário

O processo de combinação de serviços para atender uma determinada funcionalidade da aplicação é chamado de composição de serviços (PAPAZOGLOU, 2008). Um cenário que ilustra a composição de serviços é um planejamento de uma viagem, por exemplo. Esse planejamento pode ser composto pelos seguintes serviços: serviço de voo, reserva de hotéis e busca de atrações (SHENG, 2014).

A composição de serviços no âmbito da Computação Orientada a Serviços é realizada através da combinação de WS, pelo fato dessa ser a forma mais promissora de implementar serviços em aplicações (SHENG, 2014). No que diz respeito ao momento em que essa composição de serviços será realizada, de acordo com Papazoglou (2008), Sheng (2014) existem duas formas de classificar, sendo elas: *estática* e *dinâmica*.

A composição estática ocorre durante o projeto da aplicação. Nesse projeto os serviços necessários para criar uma composição devem ser selecionados, combinados, compilados e

implantados na aplicação. Pelo fato dessa composição ocorrer durante atividades do desenvolvimento da aplicação, esse tipo de composição tende a não ser flexível e adaptável, não sendo indicada para aplicações que sofrem constante mudanças. Para o melhor entendimento desse tipo de composição, considere o seguinte exemplo: uma aplicação necessita modificar a utilização de um de seus serviços para um serviço alternativo que seja melhor. Para que isso seja possível, um desenvolvedor deverá efetuar uma manutenção na aplicação para que essa modificação seja contemplada.

Uma composição dinâmica em contrapartida já é indicada para aplicações que sofrem constante mudanças, pelo fato desse tipo de composição permitir que a aplicação se adapte em tempo de execução conforme suas necessidades. Nesse tipo de composição não é necessária a intervenção de um desenvolver para modificar serviços utilizados na aplicação, a aplicação por si só é capaz de efetuar essa modificação. Para exemplificar esse tipo de composição, considere a seguinte situação: uma aplicação utiliza o serviço *Google Maps* para fornecer localização a seus usuários, porém um determinado usuário deseja que a aplicação forneça uma localização utilizando o serviço de localização*Bing Maps*. A aplicação que utiliza a composição dinâmica é capaz de se adaptar durante sua execução para utilizar esse serviço alternativo, suprindo assim a necessidade do usuário.

A composição dinâmica de serviços é dirigida pelo usuário quando o usuário é responsável por personalizar essa composição para atender suas necessidades (SILVA; PIRES; SINDEREN, 2012). Para compreender melhor esse tipo de composição, considere o seguinte exemplo: dois usuários necessitam de informações sobre hospedagem. O primeiro usuário deseja buscar e utilizar um serviço de hospedagem que seja capaz de fornecer informações de hotéis e efetuar a reserva ao mesmo tempo. Para ser capaz de fornecer essa funcionalidade, esse serviço de hospedagem poderia ser composto de um serviço de informações de hotéis e outro serviço de reserva de hotéis. Já um segundo usuário deseja primeiro buscar serviços que forneçam informações sobre hotéis e somente após obter essas informações o usuário solicitará que a aplicação efetue a composição desse serviço com o serviço responsável pelas reservas (SILVA; PIRES; SINDEREN, 2012). Nesse exemplo pode-se observar que os usuários efetuaram a composição do serviço de hospedagem de maneira distinta. Segundo (SILVA; PIRES; SINDEREN, 2012) a *Composição Dinâmica de Serviços Dirigida pelo Usuário* é uma opção para desenvolvimento de aplicações personalizadas para diferentes usuários.

2.3.1 *A-DynamiCoS: A Flexible Framework for User-centric Service Composition*

O framework A-DynamiCoS[6] resultante do trabalho de Silva, Pires e Sinderen (2012) fornece apoio para efetuar a Composição Dinâmica de Serviços Dirigida pelo Usuário. Seu objetivo é realizar a composição de serviços de maneira dinâmica, visando entregar serviços capazes de satisfazer as necessidades de diferentes usuários.

Para obter informações referentes aos serviços disponíveis para serem utilizados no processo de composição, A-DynamiCoS acessa um repositório de serviços descritos semanticamente utilizando SPATEL (ALMEIDA ALMEIDA, 2006). Essas descrições são criadas com base em ontologias contruídas para especificar os objetivos que podem ser alcançados dentro de um determinado domínio, bem como quais os parâmetros necessários para que esses objetivos possam ser alcançados. As descrições dos serviços encontradas nesse repositório contêm informações como: parâmetros de entrada, parâmetros de saída e parâmetros dos objetivos do serviço. O fato dessas descrições serem anotadas semanticamente auxilia A-DynamiCoS na busca de serviços relevantes para o processo de composição de serviços e no fornecimento de informações para a interface do usuário. Uma visão geral da arquitetura do Framework A-DynamiCoS pode ser observada na Figura 2.7.

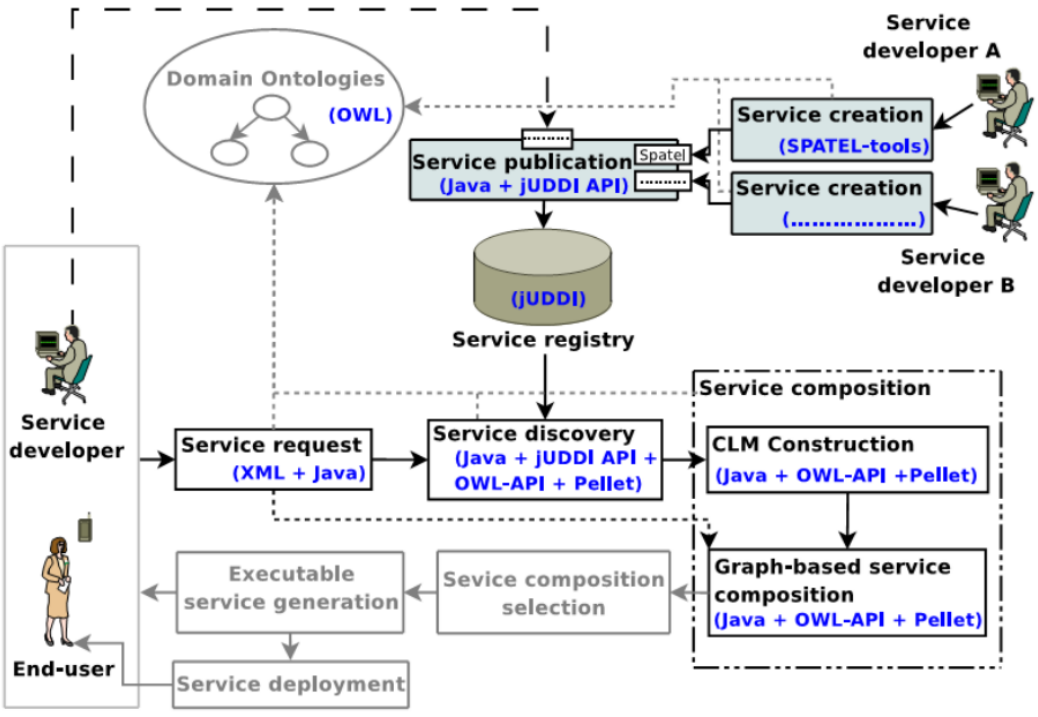

Figura 2.7: Arquitetura do Framework A-DynamiCoS(SILVA; PIRES; SINDEREN, 2012).

[6]http://dynamicos.sourceforge.net/ acessado em 12/05/2016

A-DynamiCoS é exposto como um serviço Web, tornando possível sua interação com diferentes plataformas. Essa interação ocorre via comandos primitivos, cada comando encapsula um pedido para um determinado comportamento que deve ser entregue pelo Framework A-DynamiCoS. Os comandos primitivos apresentados por Silva, Pires e Sinderen (2012) são:

- *ServTypeDiscovery:* Descobre um serviço a partir de um tipo semântico requisitado, por exemplo: um usuário solicita serviços que possuam informações de hotéis na sua atual localização;
- *SeleServ:* Seleciona um serviço para que possa ser usado individualmente ou como parte de uma composição;
- *ExecServ:* Recupera os serviços de composições de serviços que foram montadas e estão prontos para serem executados;
- *ValidateInputs:* Verifica se as entradas inseridas pelo usuário estão de acordo com o que foi solicitado pelo serviço; e
- *AddToContext:* Adiciona no contexto do usuário as entradas, os resultados, entre outras informações que possam ser utilizadas futuramente como sugestões.

A ordem como esses comandos primitivos serão organizados para apoiar uma população de usuários-alvo é chamado de *Fluxo de Comandos*. Um Fluxo de Comandos organiza esses comandos primitivos em um *workflow* adequado para satisfazer os requisitos de uma aplicação, definindo o seu comportamento.

Na Figura 2.8 é ilustrado um exemplo de fluxo de comandos descrito como diagrama de atividades UML. Neste fluxo de comandos o usuário define qual serviço deseja utilizar, interagindo com o DynamiCoS através do comando *ServTypeDiscovery*. O usuário recebe uma lista de serviços, então seleciona um deles e interage com o DynamiCoS através do comando *SeleServ*. Em seguida é apresentada uma interface para o usuário utilizar o serviço selecionado, então o usuário fornece todas as entradas solicitadas pelo serviço e interage com o DynamiCoS através do comando *ValidateInputs* que irá validar as entradas fornecidas e também verificar se o usuário não forneceu alguma das entradas necessárias. Caso alguma não seja fornecida o fluxo de comandos emite o comando *ResolveServ* que irá encontrar serviços que possam ajudar o usuário fornecer essa entrada. Após todas as entradas serem fornecidas, a composição de serviço será executada utilizando o comando *ExecServ*. O comando *AddToContext* é utilizado para armazenar uma composição realizada, para que posteriormente o usuário possa reutilizá-la.

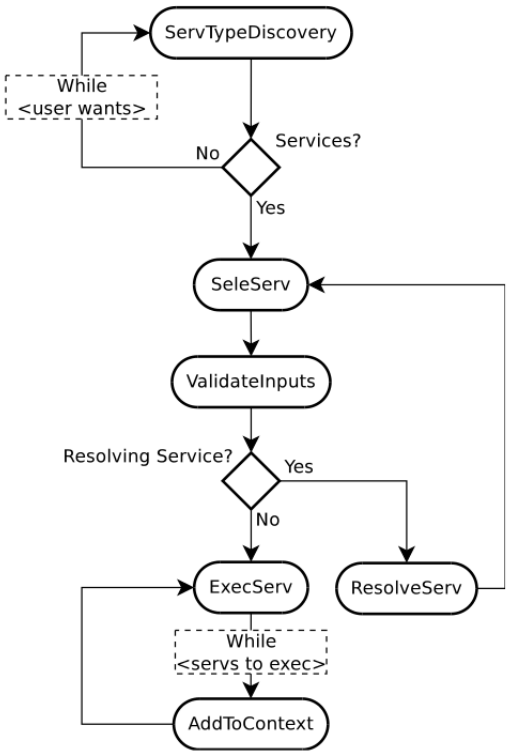

Figura 2.8: Fluxo de Comandos (SILVA; PIRES; SINDEREN, 2012).

Na Figura 2.9 é apresentada a estrutura responsável pela interação do usuário com o A-DynamiCoS. Essa estrutura fornece uma *User Interface* responsável por capturar as intenções do usuário, como por exemplo, a necessidade de encontrar um serviço de hospedagem. Essa intenção será capturada e traduzida para um comando primitivo equivalente, nesse caso *ServTypeDiscovery*. O módulo *User Support* então interage com o módulo *Coodinator* do framework A-DynamiCoS para emitir o comando primitivo. O fluxo de comandos como pode ser observado é implementado no módulo *User Support*, sendo ele responsável por organizar a ordem em que os comandos primitivos devem ser executados.

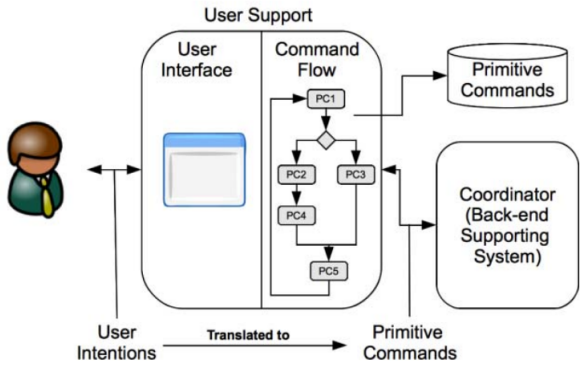

Figura 2.9: *Front-end User Support* (SILVA; PIRES; SINDEREN, 2012).

Capítulo 3

ABORDAGEM

A abordagem desenvolvida neste trabalho visa orientar o desenvolvedor durante o desenvolvimento de aplicações que necessitem se adaptar para suprir as diferentes necessidades dos usuários. A abordagem define quais atividades devem ser realizadas para este tipo de desenvolvimento e recomenda ferramentas para auxiliar nessas atividades. Na Seção 3.1 é fornecida uma visão geral da abordagem; e nas Seções subsequentes são apresentadas todas as atividades definidas na abordagem.

3.1 Abordagem

A abordagem proposta neste trabalho tem por finalidade orientar o desenvolvimento de UDSCAs, utilizando a Composição Dinâmica de Serviços Dirigida pelo Usuário (SILVA; PIRES; SINDEREN, 2012) para suprir as diferentes necessidades dos usuários. Essa abordagem define quais atividades devem ser realizadas para desenvolver este tipo de aplicação e recomenda ferramentas para auxiliar nessas atividades.

Diversos trabalhos para realizar a composição de serviços têm sido relatados na literatura (SHENG, 2014). Alguns desses trabalhos utilizam técnicas e ferramentas responsáveis por efetuar a composição de serviços. Em parceria com a Universidade de Twente foi possível integrar, na abordagem proposta, uma dessas ferramentas. Assim, o framework resultante do trabalho de Silva (SILVA; PIRES; SINDEREN, 2012) denominado *A-DynamiCoS* é utilizado na abordagem apresentada. Esse framework foi escolhido pelo fato de não possuir um ciclo de composição de serviços rígido (isto é, descobrir, compor, implantar e executar, exatamente nessa ordem). Devido a essa flexibilidade, *A-DynamiCoS* torna viável o desenvolvimento de aplicações que necessitem de adaptação dinâmica para atender as necessidades do usuário.

Uma visão geral da abordagem é apresentada no diagrama SADT (*Structured Analysis and*

Design Technique) (ROSS, 1977) da Figura 3.1. As atividades definidas na abordagem são divididas em Engenharia de Domínio e Engenharia de Aplicação (PRESSMAN, 2010). Na Engenharia de domínio as atividades são responsáveis por gerar os artefatos reutilizáveis no desenvolvimento de UDSCas de um determinado domínio. Na Engenharia de Aplicação as atividades são responsáveis pelo desenvolvimento de aplicações dirigidas pelo usuário, fazendo reuso dos artefatos gerados pela Engenharia de Domínio.

