Universidade de Trás-os-Montes e Alto Douro

Doutoramento em Informática

UM MODELO PARA A COMPOSIÇÃO DINÂMICA DE SERVIÇOS DEPENDENTES DO CONTEXTO

João Paulo Pereira de Sousa

Tese submetida à Universidade de Trás-os-Montes e Alto Douro para obtenção do grau de Doutor em Informática, elaborada sob a orientação do Professor Doutor Eurico Manuel Elias de Morais Carrapatoso do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto e Professor Doutor Benjamim Ribeiro da Fonseca do Departamento de Engenharias da Escola de Ciências e Tecnologia da Universidade de Trás-os-Montes e Alto Douro.

UTAD, 2010

RESUMO

Nos últimos anos os dispositivos móveis têm cada vez mais um papel importante na vida quotidiana dos utilizadores. Inicialmente apenas como dispositivos de comunicação por voz, evoluíram e começaram a oferecer serviços básicos de acesso a conteúdos Web. Ultimamente, tornaram-se para muitos utilizadores um dos principais meios de acesso aos mais diversos tipos de serviços que a Web oferece, entre eles serviços de comunicação, pesquisa, localização, socialização ou comércio electrónico. É ainda possível observar o interesse de grandes organizações das tecnologias de informação na área dos serviços móveis, que auxiliadas por hardware cada vez mais sofisticado começaram a explorar a utilização de informações contextuais em que o utilizador se encontra, por forma a melhorar a experiência móvel do utilizador.

Embora seja grande a diversidade dos serviços móveis que é possível encontrar actualmente na Web, haverá sempre situações em que um utilizador pretende um serviço mais ou menos complexo com o objectivo de realizar uma determinada tarefa mais específica ou personalizada e possa até englobar mais que um serviço.

Nesta Tese de Doutoramento propõe-se um modelo para a composição dinâmica de serviços em dispositivos móveis sensível à informação contextual do utilizador, suportado por tecnologias Web. Este modelo utiliza um servidor de aplicações que possibilita a execução dos seus diversos componentes e cuja interacção com o cliente é feita através de Web Services. Para a sua validação, desenvolveu-se um protótipo baseado no modelo, que suporta a composição de serviços sensíveis ao contexto do utilizador para ser aplicado num ambiente de um campus universitário, que foi sujeito a um conjunto de testes funcionais.

Palavras-chave:

Composição de Serviços, Sensibilidade ao Contexto, *Semantic Web*, Serviços Semânticos, Modelos Contextuais Semânticos.

ABSTRACT

In recent years mobile devices play an increasingly important role in the daily lives of users. Initially only voice communication devices, they have been developed evolved and began offering basic services for access to Web content. Lately, became for many users one of the principal means of access to a wide range of services that the Web offers, like communication services, research, location, socialization, or e-commerce. It is still possible to see the interest of major IT organizations in the area of mobile services, which aided by increasingly sophisticated hardware began exploring the use of user's contextual information, to improve mobile user experience.

While it is great diversity of mobile services that can be found on the Web today, there will be always situations where a user wants a more or less complex service, in order to solve a particular task more specific or personalized and may even include more than one service.

This PhD thesis proposes a model for dynamic composition of services on mobile devices sensitive to contextual information of the user, supported by Web technologies Therefore, this model uses an application server that enables the implementation of his various components of this model and whose customer interaction is done via Web Services. For validation, we have developed a prototype based on the model, which supports composition of services sensitive to the context of the user, to be applied in an environment of a university campus, which was subject to a set of functional tests.

Keywords:

Service Composition, Context-aware, Semantic Web, Semantic Web Services, Semantic Context Models.

AGRADECIMENTOS

Expresso a minha sincera gratidão aos meus orientadores, Prof. Doutor Eurico Carrapatoso e Prof. Benjamim Fonseca, pela partilha de conhecimento, permanente disponibilidade e pelas valiosas opiniões e sugestões durante a realização deste trabalho.

Agradeço aos meus colegas de departamento de Informática e Comunicações, não só pelo incentivo, mas também por assegurarem durante o regular funcionamento de projectos nos quais estava envolvido antes de iniciar este doutoramento.

Ao IPB pela concessão de dispensa de serviço de docente durante parte dos trabalhos de doutoramento, o que contribuiu significativamente para a realização desta tese.

Por fim, um eterno agradecimento à Raquel, aos meus Pais e à minha restante família a quem devo o maior suporte e incentivo. Assim como o meu reconhecimento pelo infinito apoio, dedicação e paciência que sempre demonstraram durante a realização deste trabalho.

PUBLICAÇÕES

Este trabalho originou ou contribuiu para as seguintes publicações:

Artigos em conferências internacionais:

João P. Sousa, Eurico Carrpatoso, Benjamim Fonseca, "A Service-Oriented Middleware for Composing Context Aware Mobile Services," ICIW, pp.357-362, 2009 Fourth International Conference on Internet and Web Applications and Services, 2009. ISBN: 978-0-7695-3613-2. (best papers).

http://doi.ieeecomputersociety.org/10.1109/ICIW.2009.59

João P. Sousa, Benjamim Fonseca, Eurico Carrapatoso, Hugo Paredes, "An Evolutionary Platform for the Collaborative Contextual Composition of Services," CRIWG, 5th Collaboration Researchers' International Workshop on Groupware, pp 182-189, Vol. 5784 Springer (2009). issn: 0302-9743.

http://www.springerlink.com/index/H11RVH1721764469.pdf

Benjamim Fonseca, Hugo Paredes, João P. Sousa, F. Mário Martins, Eurico Carrapatoso, "SAGA Reloaded: towards a generic platform for developing cooperative applications," CSCWD 2009. Santiago do Chile. 22-24 April 2009. IEEE Computer Society Press, pp 331-337. 978-1-4244-3534-0.

http://www.computer.org/portal/web/csdl/doi/10.1109/CSCWD.2009.4968080

João Paulo Sousa, Eurico Carrapatoso, Benjamim Fonseca, "Uma arquitectura para a composição dinâmica de serviços dependentes do contexto", "Conferência Ibero - Americana WWW/Internet 2007", pp. 286-289, Vila Real, Portugal, 2007. ISBN 978–972–8924–45-4.

http://www.iadis.net/dl/final_uploads/200713C040.pdf

Artigos em jornais científicos:

João P. Sousa, Eurico Carrapatoso, Benjamim Fonseca, Maria Pimentel, Renato Bulcão, "Composition of context aware mobile services using a semantic context model," International Journal on Advances in Software, issn 1942-2628 2(2&3): pp. 275-287.

http://www.iariajournals.org/software/soft_v2_n23_2009_paged.pdf

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

ÍNDICE REMISSIVO

LISTA DE ACRÓNIMOS

ABOX	Assertional Box
AGPL	Affero General Public License
API	Application Programming Interface
CoBrA	Context Broker Architecture
DAML	DARPA Agent Markup Language
DAML-S	DARPA Agent Markup Language for Services
DTD	Document Type Definition
FLOWS	First Order Logic Ontology for Web Services
FTP	File Transfer Protocol
GNU	GNU General Public License
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electric and Electronic Engineers
IETF	Internet Engineering Task Force
IOPE	Input, Output, Preconditions, Effects
JAVA RMI	Java Remote Method Invocation
KB	Knowledge Base
OASIS	Organization for the Advancement of Structured Information Standards
OIL	Ontology Inference Layer ou Ontology Interchange Language
OMG	Object Management Group
ORM	Object-Role Modelling
OWL-S	Semantic Markup for Web Services
PDA	Personal Digital Assistant
PLS	Process Specification Language
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format

ROWS	Rule Ontology for Web Services
RSS	Really Simple Syndication
RuleML	Rule Markup Language
SeCoM	Semantic Context Model
SLA	Service Level Agreement
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
SPARQL	SPARQL Protocol and RDF Query Language
SWRL	Semantic Web Rule Language
SWS	Semantic Web Services
SWSF	Semantic Web Services Framework
SWSI	Web Services Initiative
SWSL	Semantic Web Services Language
SWSLC	Web Services Language Committee
SWSO	Semantic Web Services Ontology
TBOX	Terminological Box
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Names
W3C	World Wide Web Consortium
WS-CDL	Web Services Choreography Description Language
WSBPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
WSDL-S	Web Service Semantics
WSFL	Web Services Flow Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Modeling Execution Environment
WWW	World Wide Web

XLANG	Web Services for Business Process Design
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XMLS	XML Schema
XSD	XML Schema Definition
DIG	DL Implementation Group

Capítulo 1

INTRODUÇÃO

Actualmente, o uso da *Web* como meio de obtenção de informação tomou proporções quase globais. Diversos tipos de serviços como acesso a vídeos, imagens, VoIP eram inicialmente oferecidos ou acedidos através de computadores de secretária, passaram nos últimos anos a estar disponíveis num número crescente de dispositivos móveis cada vez mais evoluídos. A combinação da oferta de serviços e mobilidade, logo cedo mostrou grandes potencialidades, tendo por isso vindo a fazer parte da vida de um número cada vez maior de utilizadores, que os usam para auxiliar nas suas tarefas diárias.

1.1 Motivação

É previsível que num futuro próximo o ambiente de redes móveis venha a ser caracterizado por interacções entre serviços, que poderão aparecer e desaparecer, disponibilizando-se aos utilizadores e dispositivos de uma forma dinâmica e transparente. É também possível observar que são cada vez mais os mecanismos de aquisição de informação contextual que fornecem informações como: a localização, actividade actual, informações meteorológicas, características dos dispositivos e algumas mais avançadas que poderão fornecer até parâmetros biológicos do utilizador. Esta informação contextual tem cada vez mais um papel fundamental na simplificação da interacção entre os humanos e o mundo digital.

Estes novos serviços, que deverão ser fáceis de utilizar e acrescidos dessa informação contextual, poderão tornar-se num motor da evolução dos dispositivos móveis e das redes sem fios. No entanto, na grande maioria das situações o utilizador assume apenas o papel de

consumidor dos serviços disponibilizados por terceiros, como acontece com o exemplo dos grandes operadores de comunicações móveis. A esses utilizadores é-lhes disponibilizado um grande conjunto de serviços e de informações de enorme utilidade, mas também destinados a um mercado muito amplo, ficando de fora todos os utilizadores que pretendam beneficiar de serviços e de informações mais personalizáveis.

Deste modo, um possível cenário para eliminar ou pelo menos atenuar esse problema seria o de, num ambiente de redes móveis, um utilizador receber no seu dispositivo móvel (*smartphone* ou portátil) informação contextualizada (ex. com base no local, na hora, na vizinhança, etc.) e ter à sua disposição um conjunto de serviços úteis, também baseados no seu contexto actual. Ao utilizador ser-lhe-ia ainda possível fazer a composição desses serviços em tempo real para a criação de um novo serviço. Esta composição seria conseguida utilizando várias técnicas, dependendo da integração/distribuição dos componentes, mas com um objectivo final comum a todas: criar um serviço com novas funcionalidades e utilizá-lo/partilhá-lo sempre que necessário.

1.2 Objectivos

De modo a proporcionar aos utilizadores serviços que beneficiem do ambiente em que o utilizador se encontra e que explorem as características das infra-estruturas móveis, é importante fornecer aos programadores de serviços móveis um ambiente que torne a criação destes serviços mais fácil. Esta solução será terá ainda mais potencial se proporcionar aos utilizadores finais o desenvolvimento de soluções simples que possam oferecer novas características mais adequadas às suas pretensões.

Como consequência das motivações apresentadas na secção anterior, o objectivo primordial deste trabalho é o de definir um modelo para o suporte da composição contextual de serviços. Este modelo faz uso do contexto actual do utilizador, como a sua localização, a sua disponibilidade, a sua actividade actual, o seu perfil, a hora, a sua vizinhança, tipo de dispositivo que está utilizar, para auxiliar esse utilizador na composição dinâmica de novos serviços. Deverá possibilitar ao utilizador a selecção de serviços disponíveis, que juntamente com um conjunto de regras simples, permitam fazer a composição de um novo serviço com novas características e mais adequado às suas necessidades. Poderá ainda disponibilizar informações ou serviços contextuais baseados no contexto actual.

A concretização do objectivo principal desta dissertação passa pela realização de várias etapas, necessárias ao enquadramento tecnológico do problema e à escolha das soluções que pareçam ser as mais adequadas para a definição do modelo. A seguir apresentam-se os objectivos subdivididos resultantes do desdobramento do objectivo principal da dissertação:

- aprofundamento do conhecimento sobre "contexto", as suas classes e modelos contextuais;
- estudo de arquitecturas orientadas aos serviços, de esquemas de composição de serviços, e análise das vantagens e desvantagens de cada uma delas;
- identificação dos principais sistemas de composição de serviços e *context-aware*, disponíveis na literatura científica mais relevante ou comercialmente;
- proposta de uma arquitectura inovadora que permita a composição de serviços dinamicamente por parte do utilizador, através da composição de serviços existentes;
- análise das tecnologias que possam servir de suporte à implementação de um sistema de composição de serviços sensível ao contexto baseado no modelo proposto e apreciação das mais adequadas;
- implementação de um protótipo da arquitectura proposta;
- realização de testes que permitam validar qualitativamente a solução proposta e, se possível, de uma forma quantitativa;
- definição de trabalho futuro, mediante os resultados obtidos do ponto anterior e da evolução das tecnologias disponíveis.

Este trabalho terá como cenário de estudo um *campus* universitário, onde está a ser desenvolvido um serviço de localização baseado em informações da rede sem fios já existente.

1.3 Contributos e inovação

O contributo principal desta dissertação é a proposta de um modelo capaz de fornecer um serviço de composição simples e sensível ao contexto do utilizador e que possa ser utilizado em dispositivos móveis. Das actividades de modelação e implementação orientadas pelos

objectivos acima referidos resultaram os seguintes contributos deste projecto:

- a apresentação de um modelo que integra o conceito de modelo contextual semântico para auxiliar o utilizador na construção de serviços (Sousa, Carrapatoso et al., 2009);
- uma arquitectura denominada *iCas*, orientada aos serviços que permite a criação dinâmica de serviços em dispositivos móveis de forma simples e evolutiva com propriedades colaborativas (Sousa, Fonseca et al., 2009);
- um mecanismo de selecção de serviços, que oferece ao utilizador os serviços mais adequados ao seu contexto actual e às suas preferências (Sousa, Carrapatoso et al., 2009);
- um componente que fornece funções para gerir e interpretar informações de contexto apoiadas por um modelo contextual ontológico e semântico;
- uma instanciação da arquitectura *iCas* correspondente aos principais componentes que a compõem.

A inovação deste trabalho de doutoramento consiste na apresentação de um modelo que permite composição dinâmica de serviços auxiliada pelo contexto que o utilizador encontra em dispositivos móveis. Este modelo é ainda capaz de adaptar os resultados dos serviços ou informações sensíveis ao contexto do utilizador ao tipo de terminal que está a ser usado. Este modelo permite aos utilizadores não só consumirem serviços fornecidos por terceiros, mas também puderem criar os seus próprios serviços mais adequados às suas carências.

1.4 Estrutura da tese

Esta Tese de Doutoramento está estruturada em seis capítulos, incluindo esta introdução, no qual foi exposto o contexto deste trabalho e os principais objectivos que encaminharam o trabalho desenvolvido.

No capítulo 2, "Contexto e Arquitecturas de Composição" é feita uma abordagem geral sobre o conceito de contexto. É apresentada uma caracterização das arquitecturas orientadas aos serviços, de modos e esquemas de composição de serviços. São discutidos os principais sistemas *context-aware* e de composição de serviços, nomeadamente sobre as suas principais

características e serviços que oferecem. Por fim, é feita uma comparação entre os diversos sistemas analisados, onde são indicadas as suas vantagens e desvantagens.

No capítulo 3, "*Semantic Web*", são apresentadas as diversas camadas que formam a *Semantic Web*. É revisto o conceito de ontologias e como estas representam conhecimento de domínios. É feita uma análise por camadas da arquitectura da *Semantic Web*, descrevendo cada uma das camadas e suas tecnologias, sendo dado destaque às camadas de descrição estrutural e semântica, descrição semântica e lógica e SPARQL (*SPARQL Protocol and RDF Query Language*). É também apresentado o conceito de raciocínio na *Semantic Web*. Por fim, são apresentadas diversas ferramentas semânticas essenciais ao desenvolvimento de sistemas que façam uso de tecnologias da *Semantic Web*.

No capítulo 4, "Tecnologias de suporte", é feito um levantamento de um conjunto de tecnologias que candidatas a serem utilizadas na concepção e implementação da arquitectura proposta. Para tal são estudadas as tecnologias padrão que compõem a arquitectura de *Web Services*. Em seguida é descrita a linguagem normalizada reconhecida pela *Organization for the Advancement of Structured Information Standards* (OASIS) para a composição de serviços e são discutidas as suas vantagens e limitações no contexto dos objectivos propostos para esta arquitectura. Posteriormente são apresentadas várias soluções para a composição de *Web Services Semânticos*, finalizando-se o capítulo com uma comparação entre elas.

No capítulo 5, "Um modelo para o suporte da composição contextual de serviços", é proposto um modelo para um sistema de composição dinâmica de serviços que deverá explorar o contexto do utilizador para lhe oferecer os serviços mais adequados. Este modelo poderá ainda oferecer informações que possam ser úteis ao utilizador, recorrendo para isso a informações do ambiente em que este se encontra. Será descrito a arquitectura genérica que é proposta neste modelo, detalhando cada um dos seus componentes. Este capítulo termina com uma discussão de possíveis cenários de aplicação do modelo.

No capítulo 6, "Implementação do modelo", é apresentada a implementação de um protótipo desta arquitectura, sendo discutidas as opções de implementação mais relevantes e os motivos para as escolhas efectuadas para a sua concretização.

O capítulo 7, "Análise do desempenho do sistema", é feita uma avaliação de desempenho do protótipo implementado, através de observações qualitativas e medições temporais.

O capítulo 8, "Conclusões e trabalho futuro", recorda-se o problema e faz-se uma análise

global do trabalho realizado e dos resultados obtidos. Apresentam-se alguma conclusões relativas a esse resultados e aos objectivos propostos. Finalmente identificam-se trabalhos futuros, apresentando-se linhas de evolução futura da arquitectura e do protótipo.

Capítulo 2

CONTEXTO E ARQUITECTURAS DE COMPOSIÇÃO

Verifica-se actualmente a existência de uma crescente gama de sistemas computacionais que se tornaram quase indispensáveis no nosso quotidiano, os dispositivos móveis. Inicialmente com poucos recursos e funcionalidades, têm evoluído nos últimos anos de forma considerável, apresentando-se como ferramentas poderosas para a melhoria da qualidade de vida dos seus utilizadores. Contudo, esses dispositivos têm por natureza várias limitações, tais como as dimensões do visor, a interface de entrada de dados, a autonomia, o poder computacional, entre outras. Acreditamos que os serviços poderão superar essas limitações em que o paradigma de o utilizador ter que procurar os serviços que lhe interessam deverá ser substituído pela oferta de serviços baseados nas suas necessidades e adaptados ao seu contexto (ex. espacial, temporal, social, entre outros), e quando esses não existirem, poderem ser criados de forma *ad-hoc*. Este paradigma de computação de serviços orientados para ambientes móveis deverá disponibilizar funcionalidades para que os serviços possam ser descritos, publicados, descobertos e compostos. Eventualmente, esses serviços poderão ter também a capacidade de operar entre eles, mesmo que não tenham sido projectados para tal.

Tal como acontece entre humanos em que informação adicional sobre o contexto em que estes se encontram (ex. local, hora, vizinhança) pode melhorar a sua forma de interagir ou de comunicar, de forma análoga, a sensibilidade ao contexto poderá trazer a este paradigma novas formas de enriquecimento na interacção entre homem-máquina. Esta sensibilidade ao contexto deverá ser possível através de sistemas que sejam capazes de capturar e processar de forma eficaz o contexto em que o utilizador está inserido.

Neste capítulo é feita uma revisão geral de sistemas orientados aos serviços e sensíveis ao

contexto. Após um breve enquadramento, introduz-se a definição de contexto e a sua importância nos sistemas computacionais. Com o intuito de auxiliar a escolha do contexto a usar no desenvolvimento de arquitecturas baseadas em contexto, são apresentadas diferentes classes de contexto. As secções seguintes abordam a classificação das aplicações de contexto, definição de *context-awareness* e quais as propriedades que uma aplicação *context-aware* deve possuir. Devido à natureza do modelo orientados aos serviços proposto neste trabalho, são estudadas metodologias de composição de serviços com principal destaque para a tecnologia *Web Services*. Por último, são referidos alguns exemplos de implementação de arquitecturas que se assemelham ou que possuam partes em comum como a arquitectura proposta neste trabalho.

2.1 O que é o Contexto

O desenvolvimento de arquitecturas que utilizem informações de contexto requer a percepção do que é o significado de contexto e de como pode ser usado nestas arquitecturas. Um dos fenómenos que se observam quando se questiona alguém sobre de que se trata o contexto é que implicitamente a maior parte das pessoas sabem do que se trata, no entanto, sentem dificuldades para o explicar (Abowd, Dey et al., 1999). Por isso, muitas vezes as definições de contexto são feitas através de citações de exemplos ou escolhendo sinónimos para contexto.

Quem introduziu pela primeira vez o termo de *context aware* foram (Schilit, Adams et al., 1994), referindo-se ao contexto como a localização, identidade de pessoas e objectos na vizinhança e alterações a esses objectos.

Em (Brown, 1996), o autor define contexto como localização, identidade das pessoas na vizinhança do utilizador, a hora do dia, temperatura entre outros. Em (Pascoe, 1998), define-se contexto como sendo a localização do utilizador, ambiente, identidade e tempo.

Em (Abowd, Dey et al., 1999) os autores justificam porque é que a definição de contexto através de exemplos e sinónimos não é a melhor abordagem. Segundo eles, corre-se o risco de as definições serem demasiado específicas ou então não se saber ao certo se determinadas informações não consideradas na definição são ou não contexto. Deste modo, os autores definem contexto como: “Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, lugar, ou objecto que é considerado relevante para a interacção entre um utilizador e uma aplicação, incluindo o

utilizador e a própria aplicação." Se uma porção de informação pode ser usada para caracterizar a situação de um participante numa interacção, então essas informação é contexto. Considere-se o seguinte cenário para exemplificar a definição: a utilização de um serviço de guia móvel num museu por um utilizador. É óbvio que a entidade neste caso é o utilizador e a aplicação o serviço de guia. Supondo que se têm disponíveis duas peças de informação – tempo e presença de outro utilizador, então através da utilização da definição irá tentar-se determinar se alguma delas é considerada contexto. Pode-se dizer que o estado do tempo não afecta o serviço já que este é usado dentro do museu, por isso não é considerado contexto. A presença de outra pessoa na vizinhança pode ser utilizada para caracterizar a situação de um utilizador, por exemplo, para fazer uma reserva de bilhetes para o teatro pode ser inicialmente proposto ao utilizador a compra de dois bilhetes. Neste caso a presença de outra pessoa (amiga) é considerado contexto, porque essa informação é utilizada para caracterizar a situação do utilizador. Este exemplo serve ainda para exemplificar a definição de um sistema com percepção de contexto segundo (Dey, 2000): "Um sistema tem percepção do contexto se utiliza o contexto para disponibilizar informação e/ou serviços ao utilizador, e em que a relevância depende das tarefas do utilizador". Todavia, com o aparecimento de grupos de investigação nesta área, surgiram outras definições sobre *context awareness*. Algumas dessas definições serão enunciadas seguidamente para se ter o conhecimento de como diferentes grupos de investigação abordaram este conceito.

Em (Schmidt, Beigl et al., 1999) os autores definiram um modelo de trabalho com contexto, que descreve a situação e o ambiente em que um dispositivo ou utilizador se encontra, como ilustra a Figura 2.1. Neste modelo o contexto é identificado por um único nome e para cada contexto, é identificado um conjunto de características relevantes que podem conter diversas variáveis próprias.

Figura 2.1 – Espaço como característica de contexto (Schmidt, Beigl et al., 1999).

De acordo com (Korkea-Aho, 2000), a informação de contexto é qualquer tipo de informação disponível durante o tempo de uma interacção: identidade, informação espacial (ex. localização, orientação, velocidade), informação do ambiente envolvente (ex. temperatura, luminosidade), informação temporal (hora, data, estação do ano), localização social (ex. acompanhantes, pessoas na vizinhança), recursos na vizinhança, disponibilidade dos recursos (ex. bateria, rede, largura de banda), parâmetros fisiológicos (ex. ritmo cardíaco, pressão arterial), actividade (ex. a correr, sentado, a conversar), tarefas e agenda.

Segundo Chen e Kotz (2000), existem duas definições de contexto: contexto activo que é um conjunto de estados de ambiente e configurações que conjuntamente determinam o comportamento da aplicação; e contexto passivo, em que não é necessário que a aplicação se adapte a ele, mas pode ser interessante mostrá-lo a potenciais utilizadores interessados. Estes autores definem ainda cinco tipos de categorias de contexto: contexto de computação (ex. largura de banda), contexto de utilizador (ex. perfil), contexto físico (ex. nível de luminosidade, temperatura), contexto temporal (ex. hora do dia, mês) e contexto histórico.

Lieberman e Selker (Lieberman and Selker, 2000) apresentaram outro entendimento de contexto, em que definem contexto com sendo tudo o que afecta a computação, excepto as entradas e saídas explícitas de dados, como se pode ver na Figura 2.2.

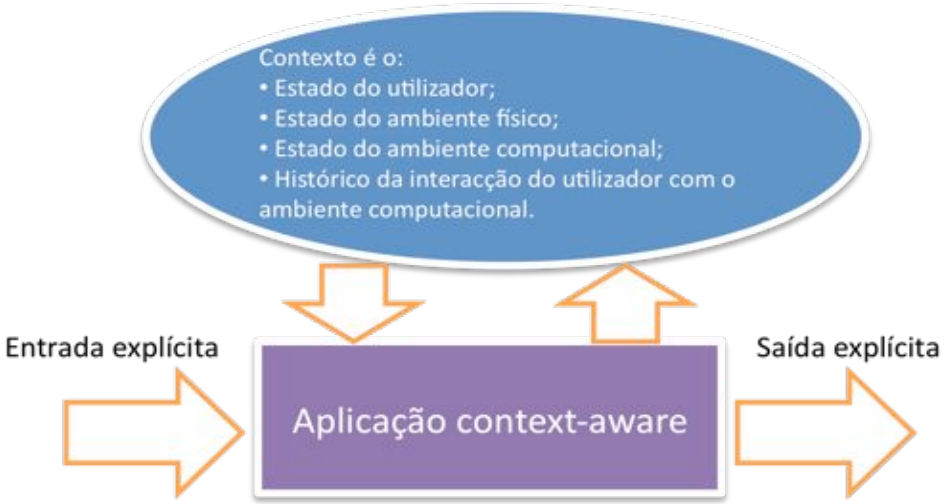


Figura 2.2 – Contexto na interacção utilizador versus Ambiente computacional (Lieberman and Selker, 2000).

Existem ainda outras definições de contexto, muitas delas feitas através de exemplos e que estão na maior parte contidas dentro das definições já abordadas. O entendimento sobre o que é o contexto nesta tese será baseado na definição de Abowd, Dey et al. (1999): "Define-se contexto como sendo qualquer informação que pode ser usada para caracterizar a situação de

uma entidade, em que uma entidade pode ser uma pessoa, lugar, ou objecto físico ou computacional.". Os mesmos definem ainda computação *context-awareness* ou computação *context-aware* como sendo a utilização de contexto de modo a disponibilizar informação relevante de tarefas e/ou serviços a um utilizador.

2.1.1 A importância da utilização do contexto nos sistemas computacionais.

Quando os humanos comunicam entre si, utilizam normalmente uma linguagem comum, à qual podem em determinadas situações adicionar outro tipo de informação para enriquecer a sua forma de comunicar e de expor melhor a suas ideias, como são os casos dos gestos, expressões faciais, volume e tom da voz, características da língua utilizada, entre outras. Contudo, o mesmo não acontece quando existe a necessidade de haver comunicação entre humanos e máquinas. Estas não sabem interpretar a forma de interagir dos humanos e não são sensíveis à informação contextual existente, pelo menos não tão facilmente como a maioria dos humanos. Na computação tradicional, os utilizadores usam dispositivos de entrada/saída de dados, como teclados e monitores para comunicarem com os sistemas computacionais. Habitualmente, neste tipo de comunicação os utilizadores traduzem o que pretendem em comandos ou eventos, para que os sistemas computacionais possam realizar o que pretendem. Esta comunicação é no entanto bastante precária e tende a eliminar informação contextual que tornaria o processo mais simples e fácil. Deste modo, só melhorando o acesso dos sistemas computacionais à informação contextual é possível melhorar a interacção homem-máquina e produzir serviços computacionais mais eficazes. Pedir aos utilizadores que introduzam cada vez mais e mais informação deliberadamente torna-se uma tarefa difícil e fastidiosa. Para além disso a maior parte dos utilizadores podem ainda não saber qual a informação potencialmente relevante, não sabendo deste modo que informação anunciar. Deste modo, o desenvolvimento de aplicações *context-aware* vem assumir essa tarefa, através da recolha automática de informação contextual, e tornar mais fácil o ambiente de interacção homem-máquina. O contexto fornece ainda grande potencial na selecção de informação e serviços relevantes para o utilizador, sendo essa selecção dinâmica dependente do contexto em que o utilizador se encontra.

Alguns dos autores de referência no estudo do *context-aware*, indicam algumas razões para utilizar *context-awareness*. Em (Schilit, Adams et al., 1994) os autores foram os primeiros a

discutir a computação *context-aware* e indicaram algumas razões para utilizar *context-awareness*:

- fornecer melhorar as aplicações móveis;
- aumentar a interacção entre o utilizador e o dispositivo móvel, filtrar informação endereçada ao utilizador, de modo a resolver os problemas de excesso de informação;
- melhorar a compreensão do significado da informação que é dada pelo utilizador à aplicação.

Por sua vez Pascoe em (Pascoe, 1998) indicava que o *context-aware* iria desempenhar um papel importante nos sistemas móveis devido às diversas situações que estes podiam ser utilizados e na sua integração com sistemas ubíquos, apresentado um sistema que permitia a utilizadores num determinado cenário realizar mais trabalhos, graças a um sistema *context-aware*.

Já Abowd e Dey em (Abowd, Dey et al., 1999) apresentam uma motivação mais genérica em que diziam que o objectivo da computação *context-aware* é tornar a interacção com os computadores mais fácil, apresentando nos anos seguintes diversos trabalhos nessa área (Abowd, Mynatt et al., 2002), (Dey, 2000), (Dey, 2000).

2.1.2 Classes de Contexto

Reconhecer as classes de contexto existentes e ter conhecimento sobre cada uma das delas é extremamente importante para o desenvolvimento de uma arquitectura *context-aware*. O reconhecimento de classes de contexto deve incluir tanto a recolha implícita como explícita de informação. Existem vários tipos de contexto e dependendo do cenário em questão a sua importância poderá ser diferente.

Em (Ryan, Pascoe et al., 1998) os autores identificam como tipos de contexto: a localização, o ambiente, a identidade e o tempo. Em (Schilit, Adams et al., 1994) identificam as classes de contexto como sendo: “onde estás”, “com quem estás” e “quais os recursos que estão na vizinhança”.

Dey e Abowd em (Abowd, Dey et al., 1999) identificaram quatro tipos básicos de contexto, que são: identidade, localização, actividade e tempo. Como se pode verificar, a única

diferença com os tipos definidos em (Ryan, Pascoe et al., 1998), é a substituição da actividade por ambiente. Os autores justificam que ambiente não passa de um sinónimo de contexto, enquanto que a actividade responde ao que está acontecer num determinado momento. Em (Abowd, Dey et al., 1999) os autores não se limitam apenas a definir as classes básicas de tipos de contexto para caracterizar a situação de uma entidade em particular. Os autores acrescentam que estes tipos primários de contexto não só respondem a todas as questões que podem ser colocadas sobre informação contextual, como também servem de índices para outras fontes de informação contextual, podendo até formar uma estrutura hierárquica. Por exemplo, a identidade de um utilizador pode ter diversos atributos, como morada, telefone, email, data de nascimento, ou uma lista de amigos. Do email pode por exemplo ser extraído o nome do utilizador da conta de email e o domínio. Neste caso ter-se-ia que a identidade faria parte do primeiro nível de contexto, o email faria parte do segundo nível de contexto e por fim o nome do utilizador faria parte do terceiro nível de contexto. Contudo, não nos parece que esta estrutura hierárquica advenha simplesmente da identificação das quatro classes primárias e muito menos da substituição de um dos tipos das classes de contexto em relação à proposta apresentada em (Ryan, Pascoe et al., 1998). Isto porque para definir a identidade de algo é necessária informação suficiente que caracterize essa entidade.

Esta caracterização permite ajudar os programadores a procurar, seleccionar e estruturar o contexto relevante a usar no desenvolvimento das suas aplicações.

2.1.3 Classificação de aplicações de contexto

Devido à complexidade de definir o conceito de contexto, faz sentido perceber como as aplicações de contexto podem ser classificadas. Isto permite saber que tipo de aplicações *context-aware* se pretende suportar e quais as características a estudar para o desenvolvimento de aplicações *context-aware*.

Em (Schilit, Adams et al., 1994), os autores classificam as aplicações *context-aware* em quatro classes:

- selecção próxima – é uma técnica que faz com que os objectos existentes na vizinhança sobressaiam automaticamente ou então que sejam mais fáceis de serem seleccionados;
- reconfiguração contextual automática – é um processo de reconfiguração que pode

adicionar ou remover componentes existentes, ou alterar a sua comunicação, devido a alterações de contexto;

- informação contextual e comandos – este processo pretende antecipar as acções dos utilizadores através da análise de determinadas informações. Desta forma o sistema deve ter consciência para oferecer serviços mais eficientes (p. ex. imprimir para a impressora mais próxima);
- acções desencadeadas por contexto – são as regras de decisão utilizadas para especificar como sistemas *context-aware* se devem comportar e eventualmente adaptar quando um determinado tipo de contexto ocorre.

Pascoe em (Pascoe, 1998) propõe uma taxonomia que visa identificar as principais características de *context-awareness*. A taxonomia proposta nesse trabalho é composta por quatro categorias: detecção de contexto (*contextual sensing*), equivalente à selecção próxima; adaptação contextual (*contextual adaptation*), equivalente às acções desencadeadas por contexto; descoberta de recursos de contexto (*contextual resource discovery*), equivalente à reconfiguração contextual automática; incremento contextual (*contextual augmentation*), que é a capacidade de associar dados digitais ao contexto do utilizador (utiliza a filosofia de anexar notas).

Em (Abowd, Dey et al., 1999) os autores propõem uma taxonomia de aplicações *context-aware* com três categorias: *apresentação de informação e serviços ao utilizador*, a *execução automática de serviços* e a *etiquetagem de contexto da informação para recuperação futura*. Embora os últimos três trabalhos ((Pascoe, 1998), (Abowd, Dey et al., 1999) e (Schilit, Adams et al., 1994)) apresentem diferentes definições sobre a classificação das aplicações *context-aware*, eles acabam por serem semelhantes no seu conteúdo. A *apresentação* de *informação* é a combinação da *selecção próxima* com a *informação contextual e comandos* de Schilit et. al (Schilit, Adams et al., 1994) e o mesmo que a propriedade *detecção de contexto* de Pascoe. Um exemplo desta primeira característica é o de um aluno que entra no *campus* universitário e que lhe são apresentadas as actividades da agenda do dia, por ordem de interesse. A categoria *execução automática* é igual à propriedade de *acções desencadeadas por contexto* de Schilit et. al. (Schilit, Adams et al., 1994) e à *adaptação contextual* de Pascoe (Pascoe, 1998). Um exemplo desta segunda característica é o caso de um docente que imprime um documento e essa impressão ocorre na impressora que está mais próxima. A última característica, *etiquetagem de contexto*, é semelhante ao *incremento contextual* de

Pascoe (Pascoe, 1998). Um exemplo desta característica é o caso de um docente que teve de mudar de sala à última hora e é colocada uma *nota virtual* à entrada da sala; quando um aluno se aproxima é informado no seu PDA que a aula está a decorrer noutra sala. Em (Dey, 2000) o autor indica que não utilizou a exploração de recursos locais, que corresponde às características de *reconfiguração automática contextual* de Schilit et. al. (Schilit, Adams et al., 1994) e à *descoberta contextual de recursos* de Pascoe (Pascoe, 1998). Segundo ele, não vê estas características como uma categoria de características separadas, mas, pelo contrário, como estando integradas nas duas primeiras categorias que definiu (*apresentação de informação e serviços ao utilizador* e a *execução automática de serviços*).

2.1.4 Modelos Contextuais

Os modelos contextuais exprimem a visão de alto nível da arquitectura de um sistema sensível ao contexto. Como tal, o seu desenho é desenvolvido de acordo com o âmbito da aplicação a que se destina. Deste modo, o facto de uma arquitectura utilizar diferentes domínios contextuais (ex. domínios de escritório, descrição de dispositivo computacionais) vai interferir na modelação dos elementos que afectam o conhecimento, serviços e acções que irão estar disponíveis para o utilizador quando o contexto for utilizado.

Um modelo contextual é necessário para definir e armazenar os dados relativos ao contexto de uma forma computacionalmente processável. Em (Strang and Popien, 2004) os autores identificam as soluções de modelos contextuais mais relevantes da comunidade científica, baseada nos dados das estruturas usadas para representar e trocar informação contextual, descrita em seguida:

- modelos par chave-valor (*Key-Values models*) – este modelos representam a estrutura mais simples de modelação de contexto. O par *chave-valor* é usado para descrever as capacidades do recurso. Este modelo é utilizado no sistema *Context Toolkit* (Dey, 2000) que será descrito na secção 2.5;
- modelos de esquemas de marcação (*Markup Scheme Models*) – todos este modelos têm em comum a utilização de uma estrutura hierárquica de dados em que se utilizam *tags* de marcação com atributos e conteúdo;
- modelos gráficos (*graphical models*) – linguagens de modelação como a *Unified Modelling Language* (UML) e a *Object-Role Modelling* (ORM) são utilizadas para

modelar aspectos contextuais;

- modelos orientados aos objectos (*object oriented models*) – são usadas técnicas que a orientação aos objectos oferece (ex. encapsulamento, reutilização, herança) para modelar o contexto;
- modelos baseados na lógica (*logic based models*) – este modelos oferecem uma alto grau de formalismo. Factos, expressões e regras são normalmente usados para definir o modelo contextual. Podem ser utilizados motores de inferência para inferir novo conhecimento;
- modelos baseado em ontologias (*ontology based models*) – são usadas ontologias que representam a descrição de conceitos e relacionamentos. Permitem modelar informação fornecendo um alto nível de expressividade. Podem ainda ser utilizadas diferentes técnicas de inferência segundo o nível de expressividade utilizado nas ontologias. Este modelo é utilizado no sistema CoBrA (Chen, 2004), SOCAM (Gu, Pung et al., 2005) e outros que serão descritos na secção 2.5.

Estes modelos aparecem algumas vezes associados a linguagens, como é o caso do modelo gráfico que utiliza a linguagem UML, ou o modelo baseado em ontologias que usa a linguagem *Ontology Web Language* – OWL (McGuinness and Harmelen, 2004). No desenvolvimento de um modelo contextual para um sistema, existem essencialmente duas fases: modelação e implementação. Em cada uma destas fases podem ser utilizados diferentes modelos (Bolchini, Curino et al., 2007), por exemplo na fase de modelação pode ser utilizado o modelo gráfico/UML, enquanto que, na implementação do sistema e na sua utilização, as instâncias de contexto podem ser descritas por outro modelo, por exemplo o modelo baseado em ontologias/OWL. Na conclusão do seu trabalho (Strang and Popien, 2004) os autores, baseando-se em seis requisitos que indicam como sendo essenciais aos sistemas sensíveis ao contexto, avaliam os diversos modelos e concluem que aquele que preenche mais requisitos é o modelo baseado em ontologias.

Em (Bolchini, Curino et al., 2007) os autores apresentam uma *framework* de avaliação, que permite aos programadores de aplicações comparar modelos contextuais segundo o âmbito da aplicação a que se destina. Ainda neste trabalho e utilizando esta *framework*, os autores fizeram uma comparação dos modelos mais importantes da literatura actual. Após uma análise orientada aos dados de modelos de contexto os autores apresentam diversas conclusões. Como vantagens, indicam que a utilização de ontologias nos modelos contextuais permite a

reutilização de ontologias já existentes e a inclusão de novos vocabulários; as ontologias facilitam a partilha e integração de vocabulários de contexto, embora em algumas situações sejam necessários mais padrões. Entre as desvantagens e problemas a resolver apontam que: é necessário desenvolver técnicas de gestão distribuída de contexto, principalmente em modelos de armazenamento contextual entre terminais finais; a inferência em contexto e a qualidade do contexto ainda não estão bem desenvolvidos; o suporte para a inferência é limitado havendo a necessidade de mais investigação nesta área; e as fontes do modelos contextuais existentes são diversas, mas são ainda poucos trabalhos na área de avaliação de modelos contextuais.

Na secção seguintes, são apresentados dois modelos ontológicos de referência na área das aplicações sensíveis ao contexto. Posteriormente, na secção 4.5, será descrito com mais detalhe o modelo ontológico SeCoM (Neto and Pimentel, 2005) utilizado no desenvolvimento da arquitectura que é apresentada neste trabalho.

2.1.4.1 SOCAM

O modelo contextual SOCAM proposto em (Xiao, Wang et al., 2004) é dividido em ontologias superiores e ontologias de domínios específicos, como ilustra a Figura 2.3. As ontologias superiores capturam conhecimento contextual sobre o mundo físico em ambientes de computação (*Service, Application, Device, Network, Agent, CompEntity, Location, IndoorSpace, OutdoorSpace, ContextEntity, Activity, ScheduledActivity, DeducedActivity, Person*). As ontologias de domínio definem detalhes de conceitos gerais e as suas propriedades em cada sub-domínio (ex. ontologia que descreve o domínio veículo, ou vinho).

Este modelo é especificado em linguagem OWL, que descreve as ontologias contextuais e que permite fazer inferência de lógica de primeira ordem (*first order logic*) (Fitting, 1996).

Na Figura 2.3 é possível ver as duas camadas de ontologias que compõem o modelo SOCAM, e a ligação das ontologias que definem os detalhes às ontologias superiores. Esta ligação é dinâmica e pode ser realizada sempre que necessário (ex. maior detalhe devido a alterações do ambiente contextual).

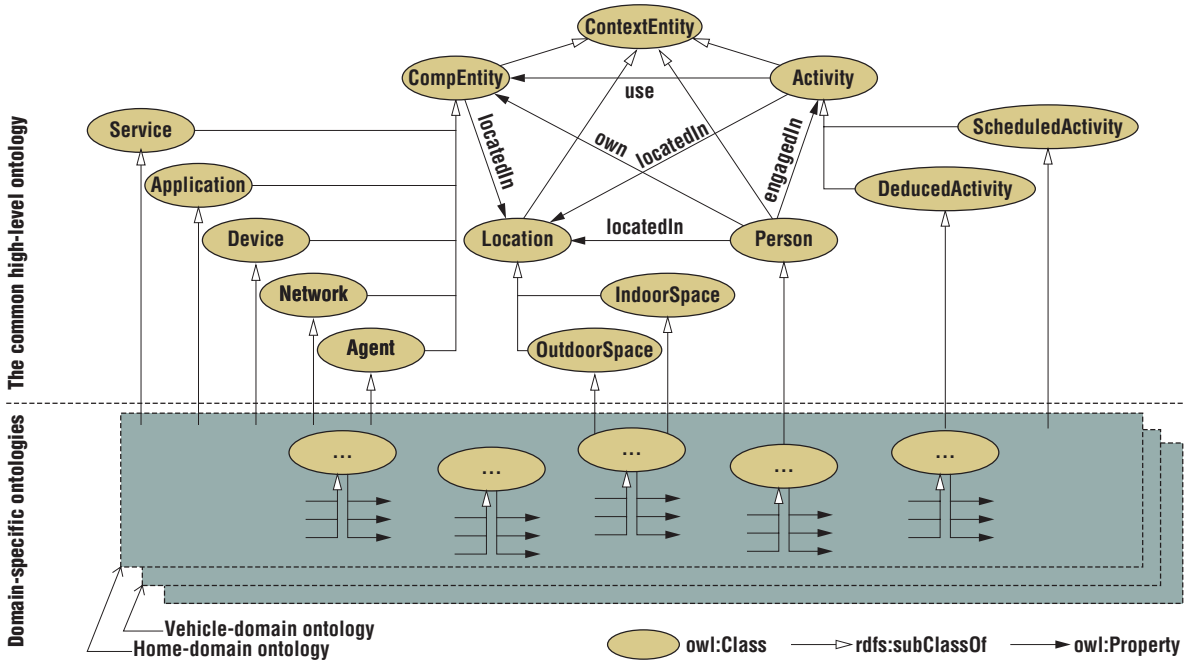


Figura 2.3 - Diagrama de classes das ontologias contextuais do SOCAM (Xiao, Wang et al., 2004).

2.1.4.2 SOUPA

O modelo contextual SOUPA (Chen, Perich et al., 2004) apresenta uma estrutura modelar de vocabulários que é usada no sistema CoBrA (Chen, 2004) (explicado na secção 2.5.8). O objectivo desta ontologia é definir ontologias para suportar aplicações contextuais. A ontologia SOUPA é expressa utilizando a linguagem OWL para descrever os domínios ontológicos ilustrados na Figura 2.4. Como é possível ver na mesma figura, o SOUPA é dividido em dois conjuntos de ontologias: (1) o núcleo SOUPA, composto por nove ontologias: *Agent*, *Person*, *Policy*, *BDI*, *Space*, *Geo-spatial*, *Time*, *Action*, *Event*; estas ontologias definem vocabulários genéricos que são universais para a construção de aplicações *pervasive*; (2) o conjunto de extensões SOUPA é constituído por seis ontologias: *Conditonal Belief*, *Regional Connection Calculus*, *Priority*, *Contact Preference*, *Schedule*, *Meeting*. Estas ontologias estendem o conjunto de ontologias do núcleo SOUPA. Definem vocabulários adicionais para suportar tipos específicos de aplicações e fornecem exemplos para definir novas extensões ontológicas.

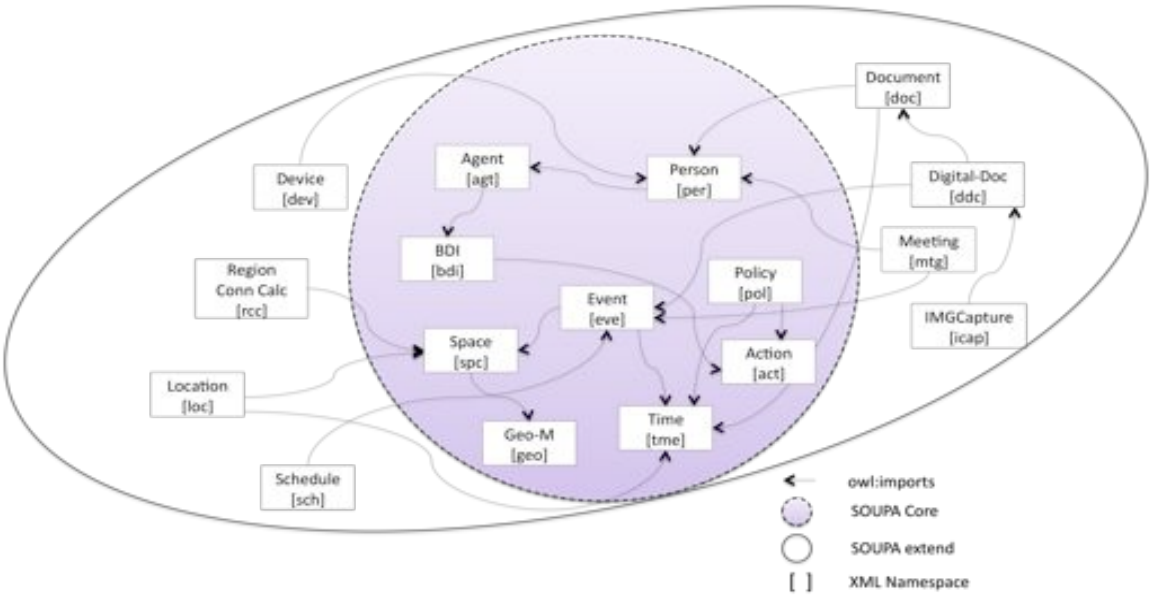


Figura 2.4 – Modelo contextual SOUPA (Chen, Perich et al., 2004).

2.2 Arquitecturas Orientadas aos Serviços

Em (Haas and Brown, 2004) os autores definem uma Arquitectura Orientada aos Serviços (*Service Oriented Architecture* – SOA) como sendo um conjunto de componentes que podem ser invocados, cujas descrições de interfaces podem ser publicadas e descobertas. Dizem ainda que o serviço é um recurso abstracto, que representa a capacidade de executar tarefas que representam funcionalidades coerentes do ponto de vista das entidades fornecedoras e requisitantes. Para ser usado, o serviço tem de ser fornecido por um agente fornecedor. Existem ainda outras definições que contribuem para o esclarecimento sobre as arquitectura SOA e que são enunciadas em seguida.

Para (Mahmoud, 2005), SOA é um estilo de arquitectura para a construção de aplicações de software que utiliza os serviços que estão disponíveis numa rede como os serviços na *Web*, e que promove a independência entre componente de software para potenciar a sua reutilização. Um serviço é uma implementação de uma funcionalidade de negócio e esses serviços podem ser utilizados por clientes em diferentes aplicações ou processos de negócios. A arquitectura SOA permite ainda a reutilização de recursos existentes e novos serviços podem ser criados a partir de uma infra-estrutura de sistemas de tecnologias de informação já existentes.

Outra definição menos genérica é a de (Hashimi, 2003). Em SOA as aplicações são construídas a partir de componentes básicos chamados serviços, processos, etc, definidos em

termos da sua funcionalidade, executando habitualmente operações ao nível do negócio.

A SOA é uma forma de arquitectura de sistemas distribuídos que é habitualmente caracterizada pelas seguintes propriedades (Booth, Haas et al., 2004):

- vista lógica – os serviços são componentes de software com interfaces bem definidas que são independentes da implementação. A SOA faz claramente a separação da funcionalidade do serviço (*o que faz*) da sua implementação (*como faz*). A SOA foca-se claramente no que faz e não como faz, isto é os clientes desses serviços não se preocupam como esses serviços vão executar os seus pedidos, mas apenas com o resultado que é obtido;
- orientado às mensagens – utiliza um modelo de comunicação orientado à mensagens. Isto é, o serviço é formalmente definido em termos de trocas de mensagens entre o fornecedor e o cliente e não nas propriedades dos próprios agentes;
- orientada à descrição – os serviços são projectados para serem descritivos externamente para que possam ser descobertos e invocados por mecanismos de descoberta;
- orientado às redes – os serviços oferecem características que favorecem a sua utilização em rede, no entanto não é um requisito obrigatório;
- independentes das plataformas – os serviços são independentes da tecnologia garantindo a autonomia da plataforma que os suporta;
- granularidade – é uma forma de medir o tamanho dos componentes ou a descrição dos componentes necessários que compõem um sistema, ou mais simples o grau de modularidade do serviço. Pode haver dois tipos de granularidade a granularidade: grossa (*coarse-grained*) e a granularidade fina (*fine-grained*). Um serviço de granularidade fina é mais modular e mais flexível porque existem mais unidades funcionais. Pelo contrário, um serviço de granularidade grossa tende a possuir um menor número de operações cada uma delas com maior capacidade funcional. Existem vários factores que influenciam a granularidade do serviço. Um exemplo de um serviço de granularidade fina é por exemplo, um serviço que dispõe de vários métodos que permitem a pesquisa por categorias, produtos e outros itens que oferece grande flexibilidade ao cliente; no entanto, pode provocar a sobrecarga

da rede e ter fraco desempenho. O mesmo serviço mas com características de granularidade grossa permite por exemplo, apenas o retorno do catálogo num conjunto de categorias, reduzindo desta forma a sobrecarga na rede e melhorando desempenho à custa de menos flexibilidade para o cliente;

- modularidade – os serviços executam tarefas bem específicas potenciando o desenvolvimento incremental;
- baseado em contratos – permite a definição de contratos, em que os serviços estabelecem acordos definidos por um ou mais documentos de descrição de serviços. Existem duas propriedades chave na definição de contratos: as propriedades funcionais e as não funcionais. As propriedades funcionais indicam funcionalidades específicas que o serviço oferece e necessita (ex. as operações do serviço, entradas e saídas das operações dos serviços). As propriedades não funcionais abordam o modo como os serviços são fornecidos (ex. *Service Level Agreement* – SLA, QoS, custos, requisitos de segurança);
- composição – permite a construção de novos serviços mais complexos (serviços compostos) através da agregação de outros serviços.

O funcionamento da SOA define-se habitualmente por um paradigma *procura-liga-executa* (*find-bind-execute*) como é exemplificado na Figura 2.5. Neste paradigma existem três entidades: o cliente do serviço, o fornecedor do serviço e o mediador de serviços (*service broker*). O mediador é utilizado pelo cliente para encontrar um serviço através de diversos critérios. Se o mediador tiver esse serviço, disponibiliza-o ao cliente com um contrato e a sua localização. Esta informação disponibilizada ao cliente é sempre fornecida pelo servidor de serviços ao mediador quando regista o serviço.

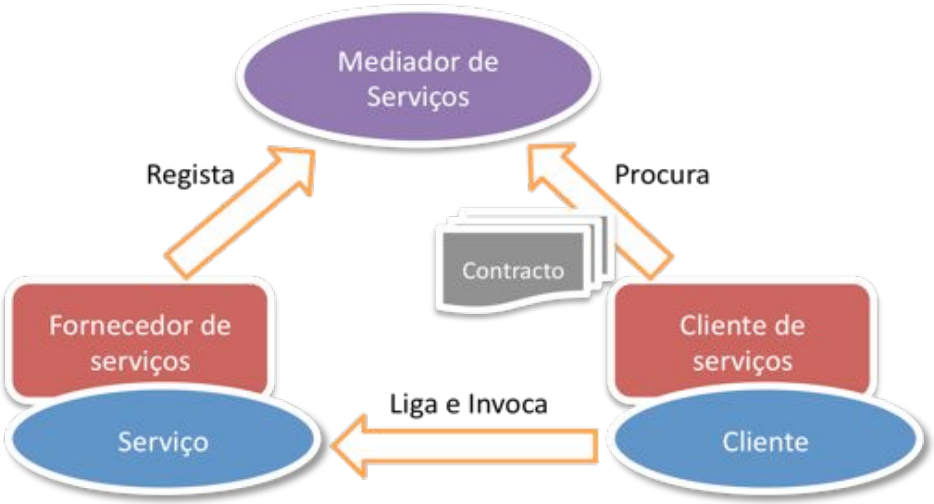


Figura 2.5. Paradigma procura-liga-executa das arquitecturas SOA. Baseado em (Mahmoud 2005).

Embora alguns dos conceitos SOA fossem especificados antes do aparecimento dos *Web Services*, a verdade é que estes são na actualidade praticamente a única tecnologia utilizada para a construção de arquitecturas SOA. Isto verifica-se pelo facto dos *Web Services* se basearem em normas abertos e são construídos em cima de protocolos bem conhecidos e independentes da plataforma. Entre esses protocolos figuram o HTTP, *eXtensible Markup Language* – XML (Bray, Paoli et al., 1998), *Simple Object Access Protocol* – SOAP (Gudgin, Hadley et al., 2003), *Universal Description, Discovery and Integration* – UDDI (Oasis, 2005) e *Web Services Description Language* – WSDL (Christensen, Curbera et al., 2001).

É a combinação destes protocolos que preenche os principais requisitos da SOA. Isto é, os protocolos UDDI, WSDL e SOAP permitem que os serviços sejam descobertos e invocados dinamicamente, o XML possibilita que os serviços sejam independentes da plataforma, o WSDL oferece uma vista lógica, e o HTTP fornece a interoperabilidade e os serviços orientados à redes.

2.3 Composição de serviços

A composição de serviços permite aos programadores ou utilizadores criar novos serviços ou aplicações numa arquitectura orientada aos serviços. O conceito de composição de serviços não é recente, foi importado do conceito de reutilização de software utilizado na Engenharia de Software, por exemplo como acontece no paradigma de orientação aos objectos. A reutilização de software consiste no aproveitamento de software escrito (herança) para

desenvolver novas aplicações com novas funcionalidades. Tal como na programação orientada aos objectos, na programação orientada aos serviços os programadores utilizam os serviços como componentes fundamentais nos seus processos de desenvolvimento de aplicações. Para que isso seja possível, os serviços oferecem um conjunto de funcionalidades essenciais, como ser independentes da plataforma e da rede e mecanismos que suportam a sua descrição, descoberta e invocação (explicados com maior detalhe nas secções 4.1 e 4.2).

A composição de serviços refere-se à capacidade de compor novos serviços mais complexos a partir de serviços relativamente mais simples, de modo a que os programadores possam resolver problemas mais complicados. Numa composição os serviços seleccionados para compor um novo serviço podem ser atómicos ou compostos. Um serviço atómico caracteriza-se por não poder ser dividido em dois ou mais serviços identificáveis separadamente; por exemplo, informação de localização, informação de estado de um utilizador, apresentação de áudio/vídeo, envio/recepção de uma mensagem. Um serviço composto é um serviço que é constituído por dois ou mais serviços que oferece uma funcionalidade de maior complexidade (ex. um serviço que envia mensagens para os utilizadores na proximidade).