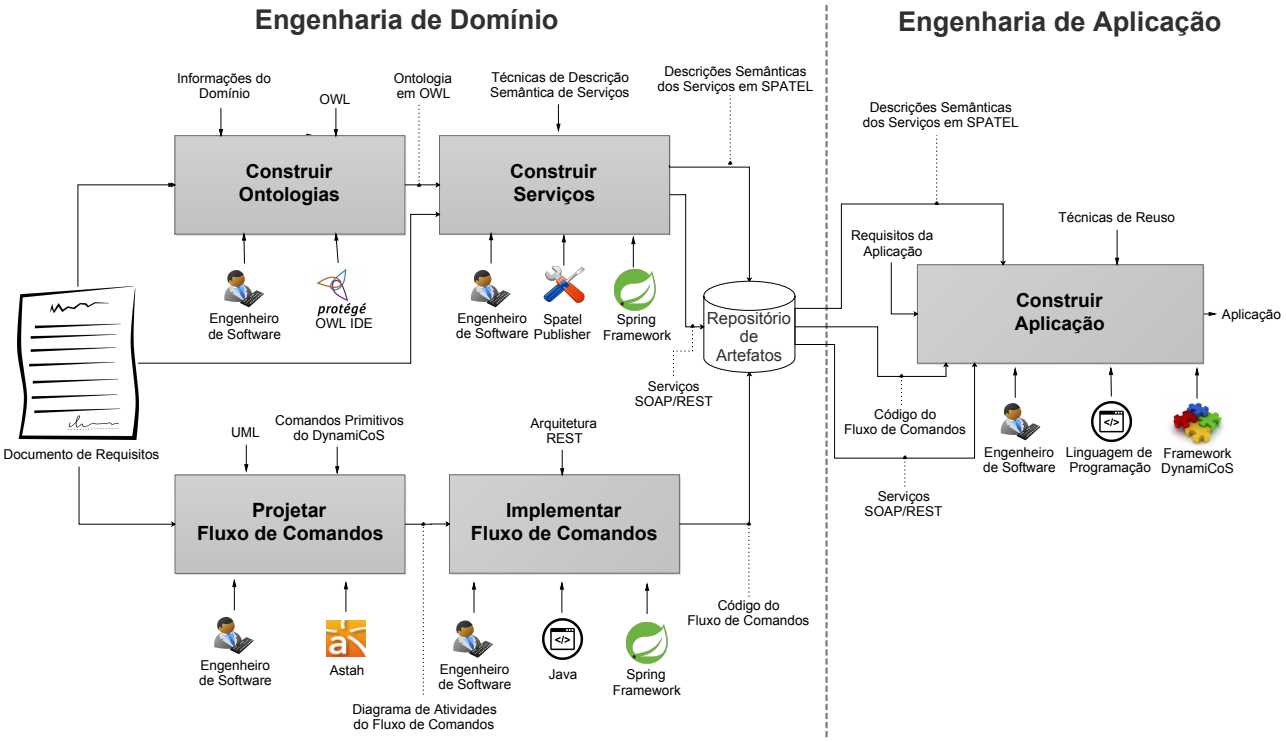

Figura 3.1: Visão geral da abordagem.

Para facilitar o entendimento da abordagem, suas atividades são apresentadas através de exemplos do domínio de Turismo. Nesse domínio pessoas são capazes de planejar uma viagem através da composição de serviços de entretenimento, hospedagem e localização. As atividades da abordagem são apresentadas com mais detalhes nas subseções a seguir, iniciando pela atividade Construir Ontologias.

3.1.1 Construir Ontologias

Nesta atividade o Engenheiro de Software constrói as ontologias do domínio do problema. Essas ontologias contém informações que são utilizadas para descrever e buscar os serviços semanticamente. Para apoiar a criação dessas ontologias é fornecido como entrada um docu-

mento de requisitos[1] contendo informações sobre o domínio do problema. Esse documento é especificado textualmente e contém informações que descrevem os objetivos a serem atendidos pelas aplicações desse domínio. O Engenheiro de Software nessa atividade é apoiado pela ferramenta *protégé OWL IDE* [2], e orientado por técnicas de construção de ontologias, no caso a OWL (MICHAEL K. SMITH DEBORAH MCGUINESS; WELTY, 2002) considerando que essa linguagem é adotada como padrão pela W3C[3]. As seguintes ontologias são construídas:

- Ontologia de Objetivos: Especifica os objetivos que podem ser alcançados num determinado domínio.
- Ontologia de Parâmetros: Especifica os parâmetros de entrada e saída necessários para alcançar um objetivo num determinado domínio.

Essas ontologias são básicas para o desenvolvimento das funcionalidades dos serviços que são reutilizados na Engenharia de Aplicação. A Figura 3.2 mostra um exemplo dessas ontologias para o domínio de turismo. Por exemplo, para o objetivo *EncontrarHotel* tem-se as entradas *cidade, dataEntrada, dataSaida, quantidaDePessoas e quantidaDeQuartos* e a saída *endereco e cidade*.

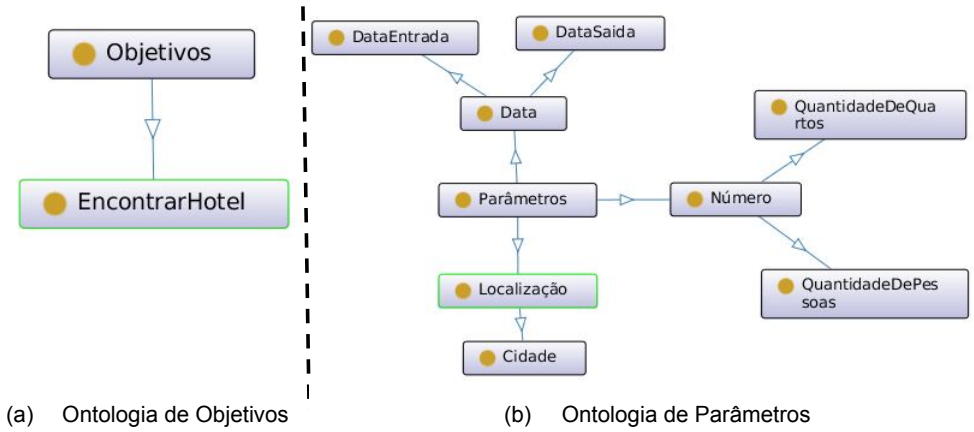

Figura 3.2: Fragmentos das Ontologias de Objetivos e Parâmetros.

3.1.2 Construir Serviços

Nesta atividade, os serviços que serão utilizados pelas aplicações do domínio, são construídos e descritos semanticamente com base nas informações do documento de requisitos e

[1]Parte da definição desse documento de requisitos é fundamentada no trabalho apresentado por Vlieg (VLIEGS, 2010)

[2]http://protege.stanford.edu/ acessado em 04/03/2016

[3]http://www.w3.org/ acessado em 04/03/2016

ontologias. Caso o serviço já exista, esse é integrado ao domínio. Os serviços e descrições geradas nessa atividade são publicadas num repositório pelo Engenheiro de Software viabilizando assim sua reutilização.

A construção dos serviços é realizada da seguinte forma: (1) o Engenheiro de Software especifica e implementa os requisitos para atender cada objetivo do domínio do problema; (2) o código gerado é empacotado como um serviço web e (3) o serviço web é publicado num repositório de serviços. A abordagem adotou para a construção dos serviços o *Spring Framework*[4], pelo fato de suportar a construção de serviços web em SOAP e REST.

Para que os serviços construídos possam ser descobertos e reutilizados, cada serviço é descrito semânticamente e essa descrição é publicada num repositório. Para efetuar essa descrição semântica a abordagem adotou a linguagem SPATEL (ALMEIDA, 2006), a qual permite anotar as seguintes operações: parâmetros de entrada, saída, pré-condições, efeitos, objetivos do serviço e propriedades não-funcionais. O Código 3.1 mostra um exemplo dessa descrição semântica, onde o objetivo do serviço foi anotado através dos atributos `name="OperationGoal"` e `semType="Objetivos.owl:EncontrarHotel"` (Linhas 15 e 17) e suas respectivas entradas e saídas através das tags `<ownedParameter />` (Linhas 6 à 13). Como pode-se observar os valores dos atributos `semType=` são fundamentados nas ontologias.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <spatel:ServiceLibrary>
3      <service name="EncontrarHotel">
4        <ownedOperation name="EncontrarHotel">
5          <!-- Input -->
6              <ownedParametername="data_entrada" semType="Parametros.owl:DataEntrada"/>
7              <ownedParameter name="data_saida" semType="Parametros.owl:DataSaida"/>
8          <ownedParameter name="cidade" semType="Parametros.owl:Cidade"/>
9          <ownedParameter name="quantidade_quartos"
               semType="Parametros.owl:QuantidadeDeQuartos"/>
10         <ownedParameter name="quantidade_pessoas"
               semType="Parametros.owl:QuantidadeDePessoas"/>
11
12         <!-- Output -->
13         <ownedParameter semType="Parametros.owl:Endereco"/>
14         <ownedParameter semType="Parametros.owl:Cidade"/>
15
16         <semTag xmi:id="id" name="OperationGoal"
               semType="Objetivos.owl:EncontrarHotel" kind="goal"/>
17       </ownedOperation>
18       <semTag xmi:id="id" name="ServiceGoal" semType="Objetivos.owl:EncontrarHotel"
             kind="objetivo"/>
19       <ontology xmi:id="id" name="Objetivos.owl"
             uri="http://localhost:8080/ontologies/Objetivos.owl"/>
20       <ontology xmi:id="id" name="Parametros.owl"
             uri="http://localhost:8080/ontologies/Parametros.owl"/>
21       <ontology xmi:id="id" name="core.owl"
             uri="http://localhost:8080/ontologies/core.owl"/>
22     </service>
```

[4]http://projects.spring.io/spring-framework/ acessado em 13/07/2016

```
23 </spatel:ServiceLibrary>
```

Código 3.1: Descrição Semântica em SPATEL.

As descrições semânticas contruídas devem então ser publicadas num *Universal Description, Discovery and Integration (UDDI)*. Para auxiliar nesse processo de publicação a abordagem adotou o projeto *Spatel Publisher* [5]. Spatel Publisher simplifica o processo de publicação por automatizar parte das etapas do processo de publicação e também permitir a publicação de descrições semânticas em SPATEL. Para efetuar a publicação das descrições semânticas através do Spatel Publisher é necessário fornecer os seguintes parâmetros:

- Descrição: uma breve descrição em linguagem natural sobre o serviço e seus objetivos;
- URL do Serviço: endereço do serviço na rede, seja a rede internet ou intranet; e
- URL da Descrição Semântica: endereço do arquivo criado para descrever o serviço semanticamente.

Após fornecido esses parâmetros, o Spatel Publisher registra a descrição semântica do serviço no UDDI. A extensão do jUDDI[5]fornecido por Silva (SILVA; PIRES; SINDEREN, 2011), foi adotado pela abordagem por permitir o armazenamento de todas anotações semânticas necessárias para descrever um serviço.

3.1.3 Projetar Fluxo de Comandos

Nesta atividade o fluxo de comandos é modelado pelo Engenheiro de Software em Unified Modeling Language (UML), usando a técnica do Diagrama de Atividades[6]. Esse fluxo de comandos define a ordem de chamada dos comandos primitivos fornecidos pelo Framework A-DynamiCoS para um determinado domínio. Essas chamadas são realizadas pelas aplicações desenvolvidas para apoiar o usuário final na realização dos seus objetivos nesse domínio. A definição desse fluxo de comandos é fundamentada no documento de requisitos (especificação textual) e deve ser capaz de cobrir todos os cenários especificados nesse documento.

A modelagem do fluxo de comandos é realizada da seguinte forma: (1) os comandos primitivos são representados na forma de retângulos; (2) o fluxo entre os comandos primitivos é representado através de setas que indicam a origem e destino; e (3) os desvios condicionais

[5]http://sourceforge.net/projects/dynamicos/files/ acessado em 10/07/2016
[6]http://www.uml.org/ acessado em 12/07/2016

são representados na forma de losangos. A UML foi adotada pela abordagem por dar suporte, adequado a este tipo de modelagem.

Para auxiliar nessa modelagem, a abordagem adotou a ferramenta *Astah*[7], que tem uma interface gráfica para modelar o diagrama de atividades da UML de maneira simples e prática. Um exemplo desse diagrama pode ser visto na Figura 3.3, que representa as atividades do fluxo de comandos para descoberta de serviço.

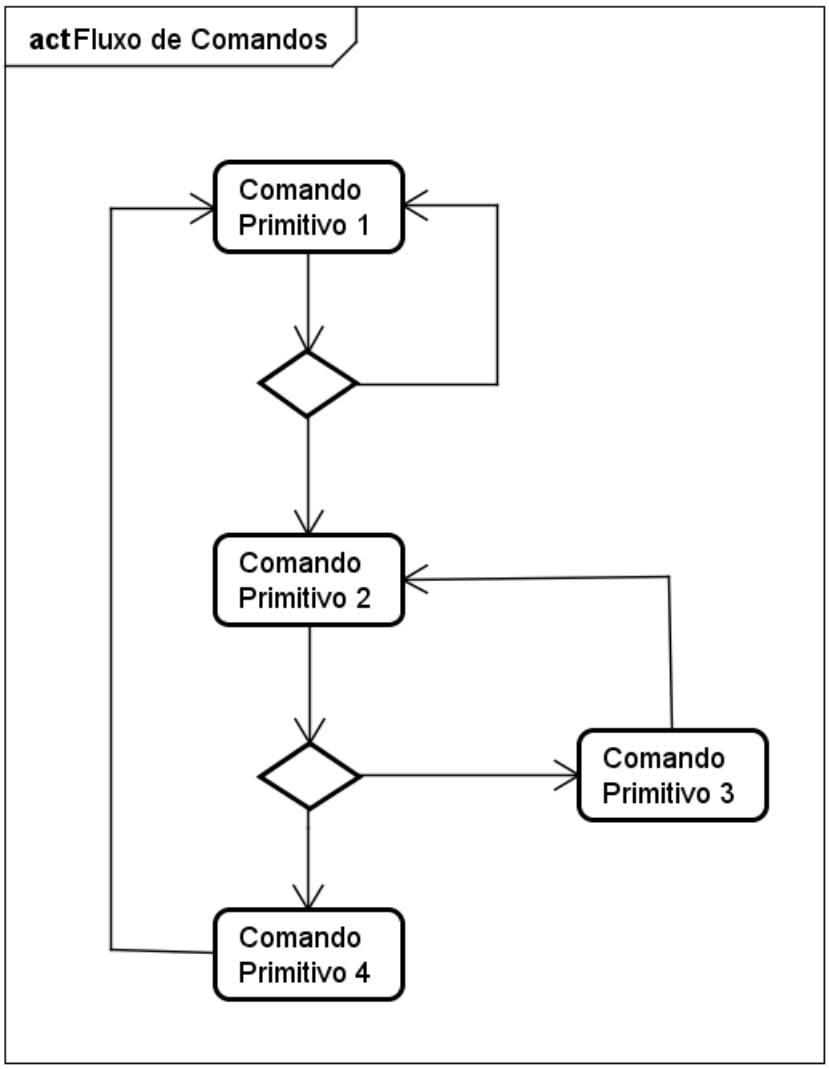

Figura 3.3: Diagrama de Atividades do Fluxo de Comandos.

3.1.4 Implementar Fluxo de Comandos

Nesta atividade o fluxo de comandos é codificado e fornecido como uma API RESTful. A finalidade principal dessa API RESTFul é atuar como um middleware independente de plata-

[7]http://astah.net/ acesso em 13/07/2016

forma entre o Framework A-DynamiCoS e a aplicação desenvolvida auxiliando o usuário final à alcançar seus objetivos.

A codificação do fluxo de comandos é realizada pelo Engenheiro de Software da seguinte forma: (1) é analisado o diagrama de atividades fornecido como entrada para a atividade, para identificar quais os comandos primitivos disponibilizados pelo Framework A-DynamiCoS serão reutilizados; (2) cada comando primitivo identificado é codificado na linguagem Java; (3) Com foco na reutilização o código gerado de cada comando primitivo é disponibilizado como um recurso REST num repositório de serviços. Adotou-se o Framework Spring pelo fato de fornecer suporte para expor código Java como recursos REST de maneira simples e fácil; (4) é analisado o diagrama de atividades visando identificar qual a ordem de invocação dos comandos primitivos; (5) são adicionados nas respostas dos recursos, os endereços dos próximos recursos que podem ser invocados, através de mecanismos disponibilizados pelo Framework Spring HATEOAS, seguindo o princípio HATEOAS; e (6) esses recursos são disponibilizados através de uma *Uniform Resource Identifier (URI)*, viabilizando sua reutilização em diferente plataformas através de um conjunto de operações fornecidas pelo protocolo HTTP.

Um trecho de código, gerado para reutilizar o comando primitivo ServTypeDiscovery, pode ser observado no Código 3.2. Esse código é exposto como recurso REST (Linha 1) pela classe `CommandFlowController` (Linha 2) e pode ser requisitada através da URI `"/serv_type_discovery"` (Linha 4), essa URI irá produzir conteúdo em formato JSON devido ao valor `"application/json"` presente na tag `produces`. O conteúdo presente na resposta é montado no método `servTypeDiscoveryAsJson()` possui os seguintes objetos: `DynamicosConfiguration` (Linha 8) possui as configurações necessárias para conectar-se ao Framework A-DynamiCoS, *SoapConsumer* (Linha 10 e 12) monta a requisição para o Framework A-DynamiCoS com os parâmetros necessários e efetua a comunicação, `CoordinateResponse` (Linha 12) obtém a resposta retornada pelo Framework A-DynamiCoS e `ServTypeDiscoveryResponse` objeto que é preenchido com o conteúdo da resposta e que será retornado (Linha 15 e 20), esse objeto é também responsável por adicionar em seu conteúdo qual o próximo recurso que poderá ser invocado (Linha 17 e 18).

```
1 @RestController
2 public class CommandFlowController{
3
4     @RequestMapping(value = "/serv_type_discovery.json", produces =
          "application/json")
5     @ResponseBody
6     public HttpEntity<ServTypeDiscoveryResponse> servTypeDiscoveryAsJson()
          throws IOException{
7
```

```
8       DynamicosConfiguration config = new DynamicosConfiguration();
9       Jaxb2Marshaller marshaller = config.marshaller();
10      SoapConsumer consumer = config.soapConsumer(marshaller);
11
12      CoordinateResponse response =
            consumer.getResponseDynamicos(EnumPrimitiveCommand.SERV_TYPE_DISCOVERY);
13
14      InputStream inputStream = new
            ByteArrayInputStream(response.getReturn().getValue().getBytes());
15      ServTypeDiscoveryResponse servTypeDiscovery = JAXB.unmarshal(inputStream,
            ServTypeDiscoveryResponse.class);
16
17      servTypeDiscovery.add(linkTo(methodOn(CommandFlowController.class)
18      .servTypeDiscoveryAsJson()).withSelfRel());
19
20      return new ResponseEntity<ServTypeDiscoveryResponse>(servTypeDiscovery,
            HttpStatus.OK);
21
22  }
23 }
```

Código 3.2: Código do Fluxo de Comandos.

Mais detalhes referentes as classes CommandFlowController, DynamicosConfiguration e SoapConsumer podem ser encontrados no Anexo.

Um exemplo de resposta do código do fluxo de comandos pode ser observada no Código 3.3 através de uma requisição ao recurso `servTypeDiscovery`. Essa requisição foi realizada através da operação GET do protocolo HTTP com o valor do parâmetro `serviceType` definido como EncontrarHotel. O serviço encontrado, para o parâmetro solicitado, está localizado no conteúdo da resposta (Linha 2-5) em conjunto com os próximos recursos que podem ser invocados (Linha 6-8).

```
1  {
2     "service":[
3        {
4           "service_id":1,
5           "name":"EncontrarHotel",
6           "goal":"EncontrarHotel",
7           "links":[
8              {
9                 "rel":"sele_serv",
10                "href":"http://localhost:8080/sele_serv"
11             }
12          ]
13       }
14    ]
15 }
```

Código 3.3: Resposta do Fluxo de Comandos.

3.1.5 Construir Aplicação

Nessa atividade uma vez concluída a Engenharia de Domínio, é desenvolvida a aplicação que irá ajudar o usuário final a alcançar seus objetivos num determinado domínio. No desenvolvimento dessa aplicação são realizadas as seguintes atividades: reutilização dos artefatos disponibilizados pela engenharia de domínio, visando reduzir esforços no desenvolvimento; desenvolvimento da interface gráfica de interação com o usuário; e desenvolvimento do código específico da aplicação.

A reutilização dos artefatos disponibilizados pela engenharia de domínio é realizada via operações HTTP (POST, GET, PUT e DELETE), pelo fato desses artefatos serem disponibilizados como API RESTful. O Engenheiro de Software deve decidir como a aplicação tratará as repostas dos serviços, de forma a atender seus objetivos.

O desenvolvimento da interface gráfica de interação com o usuário e código específico (por exemplo, controle de usuário) podem ser desenvolvidos em qualquer plataforma, ficando a critério da equipe de desenvolvimento escolher qual a mais apropriada para alcançar seus objetivos. Essa flexibilidade ocorre pelo fato dos artefatos que fornecem o conteúdo para a aplicação serem desenvolvidos como serviços web, tornando-os assim acessíveis em diferentes plataformas. Uma atividade importante a ser realizada nesta etapa é a associação dos componentes gráficos e seus eventos com as ontologias de objetivos e parâmetros. Por exemplo, quando um botão com o objetivo de encontrar um hotel é clicado, o evento de clique do botão deve estar associado ao objetivo *Objetivos.owl:EncontrarHotel*, para que seja possível identificar qual objetivo deverá ser requisitado.

Um exemplo de como a aplicação produzida pela abordagem é executada por um usuário final pode ser observada na Figura 3.4. Nesse exemplo o usuário segue os seguintes passos para executar a aplicação: (1) faz uma solicitação para a aplicação encontrar um serviço que disponibilize informações sobre hotéis; (2) a aplicação retorna uma lista de serviços capazes de fornecer esse tipo de informação. Prosseguindo, o o usuário escolhe um desses serviços (no caso hotelurbano.com) e solicita que a aplicação se conecte a este serviço; (3) a aplicação faz a conexão e é apresentado para o usuário uma interface para fornecer as entradas necessárias para utilizar o serviço. Caso o usuário não seja capaz de fornecer todas as entradas, a aplicação busca serviços capazes de ajudar o usuário a fornecer essas entradas, efetuando assim a composição do serviço anterior com outros serviços conforme a necessidade do usuário (composição dirigida pelo usuário); e (4) a aplicação executa o serviço retornando seus resultados para o usuário final.

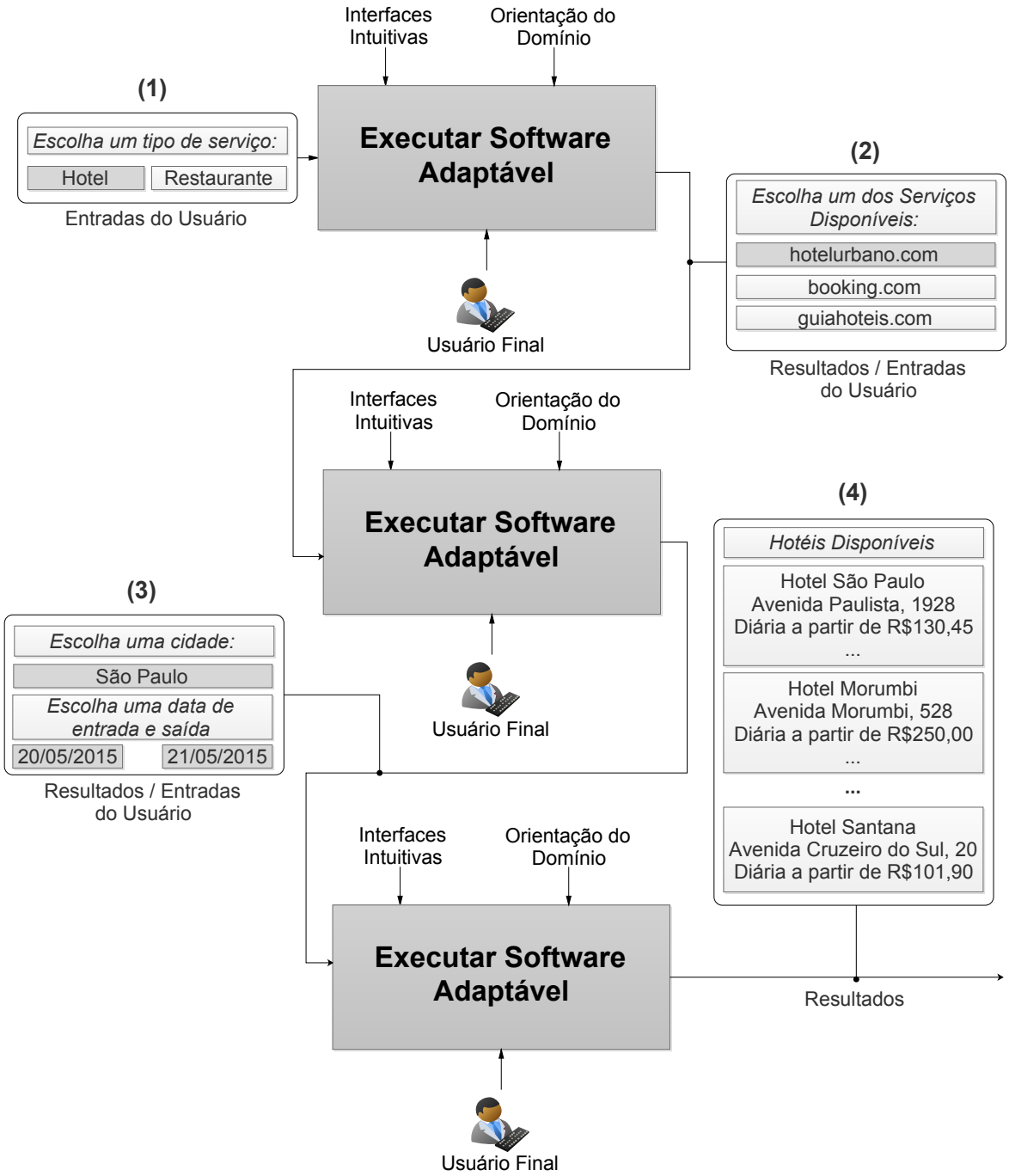

Figura 3.4: Execução da Aplicação.

Capítulo 4

ESTUDO DE CASO

Neste capítulo é apresentado um estudo de caso da abordagem que compreende o desenvolvimento de uma UDSCA no domínio de serviços urbanos. Na seção 4.1 é detalhado como foi realizado o desenvolvimento orientado pela abordagem desenvolvida; e na seção 4.2 é apresentada a avaliação dos resultados obtidos neste estudo de caso.

4.1 User-Driven Service Composition Application

O estudo de caso consistiu no desenvolvimento de uma UDSCA no domínio de serviços de Manutenção Predial. Essa UDSCA tem como objetivo anunciar, encontrar e planejar serviços de manutenção de imóveis numa determinada zona urbana. Para alcançar esses objetivos, a aplicação faz uso de 28 Web Services construídos utilizando REST/SOAP, sendo 3 Web Services responsáveis por fornecer informações de localização.

Como a finalidade de uma UDSCA é atender as diferentes necessidades dos usuários, a mesma foi desenvolvida para combinar esses Web Services durante sua execução, com o intuito de adaptar-se ao usuário que a utiliza. Por exemplo, ao anunciar ou procurar um serviço de manutenção predial o usuário pode necessitar de ajuda no que se diz respeito a sua localização, sendo assim os Web Services responsáveis por anunciar e buscar serviços de manutenção predial podem ser compostos com os Web Services responsáveis por fornecer informações de localização para auxiliar os usuários. A busca e composição desses serviços é realizada pelo Framework A-DynamiCoS.

O desenvolvimento dessa UDSCA foi orientado pelas atividades definidas pela abordagem e realizado pelo Engenheiro de Software conforme apresentado a seguir:

4.1.1 Construir Ontologias

Nesta atividade foi fornecido um documento de requisitos (Exemplo - Figura 4.1) contendo informações do domínio, seus usuários e objetivos que podem ser alcançados.

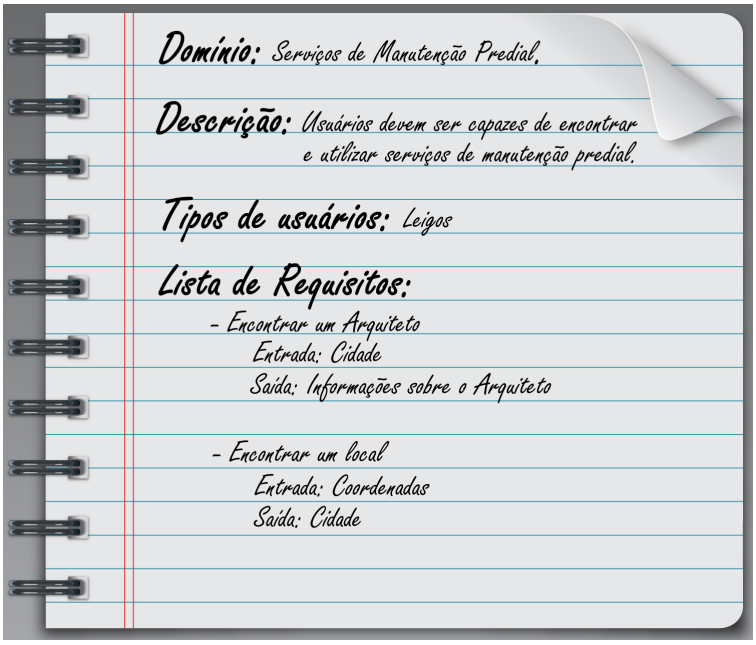

Figura 4.1: Documento de Requisitos com informações sobre o domínio

A partir dessas informações foram construídas as ontologias de objetivo e parâmetros (Figura 4.2 e 4.3) apoiada pela técnica de construção de ontologias OWL e pela ferramenta protégé IDE.

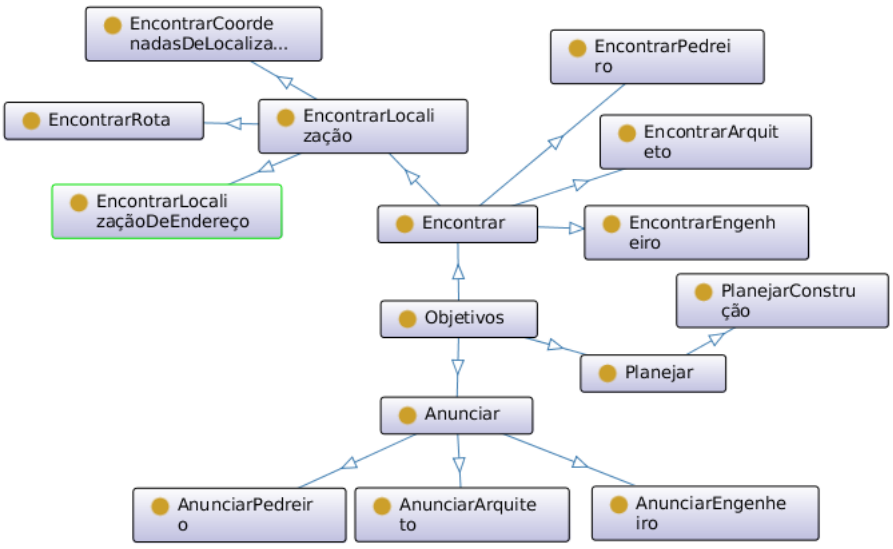

Figura 4.2: Ontologias de Objetivos.

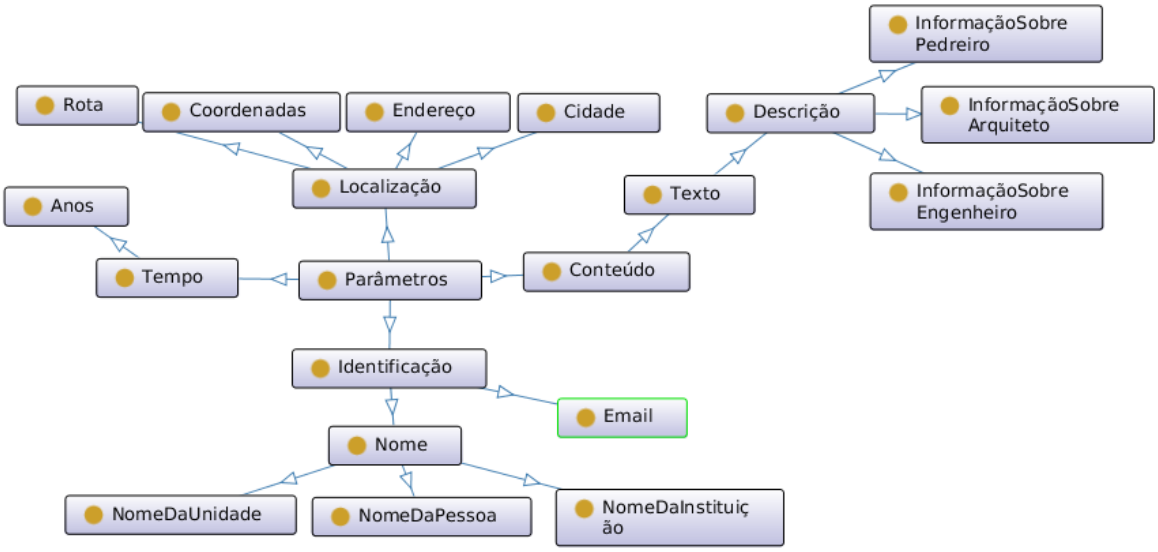

Figura 4.3: Ontologia de Parâmetros.

4.1.2 Construir Serviços

Nesta atividade foram extraídas informações do documento de requisitos e da ontologia de objetivos visando identificar os serviços necessários. Com base nos objetivos definidos, foram identificados que eram necessários implementar/reutilizar, os seguintes serviços:

- AnunciarArquiteto;
- AnunciarEngenheiro;
- AnunciarPedreiro;
- EncontrarArquiteto;
- EncontrarEngenheiro;
- EncontrarPedreiro;
- EncontrarRota;
- EncontrarEnderecoDeCoordenadas;
- EncontrarCoordenadasDeEndereco; e
- PlanejarConstrucao.

Os serviços responsáveis por fornecer dados de localização foram reutilizados da API fornecida pelo Google Maps e os demais foram desenvolvidos com apoio da ontologia de parâmetros e da ferramenta Spring Framework. Após concluído o desenvolvimento, esses serviços foram empacotados como serviços web e publicados num repositório de serviços.

Para cada serviço também foi gerada uma descrição semântica (Código 4.1), na linguagem SPATEL. Nessa descrição cada parâmetro de entrada e saída foi anotado com base na ontologia de parâmetros, e o objetivo do serviço foi anotado com base na ontologia de objetivos.