2.3.1 Modos de composição

Em (Chakraborty and Joshi, 2001), os autores identificam diferentes modos de composição de serviços: composição pró-activa, composição reactiva, composição obrigatória de serviços e composição opcional:

- composição pró-activa – neste tipo de composição os serviços componentes são compilados criando um processo para distribuição e que será posteriormente executado. A composição do serviço pode integrar sub-componentes residentes na mesma plataforma, ou utilizar técnicas para comunicar com diferentes componentes em diferentes plataformas. Os serviços usados neste tipo de composição são normalmente estáveis e sujeitos a um número elevado de invocações. Um exemplo de composição pró-activa é o que se utiliza em sistemas de reserva on-line de voos das companhias aéreas. Neste caso, o serviço é criado através da composição de vários serviços e é colocado on-line para atender um volume elevado de pedidos, em vez de se criar o serviço de cada vez que um cliente o solicite;
- a composição reactiva – refere-se ao processo de criar um serviço dinamicamente

em tempo de execução "*on the fly*". Estes tipos de serviços são criados apenas quando existe um pedido de um cliente para a utilização desse serviço. Este tipo de composição é apropriado quando existem poucos pedidos para a composição de um serviço particular e os diferentes componentes se alteram com frequência. A composição de serviços reactiva explora melhor o estado actual dos serviços e pode ainda ser capaz de exercer optimizações em tempo de execução, monitorizando parâmetros em tempo real, como a largura de banda ou *Qualidade de Serviço* (QoS), entre outros. No entanto, a composição de serviços reactiva traz consigo um conjunto de problemas, tais como a validação da composição do serviço, a segurança, a verificação do correcto funcionamento do serviço (ex. *deadlocks*, ciclos infinitos, etc.), entre outros;

- a composição obrigatória – refere-se à classe de serviços compostos em que todos os seus sub-componentes têm que participar para a execução ser bem sucedida. O resultado da composição irá depender da execução correcta de todos os seus sub-componentes;

- a composição opcional – não necessita da participação de determinados sub-componentes para o correcto funcionamento do serviço composto. Por exemplo, numa vídeo-conferência entre duas pessoas, em determinado momento o componente de vídeo falha devido a restrições de largura de banda, contudo, dado que esta não é fundamental, a ligação entre os intervenientes mantém-se.

2.3.2 Esquemas de composição

Uma arquitectura que suporte à composição de serviços pode utilizar diferentes esquemas de construção de serviços. Alguns dos sistemas descritos em seguida apresentam diferentes esquemas de construção (Fujii and Suda, 2004):

- sistema baseado em modelos (*Templates*) – permite compor um serviço baseado num modelo pré-definido. O modelo pode definir tipos e regras de componentes necessárias para a composição de uma aplicação. Pode ainda especificar a própria estrutura da aplicação, como a ordem de execução dos componentes. Quando o utilizador pretende criar um serviço, são-lhe oferecidos vários modelos disponíveis num repositório. O serviço é construído através da selecção dos componentes e pela combinação de acordo com a estrutura descrita no modelo;

- sistema baseado em interfaces – permite criar um serviço com base na informação de um ou mais componentes. A composição do serviço consiste em juntar os componentes segundo um modelo de encadeamento, em que as saídas de uns serão as entradas de outros. Este sistema não suporta todo o tipo de serviços, dado que alguns deles não podem ser representados através de um conjunto de entradas e saídas;
- sistema baseado em lógica – neste caso, a composição é representada por uma fórmula que representa a lógica que deve ser satisfeita pela arquitectura. Este sistema consegue suportar uma grande variedade de serviços.

O sistema baseado em modelos será aquele que oferece um método de composição mais fácil ao utilizador; no entanto, a composição está sempre limitada aos modelos existentes sendo por isso menos flexível. Ao contrário do sistema anterior, o sistema baseado em interfaces permite criar qualquer tipo de serviço, desde que existam os componentes desejados, com as características desejadas. O sistema baseado em lógica apresenta um tipo de composição feita através da utilização de fórmulas, o que poderá trazer dificuldades para os utilizadores.

2.3.3 Orquestração e coreografia

A composição de serviços refere-se à construção de um processo de alto nível a partir de um conjunto de serviços. A *orquestração* e *coreografia* de serviços são dois conceitos relacionados como a integração e coordenação da execução de processos constituídos pela composição de serviços utilizados no desenvolvimento de sistemas distribuídos.

A *orquestração* de serviços descreve como os serviços podem interagir entre eles ao nível das mensagens, incluindo a lógica de negócio e a ordem de execução das interacções. Na orquestração existe sempre um ponto central, normalmente um serviço ou actividade de negócio, que coordena a execução de outros serviços para obter uma funcionalidade de maior granularidade. Um motor de orquestração implementa a lógica aplicacional necessária para orquestrar serviços atómicos e disponibilizar uma interface de alto nível para serviços compostos. Na orquestração o processo é sempre controlado da perspectiva de um dos parceiros de negócios. As linguagens de orquestração podem ser usadas para especificar os *workflow*s do serviço. Tipicamente é uma linguagem que fornece primitivas existentes nas linguagem de *workflow*, como as primitivas de condição *if-then-else*, de ciclos *while*, *split+join*, entre outras. A linguagem mais conhecida para a orquestração de serviços é a *Web*

Services Business Process Execution Language – WSBPEL, introduzia na secção 2.4 e que será descrita mais à frente na secção 4.2.7.

A *coreografia* descreve a colaboração entre o conjunto de serviços seleccionados para atingir um objectivo. Segue a sequência das mensagens que podem envolver múltiplas partes e múltiplas fontes, incluindo clientes, fornecedores e parceiros. Está também associada com a troca de mensagens pré-definida antes da execução e que ocorre entre múltiplos serviços. A coreografia é por natureza mais colaborativa, onde cada uma das entidades envolvidas no processo descreve a parte que representa na interacção. A linguagem mais utilizada para descrever a coreografia de serviços é a *Web Services Choreography Description Language* – WS-CDL (Kavantzas, Burdett et al., 2005), submetida ao W3C em Novembro de 2005 e engloba organizações como a Oracle, Novel, entre outras. É uma linguagem baseada em XML e que descreve colaborações entre participantes. Um dos objectivos da WS-CDL é fornecer formas para que as ferramentas validem a conformidade das descrições de modo a assegurar a interoperabilidade entre os *Web Services* colaborantes.

2.4 Composição de *Web Services*

Com o aumento da utilização dos *Web Services* nas arquitecturas orientadas aos serviços a composição de *Web Services* torna-se cada vez mais uma das principais preocupações dos processos de desenvolvimento de aplicações. O paradigma da arquitectura SOA tem como objectivo eliminar as dependências tecnológicas entre agentes de software e permitir a reutilização de funcionalidades. Deste modo, torna-se possível transformar aplicações mono-tecnológicas e rígidas em componentes de software flexíveis, permitindo a interoperabilidade entre diferentes tecnologias. Esta flexibilidade ocorre devido à própria natureza dos serviços fracamente ligados (*loosely-coupled*), assíncronos e baseados em documentos (*document-based*) possibilitando ainda a escalabilidade e adaptação às constantes mutações do negócio.

Grande parte das abordagens para a composição de serviços baseiam-se na utilização de *Web Services* e os motivos que estão na base da escolha desta tecnologia prendem-se, entre outros, com os factos de:

- os *Web Services* permitirem o desenvolvimento independente da tecnologia utilizada para construir as aplicações;

- serem uma componente aplicacional programável e acessível através de protocolos utilizados na Internet como HTTP, SMTP, FTP, etc;
- existirem arquitecturas que permitem a composição de serviços.

Foram identificadas seis abordagens distintas para a composição de *Web Services* (Milanovic and Malek, 2004): *Web Services Business Process Execution Language* (WSBPEL) (Oasis, 2008), *Semantic Markup for Web Services* – OWL-S (David Martin, 2004), Componentes *Web* (Yang and Papazoglou, 2002), Composição Algébrica Processual (Milner, Bauer et al., 1993), Redes Petri (Hamadi and Benatallah, 2003) e o Modelo de Verificação e Máquinas de Estado Finitas (Bultan, Fu et al., 2003). Estas abordagens são descritas seguidamente:

- WSBPEL – é uma linguagem baseada em XML que permite a especificação formal de processos de negócio e protocolos de interacção. Esta linguagem é uma continuação da linguagem *Business Process Execution Language for Web Services* (BPEL4WS) desenvolvida pela IBM, Microsoft e SEA Systems entre outras. Foi adoptada como norma pela OASIS. Será descrita mais à frente na secção 4.2.7;
- OWL-S – baseia-se no mesmo conceito da *Semantic Web*. Através dos conteúdos e dos metadados[1], os recursos da *Web* devem poder ser descobertos, compostos e invocados, quer pelos utilizadores, quer por agentes. Estas propriedades são possíveis através de uma linguagem de ontologias, que possibilita a descoberta automática de serviços, invocação, inter-operação, execução e monitorização de serviços;
- componentes *Web* – trata os serviços como componentes, de modo a suportar princípios básicos de desenvolvimento de software, como reutilização, especialização e extensão. Um componente *Web* é especificado através de duas forma: a definição de uma classe que encapsula informação lógica de composição e uma descrição do processo lógico em *Service Composition Specification Language* (SCSL), que contém as tarefas a serem realizadas para se obter a funcionalidade do componente *Web*.

[1] Metadados são elementos descritores que permitem localizar, caracterizar e relacionar recursos de um modo que possa ser compreendido por máquinas Berners-Lee, T. (1997). "Metadata architecture." Retrieved 11/2008, from http://www.w3.org/DesignIssues/Metadata.

- composição algébrica processual – este método modela o serviço como um processo móvel, isto é torna-o mais independente e mais abstracto, de modo a possibilitar a verificação de determinadas propriedades como a segurança, o ciclo de vida e a gestão de recursos;
- redes Petri – é uma técnica de modelação de processos bem definida. A sua utilização no modelo de serviços permite usufruir de determinadas propriedades que estas oferecem como: a validação da composição do serviço, verificando se não existem *deadlocks* e a composição irá terminar num número finito de passos. Estes aspectos são importantes para determinados cenários de sistemas distribuídos;
- modelo de verificação – o modelo de verificação é normalmente utilizado para a verificação de sistemas concorrentes de máquinas de estados. Este modelo pode ser utilizado para verificar a composição de *Web Services* numa especificação de *Workflow*, como por exemplo a consistência de dados, situações de *deadlock* e QoS. Os serviços comunicam entre si através de mensagens assíncronas e cada serviço tem uma fila sua. Existe um agente global que observa e rastreia todas as mensagens e que tem como objectivo mais ambicioso a composição automática de serviços, podendo-se saber se essa composição automática termina e em quantos ciclos.

As abordagens enunciadas anteriormente pretendem resolver problemas identificados na composição de serviços, como por exemplo a componente que está responsável pela verificação da sintaxe e da semântica da composição do serviço, a reserva de recursos, *deadlocks*, etc. Em (Milanovic and Malek, 2004) e (Srivastava and Koehler, 2003) é feita uma comparação das diversas abordagens no que diz respeito a características como a conectividade e propriedades não funcionais, a verificação da composição, a composição automática e a escalabilidade da composição. No entanto, outras preocupações deverão ser consideradas, tais como a complexidade da construção de serviços e a limitação dos recursos dos dispositivos móveis. Em relação à primeira, deve haver a preocupação do compromisso da criação de serviços de modo simples em função da flexibilidade pretendida. Um serviço mais flexível exige regras mais complexas e possivelmente conhecimentos técnicos específicos, perdendo-se a simplicidade necessária para os utilizadores finais. Em sistemas de composição através de fórmulas deve-se ter em atenção a utilização de uma sintaxe comum e

uma lógica fácil de definição da fórmula. A outra preocupação prende-se com o facto de os recursos limitados dos dispositivos móveis, como processamento e memória, poderem comprometer ou limitar a escolha da solução mais adequada para a composição de serviços.

2.5 Trabalhos na área

O aparecimento de arquitecturas orientadas aos serviços e sensíveis ao contexto tem progressivamente chamado atenção de grupos de investigação e da indústria das telecomunicações que vêem nesta área uma forma de proporcionarem novos serviços aos seus utilizadores/clientes e o acesso a novas formas de negócio.

Um número de sistemas *context-aware* tem sido desenvolvidos para demonstrar a utilidade da utilização do contexto. Os primeiros estudos nesta área foram essencialmente no desenvolvimento de aplicações *context-aware* focadas em escritórios, como o caso do *Active Badge* (Want, Falcao et al., 1992), projecto que permitia o redireccionamento de chamadas de uma central telefónica tendo em consideração a localização das pessoas dentro do edifício de trabalho.

Com o propósito de introduzir uma perspectiva global e mais concreta das arquitecturas que proporcionem a oferta de serviços sensíveis ao contexto para dispositivos móveis, são focados alguns exemplos que têm como âmbito a composição de serviços e/ou arquitecturas de fornecimentos de serviços baseadas em contexto. Entre estas foram escolhidas as arquitecturas mais relevantes, indicadas pelo número de publicações que contêm referências a elas.

2.5.1 ParcTab

O ParcTab (Schilit and Want, 1995) (Schilit, 1995), desenvolvido pelos laboratórios *Computer Science Lab* (SCL) na Xerox PARC, foi um dos primeiros sistemas a oferecer uma *framework context aware*. O ParcTab integrava um dispositivo móvel (neste caso um PDA) num ambiente de rede de um escritório e serviu como teste para a investigação na computação ubíqua. O dispositivo móvel é designado neste sistema por Tab e comunica através de infra-vermelhos (IR) com uma rede de *transceivers* IR. Este sistema pertence à categoria das aplicações *context-aware* para escritórios em que cada sala funciona como uma célula de

comunicação. Enquanto o Tab se vai deslocando entre células o ParcTab é capaz de suportar conectividade constante de modo a fornecer serviços de forma fiável. Esta infra-estrutura permite ainda saber em qualquer momento a localização do Tab dentro do edifício.

O ParcTab apresenta uma arquitectura composta por seis elementos (Figura 2.6), em que três deles são componentes de hardware: *Tabs*, *Transceivers* e *Rede Ethernet*; e os outros três são componentes de software: *ParcTab Agents*, *Infrared Gateways e Aplicações* (Schilit, Adams et al., 1993). Cada *Tab* é representado por um agente que é responsável por saber a localização do seu Tab e por fornecer através de procedimentos remotos essa localização a terceiros. As *Infrared Gateways* são responsáveis por enviar e receber os pacotes de dados utilizando sinais IR. E as *Aplicações* que permitem aceder a informações na rede e comunicar com outros utilizadores, por exemplo através de email.

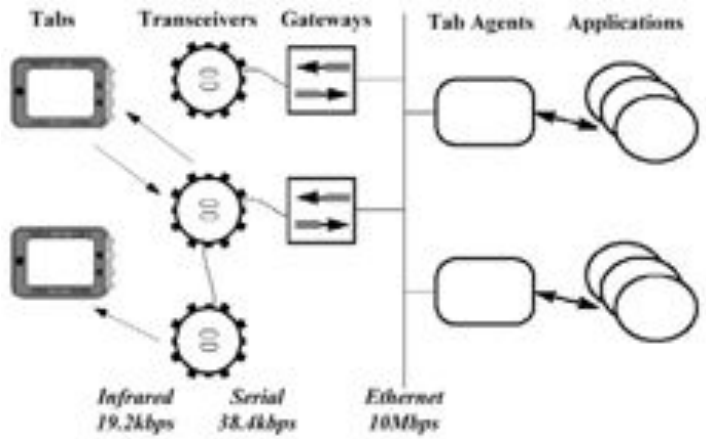


Figura 2.6 – Arquitectura ParcTab (Schilit, Adams et al., 1993).

Em (Want, Schilit et al., 1996) é descrito como o ParcTab foi implementado no laboratório da Xerox, chegando a existir 40 Tabs e cerca de 25 transceivers. Foram construídas várias aplicações para explorar os recursos da arquitectura. O Modula-3 foi uma dessas aplicações e oferecia diferentes níveis de acesso ao ParcTab e suas capacidades. Outra das aplicações desenvolvidas foi o MacTabbit, que permitia ao utilizadores através de um Tab aceder a aplicações Macintosh.

2.5.2 Context Toolkit

A arquitectura Context Toolkit resulta de um projecto com o mesmo nome desenvolvido no Georgia Institute of Technology (Dey, 2000). O Context Toolkit é um arquitectura que suporta o desenvolvimento de aplicações *context-aware*. Para isso, apresenta uma *framework*

modular com componentes reutilizáveis, que permite aos programadores construir de forma mais simples sistemas *context-aware* interactivos baseados em sensores.

A arquitectura do *Context Toolkit* (Figura 2.7) segue uma abordagem orientada aos objectos e consiste em três tipos de objectos: *context widgets*, *context interpreters* e *context servers*. (1) Os *context widgets* são componentes configuráveis que fornecem componentes (aplicações, *context servers*) com informação contextual obtida, por exemplo, através de sensores. Uma *widget* possui *atributos* (*attributes*) e *retornos* (*callbacks*). Os *atributos* são peças de informação disponíveis a outros componentes e os *retornos* são eventos que são utilizados para notificar outros componentes. (2) Os *Context Interpreters* são utilizados pelas aplicações para interpretarem e transformarem diferentes tipos de contexto. (3) Os *Context Servers* agregam dados contextuais de forma a ser possível obter um alto nível contextual. São particularmente adequados para a partilha de contexto, acesso ao histórico deste e funcionam como um proxy para a aplicação. A arquitectura contém ainda um componente de descoberta que funciona como *Registry*, capaz de suportar o registo de componentes. Esse registo acontece quando os *widgets*, *context intrepreters* e os *context servers* são instanciados e utilizam o *Registry* para armazenar diversas informações, por exemplo as suas funcionalidades. A comunicação entre as aplicações e estes componentes é feita através de mensagens XML trocadas em cima do protocolo HTTP. O contexto também é descrito utilizando a linguagem XML.

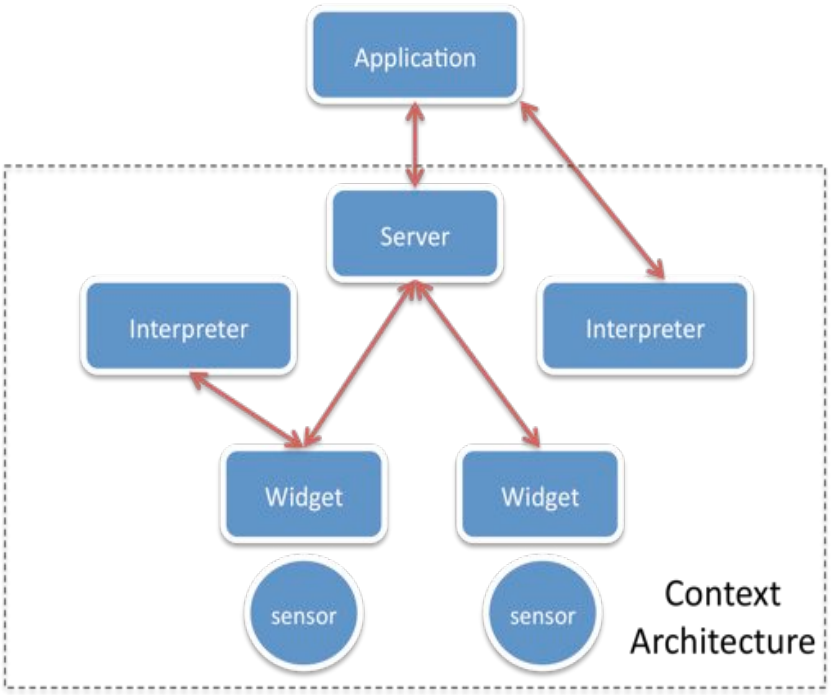


Figura 2.7 – Arquitectura *Context Toolkit* (Dey, 2000).

Várias aplicações foram desenvolvidas para validar a arquitectura do *Context Toolkit.* Um dos exemplos é a aplicação *Conference Assistant* (Dey, Salber et al., 1999), um protótipo móvel de aplicação *context-aware* para ajudar os participantes de uma conferência a escolher apresentações, tomar notas e recuperar essas notas. Outro exemplo foi a aplicação *Family Intercom*, que tinha como objectivo facilitar a comunicação entre utilizadores de uma residência equipada com diversos tipos de sensores.

2.5.3 Context-Aware Composition of Mobile Services

Em (Panagiotakis and Alonistioti, 2006) é apresentada uma arquitectura (Figura 2.8) que disponibiliza serviços multimédia sensíveis ao contexto para dispositivos móveis. A arquitectura proposta permite: a adaptação dos serviços ao terminal e à rede, às preferências do utilizador e adaptação dos serviços à localização. Esta *framework* abrange uma componente lógica aplicacional que orquestra a adaptação dos serviços ao contexto, permitindo criar os serviços de uma forma dinâmica, adaptando a entrega ao contexto do seu consumidor.

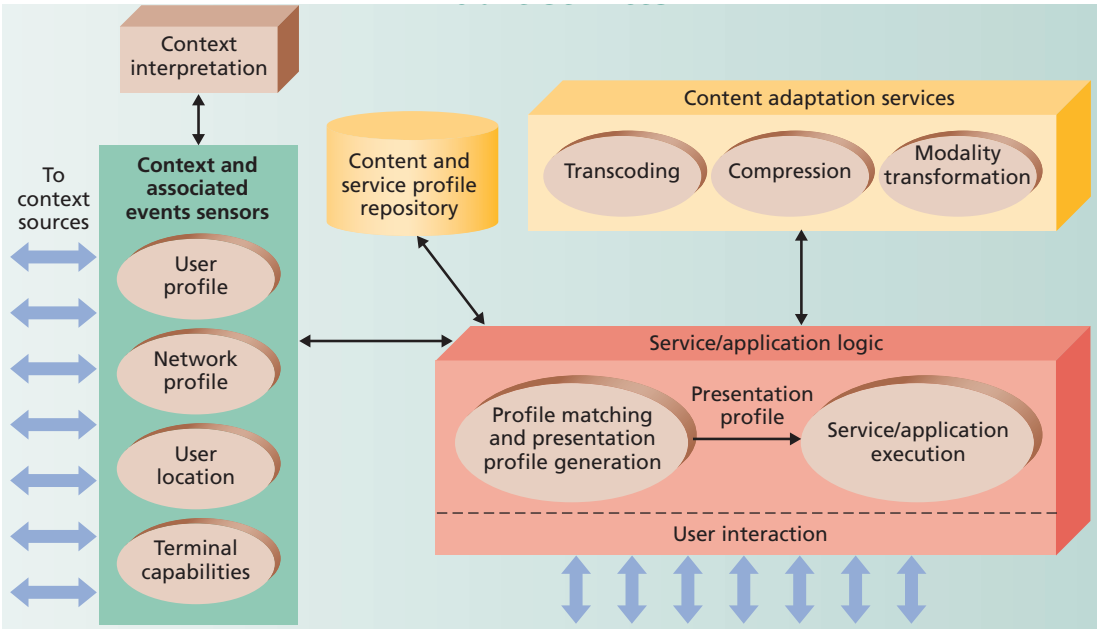


Figura 2.8 – *Framework* apresentada em (Panagiotakis and Alonistioti, 2006).

Um cenário prático será, por exemplo, o de um utilizador que está em sua casa a receber notícias em tempo real em vídeo, através de uma aplicação que as mostra no seu televisor. Quando o utilizador sai de casa para o escritório, passa a utilizar o seu telemóvel como fonte de visualização de informação. Através do contexto de localização, o servidor de informação

notifica a aplicação do novo terminal utilizado. A aplicação reconhece, através da configuração do seu perfil do utilizador ou interagindo com este, que ele deseja adaptar o vídeo para o formato do dispositivo móvel.

Trata-se de uma arquitectura modular, que permite adicionar e remover componentes da arquitectura, diferentes composições de serviços e configurações de distribuição física. A arquitectura contém um componente lógico que organiza e adapta os serviços ao contexto. O contexto é definido em termos de perfis por linguagens baseadas em XML, como a WSDL e OWL, entre outras. Estas linguagens são utilizadas na descrição dos seguintes tipos de componentes:

- *user preferences profile* – expressa preferências particulares do utilizador em relação ao modo como os serviços são disponibilizados. Pode incluir informações como: parâmetros de QoS dos serviços, linguagens preferidas, tamanho das fontes, preços, entre outros;
- *ambient profile* – informação sobre o ambiente como a temperatura e presença de outras pessoas;
- *terminal profile* – informações sobre a plataforma de hardware, software, protocolos, entre outros;
- *network profile* – identificação e descrição das redes próximas, monitorização em tempo real, variações de QoS fornecida pela infra-estrutura de rede e características técnicas das redes.

Quando um utilizador solicita um determinado serviço, o componente *service logic* utiliza a informação contextual do utilizador e o perfil do serviço solicitado. Através da análise da informação contextual e do perfil do serviço solicitado a *service logic* é capaz de gerar/adaptar um serviço que obedeça a todos os requisitos. Embora por diversas vezes os autores se refiram à funcionalidade de composição de serviços nesta arquitectura, fica a sensação que se trata de uma adaptação ou criação de um novo serviço com a mesma funcionalidade, mas com características que estão dependentes do contexto, como foi explicado no exemplo referenciado anteriormente.

2.5.4 myCampus

O *myCampus* é o resultado da pesquisa apresentada em (Sheshagir, Sade et al., 2004). Esta arquitectura é baseada em *Sematic Web* e utiliza agentes (representados na Figura 2.9 como *smiles*) capazes de encontrar informação contextual para os utilizadores de um *campus*.

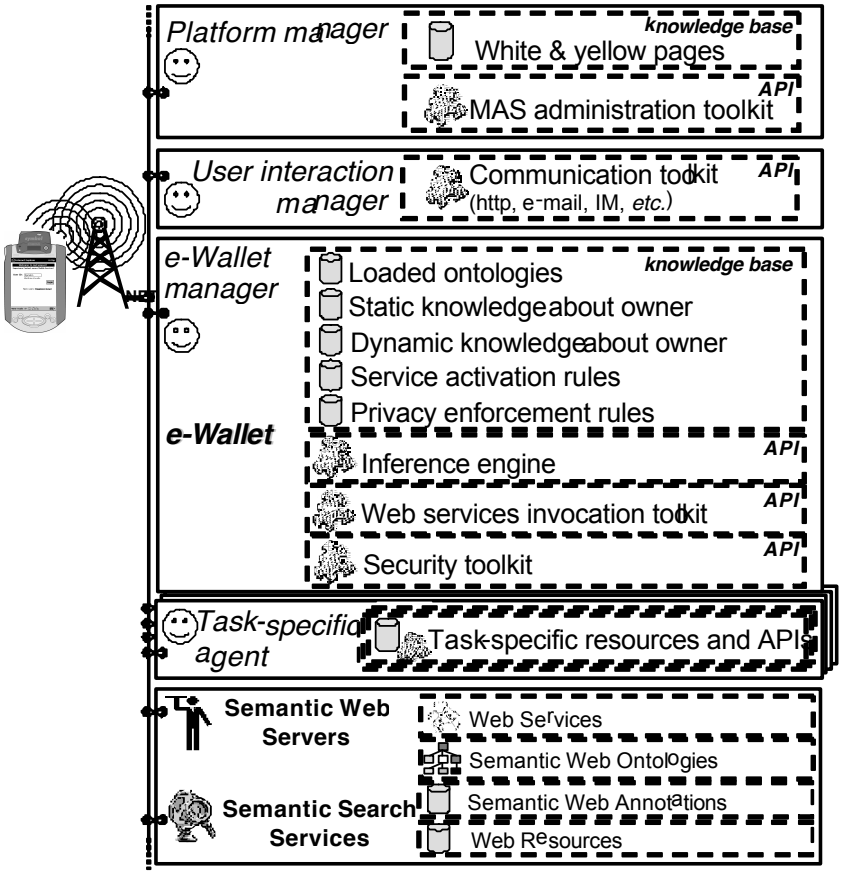


Figura 2.9 – Arquitectura myCampus (Sheshagir, Sade et al., 2004).

O componente principal da arquitectura é o *e-Wallet manager* ou simplesmente *e-Wallet*, que é um contentor de conhecimento estático sobre o utilizador, que suporta a descoberta automática e o acesso ao contexto. O *e-Wallet* também contém conhecimento, sobre como aceder a mais informação sobre o utilizador através da invocação de recursos representados por *Web Services*. Este conhecimento é guardado sob a forma de regras, que mapeiam diferentes atributos numa ou mais possibilidades de invocação de serviços, permitindo ao *e-Wallet* identificar de forma automática e activar o melhor recurso para responder às *queries* sobre o contexto do utilizador. Os utilizadores podem subscrever agentes especializados numa determinada tarefa para os assistir em diferentes tarefas contextuais usando a informação semântica que se encontra no *e-Wallet*. Estes agentes são capazes de descobrir, executar e compor automaticamente *Web Services Semânticos* utilizando a linguagem de descrição

semântica para *Web Services* (OWL-S). A composição automática de serviços pode ser vista como um planeamento. É feita através do mapeamento de serviços atómicos em operadores de planeamento e por algoritmos que ligam esses operadores. O plano gerado utilizando esses operadores constitui o serviço composto.

Um cenário de composição de serviço é descrito em (Sheshagir, Sade et al., 2004). Quando um utilizador pretende saber as previsões meteorológicas num determinado local, o sistema precisa de encontrar a localização actual do utilizador antes de invocar o serviço de previsão meteorológica. A localização do utilizador é obtida pelo *e-Wallet* do utilizador que invoca um *Web Service* para obter essa informação. Este tipo de composição é do tipo sequência e para ser possível é necessário utilizar as pré-condições. Quando um serviço tem uma pré-condição então esse serviço só pode ser invocado se a condição for satisfeita. A pré-condição é satisfeita com base no valor de saída ao qual está associada.

Esta arquitectura foi testada num ambiente controlado chamado também *myCampus*, o qual se caracteriza por ser um ambiente semântico, que tem como objectivo facilitar as tarefas do dia a dia dentro do *campus*. Os utilizadores podem subscrever agentes especializados em tarefas que os ajudam a realizar diferentes tipos de tarefas como sejam a marcação de reuniões, partilha de documentos, organização de eventos e filtragem e encaminhamento de mensagens. Um dos serviços implementados foi a recomendação de restaurantes, que sugere ao utilizador uma lista de restaurante e os seus pratos principais.

2.5.5 SOCAM

O SOCAM (Gu, Pung et al., 2005) é um *middleware* distribuído (Figura 2.10.) e sensível ao contexto que oferece suporte para a descoberta, aquisição, interpretação e acesso a informação contextual. O SOCAM apresenta um dos primeiros modelos de descrição do contexto semântico. Este modelo é constituído por um conjunto de ontologias que descrevem as principais classes do contexto: o domínio pessoa é descrito através da ontologia *person*, a localização através da ontologia *location*, o domínio actividade através da ontologia *activitiy* e o domínio computador através da ontologia *computer*. Baseado neste modelo contextual, o SOCAM converte o espaço físico onde o contexto é capturado em espaços semânticos, que são partilhados e acedidos por aplicações *context aware*.

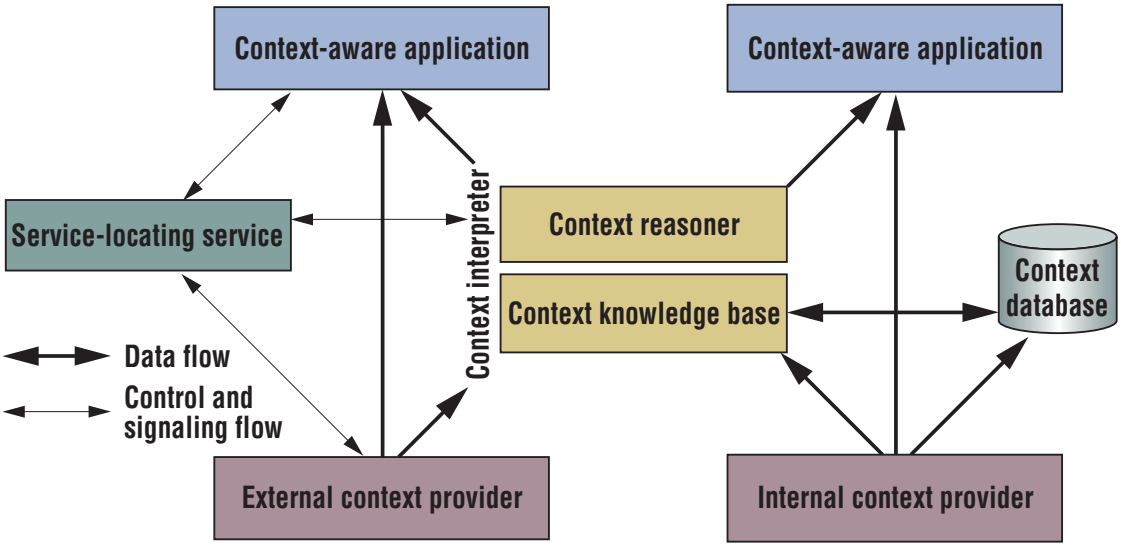


Figura 2.10 – Arquitectura SOCAM (Gu, Pung et al., 2004).

A arquitectura do SOCAM é composta pelos seguintes componentes:

- fornecedor de contexto (*context provider*) – permite a abstracção de diversas fontes de contexto internas ou externas e converte esses dados em representação *Resource Description Framework* – RDF (Lassila and Swick, 1999) e OWL para que os serviços possam ser partilhados e reutilizados. Os fornecedores de contexto interagem com uma camada de baixo nível responsável pela captura de contexto através de sensores que podem ser sensores físicos (ex. sensor de temperatura ou luminosidade) ou virtuais. Os sensores virtuais são representados por *Web Services*, como por exemplo um serviço de previsão meteorológica;

- interpretador de contexto (*context interpreter*) – fornece um serviço de raciocínio lógico para processar informação contextual. Este componente inclui dois elementos básicos: uma base de conhecimento de contexto (*context knowledge base*) e um motor de inferência de contexto (*context reasoner*), que permitem o acesso e a inferência de contexto;

- base de dados contextual (*context database*) – guarda ontologias de contexto e informação histórica sobre um domínio particular;

- aplicações contextuais (*context-aware applications*) – usam diferentes níveis de contexto, quer no formato de baixo nível capturado pelos sensores físicos ou na forma já processada pelo *interpretador de contexto*. Estas aplicações adaptam o seu comportamento ao contexto actual e segundo as regras especificadas;

- serviço de localização de serviço (*service locating services*) – fornece um mecanismo para que os *fornecedores de contexto* e os *interpretadores de contexto* possam anunciar a sua presença e que os utilizadores e as aplicações possam localizar serviços.

Para validar a arquitectura os autores desenvolveram uma aplicação (Gu, Pung et al., 2004) que passa música e ajusta as condições de luminosidade de uma sala, quando os membros da família estão a jantar na sua sala de jantar. Outras da aplicações desenvolvidas foi a *SituAwarePhone* (Wang, Dong et al., 2004) que adapta os dispositivos móveis a novas situações, minimizando a distracção do utilizador. Um dos cenários é o de um utilizador que ao entrar para uma reunião, a aplicação do telemóvel detecta um novo contexto, e vai inferir novas configurações para o telemóvel colocando-o em modo de vibração, para não perturbar a reunião.

2.5.6 CACS

Em (Nan, Junwei et al., 2006) os autores apresentam uma *framework* que permite a composição baseada no contexto de *Web Services*. Esta *framework* suporta a composição de serviços por objectivos (*goal-driven composition services flow*). A arquitectura CACS, ilustrada na Figura 2.11, é baseada em agentes, que são capazes de descobrir, compor, seleccionar e executar automaticamente *Web Services* usando a linguagem OWL-S. Esta é utilizada para descrever as capacidades dos serviços e as suas entradas/saídas, pré-condições e efeitos também conhecidas como: IOPEs (*Inputs, Outputs, Preconditions and Effects*[2]).

Os agentes podem executar algumas tarefas como tomadas de decisão baseadas nas necessidades dos utilizadores.

[2] Corresponde às entradas e saídas de um serviço, pré-condições para que seja possível a sua execução e quais os efeitos após a sua execução.

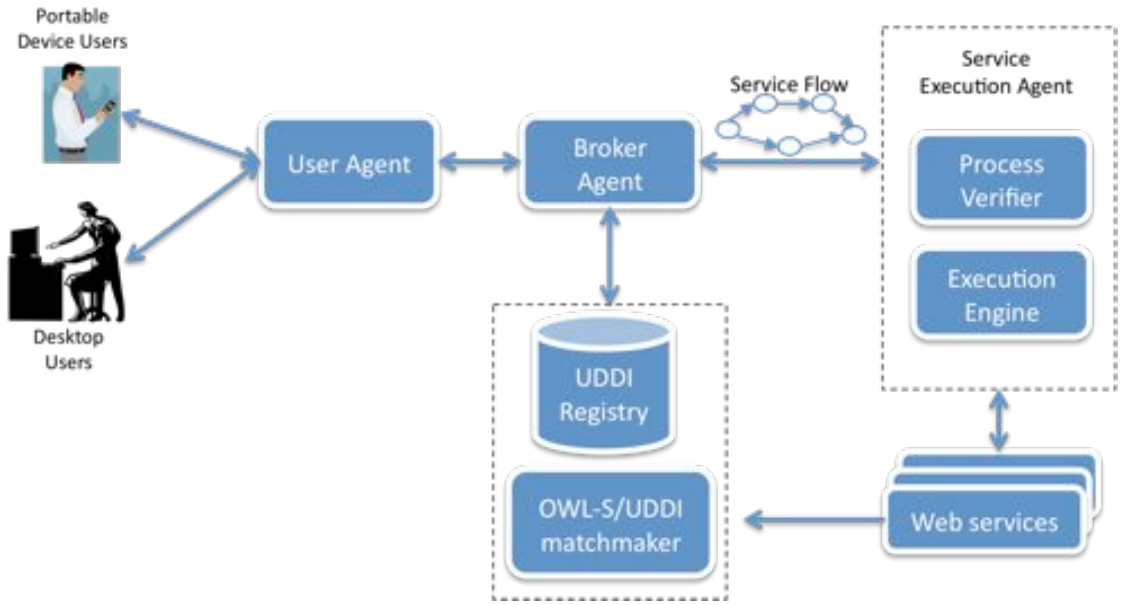


Figura 2.11 – Arquitectura da *framework* CACS (Nan, Junwei et al., 2006).

Os componentes que compõem a arquitectura CACS são descritos em seguida:

- *User Agent* – é usado para aceitar os pedidos que utilizadores fazem aos serviços e para retornar o seu resultado final de modo adaptado. Têm ainda a função de recolher informação contextual e memorizar as preferências do utilizador, sendo depois usadas quando o utilizador requisita um novo serviço;

- *Agent Broker* – recebe os pedidos do *User Agent* e extrai a informação contextual. Utiliza o UDDI *Registry* para localizar os melhores serviços para executar as tarefas descritas nos pedidos. Se o *Agent Broker* falhar em encontrar os serviços pedidos, irá decompor o pedido em múltiplos sub-objectivos de modo a encontrar serviços combinados similares como serviços candidatos. A decomposição irá parar quando todos os serviços candidatos forem encontrados. Após encontrar todos os serviços, o *Agent Broker* irá analisar as informações contextuais e as preferências do utilizador e irá escolher os melhores serviços do conjunto. Finalmente, a sequência de execução irá ser gerada e a informação sobre os serviços, endereços, portas e protocolos de invocação dos serviços envolvidos na sequência de execução será entregue ao *Service Execution Agent*;

- *Service Execution Agent* – é responsável por invocar serviços envolvidos na sequência de execução. Lida com o *binding* dos serviços e a execução de processos. Este componente contém dois elementos: (1) o *Process Verifier* que tem como responsabilidade verificar se a sequência de execução que o *Broker Agent*

gerou é ou não viável e verificar se existem erros (ex. de incompatibilidade) entre serviços que fazem parte dessa sequência. (2) O *Execucion Engine* que é usado para executar o fluxo de serviços previamente verificado.

Para modelar o serviço composto e observar as propriedades comportamentais (como a segurança, ou se é possível atingir o objectivo pedido) este sistema baseia-se na simulação do modelo. Posteriormente este modelo é validado através da utilização de modelos de Redes de Petri e vários métodos de análise como o *reachability tree*, *state equation* e *incidence matrix* que podem ser aplicados às Redes de Petri (Nan, Junwei et al., 2006).

2.5.7 Service Composition for Mobile Environments

Em (Chakraborty, Joshi et al., 2005) é proposta uma arquitectura e protocolos associados para a composição de serviços em ambientes móveis. Este estudo destaca factores que permitem a composição de serviços em redes *ad-hoc* como a mobilidade, alteração dinâmica da topologia dos serviço, heterogeneidade dos dispositivos, tolerância a falhas e consistência do desempenho. Os protocolos de composição são baseados em mecanismos de mediação e utiliza um processo de descoberta de serviços distribuído sobre redes de conectividade *ad-hoc*.

A parte esquerda da Figura 2.12 representa o ambiente de execução onde podem existir as seguintes entidades:

- *Request Source* – dispositivo móvel de onde é originado um pedido de composição de serviços;
- *Service Provider* – dispositivo móvel que contém o serviço que será invocado pelos outros nós;
- *Composition Manager* – o dispositivo que gere a descoberta, integração e execução de um pedido de composição;
- *Description-level Service Flow* – descrição declarativa de uma composição ou de um pedido de composição. É usada a linguagem *DARPA Agent Markup Language for Services* (DAML-S) para especificar a composição do serviço;
- *Execution-level Service Flow* – uma especificação completa do serviço composto com detalhes ao nível da execução, necessárias para invocar os serviços dos

Service Providers;

- *Atomic Services* – são serviços que residem num único *Service Provider* e que podem ser invocados por outros serviços. Podem depender de outros componentes mas estes têm de estar no mesmo *Service Provider*, para continuar a ser considerados atómicos;

- *Composite Service Length* – número de serviços atómicos que constituem o serviço composto.

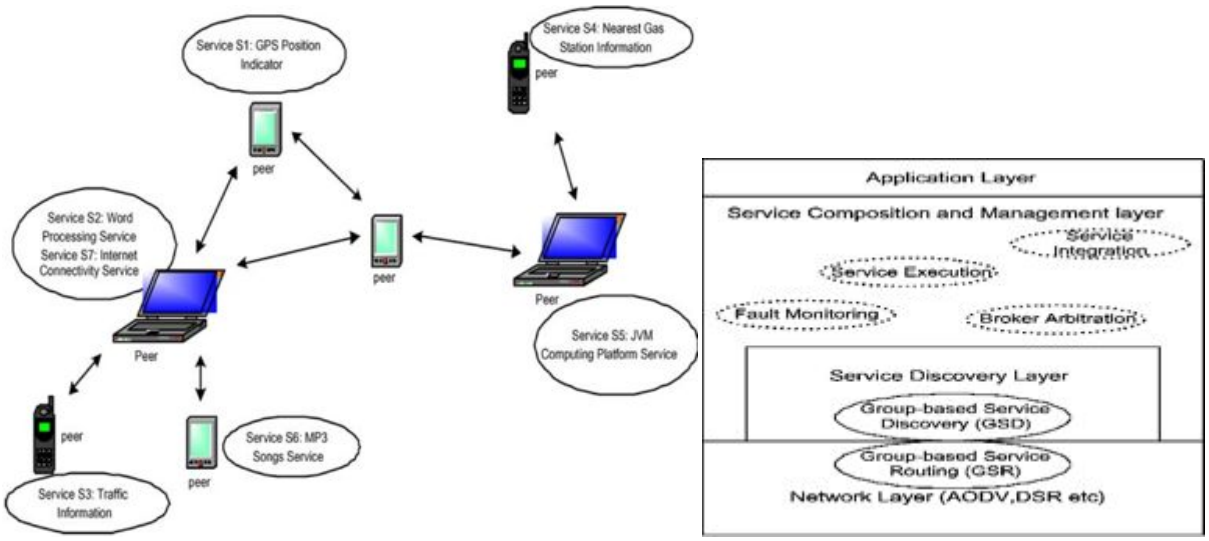


Figura 2.12 - Infra-estrutura do ambiente de execução (esquerda) e arquitectura (direita) (Chakraborty, Joshi et al., 2005).

A parte direita da Figura 2.12 mostra os componentes que residem em cada um dos nós móveis do ambiente de execução e que são:

- *Network Layer* – encapsula os protocolos de rede que fornecem conexões *ad-hoc* aos nós vizinhos;

- *Service Discovery Layer* – fornece ao sistema os protocolos necessários de modo a descobrir serviços que residem nos nós do ambiente móvel;

- *Service Composition Layer* – engloba os protocolos necessários para suportar o processo de gestão de descoberta e integração e execução da composição serviços.

- *Application Layer* – personifica qualquer camada de software que utilize esta plataforma de composição.

Embora com objectivos e motivações diferentes, este estudo forneceu várias respostas na área da composição de serviços para ambiente descentralizados lidando com características como a

utilização eficiente dos recursos, heterogeneidade de recursos, gestão da mobilidade, tolerância a falhas e fiabilidade.

2.5.8 Context Broker Architecture

A *Context Broker Architecture* – CoBrA (Chen, 2004) é uma arquitectura baseada em agentes que usa informação contextual para suportar espaços inteligentes. Estas sítios inteligentes são espaços físicos (ex. escritórios, salas de reunião, veículos) que são populados por sistemas inteligentes que auxiliam os utilizadores através de serviços recorrendo ao contexto.

A Figura 2.13 ilustra a arquitectura CoBrA e os seus relacionamentos com outros agentes num espaço inteligente. Todas as entidades computacionais num espaço inteligente tem um conhecimento sobre a presença do *Context Broker* e os agentes utilizam a norma *FIPA Agent Communication Language* (Agents, 2002) para comunicar. O coração do CoBrA é o *Context Broker* que gere o modelo contextual partilhado por uma comunidade de agentes. Estes agentes podem ser aplicações, serviços que podem correr em diferentes tipos de dispositivos móveis, dispositivos que estão na sala entre outros.

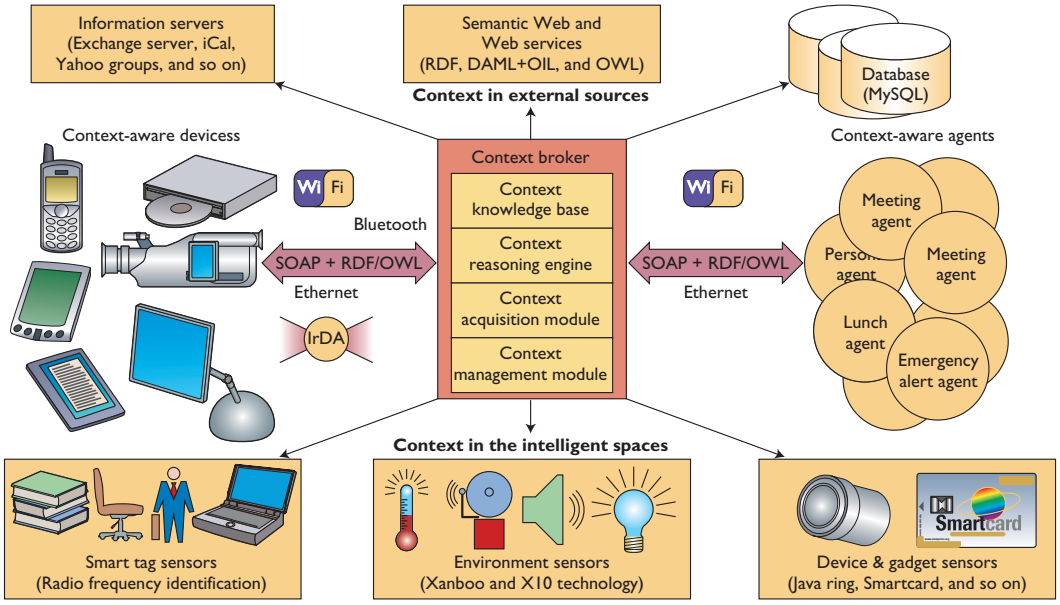


Figura 2.13 – Arquitectura CoBrA (Chen, Finin et al., 2004).

O *Context Broker* consiste em quatro componentes principais:

- *Context Knowledge Base* – é um armazenamento persistente do conhecimento contextual em formato RDF. Fornece um conjunto de API's para outros componentes para que estes acedem ao conhecimento guardado;

- *Context Inference Engine* – é um motor de inferência que raciocina sobre o conhecimento guardado. E capaz de deduzir novas informações de contexto a partir de informações que são obtidas das diversas fontes de contexto. Este motor permite dois tipos de inferência: a inferência baseada no modelo contextual semântico que esta arquitectura usa e a inferência baseada em regras do domínio da aplicação;
- *Context Acquisition Module* – é uma biblioteca de procedimentos que forma um *middleware* de abstracção da complexidade de aquisição de contexto de baixo nível, por exemplo dados de sensores. O papel deste componente é similar ao *Context Widget* no *Context Toolkit*;
- *Privacy Management Module* – um conjunto de regras de inferência que deduzem instruções para aplicarem as políticas do utilizador.

O CoBrA utiliza linguagens semânticas como as linguagens *Resource Description Framework* (RDF) e OWL, para definir ontologias de contexto, permitindo uma representação explícita do contexto que é apropriada para inferir e partilhar conhecimento. A ontologia Cobra importa diversas ontologias do modelo ontológico *Standard Ontology for Ubiquitous and Pervasive Applications* – SOUPA (Chen, Perich et al., 2004), apresentado na secção 2.1.4.2. Este modelo contextual semântico foi desenvolvido pelo grupo *Semantic Web in UbiComp Special Interest Group* (SWUCSIG, 2004) para suportar aplicações *prevasive* utilizando ontologias.

Para validar esta arquitectura os autores desenvolveram a aplicação *EasyMeeting*, cujo objectivo é apoiar actividades comuns de utilizadores em ambientes de reuniões. Assim os intervenientes numa reunião ou palestra eram auxiliados por serviços contextuais durante o decorrer da reunião ou palestra (ex. controlar a iluminação da sala).

2.5.9 Considerações finais dos trabalhos na área

Na Tabela 2.1 são resumidos os principais aspectos das soluções discutidas neste capítulo. A maior parte das arquitecturas aqui abordadas são baseadas em agentes, que em alguns casos se confundem com componentes que fazem parte do servidor, cujo sistema segue uma arquitectura cliente/servidor. Por este facto, à excepção da arquitectura CACS, todas as outras anunciam suporte para dispositivos móveis.

Tabela 2.1 – Sumário dos sistemas discutidos.

Caracterísi-cas	ParcTab	Context Toolkit	SOCAM	(Panagiotakis and Alonistioti,	myCam-pus	CACS	(Chakraborty, Joshi et al.,	CoBrA
Suporte a dispositivos móveis	Sim	Sim	Sim	Sim	Sim	Não especifica-do	Sim	Sim
Arquitectu-ra	Agentes	Cliente/ Servidor	Server	Cliente/ Servidor	Agentes	Agentes	Agentes	Agentes
Contexto	Apenas localiza-ção. Acoplado ao modelo de progra-mação	Na forma de *widgets* Acoplado ao modelo de programaç ão	Sim. Modelo contextual limitado	Sim. Em forma de perfis, utilizando linguagens baseadas XML	Sim, utilizando o OWL não modelo descritivo	Sim, utilizando o OWL não modelo descritivo	Sim, utilizando o OWL não modelo descritivo	Sim. Modelo contextual SOUPA
Processa-mento do Contexto	Orienta-do aos objectos	Pares atributo-valor	OWL	Diversas linguagens baseadas em XML	OWL	OWL	DAML	OWL
Composição/ Tipo	Não	Não	Não	Não	Sim/Auto-mática	Sim/Auto-mática	Supõem serviço já composto	Não
Abrangên-cia	Aplica-ções de escritório	Aplicações de escritório e colaborati-vas	Genérica	Serviços multimé-dia	Genérica	Genérica	Ambientes móveis altamente dinâmicos	Espaço inteligen-tes
Baseado em normas *Web*	Não	Não	Parcial	Parcial	Sim	Sim	Sim	Sim

Não existe até ao momento nenhuma linguagem descritiva normalizada ou modelo ontológico para a informação contextual, de modo a permitir a sua reutilização e partilha através das várias *frameworks* e *middlewares* contextuais. Embora esse normalização não exista, no entanto é essencial haver um desacoplamento entre o modelo contextual e o modelo de programação dos sistemas. Grande parte das arquitecturas aqui analisadas fazem essa separação e optam pela utilização da OWL como linguagem para descrever a semântica do contexto. Embora todas as arquitecturas utilizem diferentes modelos proprietários para suportar o contexto, o que utiliza um dos modelo mais apropriados é o CoBrA, porque se trata de um modelo que usa várias ontologias conhecidas na comunidade científica e que descrevem domínios particulares. A utilização de ontologias conhecidas traz vantagens ao nível da interoperabilidade com outras plataformas e partilha de informação contextual. Já o uso de ontologias específicas de domínios possibilita fornecer melhor detalhe na especificação e formalização do contexto.

Embora parte das arquitecturas não suportem a composição, foram ainda assim analisadas devido à sua importância na área do contexto. Entre as que suportam a composição de serviços, essa é feita de forma automática, existindo habitualmente um componente responsável por analisar a tarefa que o utilizador pretende executar, traduzindo-a em pré-condições e efeitos necessários pelo serviço composto. É gerado automaticamente um plano ou processo por entidade de inteligência artificial que não tem qualquer conhecimento prévio do *workflow*. Em algumas arquitecturas esse processo gerado verifica se existe algum serviço composto que possa satisfazer esse pedido. Caso não exista, então o objectivo dessa tarefa é subdividido em diversos objectivos de menor granularidade, por forma a serem mais fácies de satisfazer. Este processo cíclico termina quando se verificar o mapeamento de todos os objectivos com os serviços, que serão depois orquestrados por forma a responder ao pedido inicial do utilizador. Todo este processo exige serviços e ontologias com descrições pormenorizadas assim como a descrição de regras que prevejam os diversos cenários possíveis de invocação de serviços. Como é de esperar a composição automática de serviços é bastante complexa, exige a árdua tarefa de manter todos os detalhes dos seus intervenientes e as regras de invocação de serviços. Alguns estudos (Hull, Benedikt et al., 2003) e (Rao and Su, 2004) anunciam a composição automática como altamente complexa não sendo praticável a geração dos vários processos de forma automática.

A utilização de normas *Web* é uma mais valia na possibilidade de integração das arquitecturas com outros sistemas, permitindo quer explorar as suas funcionalidades quer partilhar, reutilizar ou trocar informação contextual com outras plataformas que utilizam informação contextual.

2.6 Resumo

Este capítulo teve como principal objectivo apresentar o estado da arte referente às áreas chave deste trabalho: o contexto e a computação sensível ao contexto, a arquitectura orientada aos serviços e a composição de serviços.

Este trabalho requer a interpretação de informações contextuais de modo a fornecer as melhores condições aos utilizadores no contexto em que eles se encontram em determinado momento. Como tal, é essencial conhecer a importância do contexto e as suas classes e de que forma estas podem classificar as aplicações contextuais. Deste modo, será mais fácil explorar

a semântica da informação contextual e as suas funções, como a pesquisa e extracção de conhecimento de um qualquer modelo contextual.

A Internet tornou-se uma plataforma global comum, em que pessoas e organizações interagem e se auxiliam no dia-a-dia para a execução de diversas actividades. São cada vez mais as ferramentas e aplicações orientadas a explorar os recursos de *Web*, como se pode observar actualmente em organizações como a *Google*, *Amazon*, *Microsoft*, *MySpace*, que abrem as suas plataformas, disponibilizando-as sob a forma de serviços na Internet. Estas plataformas são designadas arquitecturas orientadas aos serviços. Caracterizam-se por fornecer grande flexibilidade quer no modelo de desenvolvimento de software que na definição de processos de negócio, característica de extrema importância devido às constantes alterações dos modelos de negócio que as empresas tecnológicas estão sujeitas. Neste cenário, a composição de serviços como funcionalidade de uma aplicação surge naturalmente como um estímulo à exploração e combinação desses serviços bem como à sua reutilização e ao rápido desenvolvimento de novas aplicações. Dentro desta classe de serviços, os *Web Services* são actualmente a tecnologia mais utilizada, quer pelo sector empresarial, principalmente nas aplicações *bussiness-to-bussiness*, quer também por uma nova comunidade da *Semantic Web*, que pretende preencher a falta de semântica existentes nas normas desta tecnologia.

No final deste capítulo foram apresentados vários trabalhos na área da composição de serviços e na computação sensível ao contexto. Todos os trabalhos analisados partilham o mesmo objectivo de apoiar o desenvolvimento de sistemas sensíveis ao contexto e outros oferecem a possibilidade de composição de serviços.

Capítulo 3

SEMANTIC WEB

Inicialmente a *Web* apareceu como um modo de publicação de conteúdos, alterando a ideia de utilização dos computadores que evoluíram da utilização exclusiva para processamento de informação para uma porta de acesso global à informação. Bem recebida por parte dos utilizadores rapidamente cresceu tornando-se no maior repositório de informação que disponibiliza todo o tipo de conteúdos, revolucionando o modo como as pessoas comunicam e o modo como os negócios são conduzidos. Embora tenham aparecido ferramentas que ajudem a trabalhar com os conteúdos *Web* (p. ex. motores de busca), a dimensão da WWW é de tal forma gigantesca que se torna difícil encontrar as informações que se pretendem, sendo muitas vezes os resultados infrutuosos quer quantitativamente quer qualitativamente. O principal motivo para tal ocorrência deve-se a que grande partes das pesquisas na *Web* utilizam uma estratégia baseada na sintaxe, deixando a responsabilidade da legitimação semântica do seu conteúdo para o utilizador. A *Semantic Web* tem como objectivo fazer com que essa validação semântica possa também ser realizada por uma máquina (p. ex. agentes de software, ferramentas, *Web Services*, sistemas de informação). Assim, com a *Semantic Web* pretende-se a busca do significado e enriquecimento dos dados, assim como a interacção entre máquinas. Há quase uma década Berners-Lee (Tim Berners-Lee, 2001) considerou que a *Semantic Web* era uma extensão da *Web* e que iria permitir à *Web* tornar-se numa rede em que a informação teria um significado, facilitando a comunicação e a cooperação entre os humanos e agentes de software.

A *Semantic Web* é uma área emergente que tem como objectivo tornar a *Web* mais rica, tornando possível expressar informação que as máquinas possam entender. Resultado de um esforço conjunto de um grande número de empresas e investigadores liderado pelo W3C, a

Semantic Web tem dois propósitos: (1) a integração de formatos comuns e concertação de dados de diversas fontes, que na *Web* original se baseiam na troca de documentos. (2) A compreensão semântica da informação tanto na óptica dos humanos como dos agentes de software, isto é, permitir a um humano ou agente de software extrair informação semântica da *Web*. Segundo Berners-Lee (Tim Berners-Lee, 2001) a *Semantic Web* não é uma *Web* separada, mas uma extensão da *World Wide Web* (WWW). O conteúdo *Web* pode assim ser expresso não só na sua linguagem natural, mas também numa forma em que a informação é dada com um significado bem definido. Para isso, é necessário organizar e combinar a informação e depois adicionar um significado, habilitando a cooperação entre agentes de software e humanos. O objectivo da *Semantic Web* é transformar o conteúdo actual da *Web* num formato em que utilizadores e agentes de software entendam o significado da informação, permitindo-lhes descobrir, partilhar e integrar informação de forma mais fácil (Berners-Lee, Connolly et al., 2007).