```
<?xml version="1.0" encoding="UTF-8"?>
<spatel:ServiceLibrary>
    <service name="EncontrarArquiteto">
      <ownedOperation name="EncontrarArquiteto">
        <!-- Input -->
            <ownedParametername="nome" semType="Parametros.owl:NomeDaPessoa"/>
            <ownedParameter name="cidade" semType="Parametros.owl:cidade"/>
        <ownedParameter name="anos" semType="Parametros.owl:Anos"/>
        <ownedParameter name="coordenadas" semType="Parametros.owl:Coordenadas"/>

        <!-- Output -->
        <ownedParameter semType="Parametros.owl:InformacaoSobreArquiteto"/>

        <semTag xmi:id="id" name="OperationGoal"
            semType="Objetivos.owl:InformacaoSobreArquiteto" kind="goal"/>
      </ownedOperation>
      <semTag xmi:id="id" name="ServiceGoal"
          semType="Objetivos.owl:EncontrarArquiteto" kind="goal"/>
      <ontology xmi:id="id" name="Goals.owl"
          uri="http://localhost:8080/ontologies/Goals.owl"/>
      <ontology xmi:id="id" name="IOTypes.owl"
          uri="http://localhost:8080/ontologies/IOTypes.owl"/>
      <ontology xmi:id="id" name="core.owl"
          uri="http://localhost:8080/ontologies/core.owl"/>
    </service>
</spatel:ServiceLibrary>
```

Código 4.1: Descrição Semântica em SPATEL para o serviço Encontrar Arquiteto.

Após realizadas as descrições semânticas dos serviços, essas foram publicadas num UDDI com o apoio da ferramenta de publicação Spatel Publisher. Essa ferramenta ao ser executada registrou no UDDI as descrições semânticas desses serviços. Para cada serviço foi incluída uma breve descrição sobre seus objetivos e o endereço do serviço e sua descrição semântica na rede.

4.1.3 Projetar Fluxo de Comandos

Nessa atividade foi analisado o documento de requisitos para descobrir todos os possíveis caminhos para alcançar os objetivos do domínio. Para cobrir todos os cenários descobertos, foi projetado um fluxo de comandos que define a ordem em que os comandos primitivos devem ser invocados pelas aplicações. Essas aplicações são responsáveis por auxiliar o usuário final a

alcançar os objetivos do domínio. O projeto desse fluxo de comandos foi realizado utilizando o diagrama de atividades da UML com o apoio da ferramenta Astah (Figura 4.4).

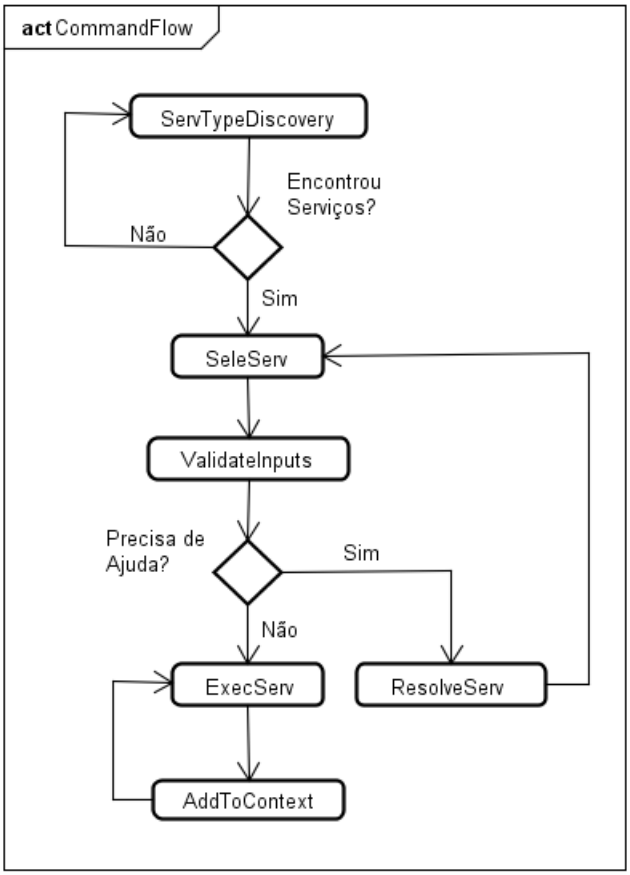

Figura 4.4: Diagrama de Atividades do Fluxo de Comandos. Adaptado de (SILVA; PIRES; SINDEREN, 2012)

4.1.4 Implementar Fluxo de Comandos

Nesta atividade foram implementadas as atividades do diagrama e disponibilizadas como recursos REST. Cada atividade associou-se a um comando primitivo disponibilizado pelo Framework A-DynamiCoS. Prosseguindo, foi gerado o código com foco na reutilização (o mesmo apresentado no Código 3.2 da Abordagem). Para codificar a lógica das atividades foram utilizadas a linguagem Java e o Framework Spring HATEOAS para adicionar à respostas dos recursos informações sobre quais os próximos recursos que podem ser invocados. Esse conjunto de recursos foi disponibilizado como uma API RESTful que será reutilizada pelos desenvolvedores responsáveis por desenvolver a aplicação que atenderá os usuários finais.

4.1.5 Construir Aplicação

Nesta atividade foi desenvolvida uma aplicação cliente na plataforma Android fazendo reuso dos artefatos gerados pela Engenharia de Domínio. O desenvolvimento compreendeu na construção da interface gráfica (Figura 4.5) responsável pela interação com o usuário e código específico para reutilização dos artefatos gerados pela Engenharia de Domínio.

O evento de cada botão foi associado a um objetivo (especificado na ontologia de objetivos), para que seja possível identificar qual o tipo de serviço que deverá ser buscado. Para cada entrada e saída foi associado um tipo (especificado na ontologia de parâmetro), essa associação permite buscar serviços que complementam as entradas a serem fornecidas pelo usuário.

A aplicação final é executada pelo usuário final da seguinte maneira: (1) o usuário escolhe qual objetivo quer realizar (exemplo: Encontrar Arquiteto); (2) uma lista de serviços capazes de

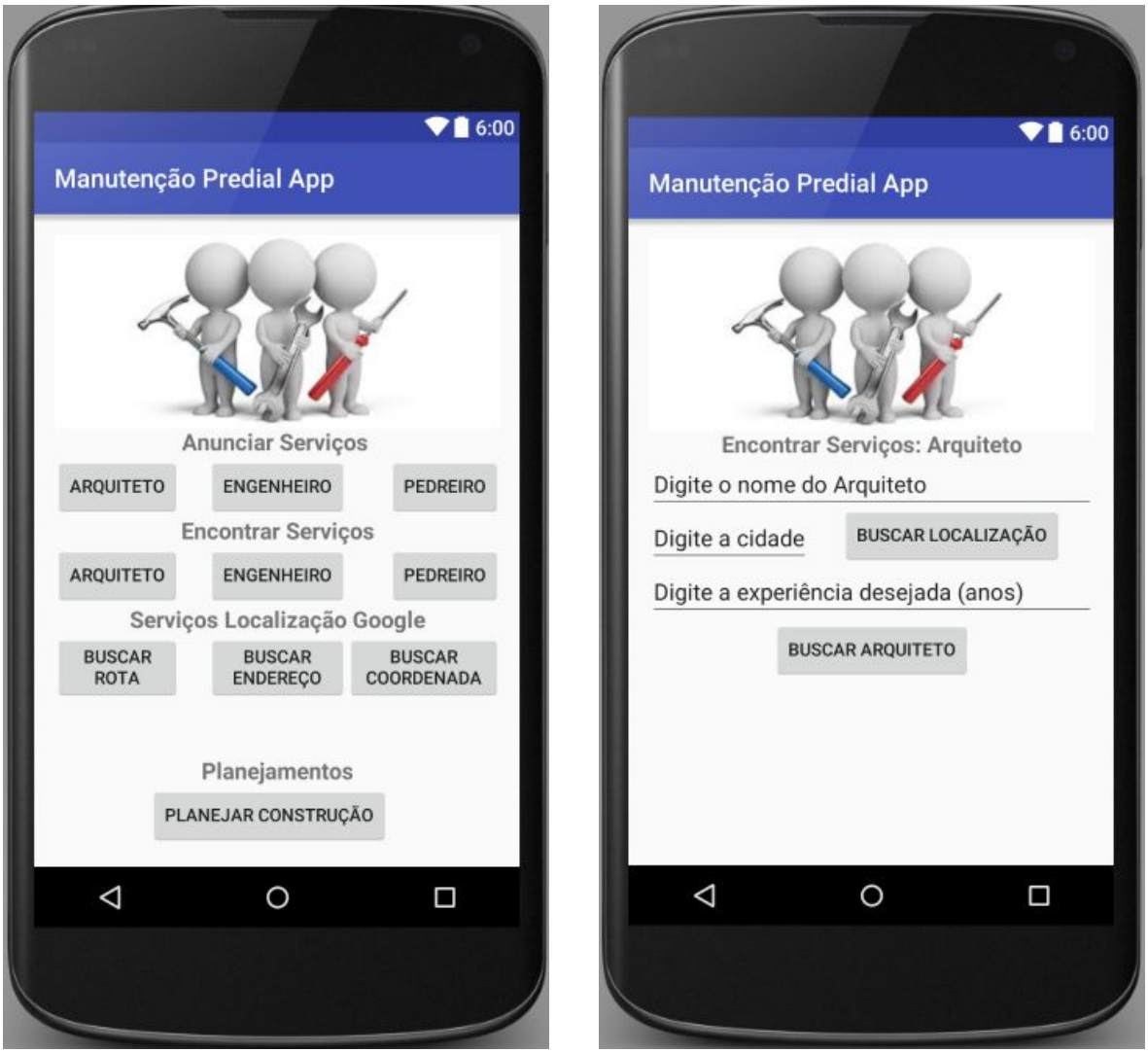

Figura 4.5: Interface Gráfica de Interação com o Usuário.

realizar esse objetivo é retornada e o usuário escolhe o serviço que deseja utilizar; (3) o usuário fornece as entradas solicitadas, caso o mesmo não seja capaz de fornecer sua localização, o serviço é composto com outro serviço de localização orientado pelo usuário; e (4) a aplicação retorna os resultados para o usuário.

4.2 Avaliação

Para avaliar a abordagem desenvolvida neste trabalho, foi realizado um experimento com 10 Engenheiros de Software (ES), os quais não possuiam conhecimento sobre as técnicas utilizadas neste projeto de mestrado. Esses integrantes foram divididos em dois grupos, onde foi avaliado aspectos do desenvolvimento de uma aplicação com e sem a utilização desta abordagem. O primeiro grupo (Grupo A) iniciou o desenvolvimento sem a utilização da abordagem e posteriormente utilizando-a, já o segundo grupo (Grupo B) iniciou o desenvolvimento com a utilização da abordagem e posteriormente sem utilizá-la. Dessa forma foi possível comparar os resultados obtidos afim de evidenciar as contribuições proporcionadas pela abordagem.

Na avaliação foi proposto o desenvolvimento da aplicação utilizada como base para o estudo de caso apresentado nesta seção. O desenvolvimento foi dividido em duas etapas, onde na primeira etapa o Grupo A realizou o desenvolvimento sem o apoio da abordagem, enquanto o Grupo B realizou a mesmo desenvolvimento com o apoio da abordagem. De forma análoga, na segunda etapa o Grupo A passou a utilizar a abordagem no desenvolvimento, enquanto que o Grupo B passou a desenvolver sem a utilização da abordagem. Essas etapas foram realizas em dias diferentes, com o intuito de não sobrecarregar os participantes, visto que para cada etapa foi previsto um tempo máximo de desenvolvimento de 4 horas. Para ambas as etapas os tempos despendidos no desenvolvimento da aplicação foram cronometrados para cada participante.

No desenvolvimento com apoio da abordagem, os desenvolvedores seguiram as atividades definidas na abordagem e as ferramentas recomendadas, enquanto que no desenvolvimento sem apoio da abordagem, os desenvolvedores seguiram o processo "ad-hoc"de desenvolvimento partindo dos mesmos requisitos, seguindo as atividades tradicionais de Análise, Projeto e Implementação.

Como pode ser observado na Tabela 4.1 o uso da abordagem resultou na redução do tempo de desenvolvimento da aplicação. Esse ganho deve-se ao fato dos desenvolvedores seguirem as atividades da abordagem com o apoio das ferramentas nela integradas. Dessa forma, não despenderam tempo pesquisando quais atividades devem ser realizadas para desenvolver a aplicação, bem como quais ferramentas e tecnologias necessárias para auxiliar nas atividades.

Tabela 4.1: Resultados obtidos da análise manual.

Grupo	ES	Tempo em minutos	
		Com abordagem	Sem Abordagem
A	1	151	212
A	2	138	198
A	3	132	202
A	4	164	223
A	5	123	182
B	6	131	197
B	7	148	217
B	8	173	237
B	9	161	189
B	10	121	228
Tempo Médio		**144.2**	**208.5**

Para melhor visualizar a redução de tempo de desenvolvimento proporcionada pela abordagem, os resultados apresentados na Tabela 4.1 foram base para desenvolver o gráfico da Figura 4.6, o qual deixa evidente que o tempo necessário para desenvolver a aplicação sem o uso da abordagem foi maior para todos os participantes, independente do grupo do qual ele fazia parte.

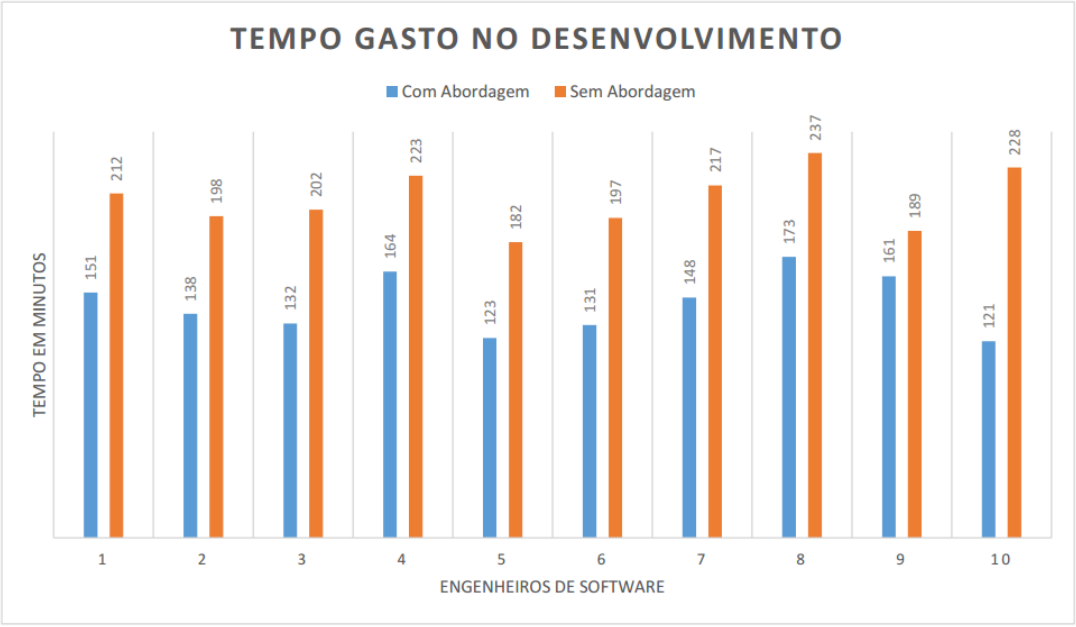

Figura 4.6: Tempo Gasto no Desenvolvimento.

Com o intuito de fazer uma estimativa da eficiência proporcionada pela abordagem, foi calculado o tempo necessário para realizar o desenvolvimento da aplicação sem o apoio da

abordagem fracionado pelo tempo necessário para realizar o desenvolvimento da aplicação com o apoio da abordagem, conforme a Fórmula 4.1.

$$\text{Eficiência} = \frac{\text{Tempo de desenvolvimento sem apoio da abordagem}}{\text{Tempo de desenvolvimento com apoio da abordagem}} \tag{4.1}$$

Comparado ao processo de desenvolvimento sem o uso abordagem, o desenvolvimento com o apoio da abordagem teve uma eficiência de 44%, conforme pode ser observado na Fórmula 4.2 .

$$\text{Eficiência} = \frac{208.5 \text{ minutos}}{144.2 \text{ minutos}} \approx 1.44 = 44\% \tag{4.2}$$

Outro ponto importante da abordagem se deve ao fato de que ao realizar futuros desenvolvimentos de aplicações para o mesmo dóminio em plataformas diferentes, o tempo despendido para desenvolver os artefatos da Engenharia de Domínio será reduzido, pelo fato desses artefatos já terem sido construídos em desenvolvimentos anteriores e serem reutilizáveis. Esse cenário reduz a quantidade de código que o desenvolvedor precisará implementar.

Os resultados obtidos são considerados significativos, o que comprova a aplicabilidade da abordagem

Capítulo 5

TRABALHOS RELACIONADOS

O desenvolvimento de aplicações capazes de se adaptar para atender as necessidades de diferentes usuários é um tema quem tem gerado diversas propostas de pesquisa pela comunidade científica. Este capítulo apresenta uma breve descrição de alguns desses trabalhos e os compara com a abordagem proposta neste trabalho.

5.1 *Usage Patterns for User-Centric Service Composition*

Vliegs (2010) apresenta em seu trabalho alguns domínios em que a composição dinâmica de serviços centrada nas necessidades do usuário pode ser aplicada. A utilização desse tipo de composição tem como intuito ajudar os usuários a utilizarem os serviços disponibilizados no ambiente em que estão imersos para suprir suas necessidades. Os domínios apresentados são: empresarial, automação residencial, entretenimento, *e-commerce* e *e-health*.

Foram utilizados quatro critérios para avaliar cada um dos domínios, descritos a seguir:

- *Tipos de Usuários:* Refere-se aos usuários que irão utilizar a composição dinâmica de serviços para suprir suas necessidades. Esses usuários podem ser classificados como leigos, conhecedores do domínio, conhecedores técnicos e avançados. O usuário leigo não tem conhecimento sobre o domínio em que está imerso e também não tem conhecimento na realização de uma composição dinâmica de serviços. Em contrapartida, o usuário avançado tem ambos conhecimentos;

- *Tipos de Serviços:* Refere-se aos tipos de serviços que serão disponibilizados no domínio. Por exemplo, no domínio do turismo poderíamos ter serviços capazes de fornecer informações sobre eventos, restaurantes e monumentos históricos, a partir de uma localização fornecida;

- *Interação Humano-Computador:* Refere-se a forma como será realizada a interação do usuário com a aplicação capaz de realizar a composição dinâmica de serviços. Informações sobre a interface de interação com o usuário e dispositivos capazes de executar a aplicação são relevantes nesse critério; e

- *Condições de Composição:* As condições do usuário podem afetar na maneira como o usuário irá criar uma composição dinâmica de serviços. Essas condições podem estar relacionadas com o tempo e informação disponível para a criação de uma composição e também na forma em como será persistida a composição após ser criada.

Após realizada essas avaliações, dois domínios foram selecionados para desenvolver um protótipo. O desenvolvimento desse protótipo tem como objetivo validar a utilização da composição dinâmica de serviços nos domínios selecionados.

Os domínios do entretenimento e *e-commerce* foram escolhidos, pelo fato de possuírem usuários com pouco conhecimento técnico e do domínio em que estão imersos. Essas características são importantes para o desenvolvimento de um protótipo que possa ser utilizado por usuários leigos.

Nesses domínios foram definidos quais serviços e composições estariam disponíveis, e com base nessas definições onze cenários foram estabelecidos para serem suportados pelo protótipo.

A arquitetura do protótipo apresentada na Figura 5.1 foi projetada para apoiar os cenários estabelecidos. O protótipo é composto dos seguintes módulos:

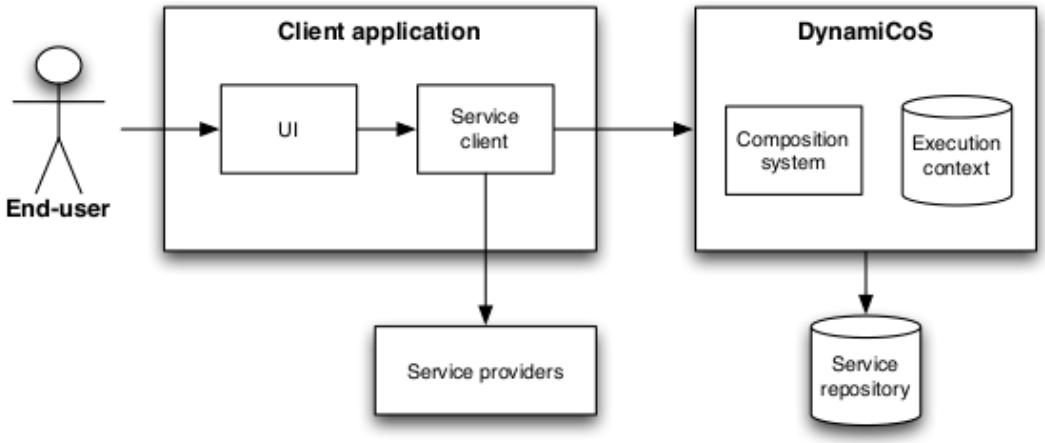

Figura 5.1: Arquitetura do protótipo (VLIEGS, 2010)

.

- *Client Application:* Este módulo tem como finalidade interagir com o usuário e comunicar-se com outros módulos. O módulo é composto pelos seguintes componentes: *User In-*

terface (UI) e *service client*. A UI terá como finalidade a interação do usuário com a aplicação, para que o usuário seja capaz de conduzir a composição de serviços. *Service client* será responsável pela comunicação com o *framework DynamiCoS* e fornecedores dos serviços retornados por esse framework; e

- *Framework DynamiCoS:* Tem como responsabilidade realizar a composição dinâmica de serviços centrada nas necessidades do usuário. Para que a comunicação com esse framework seja realizada de maneira interoperável, o framework é exposto como um serviço web.

O módulo *Client Application* foi desenvolvido utilizando o *framework Ruby on Rails*[1], o qual auxilia o desenvolvimento de aplicações para a plataforma Web. A interação com os outros módulos foi realizada através de protocolos padrões, como HTTP e SOAP.

Para avaliar o protótipo, um grupo de pessoas com diferentes níveis de conhecimento foi selecionado. Esse grupo realizou diversas tarefas interagindo com a interface (UI) e essas tarefas foram definidas de modo a cobrir todas as funcionalidades do protótipo desenvolvido.

Os resultados mostraram que o protótipo necessita de algumas melhorias, porém provou que a utilização da composição dinâmica de serviços quando centrada nas necessidades do usuário produz aplicações capazes de atender as diferentes necessidades dos usuários. Também foi observado que o *framework DynamiCoS* dá o suporte necessário para o desenvolvimento desse tipo de aplicação, oferecendo uma solução flexível para a composição dinâmica de serviços.

O trabalho proposto nessa qualificação pretende apresentar alguns pontos que não foram abordados no trabalho de Vliegs (2010). Esses pontos são: quais atividades e ferramentas são necessárias para o desenvolvimento deste tipo de aplicação; e como desenvolver artefatos reutilizáveis, de modo a viabilizar o desenvolvimento desta aplicação em diferentes plataformas com redução de esforços durante o desenvolvimento.

5.2 *Public service improvement using runtime service composition strategies*

Santos (2010) apresenta em seu trabalho uma abordagem para melhorar a forma de disponibilizar e utilizar serviços públicos, sendo esses serviços prestados pelo governo para seus usuários[2]. Um dos principais desafios encontrados está na forma como uma instituição governa-

[1] http://rubyonrails.org/ acessado em 11/05/2016

[2] O termo "usuário" refere-se ao cidadão que estará utilizando os serviços públicos prestados por uma Instituição Governamental.

mental é fragmentada. Essa fragmentação cria obstáculos para que os serviços públicos possam ser encontrados, e torna algumas vezes inviável a cooperação desses serviços para alcançar um objetivo maior.

A abordagem proposta sugere a interação dos portais de governo eletrônico (e-gov) com os usuários. Essa interação visa entender qual a necessidade de um determinado usuário ao utilizar o portal. Essa percepção irá auxiliar o portal na localização de serviços públicos relevantes que satisfaçam as necessidades dos usuários. Os serviços mais relevantes para os usuários referem-se a eventos que ocorrem em sua vida, como por exemplo, nascimento, mudança de endereço e casamento. O portal ao apresentar apenas serviços relevantes, estará protegendo seus usuários da complexidade existente na estrutura governamental.

Outro ponto importante está na forma como o portal e-gov deve interagir com o cidadão. Para que a interação ocorra de maneira simples, o portal deve fornecer um mecanismo de pesquisa capaz de capturar as intenções do usuário expressadas em linguagem natural e um único ponto de acesso a todos os serviços públicos, protegendo assim o usuário da complexa estrutura existente no portal.

A arquitetura apresentada na Figura 5.2 foi projetada para apoiar o desenvolvimento desses tipos de portais governamentais. A arquitetura é composta dos seguintes módulos:

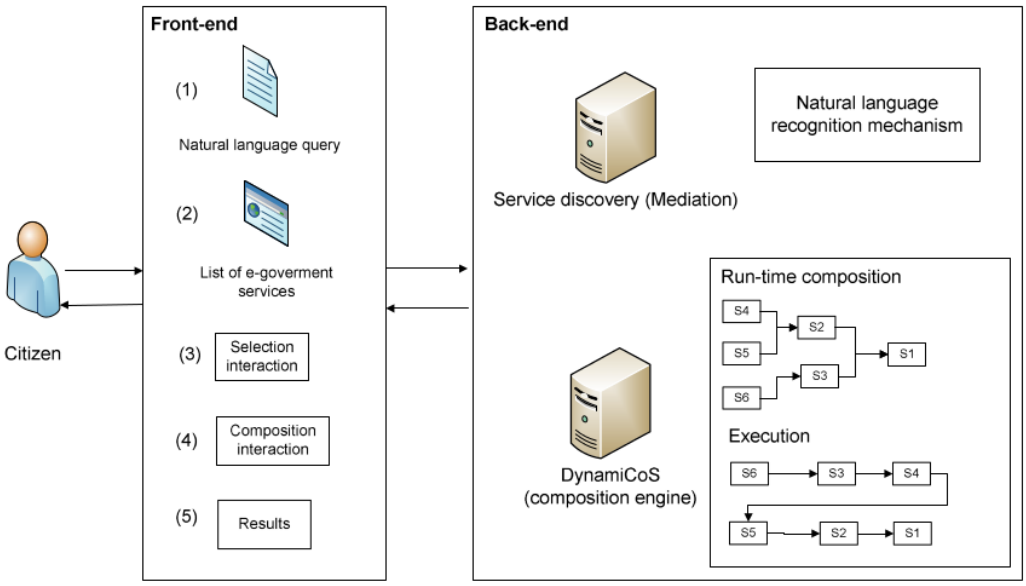

Figura 5.2: Arquitetura do protótipo (SANTOS, 2010)

.

- *Front-end:* Esse módulo possui uma página Web que será responsável por interagir com o usuário. Essa interação é realizada usando linguagem natural, e tem como objetivos

descobrir quais as necessidades do usuário e apresentar o serviço público para o usuário utilizar, a fim de suprir suas necessidades.

- *Back-end:* Esse módulo é responsável por buscar serviços públicos e suas composições. A busca é realizada a partir de informações recebidas de uma requisição do módulo *front-end*. Esse módulo é composto dos seguintes componentes:
  - *Service Discovery:* Esse componente utiliza a solução *Public Service Experience (PSE)* proposta pela iniciativa holandesa *Netherlands Organisation for Applied Scientific Research*. PSE tem como objetivo auxiliar a descoberta dinâmica de serviços públicos. Os serviços encontrados devem ser capazes de suprir as necessidades dos usuários.
  - *DynamiCoS:* Este componente será responsável por entregar serviços e suas composições de maneira centralizada nas necessidades do usuário. A busca desses serviços e composições utilizará o componente *service discovery* como apoio. A entrega é realizada através de interações com o módulo *front-end*.

Para avaliar o protótipo, alguns casos de uso foram definidos e aplicados. Os resultados mostraram que a protótipo superou os desafios encontrados, validando assim a abordagem.

O trabalho apresentado por Santos (2010) também não abordou os seguintes pontos: quais atividades e ferramentas são necessárias para o desenvolvimento deste tipo de aplicação; e como desenvolver artefatos reutilizáveis, de modo a viabilizar o desenvolvimento desta aplicação em diferentes plataformas com redução de esforços durante o desenvolvimento. Sendo assim, o trabalho proposto nessa qualificação irá abordar esses tópicos.