A Figura 3.1 (Miller, 2003) ilustra as diferenças entre a *Web* actual e a *Semantic Web*. A *Web* actual baseia-se na interligação de recursos feitas através de links limitados e não descritivos em que o significado apenas é entendido pelos humanos. Isto é, um utilizador quando acede a um recurso lê e interpreta a informação que ele contém e utiliza os links para navegar nesses recursos.

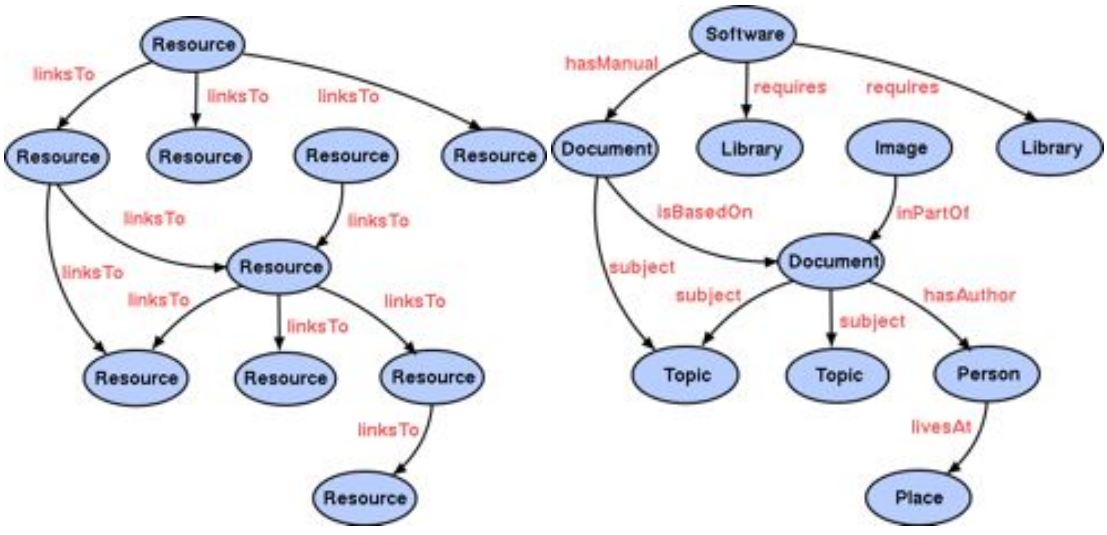


Figura 3.1 - A *Web* actual e a *Semantic Web* (Miller, 2003).

Na *Semantic Web* qualquer recurso é identificado globalmente por um *Uniform Resource Identifier* – URI (IETF, 1998). Os links entre recursos definem o tipo de ligação entre eles e tal como nos recursos, uma ligação é também identificada por um URI e contém propriedades que os podem relacionar. Com este tipo de informação estamos a adicionar uma camada de

metadados que fornece a semântica aos agentes de software. Os humanos conseguem assim extrair mais informação e os agentes de software conseguem obter informação processável.

3.1 Ontologias

As ontologias são consideradas um dos pilares da *Semantic Web* e com elas é possível representar explicitamente a semântica dos dados de um determinado domínio, permitindo a sua reutilização e partilha de conhecimento.

O termo ontologia deriva da filosofia e neste contexto é usado como nome de uma subárea da filosofia, ou seja, o estudo natural da existência, o ramo da metafísica preocupado com a identificação das coisas que existem e como as descrever (Antoniou and vanHarmelen, 2004). Entre a comunidade científica são encontradas várias definições para caracterizar as ontologias, que serão em seguida expostas de modo a perceber melhor o que é uma ontologia. Segundo Gruber em (Gruber, 1995), uma ontologia é uma especificação explícita de uma conceptualização. Já Guarino em (Guarino, 1997) definiu ontologia como uma caracterização axiomática do significado lógico. Em (Swartout and Tate, 1999) os autores entendem uma ontologia como um conjunto de conceitos e termos ligados entre si que podem ser usados para descrever um domínio de conhecimento ou construir uma representação de conhecimentos. Em (Sowa and Shapiro, 2006), uma ontologia define os tipos de coisas que existem no domínio de uma aplicação.

No contexto da computação e principalmente no contexto da *Web*, as ontologias fornecem o conhecimento sobre um domínio, tipicamente o senso comum conhecido do domínio, para que se possa partilhar a compreensão sobre um assunto entre aplicações. Qualquer ontologia típica para a *Web* utiliza uma taxonomia e um conjunto de regras de inferência. A taxonomia define as classes dos objectos e relações entre eles (Alesso, 2006). É possível expressar um elevado número de relações entre entidades atribuindo propriedades às classes e permitindo também que subclasses herdem essas propriedades. As regras de inferência nas ontologias podem expressar regras para manipular informação (ex.: se uma localidade tem associada a si um código postal e se uma qualquer morada utiliza essa localidade, então a morada está associada a esse código postal).

Segundo Berners Lee (Berners-Lee, Hendler et al., 2002), com a colaboração de ontologias será possível obter uma *Web* onde a informação será acessível e interpretada não só pelos

humanos mas também por processos automatizados. As ontologias são vistas como metadados que representam explicitamente a semântica dos dados, de uma forma processável por máquinas, sendo assim um factor chave para o desenvolvimento da *Semantic Web*.

Para representar o conhecimento num determinado domínio, as ontologias utilizam os seguintes componentes (Gruber, 1993) e (Asuncion, Oscar et al., 2003):

- conceitos – representam um conjunto de classes de entidades ou coisas dentro de um domínio (ex: hardware é um conceito dentro do domínio da informática);
- funções – são um tipo concreto de relações em que se identifica uma instância mediante o cálculo de uma função que considera vários elementos de uma ontologia;
- relações – descrevem as interacções entre conceitos e as suas propriedades. São normalmente usadas para formar a taxonomia de um domínio;
- instâncias – também conhecidas por indivíduos ou objectos, são utilizadas para representar os objectos de um determinado domínio;
- axiomas – são regras que se declaram sobre que relações devem ter os elementos de uma ontologia e são usados para modelar afirmações sempre verdadeiras. Permitem inferir conhecimento que não está declarado explicitamente na taxonomia dos conceitos (ex.: através de um mecanismo de herança, ou de relações). Exemplo de um axioma de relação: *se uma organização emprega uma pessoa então essa pessoa é empregada da organização*.

A Figura 3.2 ilustra um exemplo de uma ontologia para o domínio de um ser vivo. Representa uma hierarquia de conceitos encontrados no domínio de um ser vivo. As relações são definidas através de ligações entre os conceitos. Numa representação de uma ontologia, podem existir diversos tipos de relações. No caso da Figura 3.2, as relações mais predominantes são as relações de hereditariedade, que indicam que as subclasses herdam as características das super-classes. Por exemplo, qualquer humano é considerado um mamífero, herdando determinadas propriedades comuns aos mamíferos, como, por exemplo, quando nascem são amamentados. É possível ainda definir qualquer tipo de relacionamento característico de um domínio e permitir atribuir propriedades às classes. No exemplo da Figura 3.2 é definida uma relação denominada *come* entre *humano* e *peixes*, que define que *humanos comem peixes*.

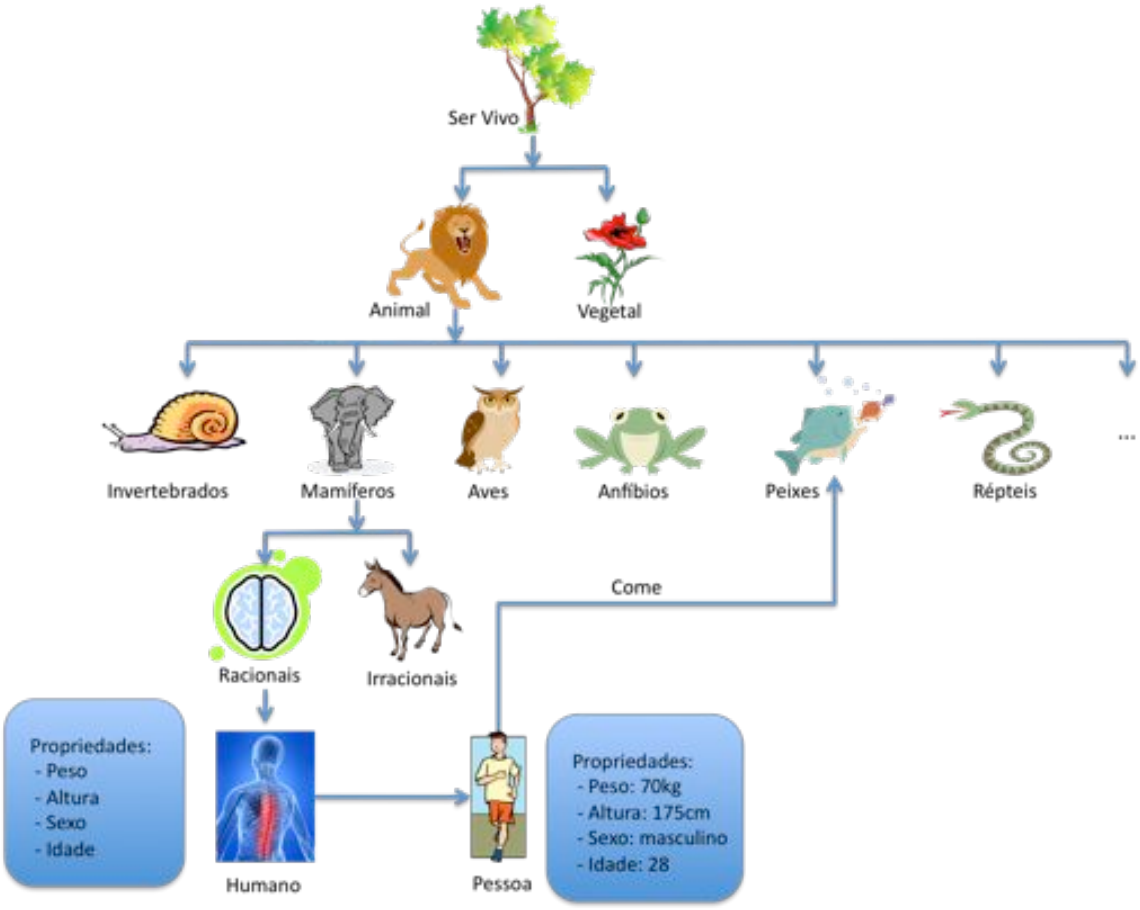


Figura 3.2 – Ontologia para um ser vivo.

Em (Alesso, 2006) os autores realçam que o poder da *Semantic Web* será atingido quando forem criados programas que sejam capazes de recolher conteúdos *Web* de diversas fontes, processarem essa informação e trocarem resultados. A eficácia do software irá aumentar exponencialmente à medida que apareçam mais máquinas capazes de interpretar a *Web* e serviços automatizados ficarem disponíveis.

3.1.1 Tipos de ontologias

Segundo (Guarino, 1998) as ontologias podem ser classificadas de acordo com o seu nível de generalidade em quatro tipos diferentes de ontologias: ontologias genéricas, de domínio, de tarefa e de aplicação. A Figura 3.3 ilustra estas quatro ontologias, descritas em seguida, e os seus relacionamentos de especialização, que são representados pelas setas:

- ontologias genéricas – descrevem conceitos genéricos como espaço, tempo, objectos, eventos, etc. São conceitos independentes de um domínio ou problema particular. O objectivo destas ontologias é unificar ontologias de topo destinadas a

grandes comunidades de utilizadores;

- ontologias de domínio – descrevem um vocabulário relacionado com um domínio genérico (p. ex. "ensino", "barcos") ou uma tarefa especializada (p. ex. "ensinar", "velejar") especializando os termos introduzidos nas ontologias genéricas;
- ontologias de tarefa – descrevem conceitos que dependem quer de domínios particulares quer dos domínios de tarefa, e a maior parte das vezes de especializações de ambas. Este conceitos normalmente correspondem às regras assumidas pelas entidades do domínio enquanto exercem um determinada actividade, por exemplo "pneu sobresselente";
- Ontologias de aplicação – descrevem conceitos que dependem de um domínio ou de uma tarefa particular e são normalmente especializações de ambas as ontologias. Estes conceitos correspondem algumas vezes a papéis assumidos por entidades de um domínio enquanto executam uma determinada actividade (ex. unidade de substituição ou componente sobresselente).

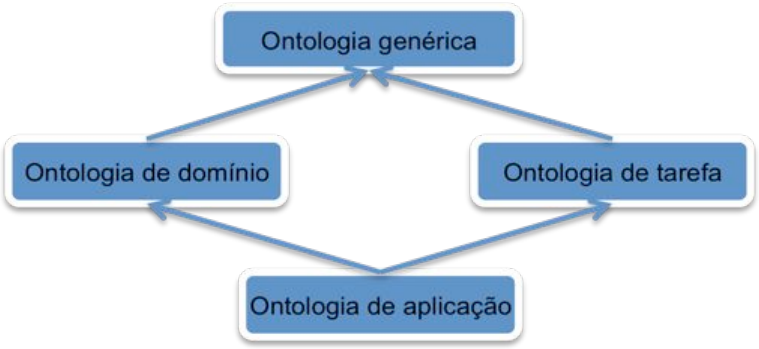


Figura 3.3 – Tipos de ontologias e os seus relacionamentos (Guarino, 1998).

3.2 Arquitectura da *Semantic Web*

A arquitectura da *Semantic Web*, ilustrada na Figura 3.4, é a versão 4 proposta por Berners-Lee em (Berners-Lee, 2006) e que se mantém desde 2006. As restantes versões podem ser encontradas em: versão 1 (Berners-Lee, 2000), versão 2 (Berners-Lee, 2003) e versão 3 (Berners-Lee, 2003). Em (Al-Feel, Koutb et al., 2008) os autores apresentam uma perspectiva da evolução das várias versões desta arquitectura.

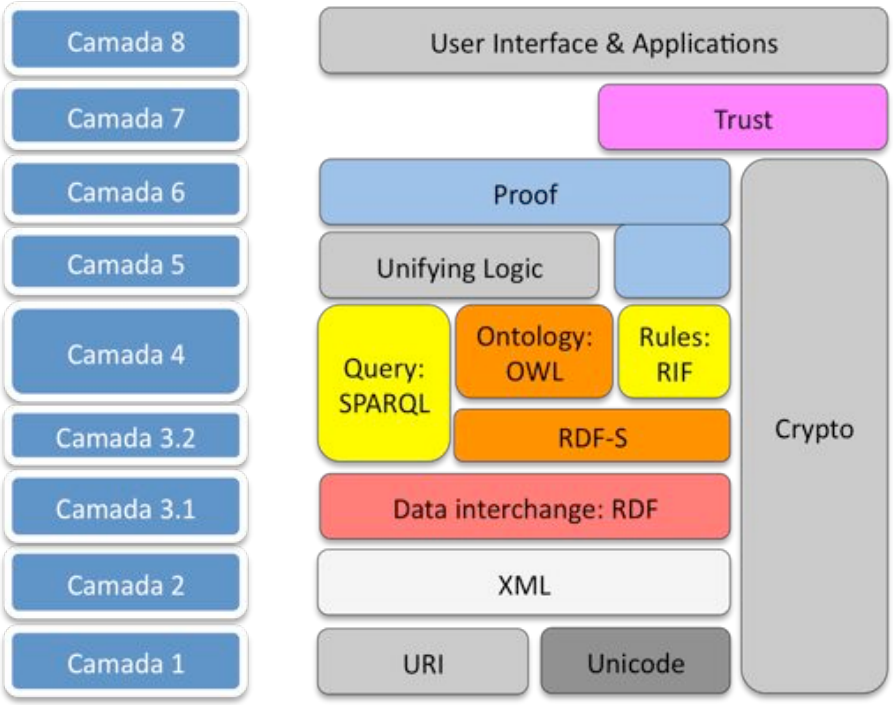


Figura 3.4 - Arquitectura da *Semantic Web* V4 (Berners-Lee, 2006).

A actual versão 4 da arquitectura *da Semantic Web* é composta por oito camadas em pilha e uma camada vertical que é comum a parte delas. As várias camadas e os seus objectivos serão brevemente descritos em seguida, sendo posteriormente nas secções seguintes abordadas com mais profundidade as camadas de descrição de descrição de dados, semântica e lógica:

- camada 1 (URI e *Unicode*) – também conhecida como camada básica de dados e a base da arquitectura é constituída pelo *Unicode* (Unicode, 2008) e o pelo URI fornecendo respectivamente as valências de legibilidade e de identificação da *Semantic Web*, respectivamente;

- camada 2 (XML) – também chamada a camada de descrição de dados é composta pela linguagem XML (Bray, Paoli et al., 2006). A XML é uma linguagem de marcação para documentos, que contém informação estruturada e como tal fornece um formato para o intercâmbio de informação. Esta camada incorpora ainda outras linguagem e conceitos como a linguagem *XML Schema* – conhecida como XMLS ou XML-S ou até *XML Schema* (W3C, 2001);

- camada 3.1 (RDF) – a camada *data interchange*: RDF (Manola and Miller, 2004), é utilizada para descrever recursos distribuídos, integrando os dados estruturados a partir de múltiplas fontes. Esta camada utiliza a linguagem RDF e é usada para representar a informação e fazer declarações sobre recursos;

- camada 3.2 (RDF *Schema*) – a *RDF Schema* também conhecida por RDFS ou RDF-S (Brickley and Guha, 2004) é utilizada para definir os vocabulários que podem ser referenciados por um URI. A RDFS fornece os elementos básicos necessários para a definição de ontologias, que podem ser usados para descrever os vocabulários e usar os vocabulários para descrever objectos, permitindo assim a inferência. Juntamente com a camada 3.1 forma a camada de descrição estrutural e semântica;

- camada 4 (*Ontology OWL e Rules: RIF*) – também chamada de camada de descrição semântica e lógica, é composta por duas linguagens: a OWL e uma nova linguagem chamada *Rule Interchange Format* – RIF (Hawke, 2006). A OWL é a linguagem recomendada pelo W3C para o desenvolvimento de ontologias. Ainda como *draft*, a linguagem RIF especifica um formato para o intercâmbio de regras entre sistemas baseados em regras. O objectivo é criar um formato de intercâmbio para linguagens de regras diferentes e motores de inferência também designados motores de raciocínios (*inference engines* ou *reasoners*);

- camada 5 (*Unifying Logic*) – acima da camada RIF, nenhumas tecnologias são enunciadas na versão V4 da arquitectura e por isso instâncias destas ainda não estão em discussão (Gerber, Merwe et al., 2008);

- camada 6 (*Proof*) – ainda não existem definições específicas para as camadas *Proof* e *Trust*. Mas a camada *Proof* deverá fornecer indicações aos agentes se eles deverão ou não acreditar nos resultados. Isto é, será usada para verificar a veracidade de uma declaração especifica (Al-Feel, Koutb et al., 2008);

- a camada 7 (*Trust*) – vários esforços têm sido feitos para se conseguir uma *Web* de confiança, mas esta é uma tarefa muito complicada que ainda não foi alcançada. A confiança na *Semantic Web* depende das fontes de informação assim como das políticas existentes nessas fontes prevenirem o acesso de utilizadores ou aplicações a essas fontes. É suposto disponibilizar um mecanismo para a confiança e confidencialidade entre fontes de informação (Al-Feel, Koutb et al., 2008). Esta camada está ainda em fase de pesquisa, não havendo muita coisa escrita sobre ela, nem existe nenhuma tecnologia recomendada pelo W3C, que seja responsável por esta tarefa;

- a camada 8 (*User Interface and Applications*) – define uma base de requisitos que

todas as interfaces e aplicações de utilizador deverão respeitar;

- a camada vertical (*Query:SPARQL*) – é uma camada de consultas que utiliza a linguagem SPARQL (Prud'hommeaux and Seaborne, 2008). A SPARQL é uma linguagem de consultas RDF, que possibilita a realização de consultas sobre documentos OWL e RDFS;
- a camada vertical (*Crypto*) – esta camada irá ser usada para identificar a identidade dos recursos através e assinaturas digitais. A cifragem da *Semantic Web* para questões de segurança estará a cargo do *XML Encryption* (Reagle, 2008), da responsabilidade do *XML Encryption Working Group*.

3.2.1 Camada básica de dados

A norma *Unicode* (Unicode, 2008) permite que máquinas representem e manipulem texto de forma consistente em múltiplas plataformas. A Unicode fornece um único número para cada carácter independente da plataforma, o que possibilita que software, textos ou um apenas um único carácter seja transportado para outros sistemas sem ser corrompido.

Um URI é uma *string* formatada com o significado de identificar recursos físicos ou lógicos. Os URIs são definidos em esquemas, que por sua vez definem uma sintaxe especifica e protocolos associados. Um URI pode ser classificado como um localizador ou nome ou os dois. Um *Uniform Resource Locator* – URL (IETF, 1994) é um URI que identifica recursos via representação do seu mecanismo de acesso primário, permitindo um modo de actuar ou obter uma representação do recurso através da sua localização. Um *Uniform Resource Names* – URN (IETF, 1997) é um URI que identifica um recurso pelo nome num *namespace* (espaço de nomes) particular. Um URN pode ser utilizado para identificar um recurso sem utilizar a sua localização ou sem utilizar o endereço para os dados.

3.2.2 Camada de descrição de dados

A linguagem XML é uma recomendação W3C como linguagem de marcação (*markup*) para descrever dados estruturados nas mais variadas aplicações independente de qualquer plataforma. A XML é razoavelmente legível aos humanos e tem como principais funções facilitar a partilha de dados entre diferentes sistemas de informação, particularmente sistemas conectados à Internet.

O Código 3.1 é o exemplo de um excerto de um documento XML que identifica um contacto de uma pessoa. O documento inclui diversas *tags* de marcação com <nome>, <email> e <organização> que descrevem detalhes sobre o recurso "Contacto".

```
1: <Contacto contactoId="1">
2:   <nome> João Paulo </nome>
3:   <apelido> Sousa </apelido >
4:   <email> jpaulo@ipb.pt </email>
5:   <organizacao> Instituto Politécnico de Bragança </organizacao>
6:   <telefone> +351 273331570 </telefone>
7: </Contacto>
```

Código 3.1 – Representação de um recurso "Contacto" em XML.

A XML tem sido bastante utilizada como formato normalizado para a suportar a interoperabilidade entre sistema através da troca de dados estruturados, como é o caso da especificação *XML Metadata Interchange* – XMI (OMG, 2005) que utiliza a XML na especificação de uma norma para o intercâmbio de metadados entre sistemas de modelação de dados.

Embora a XML seja mundialmente adoptada como uma maneira simples de normalizar formatos de dados, no entanto do ponto de vista de interoperabilidade semântica a XML evidencia limitações, não permitindo reconhecer semânticas sobre domínios particulares. Além disso, a XML tem um modelo de dados incapaz de capturar semânticas, relacionamentos ou restrições. A XML é portanto uma norma simples para a formatação de dados, que faz parte de um conjunto de tecnologias que constituem as fundações da *Semantic Web*.

3.2.3 Camada de descrição estrutural e semântica

A camada de descrição estrutural e semântica, como já foi referido, é composta pelas linguagens RDF e RDFS, descritas a seguir.

3.2.3.1 *Resource Description Framework*

A RDF é uma linguagem baseada em XML que é utilizada para descrever recursos, principalmente recursos na *Web*. Cada recurso é identificado através de um URI, por forma a poder garantir que um recurso esteja apenas relacionado com um conceito. Em adição à sua forma semântica, a RDF possui uma estrutura de dados simples que é modelada utilizando grafos. A RDF é dividida em três componentes: o modelo de dados, a sintaxe para a troca de

dados e a linguagem para descrição de esquemas.

O modelo de dados RDF é construído com descrições sobre recursos. Uma descrição RDF é baseada em três tipos de objectos diferentes e que constituem a base do modelo de dados:

- recurso (sujeito) – um recurso é um dado ou fonte de informação que se pretende descrever em RDF. Pode ser um recurso físico ou lógico e é sempre identificado por um URI. Um livro, uma página *Web* ou uma pessoa são exemplos de recursos;
- propriedade (predicado) – a propriedade é uma característica ou relação utilizada para descrever um recurso. Por exemplo um "livro" pode ser reconhecido pelo "título" enquanto que uma "pessoa" pode ser reconhecida pelo seu "nome" Ambos são atributos para o reconhecimento dos recursos "livro" e "pessoa";
- valor (objecto) – uma propriedade tem que ter um valor, por exemplo uma sequência de caracteres. Seguindo o exemplo anterior o "João Paulo" seria o valor da propriedade "Nome" do recurso "Pessoa".

Cada declaração em RDF é designada por uma tripla e é formada por um recurso, uma propriedade e um valor para a propriedade desse recurso. Por exemplo, na declaração "A página *Web* http://www.ipb.pt/~jpaulo tem como autor João Paulo", o recurso corresponde a "http://www.ipb.pt/~jpaulo", a propriedade corresponde a "autor" e o valor seria "João Paulo". Esta declaração pode ser representada em forma de grafo como ilustra a Figura 3.5. Num grafo de um modelo de dados RDF um recurso é representado por uma elipse identificada por um URI, o valor da propriedade por um rectângulo e a propriedade por um arco que liga o recurso ao valor da propriedade. As propriedades são também identificadas por um URI que identifica o espaço de nome no qual foram definidas.

A RDF tem um conjunto de construtores sintácticos muito limitado em que não são permitidos outros construtores para criar as triplas. Cada documento RDF é constituído por um conjunto de triplas.

Figura 3.5 - Representação gráfica de um grafo referente a um modelo de dados básico.

O modelo sintáctico para a troca de dados mais utilizado na RDF é a sintaxe XML embora existam outras tais como a N3 (Berners-Lee, 1998) e a N-Triples (Grant and Beckett, 2004).

A recomendação RDF que define a sintaxe XML para RDF é descrita no documento RDF/XML Syntax (Beckett and McBride, 2004).

O Código 3.2 é o resultado da aplicação da sintaxe RDF/XML ao grafo da Figura 3.5. O prefixo *my* é utilizado como sinónimo para o URI do espaço de nomes XML no qual a propriedade "autor" foi definida. O prefixo *rdf:* (linha 2) referencia o espaço de nomes XML no qual o modelo da linguagem RDF é descrita. Os espaços de nomes (*namespaces*) permitem a criação de identificadores unívocos globais para os elementos da linguagem de marcação. Os espaços de nomes desempenham duas funções importantes: permitem evitar conflitos de significados de nomes idênticos em diferentes linguagens de marcação e permitem que diferentes linguagens de marcação possam ser misturadas sem ambiguidades. As linguagens de marcação mais utilizadas, como é o caso do XML Schema, suportam totalmente *namespaces*.

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:          xmlns:my="http://example.org.org/schema/">
4:  <rdf:Description rdf:about="http://www.ipb.pt/~jpaulo">
5:   <my:autor>João Paulo</my:autor>
6:  </rdf:Description>
7: </rdf:RDF>
```

Código 3.2 - Representação do modelo de dados básico utilizando a sintaxe RDF/XML.

Se tivermos uma colecção de declarações RDF representadas como um grafo de vários nós interligados, essas ligações (propriedades) entre eles representam relacionamentos. Por exemplo, se tivermos várias pessoas em que cada uma delas representa um nó, cada pessoa pode ter relações com outras pessoas, tais como: parentesco, amizade, profissionais, amorosas. Cada uma das relações é identificada com o nome do relacionamento. Os nós podem ter ainda outras propriedades, como nome, data de nascimento ou estado civil. Além de ser possível descrever todos estes relacionamentos, é também possível em qualquer momento introduzir um novo nó e relacioná-lo apropriadamente com os outros nós.

Embora a RDF seja utilizada para descrever recursos através de declarações, permite ainda transformar uma declaração num recurso e usar esse novo recurso como um sujeito ou objecto em outras declarações. Isto é, permite que uma declaração RDF possa ser descrita por outra declaração RDF. Esta propriedade é útil, por exemplo, para um grupo de utilizadores debater uma declaração.

A RDF suporta diversas necessidades de metadados necessários para os diferentes fornecedores de informação. Para facilitar a descrição de recursos e a interoperabilidade entre

estes agentes, a RDF faz uso das normas de metadados que contêm elementos descritores para localizar, caracterizar e relacionar recursos. Devido à diversidade de recursos acabaram por surgir várias normas de metadados para diferentes tipos de recursos. Alguns dos mais conhecidos são: o *vCard* (IETF, 1998) para troca de informações pessoais; *iCalendar* (IETF, 1998) para informações de agendamento de eventos; *Dublin Core* (IETF, 2008) usado para descrever recursos, embora seja mais utilizado para descrever recursos digitais, e que será explicado mais à frente. Poderá encontrar-se mais informações sobre normas de metadados em (Brand, Daly et al., 2003). A Figura 3.6 ilustra o grafo do Código 3.4 que faz uso da norma de metadados *Dublin Core* descrito brevemente de seguida.

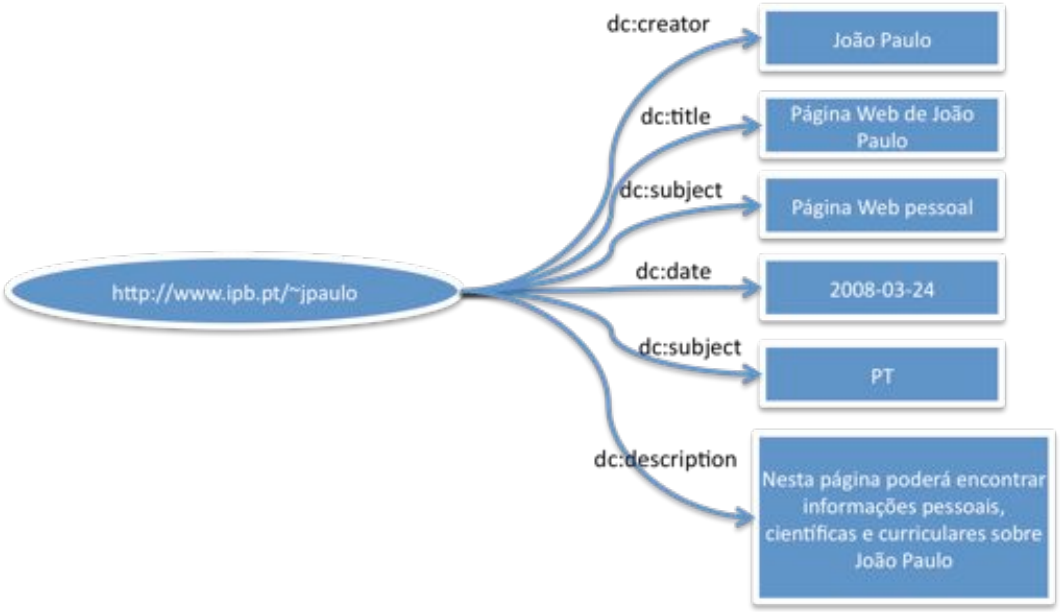


Figura 3.6 – Representação gráfica de um grafo referente a uma página *Web*.

A utilização do vocabulário Dublin Core é feita usualmente utilizando o prefixo dc: que aponta para o URI e *namespace* do *Dublin Core*.

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:             xmlns:dc="http://purl.org/dc/elements/1.1/">
4:   <rdf:Description rdf:about="http://www.ipb.pt/~jpaulo">
5:     <dc:creator>João Paulo</dc:creator>
6:     <dc:title>Página Web de João Paulo</dc:title>
7:     <dc:subject>Página Web pessoal</dc:subject>
8:     <dc:date>2008-03-24</dc:date>
9:     <dc:language>PT</dc:language>
10:    <dc:description>Nesta página poderá encontrar informações pessoais,
científicas e curriculares sobre João Paulo</dc:description>
11:  </rdf:Description>
12:</rdf:RDF>
```

Código 3.3 – Código RDF/XML do grafo da Figura 3.6, utilizando a norma de metadados *Dublin Core*.

O *Dublin Core* (DCMI, 2006) é um projecto cujo objectivo é desenvolver normas de metadados que suportem uma ampla gama de propósitos e modelos de negócios. Este projecto mantém um registo de metadados que podem ser consultados ou usados. Estes metadados constituem vocabulários para descrever recursos, permitindo desenvolver sistemas de descoberta e troca de informação mais avançados. O *Dublin Core Metadata Element Set* (IETF, 2008) utiliza quinze elementos para descrever quaisquer recurso de informação, como se pode ver resumidamente na Tabela 3.1.

Tabela 3.1 – Elementos principais do *Dublin Core Metadata Element Set* (IETF, 2008).

Nome	Descrição resumida
Title	Título ou nome do recurso
Creator	Criador do recurso
Subject	Conjunto de palavras que caracterizam o recurso
Description	Descrição do recurso
Publisher	Responsável por editar ou publicar o recurso
Contributor	Responsável por contribuir para o recurso
Type	Natureza ou género do recurso
Date	Data ou período de tempo relacionado com um evento no tempo de vida do recurso
Format	Formato, meio físico ou dimensões do recurso
Identifier	Uma referência unívoca do recurso num determinado contexto
Source	Referência a uma fonte da qual o recurso actual deriva
Language	Língua do conteúdo recurso
Relation	Referencia a um recurso referenciado
Rights	Informação sobre direitos de propriedade sobre o recurso

A utilização do *Dublin Core* tem vindo a crescer na interoperabilidade entre sistemas, pois permite que termos práticos normalizados sejam aplicados quer a novos trabalhos quer a trabalhos já existentes (ex: *Title*, *Creator* e *Subject*). O *Dublin Core* pode ser expresso de diversas formas sendo a RDF uma delas.

3.2.3.2 RDF Schema

A RDF Schema (Brickley and Guha, 2004) é uma linguagem de descrição de vocabulário RDF, funcionando como uma extensão da RDF. Esta linguagem fornece mecanismos para descrever grupos de recursos relacionados e os relacionamentos entre esses recursos. Para isso a RDFS define classes e propriedades que podem ser usadas para descrever classes, propriedades e outros recursos. As descrições RDF são escritas em RDF usando os termos

descritos na Tabela 3.2 e na Tabela 3.3. Esses recursos são usados para determinar as características de outros recursos, como é o caso do domínios (domains) e intervalos (ranges) das propriedades.

A Tabela 3.2 apresenta as classes RDF e RDFS mais utilizadas, enquanto que a Tabela 3.3 apresenta as propriedades. Os prefixos rdf e rdfs em cada uma das tabelas correspondem ao espaço de nomes http://www.w3.org/1999/02/22-rdf-syntax-ns# e http://www.w3.org/2000/01/rdf-schema respectivamente.

Tabela 3.2 – Classes RDF e RDFS.

Classe	Descrição resumida
rdfs:Resource	Classe de todas as classes
rdfs:Literal	Valores literais como *strings* e inteiros
rdfs:Class	Recursos que têm como classe *rdfs:Class*
rdfs:Datatype	Classe de todos os *datatype*
rdfs:Statement	Classes de todas as declarações RDF
rdfs:XMLLiteral	Classe à qual pertence os valores
rdfs:Container	Um conjunto de recipientes (*containers*)
rdf:Seq	Uma colecção ordenada
rdf:Property	Classe de todas as propriedades
rdf:Bag	Uma colecção desordenada
rdf:Alt	Uma colecção de alternativas

O conceito de classe RDF é similar ao de classe em programação orientada aos objectos, como o Java ou C++. Uma classe é uma estrutura de coisas similares em que a herança é permitida. Esta propriedade possibilita organizar classes de uma forma hierárquica e definir recursos como instâncias de classes e subclasses de outras classes.

Tabela 3.3 – Propriedades RDF e RDFS.

Propriedade	Descrição resumida	domínio	intervalo
rdf:type	O sujeito é uma instância da classe	rdfs:Resource	rdfs:Class
rdfs:subClassof	Define que uma classe é subclasse de outra	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	Define que uma propriedade é subpropriedade de outra	rdf:Property	rdf:Property
rdfs:domain	Classes de domínio para uma propriedade	rdf:Property	rdfs:Class
rdfs:range	Intervalo de classes para uma propriedade	rdf:Property	rdfs:Class
rdfs:label	Versão interpretável por humanos do nome de um recurso	rdfs:Resource	rdfs:Literal
rdfs:comment	Utilizado para descrições	rdfs:Resource	rdfs:Literal
rdfs:member	Membro de um recipiente (*container*)	rdfs:Resource	rdfs:Resource
rdfs:seeAlso	Um recurso que fornece mais informações sobre outro recurso	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	Usado para indicar o vocabulário RDF no qual um recurso é descrito	rdfs:Resource	rdfs:Resource
rdf:first	O primeiro elemento de uma lista RDF	rdf:List	rdfs:Resource
rdf:value	Usado para descrever valores estruturados	rdfs:Resource	rdfs:Resource
rdf:subject	Usado para representar o sujeito de uma declaração	rdf:Statement	rdfs:Resource
rdf:rest	O resto de uma lista RDF após o primeiro item	rdf:List	rdf:List
rdf:predicate	Usado para representar o predicado de uma declaração	rdf:Statement	rdfs:Resource
rdf:object	Usado para representar o objecto duma declaração	rdf:Statement	rdfs:Resource

Como a RDF permite exprimir uma declaração sobre um recurso qualquer, e que tudo que tenha um URI pode ser um recurso, então, se precisarmos de dizer que a "Página Web" é uma subclasse de *rdfs:Resource*, ou uma subclasse de Documentos, isso é descrito da seguinte forma: (*rdfs:PaginaWeb rdfs:subClassOf rdf:resource=Documentos*). O prefixo *rdfs:* corresponde ao espaço de nomes XML no qual a linguagem de descrição de esquemas RDF foi definida (http://www.w3.org/2000/01/rdf-*schema*#).

Outro exemplo de aplicação utilizando a RDFS é mostrado na Figura 3.7, que descreve a classe *Veículo* que possui duas subclasses *Automóvel* e *Autocarro*. É também definido que o recurso *Marca* possui um domínio *Veículo* e um valor literal que neste caso será uma *string*.

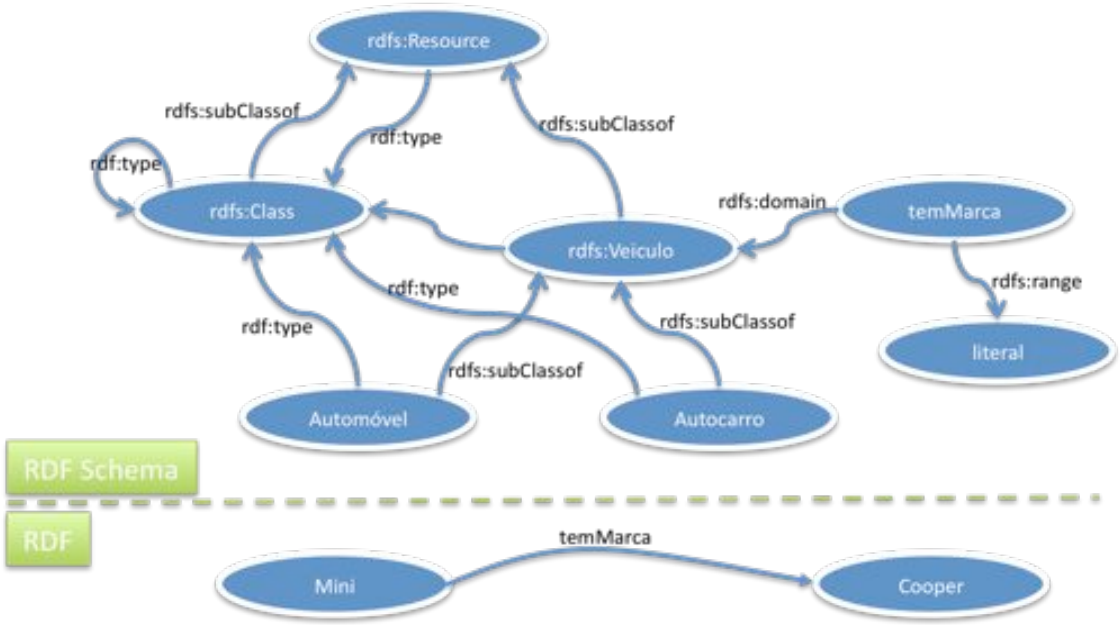


Figura 3.7 – Exemplo de aplicação RDF e RDFS.

A codificação deste exemplo na sintaxe RDFS está representada em Código 3.4.

```
1: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22/rdf-syntax-ns#"
2:          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
3:  <rdf:Description ID="Veiculo">
4:   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
5:   <rdfs:subClassOf resource="http://www.w3.org/TR/rdf-schema#Resource"/>
6:  </rdf:Description>
7:  <rdf:Description ID="Automovel">
8:   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
9:   <rdfs:subClassOf resource="#Curso"/>
10: </rdf:Description>
11: <rdf:Description ID="Autocarro">
12:  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
13:  <rdfs:subClassOf resource="#Curso"/>
14: </rdf:Description>
15: <rdf:Property rdf:ID="temMarca">
16:  <rdfs:domain rdf:resource="#Curso"/>
17:  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
18: </rdf:Property>
19:</rdf>
```

Código 3.4 - Codificação do grafo da Figura 3.7 em RDF.

3.2.3.3 Limitações da expressividade da camada estrutural e semântica

Embora a RDF e a RDFS forneçam a possibilidade de representar conhecimento ontológico, esta representação é no entanto bastante limitada e um conjunto de características significativas não estão disponíveis. A lista seguinte apresenta algumas dessas características que não é possível encontrar numa representação ontológica usando apenas a RDF e a RDFS:

- âmbito local das propriedades (*Local Scope of Properties*): o rdfs:range define o

alcance de uma propriedade, por exemplo *temPai* para todas as classes. Com a RDFS não podemos declarar intervalo de restrições que apenas se aplicam a algumas classes. Por exemplo, filho de mamífero é mamífero enquanto filho de peixe é peixe;

- classes disjuntas – com a RDFS apenas existe a relação de subclasse, por exemplo pessoa é subclasse de animal. Não é possível específicar que masculino seja uma classe disjunta de feminino;
- combinações booleanas de classes – com esta propriedade é possível definir novas classes através da combinações de classes, por exemplo a classe pessoa pode ser uma união disjunta das classes masculino e feminino (i.e. união de classes que não tem qualquer elemento em comum);
- restrições de cardinalidade – permite definir o número de valores que uma propriedade deve ou deverá ter. Por exemplo uma pessoa tem exactamente dois pais;
- características especiais das propriedades – permite atribuir características especiais às propriedades, como por exemplo a propriedade transitiva (ex. *subRegião*), univocidade (ex. *temPai*) ou inverso (ex. *temFilho* e *temPai*).

Resumindo, a RDFS é uma extensão semântica da RDF que define o respectivo vocabulário permitindo descrever os recursos RDF e os seus significados. As estruturas oferecidas pela RDFS preocupam-se com a simplicidade e por isso contêm apenas mecanismo básicos para descrever classes de objectos, propriedades e relacionamento entre classes. Isto faz com que mesmo utilizando a RDF e a RDFS não seja possível fazer representações com semântica suficiente para representar ontologias. Mais detalhes sobre RDF e RDFS podem ser encontrados em (Grant and Beckett, 2004; Hayes, 2004; Klyne and Carroll, 2004).

3.2.4 Camada de descrição semântica e lógica

Embora como já foi referido na secção anterior, a camada estrutural e semântica permita representar conhecimento é no entanto reduzida. Isto acontece porque o conjunto de elementos para modelação da RDFS é limitado e poucos desses permitem a inferência de novos factos. Esta camada tem como objectivo estender o vocabulário da camada de descrição de dados com a inclusão de novos elementos, com maior poder de expressividade e inferência

na arquitectura da *Semantic Web*.

3.2.4.1 Camada de descrição semântica

Embora a camada de descrição semântica seja composta pelas linguagens OWL e RIF, apenas a primeira é utilizada para descrever a semântica dos dados, sendo a segunda mais usada para a troca de regras. A linguagem OWL é uma extensão semântica da linguagem RDF e RDFS e derivou da linguagem ontológica DAML+OIL (Connolly, Harmelen et al., 2001). Ao recomendar a OWL, o W3C criou uma linguagem para normalizar uma *framework* de línguas ontológicas mais capaz do que aquela constituída apenas pelas RDF e RDFS. A OWL introduziu restrições relacionadas com a estrutura e os conteúdos dos documentos RDF de modo a tornar o processamento e a inferência ou raciocínio (*reasoning*) mais resolúvel computacionalmente. O W3C define a OWL como uma linguagem para ser usada por aplicações que necessitem de processar o conteúdo de uma informação, não ficando assim restringida apenas à consulta humana. Através do fornecimento de vocabulário adicional combinado com uma semântica formal, a OWL vem facilitar a interpretação de conteúdo da *Web*, antes não existente ou muito limitado pelas linguagens XML, RDF e RDFS (McGuinness and Harmelen, 2004).

As características principais do desenvolvimento da OWL, são: independente da tecnologia, escalável, compatível com as normas *Web* já existentes, ser aberta e extensível.

A OWL é dividido em três subclasses a OWL Full, OWL DL (*Description Logic*) e a OWL Lite. Cada uma destas classes fornece um capacidade de expressividade diferente que pode ser usada dependendo das necessidades de representação e inferências necessárias para a ontologia:

- OWL Full: esta linguagem utiliza todas as primitivas da OW e permite a combinação destas primitivas com a RDF e RDFS. Possibilita a alteração do significado de primitivas RDF e OWL já predefinidas. A sua grande vantagem é ser totalmente compatível com a RDF sintacticamente e semanticamente. Tem como desvantagem oferecer um nível máximo de expressividade que se poderá tornar indecidível, perdendo a aptidão de suportar a capacidade de inferência eficiente ou mesmo a total capacidade;
- OWL DL – é uma sublinguagem da OWL Full que restringe o modo como os construtores da OWL e RDF podem ser usados. Permite o formalismo de

representação de conhecimento baseado na lógica de descrição. Ao contrário da OWL Full, não permite a aplicação de construtores OWL a eles mesmos, garantindo uma descrição lógica bem estruturada. Tem a vantagem de apresentar um nível máximo de expressividade sem perder a capacidade de processar todas as associações que suporta (todas as associações são computáveis) e a capacidade de solução finita (garante que todas as computações serão finalizadas em tempo finito). Como desvantagem, perde a compatibilidade total com documentos em RDF, para que exista, um documento RDF poderá ter que ser alterado;

- OWL Lite – corresponde ao OWL DL com mais restrições (p. ex. são restringidas as classe enumeradas e as declarações disjuntas). O objectivo é facilitar a iniciação à linguagem por parte dos utilizadores e o desenvolvimento de processadores por parte dos investigadores, evoluindo depois para as outras classes. A desvantagem é claramente as limitações de expressividade.

A escolha da classe da linguagem OWL deve ter em consideração qual delas se adapta melhor às necessidades. A escolha entre a OWL DL e a OWL Lite está normalmente na capacidade de expressividade oferecida por cada uma delas. Já a escolha entre a OWL Full e a OWL DL depende principalmente no nível de facilidades de meta-modelação da RDFS. Ao nível do raciocínio, o suporte à OWL Full é menos previsível porque algumas vezes é impossível terminar implementações em OWL Full.

A OWL é baseado na linguagem de descrição lógica SHOIN(D+)[3] que tem vários motores de inferência que podem ser usados para verificação de restrição de conceitos, propriedades e instâncias, e para classificação automática de conceitos em hierarquias.

Sintaxe da OWL

A OWL é construída em cima da RDF e da RDFS e usa uma sintaxe baseada em RDF/XML. No entanto como esta sintaxe não é de compreensão fácil, a OWL pode ainda ser definida por:

- uma sintaxe baseada em XML mas que não segue as convenções RDF, de modo a tornar-se mais compreensível pelo humanos. Pode ser encontrado este tipo de

[3] Informações detalhadas sobre este e outros tipos de expressividade de descrição lógica podem ser consultadas em Zolin, E. (2009, 09/2009). "Complexity of reasoning in Description Logics." Retrieved 09/2009, from http://www.cs.man.ac.uk/~ezolin/dl/.

sintaxe em (Hori, Euzenat et al., 2003);

- uma sintaxe abstracta mais compacta e compreensível, usada como linguagem de especificação em (Patel-Schneider, Hayes et al., 2004);
- uma sintaxe gráfica baseada em convenções UML, usada para descrição entre humanos.

A OWL usa classes e propriedades RDFS e adiciona outras novas. A Tabela 3.4 contém todas as classes e propriedades da OWL, e quando comparada com as Tabela 3.2 e Tabela 3.3 verifica-se que é mais rica semanticamente. Os nomes sublinhados indicam que apenas são definidos para a OWL DL e OWL Full, enquanto que os nome a itálico são usados na OWL Lite, mas com restrições. Como os nomes das classes e propriedade da OWL traduzem do seu propósito, optou-se por não colocar a suas descrição na Tabela 3.4.

Tabela 3.4 – Classes e propriedades da OWL.

Classes	Propriedades	
owl:Class	owl:allValuesFrom	owl:inverseOf
owl:Ontology	owl:backwardCompatibleWith	owl:maxCardinality
owl:DatatypeProperty	owl:cardinality	owl:minCardinality
owl:ObjectProperty	owl:complementOf	owl:oneOf
owl:Restriction	owl:differentFrom	owl:onProperty
owl:AllDiferent	owl:disjointWith	owl:priorVersion
owl:DataRange	owl:distinctMembers	owl:sameAs
owl:DeprecatedClass	owl:equivalentClass	owl:sameIndividualAs
owl:DeprecatedProperty	owl:equivalentProperty	owl:someValuesFrom
owl:FuncionalProperty	owl:hasValue	owl:subClassOf
owl:InverseFunctionalProperty	owl:imports	owl:TransitiveProperty
owl:Nothing	owl:incompatibleWith	owl:unionOf
owl:SymmetricProperty	owl:intersectionOf	owl:versionInfo

A OWL altera a definição de classe, mas basicamente é feita da mesma forma que na RDFS através da declaração `owl:Class`. Na OWL DL e na OWL Lite uma classe é um esquema de classificação associado a um conjunto de indivíduos ou instâncias, em que instâncias de uma classe têm que ser indivíduos e não outras classes como pode acontecer na OWL Full.

Os documentos OWL são normalmente chamados ontologias OWL e são documentos formatados em RDF/XML. Para que os termos utilizados nas ontologias OWL não sejam ambíguos a primeira secção de um documento OWL é a indicação dos vocabulários através

do *namespace* que vão ser usados. Como mostra o Código 3.5, o conjunto de *namespaces* XML estão contidos dentro da *tag* de início `rdf:RDF`. Esta declaração identifica os *namespaces* associados com a ontologia *person*. Por exemplo a linha 3 utiliza um *namespace* normalizado para a ontologia actual e a linha 4 identifica o URI para a base do documento. A linha 5 identifica o *namespace* da ontologia com o prefixo `per:`. As linha 6 e 7 identificam um *namespace* para o vocabulário para a OWL e RDFS. Como uma ontologia é ela própria um recurso, pode ser descrita utilizando as propriedades OWL e *namespaces* XML. Estas informações devem ser agrupadas dentro da *tag* `owl:Ontology` como acontece nas linhas de 9 a 13, em que é definida a versão da ontologia, um comentário indicando informações sobre a ontologia e uma referência a outra ontologia OWL que contém definições e que irão ser usadas como parte do significado da ontologia actual. Na linha 12 a propriedade `owl:imports` ordena a importação de uma ontologia de modo a ser possível utilizar o seu vocabulário. Esta propriedade tem a característica de ser transitiva, isto é, se a ontologia A importa B e por sua vez B importa C então A importa B e C.

```
1: <rdf:RDF
2:      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:      xmlns="http://www.my-ontologias.com/person.owl#"
4:      xml:base="http://www.my-ontologias.com/person.owl#">
5:      xml:per=" http://www.my-ontologias.com/person.owl#">
6:      xmlns:owl="http://www.w3.org/2002/07/owl#"
7:      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8:
9:   <owl:Ontology rdf:about="">
10:   <owl:versionInfo>v 1.10 2009/08/26 13:16:51</owl:versionInfo>
11:   <rdfs:comment>Ontology Person</rdfs:comment>
12:   <owl:imports rdf:resource="http://www.example.org/foo"/>
13: </owl:Ontology>
```

Código 3.5 – Cabeçalho do documento define a ontologia *person*.

Classes e indivíduos

Na linguagem OWL, tudo o que pode ser definido é uma subclasse de `owl:Thing`, enquanto que `owl:Nothing` é exactamente o oposto, é uma classe sem membros. As classes fornecem um mecanismo de abstracção para agrupar recursos com características semelhantes. Cada classe está associada a um conjunto de indivíduos, chamado extensão da classe. Os indivíduos ou instâncias de um extensão da classe designam-se instâncias das classes (Bechhofer, Harmelen et al., 2004). As linhas 14, 21 e 26 do Código 3.6 (continuação da definição da ontologia *person*), mostram a definição da classe `Male`, `Female` e `Person` respectivamente. A sintaxe `rdf:ID="Male"` é usada para atribuir um nome à classe. As linhas 18, 22 e 27 mostram um exemplo de hierarquia de classes usando o elemento

`owl:subClass`. Os indivíduos que são instâncias de classes são definidos com factos. Um facto é uma declaração que é sempre verdadeira num determinado domínio. As linhas 36 a 39 mostram a criação de um indivíduo que é uma instância da classe `Person` com o nome Maria e com duas propriedades que serão discutidas a seguir. A linha 15 utiliza um construtor adicional que é usado na criação de classes chamado *class expressions*. Neste caso é feita uma afirmação indicando que as classes `Male` e `Female` são disjuntas, isto é, elas não tem indivíduos comuns.

```
14: <owl:Class rdf:ID="Male">
15:     <owl:disjointWith>
16:       <owl:Class rdf:about="#Female"/>
17:     </owl:disjointWith>
18:     <rdfs:subClassOf rdf:resource="#Gender"/>
19:  </owl:Class>
20:
21:  <owl:Class rdf:ID="Female">
22:     <:subClassOf rdf:resource="#Gender"/>
23:     <owl:disjointWith rdf:resource="#Male"/>
24:  </owl:Class>
25:
26:  <owl:Class rdf:ID="Person">
27:     <rdfs:subClassOf>
28:         <owl:Restriction>
29:            <owl:onProperty rdf:resource="#hasGender"/>
30:            <owl:someValuesFrom rdf:resource="#Gender"/>
31:         </owl:Restriction>
32:     </rdfs:subClassOf>
33:    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
34:  </owl:Class>
35:
36:  <Person rdf:ID="Maria">
37:     <hasName rdf:datatype="&xsd;string">Maria</hasName>
38:     <hasGender rdf:resource="#female"/>
39:  </Person>
```

Código 3.6 – Continuação da ontologia `person` – Classes e instâncias.

Propriedades

As propriedades são relações binárias que podem ser usadas para criar relacionamentos entre indivíduos ou entre indivíduos e dados. Estes relacionamentos permitem descrever factos quer sobre os membros das classes quer sobre os indivíduos.

Na OWL existem dois tipos de propriedades:

- propriedade de objectos (*object properties*) – relacionam objectos com outros objectos. Exemplo as linha 40 a 42 do Código 3.7 declaram o *ObjectProperty* `hasGender`. Recorrendo ao RDF e RDFS é ainda declarada como uma propriedade funcional indicando que apenas poderá ter uma única instância e que

terá como domínio `Person` e contra-domínio `Gender`. Isto é, é uma definição parcial para esta ontologia, que uma pessoa terá apenas um sexo que será feminino ou masculino;

- propriedade de tipos de dados (*datatype properties*) – relacionam objectos com tipos de dados literais. Por exemplo a linha 44 define uma *DataTypeProperty* chamada `hasName` como sendo uma propriedade de tipos de dados funcional e que tem como domínio `Person` e o contra-domínio será uma *string* definida por (`&xsd;string`). A OWL não tem tipo de dados predefinidos, em vez disso utiliza o tipo de dados do XML Schema, fazendo assim uso da arquitectura por camadas da *Semantic Web*.

Nas linhas 36 a 39 são declaradas propriedades sobre os indivíduos. Estas propriedades são chamadas factos sobre os indivíduos. Neste caso o indivíduo Maria tem nome Maria e tem sexo feminino.

```
40:    <owl:ObjectProperty rdf:ID="hasGender">
41:     <rdf:type rdf:resource="&owl;FunctionalProperty"/>
42:     <rdfs:domain rdf:resource="#Person"/>
43:     <rdfs:range rdf:resource="#Gender"/>
44: </owl:ObjectProperty>
45:
46: <owl:DatatypeProperty rdf:ID="hasName">
47:     <rdf:type rdf:resource="&owl;FunctionalProperty"/>
48:     <rdfs:domain rdf:resource="#Person"/>
49:     <rdfs:range rdf:resource="&xsd;string"/>
50: </owl:DatatypeProperty>
```

Código 3.7 - Continuação da ontologia *person – DatatypesProperty* e *ObjectProperty*.

A OWL permite especificar que as propriedades podem ter características. Para isso suporta as seguintes propriedades: `owl:TransitiveProperty`, `owl:SymmetricProperty`, `owl:FunctionalProperty`, `owl:InverserFunctionalProperty` e `owl:inverseOf`. Por exemplo na linha 41 é aplicada a propriedade `FunctionalProperty` à propriedade `hasGender`, indicando que esta apenas poderá ter um único valor para cada indivíduo, neste caso um sexo.

A OWL permite também especificar restrições sobre propriedades. As restrições que a OWL fornece podem ser de valores (`owl:hasValue`, `owl:someValuesFrom`, `owl:AllValuesFrom`) ou de cardinalidade (`owl:minCardinality`, `owl:maxCardinality` e `owl:Cardinality`). As linhas 29 e 30 definem a propriedade `hasGender` que terá pelo menos um indivíduo do tipo `Gender`, neste caso uma pessoa terá

que ter sexo feminino ou masculino.