5.3 *Employing Collective Intelligence for User Driven Service Creation*

Jung (2011) apresenta em seu trabalho que o desenvolvimento de aplicação com foco no reuso de serviços disponíveis na Internet é capaz de suprir as necessidades de um usuário final específico. Para que a aplicação seja capaz de suprir essas necessidades, a busca dos serviços que serão reutilizados pela aplicação deve ser fundamentada em informações disponibilizadas pelo usuário final. Porém, a falta de conhecimento do usuário pode levá-lo a fornecer informações com baixa relevância. Devido a isto, a aplicação não será capaz de encontrar serviços relevantes para uma determinada situação.

O objetivo do trabalho é auxiliar o usuário a encontrar serviços relevantes que sejam capazes de atender suas necessidades. Para que isso seja possível o autor sugere a utilização da inteligência coletiva como forma de complementar o conhecimento do usuário. Na Figura 5.3 é apresentado o ambiente proposto pelo trabalho. Esse ambiente interage com os usuários finais e conta com o apoio de dois tipos de inteligência coletiva para encontrar serviços relevantes, descritas a seguir:

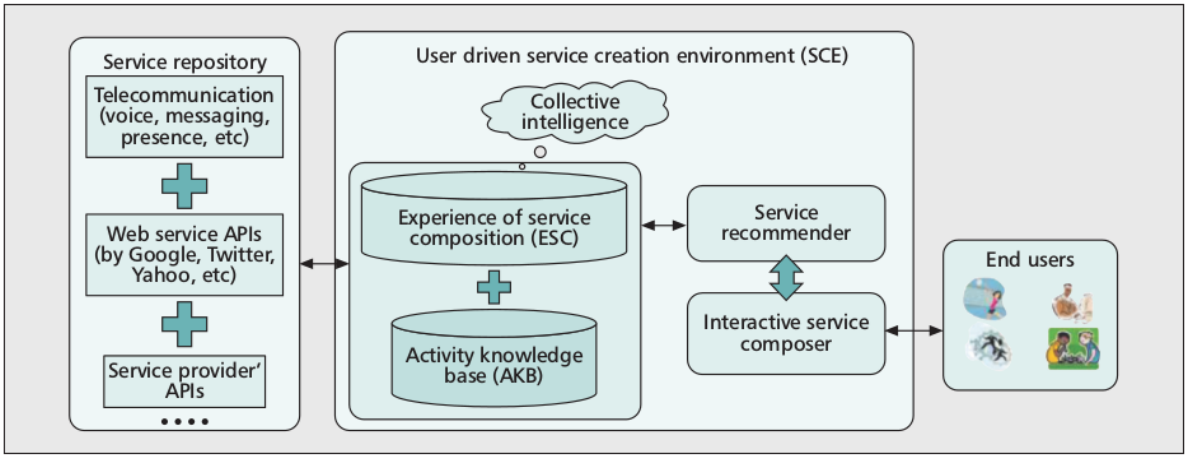

Figura 5.3: Ambiente de criação de serviços dirigidos pelo usuário utilizando inteligência coletiva (JUNG, 2011)

.

- Experiência de composição de serviços: Esse tipo de inteligência coletiva armazena serviços e composições criadas ou modificadas pelo usuário para que sejam reutilizadas futuramente. Basicamente é armazenado o nome dos serviços disponíveis (indisponíveis ou depreciados não são considerados). Para aumentar a relevância dos serviços encontrados é utilizada semântica funcional. Esse tipo de semântica descreve o que o serviço é capaz de fazer e seus objetivos; e

- Conhecimento de atividades a partir da web: Este outro tipo de inteligência coletiva faz extração de informações a partir de recursos da Web 2.0. Estes recursos podem ser documentos e blogs que descrevem atividades humanas e podem ser convertidos para uma semântica funcional baseada em linguagem natural (Programação Neolinguística - PNL).

O complemento do conhecimento do usuário através de experiências anteriores e do conhecimento adquirido a partir da extração de informações da Web irá auxiliar na busca de serviços relevantes que sejam capazes de atender suas necessidades.

O trabalho proposto por Jung (2011) apresentou como a utilização de técnicas de inteligência coletiva podem ajudar na busca de serviços relevantes para suprir as necessidades dos

usuários finais, porém não abordou como aplicá-los no desenvolvimento de software. O foco do presente trabalho é a aplicação dos conceitos, técnicas e ferramentas encontradas na literatura no desenvolvimento de aplicações, com o intuito de produzir aplicações capazes de atender a heterogeneidade de usuários e dispositivos.

5.4 *User-Centric Software Development Process*

Augusto (2014) apresenta em seu trabalho um processo de desenvolvimento de aplicações para ambientes inteligentes utilizando o modelo tradicional em cascata (SOMMERVILLE, 2011; PRESSMAN, 2010). Na Figura 5.4 é ilustrada uma visão geral do processo o qual é composto por atividades do modelo tradicional, bem como atividades específicas para desenvolver esse tipo de aplicação.

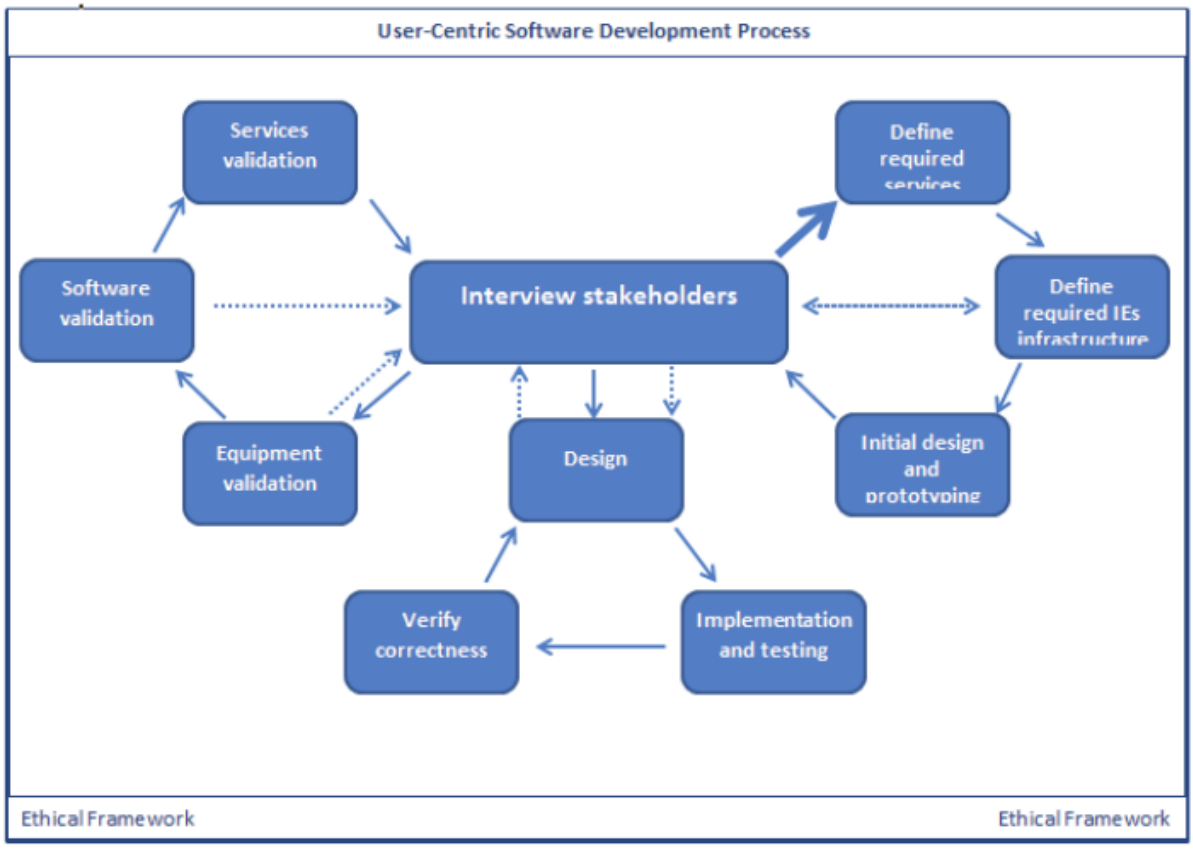

Figura 5.4: Visão Geral do Processo (AUGUSTO, 2014)

.

As setas sólidas indicam que a atividade é obrigatória e as pontilhadas são opcionais. As atividades podem ser realizadas em diferentes sentidos, embora sejam realizadas no sentido horário. As atividades da direita se referem ao escopo inicial do desenvolvimento, as centrais se referem ao desenvolvimento e as da esquerda se referem à instalação e manutenção.

A execução do processo é centrada nos interessados, isto é, a maioria de suas atividades possui interação com esses interessados a fim de verificar se estão de acordo com o que está sendo realizado. Pelo fato dos interessados participarem de forma efetiva no processo de desenvolvimento, as funcionalidades implantadas na aplicação tendem a estar de acordo com o que foi solicitado.

A diferença da abordagem proposta no presente trabalho é a utilização de técnicas, métodos e ferramentas que viabilizam a adaptação da aplicação caso as necessidades dos interessados se modifiquem durante sua execução, sem a necessidade de interação com os desenvolvedores e interessados, o que não ocorre no trabalho apresentado por Augusto (2014).

Capítulo 6

CONCLUSÃO

Neste capítulo é discutida as conclusões da pesquisa desta dissertação de mestrado. Na seção 6.1 são apresentadas as conclusões, contribuições e publicações resultantes desta dissertação; e na seção 6.2 são apresentados possíveis trabalhos futuros.

6.1 Conclusão

Um dos atuais desafios da Engenharia de Software é o desenvolvimento de aplicações capazes de adaptar-se para atender as diferentes necessidades dos usuários. O desenvolvimento desse tipo de aplicação apresenta alguns desafios, como por exemplo: inviabilidade em desenvolver as funcionalidades, pelo fato dessas funcionalidades mudarem conforme as necessidades do usuário. A técnica de Composição Dinâmica de Serviços Dirigida pelo Usuário é uma das solucões que tem se motrado bem-sucedida para deselvolver aplicações capazes de superar esses desafios. Porém a falta de orientação em como desenvolver este tipo de aplicação pode tornar o desenvolvimento uma tarefa complexa, pelo fato de agregar soluções que possam ser desconhecidas pelos desenvolvedores.

Em função disso, este trabalho apresentou uma abordagem para apoiar o desenvolvimento de User-Driven Service Composition Applications. A abordagem definiu quais atividades devem ser realizadas para desenvolver esse tipo de aplicação, bem como os conceitos, técnicas, artefatos, tecnologias e ferramentas necessárias para realizar essas atividades.

Outra contribuição da abordagem refere-se aos artefatos reutilizáveis produzidos pela Engenharia de Domínio. Esses artefatos reduzem esforços no desenvolvimento pelo fato do desenvolvedor não precisar desenvolver todo o código fonte e sim apenas complementá-lo.

Com o intuito de verificar a aplicabilidade da abordagem proposta, foi conduzido um estudo

de caso que compreendeu no desenvolvimento de uma aplicação no domínio de serviços de manutenção predial. O desenvolvimento dessa aplicação foi realizado de duas maneiras: com uso da abordagem e sem uso da abordagem, visando estimar o tempo e esforço possivelmente poupados com a utilização da abordagem proposta.

Em suma, as contribuições desta dissertação de mestrado são:

- Abordagem para orientar o desenvolvimento de UDSCAs, definindo quais atividades devem ser realizadas e quais ferramentas podem apoiar essas atividades; e
- Geração de artefatos reutilizáveis (independente de plataforma), reduzindo esforços no desenvolvimento.

Além das contribuições apresentadas acima, o presente trabalhou resultou na seguinte publicação:

- Alex Roberto Guido, Antonio Francisco do Prado, Wanderley Lopes de Souza, Eduardo Gonçalves da Silva. *"Supporting the Development of User-Driven Service Composition Applications"*. In: **2016 13th International Conference on Information Technology-New Generations (ITNG)**. Springer, 2016. pp 519-530. Las Vegas, NV. DOI 10.1007/978-3-319-32467-8_46.[1](*GUIDO*, 2016).

6.2 Trabalhos Futuros

Mesmo explorando diversas questões durante o desenvolvimento dessa pesquisa, ainda há diversas possibilidades de melhorias relacionadas a este trabalho. Dentre os trabalhos futuros que podem contribuir para a evolução dessa pesquisa, destaca-se o desenvolvimento de ferramentas para automação de algumas atividades da abordagem, como por exemplo:

- *Projetar Fluxo de Comandos*: para essa atividade é possível desenvolver uma ferramenta que ao projetar o fluxo de comandos seja realizada anotações que irão viabilizar a geração automática de código na atividade posterior; e
- *Implementar Fluxo de Comandos*: para essa atividade é possível desenvolver uma ferramenta que irá gerar de forma automática o código do fluxo de comandos a partir das anotações realizadas no diagrama de atividades fornecido como entrada da atividade.

[1] Disponível em: `http://link.springer.com/chapter/10.1007\%2F978-3-319-32467-8_46`

Além disso, a abordagem pode ser avaliada em outros domínios, e para diferentes linguagens e plataformas possibilitando comparar os resultados e verificar sua efetividade.

REFERÊNCIAS

ALI, N. B.; PETERSEN, K.; WOHLIN, C. A systematic literature review on the industrial use of software process simulation. *Journal of Systems and Software*, v. 97, n. 0, p. 65 – 85, 2014. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121214001502>.

ALMEIDA, J. P. et al. Service creation in the spice service platform. In: *Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17)*. 1-7.: [s.n.], 2006. Citation: Almeida, J.P., Baravaglio, A., Belaunde, M., Falcarin, P., Kovacs, E., (2006) ?Service Creation in the SPICE Service Platform?, in Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17). Heidelberg: Wireless World Research Forum, pp.1-7.. Disponível em: <http://roar.uel.ac.uk/1041/>.

ALMEIDA ALMEIDA, P. et al. Service creation in the spice service platform. EURESCOM, 2006.

AUGUSTO, J. User-centric software development process. In: *Intelligent Environments (IE), 2014 International Conference on*. [S.l.: s.n.], 2014. p. 252–255.

BELQASMI, F.; GLITHO, R.; FU, C. Restful web services for service provisioning in next-generation networks: a survey. *Communications Magazine, IEEE*, v. 49, n. 12, p. 66–73, December 2011. ISSN 0163-6804.

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. Uniform resource identifiers (uri): Generic syntax. RFC Editor, United States, 1998.

BERTOLINO, A. et al. Software engineering for internet computing: Internetware and beyond [guest editors' introduction]. *Software, IEEE*, v. 32, n. 1, p. 35–37, Jan 2015. ISSN 0740-7459.

BRHEL, M. et al. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, v. 61, p. 163–181, 2015. Cited By 0. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84924737763partnerID=40md5=ee6a75c59b38a884ab7017eea6d50a43>.

CHEN, N.; CLARKE, S. A dynamic service composition model for adaptive systems in mobile computing environments. In: FRANCH, X. et al. (Ed.). *Service-Oriented Computing*. Springer Berlin Heidelberg, 2014, (Lecture Notes in Computer Science, v. 8831). p. 93–107. ISBN 978-3-662-45390-2. Disponível em: <http://dx.doi.org/10.1007/978-3-662-45391-9$_7$>.

CIRILO, C. et al. Model driven richubi - a model-driven process to construct rich interfaces for context-sensitive ubiquitous applications. In: *Software Engineering (SBES), 2010 Brazilian Symposium on*. [S.l.: s.n.], 2010. p. 100–109.

ERL, T. *Soa Principios De Design De Serviços*. PRENTICE HALL BRASIL, 2009. ISBN 9788576051893. Disponível em: <http://books.google.com.br/books?id=UurZPgAACAAJ>.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado), 2000.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, ACM, New York, NY, USA, v. 2, n. 2, p. 115–150, maio 2002. ISSN 1533-5399. Disponível em: <http://doi.acm.org/10.1145/514183.514185>.

GARTNER. *Highlights Key Predictions for IT Organisations and Users in 2010 and Beyond.* [S.l.], 2010.

GUIDO, A. R. et al. Supporting the development of user-driven service composition applications. In: ____. *Information Technology: New Generations: 13th International Conference on Information Technology*. Cham: Springer International Publishing, 2016. p. 519–530. ISBN 978-3-319-32467-8. Disponível em: <http://dx.doi.org/10.1007/978-3-319-32467-8$_4$6>.

HAN, B. et al. Towards runtime adaptation in context-oriented programming. In: *Electrical Engineering, Computing Science and Automatic Control (CCE), 2013 10th International Conference on*. [S.l.: s.n.], 2013. p. 201–208.

HIRSCHFELD, R.; COSTANZA, P.; NIERSTRASZ, O. Context-oriented programming. *Journal of Object Technology*, v. 7, n. 3, p. 125–151, 2008. Cited By 183. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-43049088796partnerID=40md5=ebe7d3b01fe865d744b110473ee54f7b>.

JUNG, Y. et al. Employing collective intelligence for user driven service creation. *Communications Magazine, IEEE*, v. 49, n. 1, p. 76–83, January 2011. ISSN 0163-6804.

KOTLER, P.; BLOOM, P. N. *Marketing de Serviços Profissionais*. [S.l.]: Manole Interesse Ger, 2002.

LISKIN, O.; SINGER, L.; SCHNEIDER, K. Teaching old services new tricks: Adding hateoas support as an afterthought. In: . [s.n.], 2011. p. 3–10. Cited By 0. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-79956030657partnerID=40md5=53363424718c0a2d3de238fd3a682eb2>.

MICHAEL K. SMITH DEBORAH MCGUINESS, R. V.; WELTY, C. *Web Ontology Language (OWL) guide*. [S.l.], 2002. Disponível em: <http://www.w3.org/TR/2002/WD-owl-guide-20021104/>.

OREIZY, P.; MEDVIDOVIC, N.; TAYLOR, R. N. Runtime software adaptation: Framework, approaches, and styles. In: *Companion of the 30th International Conference on Software Engineering*. New York, NY, USA: ACM, 2008. (ICSE Companion '08), p. 899–910. ISBN 978-1-60558-079-1. Disponível em: <http://doi.acm.org/10.1145/1370175.1370181>.

PAPAZOGLOU, M. Service-oriented computing: concepts, characteristics and directions. In: *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*. [S.l.: s.n.], 2003. p. 3–12.

PAPAZOGLOU, M.; HEUVEL, W.-J. van den. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, Springer-Verlag, v. 16, n. 3, p. 389–415, 2007. ISSN 1066-8888. Disponível em: <http://dx.doi.org/10.1007/s00778-007-0044-3>.

PAPAZOGLOU, M. P. et al. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 40, n. 11, p. 38–45, nov. 2007. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2007.400>.

PAPAZOGLOU, M. P. et al. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, v. 17, n. 02, p. 223–255, 2008. Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/S0218843008001816>.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. Restful Web Services vs. "Big"' Web Services: Making the Right Architectural Decision. In: *Proceedings of the 17th International Conference on World Wide Web*. New York, NY, USA: ACM, 2008. (WWW '08), p. 805–814. ISBN 978-1-60558-085-2. Disponível em: <http://doi.acm.org/10.1145/1367497.1367606>.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2010. (McGraw-Hill higher education). ISBN 9780073375977. Disponível em: <http://books.google.com.br/books?id=y4k_AQAAIAAJ>.

RICHARDSON, L.; RUBY, S. *Restful Web Services*. First. [S.l.]: O'Reilly, 2007. ISBN 9780596529260.

ROSS, D. Structured Analysis (SA): A Language for Communicating Ideas. *Software Engineering, IEEE Transactions on*, SE-3, n. 1, p. 16–34, Jan 1977. ISSN 0098-5589.

SALVANESCHI, G.; GHEZZI, C.; PRADELLA, M. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software*, v. 85, n. 8, p. 1801 – 1817, 2012. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S016412121200074X>.

SANTOS, J. Hoppen dos. *Public service improvement using runtime service composition strategies*. 2010. Disponível em: <http://essay.utwente.nl/59920/>.

SHENG, Q. Z. et al. Web services composition: A decade's overview. *Information Sciences*, v. 280, n. 0, p. 218–238, 2014. ISSN 0020-0255. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025514005428>.

SHI, X. Sharing service semantics using soap-based and rest web services. *IT Professional*, v. 8, n. 2, p. 18–24, March 2006. ISSN 1520-9202.

SILVA, E. da; PIRES, L.; SINDEREN, M. van. Towards runtime discovery, selection and composition of semantic services. *Computer communications*, Elsevier, Amsterdam, v. 34, n. 2, p. 159–168, February 2011. ISSN 0140-3664.

SILVA, E. da; PIRES, L.; SINDEREN, M. van. A-DynamiCoS: A Flexible Framework for User-centric Service Composition. In: *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*. [S.l.: s.n.], 2012. p. 81–92. ISSN 1541-7719.

SOMMERVILLE, I. *Software Engineering*. Pearson, 2011. (International Computer Science Series). ISBN 9780137053469. Disponível em: <http://books.google.com.br/books?id=l0egcQAACAAJ>.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. A process for the design of context-sensitive systems. In: *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on*. [S.l.: s.n.], 2009. p. 143–148.

VLIEGS, E. *Usage patterns for user-centric service composition*. November 2010. Disponível em: <http://essay.utwente.nl/59753/>.

WEISER, M. Ubiquitous computing. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 26, n. 10, p. 71–72, 1993. ISSN 0018-9162.

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

CRUD – Create, Read, Update and Delete

ES – Engenharia de Software

HATEOAS – Hypermedia as the Engine of Application State

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment

IEEE – Institute of Eletrical and Electronics Engineers

JSON – JavaScript Object Notation

OWL – Web Ontology Language

PDF – Portable Document Format

PNL – Programação Neurolinguística

PSE – Public Service Experience

REST – REpresentational State Transfer

SADT – Structured Analysis and Design Technique

SBC – Sociedade Brasileira de Computação

SOAP – Simple Object Access Protocol

SOA – Service-Oriented Architecture

SOC – Service-Oriented Computing

UDDI – Universal Description, Discovery and Integration

UDSCA – User-Driven Service Composition Application

UI – User Interface

UML – Unified Modeling Language

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

W3C – World Wide Web Consortium

WSDL – Web Services Description Language

WS – Web Service

XML – eXtensible Markup Language

Anexo A

Classe CommandFlowController