O desenvolvimento da OWL beneficiou de várias gerações prévias de linguagens ontológicas com bases teóricas maduras e a participação de uma comunidade empenhada em desenvolver uma linguagem adequada para a *Semantic Web*. Faz actualmente parte da lista de recomendações da W3C para o desenvolvimento da *Semantic Web* e posiciona-se para se tornar a linguagem ontológica mais utilizada na *Web*. Na altura de escrita desta Tese, está em desenvolvimento uma segunda versão da OWL e encontra-se no estado de candidata a recomendação (Motik, Grau et al., 2009), com previsão da sua conclusão em finais de 2009. A OWL 2 (Motik, Grau et al., 2009) é uma extensão da OWL compatível com a versão anterior e que adiciona um novo conjunto de funcionalidades motivadas pelas opiniões dos utilizadores que usam a versão original.

3.2.4.2 Camada lógica

A parte lógica desta camada fornece suporte para a descrição de regras por forma a expressar relações sobre conceitos de uma ontologia, as quais não é possível expressar por meio de construtores de uma linguagem ontológica como a OWL. Assim aparecem com destaque duas linguagens para a descrição de regras na *Semantic Web*: a *Rule Markup Language* (*RuleML*) e a *Semantic Web Rule Language* (SWRL). Neste sentido, o W3C decidiu criar um grupo de actividade chamado *RIF Working Group*, com o propósito de desenvolver normas *Web* para fornecer suporte ao intercâmbio das diversas tecnologias baseadas em regras. As três tecnologias serão descritas brevemente em seguida:

- a *RuleML* – definido pela *Rule Markup Initiative* (RMI, 2009), uma rede aberta de indivíduos e grupos dos sectores académico e industrial. A *RuleML* é uma linguagem de *markup* para a publicação e partilha de regras baseadas na WWW e constrói uma hierarquia de sublinguagens em cima de XML, RDF, XSLT e OWL;
- a *Semantic Web Rule Language* – SWRL (Horrocks, Patel-Schneider et al., 2004) – foi submetida ao W3C como uma recomendação para a criação de linguagens de regras na *Semantic Web*. É uma combinação das linguagens OWL DL e OWL Lite com as sublinguagens da RuleML. Permite a criação de factos e regras sobre indivíduos de ontologias OWL, de modo a obter novos conhecimentos;
- a *Rule Interchange Format* – consiste numa *framework* extensível para linguagem baseadas em regras, chamadas dialectos RIF, que incluem especificações precisas

e formais da sintaxe, semântica e serialização XML. Esta troca de regras através do RIF irá permitir que diferentes sistemas sejam capazes de mapear as suas linguagens ou parte delas em dialectos RIF preservando a sua semântica. Será também possível transferir esse conjunto de regras e dados entre sistemas, suportando a função de que os sistemas possam acordar o dialectos adequados que cada um deles suporte. A linguagem RIF utiliza o XML como formato para a especificação das regras.

O *RIF Working Group* publicou oito *working drafts*, em que três deles definem formatos XML com semânticas formais para o armazenamento e transmissão de regras: o *RIF Prodution Rule Dialect* – RIF-PRD (Marie, Paschke et al., 2009), o *RIF Basic Logic Dialect* – RIF-BLD (Boley and Kifer, 2009) e o *RIF Core Dialect* (Boley, Hallmark et al., 2009).

3.2.5 SPARQL Protocol and RDF Query Language

Após a aceitação da RDF por parte do W3C, como método genérico para a descrição conceptual ou modelação de informação que é implementada em recursos *Web*, logo se colocou o problema natural de se criar uma linguagem para pesquisas em dados RDF. Várias linguagens foram propostas desde então, como por exemplo: a linguagem RQL (Karvounarakis, Alexaki et al., 2002), SeRQL (Karvounarakis, Alexaki et al., 2002), RDQL (Seaborne, 2004), entre outras. Em (Haase, Broekstra et al., 2004) os autores fazem uma breve descrição e comparação destas e outras linguagens de pesquisas para RDF. Em 2004 o *Data Access Working Group* publicou a primeira versão do esboço de uma linguagem para pesquisas em dados RDF, chamada SPARQL (Prud'hommeaux and Seaborne, 2005), que é actualmente é uma recomendação do W3C (Prud'hommeaux and Seaborne, 2008).

A SPARQL é uma linguagem para executar *queries* em diferentes tipos de fontes de dados, quer eles estejam armazenados no formato nativo RDF, quer possam ser vistos como dados RDF através de um qualquer *middleware*. Fornece ainda facilidades para extrair informação em forma de URIs, nós do tipo *blank nodes* ou *bnodes* (representa um recurso sem URI), literais e subgrafos RDF, e permite construir novos grafos RDF baseados na informação de *queries* prévias.

Algumas das características que a SPARQL fornece são:

- capacidade de correspondência de padrões de grafos – esta linguagem permite

construir uma *query* que contém um grafo composto através de uma ou mais triplas RDF para ser correspondido na fonte de dados;

- capacidade de os resultados serem devolvidos como um subgrafo do grafo de dados original;
- explorar dados através de pesquisas em relações desconhecidas;
- suporte a *queries* locais e remotas. As fontes de dados podem estar armazenadas localmente ou remotamente;
- executar junções complexas de diferentes base de dados num única *query*;
- transformar dados RDF de um vocabulário para outro;
- suporte para definir um número máximo de resultados;
- a linguagem SPARQL é baseada na correspondência de padrões de grafos. O padrão mais simples do grafo é a tripla, que é como uma tripla RDF mas com a possibilidade de colocar uma variável no lugar de um dos termos sujeito, ou objecto.

```
1: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2: PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3: SELECT ?c
4:    FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
5:  WHERE {
6:          ?c rdf:type rdfs:Class .
7:  }
```

Código 3.8 – Exemplo de uma *query* SPARQL básica.

O Código 3.8 mostra um exemplo de uma *query* SPARQL simples. Esta *query* irá retornar todas as triplas que tenham propriedade `rdf:type` e objecto `rdf:Class`, isto é, irá retornar todas as classes que o grafo contém. Tal como a RDF/XML a SPARQL também usa um mecanismo de *namespace* para definir os prefixos para os *namespaces*. Isso é feito no início da *query* com a palavra PREFIX para declarar os *namespaces*, o que permite construir *queries* mais curtas e simples de ler.

A seguir à declaração dos *namespaces* vem a palavra SELECT, tal como no SQL a instrução é definida para que dados irão sejam retornados na *query*. A palavra FROM identifica os dados onde a *query* irá actuar, neste caso a *query* será realizada numa tabela periódica em RDF referida pelo URI. Uma *query* pode incluir múltiplas fontes correspondendo nesse caso a múltiplas palavras-chave FROM. A instrução WHERE indica o grafo a ser procurado e que é

constituído por uma ou mais triplas e que irão ser correspondidas no grafo.

A parte de correspondência da *query* pode incluir triplas opcionais. Este tipo de triplas são úteis para realizar *queries* que retornem dados se estes estiverem disponíveis, e não rejeitar a solução se parte da *query* normalizada não corresponder. A correspondência opcional fornece essa facilidade: se as parte opcionais não correspondem ela não cria ligações mas também não elimina a solução. Estas partes podem ainda ser juntas, isto é, unir vários grafos normalizados para correspondência através da palavra UNION: se ocorrer algumas das correspondências então será retornado um resultado. A sequência de resultados pode ser ainda alterada utilizando umas das palavras-chave com significado similar ao SQL:

- ORDER BY – ordenar pelo valor de uma variável;
- DISTINCT – apenas devolver resultados únicos;
- LIMIT – número máximo de resultados;
- OFFSET – offset a partir do qual os valores serão apresentados.

Existem ainda quatro tipo de *queries*, são elas:

- SELECT – já apresentada anteriormente, retorna o valor das variáveis descritas na *query* de pesquisa;
- CONSTRUCT – retorna um grafo RDF construído substituindo as variáveis descritas na *query*;
- DESCRIBE – retorna um grafo RDF descrevendo os recursos que foram encontrados;
- ASK – retorna um valor booleano indicando se o grafo padrão definido na *query* foi encontrado ou não.

A *query* do Código 3.9 ilustra o exemplo de uma *query* do tipo ASK para determinar se um dispositivo iPhone possui ou não GPS e terá um resultado do tipo booleano.

```
1: PREFIX devi: <http://icas.ipb.pt/secom/device>
2: ASK {
3:    ?dev devi:hasDeviceComponent ddevi:GPS .
4:    ?dev devi:deviceModel ?deviceModel .
5:    FILTER regex(?deviceModel, "iPhone")
6:   }
```

Código 3.9 – Exemplo de uma *query* ASK à ontologia *Device* do modelo SeCoM.

O Código 3.10 mostra outro exemplo de uma *query* SPARQL, que retorna todos os nós (pessoas) que contenham o nome Pedro na propriedade `hasName`.

```
1: PREFIX acto: <http://icas.ipb.pt/secom/actor>
2: SELECT ?hasFirstName ?hasName ?hasSurName ?hasBirthday WHERE {
3:       ?person acto:hasName ?hasName .
4:       ?person acto:hasFirstName ?hasFirstName .
5:       ?person acto:hasSurName ?hasSurName .
6:       ?person acto:hasBirthday ?hasBirthday .
7:       FILTER regex(?hasName, "Pedro")
8:      }
```

Código 3.10 – Exemplo de uma *query* SELECT à ontologia Actor do modelo SeCoM.

Os resultados das *queries* SPARQL podem ser em dois formatos: conjunto de resultados e grafos RDF. Os conjuntos de resultados são ilustrados de uma forma tabelar, como aparece na Tabela 3.5.

Tabela 3.5 – Resultado da *query* do Código 3.10 no formato conjunto de resultados.

hasFirstName	hasName	hasSurName	hasBirthday
"João"	"João Pedro"	"Marques"	"12/02/1979"
"Pedro"	"Pedro João"	"Santos"	"09/09/1971"
"José"	"José Pedro"	"Fernandes"	"16/01/1980"

O resultado acima no formato conjunto de resultados, pode depois ser serializado em XML (Beckett and Broekstra, 2008), como mostra o Código 3.11.

```
1: <?xml version="1.0"?>
2: <sparql xmlns="http://www.w3.org/2005/sparql-results#">
3:   <head>
4:     <variable name="hasFistName"/>
5:     <variable name="hasName"/>
6:     <variable name="hasSurName"/>
7:     <variable name="hasBirthday"/>
8:   </head>
9:   <results>
10:    <result>
11:      <binding name="hasFistName">
12:        <literal>João</literal>
13:      </binding>
14:      <binding name="hasName">
15:        <literal>João Pedro</literal>
16:      </binding>
17:      <binding name="hasSurName">
18:        <literal>Marques</literal>
19:      </binding>
20:      <binding name="hasBirthday">
21:        <literal>12/02/1979</literal>
22:      </binding>2
23:   </result>
24:    <result>
25:      <binding name="hasFistName">
26:        <literal>Pedro</literal>
```

```
27:       </binding>
28:       <binding name="hasName">
29:         <literal>Pedro João</literal>
30:       </binding>
31:       <binding name="hasSurName">
32:         <literal>Santos</literal>
33:       </binding>
34:       <binding name="hasBirthday">
35:         <literal>09/09/1971</literal>
36:       </binding>
37:     ...
38:     </result>
39:   </results>
40: </sparql>
```

Código 3.11 - Resultado da *query* do Código 3.10 serializado em XML

As *queries* que pode retornar o formato grafo RDF são as *queries* DESCRIBRE e CONSTRUCT. O Código 3.12 mostra o exemplo de uma *query* CONSTRUCT mais simples.

```
1: PREFIX acto: <http://icas.ipb.pt/secom/actor>
2: PREFIX rela: <http://icas.ipb.pt/secom/relationship>
3:
4: CONSTRUCT { ?p acto:hasFirstName ?hasFirstName }
5: WHERE { ?x rela:isFriendof ? ?hasFirstName }
```

Código 3.12 – Exemplo de uma *query* do tipo CONTRUCT.

O resultado da *query* CONSTRUCT poderia ser o grafo RDF representado no Código 3.13. Este grafo é construído baseado num modelo que é usado para gerar triplas RDF segundo o resultado da correspondência do grafo padrão da *query*, como mostra o Código 3.13.

```
1: @prefix  <acto: http://icas.ipb.pt/secom/actor>
2:
3: _:x acto:hasFirstName "João" .
4: _:y acto:hasFirstName "José" .
```

Código 3.13 – Grafo RDF como resultado da *query* do Código 3.12.

O Código 3.14. é o resultado serializado em RDF/XML retornado pela query definida no Código 3.13.

```
1: <rdf:RDF
2:      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:      xmlns:acto=”http://icas.ipb.pt/secom/actor">
4:    <rdf:Description>
5:      <acto:hasFirstName >José</foaf:hasFirstName >
6:    </rdf:Description>
7:    <rdf:Description>
8:      <acto:hasFirstName>José</acto:hasFirstName >
9:    </rdf:Description>

10: </rdf:RDF>
```

Código 3.14 – Resultado da *query* do Código 3.12 serializado em XML.

Juntamente com a linguagem SPARQL foi desenvolvido o SPARQL Protocol, que transporta as *queries* dos clientes até aos processadores de *queries* (Figura 3.8). O SPARQL Protocol é descrito de dois modos: o primeiro utilizando uma interface abstracta de qualquer implementação ou protocolo usando para isso a WSDL 2.0; e o segundo é a utilização do HTTP e SOAP para ligar a essa interface. Os resultados são devolvidos em XML sendo o formato do documento definido pela *SPARQL Query Results XML*. Por fim, o motor de inferência extrai os dados das fontes RDF ou de outras fontes de informação, através de ferramentas de conversão, como por exemplo, as linguagens de descrição de recurso GRDDL (Connolly, 2007) e RDFa (Adida, Birbeck et al., 2008).

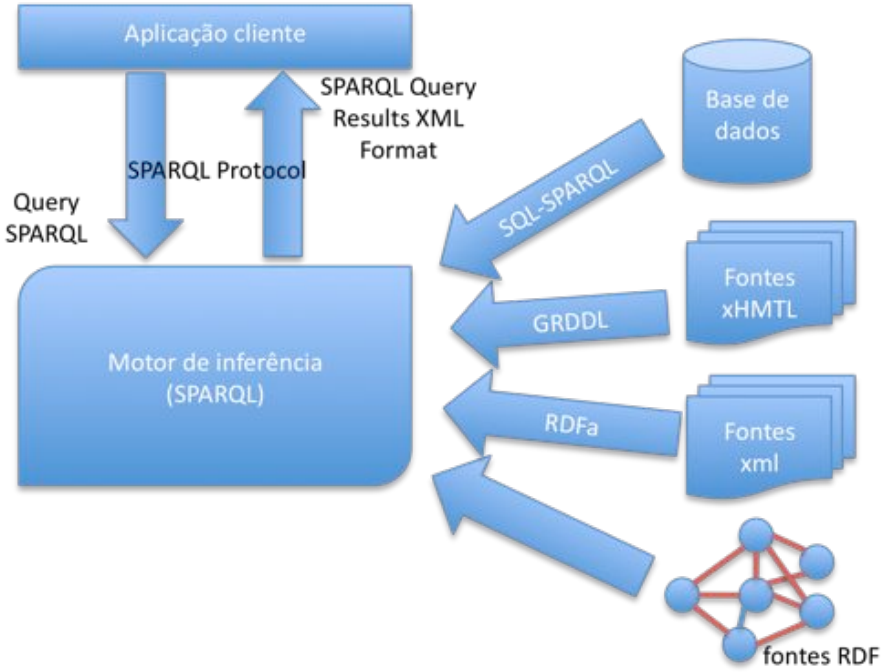


Figura 3.8 – Modelo genérico de uma aplicação cliente a realizar *queries* a um motor de inferência.

Como uma linguagem de *queries*, a SPARQL apenas pesquisa informação em grafos, isto é não existe nenhum mecanismo de inferência na própria linguagem. Alguns motores de inferência podem no entanto aparentar que certas triplas existem através da sua criação em tempo real, utilizando um motor de inferência baseado em RDFS, como por exemplo o ARQ, um processador SPARQL para a *framework* Jena (Hewlett-Packard, 2009).

Embora existam várias implementações que usam a sintaxe SPARQL como linguagem de *query* em grafos RDF e existem também muitas expectativas na possibilidade de usar a SPARQL para pesquisar OWL. Uma das propostas que mais se destaca é a SPARQL DL, que segundo os autores em (Sirin and Parsia, 2007) é significativamente mais expressiva do que as *queries* baseadas em lógica de descrição já existentes.

Existem diversas ferramentas e APIs que suportam motores de *queries* SPARQL. A maior parte deles está actualizado com a sua última especificação e alguns deles serão abordados na secção 3.3.1.

3.2.6 Raciocínio (Reasoning)

A semântica é uma das bases para suportar o raciocínio e a OWL é uma linguagem baseada na descrição lógica e que suporta a semântica formal, isto é permite descrever precisamente o significado do conhecimento. Não existem referências subjectivas ou espaço para interpretações diferentes por agentes de software ou por pessoas. Algumas das utilizações da semântica formal são: raciocinar sobre o conhecimento ontológico, verificar a consistência da ontologia e do conhecimento, verificar se existem relacionamentos incongruentes entre classes e automatizar a classificação de instâncias em classes.

A partir das definições formais das classes, propriedades e indivíduos, é possível inferir novo conhecimento utilizando uma base de conhecimento já existente. Este tipo de inferência, que envolve estas três entidades e as suas associações, é chamada Lógica Descritiva (*Description Logic* – DL) e tornou-se muito popular como a base da OWL. Segundo Brussee e Pokraev (Brussee and Pokraev, 2007) existem três classes de raciocínios:

- raciocínio de propriedade – que significa inferir triplas implícitas a partir do estado das já existentes. A Figura 3.9 ilustra o raciocínio de uma propriedade transitiva, na qual se infere que "Europa" contém "Lisboa";

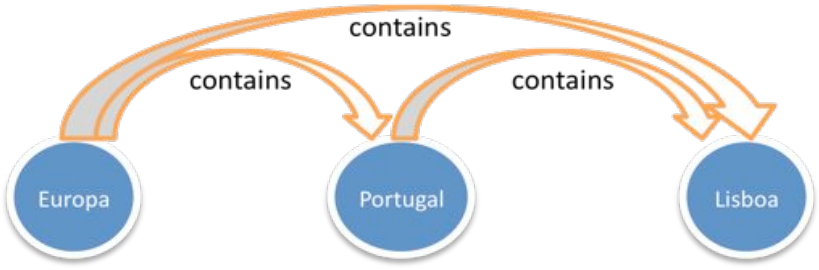


Figura 3.9 – Exemplo de uma tripla afectada por uma propriedade transitiva.

- raciocínio de classe – significa verificar se uma classe B é subclasse da classe A. Este raciocínio, que é chamado de verificação de absorção (*subsumption check)*, verifica se os critérios de ser membro da classe B implicam os critérios de ser membro da classe A. Se A e B se incluem mutuamente então as classes são equivalentes (*equivalent*) e têm os mesmos membros. Outra das verificações é a

satisfação (*satisfiability*) que é um caso especial de raciocínio de absorção (*subsumption reasoning*). Uma Classe C é *unsatisfiable* se C ⊆ *Nothing* (C está contido em *Nothing*, em que *Nothing* é a classe das quais derivam todas as outras classes) e se *Nothing* ⊆ C, fazendo com que C seja equivalente a *Nothing*. Se for construído o gráfico completo de absorção de todas as classes na ontologia, é construída a hierarquia de classes e a esta tarefa de raciocínio chama-se *classification*;

- raciocínio de indivíduo – é mais conhecido por verificação da consistência (*consistency check*) e consiste em verificar se um indivíduo pode existir num determinado modelo. Se uma classe é *unsatisfiable* então não pode ter um indivíduo como membro. Pelo contrário, existe outra tarefa relacionada chamada *realization* que consiste em encontrar classes de que um determinado indivíduo é membro. Outra das operações com grande importância é *retrieve*, que consiste em encontrar todas as instâncias que pertencem a uma classe. Esta operação permite utilizar modelos lógicos como bases de dados e permite a utilização de linguagem de *query*. Tal como no raciocínio por classes, também aqui é possível uma instância de uma classe ser equivalente a outra (*equivalent*).

Os sistemas que são baseados em descrição lógica também são designados sistemas terminológicos sendo compostos por três componentes básicos (Figura 3.10): a base de conhecimentos (*knowledge base* – kb), o motor de inferência e a interface.

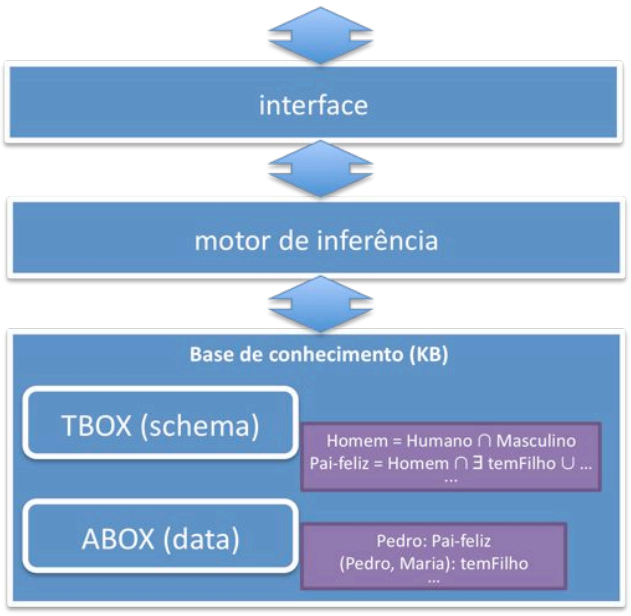


Figura 3.10 – Arquitectura do sistema terminológico (Horrocks, 2002).

A base de conhecimento contém informação sobre o domínio representado pela hierarquia de classes e sobre o relacionamento entre elas. A base de conhecimento é por sua vez constituída por dois componentes internos, a *Terminological Box* – TBOX e a *Assertional Box* – ABOX. A TBOX contém as descrições dos conceitos do domínio e é construída através de declarações que descrevem as propriedades dos conceitos. A ABOX contém factos sobre os componentes terminológicos, conhecimento específico para os indivíduos do domínio. Por outras palavras a TBOX contém as definições dos conceitos e regras e a ABOX contém as definições dos indivíduos. Utilizando a analogia das bases de dados relacionais, estas são compostas por duas partes: o esquema ou estrutura de dados e os próprios dados. Numa base de conhecimento a TBOX pode ser comparada ao esquema de bases de dados e a ABOX corresponde aos dados.

Estes sistemas possuem dois tipos de raciocínios: (1) o raciocínio terminológico que é feito sobre a TBOX e inclui o todas as funcionalidades descritas anteriormente pelo raciocínio de propriedade e de classe; (2) o racionamento sobre as instâncias é feito pela ABOX e inclui as funcionalidades do raciocínio de instância.

3.3 Ferramentas para a Semantic Web

Características únicas da *Semantic Web* criaram a necessidade de ferramentas próprias para o desenvolvimento de ontologias, a gestão de conteúdos, o processamento das bases de conhecimento e instâncias de dados, e um conjunto de bibliotecas para suporte ao desenvolvimento rápido de aplicações da *Semantic Web*. Para o desenvolvimento de ontologias apareceram os editores, validadores sintácticos e visualizadores gráficos. Para a gestão de conteúdos apareceram as APIs que permitem a marcação dinâmica de conteúdos e o seu posterior manuseamento. Para o processamento de conhecimento e instâncias de dados são necessários os motores de inferência e de *queries*. Nesta secção serão apresentadas algumas destas ferramentas para o desenvolvimento de aplicações *Semantic Web*, com particular atenção para os motores de inferência, sistemas de armazenamento e ferramentas de ontologias, nas quais se incluem os editores de ontologias e um conjunto de aplicações de apoio ao desenvolvimento de *Web Services Semânticos*.

3.3.1 Motores de inferência

Os motores de inferência, também chamados *reasoners*, conseguem deduzir (inferir) novos factos ou associações a partir de informação existente numa base de conhecimento. O motor de inferência funciona como um mecanismo de controlo, que aplica conhecimento axiomático a partir da fonte de conhecimento para chegar a alguma conclusão. Utilizando um exemplo muito simples, suponha-se que a base de conhecimento contém os dois factos seguintes: Maria é feminina; feminina é um tipo de pessoa. O motor de inferência baseado nestes dois factos pode inferir que Maria é uma pessoa. Outras das funcionalidades que os motores de inferência suportam é a validação de ontologias, de modo a verificar que não existem inconsistências, isto é, utilizando o exemplo anterior, cria-se uma regra em que diz que mulheres e homens são mutuamente exclusivos. Se posteriormente dissermos que Maria é masculino, o motor de inferência, ao validar a ontologia, irá detectar uma contradição nos dados da base de conhecimento.

As regras de inferência são normalmente especificadas em linguagem ontológica ou linguagem descritiva. A criação de um modelo de informação e relacionamentos permite aos motores de inferência retirar conclusões lógicas baseadas nesse modelo e ser possível perguntar aos motores de inferência a razão de terem chegado a uma determinada conclusão. As linguagens mais utilizadas para criar os modelos de informação e relacionamentos são RDFS, OWL e SWRL. Actualmente existem diversos motores de inferência disponíveis que podem fornecer vários níveis de inferência, mas apenas analisaremos os mais populares e que suportam inferência em RDFS e OWL.

3.3.1.1 Pellet

O *Pellet* (Kolovski, Parsia et al., 2006) foi iniciado no *Mindswap Group* da Universidade de *Maryland Institute for Advanced Computer Studies*. Posteriormente este modelo foi aproveitado pela *Clark & Parsia LLC*, que o disponibiliza sob a licença AGPLv3 (Foundation, 2007) e licença proprietária para aplicações de código fechado. Entre as diversas funcionalidades que o motor de inferência oferece destacam-se as seguintes:

- implementa a descrição lógica SROIQ (D);
- suporta um conjunto de regras SWRL chamadas *DL-Safe Rules*;
- diversas interfaces para inferir, entre as quais uma interface gráfica baseada num

navegador *Web*, com as APIs *Jena* e OWL, e também fornece uma interface para ferramentas através da ferramenta *DL Implementation Group* – DIG (Bechhofe, 2003), que fornece um acesso uniforme a motores de inferência de descrição lógica;

- suporta actualmente a maior parte das funções propostas na OWL 2 (Motik, Grau et al., 2009), apesar de esta estar ainda em especificação pelo W3C;
- análise, validação da consistência, reparação de ontologias OWL e *debug* de ontologias;
- motor de linguagem de *queries* conjuntivas SPARQL-DL (Sirin and Parsia, 2007).

Mais informações sobre este motor de inferência podem ser encontradas em (Sirin and Parsia, 2007).

3.3.1.2 Jena

O *Jena* (Carroll, Dickinson et al., 2004) é mais do que um motor de inferência, é uma *framework* para o desenvolvimento de aplicações com recurso à *Semantic Web*. É um projecto de código aberto que tem vindo a ser desenvolvido pelo *HP Labs Semantic Web Programme*. Bastante disseminada na comunidade da *Semantic Web*, providencia uma ambiente de programação para suportar RDF, RDFS e OWL. Entra as suas principais características encontram-se:

- uma API RDF para a manipulação de ontologias codificadas em RDF (leitura escrita, importação/exportação de formatos de dados);
- uma API OWL para a manipulação de ontologias codificadas em OWL;
- um motor de *queries* a dados RDF suportando linguagens RDQL e a SPARQL;
- motor de inferência baseado em regras, além de facilmente integrar motores de inferência de terceiros (ex. o *Pellet* que já vem incluído no próprio pacote de software);
- mecanismos para a manipulação de dados RDF em memória, ficheiros simples e bases de dados relacionais, as duas últimas designadas ontologias persistentes.

3.3.1.3 KAON2

O KAON2 (Hustadt, Motik et al., 2009) é um projecto descendente do KAON (Bozsak, Ehrig et al., 2002) desenvolvido sob licença GPL (Foundation, 2007) pelo *Institute AIFB* da *Universidade de Karlsruhe* e pelo FZI – *Research Center for Information Technologies*. Criado através do esforço que envolve as várias instituições, o KAON2 tem com principal diferença em relação ao seu predecessor o suporte às linguagens. Enquanto o KAON usava um extensão proprietária da RDFS, o KAON2 (Hustadt, Motik et al., 2009) é baseado em OWL-DL e *Frame Logic* – F-Logic (M. Kifer, G. Lausen et al., 1995). É considerado uma *framework* que fornece uma infra-estrutura para a gestão de ontologias para aplicações de negócios. As suas principais características são:

- suporte a expressividade lógica SHIQ (D) da OWL DL e um subconjunto de regras chamadas de DL-safe do SWRL;
- gestão de ontologias OWL-DL, SWRL, e F-Logic;
- servidor que fornece acesso a ontologias de modo distribuído;
- motor de inferência para responder a *queries* expressas em sintaxe SPARQL;
- interface de acesso as outras ferramentas através da DIG, como o Protégé (Musen, 1988);
- extracção de instâncias de ontologias de bases de dados relacionais.

Mais informações sobre este projecto podem ser encontradas em (Hustadt, Motik et al., 2009).

3.3.1.4 RacerPro

O RacerPro (Haarslev and Möller, 2001) foi desenvolvido pela empresa *Racer Systems GmbH & Co.* e é um motor de inferência para as linguagens RDF, OWL, OWL2 e um repositório de informação.

Algumas das principais características do RacerPro são:

- implementa a descrição lógica SHIQ;
- suporta a linguagem semântica de *queries* nRQLe a linguagem SWRL;
- suporta a verificação da consistência de ontologias OWL e descrição de dados RDF;

- suporta ontologias persistentes, isto é, permite o armazenamento de base de conhecimento em bases de dados relacionais como o PostgreSQL, MySQL, HSQLDB, entre outras;
- interface de acesso as outras ferramentas através da DIG;

3.3.1.5 FACT++

O FaCT++ (Tsarkov and Horrocks, 2006) tem vindo a ser desenvolvido principalmente pelo *Computing Laboratory* da Universidade de Oxford. É um motor de inferência baseado em descritores lógicos, que foi re-implementado a partir do motor de inferência FaCT. Diferencia-se do seu antecessor pelo facto de ser desenvolvido em C++ e foi-lhe ainda introduzida uma série de melhoramentos na sua arquitectura interna sendo agora personalizável (Tsarkov and Horrocks, 2003).

As características mais relevantes são apresentadas em seguida:

- implementa a descrição lógica SHOIQ;
- suporta as linguagem OWL 1.1 e parcialmente OWL 2;
- pode ser usado por outras ferramentas através da interface DIG;
- é usado como um dos motores de inferência no Protégé.

3.3.1.6 Comparação dos motores de inferência

A Tabela 3.6 apresenta uma comparação e resumo das características mais relevantes dos motores de inferência anteriormente descritos. Os motores de inferência processam conhecimento disponível na *Semantic Web* e deduzem novo conhecimento do conhecimento previamente estabelecido. Diversos motores de inferência foram anteriormente descritos, todos eles são baseados em lógica descritiva e são capazes de inferir em diferentes linguagens com diferentes níveis de expressividade, por exemplo RDFS, OWL e SWRL. Entre as suas principais tarefas estão a inferência de novo conhecimento, mas também a verificação da consistência das ontologias, processamento e classificação de factos, gestão de tipos de dados, entre outros.

Como se pôde constatar pela análise prévia, juntamente com os motores de inferência são fornecidos conjuntos de ferramentas, para o apoio ao desenvolvimento de aplicações

avançadas de *Semantic Web* e que oferecem bibliotecas de desenvolvimento, suporte para importar/exportar diferentes formatos de ontologias, interfaces com bases de dados e outras ferramentas de terceiros. Nesta breve análise dos diversos motores de inferência não foram considerados quaisquer parâmetros sobre o desempenho de cada um deles ou comparação entre eles, no entanto, existem diversos estudos comparativos sobre testes de desempenho na comunidade científica e que podem ser encontrados em (Tsarkov and Horrocks, 2003), (Pan, 2005), (Gardiner, Horrocks et al., 2006) e (Bock, Haase et al., 2008).

Tabela 3.6 – Tabela comparativa das características de diversos motores de inferência.

Características	Fact++	Jena	KAON2	Pellet	RacerPro
Suporte OWL	OWL 1.1	Parcial/Pode utilizar outros motores de inferência	OWL 1.1	OWL 1.1 e OWL 2 parcialmente	OWL 1.1
Suporte de Regras	Não	Sim (formato próprio)	SWRL (DL-Safe Rules)	SWRL (DL-Safe Rules)	SWRL (parcialmen-te)
Suporte de *queries* SPARQL	Não	Sim	Sim	Sim	Brevemente
Suporte de *queries* a fontes OWL	Não	Não	Não	SPARQL DL	OWL QL (através da nRQL)
Expressividade lógica	SROIQ (D)	SROIQ (D)	SROIQ (D)	SROIQ (D)	SHIQ(Dn)
Verificação de consistência	Sim	OWL DL (Parcial)	Sim	Sim	Sim
Suporte à DIG	Sim	Sim	Sim	Sim	Sim
Tecnologia de desenvolvimen-to	C++	Java	Java	Java	APIs em Java/LISP
Versão	1.3.0	2.6.0	Desconhecida	2.0 (rc7)	2.0
Licença	GNU	Open-Source	Gratuito (uso não comer-cial)/Código Fechado	AGPL(v3) Comercial/Có-digo fechado	Comercial/ Código Fechado

3.3.2 Sistemas de armazenamento

O método mais usado para ligar dados a ontologias é através da representação de dados RDF que descrevem os dados como instâncias da ontologia correspondente. Devido à importância do acesso a dados RDF e RDFS, tem havido um grande esforço no desenvolvimento de sistemas que permitem o armazenamento e a recuperação deste tipos de dados. Em seguida são descritos alguns dos muitos sistemas de armazenamento orientados à descrição de conceitos escritos sob a forma de triplas RDF:

- 4store – desenvolvido por Steve Harris (Harris, 2006) da empresa Garlik desde 2006 é distribuído sob licença GNUv3. É uma base de dados e um motor de *queries* que trata dados RDF. Não providencia muitas funções referindo-se apenas ao armazenamento de dados RDF, *queries* SPARQL, funções de importação de dados e de cópias de segurança/restauro, mas anuncia como pontos fortes as escalabilidade, desempenho e segurança (Harris, 2006);

- Jena – já abordado anteriormente, fornece ainda através de um dos seus componentes (SDB – *A SPARQL Databse for Jena*), um sistema de armazenamento escalável para conjunto de dados RDF utilizando bases de dados SQL, de modo a poderem ser usadas em aplicações autónomas, JEE ou outras *framework*s (Jena, 2009). Utilizando esta funcionalidade o *Jena* permite ainda o armazenamento de ontologias de forma persistente. Para isso, o *Jena* serializa as ontologias em RDF e, através da interface do modelo RDF, guarda as triplas de forma persistente numa base de dados, o que reduz o tempo de carregamento do modelo ontológico cada vez que a aplicação é executada e permite armazenar modelos significativamente maiores que a memória principal do computador;

- OpenLink Virtuoso – desenvolvido pela empresa *OpenLink Software* é distribuído sobre duas licenças, uma comercial e outra código aberto e de nome *OpenLink Virtuoso* (OpenLink-Software, 1998). O *OpenLink* Virtuoso é constituído por um *middlware* e uma base de dados híbrida para ajudar no desenvolvimentos de aplicações na área dos *Web Services*, gestão de dados RDF e XML. Fornece suporte a *queries* SPARQL, armazenamento de triplas RDF e a diversas linguagens de *queries* a bases de dados relacionais. Mais informações sobre esta *framework* podem ser encontradas em (OpenLink-Software, 1998);

- Mulgara – o *Mulgara* foi iniciado em 2006 como uma divisão do projecto Kowari

(Kowari, 2004). O *Mulgara* é uma base de dados RDF escrita inteiramente em Java e código aberto. Ao contrário das bases de dados relacionais que guardam os dados em tabelas o *Mulgara* diferencia-se pelo facto de guardar todos os dados em forma de triplas. Tem como principais características (Gearon, Wood et al., 2006): suporte RDF nativo; suporte a *queries* SPARQL; conectividade com outras ferramentas (ex. *Jena* e JRDF);

- Redland RDF Libraries – o *Redland* (Beckett, 2001) é um conjunto de bibliotecas que fornecem suporte RDF dividido em: (1) *Raptor RDF Parser Library*, para analisar e fazer a serialização de sintaxe RDF; (2) *Rasqal RDF Query Library* para a execução de *queries* RDF; (3) *Redland RDF Library* que fornece uma API RDF e guarda triplas; (4) *Redland Language Bindings*, que permitem a manipulação de dados RDF e é disponibilizada em diferentes APIs para serem usadas com as linguagens de programação *Perl*, PHP, *Python* e *Ruby*. Tem como principais características: oferecer um conjunto de APIs para a manipulação de grafos, triplas URIs e literais RDF; permitir o armazenamento de grafos de modo persistente em diferentes base de dados relacionais ou ficheiros; suporte a múltiplas sintaxes para ler e escrever dados RDF como: RDF/XML, N-Triple, RSS entre outros; suporte de *queries* SPARQL e RDQL;

3.3.3 Ferramentas para gerir Ontologias

Este tipo de ferramentas começaram aparecer a meio dos anos 90 com o objectivo de fornecerem suporte ao desenvolvimento de ontologias, através de uma interface gráfica. Nesta secção são descritas as principais características das ferramentas mais relevantes e usadas pela comunidade científica para o manuseamento de ontologias. Como no desenvolvimento da arquitectura proposta nesta tese, foi necessário desenvolver e editar ontologias de domínios e de *Web Services*, são descritos dois tipos de ferramentas, os editores de ontologias e as ferramentas de suporte ao desenvolvimento de descrições ontológicas de serviços.

3.3.3.1 Editores de ontologias

Maior parte dos editores de ontologias apresentam um conjunto de características básicas relacionadas com o manuseamento de ontologias como: a sua criação, edição, importação/exportação em outros formatos e verificação sintáctica. Outras mais avançadas

suportam várias linguagens e motores de inferências, representações gráficas de ontologias, *queries* SPARQL, etc. Alguns dos principais editores são brevemente apresentados de seguida:

- *Protégé* (Musen, 1988) – é provavelmente a ferramenta mais conhecida e usada para criar ontologias e bases de dados de conhecimento. Foi iniciado pelo *Knowledge Modeling Group at Stanford Medical Informatics* da *Stanford University School of Medicine*. sendo distribuída gratuitamente sob licença *Mozilla Public License Version 1.1* (Mozilla, 2009). A sua principal função é permitir aos programadores criar e editar ontologias. Oferece ainda uma biblioteca com dezenas *plug-ins* que permitem estender as suas funcionalidades. Permite também a escolha de vários motores de inferências como *Pellet*, Fact++, *RacerPro*, KAON2, entre outros;
- KAON2 – já explicado anteriormente na secção dedicada aos motores de inferência, oferece também um editor de ontologias com capacidades para mapeamento e geração de ontologias a partir de bases de dados, entre outros;
- *OntoEdit* (Sure, Erdmann et al., 2002) – tem como principiais funcionalidades a pesquisa, edição e inferência em ontologias. Suporta as linguagens F-Logic, RDFS, OIL e permite o armazenamento de conhecimento em XML. Tal como o Protégé baseia-se numa arquitectura extensível recorrendo para isso aos *plug-ins*;
- SWOOP (Kalyanpur, Parsia et al., 2005) – permite a criação, edição e *debug* de ontologias OWL. Distribuído sob a licença *MIT License* caracteriza-se por utilizar o motor de inferência *Pellet* e apresentar um interface gráfico inspirado em navegadores *Web*. Como principais funções oferece suporte colaborativo como anotações colaborativas; permite a validação do código, múltiplas sintaxes e a representação gráficas de ontologias; suporta operações de pesquisa, mapeamento, *debug* e comparação de ontologias.

3.3.3.2 Ferramentas OWL-S

Nesta secção são apresentadas algumas das ferramentas para trabalhar com *Web Services Semânticos* mais aceites pela comunidade científica. Entre as suas funções destacam-se as de edição, validação sintáctica e conversão de interfaces WSDL para OWL-S. Em seguida são resumidas algumas das mais populares ferramentas OWL-S:

- *OWL-S Editor* (J.Scicluna, C.Abela et al., 2004) – desenvolvido na Universidade de Malta, oferece duas metodologias para o desenvolvimento de descrições para *Web Services* utilizando a linguagem OWL: o mapeamento de WSDL para OWL-S e a modelação de um serviço composto utilizando diagramas de actividades em UML. Esta ferramenta é composta por três componentes: o *creator* que permite criar um esqueleto descritor OWL-S; o *validator* que valida a sintaxe das ontologias e os URIs; e o *visualizer* que permite, compor processos atómicos extraídos de ficheiro WSDL utilizando diagramas de actividades UML;
- *OWL-S Editor plug-in* for Protégé (Elenius, Denker et al., 2005) – este editor é implementado como um *plug-in* do editor de ontologias do Protégé e é distribuído na forma de código aberto. Permite trabalhar com todos os aspectos dos SWS, fornecendo visualizadores e um editor gráfico que possibilita relacionar os seus serviços com conceitos de domínios ontológicos. Entre as suas funções estão: a edição gráfica, visualização do fluxo de controlo e fluxo de dados, gestão de relacionamentos entre componentes dos serviços e parâmetros, e a geração do esqueleto da descrição OWL-S a partir dos ficheiros WSDL;
- *OWL-S Validator* (Sirin, 2004) – tal como SWOOP também foi desenvolvida pelo *MIND Lab* e faz a validação sintáctica e estrutural de ficheiros OWL-S de modo a encontrar os erros mais comuns;
- WSDL2OWL *Translator* – foi desenvolvida pelo *MIND LAB* da Universidade de *Maryland* e faz parte da OWL-S API. Esta ferramenta fornece uma interface gráfica que permite através de um ficheiro WSDL criar uma descrição OWL-S;
- WSDL2OWL-S *Converter* (Paolucci, Srinivasan et al., 2005) – foi desenvolvida na Universidade de *Carnegie Mellon* e é a última evolução da WSDL2DAML-S (Paolucci, Srinivasan et al., 2003). Permite a transformação da descrição de um serviço de WSDL para OWL-S, fornecendo uma estrutura básica da descrição de um *Web Service* em OWL-S.

3.4 Resumo

A *Web* actual é um enorme repositório de documentos interligados e serviços apropriados

para o consumo humano. De modo a ajudar os humanos a encontrar de forma mais eficiente informações necessárias, a W3C tem vindo a desenvolver a *Semantic Web* com o objectivo de tornar a *Web* semântica com o objectivo de permitir que os computadores consigam processar a informação que esta contém.

Neste capítulo foi descrita a arquitectura da *Semantic Web* que deverá fornecer uma *framework* para a partilha, reutilização e interpretação de dados entre aplicações. A arquitectura da *Semantic Web* pode ser resumida em quatro camadas principais: (1) a camada básica de dados que segue as características principais da WWW, fornecendo um mecanismo de codificação normalizado de todas as linguagens humanas e uma forma de identificação e localização de recursos na Internet; (2) a camada de descrição de dados que garante uma camada sintáctica comum na *Semantic Web*; (3) a camada de descrição estrutural e semântica para descrever e representar informação sobre recursos e para a criação de ontologias simples; (4) a camada de semântica e lógica usada para enriquecer mais a língua ontológica e permitir a representação de conhecimento aplicacional específico que oferece diferentes níveis de representação de conhecimento através de ontologias e de regras.

Também fazendo parte da arquitectura da *Semantic Web* foi apresentada a SPARQL, uma linguagem *query* para RDF. Foi destacada por ser umas das tecnologias largamente utilizadas para a extracção de dados e conhecimento básico das bases de dados de conhecimento.

O raciocínio nas ontologias e bases de dados de conhecimento permite inferir novas informações que não estão expressadas explicitamente nas fontes. Neste capítulo foi abordado o raciocínio em ontologias OWL, que através da descrição lógica permite ao motor de inferência a realização de várias tarefas como a extracção de conhecimento, e a verificação de consistência.

Foram também introduzidas algumas das mais conhecidas ferramentas de apoio para o desenvolvimento de *Web Services Semânticos*. Mais informações sobre ferramentas para a OWL-S e OWL podem ser encontradas em (DAML, 2005) e (Bizer and Westphal, 2007). Entre essas ferramentas, incluem-se motores de inferência, sistemas de armazenamento, editores e visualizadores avançados de ontologias, validadores sintácticos e de consistência sintáctica, bibliotecas de desenvolvimento, conversores, entre outros.

Capítulo 4

TECNOLOGIAS DE SUPORTE

Este capítulo tem como principal objectivo esclarecer os conceitos sobre as diversas tecnologias utilizadas nas arquitecturas orientada aos serviços baseadas em *Web Services* e *Web Services Semânticos*.

A composição de serviços é um dos principais objectivos desta tese. Neste sentido é incluída neste capítulo, uma breve abordagem à composição de *Web Services* tradicionais, em seguida são introduzidos os *Web Services Semânticos* e as novas propriedades que trazem e as principais diferenças em relação aos primeiros.

Posteriormente são apresentadas as principais *framework*s e linguagens que suportam os *Web Services Semânticos* e é feita uma análise comparativa entre elas.

Finalmente, é apresentado o modelo contextual semântico SeCoM, utilizado para representar a semântica da informação contextual através de um conjunto de ontologias e que pode ser usado na construção de aplicações *context-aware*.

4.1 Arquitectura de *Web Services*

As arquitecturas orientadas ao serviços seguem um paradigma de desenvolvimento e de funcionamento que fez com que a popularidade crescesse rapidamente em aplicações de negócio e científicas. A W3C define uma arquitectura SOA como "um conjunto de componentes que podem ser invocados, e cujas interfaces descritivas podem ser publicadas e descobertas" (Haas and Brown, 2004). O facto de uma arquitectura ser orientada ao serviço é só um conceito e, como tal, requer uma implementação física para ser utilizável. Duas das

tecnologias mais conhecidas actualmente e utilizadas nas arquitecturas SOA são os *Web Services* (Alonso, Casati et al., 2003) e a tecnologia *Jini* (Waldo and Arnold, 2000), que fornece uma arquitectura orientada aos serviços para a plataforma Java. Embora grande parte dos conceitos das SOA fossem especificados antes do aparecimento dos *Web Services*, estes são na actualidade praticamente a única tecnologia utilizada para a construção de arquitecturas SOA, acabando por se generalizar Arquitecturas de *Web Services* como Arquitecturas Orientadas aos Serviços.

O sucesso das arquitecturas SOA na implementação de aplicações distribuídas em relação a aproximações anteriores é notável. Como consequência, algumas dessas soluções têm vindo a desaparecer como é o caso das arquitecturas de objectos distribuídos: Java RMI (SUN, 1997), CORBA (OMG, 1991) e DCOM (Microsoft, 1997). Este êxito deve-se ao facto dos *Web Services* fornecerem o seguinte conjunto de características, que não foi possível reunir em arquitecturas anteriores:

- interoperabilidade – permitem a comunicação entre o requisitante do serviço e o fornecedor do serviço, independentemente do sistema em estejam a ser executados ou da tecnologia de implementação de cada um deles.
- normas – utilizam protocolos e tecnologias de interacção normalizadas para permitir a interoperabilidade entre parceiros independentes; todos se regem pelas mesmas normas de modo a evitar a quebra de interacção;
- descoberta – utilizam uma directoria de descrição de serviços, sendo possível pesquisar serviços pela interface e o pelo tipo de serviço. Garante que ambas partes possam ser descobertas dinamicamente e possam posteriormente comunicar;
- escalável – é possível encontrar implementações de arquitecturas SOA dentro de empresas e em fornecedores de dimensão mundial que oferecem serviços através da Internet (ex: *Amazon*, *Google*, etc);
- *loosely-coupled* (independentes) – resulta da utilização de XML para a formatação das mensagens trocadas entre os serviços; esta adopção permite que os serviços se possam conectar a outros serviços e clientes usando um mecanismo de comunicação normalizado, reduzindo assim a dependência;
- abstracção – através da utilização de tecnologias normalizadas da *Web*, são implementadas funções de interacção entre ambas as partes, que permite aos

desenvolvedores criar serviços que se concentrem mais na lógica de negócio do que na integração de sistemas; é uma arquitectura que permite a abstracção da tecnologia;

- projecto baseado em interfaces – é possível implementar a funcionalidade dos serviços separadamente das interfaces, o que permite a existência de múltiplos serviços para uma única interface comum ou que um serviço possa implementar diferentes interfaces.

Numa arquitectura SOA podem existir diversos intervenientes que executam diferentes acções. A Figura 4.1 ilustra as entidades existentes numa arquitectura SOA e o fluxo das acções executadas entre os intervenientes, e que serão explicadas em seguida.

Figura 4.1 – Arquitectura de um sistema baseado em *Web Services*.

As três entidades da arquitectura SOA são o cliente, o fornecedor do serviço e o corretor de serviços (*service broker*), que representam diferentes papeis:

- os clientes são os requisitantes dos serviços; são eles que fazem os pedidos aos fornecedores enviando os parâmetros de entradas, ordenando a sua execução e recebendo as respostas com os resultados;

- os fornecedores publicam os serviços e disponibilizam-nos aos clientes; aceitam ainda os pedidos dos clientes e executam-nos; são normalmente serviços em execução num computador endereçável numa rede;

- os corretores de serviços (*service brokers*) aproximam os clientes e os fornecedores, utilizando para isso um serviço de registo de serviços conhecidos, onde guardam as suas interfaces para que os clientes possam escolher o serviço adequado.

As arquitecturas SOA oferecem as seguintes acções: publicação, descoberta, mapeamento (*binding*) e execução:

- a publicação do serviço consiste no envio da descrição do serviço para o corretor de serviços;
- a descoberta é a acção de localizar um serviço que ofereça a funcionalidade que é pedida; envolve duas tarefas relacionadas: a descrição do *Web Services* tem que ser descoberta e tem que ser verificada a consistência com os requisitos pedidos;
- após a descoberta do serviço que satisfaz os requisitos do cliente, o corretor de serviços retorna informação sobre o fornecedor de serviços ao cliente que vai utilizar essa informação para invocar o serviço;
- quando o serviço é invocado, o cliente poderá ou não fornecer dados para a execução do serviço no servidor e por fim este retorna os resultados ao cliente.

4.2 *Web Services*

Os *Web Services* (W3C, 2002) estão entre as mais recentes evoluções de desenvolvimento de aplicações distribuídas. Esta tecnologia permite que as diferentes aplicações cooperem facilmente e partilhem informações e dados entre si. Os *Web Services* fornecem mecanismos normalizados de interoperabilidade entre diferentes aplicações de software, permitindo a sua execução numa grande variedade de plataformas e/ou *frameworks*. São caracterizados pela sua interoperabilidade e extensibilidade, assim como por descrições passíveis de processamento por computador (W3C, 2002). Podem ainda ser combinados de modo a obter operações complexas. Programas que disponibilizam serviços simples podem interagir com outros de modo a fornecer serviços acrescentados e sofisticados. Os *Web Services* permitem a utilização de serviços autónomos, identificados por URIs, com interfaces bem documentadas e processamento de mensagens e componentes básicas para a construção de arquitecturas SOA.

O conceito de *Web Services* envolve uma série de normas, tais como XML, SOAP, WSDL e UDDI. Tecnicamente os *Web Services* são serviços distribuídos invocáveis através da *Web* via mensagens XML que seguem a norma SOAP e que são enviadas através do protocolo HTTP. Um serviço pode fornecer uma ou mais operações para executar acções específicas e que podem ser invocadas pelos clientes. Estas operações e o formato das mensagens de entrada e saída são descritas utilizando a linguagem WSDL. A escolha desta linguagem neutra foi vital para o rápido crescimento da utilização dos *Web Services*. Um *Web Service* para que tenha um uso geral pode ser publicado num registo universal chamado UDDI, usado pelos fornecedores para anunciarem os seus serviços. Deste modo, os clientes podem pesquisar o serviço mais adequado às suas necessidades, procurando nos vários registos UDDI que subscreveram. O registo UDDI irá armazenar informação contida no descritor WSDL e informações sobre o fornecedor de serviços, entre outras.

É a combinação destes protocolos que preenchem os principais requisitos da SOA: a combinação de UDDI, WSDL e SOAP permite a descoberta e a invocação dinâmica de serviços; a linguagem XML permite que os serviços sejam independentes da plataforma. A linguagem WSDL oferece a visão lógica do serviço e o protocolo SOAP permite a interoperabilidade entre sistemas.

A Figura 4.2 ilustra as principais normas XML Schema que são essenciais para o desenvolvimento de soluções para arquitecturas SOA.

Todas estas normas e outras como o caso do *Namespaces* (Bray, Hollander et al., 2006) e XML Schema, são essenciais para o desenvolvimento de soluções para arquitecturas SOA são ilustradas na Figura 4.2 e são descritas em seguida.

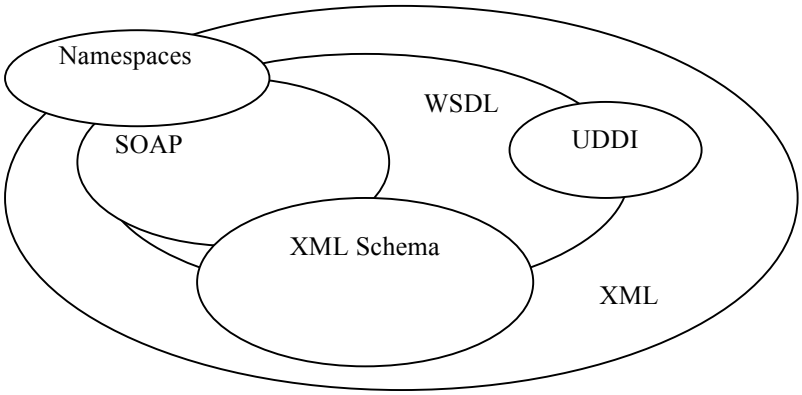


Figura 4.2 – Esquema das normas envolvidas nos *Web Services* (Lopes and Ramalho, 2005).

4.2.1 Extensible Markup Language

XML é uma linguagem de marcação de dados (*meta-markup language*) desenvolvida sob a orientação do W3C e que prevê um formato para descrever dados estruturados. É uma norma independente de qualquer plataforma de desenvolvimento e encontra-se na base dos restantes protocolos associados aos *Web Services*. Permite uma codificação flexível através da utilização de metadados para descrever a estrutura dos dados usando a linguagem *Document Type Definition* – DTD (Bray, Paoli et al., 1998) ou *XML Schema Definition* – XSD (Bray, Hollander et al., 2006). Um documento XML bem formado cria uma árvore de conjuntos de *tags*, em que cada conjunto pode incluir vários pares (*atributo*, *valor*).

4.2.2 Simple Object Access Protocol

O SOAP é um protocolo para a troca de informação estruturada num ambiente descentralizado e distribuído (Gudgin, Hadley et al., 2003). É uma norma que define o tipo e formato das mensagens baseadas em XML, que podem ser trocadas entre os intervenientes.

Não define um protocolo de transporte para a troca de mensagens para não comprometer a independência em relação à tecnologia de desenvolvimento. No entanto, possui uma descrição de como deve ser utilizado sobre HTTP, sendo esta a configuração mais amplamente utilizada.

4.2.3 Web Service Definition Language

A necessidade dos *Web Services* serem aplicações auto-descritivas levou à criação de uma linguagem normalizada que permitisse descreve-los. A norma WSDL (Chinnici, Moreau et al., 2007) é um formato XML publicado para descrever sintacticamente informação sobre *Web Services* e a forma de como os invocar. Permite a separação da descrição da funcionalidade oferecida pelo serviço dos detalhes da sua implementação, definindo a interface que esse serviço fornece aos requisitantes. O Código 4.1 corresponde a um documento WSDL que descreve um *Web Service* chamado *library*. Neste código é possível encontrar a operação que este *Web Service* fornece, parâmetro de entrada e de saída, e a sua localização, entre outra informação.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <definitions name="library"
3:    targetNamespace="http://services.icas.ipb.pt/wsdl/library"
```

```
4:     xmlns="http://schemas.xmlsoap.org/wsdl/"
5:     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6:     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7:     xmlns:tns="http://services.icas.ipb.pt/wsdl/library"
8:     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
9:     <types/>
10:    <message name="searchBookRequest">
11:       <part name="title" type="xsd:string"/>
12:    </message>
13:    <message name="searchBookResponse">
14:       <part name="isbn" type="xsd:string"/>
15:    </message>
16:    <portType name="libraryPortType">
17:      <operation name="searchBook">
18:         <input name="input" message="tns:searchBookRequest"/>
19:         <output name="output" message="tns:searchBookResponse"/>
20:      </operation>
21:     </portType>
22:     <binding name="libraryBinding" type="tns:libraryPortType">
23:       <soap:binding style="rpc" transport="http://sch...org/soap/http"/>
24:         <operation name="searchBook">
25:           <soap:operation/>
26:            <input name="input">
27:              <soap:body use="literal"
namespace="http://services.icas.ipb.pt/wsdl/library"/>
28:            </input>
29:            <output name="output">
30:              <soap:body use="literal"
namespace="http://services.icas.ipb.pt/wsdl/library"/>
31:            </output>
32:        </operation>
33:    </binding>
34:    <service name="libraryService">
35:       <port name="libraryPort" binding="tns:libraryBinding">
36:         <soap:address
location="http://localhost:${HttpDefaultPort}/libraryService/libraryPort"/>
37:       </port>
38:    </service>
39:</definitions>
```

Código 4.1 – Descrição de um *Web Service* em WSDL.

4.2.4 Universal Description, Discovery and Integration

O UDDI consiste numa base de dados de informação sobre *Web Services*, capaz de divulgar e descobrir os *Web Services*, tendo ainda a vantagem de ser independente da plataforma de desenvolvimento. O UDDI não é apenas um repositório de *Web Services*, é o registador de *Web Services* com três funções básicas (Lopes and Ramalho, 2005):

- divulgação – permite publicar informações sobre os *Web Services* e sobre os seus fornecedores;
- procura – os clientes usam estes registos para descobrir *Web Services* através das

descrições WSDL e/ou dados do seu fornecedor;

- mapeamento (*binding*) – oferece informação técnica que permite a uma aplicação cliente invocar um *Web Services* após a sua localização.

O UDDI define o conteúdo da informação e o tipo de acesso fornecido pelos serviços e utiliza XML Schema para descrever formalmente as suas estruturas de dados e define um *Web Service* baseado em SOAP para a localização das descrições WSDL.

4.2.5 Namespaces

A norma *Namespaces XML* (Bray, Hollander et al., 2006) funciona como uma etiqueta que permite identificar univocamente os diversos nomes dos elementos e atributos de um documento XML. Um *namespace* é definido através de um URI e é usado num documento XML como prefixo dos nomes dos elementos e atributos desse documento, garantindo assim, a unicidade necessária.

4.2.6 XML Schema

XML Schema é uma linguagem baseada em XML que permite a definição de regras de validação em documentos no formato XML, podendo assim descrever vocabulários partilhados permitindo às máquinas executarem as regras feitas por humanos. Fornece uma forma de definir a estrutura, conteúdo e semântica de documentos XML. O conceito de XML Schema prende-se com a definição da estrutura de um documento XML e surge como uma evolução do DTD. Ao contrário dos DTDs que possuem sintaxe própria, os dados XML Schema são especificados em XML proporcionando o seu processamento por processadores XML. Suporta ainda a utilização de *namespaces*, os tipos de dados primitivos das linguagens de programação mais populares e, como é extensível, permite ainda criar novos tipos de dados.

4.2.7 Business Process Execution Language for Web Services

Com o aparecimento dos *Web Services* e a rápida adesão para aplicações de negócios logo surgiu a necessidade de integrar aplicações dentro e fora das fronteiras das organizações, sendo necessárias novas linguagens para descreverem os comportamentos de dependência

entre as interacções dos serviços.

Várias empresas lançaram as suas próprias linguagens, por exemplo a IBM propôs a *Web Services* Flow Language – WSFL (Leymann, 2001), a Microsoft propôs a sua versão designada de *Web Services for Business Process Design* – XLANG (Thatte, 2001). Destas duas linguagens acabou por surgir a WSBPEL (Andrews, Curbera et al., 2003) que foi proposta em Abril de 2003 à Organization for the *Advancement of Structured Information Standards* (OASIS) para linguagem normalizada e foi criado o *Web Services Process Execution Language Technical Committee* (WSBPEL TC). Em Maio de 2007, o WSBPEL TC (OASIS, 2007) publicou a WSBPEL 2.0, em substituição da WSBPEL 1.1, como uma linguagem normalizada para a implementação de *Web Services* que usa um paradigma de orientação ao processo. Após a aceitação da WSBPEL pela OASIS rapidamente se concretizou a sua forte aceitação por parte da indústria, sendo actualmente uma das linguagens mais utilizadas na criação de processos utilizando *Web Services*.

A WSBPEL é um conjunto de camadas que funcionam em cima de várias especificações XML como XML Schema, WSDL e XPath. Fornece uma linguagem para a especificação formal de processos utilizando *Web Services*, possibilitando a composição, orquestração e coordenação de *Web Services* em arquitecturas SOA. O resultado da composição de *Web Services* é chamado de processo ou processo *Web*, os participantes são os parceiros, e a uma troca de mensagem ou resultado intermédio é chamada actividade. Para isso a WSBPEL define um modelo e uma gramática que descreve o comportamento de um processo de negócio baseado nas interacções entre processos e os parceiros. Entenda-se que um processo é composto por actividades que podem ser combinadas através de operadores estruturados (*invoking*, *waiting*, *sequence*, *while*, *etc*) e links de controlo (ex: sincronização de actividade concorrentes) para a criação do seu fluxo de trabalho. A WSBPEL fornece ainda tratamento de eventos, ocorrência de erros, entre outros (Andrews, Curbera et al., 2003).

4.3 Web Serviços Semânticos

Na secção anterior foi abordada a linguagem para a descrição de processos WSBPEL que fornece um modo de combinar a funcionalidade de *Web Services* numa sequência específica, com o objectivo de executar uma determinada tarefa mais complexa. A WSBPEL tem uma boa capacidade de modelação de processos, mas tem um défice de informação semântica

quando se pretende a automatização de processos BPEL. O principal motivo por que isso acontece prende-se com o facto da WSDL apenas especificar a funcionalidade do serviço a nível sintáctico, não contendo nenhuma informação semântica sobre esse mesmo serviço. Não é possível com a WSDL descrever o significado da funcionalidade do serviço a terceiros, de modo que estes façam uso dessa informação, o que pode levar a que os requisitantes não consigam encontrar o melhor serviço, quer por falta de informação quer por diferenças superficiais na especificação das interfaces, mesmo quando existam correspondências. Por exemplo, é fornecido um serviço que faz a soma de dois números. Embora a descrição sintáctica do serviço seja igual ou semelhante (é possível ter os mesmos parâmetros de entrada e saída), no entanto o resultado seria completamente diferente se tivéssemos a soma de números reais ou complexos. Utilizando a *Semantic Web,* é possível descrever uma ontologia para o domínio dos números e facilmente se criaria uma classe de números reais e outra de números complexos. Outra das limitações da WSBPEL é que não consegue expressar propriedades avançadas, como herança e relacionamentos entre *Web Services*, pré-condições necessárias para a utilização de um serviço e efeitos que esse serviço poderá produzir no seu meio com a sua execução.

Através da *Semantic Web* é possível lidar com estas limitações adicionando à camada de descrição de serviços uma camada semântica de modo a obter novas funcionalidades como a automatização da descoberta, composição, execução e monitorização. Segundo (Mcilraith, Son et al., 2001) *Web Services Semânticos* são *Web Services* cujas propriedades, capacidades, interfaces e efeitos são codificados de uma forma não ambígua e interpretável por máquinas.

A semântica contribui deste modo para um papel importante em todas as etapas do ciclo de vida dos *Web Services* (Sheth, 2003), ilustrado na Figura 4.3. Durante a fase de desenvolvimento o fornecedor de serviço pode explicar a semântica pretendida através da anotação de conceitos de um modelo semântico em determinadas partes dos *Web Services*. Como a utilização de domínios semânticos (ontologias) é possível obter consenso ao nível do significado semântico e uso dos termos, havendo menos ambiguidades na semântica oferecida pelo fornecedor de serviços.

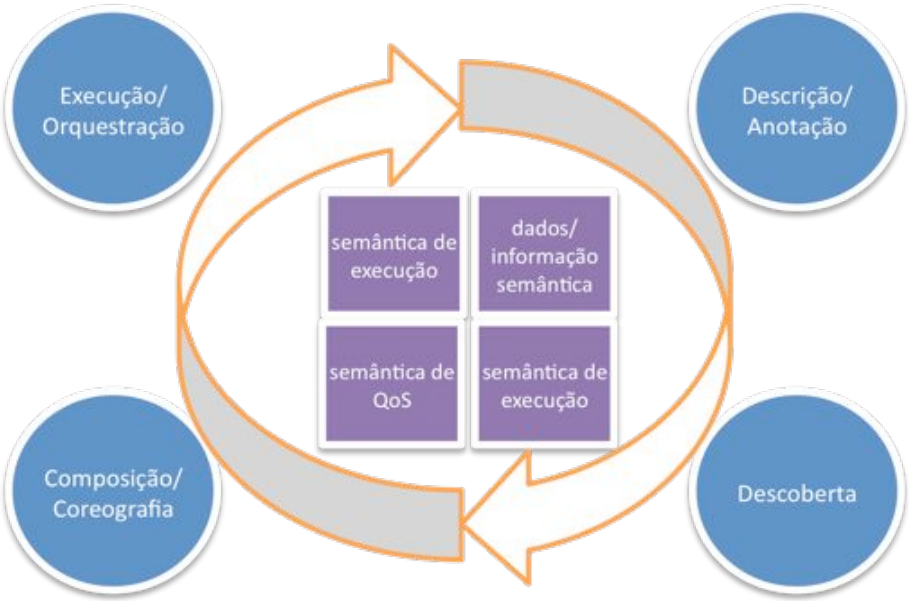


Figura 4.3 - Semântica e ciclo de vida de um processo *Web*, adaptado (Sheth, 2003).

Na fase de publicação as anotações funcionais poderão ser usadas para encontrar instâncias de serviços que mapeiem as suas interfaces. Na fase de descoberta de serviços, o requisitante terá a possibilidade de fazer pesquisas mais ricas através da utilização dos modelos semânticos e usufruir da utilização de técnicas de verificação e inferência avançadas para descobrir similaridades. Durante a composição o aspecto funcional das anotações pode ser utilizado para agregar funcionalidades de múltiplos serviços de modo a criar novas composições de serviços. Na invocação pode haver transformação de dados através do mapeamento de modelos semânticos, ou então, no caso da execução de algum serviço falhar, a semântica poderá ser usada para descobrir um novo serviço, avaliar as suas entradas, saídas, pré-condições e efeitos no processo e substituir o serviço que falhou.

Para utilizar um *Web Service* os agentes de software precisam de uma descrição interpretável do serviço. Esta funcionalidade será possível através da utilização de uma *framework* em que estas descrições são feitas e partilhadas. Essas interpretações são possíveis através da utilização de linguagens de descrição, combinadas com técnicas de inferências baseadas em ontologias (Martin, Burstein et al., 2004) (Mcilraith, Son et al., 2001):

- descoberta automática de *Web Services* – é um processo automático para a localização de *Web Services* que pode oferecer um determinado conjunto de características e ser capaz de se adaptar dinamicamente às restrições específicas exigidas pelos clientes;
- invocação automática de *Web Services* – é a invocação automática de um *Web*

Service por um programa ou agente, sendo dada uma descrição declarativa desse serviço. Por exemplo, na execução de um serviço composto por múltiplos serviços, o agente precisa de saber como interagir com o serviço de modo a terminar essa sequência. Num serviço baseado em OWL, é disponibilizada uma lista descritiva sobre o que os agentes precisam para serem capazes de executar e cumprir o serviço composto, incluindo as entradas e as saídas. Para isso, determinadas linguagens (ex, OWL-S) permitem especificar a semântica dos argumentos quando é feita a invocação de serviços e a semântica do que é retornado nas mensagens quando os serviços são bem sucedidos ou falham. Assim o agente deverá ser capaz de interpretar essas descrições para perceber quais as entradas necessárias para invocar o serviço e qual a informação que será retornada;

- composição e inter-operação automática de *Web Services* – esta tarefa deve permitir a selecção, composição e inter-operação automática de *Web Services* para atingir um objectivo. Os *Web Services* tradicionais possuem uma representação semântica muito limitada e deste modo o utilizador terá que seleccionar manualmente os serviços necessários para uma composição e fazer a correspondência de qualquer componente necessário para a inter-operação. Com a descrição semântica, a informação necessária para seleccionar e compor serviços será embebida no *Web Service*, o que permite que os agentes possam ser escritos para manipular essas representações que, juntamente com a especificação dos objectivos da tarefa, poderão atingir esse objectivo dessa tarefa e consequentemente a sua execução, isto tudo automaticamente. Para suportar esta característica, a maior parte das linguagens semânticas de serviço, disponibilizam especificações declarativas de pré-requisitos e das consequências da aplicação de serviços individuais, e uma linguagem para descrever a composição de serviços e fluxo de dados;

- monitorização da execução de *Web Services* – através de descritores para a execução de serviços é possível num determinado momento obter o estado da execução. Por exemplo, quando os serviços ficam indisponíveis, novos serviços podem ser descobertos e vinculados a uma determinada tarefa em substituição do serviço que falhou. Esta propriedade pode tornar as arquitecturas baseadas em *Web Services Semânticos* altamente robustas.

Na secção seguinte serão apresentadas as principais tecnologias para introduzir a semântica em *Web Services* e no final será feita uma breve discussão sobre as principais características de cada uma delas.

4.4 Tecnologias para a descrição semântica de *Web Services*

O W3C ao especificar a arquitectura de *Web Services* (Booth, Haas et al., 2004) define dois aspectos para descrever completamente um *Web Service*: a descrição sintáctica e a descrição semântica. Enquanto que a primeira é representada utilizando a linguagem WSDL, já a segunda é definida como a semântica do serviço, mas não é objecto de especificação. Isto é, na prática a descrição semântica não aparece na especificação, não havendo indicações sobre como descrever a informação semântica dos serviços. De modo a resolver os problemas anteriores, várias iniciativas forma de linguagens ou projectos foram iniciadas nos últimos anos, tais como: OWL-S (Martin, Burstein et al., 2004), WSMO (WSMO, 2004), METEOR-S (Verma, Sivashanmugam et al., 2003) que viria a dar origem à linguagem WSDL-S (Akkiraju, Farrell et al., 2005), e SWSA (SWSA, 2004) que depois viria dar origem à linguagem *Semantic Web Services Language* – SWSL (Battle, Bernstein et al., 2005). Estas tecnologias serão apresentadas nas secções seguintes.

4.4.1 Semantic Markup for Web Services

A OWL-S (Martin, Burstein et al., 2004) surge do projecto *DARPA Agent Markup Language* – DAML (DAML, 2000) e foi submetida ao W3C em Novembro de 2004 na versão 1.1. A OWL-S é uma ontologia para a descrição de *Web Services* baseada em OWL que oferece aos fornecedores de *Web Services* um conjunto de construtores de linguagens de marcação, para descrever propriedades e capacidades dos *Web Services*, de uma forma inequívoca e interpretável por agentes de software. A utilização do OWL-S nos *Web Services* facilita a automatização de tarefas com estes, incluindo a descoberta, execução, composição e inter-operação.

Para que a semântica de *Web Services* se torne uma realidade, a linguagem de marcação deve ser descritiva suficientemente de forma a que um agente de software possa determinar o seu

significado. Por isso a OWL-S foi desenvolvida para ser capaz de fornecer a automatização da descoberta, invocação, composição e inter-operação de *Web Services*. A OWL-S irá permitir aos utilizadores e agentes de software localizar, seleccionar, executar, compor e monitorizar *Web Services* de um modo automático.

A estruturação da ontologia OWL-S é dividida conceptualmente em quatro sub-ontologias de forma a descrever três características de conhecimento sobre os serviços:

- "o que faz o serviço?" – para responder a esta pergunta é utilizada a ontologia *Profile*. Para capturar esta perspectiva, cada instância da classe *Service* apresenta através da propriedade *presents* uma classe *ServiceProfile;*
- "como é utilizado?" – quem descreve esta informação é a ontologia *Process*. Esta perspectiva é capturada pela classe *ServiceModel*. Instâncias da classe *Service* usam a propriedade *describedBy* para se referir à classe *ServiceModel* do serviço;
- "como se interage com ele?" – quem contém essas informações é a ontologia *Grounding*. Fornece detalhes sobre o serviço (ex: formato das mensagens, protocolo de transporte, endereços). Instâncias da classe *Service* tem uma propriedade *supports* que se refere à classe *ServiceGrounding*.

A quarta ontologia *Service* contém o conceito *Service* que junta os conceitos *ServiceProfile*, *ServiceModel* e *ServiceGrounding*, através das propriedades *presents*, *describedBy* e *supports*, como ilustra a Figura 4.4.

As classes que identificam cada uma das sub-ontologias são: a classe *ServiceProfile* que identifica a sub-ontologia *Profile*, *ServiceModel* que identifica a sub-ontologia *Process* e *ServiceGrounding* que identifica a sub-ontologia *Service Grounding*.

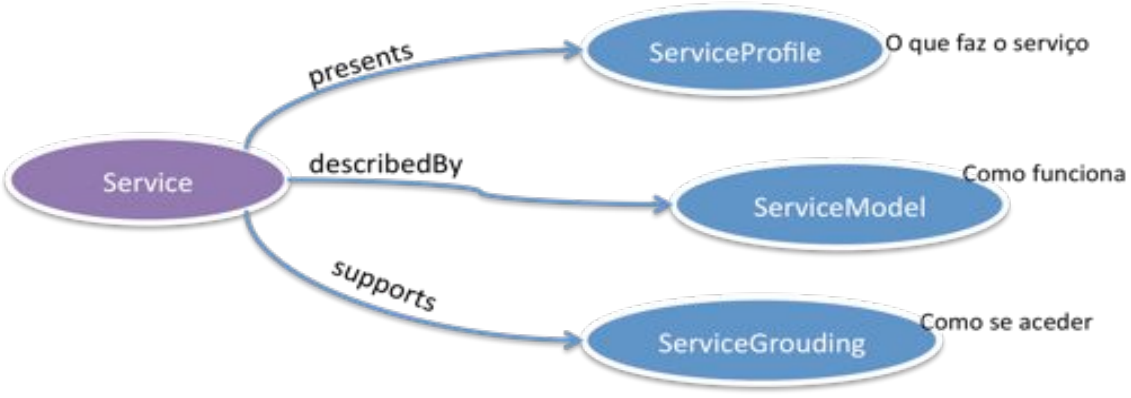


Figura 4.4 – Nível de topo da ontologia de serviços (Martin, Burstein et al., 2004).

Na restante secção serão apresentadas com mais pormenor cada uma das sub-ontologias enunciadas anteriormente.

4.4.1.1 Profile

A sub-ontologia *Profile* especifica como o serviço se anuncia ao mundo. Fornece informações que permitem a descoberta e verificação por agentes de software. Isto é, diz o que faz o serviço, quais os requisitos para ser executado com sucesso, quais as suas limitações e pode fornecer características para avaliar a qualidade de serviço (QoS). Esta sub-ontologia foi modelada para fornecer três tipos de informações: (1) informações sobre a organização que fornece o serviço; (2) a função que o serviço fornece, inclui informação sobre as entradas e os resultados que ele produz; quais os pré-requisitos para executar o serviço e os efeitos que se pode esperar com a sua execução; (3) descrição sobre as propriedades do serviço, como por exemplo a categoria do serviço segundo o sistema de classificação UNSPSC (UNSPSC, 1998), avaliação da qualidade do serviço, entre outros.

A informação essencial que no perfil do serviço vai ter um papel chave durante o processo de descoberta de serviços é a especificação da funcionalidade do serviço. A funcionalidade do serviço é descrita através da descrição das entradas/saídas/pré-condições/efeitos (IOPE). Os dois primeiros representam informação de transformação e o dois últimos representam informação de alteração de estado. Na Tabela 4.1 são apresentadas as propriedades da sub-ontologia *profile* que permitem cria o perfil de um serviço OWL-S.

Tabela 4.1 – Propriedades da sub-ontologia *Profile*.

Propriedades	Descrição resumida
presents	Descreve a relação entre a instância do serviço e a instância do perfil. Basicamente diz o que o serviço é descrito através de um perfil
presentedBy	É o inverso da propriedade anterior
ServiceName	Nome do serviço que é oferecido
textDescription	Breve descrição sobre o serviço
contactInformation	Informação sobre os responsáveis do serviço
hasParameter	Parâmetro do serviço (inclui entradas e saídas)
hasInput	Entradas do serviço
hasOutput	Saídas do serviço
hasPrecondition	Pré-condições do serviço
hasResult	Especifica um dos resultados do serviço
serviceParameter	Lista expansível de propriedades que podem acompanhar uma descrição do perfil
serviceCategory	Referência a uma ontologia ou taxonomia de serviços
serviceParameterName	É o nome do parâmetro actual
sParameter	Aponta para um valor de uma parâmetro que pertence a uma ontologia OWL
categoryName	Nome da categoria actual
taxonomy	Referência a uma taxonomia
Value	Aponta para um valor numa taxonomia
Code	Código associado a uma taxonomia
serviceClassification	Define um mapeamento para uma ontologia de serviços, como uma especificação NAICS (NAICS, 1992)
serviceProduct	Define o mapeamento para uma ontologia de produtos, como uma especificação UNSPSC

Um serviço OWL-S pode ter zero ou mais perfis, isto é zero ou mais instâncias da classe *ServiceProfile* ilustrada na Figura 4.5.

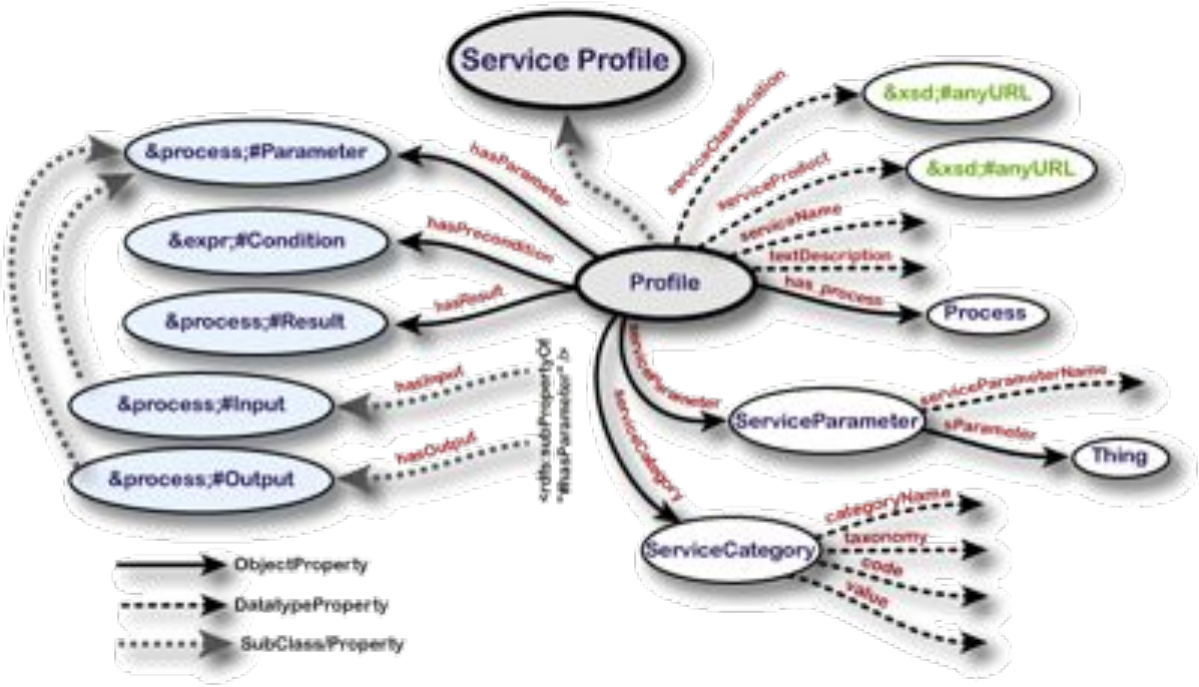


Figura 4.5 – Classes e propriedades da sub-ontologia *Profile* (Martin, Burstein et al., 2004).

4.4.1.2 Process

A ontologia *Process* tem como objectivo descrever como o serviço trabalha. Para isso descreve as entradas, saídas, pré-condições e resultados da execução do serviço. A principal utilização do *Service Model* é permitir a invocação, publicação, composição, monitorização e recuperação (caso a execução de um serviço falhe) de um *Web Service*. O modelo de serviço vê a iteração de um serviço como um processo. É importante referir que um processo não é um programa a ser executado, mas antes uma especificação dos modos como um cliente interage com o serviço. A Figura 4.6 mostra as classes e propriedades do modelo de processo.

Um processo envolve pelo menos duas partes: o cliente e o serviço. Ambos são referidos como participantes e são directamente ligados ao processo usando a propriedade *hasParticipant*. As entradas e saídas especificam a transformação de dados produzido pelo processo e estão ligadas ao processo através da utilização das propriedades *hasInput* e *hasOutput*. As entradas especificam a informação que o processo necessita para a sua execução e podem vir directamente do cliente ou de outro serviço. As saídas especificam a informação que é gerada após a execução. A presença de uma pré-condição significa que o processo não pode ser executado a não ser que esta seja satisfeita. As pré-condições estão ligadas aos processos através da propriedade *hasPrecondition*. A execução do processo pode provocar a alteração de estado e a geração de informação por parte do *Web Service*. Essas

saídas e efeitos estão ligadas ao processo através da propriedade *hasResult*.

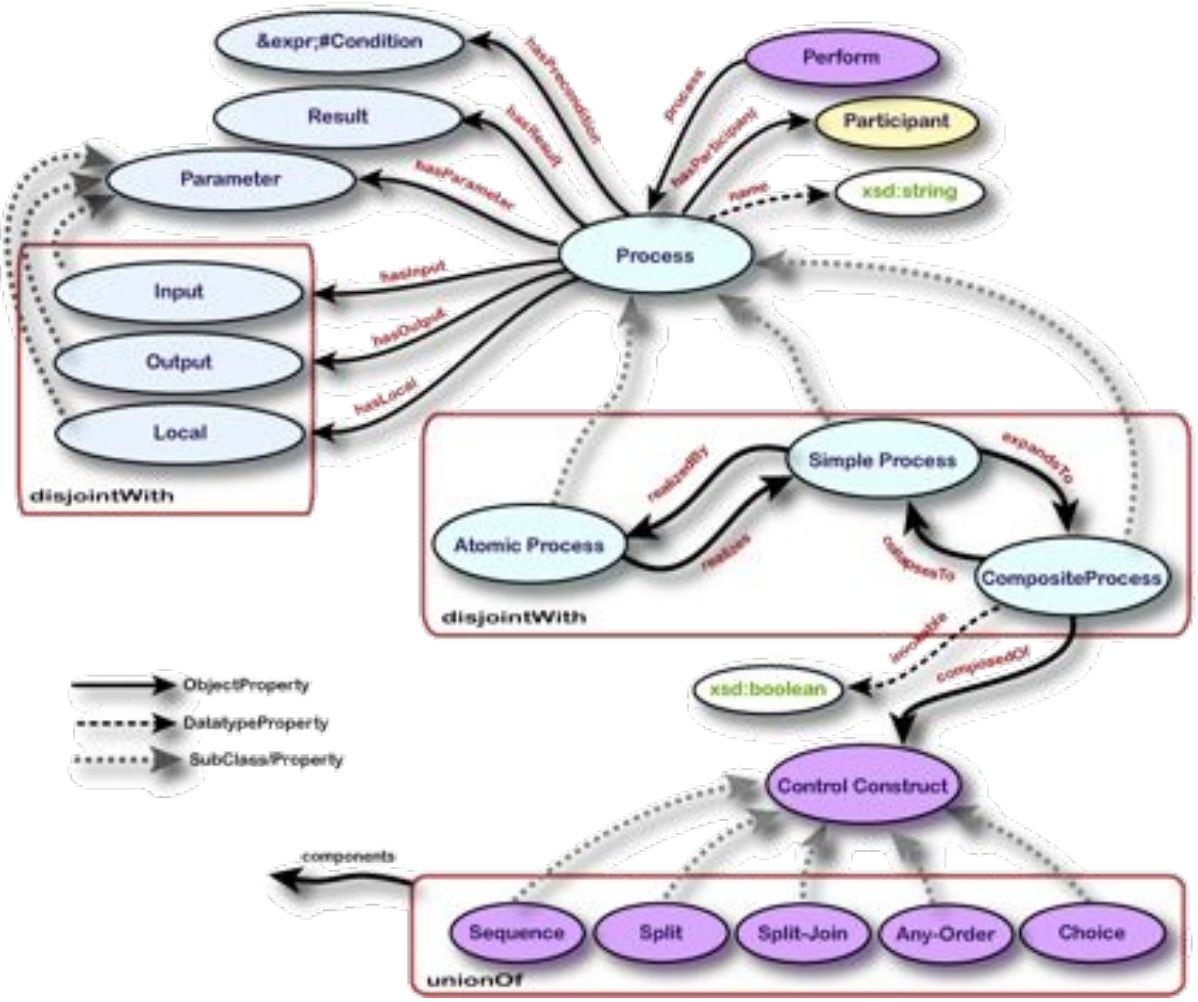


Figura 4.6 – Ontologia do modelo de processo (Martin, Burstein et al., 2004).

Existem três tipos de processos identificados pelo OWL-S: (1) processo atómico, que corresponde às acções que um serviço pode realizar numa única iteração; (2) processo simples, que apenas fornece um mecanismo de abstracção para fornecer múltiplas vistas (modo de utilização) do mesmo processo, vistas estas que podem ser usadas para fornecer uma vista diferente de um processo atómico; (3) processo composto, é um processo que pode ser decomposto noutros processos (compostos, simples ou atómicos. São construídos a partir de processos atómicos e/ou simples e através do uso de construtores de controlo (Tabela 4.2), mais a informação adicional sobre o fluxo de dados (ex: que saídas são encaminhas para determinadas entradas).

Tabela 4.2 – Construtores de controlo OWL-S.

Construtores	Descrição resumida dos construtores de controlo
Sequence	Conjunto de processos a serem executados sequencialmente
Split	Conjunto de processos a serem executados concorrentemente
Split+Join	Consiste numa execução concorrente com um ponto de sincronização de processos
Choise	Invoca para execução um único controlo de um grupo de construtores de controlo
Any-Order	Permite a execução aleatória de componentes de processos, mas não concorrentemente
If-Then-Else	Estrutura de controlo, que tem as propriedades ifCondition, then e else. Significa testar a condição e se verdadeira então Verdadeiro senão Falso
Iterate	Estrutura de controlo que executa diversas vezes um processo. Não faz referência a quantas iterações são feitas, quando acabar/finalizar ou retomar uma pausa
Repeat-While	Iteração enquanto um determinada condição é falsa ou verdadeira
Repeat-Until	Semelhante ao anterior

Existe uma relação entre as sub-ontologias *Profile* e *Process*. Como já foi dito, a *Profile* consiste na descrição do serviço para o anunciar, mas após este ser seleccionado essa descrição não é mais utilizada. Em vez disso, o cliente vai usar a sub-ontologia *Process* para controlar a iteração com o serviço. Embora a *Profile* e a *Process* tenham papeis diferentes durante a transacção com o *Web Service*, elas são representações diferentes do mesmo serviço. É natural esperar que as IOPEs de uma sejam reflectidas nas IOPEs da outra.

4.4.1.3 Grounding

A ontologia *Grounding* fornece o vocabulário para ligar a descrição conceptual do serviço especificado pelas sub-ontologias *Profile* e *Process* aos detalhes da implementação. Fazem parte desses detalhes os protocolos de comunicação, formato das mensagens e portas utilizadas para o contactar. Estas informações indicam o modo como um agente pode aceder a um serviço.

O *ServiceGrounding* estabelece a ponte entre o *Semantic Web Service* e uma implementação concreta do *Web Service* descrito em WSDL. O mapeamento entre o *ServiceGrounding* do serviço OWL-S e a descrição WSDL é feita de acordo com três regras :

- a um processo atómico corresponde uma operação WSDL;

- cada entrada de um *AtomicProcess* é mapeada numa mensagem de entrada de uma operação WSDL. O mesmo se passa para as saídas, cada saída de um *AtomicProcess* é mapeada a uma mensagem de saída da operação WSDL;
- as entradas e saídas do processo atómico OWL-S podem ser descritas utilizando tipos de dados XML Schema ou um conceito OWL (ex. classes OWL).

A OWL-S permite o mapeamento de um processo atómico numa operação WSDL e as suas entradas e saídas em mensagens WSDL. Quando é feito este mapeamento podem acontecer duas situações: (1) essas entradas e/ou saídas são do tipo de dados simples do XML Schema (ex. string, inteiros) e o mapeamento é directo; (2) se forem tipos de dados complexos (ex. estruturados) há necessidade de usar transformações XSLT nesse mapeamento, para fazer a transformação dos dados representados em OWL para o XML Schema utilizado na WSDL. Por exemplo, quando se pretende transformar uma descrição WSDL que utilize estruturas complexas em XML Schema para uma descrição OWL-S cuja estrutura terá que ser transformada numa ontologia. Embora WSDL seja a única tecnologia definida em *grounding* para a OWL-S, no entanto, esta última não está restrita apenas à WSDL como tecnologia de serviços, e deve ser percebida como uma ontologia genérica para descrição de serviços que é extensível a outros mecanismos de *grounding*.

4.4.1.4 Criar um Serviço OWL-S

Em (Alesso, 2006) é descrito o processo de criação de um *Web Service Semântico* utilizando a OWL-S através de cinco passos fundamentais:

1. descrever a funcionalidade que o serviço oferece – o modelo de processo fornece a descrição declarativa para as propriedades do serviço;
2. descrever o *grounding* de cada processo atómico – deve-se relacionar cada processo atómico com o seu *grounding*;
3. descrever composição de processo atómicos – descrever o processo composto que é uma composição de processos atómicos;
4. descrever o processo simples para o serviço (opcional);
5. descrever o perfil do serviço que é parcialmente preenchido pelas propriedades já declaradas no primeiro passo.

4.4.2 Web Service Modeling Ontology

A WSMO (Bruijn, Bussler et al., 2005) fornece uma *framework* conceptual e uma linguagem formal para descrever semanticamente os *Web Services* de modo a facilitar a automação da descoberta, combinação e invocação de serviços através da *Web*. A WSMO nasceu do grupo de trabalho ESSI WSMO (ESSI, 2004) que fez parte do ESSI Cluster (ESSI, 2004). O objectivo deste grupo era desenvolver esforços na área da *Semantic Web* entre diversos projectos europeus, trabalhar na normalização de linguagens na área de *Semantic Web* e trabalhar numa arquitectura e plataforma comuns para *Web Services Semânticos*.

A proposta WSMO é composta por três componentes descritos na Figura 4.7: o modelo conceptual para *Web Services Semânticos* – WSMO, a linguagem *Web Service Modeling Language* – WSML (Bruijn, Fensel et al., 2005), que fornece uma sintaxe formal e semântica para o WSMO, e o *Web Service Modeling Execution Environment* – WSMX (Bussler, Cimpian et al., 2005), que proporciona um ambiente de execução de *Web Services Semânticos*.

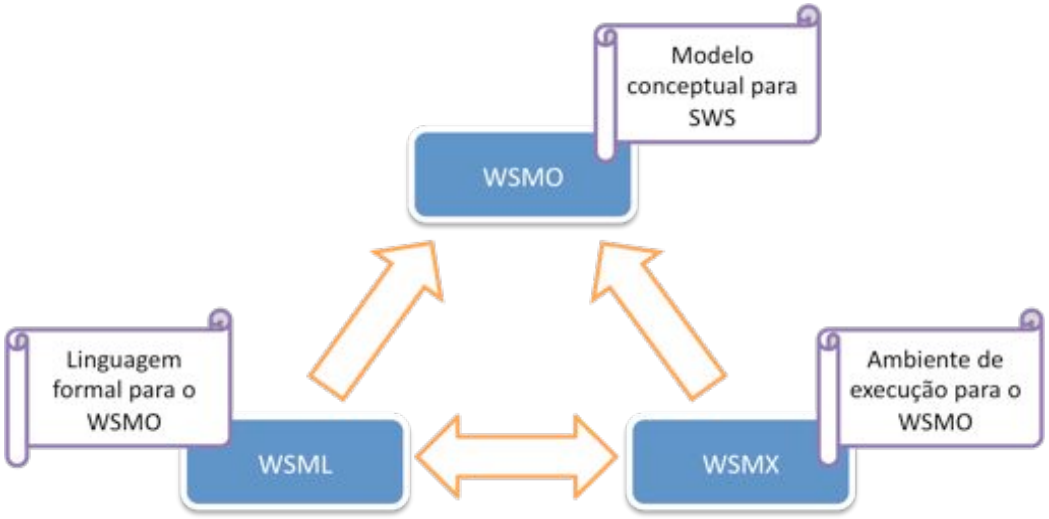


Figura 4.7 – Proposta da WSMO para o suporte de *Web Services Semânticos*.

4.4.2.1 Modelo Conceptual

Segundo a recomendação, uma *framework* para os *Web Services Semânticos* terá de integrar princípios básicos de desenvolvimento da *Web* e da *Semantic Web*, assim como princípios para computação distribuída e orientada aos serviços. Para isso a WSMO foi desenvolvida com base nos seguintes princípios (Bruijn, Bussler et al., 2005):

- conformidade com a *Web* – para tal, utiliza para a identificação única de recursos

um URI e adopta a utilização de *namespaces* e tecnologias baseadas em XML;

- baseada em ontologias – as ontologias são usadas como modelos de dados; qualquer descrição ou troca de dados durante a utilização de um serviço são baseadas em ontologias;
- dissociação restrita (*Strict Decoupling*) – todos os recursos WSMO são definidos isoladamente, de modo a seguir a natureza distribuída da *Web*;
- mediação centralizada – a mediação aborda a heterogeneidade natural em ambientes abertos;
- separação das funções ontológicas – os intervenientes (clientes) existem em contextos específicos que podem não ser os mesmos para os *Web Services*; possibilita diferenciar as necessidades dos utilizadores, clientes e *Web Services*;
- descrição *versus* implementação – a WSMO visa fornecer um modelo de descrição ontológico e ser compatível com tecnologias existentes e emergentes; para isso diferencia entre a descrição dos *Web Services* e as tecnologias de implementação;
- execução semântica – de modo a verificar a especificação WSMO; permite a execução formal semântica de implementações como o WSMX ou outros sistemas;
- service *versus Web Service* – fornece modos para descrever *Web Services* que fornecem acesso a serviços (procura, compra, etc).

A WSMO identifica quatro elementos de nível de topo como os principais conceitos para descrever de forma correcta os *Web Services Semânticos* (Bruijn, Bussler et al., 2005):

- ontologias – fornecem as terminologias usadas por outros elementos WSMO para descrever aspectos importantes do domínio;
- *Web Services* – descrevem uma entidade computacional proporcionando o acesso a serviços que fornecem algum valor num domínio;
- objectivos – representam os desejos do utilizador, para que a sua realização possa ser satisfeita através da execução de um *Web Service*; os aspectos relevantes podem ser descritos recorrendo às ontologias de domínio;
- mediadores – descrevem elementos que superam problemas de interoperabilidade entre diferentes elementos WSMO; são o núcleo do conceito para resolver

incompatibilidade de dados (diferenças entre terminologias), processos (combinação de *Web Services* e objectivos) e protocolos (comunicação entre *Web Services*).

4.4.2.2 Linguagem de modelação

Como foi descrito até agora a WSMO propõe um modelo conceptual para a descrição de ontologias, *Web Services Semânticos*, objectivos e mediadores. Para descrever formalmente estes elementos foi desenvolvida a WSML.

Esta linguagem fornece uma *framework* de unificação de tecnologias *Web* com diferentes linguagens lógicas, de modo a permitir a descrição de *Web Services Semânticos*. A partir de vários formalismos lógicos, como a *Description Logics*, *First Order Logic* e *Logic Programming*, desenvolveram um número de linguagens variantes da WSML, ilustradas na Figura 4.8. São a WSML-Core, WSML-DL, WSML-Flight, WSML-Rule e a WSML-Full, a variante mais expressiva. Uma descrição mais pormenorizada sobre a WSML e as suas variantes pode ser encontrada em (Bruijn, Fensel et al., 2005).

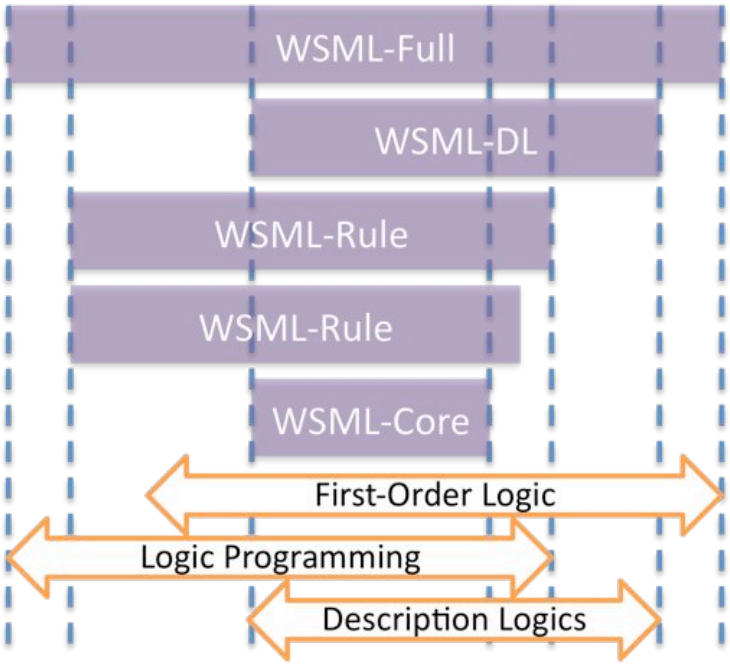


Figura 4.8 – Variantes do WSML e a sua posição lógica (Bruijn, Fensel et al., 2005).

4.4.2.3 Ambiente de Execução

A WSMX é uma arquitectura que suporta a implementação completa do modelo conceptual do WSMO e é ao mesmo tempo uma implementação de referência deste, onde é especificada

a arquitectura, os componentes do sistema, as suas interfaces e o sistema de execução semântica. A arquitectura WSMX consiste num conjunto de componentes fracamente ligados e que podem ser adicionados ou removidos do sistema. A WSMX fornece a implementação completa de todos os componentes, mas os utilizadores podem usar componentes fornecidos por terceiros.

4.4.3 Semantic Web Services Framework

A Semantic Web Services Framework (Battle, Bernstein et al., 2005) é uma proposta relativamente recente e oferece uma nova aproximação para a notação de *Web Services Semânticos*. Foi baseada em trabalhos prévios da área com destaque para a OWL-S e a *Process Specification Language* – PSL e beneficiando de trabalhos prévios na área dos *Web Services Semânticos* e foi submetida ao W3C em Setembro de 2005.

A SWSF é baseada em dois componentes: (1) uma ontologia ou modelo conceptual correspondente, através dos quais os *Web Services* podem ser descritos, chamada *Semantic Web Services Ontology* – SWSO (Battle, Bernstein et al., 2005); (2) uma linguagem usada para especificar caracterizações formais de conceitos de *Web Services* e descrições chamada *Semantic Web Services Language* – SWSL (Battle, Bernstein et al., 2005).

4.4.3.1 Modelo Conceptual

O modelo conceptual apresentado pela SWSO descreve os *Web Services* como um conjuntos de axiomas ou uma caracterização formal deste modelo, numa das duas variantes do SWSL: a SWSL-FOL baseado na *First Order Logic*, ou a SWSL-Rules baseado em *Logic Programming*. As ontologias resultantes são chamadas *First Order Logic Ontology for Web Services* (FLOWS) que se baseiam na semântica oferecida pela *First Order Logic*, e a *Rule Ontology for Web Services* (ROWS) que se baseia na semântica *Logic Programming*.

Nesta secção apenas será descrita a ontologia FLOWS, já que a dedução desta para a ontologia ROWS será directa, porque ambas partilham o mesmo modelo conceptual, diferindo no modo como é expressa a semântica. A ontologia FLOWS foi fortemente influenciada pela ontologia OWL-S, mas com a particularidade de fornecer um modelo de processo mais rico baseado na linguagem PSL. A ontologia FLOWS consiste em três componentes principais: *Service Descriptors*, *Process Model* e *Grounding*. *Service Description* é usada para descrever de forma básica o serviço; *Process Model* tem como função descrever como o serviço

funciona; *Grounding*[4] é utilizado para ligar a descrição das actividades dos serviços com as instâncias, detalhando o formato das mensagens, protocolos de transporte e endereços de rede, para que se possa chegar ao *Web Service*.

Embora havendo bastantes semelhanças entre as ontologias FLOWS e a OWL-S, existe uma grande diferença que é a expressividade que cada uma delas suporta. Enquanto a OWL-S é baseado em OWL-DL a FLOWS é baseada em *First-Order Logic*, que é mais rica e consequentemente mais expressiva.

4.4.3.2 Linguagem

A SWSL é uma linguagem para descrever conceitos de *Web Services* e para a descrição de serviços individuais (Battle, Bernstein et al., 2005). A SWSL, como já foi anteriormente referido, consiste em duas partes: SWSL-FOL e SWSL-Rules. Ambas as linguagens são consideradas linguagens por camadas em que cada camada inclui um número de novos conceitos que realçam o poder de modelação da linguagem. Foram desenvolvidas em conformidade com os princípios da *Web*, utilizando URIs, *namespaces* e linguagens baseadas em XML.

A linguagem SWSL foi desenhada para fornecer suporte a uma variedade de tarefas que vão desde a especificação do perfil do serviço até à sua descoberta, contratos, especificação de políticas, entre outras.

A linguagem SWSL-FOL é usada para especificar as propriedades dinâmicas dos serviços, nomeadamente os processos que eles pretendem suportar. Mais informações (como a sintaxe e metodologias de expressividade) de cada uma destas linguagens podem ser encontradas em (Battle, Bernstein et al., 2005).

4.4.4 Web Service Semantics

A linguagem WSDL-S (Akkiraju, Farrell et al., 2005) tem as suas origens no projecto

[4] O *grounding* também conhecido como um conceito genérico comum a todos os serviços semânticos e que corresponde à ligação entre a descrição semântica e a descrição sintáctica. Este mecanismo permite o desenvolvimento de SWS a partir de tecnologias existentes (ex.: WSDL), de modo a que possam ser amplamente adoptados.

METEOR-S (METEOR-S, 2003) da Universidade da Geórgia. Mais tarde, após ter sido revista pela IBM e pela equipa do METEOR-S, foi submetida ao W3C em Novembro de 2005. A WSDL-S propõe um mecanismo para aumentar as descrições funcionais dos *Web Services* representadas pela WSDL com a semântica (representada pela letra S no final do acrónimo). Comparado com a WSMO, SWSF e OWL-S, a WSDL-S é uma abordagem minimalista que visa uma extensão directa *Web Services* dos já existentes descritos em WSDL com semântica. Assim os programadores podem adicionar anotações semânticas aos documentos WSDL usando a *framework* de extensibilidade definida na especificação WSDL. Estas anotações semânticas podem ser referentes a conceitos definidos por ontologias externas. Através da utilização de elementos de extensibilidade WSDL é possível criar um conjunto de anotações para definir a categoria do serviço numa ontologia externa e descrever semanticamente as entradas/saídas e as operações de um *Web Service*. A WSDL-S fornece ainda mecanismos para especificar e anotar pré-condições e efeitos da execução de *Web Services,* que juntamente com as anotações de entradas e saídas possibilitam a automatização do processo de descoberta de serviços. A Figura 4.9 ilustra como o modelo de domínio se mantém separado do modelo de serviço e mostra as associações entre os conceitos WSDL e as suas correspondentes anotações semânticas, que são mantidas usando referências URI.

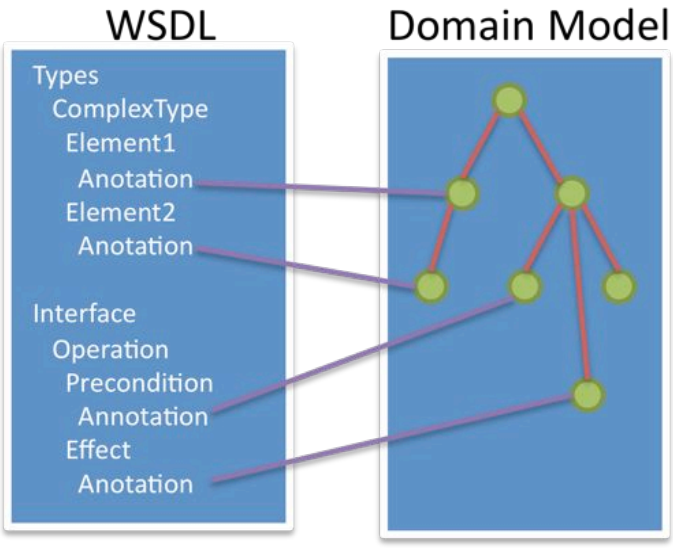


Figura 4.9 – Associação semântica a elementos WSDL.

Ao contrário da WSMO, SWSF e OWL-S, a WSDL não especifica o uso de uma linguagem ontológica em particular pois foi definida para ser agnóstica em relação à representação semântica da linguagem. Os autores da WSDL-S mencionam como possíveis candidatos a WSML, OWL e a UML. A vantagem desta aproximação é que é construída em cima de

normas largamente utilizados na indústria e matem-se próxima deles. Os principias objectivos da WSDL-S são:

- construção baseada em normas existentes dos *Web Services* e promove um mecanismo de extensão compatível para adicionar semântica aos *Web Services*;
- as anotações deverão ser agnósticas à representação semântica da linguagem podendo ser usadas diferentes representações de descrição semântica nos serviços. Como tal, a WSDL-S não especifica que linguagem de representação semântica deverá ser usada e permite a associação de múltiplas anotações escritas em diferentes linguagens de representação semântica;
- suporte de anotações do tipo XML Schema, é desejável reutilizar as interfaces já existentes e descritas em XML. A WSDL-S suporta as anotações de XML Schemas, que são utilizadas para adicionar semântica às entradas e saídas dos *Web Services*. Em alguns serviços poderão ser utilizadas nas entradas/saída dos serviços tipos de dados complexos. Neste caso o mapeamento entre a XML Schema dos tipos complexos e os conceitos ontológicos correspondentes irão estar directamente dependentes da linguagem de representação semântica escolhida. Isto porque a WSDL-S não define a linguagem ontológica a ser utilizada.

A WSDL-S propõe cinco elementos de extensibilidade para a anotação de entradas, saídas e operações de *Web Services*, para serem usados para criar anotações:

- *modelReference* – denota o mapeamento um para um entre os elementos do *schema* e os conceitos da ontologia;
- *schema*Mapping – pode ser adicionado a elementos XSD ou tipos complexos para os associar a uma ontologia;
- *precondition* e *effect* – são usados principalmente na descoberta de serviço e não são exclusivamente necessários para invocar um determinado serviço;
- *category* – consiste em atribuir uma informação de categoria ao serviço que pode ser usado quando este é publicado num registo de *Web Services* como o UDDI.

4.4.5 Análise Comparativa

Nas secções anteriores foram apresentadas as tecnologias mais importantes destacando-se os

principais conceitos que elas definem e que são agora resumidas na Tabela 4.3. Todas estas tecnologias foram submetidas ao W3C como propostas alternativas para definir uma *framework* que suporte *Web Services Semânticos*.

A OWL-S propõe uma *framework* baseada em OWL. Define uma ontologia superior para descrever as propriedades e capacidades dos *Web Services* cuja expressividade é baseada em *Description Logic*. A descrição dos *Web Services Semânticos* é feita utilizando três sub-ontologias: a ontologia *profile,* que exprime o que o serviço faz, a ontologia *process,* que diz como ele deve ser utilizado e a ontologia *grounding* que descreve como se interage com ele.

A proposta WSMO oferece uma *framework* bem definida e um modelo conceptual (WSMO) para definir os conceitos básicos dos *Web Services Semânticos*, uma linguagem formal (WSML) que proporciona a sintaxe e a semântica para o WSMO, suportando diferentes níveis de expressividade, e uma implementação de referência (WSMX) para o WSMO. A *framework* SWSF oferece um modelo conceptual para descrever *Web Services* e uma linguagem semântica (SWSL) para caracterizar conceitos de *Web Services* e descrever instâncias destes. Tal como a linguagem WSML, a SWSL também fornece diferentes níveis de expressividade. A WSDL-S propõe uma aproximação mais centrada na tecnologia, alinhando-se com a norma da indústria, a WSDL. Exclui da sua especificação o modelo conceptual e uma solução completa para o problema de integração e prefere um aproximação de anotação de padrões existentes com metadados. Enquanto outras tecnologias como o WSMO, SWSF e OWL-S estão restringidas à utilização das linguagens semânticas a WSML, SWSL e OWL, a WSDL-S é ateísta em relação às linguagens ontológicas, podendo trabalhar com qualquer linguagem ontológica porque as anotações são externas.

Tabela 4.3 – Resumo das principais aproximações para a descrição de *Web Services Semânticos*, baseada em (Sheth, Verma et al., 2006).

Tecnologia	Descrição	Linguagem Semântica	Formalismo	Relação com o WSDL
WSMO	Fornece uma *framework* conceptual (WSMO), e uma linguagem formal para a descrição semântica de *Web Services* (WSML) e um ambiente de execução (WSMX)	WSML	Description Logic, First-Order Logic e Logic Programming	Elementos WSDL estendem descrições de funcionalidade do WSMO. WSMO estende a semântica de implementações WSDL. A ligação é feita via *Grounding*
SWSF	Fornece dois componentes: um modelo conceptual SWSO e a linguagem semântica SWSL.	SWSL	Se utilizar SWSL-FOL é *First Order Logic*. Se utilizar SWSL-Rules é *Logic Programming*	Como a FLOWS é semelhante à OWL-S, utiliza o mesmo mecanismo de *Grounding*
WSDL-S	Permite adicionar anotações semânticas utilizando elementos de extensibilidade WSDL	Agnóstico à linguagem ontológica; pode trabalhar com OWL, WSMO, UML, XML	Depende da linguagem ontológica escolhida	Faz parte do WSDL. São feitas anotações nas descrições WSDL
OWL-S	Utiliza quatro ontologias OWL para descrever *Web Services*	OWL	Description Logic	Utiliza o *Grounding* para estabelecer a ligação ao WSDL

4.5 Semantic Context Model

O SeCoM (Bulcão Neto and Pimentel, 2005; Bulcão Neto and Pimentel, 2006) apresenta um modelo contextual semântico, que representa a semântica da informação contextual através de um conjunto de ontologias. Este modelo foi desenvolvido a partir das seguintes normas de metadados: *Dublin Core*, *vCard* e *iCalendar* e das ontologias *SUMO*, *OpenCyc*, *FOAF*, *SWEET*, *CC/PP* e *OWL-Time* e oferece as seguintes características:

- modela os conceitos básicos de contexto como a identidade, localização, data, eventos, actividades e dispositivos;
- fornece um modelo independente do domínio para sistemas *context aware*;

- oferece uma semântica com um alto nível de expressividade e formalismo através da Lógica de Descritiva (DL – *Description Logic*);
- oferece o conceito de modelo modelar, em que cada tipo de informação de contexto é representada por uma ontologia particular, para facilitar a reutilização e a extensibilidade;
- reutiliza conceitos de diversas ontologias consensuais e normalizadas da *Semantic Web*;
- permite a inferência de novos factos a partir de informação contextual prévia devido à sua semântica ontológica;
- utiliza normas da *Semantic Web* para representar os aspectos estruturais, semânticos e exibições lógicas de informações de contexto, como a RDF e a OWL.

Tal como o SOUPA este modelo é baseado em dois conjuntos de ontologias (Figura 4.10):

- ontologias de núcleo – definem os conceitos gerais através das ontologias *Actor* (identificação), *Spatial* (espaço), *Spatial Event* (eventos espaciais), *Temporal Event* (eventos temporais), *Device* (dispositivos), *Time* (tempo), *Activity* (actividades);
- ontologias de apoio *Knowledge* (área de conhecimento), *Relationship* (relações sociais), *Role* (regras sociais), *Contact* (informação de contacto), *Document* (documentos físicos e virtuais), *Project* (projectos reais).

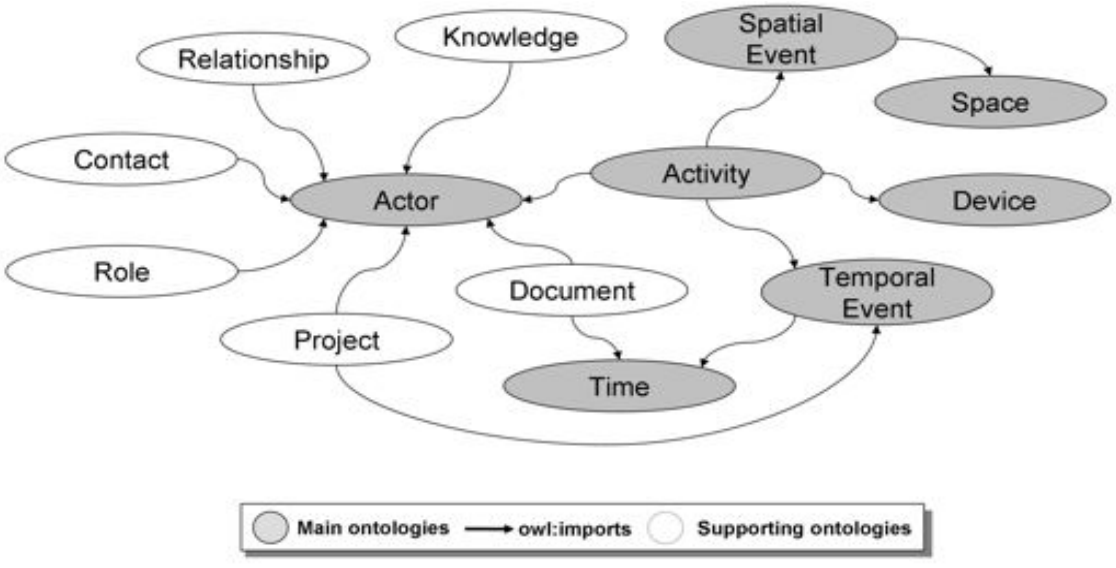


Figura 4.10 – Modelo SeCoM (Bulcão Neto and Pimentel, 2005).

As ontologias que pertencem ao núcleo são apresentadas em seguida e a sua descrição é retirada do trabalho (Sousa, Carrapatoso et al., 2009). Para obter informações mais detalhadas sobres estas ontologias, podem ser consultados os trabalhos (Bulcão Neto and Pimentel, 2005; Bulcão Neto and Pimentel, 2006):

- ontologia *Actor* – modela o perfil das entidades como pessoas, grupos e organizações que realizam acções num ambiente de computação ubíqua;
- ontologia *Time* – modela informação temporal em termos de instantes de tempo e intervalos de tempo, relações entre instantes de tempo e intervalos, relações entre intervalos de tempo e informação de calendário e relógio (duração de tempo, dia da semana, mês do ano, etc);
- ontologia *Temporal Event* – modela os eventos com extensões temporais como eventos instantâneos ou eventos que contenham intervalos de tempo. É uma extensão da ontologia *Time*. Por outras palavras, está habilitada a representar metodologias temporais entre eventos. Esta ontologia também representa informação sobre eventos temporais periódicos (ex. frequência de um evento);
- ontologia *Space* – descreve espaços virtuais ou reais interiores (ex. Sala) e exteriores (ex. Rua) e relações entre eles (ex. `spatiallyContains`, `isSpatiallyConnectedTo`), relações entre lugares, coordenadas geográficas (ex. latitude) e direcções (ex. norte) e regiões administrativas (ex. cidades);
- ontologia *Spatial Event* – modela os eventos com extensões espaciais chamados eventos espaciais, que são subclasses da classe `SpatialThing` definida na ontologia *Space*. Os eventos espaciais podem ser representados por duas classes disjuntas: eventos físicos, que são aqueles que ocorrem numa localização física (ex. entrada numa sala de reuniões), e eventos virtuais, que incluem aqueles que ocorrem em localizações virtuais (ex. entrada numa sala de *chat*). Em geral, ambos os eventos espaciais físicos e virtuais herdam todas as propriedades, relações e axiomas *PhysicalLocation* e *VirtualLocation* das classes da ontologia *Space*;
- ontologia *Device*: descreve dispositivos em termos das suas plataformas de hardware e software, relações entre componentes e aspectos de computação móvel necessários para sistemas sensíveis ao contexto. Em geral, modela informação sobre características de hardware e software como: capacidade da bateria,

componentes sem fios, resolução, sistemas operativos, navegadores *Web*, entre outros;

- ontologia *Actividade* – descreve as actividades como conjuntos de eventos espaço-temporais incluindo os actores ou dispositivos envolvidos neles. Deste modo, esta ontologia importa directamente as ontologias *Actor*, *Spatial Event*, *Temporal Event* e *Device* como é ilustrado na Figura 4.10. Sendo modeladas como eventos espaço-temporais, as actividades reutilizam os mesmos atributos e relações dos eventos espaciais e temporais. Por outras palavras, é possível inter-relacionar actividades em termos de relações descritivas e espaciais com a sua localização física/virtual (ex. "a reunião decorre na sala Da Vinci"), ou em termos de relações temporais como o instante de início ou um intervalo de tempo (ex. "a reunião começa às 9:30 e dura 1 hora"). Além disso, é também possível modelar actividades em dois tipos disjuntos: improviso e agendado. São utilizados para representar actividades que ocorrem de uma dada maneira (ex. "tomar café"), enquanto a outra representa normalmente uma actividade agendada em termos de tempo e espaço (ex. "reunião às 9:30 na sala Da Vinci"). O Código 4.2 descreve um enxerto de código RDF de uma conferência com o tema "Computer Science 2010" representada como uma actividade agendada na sala Da Vinci localizada no "Piso II" e que faz parte do IPB – Instituto Politécnico de Bragança. A conferência "Computer Science 2010" começa às 10:00 da manhã em 7 de Dezembro de 2010 e tem duas horas de duração. Os participantes desta actividade são designados através da propriedade `actvy:hasParticipant`. O prefixo `actvy:` é utilizado para representar o *namespace* XML para a ontologia Actividade. Em termos de raciocínio temporal e espacial, o motor de inferência pode inferir que a conferência continua a decorrer às 11:00 no Piso II.

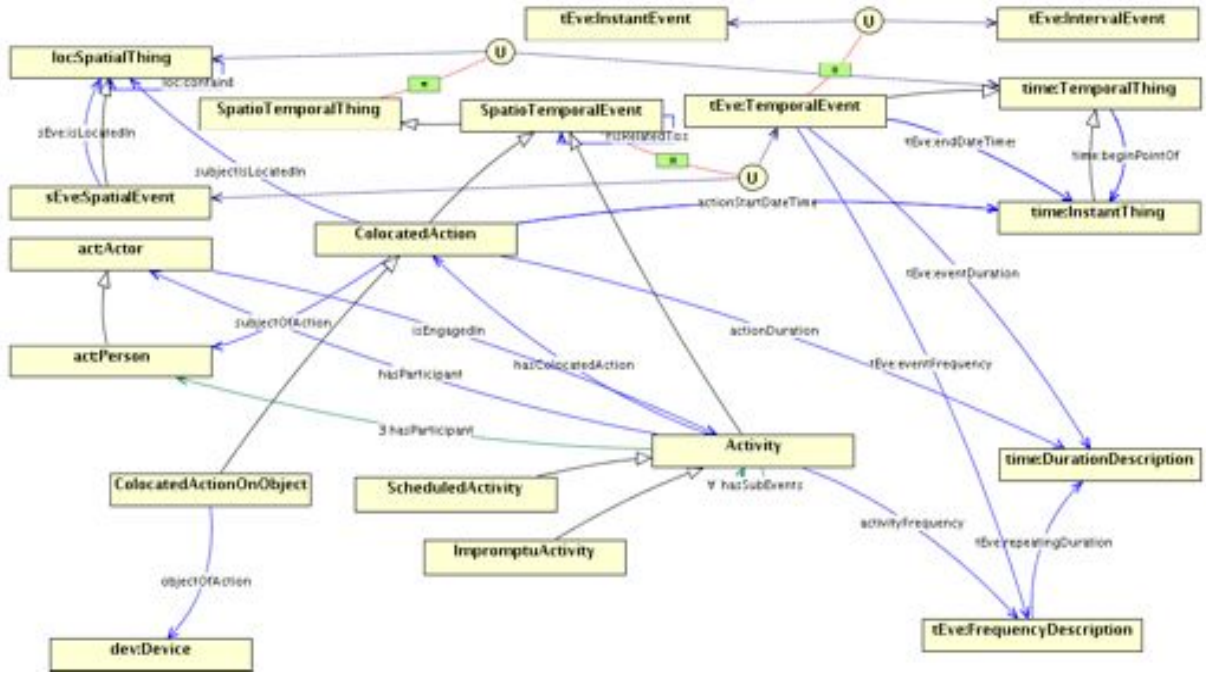


Figura 4.11 – Ontologia *Activity* (Bulcão Neto and Pimentel, 2005).