```
1  package dynamicos.commandflow;
2
3  import static org.springframework.hateoas.mvc.ControllerLinkBuilder.linkTo;
4  import static org.springframework.hateoas.mvc.ControllerLinkBuilder.methodOn;
5
6  import java.io.ByteArrayInputStream;
7  import java.io.IOException;
8  import java.io.InputStream;
9
10 import javax.xml.bind.JAXB;
11
12 import org.springframework.http.HttpEntity;
13 import org.springframework.http.HttpStatus;
14 import org.springframework.http.ResponseEntity;
15 import org.springframework.oxm.jaxb.Jaxb2Marshaller;
16 import org.springframework.web.bind.annotation.RequestMapping;
17 import org.springframework.web.bind.annotation.ResponseBody;
18 import org.springframework.web.bind.annotation.RestController;
19
20 import dynamicos.interactions.servtypediscovery.ServTypeDiscoveryResponse;
21 import dynamicos.utils.EnumPrimitiveCommand;
22 import dynamicos.wsdl.CoordinateResponse;
23
24 @RestController
25 public class CommandFlowController{
26
27     @RequestMapping(value = "/serv_type_discovery.json", produces =
        "application/json")
28     @ResponseBody
29     public HttpEntity<ServTypeDiscoveryResponse> servTypeDiscoveryAsJson()
        throws IOException{
30
31       DynamicosConfiguration config = new DynamicosConfiguration();
32       Jaxb2Marshaller marshaller = config.marshaller();
33       SoapConsumer consumer = config.soapConsumer(marshaller);
34
```