```
1: <actvy:CSConference rdf:ID="cmeeting19">
2:  <rdf:type rdf:resource= "&actvy;#ScheduledActivity"/>
3:  <actvy:hasName> Computer Science 2010 </actvy:hasName>
4:  <actvy:hasSummary>
5:    1st International Workshop on Software Engineering for Context
6:    Aware Systems and Applications
7:  </actvy:hasSummary>
8:  <actvy:hasParticipant rdf:resource="#person19"/>
9:  <sEve:isLocatedIn rdf:resource="#room82"/>
10: <time:beginPointOf rdf:resource="#bpo67"/>
11: <time:intervalDuratioDescriptionDataType rdf:datatype="&xsd;#duration">
12:    PT2H
13: </time:intervalDuratioDescriptionDataType>
14: </actvy:CSConference>
15: <act:Person rdf:ID="person19">
16:   <act:hasName>Ana Correia</act:hasName>
17: </act:Person>
18: <spc:DaVinciRoom rdf:ID="room82">
19:   <rdf:type rdf:resource="&spc;#Room"/>
20:   <spc:placeName>Da Vinci</spc:placeName>
21:   <spc:isSpatiallyPartOf rdf:resource="#floor4"/>
22: </spc:DaVinciRoom>
23: <spc:ConferenceFloor rdf:ID="floor2">
24:   <rdf:type rdf:resource="&spc;#Floor"/>
25:   <spc:placeName> Piso II </spc:placeName>
26:   <spc:isSpatiallyPartOf df:resource="#IPB"/>
27: </spc:ConferenceFloor>
28: <time:InstantThing rdf:ID="bpo67">
29:   <time:instantCalendarClockDataType rdf:datatype="&xsd;#dateTime">
30:    2010-12-07T10:00
31:   </time:instantCalendarClockDataType>
32: </time:InstantThing>
33:</actvy:CSConference>
```

Código 4.2 – Enxerto de código RDF a descrever uma actividade.

Em seguida são apresentadas as ontologias de apoio (Bulcão Neto, 2006):

- ontologia *Contact* – descreve informações de contacto de um actor (ex. morada de casa/trabalho, email). Esta ontologia está relacionada com a ontologia *Actor* através da propriedade `hasContactInformation` que liga a classe *Actor* à classe `ContactInformation`;
- ontologia *Role* – representa o papel social de um actor. (ex. aluno, empregado) Relacionam-se as classes *Actor* e *Role* através da propriedade `hasSocialRole`;
- ontologia *Relationship* – esta ontologia modela os relacionamentos sociais entre os pessoas (ex. é amigo de, é empregado de). Explora as propriedade simétricas (*SymmetricProperty*) e herança proporcionadas pela OWL. Por exemplo se "João é empregado de Pedro" então "Pedro é o empregador de João" então "Pedro e João têm um tipo qualquer de Relação";
- ontologia *Knowledge* – representa informações sobre o conhecimento que uma pessoa tem num determinado assunto ou tema de conhecimento (ex. "...tem experiência...", "...tem interesse...");
- ontologia *Document* – descreve informações de documentos físicos ou electrónicos elaborados por um actor (ex. documento de papel, autor, título);
- ontologia *Project* – representa projectos e relações entre estes e os actores. Relaciona classes das ontologias *Project*, *Actor* e *TemporalEvent* (ex. um projecto pode ter um título, membros e um prazo temporal).

4.6 Resumo

Este capítulo apresentou diversas tecnologias que podem ser usadas em arquitecturas SOA baseadas em *Web Services* e *Web Services Semânticos*. Iniciou-se o capítulo com a descrição da arquitectura que suporta os *Web Services* como base para posteriormente se compreender melhor as diversas propostas existentes na área dos *Web Services Semânticos* e quais as contribuições que estes trazem no âmbito dos sistemas distribuídos.

Observou-se que apesar dos *Web Services* possuírem normas maduras, sendo bem aceites quer pela indústria quer como tecnologia predominante nas arquitecturas distribuídas, no entanto

estes possuem também sérias restrições semânticas. Isto é, o conjunto de normas que definem a sua arquitectura, como SOAP, WSDL, UDDI, especificam apenas a funcionalidade a nível sintáctico, provocando limitações em funcionalidades como a localização e composição automática com base nas capacidades do serviço, recuperação de falhas em tempo de execução, entre outras.

Pela mão da DARPA, foi iniciado o programa DARPA DAML com o objectivo de fornecer a base para a próxima evolução da *Web*, a *Semantic Web* (Hendler and McGuinness, 2000). Deste e de outros programas semelhantes surgiram várias propostas como o objectivo de adicionar uma camada semântica aos *Web Services*, por forma a descrever as suas propriedades, capacidades e permitindo suportar um conjunto de novas funcionalidades como a descoberta automática por agentes de software e a invocação automática, entre outras. Para conhecer melhor esta proposta é apresentada neste capítulo uma revisão bibliográfica sobre essas *frameworks* e as linguagens mais populares que suportam os *Web Services Semânticos*. A partir desta revisão verificou-se que existem diversas propostas para a descrição semântica de *Web Services* que variam desde soluções complexas e bastante detalhadas, com o objectivo de explorar o máximo das características oferecidas pelos vários tipos de linguagens semânticas, a outras mais simples que podem ser vistas apenas como extensões semânticas de normas já existentes.

Finalmente, foi estudado o modelo contextual semântico SeCoM, que descreve a semântica da informação contextual através de um conjunto de módulos que representam domínios particulares, por forma a auxiliar o desenvolvimento de aplicações sensíveis ao contexto.

Capítulo 5

UM MODELO PARA O SUPORTE DA COMPOSIÇÃO CONTEXTUAL DE SERVIÇOS

Este capítulo apresenta um modelo que suporta a composição dinâmica de serviços para dispositivos móveis, utilizando informações contextuais e preferências dos utilizadores. Inicialmente são expostas as motivações para a adopção deste modelo e as directrizes que ele deverá cumprir. Posteriormente é apresentada a sua arquitectura genérica e são detalhados cada um dos seus componentes. O capítulo encerra com a discussão de alguns cenários de aplicação deste modelo.

5.1 Introdução

Com o avanço tecnológico dos dispositivos e redes móveis tem-se vindo a verificar o aparecimento de novas arquitecturas de serviços e utilitários de software cada vez mais evoluídos, que por sua vez impulsionam o uso destes dispositivos móveis como ferramentas de apoio na vida diária dos utilizadores. O actual mercado das comunicações móveis tem dado grande importância ao desenvolvimento de software para dispositivos móveis, como é o caso de aplicações *Web,* e como consequência incrementando a relevância dos serviços nesta área. Actualmente, os utilizadores podem aceder a serviços em qualquer momento e das mais diferentes formas, através de uma grande variedade de dispositivos móveis. Os serviços móveis representam hoje em dia um factor estratégico no mercado das comunicações móveis e como tal tem recebido bastante atenção das equipas de investigação. Recentemente tem-se verificado que este mercado tem iniciado a utilização de informações sobre o ambiente em que utilizador se encontra, de modo a poder-lhe fornecer serviços com melhores

funcionalidades e mais fáceis de utilizar. Esta classe de serviços. designados *context-aware*, tem como objectivos enriquecer os conteúdos, facilitar a interacção humana com os dispositivos e fornecer serviços mais personalizados ao utilizador. Acreditamos que nos próximos anos os serviços sensíveis ao contexto terão definitivamente um papel relevante na área dos serviços móveis e de todas as entidades relacionadas.

Na perspectiva de desenvolvimento na computação orientada aos serviços, os programadores utilizam os serviços como componentes fundamentais no processo de desenvolvimento de aplicações, sendo os *Web Services* um exemplo deste metodologia computacional. De uma forma geral, os *Web Services* eram usados apenas em aplicações transaccionais de negócios (Fonseca, 2005), no entanto recentemente grandes organizações na área da *Web* (*Google*, *Microsoft*, *Amazon*, *Yahoo*, *Facebook*, etc.) têm desenvolvido *frameworks* próprias, para um maior número de programadores e para um maior número de plataformas, maioritariamente na forma de *Web Services*. Esta disseminação dos *Web Services*, permite aos programadores e utilizadores expandir o seu uso para novos fins, como por exemplo, a composição de serviços para dispositivos móveis. Embora a composição de serviços seja uma área que tenha recebido grande atenção, isso aconteceu fundamentalmente na área de integração de aplicações empresarias ou *bussiness-to-bussiness*, o mesmo não se verificou em relação à composição de serviços para utilizadores finais. Um tipo de composição que permita aos utilizadores sem conhecimentos tecnológicos avançados construir um serviço e que possa ser feita em diversos tipos de dispositivos, entre quais os dispositivos móveis. Esta forma de composição permite ao utilizador através de uma combinação de serviços já existentes criar um novo serviço que vá de encontro às suas exigências.

Embora as tecnologias associadas à composição de serviços tradicionais na área do *bussiness-to-bussiness* (ex. WSDL, WSBPEL) estejam já bem estabelecidas e amplamente adoptadas, estas normas apresentam algumas limitações, como por exemplo:

- a WSDL apenas especifica o serviço a nível sintáctico, não descrevendo nenhuma funcionalidade semântica, podendo causar problemas na composição. Por exemplo, com a descrição WSDL pode acontecer que dois serviços possam ser combinados porque apresentam entras/saídas sintacticamente compatíveis, mas semanticamente serem completamente diferentes;
- a linguagem de composição de serviços WSBPEL não consegue expressar propriedades avançadas, como heranças e relacionamentos entre *Web Services*,

pré-condições necessárias para a utilização de um serviço e efeitos que esse serviço poderá produzir no seu meio com a sua execução. Como consequência, todo o processo de composição é mais limitado, não permite a composição automática de serviços, já que a composição terá que ser feita manualmente e à priori, através da selecção de serviços e construção do *workflow*;

- com a WSBPEL o processo de composição é criado em modo *off-line*, sendo o seu resultado estático e com carências de suporte de re-composição durante o processo de execução. A alteração de um serviço na composição pode resultar na re-configuração de todo o processo de composição (Hoecke, Steurbaut et al., 2009).

Estas normas não permitem acrescentar informação semântica essencial sobre o significado da funcionalidade dos serviços. Esta informação, como referido na secção 4.3, adiciona novas características ao processo de composição, como a composição dinâmica, e torna ainda possível explorar informação contextual do utilizador e as suas preferências.

Nas secções seguintes é apresentado o modelo que permite a composição dinâmica de serviços em dispositivos móveis sensível às informações contextuais e às preferências dos utilizadores. A composição dinâmica de serviços deverá permitir a criação de um processo correcto de combinação de serviços, a utilizadores sem conhecimentos técnicos sobre programação. Este processo será auxiliado através de informações externas, por forma a oferecer uma composição mais simples e eficiente.

5.2 Linhas orientadoras

Baseando-se em vários trabalhos analisados na secção 2.5, alguns dos quais apresentavam soluções para sistemas baseados em contexto e para a composição de serviços, como é o caso de (Dey, 2000), (Panagiotakis and Alonistioti, 2006), (Sheshagir, Sade et al., 2004), (Gu, Pung et al., 2005), (Chakraborty, Joshi et al., 2005) e (Chen, 2004), este modelo deverá seguir as seguintes directrizes:

1. permitir a composição e execução de novos serviços mais personalizados, utilizando todas as vantagens oferecidas pelo cenário contextual em que o utilizador se encontra;
2. a composição de um serviço deverá resultar num novo serviço *stand-alone*, de

modo que possa ser utilizado sozinho ou em novas composições, de forma a criar serviços evolutivos;

3. oferecer ao utilizador um conjunto de serviços sensíveis ao contexto, perfil e preferências do utilizador;
4. permitir o armazenamento de serviços para utilização futura e partilha com outros utilizadores;
5. fomentar a interoperabilidade e partilha da semântica contextual entre sistemas *context-aware*;
6. permitir a adaptação do conteúdo às características do terminal e, em funções mais avançadas às capacidades da rede.

As aplicações contextuais prometem fornecer ao seus consumidores serviços mais adaptados e personalizáveis. No entanto, o utilizador ficará sempre restrito às funcionalidades padrão que o serviço oferece e que por vezes não satisfazem os seus requisitos. Actualmente é já possível encontrar aplicações que fazem uso de informação contextual, como, por exemplo, um serviço que mostra todos os amigos na vizinhança do utilizador. Mas o utilizador poderá querer um serviço mais personalizado, que por exemplo permita saber quais dos seus amigos estão a estudar matemática, mostrar onde é que eles se encontram e até iniciar uma conversação com um deles. Este serviço mais avançado, pode ser conseguido através da junção dos serviços de pesquisa dos seus amigos por actividade com o serviço de localização e o serviço de mensagens instantâneas. Com a composição de serviços é possível oferecer um nível superior de flexibilidade e personalização de serviços aos utilizadores.

A oferta dos serviços em determinado momento pode beneficiar claramente de várias informações externas, mostrando assim os mais adequados ao utilizador. Considera-se que fazem parte dessas informações o perfil do utilizador, as suas preferências e o contexto em que este se encontra num determinado instante. Estas informações, recolhidas geralmente sem acção deliberada do utilizador, serão usadas para alterar estados, funcionamento ou forma de interacção do sistema com o utilizador. Desta forma, este modelo deve incluir um mecanismo de seja capaz de explorar e gerir a informação contextual de modo a oferecer uma melhor experiência ao utilizador na composição de serviços e na consulta de informações.

Para que seja possível explorar todo o potencial do contexto é necessário que os dados contextuais sejam capturados. Esses dados são recolhidos por entidades designadas fontes de

contexto, que podem ser sensores físicos, serviços ou outro tipo de aplicações. Estas informações devem ser obtidas de uma forma o mais transparente possível ao utilizador e podem variar em formato, unidades, frequência, volume de dados ou tempo de resposta. Assim, poderá ainda acontecer que essas fontes de contexto não forneçam os dados da forma mais conveniente, sendo necessário um serviço de adaptação que forneça uma camada de abstracção dessas várias fontes, fornecendo uma visão comum delas.

Tal como o contexto, as preferências do utilizador são informações importantes que indicam os desejos do utilizador. O perfil de preferência do utilizador pode expressar as características desejadas do modo como um serviço é fornecido a esse utilizador. Estas preferências podem ser mais genéricas, sendo aplicáveis a todos os serviços a que o utilizador recorre, ou mais específicas, sendo apenas aplicáveis a um serviço, contendo neste caso características particulares desse serviço. São exemplo de preferências do utilizador: língua preferida, características de visualização (fontes, tipo de média), requisitos de qualidade de serviço, segurança, privacidade, preço, etc. Além das preferências referentes aos serviços, estas também podem expressar os desejos de parâmetros do dispositivo terminal do utilizador, como, por exemplo, o contraste, a luminosidade do visor ou o volume do som.

O armazenamento de informações de contexto é um requisito das arquitecturas *context-aware* para disponibilizar em qualquer momento um histórico de informação contextual. Pode-se dizer que no espaço temporal existem dois modos de utilizar a informação de contexto: a sua utilização em tempo real, quando esta é capturada, por exemplo um serviço que mostra num mapa a localização instantânea do utilizador; ou a utilização como histórico de informações que pode fornecer novos meios de explorar o contexto (ex. esse mesmo serviço de localização pode armazenar o histórico das coordenadas, para o utilizador partilhar os sítios que visitou na sua viagem de férias ou até prever uma futura localização do utilizador). Um histórico de contexto pode ser utilizado para determinar tendências ou prever valores futuros. Também quando se utilizam informações de perfil e preferências, deve ser possível o seu armazenamento de forma persistente, para que seja possível a sua recuperação em caso de falha do sistema que provoque a perda dos dados em memória. Por estas razões um modelo que seja sensível ao contexto deve suportar o seu armazenamento.

De modo a ser possível usar serviços em diferentes tipos de dispositivos e redes, estes devem ser adaptáveis ao dispositivo em que são apresentados e à rede que suporta o seu acesso. Esta característica tem várias vantagens: permite responder à cada vez maior variedade de

dispositivos existentes no mercado (*desktops*, *laptops*, *netbooks*, *smartphones*, etc), permite maximizar a sua utilidade para o utilizador, porque pode ser usado em qualquer dispositivo e para o fornecedor de serviço pois permite minimizar o seu custo desenvolvimento. Assim, deverá existir um serviço de adaptação de serviços à interface do utilizador que, utilizando informações de perfil, preferências e contexto, possa adaptar a entrega dos conteúdos dos serviços à interface do utilizador e em casos mais avançados às características da rede.

A partilha de contexto entre aplicações sensíveis ao contexto é um aspecto importante, mas para isso é necessário que elas tenham dele a mesma interpretação semântica.

Pelos motivos apresentados nos últimos parágrafos, um sistema genérico que permita a composição de serviços baseada em contexto deverá oferecer as seguintes funcionalidades:

1. composição, descoberta, gestão e partilha de novos serviços;
2. sensibilidade ao contexto, às preferências e ao perfil;
3. captação de dados contextuais;
4. utilização de um modelo contextual semântico baseado em normas;
5. adaptação de conteúdos ao terminal e à rede.

5.3 Arquitectura do sistema

Para suportar a composição dinâmica e contextual de serviços e para fornecer informação contextual aos utilizadores, é proposta uma arquitectura orientada aos serviços (SOA) baseada em ontologias, designada *iCas*. Esta arquitectura é composta por cinco componentes principais que exploram o potencial do contexto, como mostra a Figura 5.1:

- *motor de composição* – responsável por implementar a funcionalidade nº 1 e atender às directrizes nº 1 e nº 2;
- *motor de contexto* – é o componente que fornece a funcionalidade nº 2 e atende à directriz nº 3;
- *motor de captação de dados contextuais* – implementa a funcionalidade nº 3 e exerce funções de suporte para atender à maioria das directrizes;
- *modelo semântico contextual* – permite suportar a funcionalidade nº 4 e atende à

directriz nº 5;

- *repositório de informações e serviços* – suporta parte da funcionalidade nº 1 e atende à directriz nº 4;
- *motor de adaptação* – fornece a funcionalidade nº 5 e atende à directriz nº 6.

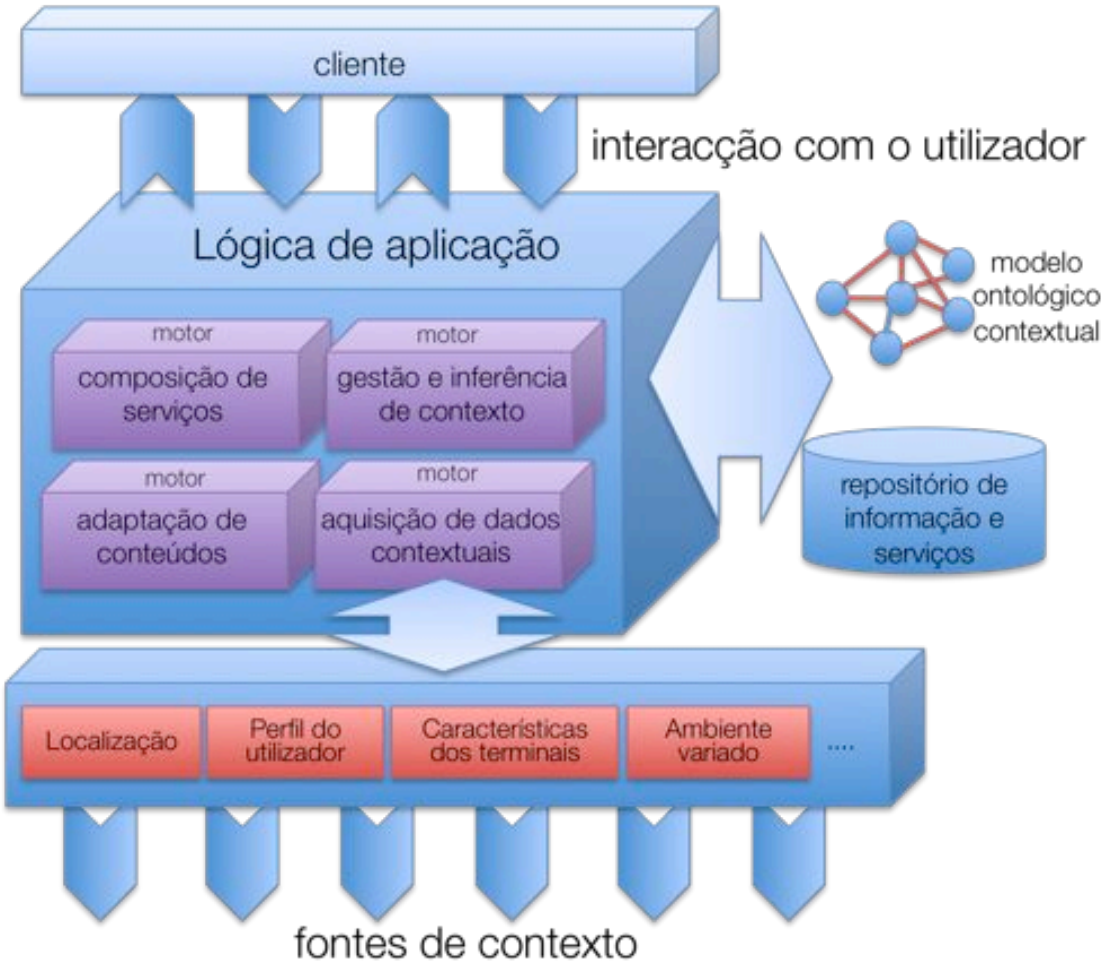


Figura 5.1 – Arquitectura genérica para o modelo proposto.

A arquitectura apresentada possui as seguintes características:

- aberta – é baseada em normas e recomendações abertas e largamente suportadas, grande parte delas propostas pelo W3C e pelo IETF;
- híbrida – baseia-se no modelo cliente/servidor centralizado na interacção com o terminal do utilizador, no entanto tem também propriedades de sistemas distribuídos, já que as funcionalidade dos serviços podem estar dispersas por diversos computadores;
- interoperabilidade – permite a comunicação entre o cliente e o servidor,

independentemente do sistema em que são executados ou das tecnologias de implementação de cada um deles;

- expressividade e formalismo – suporta um alto grau de expressividade e formalismo para representar conceitos e relações num cenário *context-aware*;
- raciocínio – permite a inferência de novos factos, a partir de informação contextual já existente;
- domínio contextual modular – é baseada num modelo de contexto independente de domínios e modular, permitindo adicionar novos domínios contextuais;
- partilha e reutilização – a descrição de contexto através de domínios ontológicos baseados em normas da *Semantic Web* facilita a troca, reutilização e partilha de informação com outros sistemas *context-aware*;

Dos vários modelos contextuais semânticos apresentados na secção 2.1.4 (SOCAM, SOUPA e SeCoM) escolheu-se como modelo contextual semântico para a arquitectura *iCas*, o modelo SeCoM apresentado na secção 4.5. O motivo desta escolha deveu-se principalmente aos seguintes motivos: este modelo oferece um alto grau de expressividade e formalismo para representar conceitos (Sousa, Carrapatoso et al., 2009) e relações num cenário sensível ao contexto; permite o raciocínio sobre o contexto; é baseado em normas da Semantic Web; dos três modelos estudados, este era aquele cuja expressividade lógica constitui, um factor importante para que se conheça a complexidade computacional dos processo de inferência ao utilizar as várias ontologias do modelo.

5.4 Casos de utilização

Na secção anterior foram identificadas as directrizes que devem ser seguidas por um modelo que suporte composição de serviços baseada em contexto e foram especificadas as funcionalidades que este modelo deverá suportar. Com base nessas funcionalidades são apresentados, na Figura 5.2, os casos de utilização que qualquer aplicação que suporte este modelo deverá implementar.

Nestes casos de utilização foram identificadas três entidades:

- utilizador – irá utilizar o sistema para usar serviços e receber informação

contextual;

- administrador – pessoa responsável pela gestão do sistema proposto; tem a responsabilidade de gerir os serviços, utilizadores e informação contextual;
- sistema – ambiente que disponibiliza serviços aos utilizadores e ao administrador.

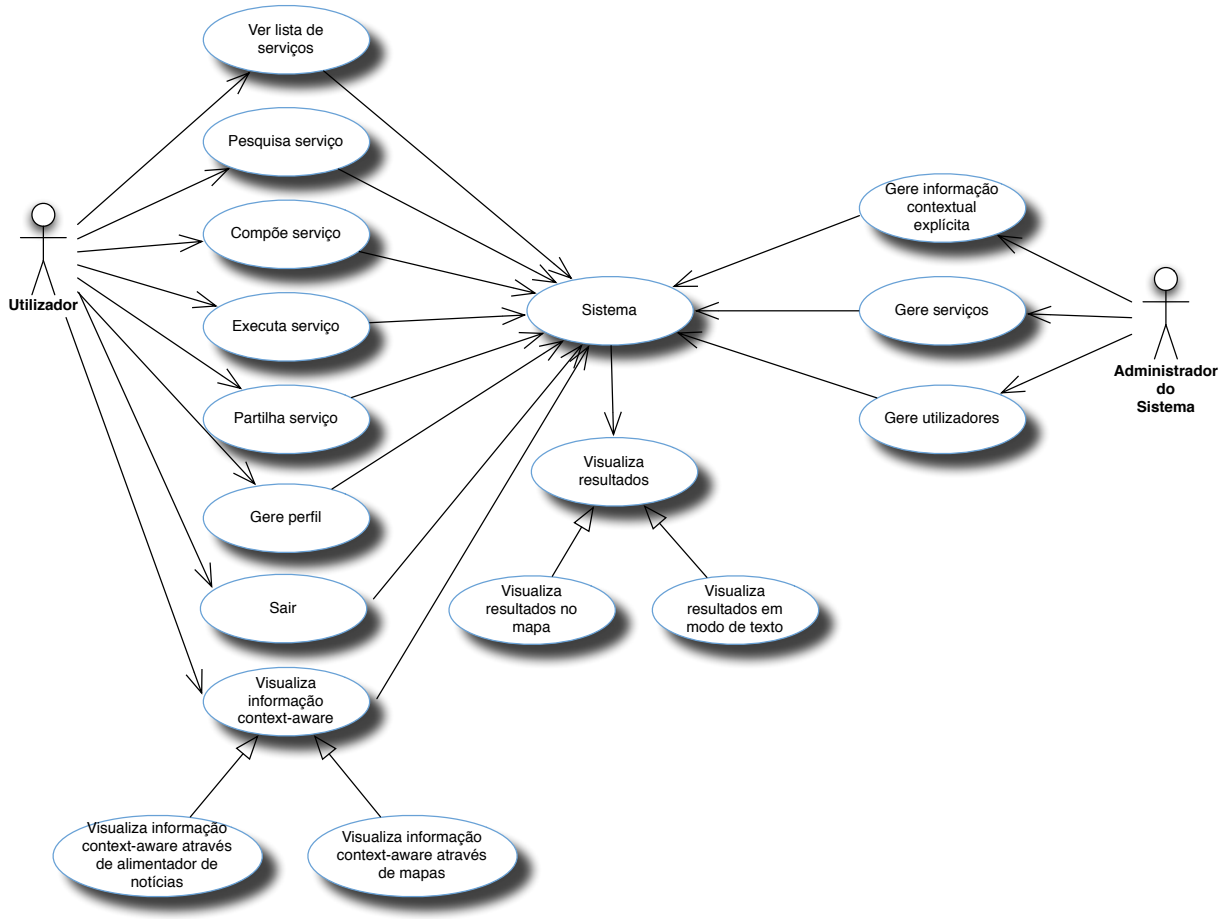


Figura 5.2 – Casos de utilização de aplicação do modelo proposto.

Os casos de utilização estão divididos em dois grupos: os que estão disponíveis aos utilizadores e aqueles a que apenas o administrador do sistema tem acesso. Segundo a divisão referida, identificam-se os seguintes casos de utilização disponíveis aos utilizadores comuns:

- ver lista de serviços – permite ao utilizador ver uma lista de serviços disponíveis para utilização ou criação de um novo serviço;
- pesquisa de serviços – permite ao utilizador procurar serviços por nomes e propriedades;
- composição de serviços – caso o utilizador não encontre nenhum serviço que vá de encontro aos seus requisitos, poderá fazer a composição de um novo serviço com

propriedades mais avançadas, através de serviços já existentes, quer eles sejam atómicos ou compostos;

- execução de serviço – permite a execução de um serviço já existente ou composto e a visualização dos resultados; o utilizador poderá ter que fornecer informações necessárias para a execução do serviço;
- partilha de serviço – o utilizador poderá partilhar qualquer serviço seu com outros utilizadores, publicando esse serviço num registo comum a todos eles;
- gestão de perfil – permite ao utilizador alterar o seu perfil pessoal e outras informações pessoais como as preferências;
- visualização de informação *context-aware* – possibilita ao utilizador receber de forma automática informações contextuais no seu dispositivo (ex. informações sobre eventos, redes sociais, notícias);
- sair – permite ao utilizador sair da aplicação, dando instruções ao sistema para que a sua sessão ser encerrada.

No segundo grupo, identificam-se os seguintes casos de utilização do administrador:

- gestão de informação contextual explícita – permite-lhe gerir qualquer informação contextual do sistema (ex. adicionar eventos, avisos, locais);
- gestão de serviços – o administrador poderá adicionar, remover ou alterar serviços púbicos ou particulares de qualquer utilizador;
- gestão utilizadores – permite adicionar ou remover utilizadores ao sistema. Ao adicionar um utilizador ao sistema é criado um registo desse utilizador que terá associado todas as informações sobre ele. Após um utilizador ser registado poderá ter acesso a todas as funções descritas anteriormente para a entidade utilizador.

5.5 Comportamento do modelo proposto

Na Figura 5.3 é apresentado o diagrama de estados na perspectiva do utilizador, mostrando as mudanças de estado do sistema motivadas por eventos.

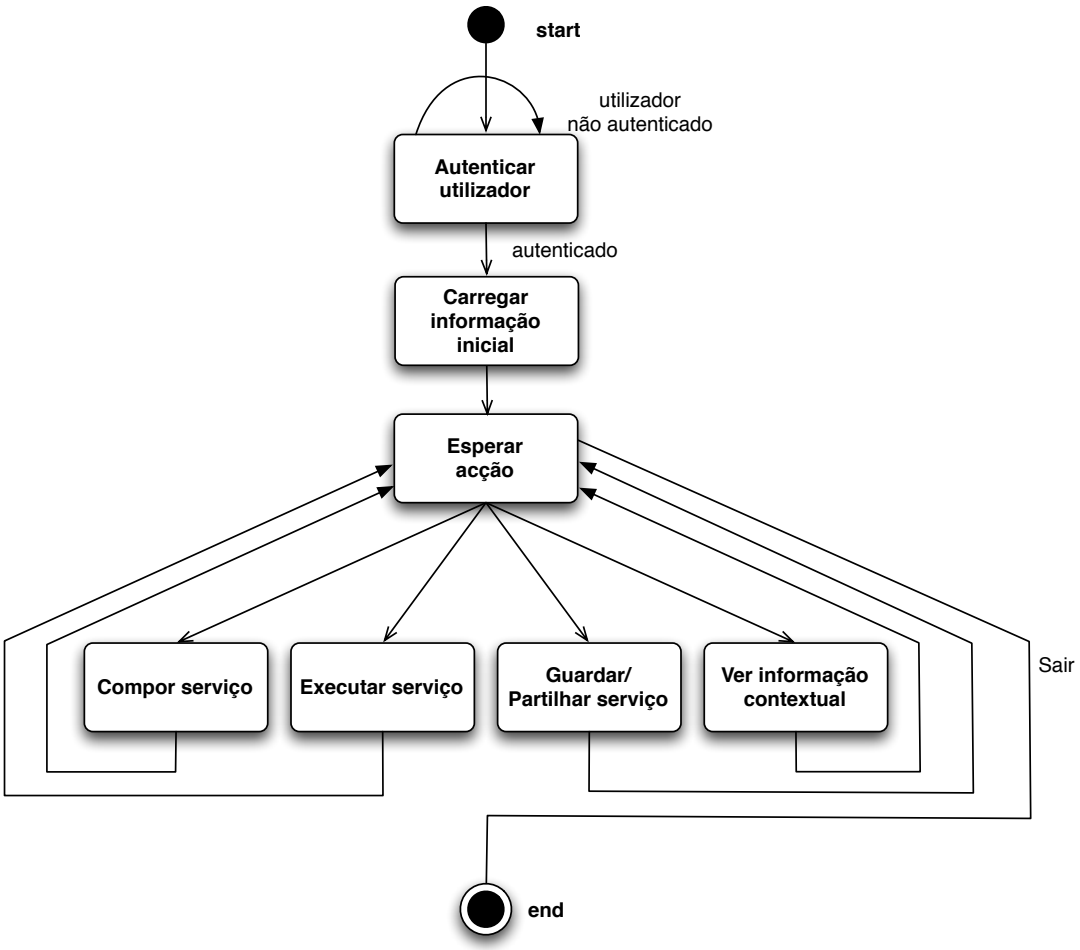


Figura 5.3 – Diagrama de estados do sistema na perspectiva do utilizador.

Inicialmente é verificado se o utilizador está ou não autorizado a usar as funcionalidades do sistema. Posteriormente são carregadas algumas das informações contextuais e do seu perfil e os serviços que terá ao seu dispor. Em seguida, o sistema fica a aguardar a escolha de alguma das funcionalidades disponíveis ao utilizador. Quando este seleccionar uma das funcionalidades o sistema irá executar o seu pedido e regressar ao estado de espera. Este ciclo irá decorrer enquanto não for escolhida a opção de abandonar o sistema.

O diagrama de estados na perspectiva do administrador é apresentado na Figura 5.4. Como ilustra o diagrama, ele só poderá ter acesso ao painel de administração após autenticação. Tal como acontece no diagrama anterior, também aqui o sistema fica a aguardar que o administrador seleccione uma das operações disponíveis. Após a realização dessa operação o sistema irá de novo para o estado de espera, até que seja escolhida a opção de abandonar o sistema.

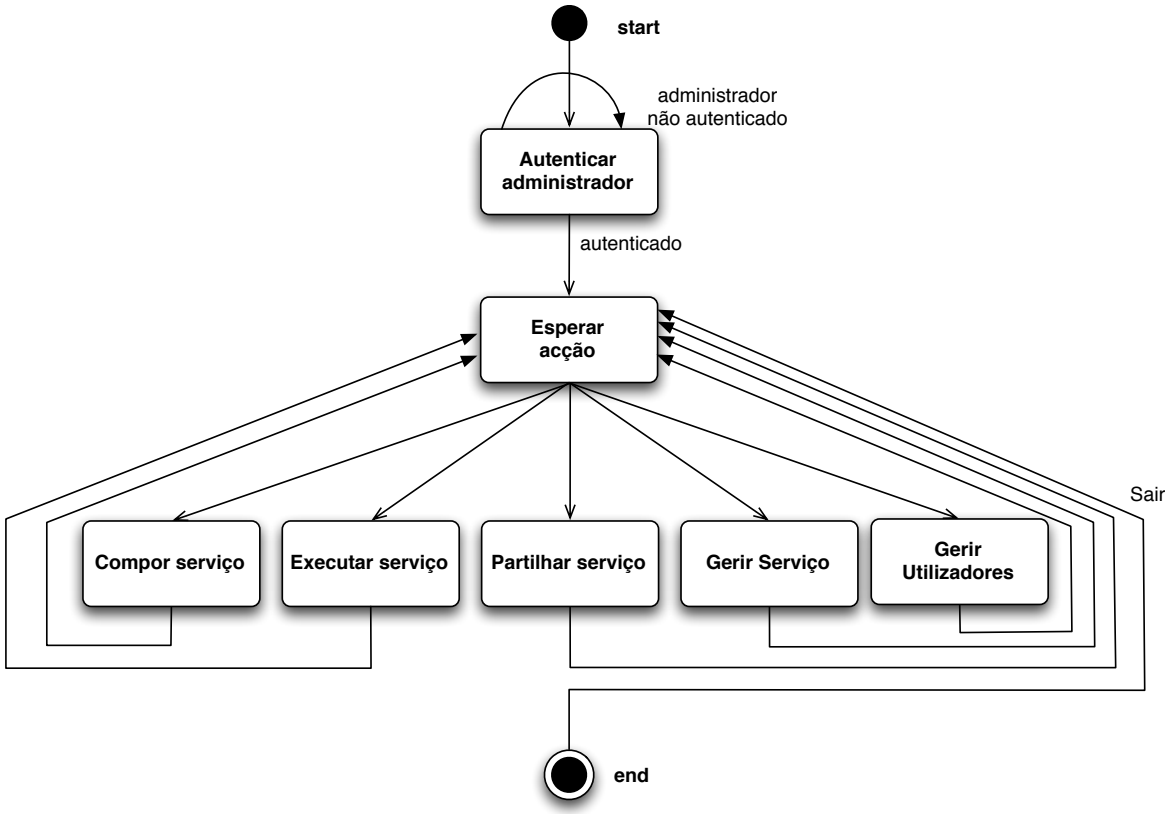


Figura 5.4 – Diagrama de estados do sistema na perspectiva do administrador.

Na Figura 5.5 é ilustrado o diagrama de actividades alusivo ao modelo proposto. As primeiras actividades executadas são a ligação à base de dados e a criação dos objectos necessários para suportar as diversas funcionalidades do modelo. Após a criação dos diversos motores, o sistema está pronto para receber os pedidos dos clientes.

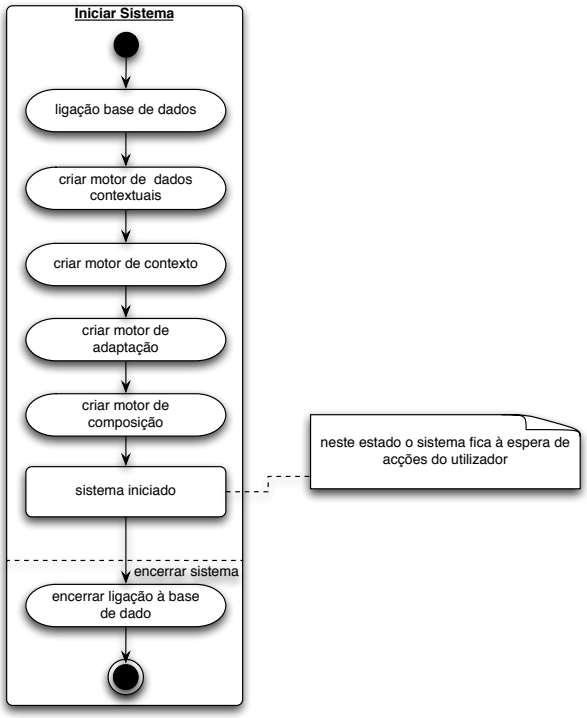


Figura 5.5 – Diagrama de actividades do sistema proposto.

Os diagramas anteriores ilustram o comportamento do modelo fornecendo uma visão geral dos principais estados e actividades do sistema. É possível concluir que o sistema é composto por diversos componentes que interagem entre si, com o objectivo de fornecer ao utilizador uma plataforma orientada aos serviços e sensível ao contexto.

5.6 Descrição dos componentes da arquitectura

Uma arquitectura especifica a estrutura geral, a organização e os aspectos funcionais de algo que se pretende criar. Na Figura 5.6 é ilustrado o diagrama de componentes da arquitectura que será proposta nesta secção e que será relevante para a fase de implementação do sistema.

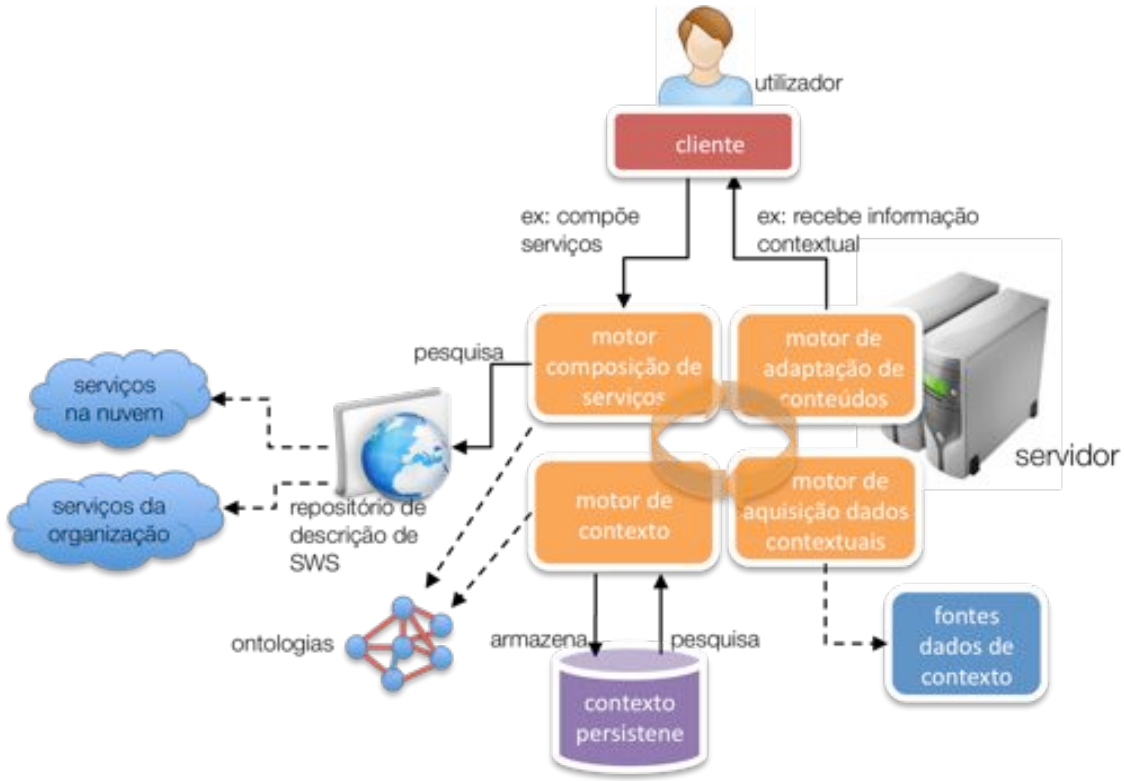


Figura 5.6 – Relacionamento geral entre os componentes da arquitectura.

A arquitectura apresentada na Figura 5.6 é baseada no modelo cliente/servidor, em que o utilizador através de um cliente com uma interface gráfica interage com o sistema que lhe fornece os serviços através de *Web Services*. Foi escolhida esta configuração para colocar do lado do servidor a carga computacional do motor de inferência utilizado na dedução de nova informação contextual e composição de serviços, suportando assim a composição *ad-hoc* de serviços em dispositivos móveis. Como é possível observar na Figura 5.6, o servidor da arquitectura apresenta quatro componentes principais: *motor de composição*, *motor de contexto*, *motor de aquisição dados contextuais e motor de adaptação de conteúdos*. Este motores actuam em conjunto com dois repositórios, um deles contém a descrição dos *Web Services Semânticos* e o outro o *contexto persistente,* e com as diversas *fontes de dados contextuais*, respectivamente.

O *motor de composição* lida com todas as funcionalidades referentes aos serviços. Suporta as funções de descoberta, selecção e composição de serviços e permite ainda a gestão de serviços como a sua inserção e eliminação num *repositório de serviços*. *O motor de contexto* faz a gestão do perfil e preferências do utilizador, manipula a semântica de informações contextuais explícitas e fornece a funcionalidade de inferência de novo conhecimento. O *motor de adaptação de conteúdos* é responsável por seleccionar os conteúdos a apresentar ao utilizador.

O motor de aquisição de dados contextuais é responsável por recolher dados de várias fontes contextuais e adaptá-los para serem utilizados pelo *motor de adaptação de conteúdos* e pelo *motor de contexto*.

Todas as operações relacionadas com os serviços são suportadas apenas através da utilização de *Web Services Semânticos*. Caso um serviço por natureza não seja semântico terá que ser transformado de modo a que seja descrito por uma interface semântica. Todos os serviços utilizados na arquitectura são registados num *repositório de serviços*, que contém as descrições das interfaces semânticas de cada um deles.

O suporte de armazenamento do contexto de forma permanente é garantido através da utilização de uma base de dados representada na Figura 5.6 por *contexto persistente*.

Através da utilização dos *Web Services Semânticos* e das ontologias que descrevem o modelo semântico contextual, é possível suportar as funcionalidades básicas da arquitectura, como o a composição dinâmica e contextual de serviços e a inferência sobre informações de contexto.

Os principais componentes da arquitectura e o seu funcionamento interno serão apresentados ao longo deste capítulo, nas próximas sub-secções.

5.6.1 Motor de composição

Quando um utilizador inicia a composição, pode já ter uma noção exacta das tarefas a executar recorrendo à composição ou pode simplesmente iniciar a composição escolhendo os serviços que lhe vão sendo sugeridos. Em ambas as situações a composição de serviços é um processo contínuo onde o utilizador pode adicionar ou remover novos serviços de uma forma dinâmica e interactiva, pretendendo esta arquitectura simplificar essa tarefa. Todas as funcionalidades relativas à composição e gestão de serviços são fornecidas pelo *motor de composição*, que dependendo da tarefa a executar irá comunicar com os outros componentes da arquitectura.

O *motor de composição* é composto por quatro componentes: *núcleo do motor de composição*, *gestor de serviços*, *execução de serviços* e *descoberta e selecção de serviços*. Na Figura 5.7 são ilustrados esses quatro componentes e as ligações a outros componentes do sistema com os quais interagem directamente.

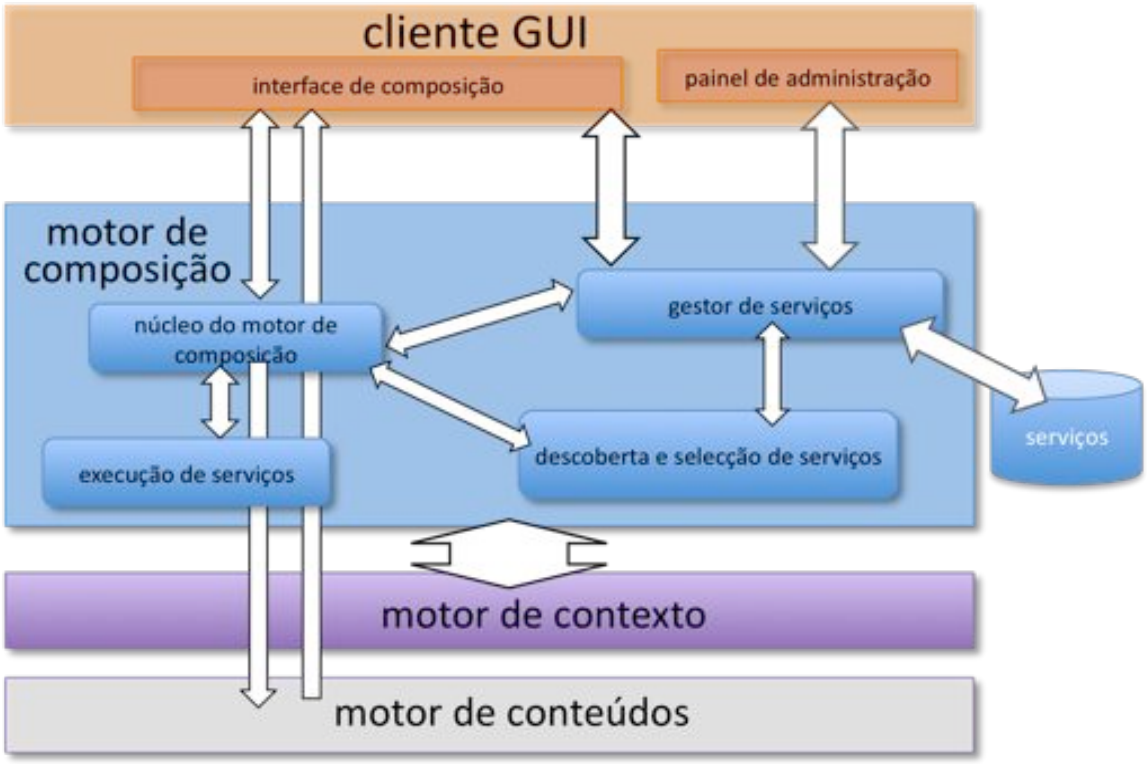


Figura 5.7 – Arquitectura do *motor de composição*.

O componente *núcleo do motor de composição* oferece a funcionalidade de composição de serviços ao cliente através de um *Web Service*. A composição de um novo serviço será conseguida através de uma visualização prévia dos serviços disponíveis e da posterior selecção de um serviço para fazer parte dessa composição. Este componente cria um *workflow* de invocação de serviços apresentando os serviços disponíveis em cada passo da composição. No final, esse *workflow* será reconhecido como um serviço próprio que poderá ser executado, armazenado ou usado noutra composição.

A apresentação de uma lista de serviços ao utilizador ocorrerá sempre que este seleccionar um novo serviço. Nessa lista irão constar apenas os serviços sintacticamente compatíveis com o serviço seleccionado e ordenados com base no contexto do utilizador. Essa lista é fornecida pelo componente *descoberta e selecção de serviços* e sofre duas operações: a primeira é uma filtragem sintáctica, pois não faz sentido apresentar serviços que não sejam compatíveis com o estado actual do fluxo de composição; a segunda é a ordenação dos serviços, de modo a que apareçam em primeiro lugar os "melhores" serviços com base na análise do contexto do utilizador. Podem existir várias formas de utilizar a informação contextual para lidar com os serviços: fazer apenas a selecção de serviços adequados ao contexto actual do utilizador, proceder à eliminação dos serviços não contextuais ou realizar uma ordenação dos serviços

atribuindo-lhes uma pontuação proporcional à sua adequação ao contexto. A solução escolhida foi a última, a de oferecer ao utilizador uma lista de serviços ordenados. Esta escolha permite ao utilizador aceder a serviços que poderiam ser considerados como dispensáveis no actual contexto em que o utilizador se encontra e como tal ficariam fora do seu alcance. Esta opção tem a desvantagem de em determinadas situações produzir uma lista de serviços demasiado extensa, com muitos serviços que não são do interesse do utilizador. O funcionamento desta solução será apresentado e descrito mais à frente. Os serviços disponíveis no repositório são carregados em memória e de cada vez que é gravado um serviço no repositório este também será adicionado à lista de serviços que se encontra em memória.

A execução de um serviço poderá ser realizada de duas formas: em qualquer momento durante a composição de um serviço, ou através da selecção de um serviço que se encontre no *repositório de serviços* e ordenando a sua execução. Quando um serviço é executado poderá ter de retornar diversos tipos de dados que serão entregues ao cliente através do componente *adaptador de conteúdos*. O componente de execução de serviço contém um motor capaz de invocar de igual forma processos atómicos ou processos compostos que contêm construtores de controlo. Quando este componente executa uma composição, ele segue um *workflow* para invocar cada serviço de forma individual e troca os dados entre eles de acordo como o fluxo construído pelo utilizador.

O gestor de serviços distingue os dois tipos de utilizadores: utilizador e administrador. Os casos de utilização que são comuns a estes dois tipos de utilizadores serão executados do mesmo modo, diferindo obviamente no alcance das permissões, pois o administrador terá permissões totais de gestão. Como o próprio nome indica, este componente permite gerir os serviços do sistema e inclui as seguintes funcionalidades: armazenamento, pesquisa, partilha e remoção de serviços. O armazenamento poderá ser feito por qualquer utilizador e permite guardar a descrição do serviço num repositório para utilização futura. Quando se fala em armazenamento de um serviço, convém realçar que apenas se trata de guardar a sua descrição ou interface, porque a sua funcionalidade continua a ser oferecida por um fornecedor de serviços interno ou externo à organização. A pesquisa de serviço permite procurar um serviço utilizando as informações que este contém na sua descrição. A partilha de um serviço consiste em armazenar esse serviço num repositório com permissões de partilha para todos os utilizadores do sistema, por forma a criar uma rede de serviços evolutivos. Esta partilha funcionará de modo semelhante ao registo UDDI, que guarda informações sobre os

descritores dos serviços e sobre os fornecedores de serviços. A remoção de serviços consiste em remover a descrição do serviço do *repositório de serviços*. O utilizador apenas poderá remover os seus serviços enquanto que o administrador poderá remover qualquer serviço.

A Figura 5.8 apresenta o diagrama de sequência do caso de utilização "composição de serviço" por parte de um utilizador.

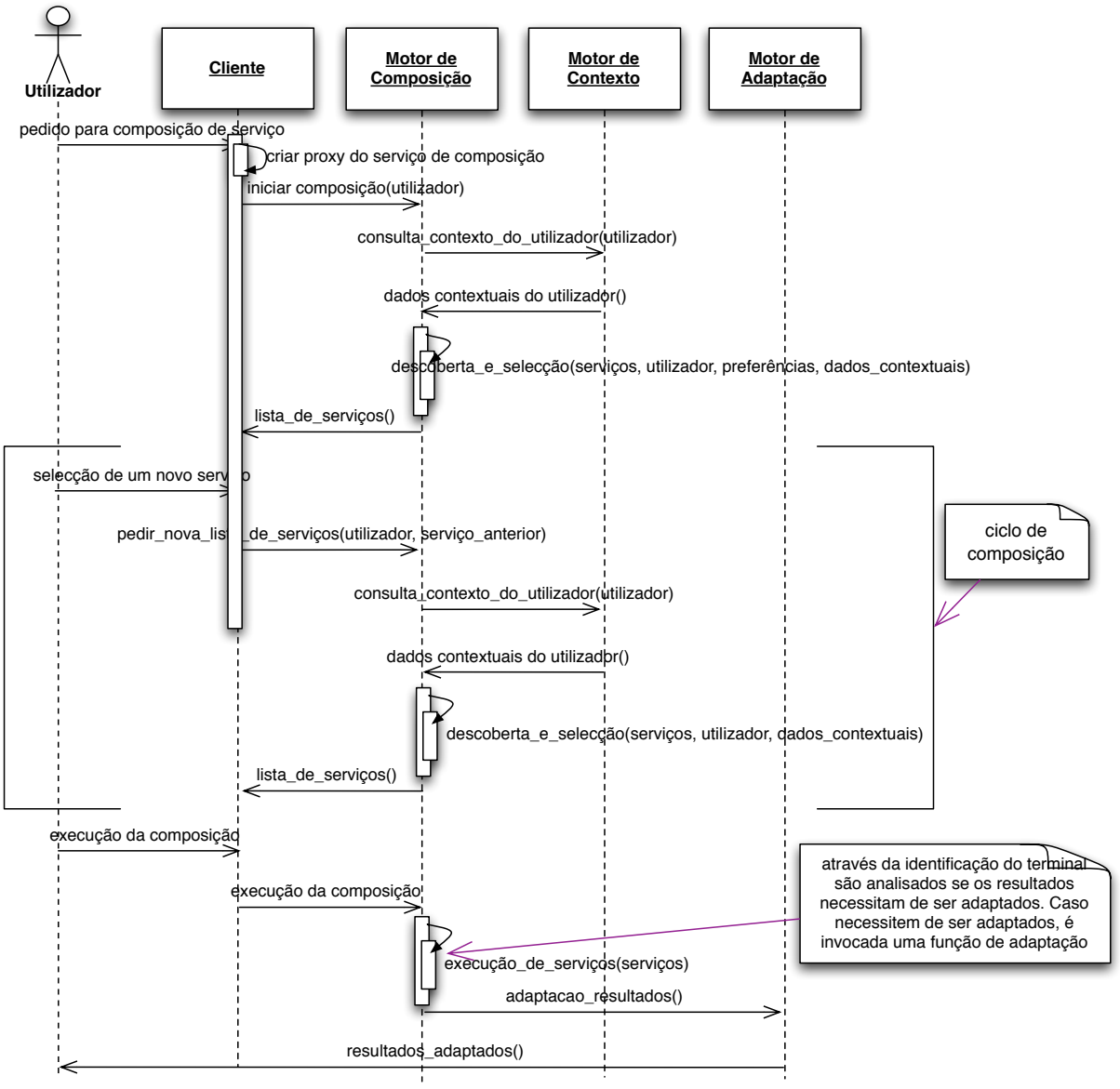


Figura 5.8 – Diagrama de sequência de uma composição de serviços.

O utilizador, através de uma interface cliente, inicia a execução seleccionando um serviço da lista que lhe é disponibilizada. Uma vez seleccionado o primeiro serviço para fazer parte da composição, o componente *descoberta e selecção de serviços* procura serviços usando os dados coleccionados pelo *motor de contexto* e retorna novas possibilidades, baseadas no actual contexto e nas preferências do utilizador, segundo o mecanismo que é ilustrado na

Figura 5.9. A descoberta e a selecção apenas são possíveis devido à descrição de serviços utilizando ontologias, que permitem criar relações com outras ontologias que podem descrever detalhes sobre o tipo de serviço e as suas características. A descoberta é executada utilizando a descrição do perfil do serviço, que caracteriza o que os serviços podem fazer e especifica o tipo de entradas/saídas, pré-condições e efeitos. Considerando-se que a composição será criada na forma de pilha, isto é, o primeiro serviço seleccionado será aquele que irá definir a saída do serviço composto, é feita uma selecção dos serviços cuja a saída seja compatível com as entradas do serviço anteriormente seleccionado. Por fim, os serviços são ordenados utilizando os parâmetros de avaliação definidos no perfil de serviços e uma política de avaliação de serviços particular do utilizador.

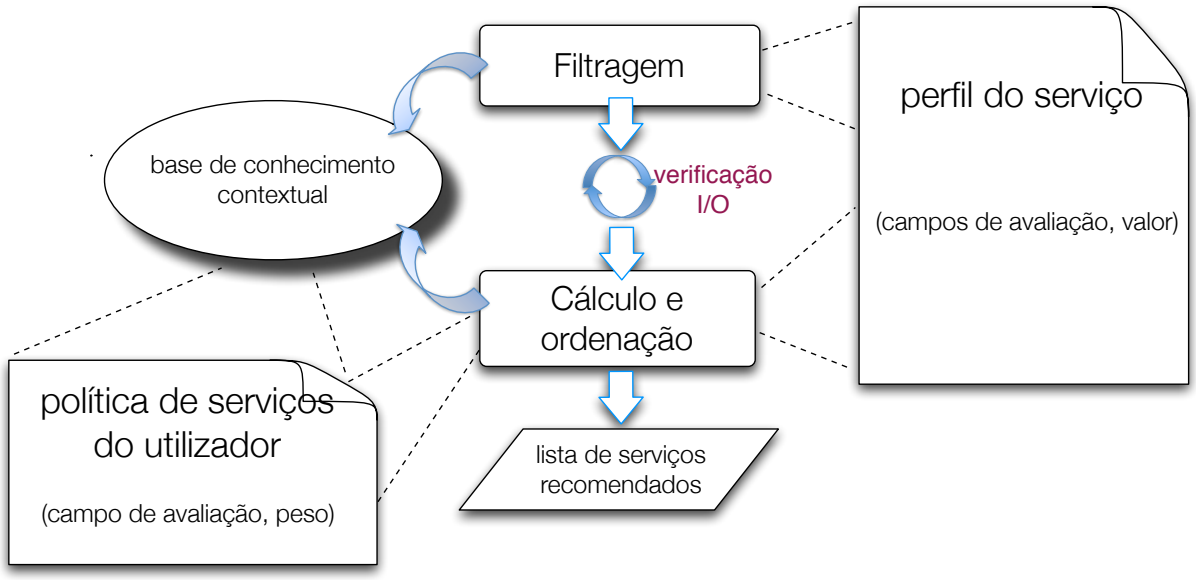


Figura 5.9 – Mecanismo de selecção de serviços, baseado em (Sousa, Carrapatoso et al., 2009).

Este ciclo de entrega de serviços ao utilizador e sua selecção é mantido até este decidir terminar a sua composição ou não haver mais serviços disponíveis que possam ser combinados. Quando o utilizador finalizar a sua composição, a entidade *núcleo do motor de composição* terá criado um serviço composto que contém um *workflow*, que terá como parâmetros de entrada os mesmos do(s) últimos(s) serviço(s) e como saída os parâmetros de saída dos primeiro(s) serviço(s) da composição. Este novo serviço composto pode depois ser guardado, executado, partilhado ou utilizado no processo de composição de um novo serviço. Se, por exemplo, o utilizador optar pelo armazenamento da composição no *repositório de serviços*, essa tarefa é executada pelo *núcleo do motor de composição* (que contém a descrição do novo serviço) que invoca funções do *gestor de serviços*.

5.6.2 Motor de contexto

O *motor de contexto* é responsável por gerir todos os dados contextuais e por raciocinar sobre o contexto. Os outros componentes do sistema interagem com este, solicitando-lhe ou fornecendo-lhe informação contextual instanciada a partir de um modelo ontológico.

Como é ilustrado na Figura 5.10, fazem parte do *motor de contexto* os seguintes componentes: *núcleo do motor de contexto*, *gestor de perfil e preferências*, *agregador de contexto*, *histórico de acções* e *aprendizagem de perfil e preferências*. Este motor interage ainda com duas bases de dados designadas *contexto persistente* e *histórico de acções*. A primeira é responsável por guardar de forma persistente a base de conhecimento contextual, enquanto a segunda mantém o registo do histórico de acções.

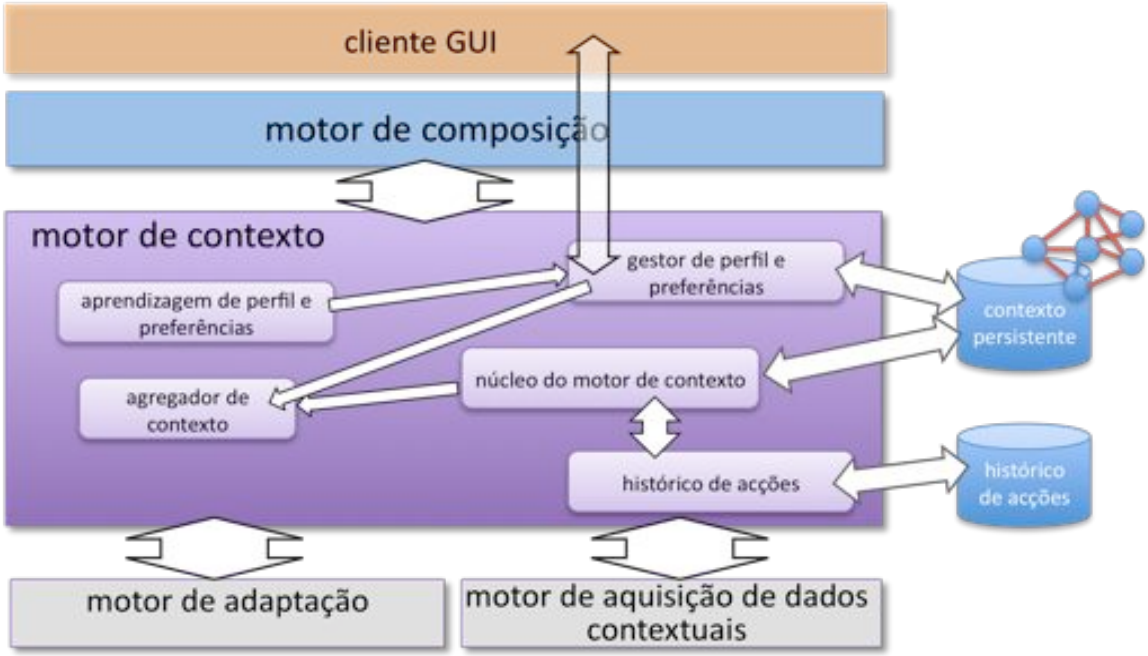


Figura 5.10 – Arquitectura do *motor de contexto*.

Os dados contextuais são fornecidos através do componente *motor de aquisição de dados contextuais* e são adicionados à base de conhecimento contextual deste sistema. Todas as informações contextuais são mantidas numa base de conhecimento, sob a forma de triplas RDF, que são geridas pelo *núcleo do motor de contexto* mas acessíveis aos outros componentes do motor. A base de conhecimento contém informação sobre os domínios abrangidos pelo modelo semântico de informações contextuais (SeCoM) que inclui as ontologias *Actor, Activity, Spatial, SpatialEvent, TemporalEvent, Time, Device, Relationship, Knowledge, Role, Contact, Project, Document*. Como já foi referido na secção 3.2.6 uma base

de conhecimento é constituída por dois componentes internos: a TBOX, que contém as descrições dos conceitos do domínio, e a ABOX, que contém factos sobre as terminologias.