```
35        CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.SERV_TYPE_DISCOVERY);
36
37        InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
38        ServTypeDiscoveryResponse servTypeDiscovery = JAXB.unmarshal(inputStream,
              ServTypeDiscoveryResponse.class);
39
40        servTypeDiscovery.add(linkTo(methodOn(CommandFlowController.class)
41        .servTypeDiscoveryAsJson()).withSelfRel());
42
43        return new ResponseEntity<ServTypeDiscoveryResponse>(servTypeDiscovery,
              HttpStatus.OK);
44
45      }
46
47      @RequestMapping(value = "/serv_type_discovery.xml", produces =
            "application/xml")
48      public ServTypeDiscoveryResponse servTypeDiscoveryAsXml() throws
            IOException{
49
50        DynamicosConfiguration config = new DynamicosConfiguration();
51        Jaxb2Marshaller marshaller = config.marshaller();
52        SoapConsumer consumer = config.soapConsumer(marshaller);
53
54        CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.SELE_SERV);
55
56        InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
57        ServTypeDiscoveryResponse servTypeDiscovery = JAXB.unmarshal(inputStream,
              ServTypeDiscoveryResponse.class);
58
59        return servTypeDiscovery;
60
61      }
62
63    @RequestMapping(value = "/sele_serv.json", produces = "application/json")
64      @ResponseBody
65      public HttpEntity<SeleServResponse> seleServAsJson() throws IOException{
66
67        DynamicosConfiguration config = new DynamicosConfiguration();
68        Jaxb2Marshaller marshaller = config.marshaller();
69        SoapConsumer consumer = config.soapConsumer(marshaller);
70
71        CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.SELE_SERV);
72
73        InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
74        SeleServResponse seleServ = JAXB.unmarshal(inputStream,
              SeleServResponse.class);
75
76        seleServ.add(linkTo(methodOn(CommandFlowController.class).seleServAsJson())
77        .withSelfRel());
78
79        return new ResponseEntity<SeleServResponse>(seleServ, HttpStatus.OK);
```

```
80
81      }
82
83      @RequestMapping(value = "/sele_serv.xml", produces = "application/xml")
84      public SeleServResponse seleServAsXml() throws IOException{
85
86        DynamicosConfiguration config = new DynamicosConfiguration();
87        Jaxb2Marshaller marshaller = config.marshaller();
88        SoapConsumer consumer = config.soapConsumer(marshaller);
89
90        CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.SELE_SERV);
91
92        InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
93        SeleServResponse seleServ = JAXB.unmarshal(inputStream,
              SeleServResponse.class);
94
95        return seleServ;
96
97      }
98
99    @RequestMapping(value = "/validate_inputs.json", produces =
          "application/json")
100     @ResponseBody
101     public HttpEntity<ValidateInputs> validateInputsAsJson() throws IOException{
102
103       DynamicosConfiguration config = new DynamicosConfiguration();
104       Jaxb2Marshaller marshaller = config.marshaller();
105       SoapConsumer consumer = config.soapConsumer(marshaller);
106
107       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.VALIDATE_INPUTS);
108
109       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
110       ValidateInputs validateInputs = JAXB.unmarshal(inputStream,
              ValidateInputs.class);
111
112       validateInputs.add(linkTo(methodOn(CommandFlowController.class)
113       .validateInputsAsJson()).withSelfRel());
114
115       return new ResponseEntity<ValidateInputs>(validateInputs, HttpStatus.OK);
116
117     }
118
119     @RequestMapping(value = "/validate_inputs.xml", produces =
            "application/xml")
120     public ValidateInputs validateInputsAsXml() throws IOException{
121
122       DynamicosConfiguration config = new DynamicosConfiguration();
123       Jaxb2Marshaller marshaller = config.marshaller();
124       SoapConsumer consumer = config.soapConsumer(marshaller);
125
126       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.VALIDATE_INPUTS);
127
```

```
128         InputStream inputStream = new
                ByteArrayInputStream(response.getReturn().getValue().getBytes());
129         ValidateInputs validateInputs = JAXB.unmarshal(inputStream,
                ValidateInputs.class);
130
131         return validateInputs;
132
133       }
134
135     @RequestMapping(value = "/resolve_serv.json", produces = "application/json")
136       @ResponseBody
137       public HttpEntity<ResolveServ> resolveServAsJson() throws IOException{
138
139         DynamicosConfiguration config = new DynamicosConfiguration();
140         Jaxb2Marshaller marshaller = config.marshaller();
141         SoapConsumer consumer = config.soapConsumer(marshaller);
142
143         CoordinateResponse response =
                consumer.getResponseDynamicos(EnumPrimitiveCommand.RESOLVE_SERV);
144
145         InputStream inputStream = new
                ByteArrayInputStream(response.getReturn().getValue().getBytes());
146         ResolveServ resolveServ = JAXB.unmarshal(inputStream, ResolveServ.class);
147
148         resolveServ.add(linkTo(methodOn(CommandFlowController.class).resolveServAsJson())
149         .withSelfRel());
150
151         return new ResponseEntity<ResolveServ>(resolveServ, HttpStatus.OK);
152
153       }
154
155       @RequestMapping(value = "/resolve_serv.xml", produces = "application/xml")
156       public ResolveServ resolveServAsXml() throws IOException{
157
158         DynamicosConfiguration config = new DynamicosConfiguration();
159         Jaxb2Marshaller marshaller = config.marshaller();
160         SoapConsumer consumer = config.soapConsumer(marshaller);
161
162         CoordinateResponse response =
                consumer.getResponseDynamicos(EnumPrimitiveCommand.RESOLVE_SERV);
163
164         InputStream inputStream = new
                ByteArrayInputStream(response.getReturn().getValue().getBytes());
165         ResolveServ resolveServ = JAXB.unmarshal(inputStream, ResolveServ.class);
166
167         return resolveServ;
168
169       }
170
171     @RequestMapping(value = "/exec_serv.json", produces = "application/json")
172       @ResponseBody
173       public HttpEntity<ExecServ> execServAsJson() throws IOException{
174
175         DynamicosConfiguration config = new DynamicosConfiguration();
176         Jaxb2Marshaller marshaller = config.marshaller();
177         SoapConsumer consumer = config.soapConsumer(marshaller);
178
```

```
179       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.EXEC_SERV);
180
181       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
182       ExecServ execServ = JAXB.unmarshal(inputStream, ExecServ.class);
183
184       execServ.add(linkTo(methodOn(CommandFlowController.class).execServAsJson())
185       .withSelfRel());
186
187       return new ResponseEntity<ResolveServ>(execServ, HttpStatus.OK);
188
189     }
190
191     @RequestMapping(value = "/exec_serv.xml", produces = "application/xml")
192     public ExecServ execServAsXml() throws IOException{
193
194       DynamicosConfiguration config = new DynamicosConfiguration();
195       Jaxb2Marshaller marshaller = config.marshaller();
196       SoapConsumer consumer = config.soapConsumer(marshaller);
197
198       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.EXEC_SERV);
199
200       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
201       ExecServ execServ = JAXB.unmarshal(inputStream, ExecServ.class);
202
203       return execServ;
204
205     }
206
207   @RequestMapping(value = "/add_to_context.json", produces = "application/json")
208     @ResponseBody
209     public HttpEntity<AddToContext> addToContextAsJson() throws IOException{
210
211       DynamicosConfiguration config = new DynamicosConfiguration();
212       Jaxb2Marshaller marshaller = config.marshaller();
213       SoapConsumer consumer = config.soapConsumer(marshaller);
214
215       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.EXEC_SERV);
216
217       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
218       AddToContext addToContext = JAXB.unmarshal(inputStream,
              AddToContext.class);
219
220       addToContext.add(linkTo(methodOn(CommandFlowController.class).addToContextAsJson())
221       .withSelfRel());
222
223       return new ResponseEntity<AddToContext>(addToContext, HttpStatus.OK);
224
225     }
226
227     @RequestMapping(value = "/add_to_context.xml", produces = "application/xml")
228     public AddToContext addToContextAsXml() throws IOException{
```

```
229
230       DynamicosConfiguration config = new DynamicosConfiguration();
231       Jaxb2Marshaller marshaller = config.marshaller();
232       SoapConsumer consumer = config.soapConsumer(marshaller);
233
234       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.EXEC_SERV);
235
236       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
237       AddToContext addToContext = JAXB.unmarshal(inputStream,
              AddToContext.class);
238
239       return addToContext;
240
241     }
242
243   @RequestMapping(value = "/clean_context.json", produces = "application/json")
244     @ResponseBody
245     public HttpEntity<CleanContext> cleanContextAsJson() throws IOException{
246
247       DynamicosConfiguration config = new DynamicosConfiguration();
248       Jaxb2Marshaller marshaller = config.marshaller();
249       SoapConsumer consumer = config.soapConsumer(marshaller);
250
251       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.EXEC_SERV);
252
253       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
254       CleanContext cleanContext = JAXB.unmarshal(inputStream,
              CleanContext.class);
255
256       cleanContext.add(linkTo(methodOn(CommandFlowController.class).cleanContextAsJson())
257       .withSelfRel());
258
259       return new ResponseEntity<CleanContext>(cleanContext, HttpStatus.OK);
260
261     }
262
263     @RequestMapping(value = "/clean_context.xml", produces = "application/xml")
264     public CleanContext cleanContextAsXml() throws IOException{
265
266       DynamicosConfiguration config = new DynamicosConfiguration();
267       Jaxb2Marshaller marshaller = config.marshaller();
268       SoapConsumer consumer = config.soapConsumer(marshaller);
269
270       CoordinateResponse response =
              consumer.getResponseDynamicos(EnumPrimitiveCommand.EXEC_SERV);
271
272       InputStream inputStream = new
              ByteArrayInputStream(response.getReturn().getValue().getBytes());
273       CleanContext cleanContext = JAXB.unmarshal(inputStream,
              CleanContext.class);
274
275       return cleanContext;
276
```

```
277     }
278 }
```

Código A.1: Classe CommandFlowController em Java

Anexo B

Classe DynamicosConfiguration

```java
package dynamicos.commandflow;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

@Configuration
public class DynamicosConfiguration {

  @Bean
  public Jaxb2Marshaller marshaller() {
    Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
    marshaller.setContextPath("dynamicos.wsdl");
    return marshaller;
  }

  @Bean
  public SoapConsumer soapConsumer(Jaxb2Marshaller marshaller) {
    SoapConsumer consumer = new SoapConsumer();
    consumer.setDefaultUri("http://200.18.98.114:8080/axis2/services/Coordinator/");
    consumer.setMarshaller(marshaller);
    consumer.setUnmarshaller(marshaller);
    return consumer;
  }

}
```

Código B.1: Classe DynamicosConfiguration em Java

Anexo C

Classe SoapConsumer

```
1  package dynamicos.commandflow;
2
3  import java.io.IOException;
4
5  import javax.xml.bind.JAXBElement;
6
7  import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
8  import org.springframework.ws.soap.client.core.SoapActionCallback;
9
10 import dynamicos.utils.DynamicosUtils;
11 import dynamicos.utils.EnumPrimitiveCommand;
12 import dynamicos.wsdl.Coordinate;
13 import dynamicos.wsdl.CoordinateResponse;
14 import dynamicos.wsdl.ObjectFactory;
15
16 public class SoapConsumer extends WebServiceGatewaySupport{
17
18   public CoordinateResponse getResponseDynamicos(EnumPrimitiveCommand
       primitiveCommand) throws IOException {
19
20     ObjectFactory objectFactory = new ObjectFactory();
21
22     DynamicosUtils dynamicosUtils = new DynamicosUtils();
23
24     JAXBElement<String> interactionType = null;
25     JAXBElement<String> interactionArgs = null;
26
27
28     switch (primitiveCommand) {
29     case SERV_TYPE_DISCOVERY:
30       String xml = dynamicosUtils.readFile(primitiveCommand);
31       interactionType =
             objectFactory.createCoordinateInteractionType("SERV_TYPE_DISCOVERY");
32       interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
33       break;
34
35     case SELE_SERV:
```

```
36        String xml = dynamicosUtils.readFile(primitiveCommand);
37        interactionType =
              objectFactory.createCoordinateInteractionType("SELE_SERV");
38        interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
39        break
40
41      case VALIDATE_INPUTS:
42        String xml = dynamicosUtils.readFile(primitiveCommand);
43        interactionType =
              objectFactory.createCoordinateInteractionType("VALIDATE_INPUTS");
44        interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
45        break
46
47      case RESOLVE_SERV:
48        String xml = dynamicosUtils.readFile(primitiveCommand);
49        interactionType =
              objectFactory.createCoordinateInteractionType("RESOLVE_SERV");
50        interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
51        break
52
53      case EXEC_SERV:
54        String xml = dynamicosUtils.readFile(primitiveCommand);
55        interactionType =
              objectFactory.createCoordinateInteractionType("EXEC_SERV");
56        interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
57        break
58
59      case ADD_TO_CONTEXT:
60        String xml = dynamicosUtils.readFile(primitiveCommand);
61        interactionType =
              objectFactory.createCoordinateInteractionType("ADD_TO_CONTEXT");
62        interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
63        break
64
65      case CLEAN_CONTEXT:
66        String xml = dynamicosUtils.readFile(primitiveCommand);
67        interactionType =
              objectFactory.createCoordinateInteractionType("CLEAN_CONTEXT");
68        interactionArgs = objectFactory.createCoordinateInteractionArgs(xml);
69        break
70
71
72      default:
73        break;
74      }
75
76
77        Coordinate request = new Coordinate();
78      request.setInteractionType(interactionType);
79      request.setInteractionArgs(interactionArgs);
80
81      CoordinateResponse response = (CoordinateResponse) getWebServiceTemplate()
82          .marshalSendAndReceive(
83              "http://200.18.98.67:8080/axis2/services/Coordinator/",
84              request,
85              new SoapActionCallback(""));
86
```

```
87
88      return response;
89    }
90  }
```

Código C.1: Classe SoapConsumer em Java

Anexo D

Classe DynamicosUtils

```
1  package dynamicos.utils;
2
3  import java.io.BufferedReader;
4  import java.io.FileReader;
5  import java.io.IOException;
6
7  public class DynamicosUtils {
8
9    private BufferedReader reader;
10
11   public String readFile(EnumPrimitiveCommand primitiveCommand) throws
         IOException {
12
13     String path = "";
14
15     switch (primitiveCommand) {
16     case SERV_TYPE_DISCOVERY:
17       path = "./XMLOperations/ServTypeDiscovery.xml";
18       break;
19
20     case SELE_SERV:
21       path = "./XMLOperations/SeleServ.xml";
22       break;
23
24     case VALIDATE_INPUTS:
25       path = "./XMLOperations/ValidateInputs.xml";
26       break;
27
28     case RESOLVE_SERV:
29       path = "./XMLOperations/ResolveServ.xml";
30       break;
31
32     case EXEC_SERV:
33       path = "./XMLOperations/ExecServ.xml";
34       break;
35
36     case ADD_TO_CONTEXT:
```

```
37        path = "./XMLOperations/AddToContext.xml";
38        break;
39
40      case CLEAN_CONTEXT:
41        path = "./CleanContext.xml";
42        break;
43
44
45      default:
46        break;
47      }
48         ;
49
50
51      reader = new BufferedReader( new FileReader (path));
52        String line = null;
53        StringBuilder stringBuilder = new StringBuilder();
54        String ls = System.getProperty("line.separator");
55
56        while( ( line = reader.readLine() ) != null ) {
57            stringBuilder.append( line );
58            stringBuilder.append( ls );
59        }
60
61        return stringBuilder.toString();
62    }
63  }
```

Código D.1: Classe DynamicosUtils em Java