O componente *núcleo do motor de contexto* fornece três funções: armazenamento de contexto, consulta de contexto e inferência de novas informações sobre o contexto. A funcionalidade de armazenamento de contexto suporta armazenamento de informação contextual de forma persistente numa base de dados ou em ficheiros. Esta funcionalidade permite guardar o contexto de forma permanente, manipular grandes volumes de dados RDF e permite reduzir o tempo de carregamento do modelo ontológico cada vez que o sistema é iniciado. A consulta de contexto permite extrair informações contextuais da base de conhecimento. Esta extracção é feita através da uma linguagem de consulta de modelos RDF capaz de obter informações baseada em conhecimento declarativo e inferências básicas.

A linguagem recomendada pelo W3C para a execução destas consultas é a linguagem SPARQL. Para proceder à consulta de contexto são executadas *queries* as quais, através de um mecanismo de verificação (*match*) de padrões de grafos, retornam os resultados que satisfazem essas condições (ex.”saber todos os eventos sobre um determinado tema a decorrer e a sua localização”). A funcionalidade de inferência permite deduzir novos factos a partir da semântica de informações de contexto baseada em ontologias. A inferência é feita através de mecanismos de raciocínio lógico que a OWL suporta e podem ser realizados vários tipos de inferência apresentados na secção 3.2.6 Alguns exemplos de deduções possíveis utilizando o modelo SeCoM são:

- através do raciocínio de classe é possível inferir que um “edifício” é um recurso do tipo “localização física”;
- “Maria” é subordinada de “Marta” e esta é subordinada de “Ana”, então através do raciocínio de propriedade é possível inferir que “Maria” é subordinada de “Ana”;
- se o “Pedro” trabalha com “Joana”, também através do raciocínio de propriedade é possível inferir que “Joana” trabalha com “Pedro”.

O componente *agregador de contexto* é responsável por agregar dados de várias fontes contextuais (ex. localização, serviços de notícias) e relaciona-los com uma entidade (ex. pessoa ou objecto). Sempre que o componente *aquisição de dados contextuais* captura novos dados ele notifica o *agregador de contexto* que existem novos dados contextuais para serem lidos. O *agregador* procede à sua leitura e relaciona-os com uma entidade, convertendo-os ao

mesmo tempo em triplas RDF e actualizando a base de conhecimento contextual. Os tipos de fontes de contexto que fornecem dados genéricos (ex. informação meteorológica na região, cotação da bolsa) são associados a uma entidade genérica que poderá também actualizar a base de conhecimento. Definiu-se ainda que o *agregador de contexto* poderá tratar os dados contextuais elevadamente dinâmicos de forma diferente, não procedendo à sua actualização na base de conhecimento. Isto acontece, porque se os dados forem muito dinâmicos (ex. localização de um utilizador obtida por GPS), as constantes actualizações da base de conhecimento poderão provocar uma alta carga computacional.

O componente *gestor de perfil e preferências* é responsável por gerir informação do perfil e preferências do utilizador de uma forma explícita. Usando o painel de administração, este componente permite ao utilizador inserir, actualizar e remover dados do seu perfil e das preferências que apenas podem ser introduzidos de forma explícita. Todas essas informações são guardadas numa base de dados de forma permanente para utilização futura. Este componente suporta também o serviço de autenticação do utilizador feito através do fornecimento das suas credenciais e tem como objectivo evitar o acesso de utilizadores estranhos ao sistema, identificar os utilizadores e saber qual o seu nível de acesso (utilizador ou administrador).

O componente *histórico de acções* monitoriza as acções executadas pelo componente *núcleo do motor de contexto* e pelo *gestor de perfil e preferências*, registando essas acções na *base de dados do histórico de acções*. Algumas das acções que são monitorizadas são a pesquisa, a inserção, a actualização e a remoção, de forma a ter o histórico de mudanças de informações de contexto das entidades.

O componente *aprendizagem de perfil e preferências* pode alterar preferências e dados de perfil através de um algoritmo de aprendizagem (ex. se um aluno pesquisa muitas vezes uma categoria de livros, essa categoria pode ser adicionada aos temas de interesse do utilizador). Este componente funciona de forma autónoma procurando acções particulares que o utilizador executou e que foram registadas na *base de dados histórico de acções*. Através de um mecanismo poderá alterar uma ou várias preferências do perfil do utilizador. Este mecanismo funciona com base em regras, especificadas no formato (tipo de acção, valor de *threshold*, recurso, propriedade e recurso e propriedade a serem alterados); para o exemplo indicado no início deste paragrafo ficaria (*pesquisa*, 5x, *pessoa*, *categoria*, *pessoa*, *temInteresse*). A pesquisa da acção sobre um recurso é feita na *base de dados histórico de acções* e se o

número de ocorrências dessa acção relativa a uma propriedade ultrapassar o valor de *threshold,* então o valor da propriedade de um recurso é alterado. Uma melhor aproximação poderá ser conseguida através da análise de utilização efectiva em campo da arquitectura e de recolha da avaliação dos utilizadores sobre essa aprendizagem. Os trabalhos (Boughanem and Tmar, 2002), (Tebri, Boughanem et al., 2005) e (Mitaim and Kosko, 1998) apresentam algumas aproximações mais avançadas para a aprendizagem de perfis e preferências que poderão ser adaptadas para este componente.

Na Figura 5.11 é ilustrado o diagrama de sequência do processo relativo à actualização do perfil de um utilizador. O cliente cria um proxy que irá invocar o serviço responsável por fornecer o serviço de actualização do perfil. O *gestor de perfil e preferências* é o componente que implementa esse serviço e irá ler os dados do utilizador da *bases de dados contexto persistente* devolvendo os dados ao cliente. Simultaneamente, lança um evento para o *histórico de acções*, indicando que o utilizador fez uma consulta de determinados dados, que será guardado na base de *dados histórico de acções*. Quando o utilizador proceder à actualização dos dados é executado um processo semelhante. O *gestor de perfil e preferências* irá fazer a actualização dos dados na respectiva *base de dados de histórico de acções* e notificar o *histórico de acções*, indicando que o utilizador realizou uma operação de actualização em determinados dados.

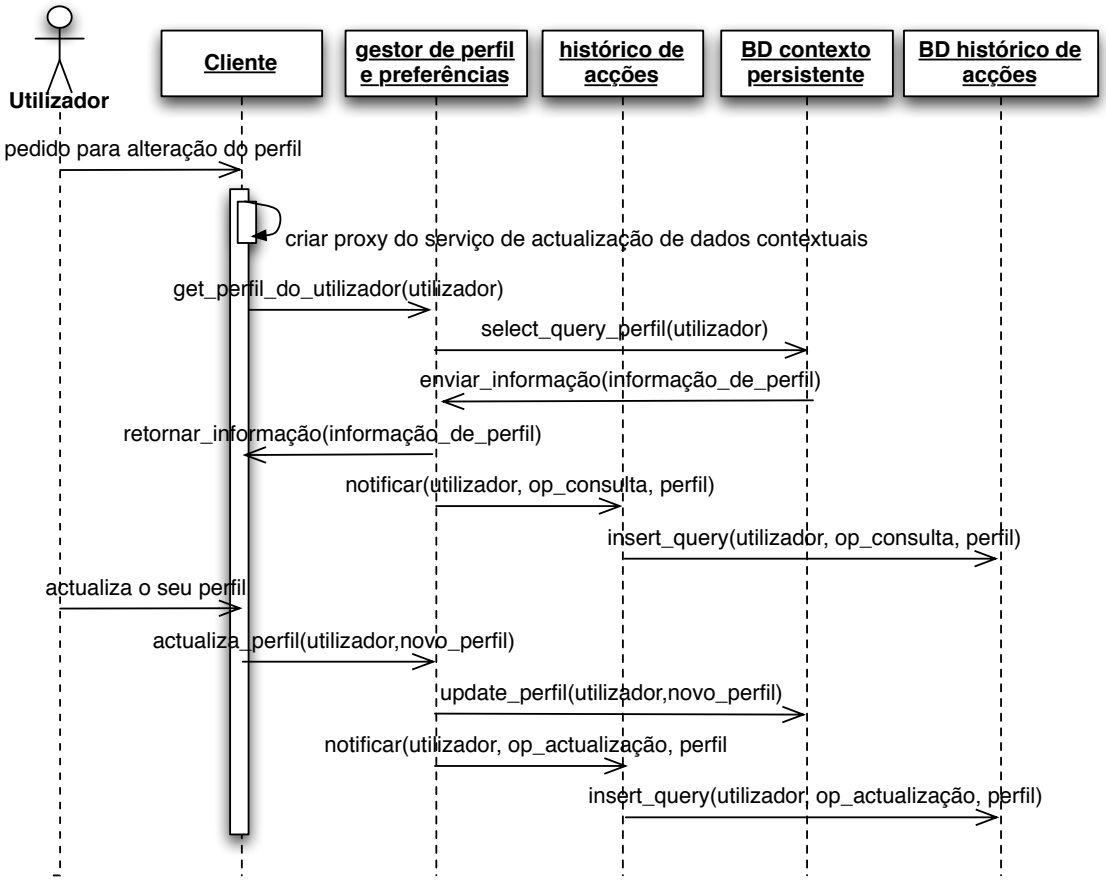


Figura 5.11 - Diagrama de sequência do processo actualização do perfil do utilizador.

A Figura 5.12, mostra o diagrama de sequência do processo de autenticação que acontece quando o utilizador inicializa o seu cliente. Este processo faz o controlo de acesso, a identificação do utilizador no sistema e o reconhecimento do terminal que o utilizador está a usar no momento, para posteriormente, se necessário, adaptar os conteúdos que lhe forem entregues. Neste processo é também instanciado o *agregador de contexto* associado ao utilizador e é feita a subscrição das fontes de contexto a esse agregador.

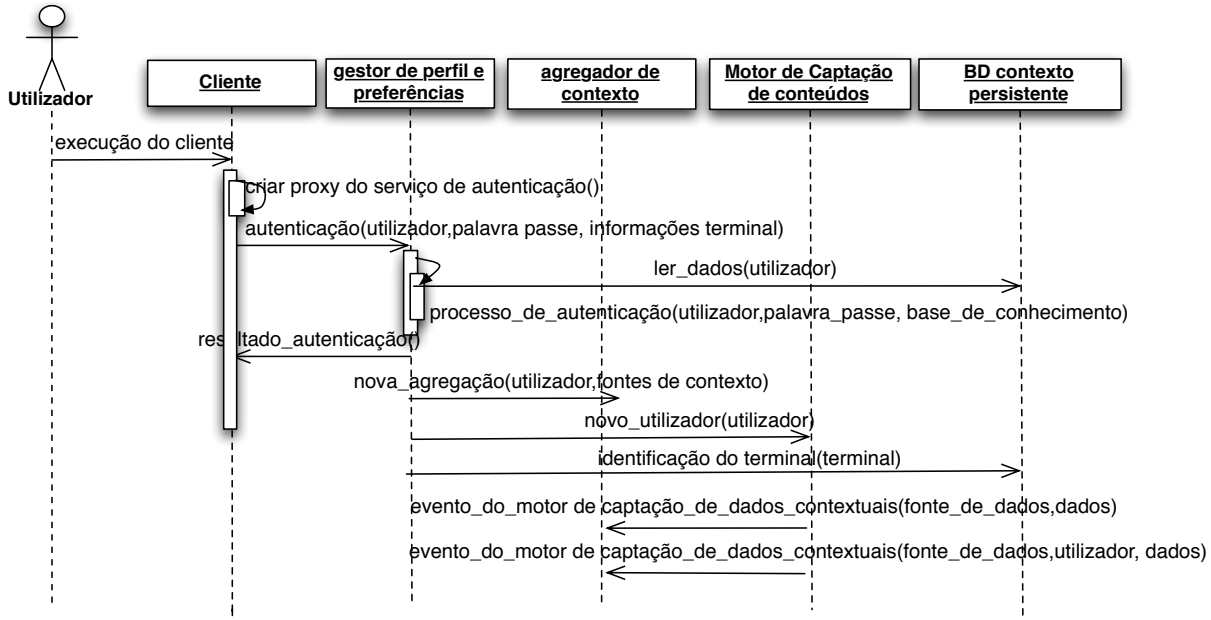


Figura 5.12 – Diagrama de sequência do processo de autenticação.

5.6.3 Motor de aquisição de dados contextuais

O motor *aquisição de dados contextuais* tem como principal função recolher dados de várias fontes e fornecê-los ao *motor de contexto*. Na Figura 5.13, são representados os componentes que constituem este motor: os diversos elementos de *abstracção de contexto* e as suas relações com as *fontes de contexto* e com o *motor de contexto*.

Figura 5.13 – Arquitectura do *motor de aquisição de dados contextuais*.

Os dados de contexto são fornecidos pelas *fontes de contexto*, que podem ser físicas (ex. sensores) ou componentes de software (ex. *Web Service* que fornece as previsões meteorológicas). Existem situações em que as fontes de contexto utilizam uma representação

ou formatação de dados própria, havendo a necessidade de adaptar esse dados para uma representação semântica comum. Para isso, este motor possui componentes *abstracção de contexto* que vão implementar essa camada de abstracção e de transformação, fornecendo uma visão comum de todas elas. Os seguintes exemplos são casos em que essa adaptação pode ocorrer: um serviço que fornece a temperatura de um determinado local em graus *Fahrenheit* e é convertida para graus *Celsius*, ou a transformação de uma localização absoluta dada pelas coordenadas de onde o utilizador se encontra, para uma localização referencial ("campo de ténis"). Por cada *fonte de contexto* existe uma *abstracção de contexto* independente que deverá capturar os dados, proceder ao seu tratamento caso seja necessário e notificar o *agregador de contexto* acerca da existência de novos dados contextuais.

A Figura 5.12 mostra o diagrama de sequência do processo de passagem dos dados contextuais capturados para o *motor contextual*. Quando o administrador inicia o sistema, são inicializados todos os quatro motores que constituem a arquitectura (ver Figura 5.5 da secção 5.5). Quando o *motor de aquisição de dados contextuais* é iniciado, são criados os diversos componentes *abstracção de contexto* (um por cada fonte) que irão ler dados da *fonte de contexto* respectiva. Estes componentes têm propriedades de temporizador para efectuarem leituras periódicas e notificam o *agregador de contexto* quando um novo dado for lido.

Como já foi referido anteriormente, quando um utilizador é autenticado, é criado um *agregador de contexto* para esse utilizador que agrega dados de diversas fontes particulares a esse utilizador (ex. localização). Posteriormente é feita a actualização dos dados contextuais desse utilizador no formato de triplas RDF na base de dados de conhecimento através do *núcleo do motor de contexto*, ficando a informação contextual disponível para ser utilizada pelos outros componentes da arquitectura.

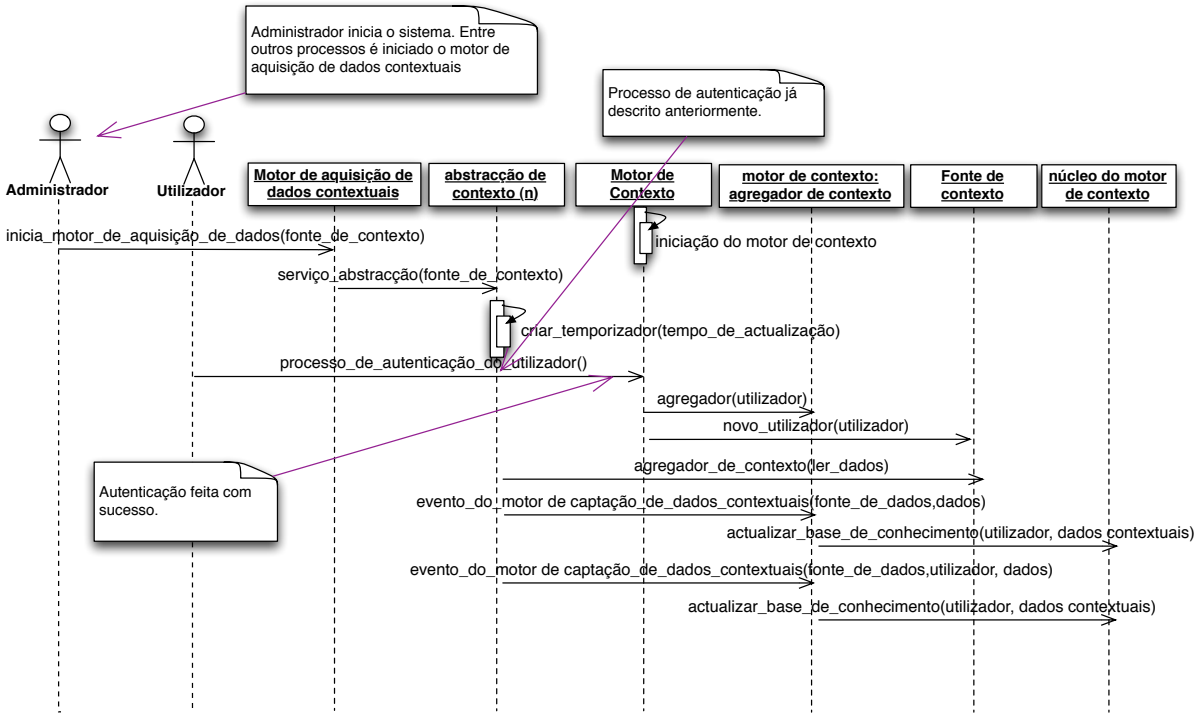


Figura 5.14 – Diagrama de sequência do processo de aquisição e tratamento de dados contextuais.

5.6.4 Motor de adaptação de conteúdos

De modo a possibilitar a consulta de informação em diversos tipos de dispositivos, os conteúdos devem ser disponibilizados de uma forma independente. Existem dois tipos de conteúdos fornecidos ao utilizador: os conteúdos resultantes da execução de serviços e os conteúdos de interesse do utilizador, seleccionados com base no seu perfil, preferências ou contexto, e que são fornecidos através de um alimentador de notícias (*RSS feeds*).

Estes conteúdos são fornecidos ao cliente através do *motor de adaptação* de conteúdos ilustrado na Figura 5.15, que é constituído por dois componentes: *adaptador de conteúdos*, que tem a responsabilidade de adaptar os conteúdos ao terminal do utilizador; e *selecção de conteúdos*, que tem a responsabilidade de seleccionar as informações contextuais do *motor de contexto* subscritas para serem entregues ao utilizador.

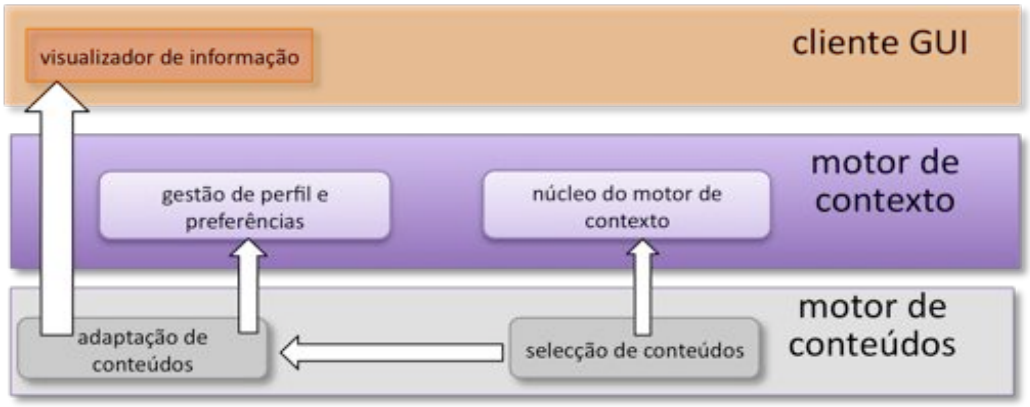


Figura 5.15 – Arquitectura do *motor de adaptação*.

Dos dois tipos de conteúdos produzidos pelo sistema e identificados anteriormente surgem dois modo de operação. (1) Quando um serviço é executado, os resultados são entregues ao componente de *adaptação de conteúdos*, que irá consultar o perfil descritivo do dispositivo que invocou essa operação e adaptar esse conteúdo ao terminal do utilizador com base nas informações desse perfil. Essa descrição do perfil foi fornecida previamente na operação de autenticação sendo guardada na base de conhecimento contextual. (2) Para entregar informação do interesse do utilizador, o *selector de conteúdos* consulta o componente *perfil e preferências do utilizador* e com base nessa informação selecciona informação do *agregador de contexto* ou da *base de conhecimento* contextual para ser entregue ao utilizador através do *adaptador de conteúdos*. Um exemplo da utilização deste serviço fornecer informações sobre eventos de possível interesse do utilizador a decorrer no campus universitário.

Embora nesta secção seja apresentada uma solução para a adaptação de conteúdos baseada em contexto, trata-se de uma solução bastantes simples, em que não são consideradas funções avançadas de adaptação contextual capaz de adaptar qualquer serviço e conteúdo, considerando as características como o perfil do utilizador, do dispositivo, ou da rede (ex. se o utilizador estiver no cinema, o conteúdo deverá ser-lhe fornecido por exemplo em texto ou sem som; ou se o resultado de um serviço for um *streaming* de vídeo que poderá ter que ser adaptado caso o utilizador mude de terminal ou de rede.). Estudos mais detalhados sobre este problema e contribuições para o melhoramento deste componente podem ser encontradas nos trabalhos (Oliveira, Roque et al., 2001), (Martins, Carrapatoso et al., 2005) relacionados com o projecto VESPER (VESPER, 2000).

5.6.5 Aplicações clientes

Todas as funcionalidades da arquitectura disponibilizadas ao cliente são invocadas através de *Web Services*. Deste modo a aplicação cliente irá interagir com os serviços de que necessita através de *proxies*, que irão propagar as invocações aos respectivos *Web Services* que se encontram no servidor da arquitectura. Esses proxies serão instanciados a partir do momento em que o utilizador ou administrador seleccionar uma determinada funcionalidade específica (ex. compor serviço, actualizar perfil, introduzir informações de forma explícita no modelo contextual).

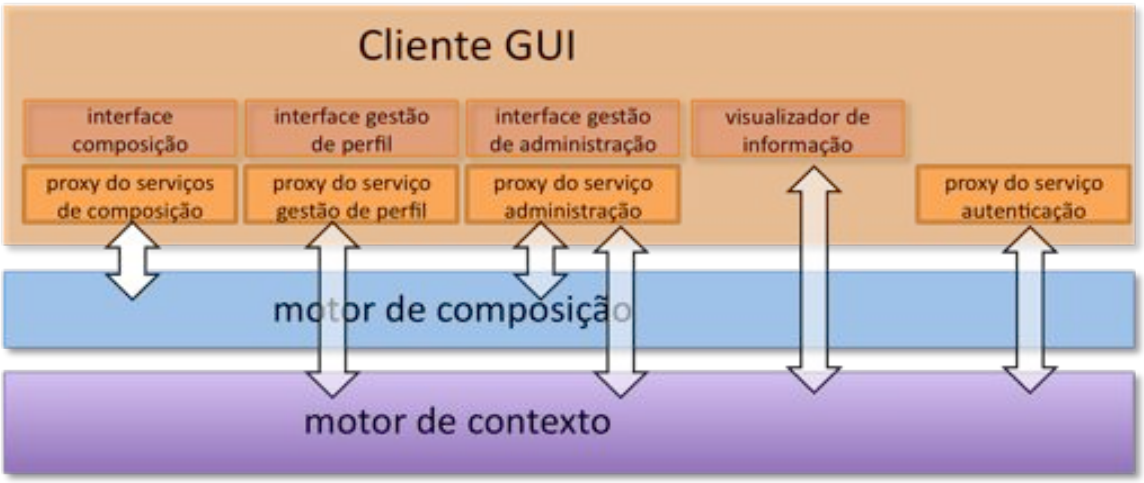


Figura 5.16 – Arquitectura de uma aplicação cliente.

A Figura 5.17 ilustra um diagrama de sequência parcial dos casos de utilização que fazem uso dos serviços disponibilizados pela arquitectura. As primeiras mensagens que aparecem no diagrama correspondem ao processo de autenticação já explicado na secção 5.6.2. Este processo tem em consideração que as credenciais de autenticação do utilizador foram previamente fornecidas e encontram-se guardadas num ficheiro. Este processo de autenticação é executado quando a aplicação cliente é iniciada. Do mesmo modo, quando é iniciado um processo de composição de um serviço é criado o proxy do serviço de composição responsável por invocar as diversas funções proporcionadas pelo *Web Service* que fornece essa função. O mesmo processo é aplicável aos restantes serviços.

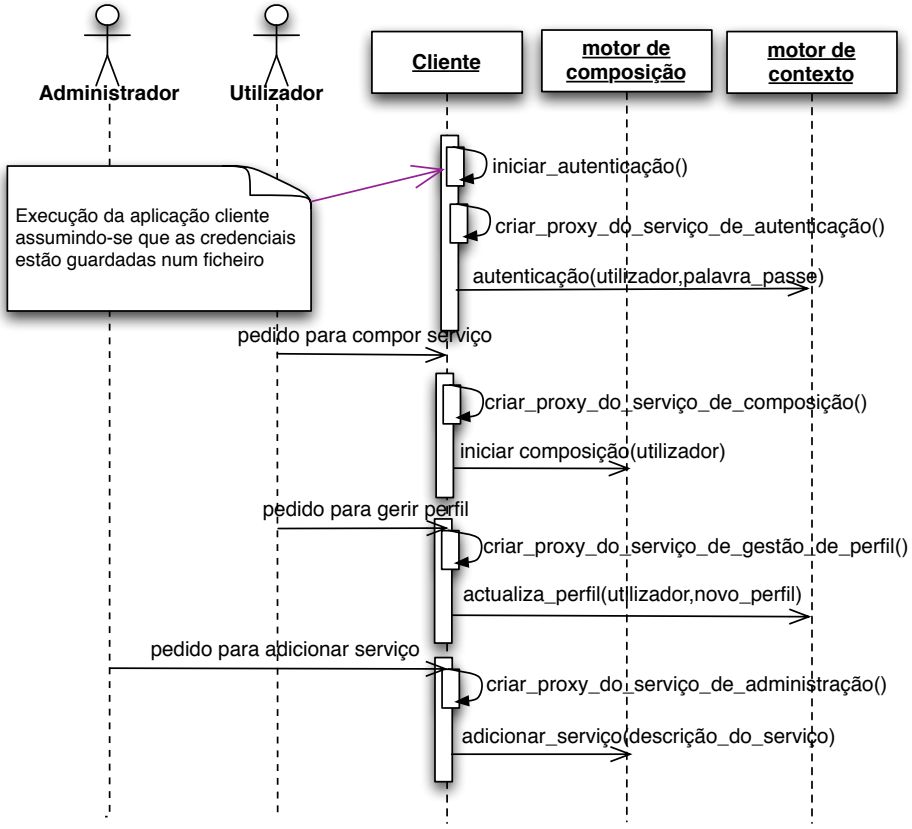


Figura 5.17 – Diagrama de sequência do processo actualização do perfil do utilizador.

5.6.6 Modelo Contextual Semântico

Alguns sistemas *context-aware* apresentam a informação de contexto como estruturas de dados simples ou modelada como objectos de programação, tornando a sua reutilização e partilha mais difícil e ineficiente. As linguagens de classificação podem ser utilizadas para estruturar, classificar, modelar e representar conceitos e relações de domínios, o que permite que uma comunidade possa adoptar o mesmo conjunto de expressões com o objectivo comum de representar informação com um determinado nível semântico de um modo uniforme, auxiliando a partilha de contexto. Entre as várias formas de representar modelos de dados de um determinado domínio estão as *ontologias*, que fornecem um vocabulário comum e um entendimento partilhado para representar conhecimento sobre um domínio. As ontologias têm sido largamente exploradas pela *Semantic Web*, permitindo que a semântica da informação seja mais facilmente integrada, compreendida e partilhada tanto por humanos como por agentes de software.

Resumindo, um modelo sensível ao contexto baseado em ontologias tem diversas vantagens: permite inferir novas informações a partir de informações já representadas por essas

ontologias, fomenta a reutilização, troca e partilha de informação contextual entre sistemas *context-aware*, oferece um modelo de informação por domínios modular e escalável e separa o modelo de informação contextual do modelo de programação.

5.7 Cenários de aplicação do modelo

O modelo para o suporte de composição contextual de serviços proposto nesta tese pode ser usado para numa grande variedade de aplicações. Das diversas características do modelo, evidenciam-se as seguintes relativamente à generalidade aplicacional:

- o facto de se tratar de uma arquitectura SOA com suporte de *Web Services* oferece grande flexibilidade e independência, relativamente ao tipo de aplicações que podem fazer uso dos seus serviços;
- ao utilizar um modelo contextual ontológico modular, oferece a possibilidade de adicionar ou remover novos domínios contextuais adequados aos novos cenários de utilização;
- alguns componentes especificados no modelo (ex. *motor contextual*, *motor de adaptação*) oferecem um conjunto de funcionalidades suficientemente genéricas para serem usadas em diversas situações.

5.7.1 Comércio electrónico

O modelo proposto poderia ser usado num cenário de aplicação para enriquecer o comércio electrónico, em que os clientes iriam receber informação de produtos ou promoções com base nas suas preferências e contexto. Em situações cujos serviços oferecidos não respondessem às suas expectativas, poderiam compor um novo serviço com base nos serviços fornecidos pelo vendedor.

Outro cenário possível seria o de um centro comercial que utilizasse a arquitectura proposta para fornecer informação sobre as suas promoções aos clientes que passam próximos da loja e permitisse a composição de um novo serviço através dos serviços que possui. Assim, quando um suposto cliente entrasse no centro comercial ser-lhe-iam oferecidos diversos serviços, entre os quais os de procura de produtos, serviços de navegação e serviços de *streaming* para produtos em lojas, todos relacionados com o centro comercial e com a sua lista de compras ou

interesses. Este serviços apareceriam entre os primeiros porque as políticas de serviço contextuais do utilizador dariam importância à localização, à avaliação desses serviços por outros utilizadores e à sua taxa de utilização. Deste modo, o utilizador poderia compor um serviço que procurasse um produto e lhe mostrasse um *streaming* de vídeo de uma câmara que apontasse para esse produto.

5.7.2 Campus universitário

Para validar a infra-estrutura escolheu-se a utilização deste modelo num campus universitário como ilustrado na Figura 5.8.

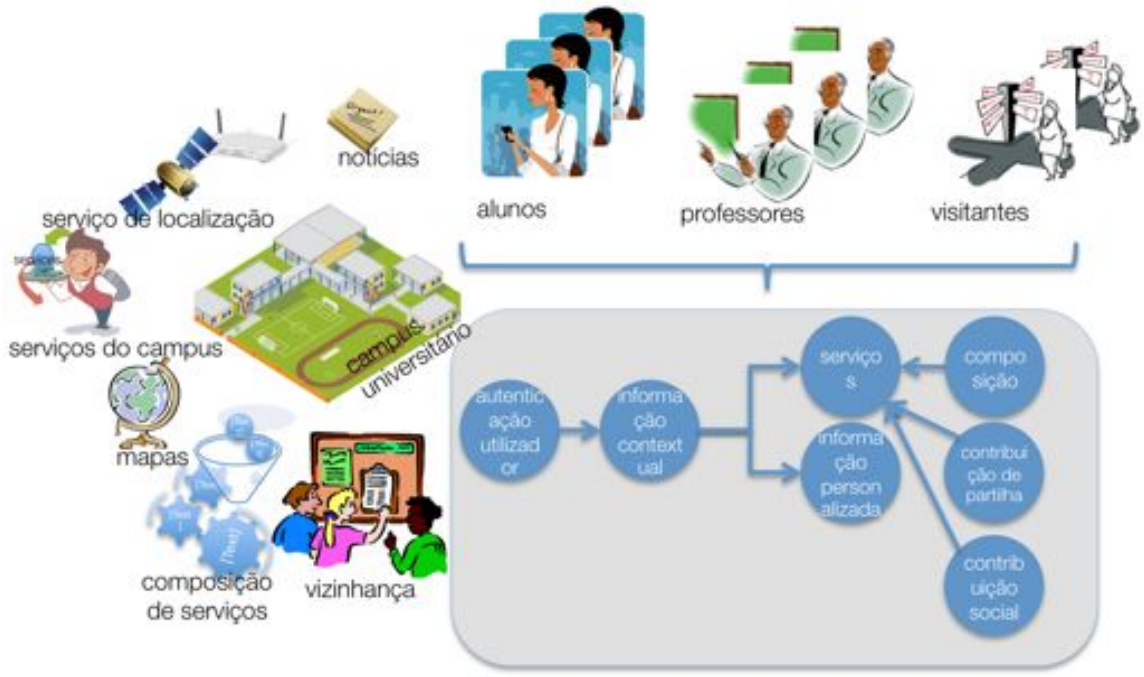


Figura 5.18 – cenário de utilização da *iCas* num *campus* universitário.

A sua aplicação tem como objectivo apoiar alunos e professores na sua vida académica, permitindo-lhes criar novos serviços e manterem-se actualizados de modo a melhorar a sua integração pedagógica e social. Assim, quando um utilizador chega ao campus e conecta o seu dispositivo móvel (*smartphone*/tablet/netbook/portátil) à rede sem fios terá que se autenticar nesse serviço. Esta autenticação será utilizada para se identificar como utilizador na rede sem fios e na arquitectura *iCas*. O campus universitário possui vários serviços que podem ser integrados na arquitectura *iCas*, entre eles: um serviço de localização (Carvalho, 2007), uma plataforma de *e-learning* que fornece informações sobre as unidades curricular e cursos, conteúdos pedagógicos, entre outros; plataforma da biblioteca que oferece serviços *Web* aos

seus utilizadores e serviços académicos que fornecem informações administrativas, como notícias, avisos, eventos, entre outros.

Em seguida são descritos alguns exemplos da utilização do modelo proposto, adequados ao cenário de um *campus* universitário:

- um aluno deseja saber se os livros recomendados pelo docente da disciplina de Matemática estão disponíveis na biblioteca. Através desta arquitectura, poderá ser-lhe apresentado um conjunto de serviços suportados pelas diversas plataformas existente do campus. Assim, o utilizador para realizar a tarefa descrita poderia combinar o serviço que lhe oferece a bibliografia da disciplina (fornecido pela plataforma de e-learning) com o serviço de pesquisa de um livro na biblioteca mais próxima (fornecido pela plataforma que suporta as diversas bibliotecas); e até a marcação de levantamento, com base no serviço que fornece a disponibilidade da sua agenda;

- um utilizador (docente, aluno, etc) tem indicado no seu perfil o interesse por determinadas áreas científicas e a disponibilidade para receber alertas sobre eventos que aconteçam no campus. Deste modo, o utilizador poderá ser notificado quando chega ao *campus* sobre um evento que vá decorrer. Nessa notificação é lhe mostrado um pequeno resumo do evento e uma opção para saber mais informações. Nessas informações, o utilizador poderá consultar a localização do evento e o seu grau de actividade, assim como se tem alguns dos seus amigos nesse evento;

- o aluno tem agendada uma actividade de estudo para uma disciplina. Quando se aproximar a hora poderão ser-lhe apresentado no seu (*smartphone*/tablet/netbook/portátil) diversos serviços, relacionados com estudo, entre os quais, um que lhe mostra num mapa do *campus* os diversos grupos de estudo na vizinhança e o grau de actividade de cada um deles. Posteriormente, o aluno poderá ainda adicionar um serviço de mensagens instantâneas por forma a mostrar a sua intenção de se juntar a um grupo.

5.8 Resumo

Neste capítulo foi apresentado um modelo para a composição dinâmica e contextual de serviços em dispositivos móveis baseado em tecnologias para a *Semantic Web*. Para tal, foi inicialmente apresentada uma visão geral do modelo proposto, onde foram introduzidas as directrizes que um modelo deste género deverá cumprir. A partir destas directrizes foram apresentadas as funcionalidades que o modelo deverá suportar e os principais componentes da sua arquitectura. Foram apresentados os casos de utilização que as aplicações que suportem este modelo deverão implementar. Posteriormente foi descrito o comportamento do modelo na perspectiva do utilizador e do administrador. Seguidamente foi descrito cada um dos componentes que constituem a arquitectura. Essa descrição incluiu dois tipos de diagramas: diagramas que ilustram a sua arquitectura interna e iteração com os restantes componentes, e diagramas de sequência que pretendem mostra o modo de funcionamento. Finalmente, foram apresentados alguns cenários em que o modelo proposto nesta Tese poderá ser aplicado.

Capítulo 6

IMPLEMENTAÇÃO

O modelo apresentado no capítulo anterior permitiu representar genericamente uma arquitectura e descrever cada um dos componentes e funcionalidades com um nível de abstracção elevado. Este capítulo preocupa-se com as características de implementação, instalação e avaliação mais relevantes do modelo proposto. É apresentado um ambiente que suporta a composição dinâmica sensível ao contexto de *Web Services,* que implementa um protótipo com as características mais importantes para a validação do modelo proposto. Durante a apresentação são ainda discutidas escolhas de implementação e as razões para as opções efectuadas.

6.1 Arquitectura do Sistema

O modelo de composição de serviços sensíveis ao contextuo e preferências do utilizador pretende ser o mais genérico possível, de forma a poder adaptar-se a diversos cenários. Contudo, sendo impossível a validação para os diversos tipos de cenários, optou-se um cenário de um campus universitário que permitisse a validação em tempo útil das suas partes mais relevantes. Com os resultados obtidos para este cenário será possível perspectivar a validade da utilização deste modelo noutros cenários semelhantes.

O protótipo do arquitectura *iCas* para a composição de serviços em dispositivos móveis sensível ao contexto é baseado em tecnologias para a *Semantic Web*, incluindo os seguintes componentes, ilustrados na Figura 6.1:

- *motor de composição* – implementa as funcionalidades de pesquisa, selecção,

composição, execução, armazenamento e partilha de serviços. Fornece ainda uma interface para que aplicações, que possam aceder a *Web Services*, possam usufruir das funcionalidades anteriormente descritas;

- *motor de contexto* – fornece todas as funções para aceder e gerir a informação contextual na base de conhecimento. Além de permitir a consulta e alteração de informação, regista todas as operações realizadas sobre a base de conhecimento contextual possibilitando assim obter um histórico das acções implícitas ou explícitas dos utilizadores no seu contexto. Através de mecanismos de inferência permite deduzir novas informações de contexto a partir das informações actuais;
- *motor de aquisição* – permite a obtenção de dados de fontes de contexto, como sejam o caso de serviços ou sensores. Este motor irá depois entregar ao motor contextual os dados que foram capturados para que possam ser usados pela arquitectura;
- *motor de adaptação* – através de informações das propriedades dos terminais este motor poderá transformar o formato dos resultados.

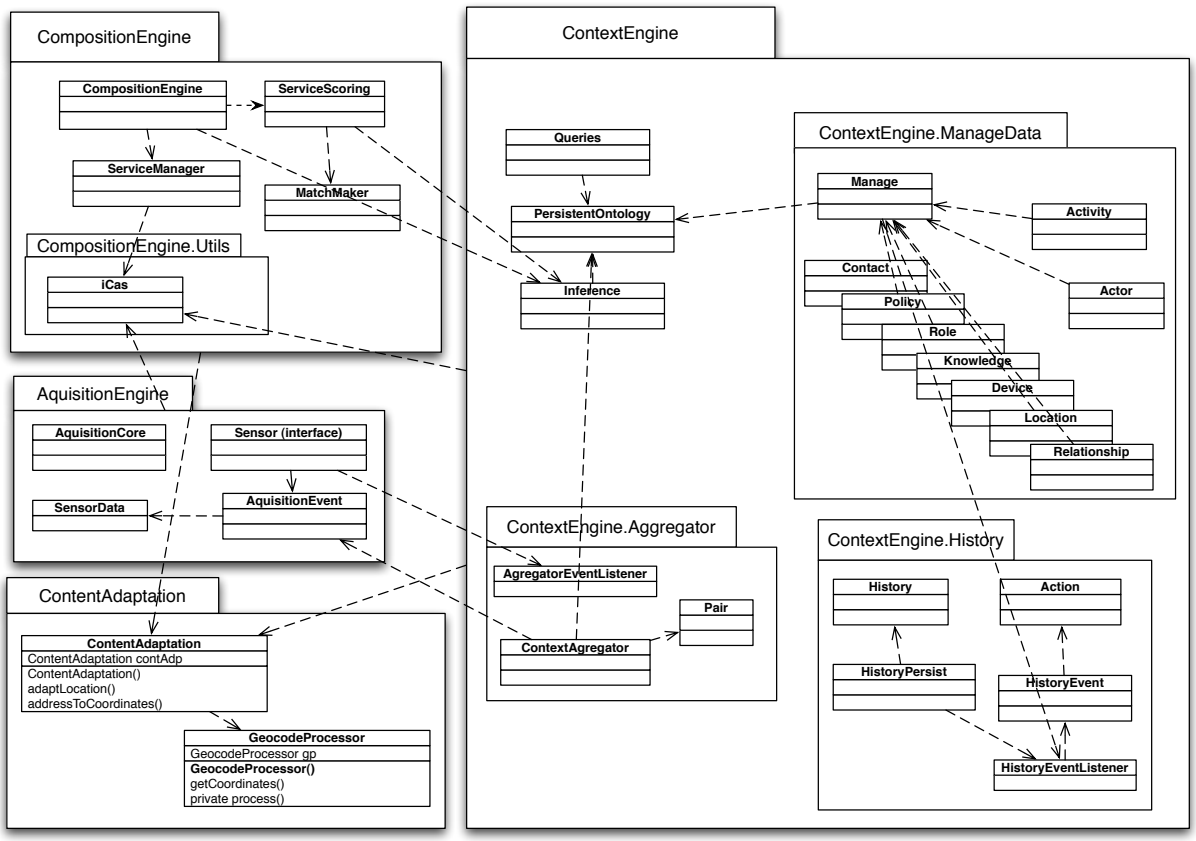


Figura 6.1 – Diagrama de interacção de classes do protótipo da arquitectura *iCas*.

Tendo em consideração o modelo apresentado na secção 5.6, este protótipo segue o modelo cliente/servidor fornecendo os seus serviços ao cliente através de *Web Services*. Esta configuração coloca a carga computacional do motor de inferência no lado de um servidor de aplicações, que é resulta na dedução de nova informação contextual e na composição de serviços, possibilitando ainda a oferta de outros serviços que façam uso da inferência em dispositivos que tenham limitações de capacidade de memória e processamento. Para suportar este tipo de configuração, instalou-se e configurou-se um servidor de aplicações, que executa do protótipo os quatro motores descritos anteriormente. Para tal, foi necessário escolher um servidor de aplicações e um conjunto de APIs que auxiliassem o desenvolvimento de algumas tarefas complexas como trabalhar com serviços OWL-S, o suporte de dados RDF e as ontologias OWL.

A construção do núcleo do sistema e dos diversos motores baseou-se na utilização da linguagem de programação Java, juntamente com o servidor de aplicações *Glassfish* e um conjunto de APIs de desenvolvimento.

Os motivos de escolha da linguagem Java para desenvolver o protótipo que é apresentado neste capítulo, estão relacionados com o facto de ser uma linguagem independente da plataforma e existir um vasto conjunto de APIs em Java para o desenvolvimento de aplicações na área da *Semantic Web*.

A escolha do servidor de aplicações ocorreu a partir de um vasto número de servidores de aplicações existentes no mercado, que suportam a instalação e execução de *Web Services*. De entre os diversos servidores de aplicações, alguns deles utilizam como linguagem de programação o Java, sendo também conhecidos como "servidor de aplicações J2EE". Como se escolheu a linguagem de programação Java para o desenvolvimento deste protótipo, embora não sendo necessário, deu-se preferência à utilização de uma plataforma que permitisse o desenvolvimento de *Web Services* em linguagem Java. Entre os diversos servidores de aplicações, existem no mercado diversas soluções populares distribuídas sob licença de software livre, pelo menos para utilização não empresarial. Destacam-se alguns dos mais difundidos: *WebSphere Application Server Community Edition* – WASCE (IBM, 2009), mantido pela *IBM*; *Apache Geronimo* (Apache, 2009) mantido pela *Apache Software Foundation*, *JBoss* (JBoss, 2009) mantido pela *Red Hat*; *Glassfish Application Server* (Glassfish, 2009) mantido pela *Sun Microsystems*; *JOnAS* (OW2, 2009) mantido pelo consorcio *OW2*. A escolha do servidor de aplicações recaiu sobre o *Glassfish Application*

Server, pois trata-se de um sistema gratuito mantido por uma empresa com créditos na área, que mantém este projecto activo sendo actualizado regularmente, possui uma extensa documentação de suporte e integra-se bem com os principais ambientes de desenvolvimento de aplicações em Java (*Integrated Development Environment* – IDE), como o *Netbeans* (Netbeans, 2009) e o *Eclipse* (Eclipse, 2009).

Para lidar com o modelo ontológico para o contexto optou-se por utilizar a API Jena. Esta API, já apresentada na secção 3.3.1.2, fornece um conjunto de componentes que facilitam a exploração dos dados RDF e das ontologias OWL, tais como: um motor de inferência baseado em regras, um motor de *queries* SPARQL a dados RDF, e uma API para manipular os dados RDF e OWL quer em memória, ficheiros ou bases de dados. Além destas características, outros factores levaram à opção pela sua utilização: o facto de ser bastante utilizada na *Semantic Web*, possuindo por isso uma comunidade de utilizadores bastante activa que contribui com actualizações regulares, documentação e apoio através de listas de discussão; é desenvolvida em Java e fornece uma API para desenvolvimento na mesma linguagem; tem licença BSD (FreeBSD, 1992-2010) .

Embora a *framework Jena* inclua um conjunto de motores de inferências (Hewlett-Packard, 2009), nenhum deles suporta a máxima expressividade dos construtores OWL (Parsia and Sirin, 2004) oferecida pelo motor de inferência *Pellet*. Escolheu-se este *reasoner* porque, além de permitir essa funcionalidade, oferece ainda outras características já apresentadas na secção 3.3.1.1. Além disso, integra-se bem com a API Jena e a API OWL (Volz, Horridge et al., 2003) e tem uma licença AGPLv3 (GNU, 2007) que pode ser utilizada para aplicações de código aberto.

Para suportar as funcionalidades relacionadas com a composição e execução de serviços OWL-S, foi necessário procurar APIs que suportassem um conjunto de características básicas, como a leitura e escrita das ontologias *Service*, *Profile*, *Process* e *Grounding*, a criação de objectos OWL e OWL-S, e a execução de descritores OWL-S. Dos diversos projectos relacionados com *Web Services Semânticos* desenvolvidos na maioria pela comunidade académica destacam-se as seguintes APIs:

- OWL-S API desenvolvida na Universidade de *Carnegie Mellon* (Srinivasan, 2005);
- OWL-S API desenvolvida no *MindSwap* (Sirin, 2004);

- ETTK *for Web Services* desenvolvida na IBM (Akkiraju, 2003).

A escolha recaiu sobre a API OWL-S desenvolvida no *MindSwap*, feita em Java, que permite a leitura, escrita e execução de descrições de serviços OWL-S. Esta API suporta a versão OWL-S 1.1 e fornece um motor de execução capaz de invocar processos atómicos que tenham interface WSDL e UPnP. Suporta ainda a utilização de construtores de controlo *Sequence*, *Unordered* e *Split*, deixando de fora construtores de controlo condicionais como o *If-Then-Else* e *RepeatUntil*. Posteriormente este projecto foi retomado em 2009 por Thorsten Möller (Möller, 2009), o código da API foi completamente reformulado e foi publicada a versão 2.0 da OWL-S API, passando a suportar a linguagem OWL-S v1.2 e também os construtores de controlo condicionais que na versão anterior não eram suportados. A data de publicação da versão 2.0 ocorreu durante a fase de desenvolvimento do protótipo e devido às diferenças significativas na API optou-se por continuar o desenvolvimento deste protótipo com a versão 1.1.0.

Nas secções seguintes são descritas as implementações dos diversos motores que compõem o protótipo da arquitectura *iCas*.

6.2 Motor de composição

Considerando a especificação do motor de composição definida na secção 5.6.1, o protótipo *iCas* implementa as seguintes características funcionais relativas aos serviços, usando *Web Services*: composição, procura, selecção, execução, armazenamento e partilha de serviços. Este motor de composição é composto por quatro classes como ilustra a Figura 6.2: *CompositionEngineCore*, *ServiceManager*, *ServiceScoring* e *MatchMaker*, que serão apresentadas em seguida, sendo abordadas com mais detalhe ao longo desta secção.

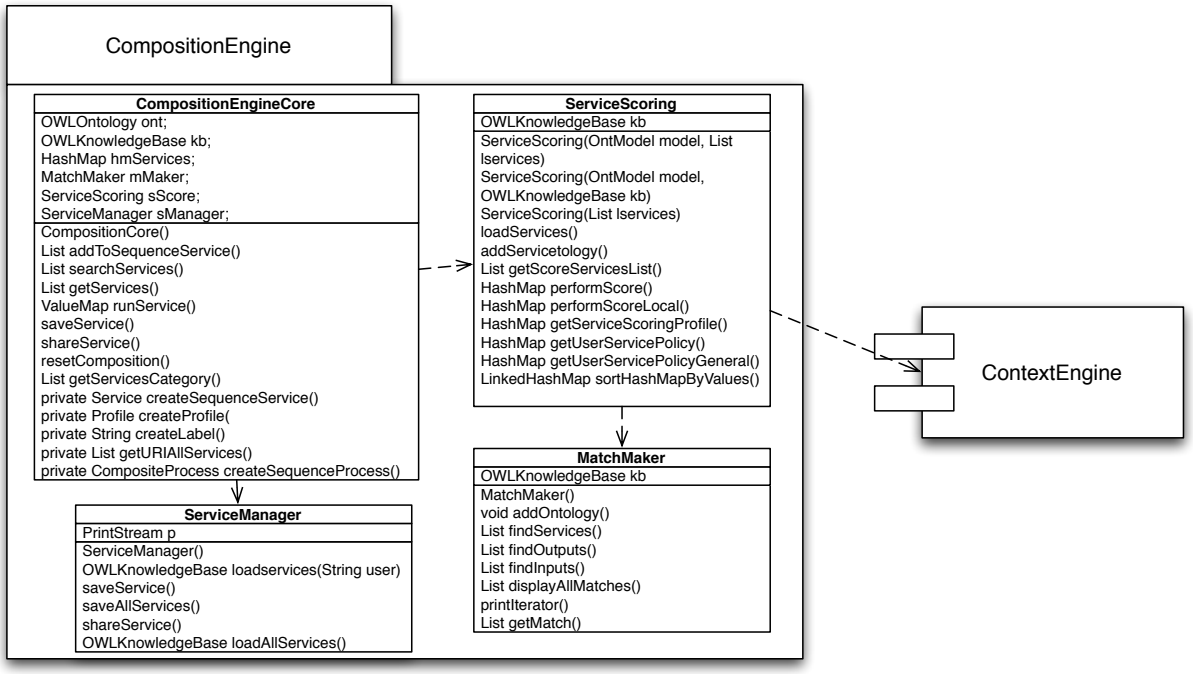


Figura 6.2 - Classes que compõem o *motor de composição*.

A classe *CompositionEngineCore* consiste num *Web Service* que implementa as funcionalidades relativas à composição de serviços e fornece, entre outros, os seguintes métodos através da sua interface WSDL:

- *searchServices*: permite a procura de serviços por categoria;
- *addToSequenceService*: cria um processo de composição no servidor associado a um utilizador. Cada vez que é invocado, adiciona à composição o serviço que lhe é passado como parâmetro e retorna ao utilizador uma lista contextual de serviços;
- *getServices*: devolve os serviços privados do utilizador juntamente com os restantes serviços comuns disponíveis na plataforma. Pode ser usada para obter a lista de todos os serviços disponíveis para o utilizador quando é iniciada a composição;
- *runService*: executa um serviço atómico ou composto e retorna os seus resultados;
- *saveService*: guarda um serviço na biblioteca privada do utilizador;
- *shareService*: partilha um serviço com os restantes utilizadores da plataforma.

A classe *ServiceManager* implementa os métodos para gerir as descrições de serviços numa base de conhecimento, e será utilizada para a gestão dos serviços do utilizador e dos serviços comuns a todos os utilizadores. Entre as opções de gestão incluem-se o carregamento de

descrições de serviços através do protocolo http, ou que estejam localizados no sistema de ficheiros, gravar serviços no sistema de ficheiros, adicionar e remover serviços. Para saber que descrições de serviços o *ServiceManager* deve carregar, existe um ficheiro que contém o URL dos serviços.

As classes *ServiceScoring* e *MatchMaker* funcionam como classes auxiliares da classe *CompositionEngineCore*. Estas duas classes permitem implementar o mecanismo de oferta de serviços ao utilizador baseando-se no seu contexto e nas suas preferências, que será explicado na sub-secção seguinte.

Quando a arquitectura *iCas* é iniciada, é lido um ficheiro de configuração que contém os URLs de todos os serviços que devem ser carregados para a base de conhecimento comum. Os serviços relativos a cada utilizador são carregados quando se inicia o cliente. Neste caso, cada utilizador tem um ficheiro próprio, que contém os serviços que devem ser carregados e que vai sendo actualizado mediante as funções que executa (ex. criar ou remover serviços).

Quando um utilizador selecciona a interface para a composição de serviços na aplicação cliente, esta faz um pedido ao *motor de composição* para lhe serem retornados os serviços disponíveis na plataforma. A partir do momento em que o utilizador selecciona um serviço da lista de serviços disponíveis, é iniciada a composição de um serviço e é criada uma sessão de composição no motor de composição. Esta sessão é identificada pelo seu nome do utilizador e é capaz de manter o estado de uma composição de serviços. O cliente dá instruções ao motor de composição indicando qual o serviço escolhido para adicionar à composição e este cria um *workflow* do respectivo processo na sessão de composição. Como foi descrito na secção 5.6.1, cada vez que um serviço é seleccionado para fazer parte da composição, o componente *descoberta e selecção de serviços* procura serviços utilizando os dados recolhidos da base de conhecimento contextual através do motor de contexto, e propõe novos serviços passíveis de serem combinados, baseando-se no contexto do utilizador e nas suas políticas. A procura e selecção de serviços apenas é possível devido à linguagem de descrição de serviços OWL-S, que permite a criação de relações com outras ontologias que podem descrever detalhes sobre um tipo de serviço e as suas características. Nas próximas subsecções irá ser descrito o modo de funcionamento de selecção de serviços e posteriormente será descrito o processo de criação do *workflow* do serviços composto.

6.2.1 Selecção de serviços

A selecção de um serviço cliente compatível para ser adicionado à composição é conseguida utilizando a descrição da classe da ontologia *ServiceProfile*, que apresenta o serviço e especifica os seus parâmetros de entrada e saída, pré-condições e efeitos. O processo de selecção de um serviço é dividido em dois passos: a filtragem de serviços compatíveis e o *scoring* considerando o contexto e as preferências do utilizador, como mostra o mecanismo ilustrado na Figura 6.3. A filtragem dos serviços é feita à custa de uma comparação (*matching*), em que são seleccionados os serviços cuja saída possa ser combinada com a entrada do último serviço seleccionado, utilizando para isso as informações referentes aos serviços, extraídas da base de conhecimento de serviços. Posteriormente é executado uma classificação (*scoring*), utilizando também as informações referentes aos serviços, o contexto e as preferências do utilizador extraídas da base de conhecimento contextual. Por fim os serviços propostos são ordenados e retornados à aplicação cliente.

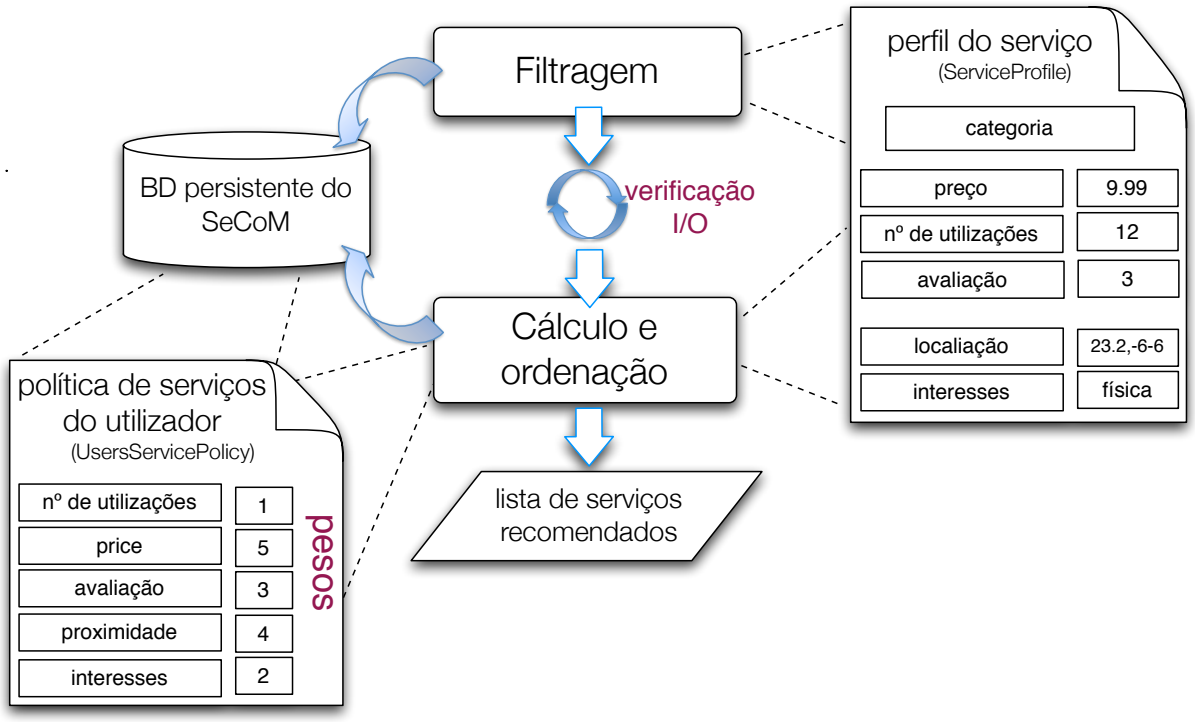


Figura 6.3 – Mecanismo de selecção de serviços.

O processo de filtragem apresentado no processo de selecção recorre a um mecanismo para a combinação de serviços através de verificação semântica. Este processo de filtragem é adaptado a partir de uma solução apresentada em (Paolucci, Kawamura et al., 2002), que permite a selecção de *Web Services* baseada na verificação semântica dos serviços que se pretendem combinar. Este mecanismo é baseado na linguagem OWL-S e num algoritmo de

matching que permite determinar a relação entre os requisitos de um serviço pedido e de outro que foi publicitado. Essa relação pode ser definida em quatro níveis de similaridade:

- *exact*: este grau de similaridade indica que os dois tipos de atributos são exactamente equivalentes;
- *plugin*: se o atributo pedido é um sub-conceito do atributo anunciado, então trata-se de uma similaridade do tipo *plugin*;
- *subsume*: se o atributo pedido é um super-conceito do atributo, então verifica-se uma similaridade do tipo *subsume*;
- *fail*: não existe nenhuma das relações anteriores e por isso não existe uma relação relevante entre os atributos fornecidos.

Para criar a lista de serviços compatíveis são executados dois processos: o primeiro é uma consulta à base de conhecimento de serviços que através de *queries* SPARQL extrai informações sobre quais as entradas, saídas, pré-condições e efeitos do *ServiceProfile* do serviço. O segundo processo é a utilização do motor de inferência para determinar as relações entre o serviço seleccionado e os serviços existentes na base de conhecimento, segundo a aproximação descrita anteriormente.

O motor de composição tem acesso à base de conhecimento geral através de uma instância da classe *OWLKnowledgeBase* que contém todas as descrições de serviços que são visíveis a todos utilizadores. O *OWLKnowledgeBase* é um dos principais componentes da API OWL-S, que, além de guardar as descrições dos serviços em OWL-S, permite a sua leitura e escrita em ficheiros no formato RDF e possui um motor de inferência (*Pellet*), que é capaz de descobrir relações entre classes, propriedades e indivíduos através dos axiomas para regras de inferência OWL.

Assim, através de um algoritmo (Sirin, Parsia et al., 2004) que utiliza o motor de inferência *Pellet*, são seleccionados todos os serviços que contenham uma entrada compatível com o serviço escolhido. Consideram-se como serviços compatíveis todos os serviços que no algoritmo de *matching* tenham uma relação do tipo *exact* ou *plugin*. O Código 6.1 mostra o algoritmo para a descoberta de serviços compatíveis que recebe como parâmetros de entrada o serviço que faz parte da composição, a sua entrada e a lista de serviços candidatos.

```
1: public List getMatches(Service service, Input input, List services){
2: ...
3:  Iterator i = candidates.iterator();
```

```
4:  while (i.hasNext()) {
5:    ValueMap candidate = (ValueMap) i.next();
6:    Process process = (Process) ((OWLIndividual)
7:          candidate.getValue("process")).castTo(Process.class);
8:    Output output = (Output) ((OWLIndividual)
9:          candidate.getValue("param")).castTo(Output.class);
10:   OWLType outputType = output.getParamType();
11:
12:   int matchType = getMatchType(outputType, inputType);
13:
14:   if ((matchType == Match.EXACT) || (matchType == Match.SUBSUME)) {
15:     Service outputService = process.getService();
16:     if (outputService != null)
17:       matches.add(new Match(matchType, output, input));
18:   }
19: }
20: return matches;
21:}
```

Código 6.1 – Código para a filtragem de serviços compatíveis.

As relações do tipo *subsume* e *fail* não são usadas pelo facto de não serem capazes de produzir composições fiáveis. Para se perceber melhor o funcionamento do algoritmo considere-se o cenário de alguém que está compor um serviço cujas entradas do último serviço da composição são definidas pela ontologia *Location* ilustrada na Figura 6.4 e pretende-se seleccionar um serviço compatível para a composição:

- tem-se uma similaridade *exact*, se a classe do atributo de entrada do último serviço seleccionado pelo utilizador (*inS*) for a mesma que a classe do atributo de saída do serviço analisado (*outA*). Por exemplo *inS* tipo *VirtualLocation* e *outA* tipo *VirtualLocation* e neste caso o serviço analisado é considerado compatível;

- uma similaridade to tipo *plugin* acontece se a classe do atributivo *inA* é uma subclasse do atributo *inS*. Por exemplo *outA* do tipo *OutdoorLocation* e *inS* do tipo *GeographicLocation*. Considera-se que a localização externa é uma especialização de uma localização geográfica e também neste caso o serviço analisado é considerado compatível;

- é identificada uma similaridade do tipo *subsumes* quando o atributo *inS* é uma subclasse do atributo *outA*. Por exemplo, *inS* do tipo *IndoorLocation* e *outA* do tipo *GeographicLocation*. A classe *IndoorLocation* define um conceito mais específico que adiciona características à classe *GeographicLocation*, portanto o serviço analisado não satisfaz o pedido;

- a similaridade do tipo *fail* acontece quando os atributos *inS* e *outA* não têm nenhuma das relações anteriores. Por exemplo, *inS* do tipo *PhysicalLocation* e

outA do tipo *VirtualLocation*; também neste caso o serviço não é compatível.

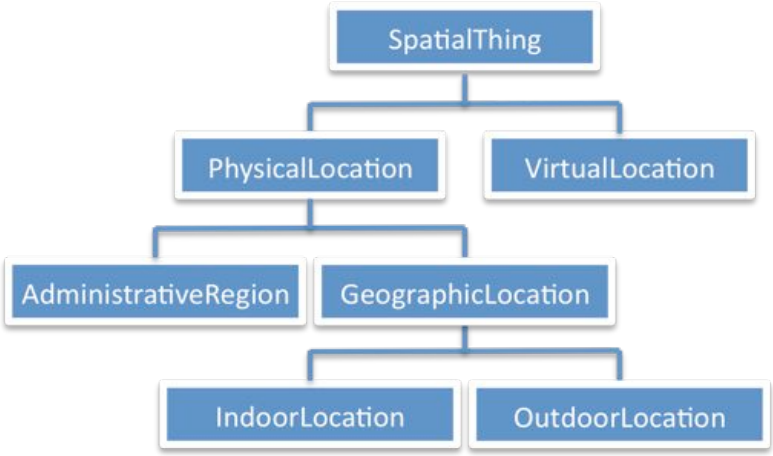


Figura 6.4 – Parte da relação hierárquica da ontologia *Location*

O processo de *scoring* utiliza os parâmetros para avaliação dos serviços definidos no perfil do utilizador através da ontologia de apoio *iCasProfileHierarchy* e o coeficiente de ponderação para avaliação de serviços definidos nas preferências do utilizador através da ontologia *UsersServicePolicy*. Como é possível observar na Figura 6.3, a ontologia *UsersServicePolicy* contém vários campos para atribuir um coeficiente de ponderação tais como: frequência de utilização do serviço, preço, avaliação que utilizadores fizeram do serviço, proximidade, interesses, entre outros. A parte da ontologia *iCasProfileHierarchy* que estende as propriedades do perfil do serviço, está dividida em duas partes. A primeira, comum a todos os serviços para que possa ter um perfil de cariz participativo (Sousa, Fonseca et al., 2009), inclui os parâmetros preço, número de vezes de utilização do serviço e avaliação que os utilizadores fizeram do serviço. A segunda parte poderá ter diversos parâmetros descritivos do contexto (ex. localização, data) ou até de aspecto funcional (ex. latência). O Código 6.2, mostra parte da descrição de um serviço de reserva de livros. As primeiras linhas descrevem a avaliação do serviço e outras características como o preço e a localização.

```
1:  ...
2:  <icash:LibraryService rdf:ID="reserveBookProfile">
3:   <icash:usageFrequency rdf:datatype="&xsd;float">
4:      2.0 </icash:usageFrequency>
5:   <icash:userEvaluation rdf:datatype="&xsd;float">
6:      3.0  </icash:userEvaluation>
7:   <icash:price rdf:datatype="&xsd;float">
8:      1.2  </icash:price>
9:   <profile:serviceName rdf:datatype="&xsd;string">
10:     reserveBook   </profile:serviceName>
11:  <profile:textDescription rdf:datatype="&xsd;string">
12:     Service to reserve a Book in the Campus Library
13:  </profile:textDescription>
14:  <icash:location rdf:resource="#Latitude_1"/>
```

```
15: <icash:location rdf:resource="#Longitude_2"/>
16: <profile:hasInput rdf:resource="#isbn"/>
17: <profile:hasOutput rdf:resource="#result"/>
18: <service:presentedBy rdf:resource="#reserveBookService"/>
19: </icash:LibraryService>
20:...
```

Código 6.2 – Exemplo da descrição do perfil de um serviço.

O mecanismo de *scoring* é implementado pelo método *performScore* da classe *ServiceScoring*. Este método recebe o nome do utilizador e a lista dos serviços que podem ser combinados com o serviço anteriormente seleccionado e calcula o *score* para cada um deles, como mostra o Código 6.3. Por fim, os serviços são ordenados de forma descendente e retornados à aplicação cliente.

```
1: ...
2:   while (u.hasNext()) {
3:     Map.Entry muser = (Map.Entry) u.next();
4:     if (msev.getKey().equals(muser.getKey())) {
5:       String param = (String) msev.getKey();
6:
7:       if (param.equals(iCas.location)) {
8:         score += uvalue * distance(getUserLocation(user, iCas.lat),
9:           getUserLocation(user, iCas.lon), serviceLocation(serv,
10:            iCas.lat), getUserLocation(serv, iCas.log), 'K');
11:      } else {
12:        double svalue = Double.parseDouble(msev.getValue().toString());
13:        int uvalue = Integer.parseInt(muser.getValue().toString());
14:        score += uvalue * svalue;
15:      }
16:     break;
17:    }
18:  }
19:...
```

Código 6.3 – Parte do método que calcula o *score* dos serviços.

Quando é iniciada a interface de composição na aplicação cliente, esta deve apresentar uma lista inicial de serviços. Essa lista pode ser devolvida pelo métodos *getServices* ou *searchServices* disponibilizados pelo *Web Service CompositionEngineCore*, como referido no início desta secção. O método *getServices* devolve os serviços privados do utilizador juntamente com os serviços disponíveis a todos os utilizadores que existam na plataforma. O método *searchServices* permite fazer uma filtragem de serviços por categoria. Desta forma, o cliente deverá invocar um outro método, *getServicesCategory*, que devolve as categorias de serviços definidas na ontologia *iCasProfileHierarchy* serviços (ex: *Informative Services*, *eLearning Services*, *Administrative Services*). Posteriormente, ao invocar o método *searchServices*, deverá usar como parâmetro uma das categorias, sendo devolvidos todos os serviços que pertençam a essa categoria ou subcategorias. Tal como o mecanismo de

matching de serviços explicado anteriormente, também este recorre ao raciocínio de classes para filtrar os serviços por categoria. Como tal, a implementação é semelhante à apresentada no Código 6.1.

6.2.2 Composição de serviços

Quando um utilizador inicia uma composição é criada uma sessão de composição na *iCas*, por forma a manter todo o processo de composição do lado do servidor. Essa sessão associa ao nome do utilizador um conjunto de variáveis que permitem a individualização da composição. O processo de composição consiste num ciclo em que se adiciona o serviço seleccionado pelo utilizador ao último estado da composição guardado numa *hashtable*. Esta adição é conseguida através da criação de um novo serviço composto. Posteriormente a *hashtable* é actualizada com a última versão do serviço composto, estando pronta para lhe ser adicionado um novo serviço seleccionado pelo utilizador. Em qualquer momento a composição pode ser suspensa, estando o serviço pronto para ser executado, guardado ou utilizado noutra composição. A adição de novos serviços ao processo de composição é feita através do método *addToSequenceService.* Enquanto o utilizador não sair do sistema, todos os seus serviços são mantidos em memória numa *base de conhecimento* (*OWLKnowledgeBase*) privada. Esta base de conhecimento é criada quando o utilizador inicia o seu cliente e todos os serviços privados são carregados nesta base de conhecimento.

Para a composição de um serviço é criado um novo serviço através da classe *Service*, ao qual são adicionadas uma instância da classe *Profile*, uma instância da classe *Grounding* e uma instância da classe *CompositeProcess*. Esta última corresponde à classe *ProcessModel*; no entanto por se tratar de uma composição o modelo de processo é definido por uma instância da classe *CompositeProcess*. De forma análoga acontece para um serviço atómico que corresponde a uma instância da classe *AtomicProcess*. Este *CompositeProcess* contém os construtores de *workflow* (*sequence*, *split*, etc) e informação sobre o fluxo de dados que liga as saídas e entradas dos diversos serviços. Já as instâncias das classes *Profile* e *Grounding* são iguais para qualquer tipo de serviço, seja ele atómico, composto ou simples. Na instância da classe *Profile* são especificadas as informações para a descoberta e comparação do serviço, como as entradas e as saídas do serviço, o seu nome e uma breve descrição, podendo ser acrescentadas outras informações. A instância da classe *Grounding* contém os detalhes de como é possível aceder ao serviço, especificando os protocolos de comunicação, as portas e

outras informações. O Código 6.4 mostra parte do método *addToSequenceService* que permite a criação de um processo composto, do seu perfil e do seu *grounding*, o mapeamento dos *groundings* dos diversos serviços no serviço composto e por fim o mapeamento do perfil *grounding* no novo serviço.

```
1:  ...
2:    CompositeProcess process =
3:      ont.createCompositeProcess(URIUtils.createURI(baseURI, "Process"));
4:    Profile profile = ont.createProfile(URIUtils.createURI(baseURI,
5:     "Profile"));
6:    Grounding grounding = ont.createGrounding(URIUtils.createURI(baseURI,
7:     "Grounding"));
8:
9:    createSequenceProcess(process, services);
10:
11:   ProcessList list = process.getComposedOf().getAllProcesses();
12:    for (int i = 0; i < list.size(); i++) {
13:     Process pc = list.processAt(i);
14:      if (pc instanceof AtomicProcess)
15:       grounding.addGrounding(((AtomicProcess) pc).getGrounding());
16:   }
17: ...
18:     service.setProfile(profile);
19:     service.setProcess(process);
20:     service.setGrounding(grounding);
21: ...
```

Código 6.4 – Parte do serviço de composição de serviços.

Quando um utilizador compõe um serviço este fica associado à sua *OWLKnowledgeBase* privada, não sendo visto pelos restantes utilizadores, independentemente do novo serviço utilizar serviços "públicos" ou "privados" na sua composição.

6.2.3 Execução de serviços

A execução de serviços é implementada também na classe *CompositionEngineCore* pelo método *runService*. No Código 6.5 é apresentada parte do método que executa o último serviço composto pelo utilizador, porque se assume que após a composição o serviço será executado. Na execução do serviço composto a instância da classe *ProcessExecutionEngine* vai invocar cada um dos serviços individuais e passar os dados entre as saídas e entradas dos serviços conforme o fluxo que foi definido pelo utilizador e que se encontra na instância da classe *Process* do serviço. As linhas de 10 a 15 fazem a atribuição dos valores de entrada introduzidos pelo utilizador a cada uma das entradas do serviço a ser executado. A linha 20 corresponde à criação de um motor de execução de processo e a linha 21 à execução do processo do serviço e retorno dos resultados. Estes resultados serão também retornados a

quem invoca o *Web Service* que ordenou a execução do serviço composto. Se o utilizador pretende executar um serviço (atómico ou composto) seleccionado da lista de serviços que lhe é proposta, existe um método semelhante a este diferindo nos parâmetros de entrada onde deve ser indicado o nome do serviço a ser executado.

```
1:  public List runService(String user, List inputValues){
2:  ...
3:    s = (Service) userServices.get(user);
4:    Process process = s.getProcess();
5:    // nome dos parâmetros de entrada
6:    Iterator itinputs = process.getInputs().iterator();
7:    // valores dos parâmetros de entrada
8:    Iterator iTinputValues = inputValues.iterator();
9:  ...
10:   while (iTinputValues.hasNext()) {
11:     Input i = (Input) iTinputValues.next();
12:     String str = i.getLabel();
13:     values.setValue(process.getInputs().inputAt(0),
14:        EntityFactory.createDataValue(iTinputValues.next()));
15:   }
16:
17:   if (iCas.DEBUG) {
18:            System.out.print("Executing...");
19:   }
20:   ProcessExecutionEngine exec = OWLSFactory.createExecutionEngine();
21:   ValueMap results = exec.execute(process, values);
22: ...
```

Código 6.5 – Código para a execução de serviços.

6.2.4 Armazenamento de serviços

O armazenamento de serviços permite que um serviço composto por um utilizador possa ser guardado para utilização futura. Esta funcionalidade é fornecida pelo método *saveService* e tem como parâmetros o nome do utilizador e o nome do serviço, já que este se encontra na *HashTable* que mantém a sessão da composição. Deste modo, se o utilizador compuser um novo serviço o anterior será perdido. Existe ainda um método do *Web Service* chamado *resetComposition* que prepara uma nova sessão de composição. Foram ainda implementados outros métodos de armazenamento, como *saveAllServices*, que guarda todos os serviços comuns numa directoria.

Quando o utilizador grava um serviço ele é armazenado numa directoria própria para serviços e ao ficheiro de serviços do utilizador é acrescentado um novo registo com o nome desse serviço. A gravação dos serviços do utilizador é feita num ficheiro de texto e as descrições dos serviços são guardadas no formato de triplas RDF, proporcionadas pela classe *OWLOntology*.

6.2.5 Partilha de serviços

O processo de partilha de um serviço, implementado pelo método *shareService*, consiste em mover a descrição desse serviço da *OWLKnowledgeBase* privada do utilizador para a *OWLKnowledgeBase* principal. Trata-se de um mecanismo de implementação simples, uma solução de partilha mais evoluída passaria por implementar um mecanismo que fizesse uso da ontologia *Policies*, para permitir um conjunto de funções mais avançadas como: quem pode utilizar o serviço, condições de partilha (ex. só execução, execução/composição, etc), remover partilha, entre outras.

6.3 Motor de contexto

A Figura 6.5 mostra o diagrama de classes do *motor de contexto* que gere o modelo contextual semântico. O *motor de contexto* está dividido em quatro componentes: (1) *ContextEngine,* que irá implementar as funcionalidades de consulta de contexto, inferência e carregamento do modelo persistente; (2) *ContextEngine.ManageData,* que fornece as funcionalidades de adicionar, actualizar e remover dados contextuais de cada uma das ontologias aos outros motores e permite gerir as informações de perfil do utilizador através de um *Web Service*; (3) *ContextEngine.History,* que captura os eventos lançados quando existe uma acção no modelo contextual e regista essas acções numa base de dados; (4) *ContextEngine.Aggregator,* que agrega dados de diversas fontes e relaciona-os com uma entidade e actualiza a base de conhecimento contextual. Este componente pode apenas manter os dados em memória, não os guardando na base de conhecimentos contextual. Esta opção é disponibilizada pelo facto de haver situações em que não existe interesse em armazenar esses dados ou para situações em que os dados são muito dinâmicos e que podem causar um grande carga de entrada/saída de dados na base de dados que mantém a base de conhecimentos contextual de forma persistente.

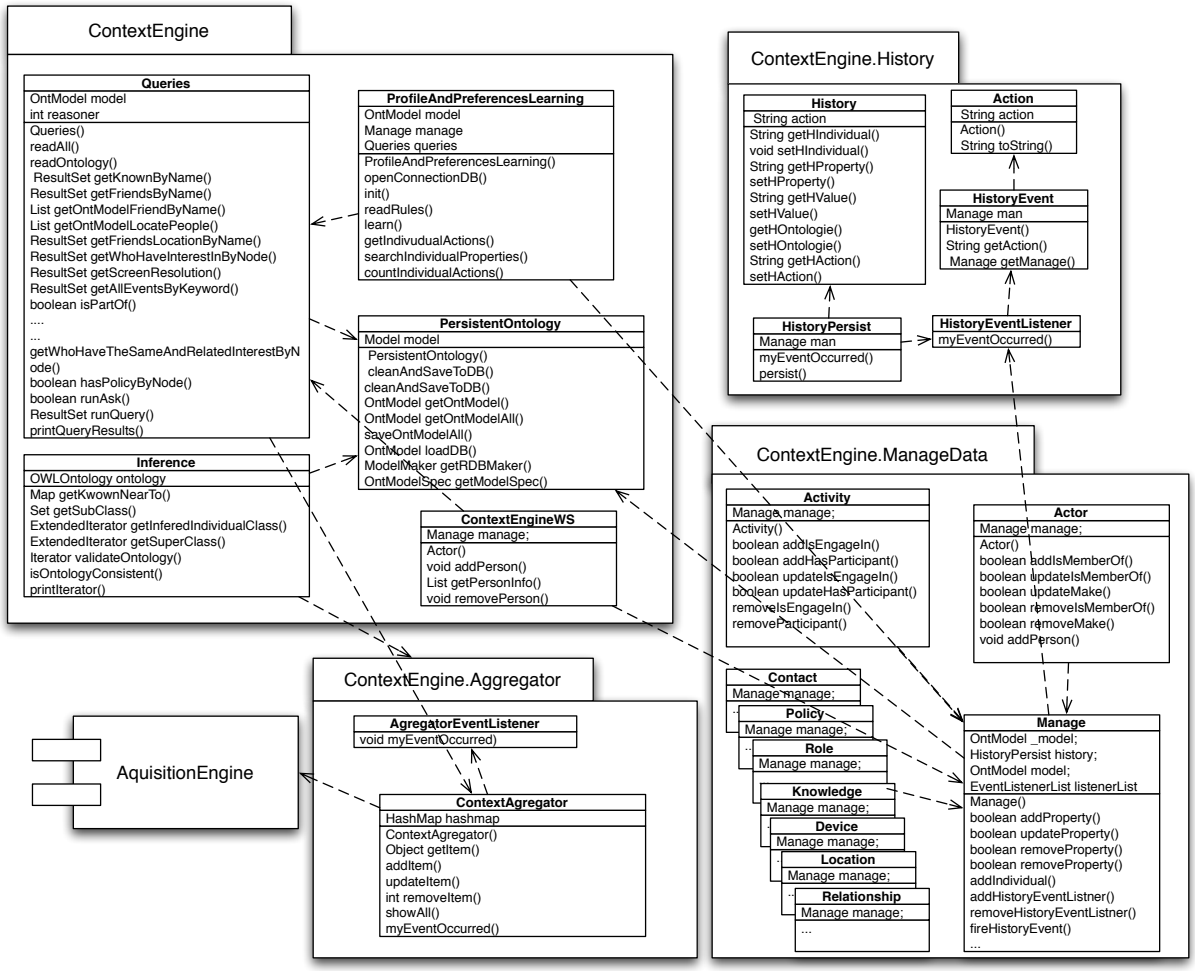


Figura 6.5 - Classes que compõem o *motor de contexto*.

O armazenamento de contexto de forma persistente é feito numa base de dados relacional, isto é, todos os dados contextuais são guardados no formato RDF. Esta abordagem evita a necessidade de carregar todo o modelo RDF cada vez que se pretende aceder à informação que está no modelo, situação que seria problemática para modelos de grandes dimensões (Jena, 2009). Outra das vantagens está relacionada com as operações de escrita, que sendo frequentes, é possível explorar as transacções da base de dados, resolvendo problemas de concorrência e consistência, que existiriam se o modelo ontológico fosse armazenado num ficheiro. Permite ainda lidar com modelos contextuais maiores que a memória principal do computador. Como já foi referido no início do parágrafo todas as informações de contexto são guardadas no formato de triplas RDF em base de dados relacional, enquanto que as ontologias que compõem o modelo ontológico são mantidas em ficheiros OWL.

A classe que implementa as funções de armazenamento de contexto de forma persistente é a classe *PersistentOntology*. Ela é responsável por criar o modelo contextual suportado pela bases de dados. Inicialmente é necessário invocar o driver do motor base de dados que neste

caso é o *PostgreSQL* (Group, 2004) e criar uma instância da classe *DBConnection*. Para isso o construtor da *DBConnection* da Jena API tem que ter o nome do utilizador e a palavra-passe da base de dados, o URL da bases de dados e o tipo de dados, que para o *PostgresSQL* é "*PostgreSQL*". Em seguida é criado o modelo com suporte a bases de dados para a ontologia (ex. *Actor*), que é depois conectado a uma ontologia OWL, a qual utiliza novamente a instância *ModelMaker* para criar o armazenamento para os modelos importados. Por fim, é carregada a ontologia fonte e são importadas todas as referências a outras fontes que esta contenha.

```
1: String DB_URL = "jdbc:postgresql://127.0.0.1:5432/icasdb";
2: String DB_USER = "postgres";
3: String DB_PASSWD = "postgres";
4: String DB = "PostgreSQL";
5: String DB_DRIVER = "org.postgresql.Driver";
6: Class.forName(iCas.DB_DRIVER);
7: IDBConnection conn = new DBConnection(DB_URL, DB_USER, DB_PASSWD, DB);
8:
9: ModelMaker maker = ModelFactory.createModelRDBMaker(conn);
10: Model base = maker.createModel("http://icas.ipb.pt/secom/actor",false);
11: OntModel ontModel=
ModelFactory.createOntologyModel(getModelSpec(maker),base);
12: ontModel.read(http://icas.ipb.pt/secom/actor);
```

Código 6.6 – Mapeamento da ontologia persistente à base de dados do modelo.

A gestão de perfil e preferências é composta pela classes *Actor, Activity, Contact, Device, Knowledge, Location, Policy, Relactionship, Role* e a classe auxiliar *Manage*. Cada uma das classes implementa funções para gerir os dados da ontologia correspondente. O Código 6.7 ilustra duas funções da classe Actor, a primeira *updateIsMemberOf* permite actualizar o grupo a que o utilizador pertence, e a segunda adiciona um indivíduo do tipo *Person*, o seu nome e apelido.

```
1: public boolean updateIsMemberOf(OntModel model,String user,String group)
2: {
3:   return manage.updateProperty(model, iCas.actorData + user,
4:   iCas.actorData + group, iCas.isMemberOf);
5: }
6: public void addPerson(OntModel model, String name, String surName) {
7:  Random r = new Random();
8:  String token = Long.toString(Math.abs(r.nextLong()), 36);
9:  OntClass Person = model.getOntClass(iCas.actor + iCas.person);
10: Individual ind = model.createIndividual(iCas.actorData + token,
Person);
11: DatatypeProperty namedp = model.getDatatypeProperty(iCas.actor +
"hasName");
12: ind.addProperty(namedp, name);
13: ind.addProperty(model.getDatatypeProperty(iCas.actor + "hasSurname"),
surName);
14: }
```

Código 6.7 – Actualização e adição de dados no modelo contextual.

A classe *Manage* fornece métodos genéricos às restantes classes, para a adicionar, actualizar e remover propriedades de objectos, como foi explicado na secção 3.2.4.1. Estes métodos são depois usados por cada uma das classes correspondestes às diferentes ontologias, para implementar métodos particulares, de modo a gerir as propriedades dos indivíduos de cada uma das ontologias. O Código 6.8 mostra o método para actualizar uma propriedade dado o modelo de dados, a propriedade e o novo valor a atribuir a essa propriedade.

```
1: public boolean updateProperty(OntModel model, String individual, String
updateValue, String property) {
2: // return true if the property exists or false if dosen't
3:
4: setVars(model, individual, updateValue, property, Action.UPDATE);
5:
6:  Individual ind = model.getIndividual(individual);
7:  OntProperty ontProp = model.getOntProperty(property);
8:  Individual updateInd = model.getIndividual(updateValue);
9:  if (!ind.hasProperty(ontProp, ind.getPropertyValue(ontProp))) {
10:     ind.addProperty(ontProp, updateInd);
11:     return false;
12: }
13: ind.removeProperty(ontProp, ind.getPropertyValue(ontProp));
14: ind.addProperty(ontProp, updateInd);
15: return true;
16:}
```

Código 6.8 – Método genérico para actualização das propriedades dos indivíduos no modelo contextual.

O *Web Service* que fornece a funcionalidade de edição do perfil do utilizador é implementado pela classe *Web Service ContextEngineWS* que fornece três métodos para que o utilizador visualize, actualize ou remova o seu perfil. Internamente este *Web Service* implementa basicamente funções de *proxy*, recorrendo às funcionalidades da classe *Queries* (explicada mais à frente) para extrair a informação do utilizador e das classes *Actor, Activity, Contact, Device, Knowledge, Location, Policy, Relactionship, Role* e *Manage*, para actualizar o modelo de dados contextual.

A classe *Manage*, juntamente com as classes *HistoryEventListener*, *HistoryEvent* e *HistoryPersist*, implementam o mecanismo de notificação de eventos, de modo a que quando é invocado um dos métodos de adição, actualização ou remoção de dados seja lançado um evento que será tratado por *HistoryPersist*. Este último irá identificar a operação e armazenar alguns dos dados desta numa base de dados, criando assim o histórico de todas as operações realizadas no modelo contextual. Os registos do histórico de acções são armazenados no seguinte formato: *acção* + *tripla alvo* (ex. *update: Pedro isMemberOf Núcleo de alunos de Física*). Estes registos são guardados na base de dados através do método de persistência de objectos em bases de dados relacionais utilizando a *framework Hibernate* (Hat, 2009). O

Código 6.9 mostra o método *myEventOccurred* da classe *HistoryPersist*, que captura os eventos, actualiza os vários atributos do objecto *History* e em seguida invoca o método *persist* que armazena os dados na bases de dados.

```
1: public void myEventOccurred(HistoryEvent evt) {
2:
3:      if(iCas.HISTORY) {
4:         Manage man = (Manage) evt.getManage();
5:
6:         History history = new History();
7:
8:         history.setHAction(man.getAction());
9:         history.setHOntologie(man.getOntologie());
10:        history.setHIndividual(man.getIndivudual());
11:        history.setHProperty(man.getProperty());
12:        history.setHValue(man.getValue());
13:
14:        persist(history);
15:     }
16:     ...
17: }
18:
19: public void persist(Object object) {
        EntityManagerFactory  emf =
20:                Persistence.createEntityManagerFactory("iCasHistoryPU");
21:     EntityManager em = emf.createEntityManager();
22:     em.getTransaction().begin();
23:
24:     try {
25:       em.persist(object);
26:       em.getTransaction().commit();
27:     } catch (Exception e) {
28:       System.out.println(e.getMessage());
29:       em.getTransaction().rollback();
30:     } finally {
31:       em.close();
32:     }
33:   }
```

Código 6.9 – Classe *HistoryPersist* responsável por armazena o histórico de acções.

A extracção de conhecimento da base de conhecimento é feita de duas formas já explicadas na secção 5.6.2: consultas de contexto e inferência de contexto. A primeira é conseguida através da classe *Queries*, que implementa mais de 35 métodos de consulta de dados ao modelo contextual. Todas as consultas são feitas utilizando a linguagem de consulta SPARQL ao motor de consultas ARQ da Jena API. Na classe *Queries* são implementados dois tipos de *queries*: umas que retornam conjuntos de resultados ou grafos com dados RDF, através de uma consulta utilizando o comando SELECT, outras verificam se uma determinada consulta é verdadeira ou falsa através do comando ASK.

O Código 6.10 representa uma consulta ao modelo contextual utilizando a linguagem *query* SPARQL, questionando o grafo sobre quem tem os mesmos interesses ou interesses

relacionados com uma determinada pessoa. As ontologias que são utilizadas são a ontologia *Actor* e a ontologia *Knowledge*. A primeira parte da *query* (linhas 6 a 7) selecciona os utilizadores com o mesmo interesse através da propriedades *hasInterestIn* da pessoa indicada no parâmetro de entrada da função, enquanto a segunda parte da *query* selecciona pessoas com interesses relacionados através da propriedade *isKnowledgeRelatedTo*. Por fim, os resultados das duas partes são unidos através do comando UNION, resultando num conjunto de dados que retorna indivíduos da classe *Person*. A *query* é construída numa *String* (*queryStr*) cujo início é composto pelos prefixos para os *namespaces*, que neste caso são substituídos pelas constantes *iCas.actorPFX*, *iCas.actordataPFX*. *iCas.knowledgePFX* e *iCas.knowledgedataPFX* definidas na classe *iCas*.

```
1: public ResultSet getWhoHaveTheSameAndRelatedInterest(String person) {
2:  String queryStr = iCas.actorPFX + iCas.actordataPFX + iCas.knowledgePFX
3:           + iCas.knowledgedataPFX +
4:   "SELECT ?person \r\n" +
5:     "WHERE { \r\n" +
6:       "{?person know:hasInterestIn ?interestIn . \r\n" +
7:       "dacto:" + person + " know:hasInterestIn ?interestIn .}\r\n" +
8:     "UNION  { \r\n" +
9:       "?person know:hasInterestIn ?knowledgerelated . \r\n" +
10:      "?interestIn know:isKnowledgeRelatedTo ?knowledgerelated . \r\n" +
11:      "dacto:" + person + " know:hasInterestIn ?interestIn . \r\n" +
12:    "} \r\n" +
13:  "}";
14: return runQuery(queryStr);
15:}
```

Código 6.10 – Exemplo de uma consulta ao grafo do modelo contextual.

Após a construção de uma *query* do tipo SELECT é invocado o método *runQuery* da mesma classe *Queries* descrito no Código 6.11. A *query* é criada a partir de uma *string* através da classe *QueryFactory*, que juntamente com o modelo RDF, são passados ao objecto *QueryExecutionFactory* para criar uma instância de execução da *query*. Os resultados devolvidos pela *query* são do tipo *ResultSet*, que normalmente são manuseados através de um ciclo, como mostram as linhas 28 a 34.

```
1: public ResultSet runQuery(String queryStr) {
2:    if (iCas.DEBUG) {
3:      System.out.println(queryStr);
4:    }
5:
6:    Query query = QueryFactory.create(queryStr);
7:    QueryExecution queryExec;
8:    queryExec = QueryExecutionFactory.create(query, model);
9:      if (iCas.DEBUG) {
10:       System.out.println("Query SPARQL: \n ");
11:     }
12:   }
13:   ResultSet results = queryExec.execSelect();
```

```
14:
15:    if (iCas.DEBUG) {
16:      try {
17:       query.serialize(new IndentedWriter(System.out, true));
18:      } catch (Exception e) {
19:        System.out.println(e);
20:      }
21:    }
22:   queryExec.close();
23:   return results;
24: }
25:
26:
27: ...
28:   ResultSet results = qexec.execSelect() ;
29:     for ( ; results.hasNext() ; )
30:     {
31:       QuerySolution sol = results.nextSolution() ;
32:       RDFNode x = soln.get("person") ;
33:       Resource r = soln.getResource("person") ;
34:  ...
```

Código 6.11 – Método genérico para execução de *queries* SPARQL do tipo SELECT.

O Código 6.12 mostra uma *query* que faz uso do comando ASK para saber se um determinado dispositivo suporta Java. Para tal esta *query* consulta o grafo RDF definido pela ontologia *Device* fazendo usos dos prefixos que mapeiam esse *namespace*.

```
1: public boolean hasDeviceJava(String deviceName) {
2:      String queryStr = iCas.devicePFX
3:      "ASK { \r\n" +
4:        "?hasDeviceComponent devi:isJavaEnabled true . \r\n" +
5:        "?dev devi:hasDeviceComponent ?hasDeviceComponent . \r\n" +
6:        "?dev devi:deviceModel ?deviceModel . \r\n" +
7:        "FILTER regex(?deviceModel, \"" + deviceName + "\",\"i\")\r\n" +
8:     "}";
9:      return runAsk(queryStr);
10:   }
```

Código 6.12 – Consulta ao modelo contextual para saber as capacidades de um dispositivo.

Uma *query* do tipo ASK é tratada por um método diferente da *query* do tipo SELECT. Neste caso a *query* é executada pelo método *runAsk* (Código 6.13). Este código é semelhante ao Código 6.12, excepto que neste caso o objecto *queryExec* executa o método *execAsk()* e o resultado será do tipo *booleano* indicando se o padrão definido na *query* existe no grafo ou não.

```
1: public boolean runAsk(String askStr) {
2:    if (iCas.DEBUG) {
3:       System.out.println("Ask query:  " + askStr);
4:    }
5:    Query query = QueryFactory.create(askStr);
6:    query.serialize(new IndentedWriter(System.out, true));
7:   QueryExecution queryExec = QueryExecutionFactory.create(query, model);
8:
```

```
9:     boolean results = queryExec.execAsk();
10:    System.out.println("ask result = " + ((results) ? "TRUE" : "FALSE"));
11:    return results;
12: }
```

Código 6.13 - Método genérico para execução de *queries* SPARQL do tipo ASK.

A função de inferência faz uso do processo de inferência baseado em ontologias, em que é possível inferir novas informações contextuais a partir das descrições dos conceitos do domínio (TBOX) e dos factos instanciados desses descritores (ABOX). Esta função permite explorar o raciocínio de classe, de propriedade e de indivíduo abordados na secção 3.2.6, a partir de relações de especialização e generalização de classes, propriedades e características das propriedades como a simetria, inversão, funcional, entre outras.

A classe *Inference* é a responsável por implementar o serviço de inferência e faz uso do motor de inferência *Pellet*. O Código 6.14 mostra a invocação do motor de inferência *Pellet* o qual fazendo uso das relações de especialização e generalização entre classes, infere quais os indivíduos que pertencem a uma classe. As linhas 2 a 5 correspondem ao carregamento das ontologias utilizadas para o processo de inferência e os conjuntos de factos instanciados a partir desse conjunto de ontologias, a TBOX e a ABOX respectivamente. As linhas 6 a 8 correspondem à criação do motor de inferência *Pellet*, que irá resultar num grafo RDF que, além das informações da ABOX, irá conter também novas deduções. Por fim, na linha 10, são retornados todos os indivíduos que foram inferidos numa determinada classe.

```
1:  ....
2:  //Carregas as ontologias
3:   OWLOntology onto = manager.loadOntology(URI.create(iCas.spatial));
4:   manager.loadOntology(URI.create(iCas.spatialData));
5:  ...
6:   Reasoner pellet = new Reasoner(manager);
7:   pellet.loadOntology(onto);
8:   pellet.getKB().setDoExplanation(true);
9:
10:  Set<OWLIndividual> inferedIndividuals = pellet.getIndividuals(owlClass,
11:  false);
12: ...
```

Código 6.14 – Utilização do motor de inferência para inferir novos factos.

A Figura 6.6, ilustra a utilização da aplicação *Protégé* juntamente com o *Pellet* para realizar a inferência (optou-se por mostrar os resultados utilizando o *Protégé* por permitir a visualização mais amigável dos resultados). A máquina de inferência é capaz de inferir que um recurso da classe *Room* é indivíduo das classes *IndoorLocation*, *GeographicLocation*, *PhysicalLocation* e *SpatialThing*. Isto acontece porque *Room* é uma subclasse de *IndoorLocation*, que por sua vez é subclasse de *GeographicLocation*, por sua vez subclasse de *PhysicalLocation*, sendo

esta por fim subclasse da classe *SpatialThing*. As circunferências a vermelho assinaladas na Figura 6.6 destacam o número de instâncias declaradas no grafo (à direita) e o número de instâncias deduzidas pelo motor de inferência (à esquerda).

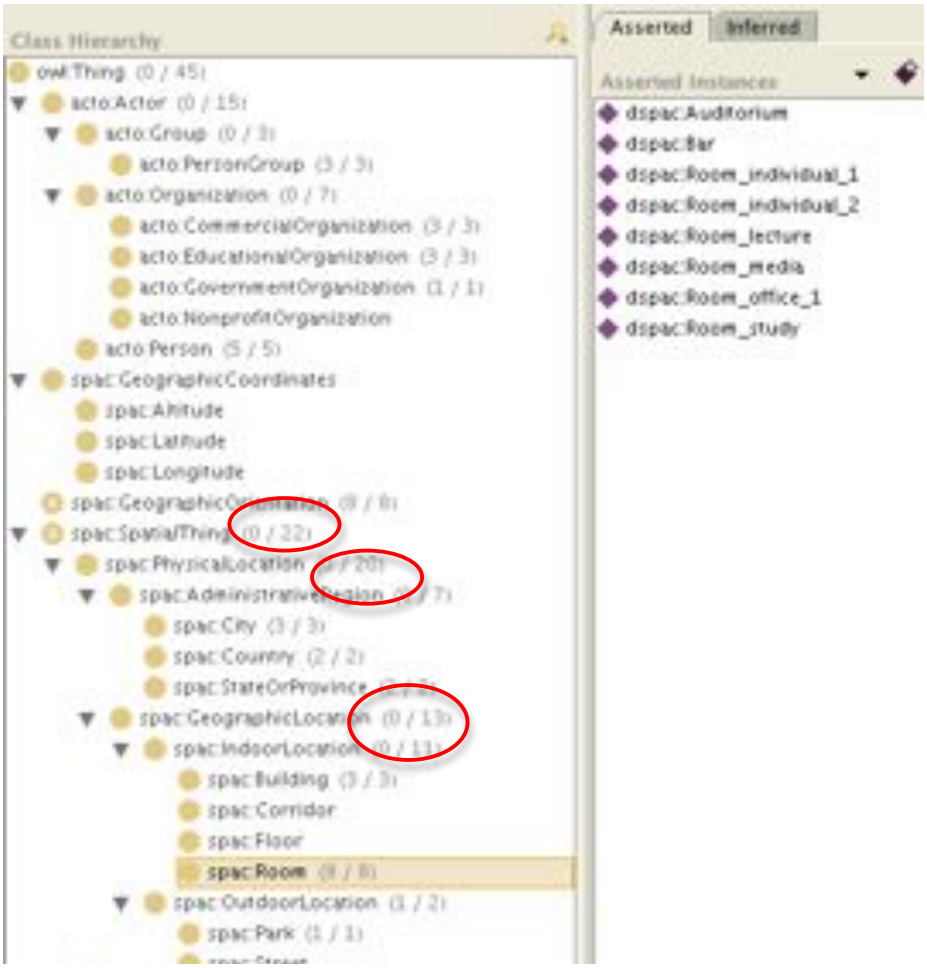


Figura 6.6 – Instanciação dos indivíduos após a inferência utilizando o motor *Pellet*.

O Código 6.15 mostra um trecho de código que através do uso da característica transitiva da propriedade *isPartOf* permite saber por exemplo que se um utilizador está no "auditório" então ele está dentro do "campus da UTAD".

```
1:  ...
2:
3:   Queries query = new Queries();
4:   String location = query.getMyLocationByNode(personNode);
5:    ResultSet results = query.getKnownLocationByNode(personNode);
6:  ....
7:    OWLIndividual otherLoc =
 factory.getOWLIndividual(URI.create(iCas.spatialData + data));
8:    OWLIndividual myLoc =
 factory.getOWLIndividual(URI.create(iCas.spatialData + location));
9:
10:  OWLObjectPropertyAssertionAxiom relation =
 factory.getOWLObjectPropertyAssertionAxiom(otherLoc, isPartOf, myLoc);
11:  if (pellet.hasObjectPropertyRelationship(otherLoc, isConnectedTo,
```

```
myLoc)) {
12:    answer.put(personToAdd, val);
13: ...
```

Código 6.15 – Uso da propriedade transitiva para obter a localização relativa de um utilizador.

Como é possível ver na Figura 6.7, o recurso "auditório" faz parte do edifício da "Reitoria" (contém a propriedade *isPartOf* cujo valor é o recurso "Reitoria") e esta por sua vez também pertence ao "campus da UTAD". Como a propriedade *isParOf* é transitiva, então é possível deduzir que o "auditório" faz parte do "campus da UTAD", logo o utilizador está dentro do campus da UTAD.

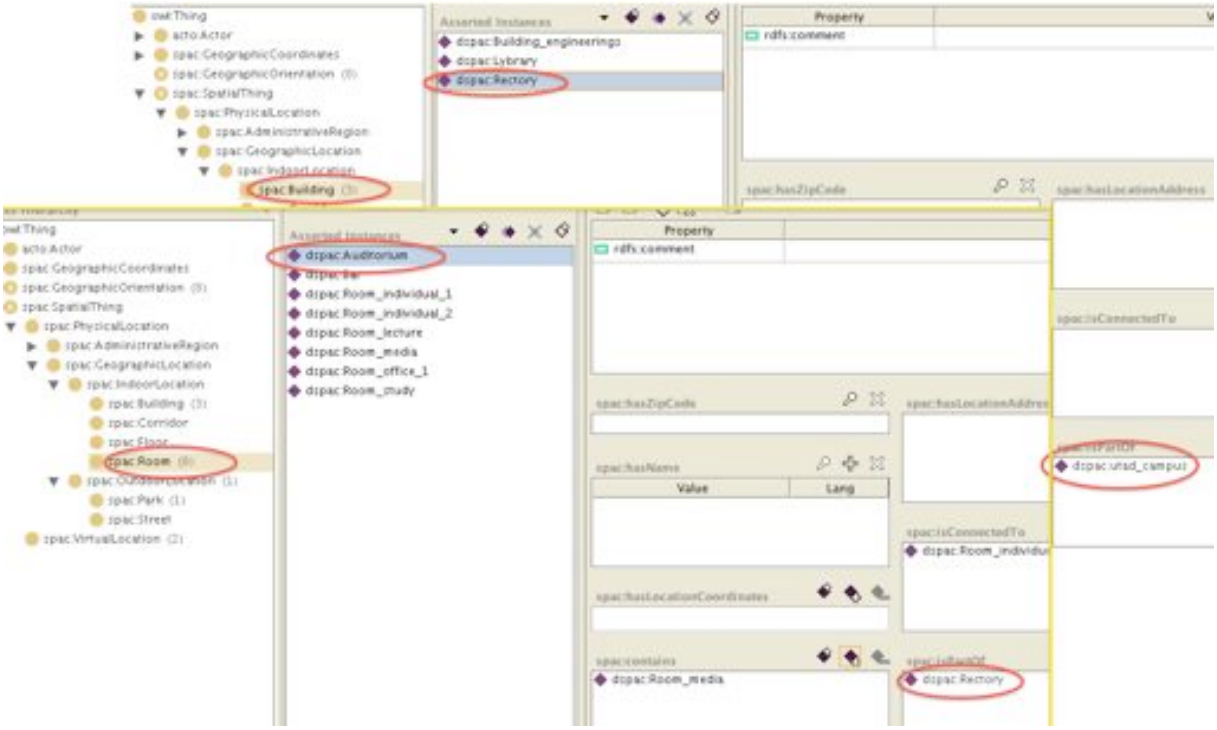


Figura 6.7 – Visualização de recursos das ontologias *Location*.

Para relacionar dados de várias fontes contextuais com uma única entidade foi implementada a classe *ContextAggregator* que é uma subclasse da classe *HashMap*. Através dos métodos *getItem*, *addItem*, *updateItem*, *removeItem*, entre outros, são implementadas operações numa *HashMap* capaz de gerir uma lista ligada de pares (atributo, valor), como é ilustrado no Código 6.16.

```
1: public class ContextAgregator extends HashMap implements
2:                                    AgregatorEventListener {
3:
4:      HashMap hashmap;
5:      OntModel ontModel;
6:      HashMap<Integer, Integer> counter;
7:
8:      public ContextAgregator(OntModel model) {
```

```
9:          super();
10:         hashmap = (HashMap) this;
11:         ontModel = model;
12:     }
13: ...
14:    public void updateItem(Object firtKey, Pair pp) {
15:
16:         LinkedList ll = (LinkedList) this.get(firtKey);
17:         if (ll == null) {
18:             addItem(firtKey, pp);
19:         } else {
20:             ListIterator i = ll.listIterator();
21:             while (i.hasNext()) {
22:                 Pair p = (Pair) i.next();
23:                 String secondKey = (String) pp.getKey();
24:                 if (p.getKey().toString().equalsIgnoreCase(secondKey)) {
25:                     i.remove();
26:                     i.add(pp);
27:                     this.put(firtKey, ll);
28:                     return;
29:                 }
30:             }
31:             addItem(firtKey, pp);
32:         }
33:     }
```

Código 6.16 – Classe *ContextAgregator* e método de actualização dos dados do agregador de contexto.

Deste modo, mantém-se em memória uma lista de pares (atributo, valor) para armazenar os dados contextuais associados a um chave que será a entidade relacionada (ex. utilizador ou objecto). Quando o *AquisitionEngine* captura dados de uma qualquer fonte (ex. serviço, sensor), ele lança um evento que é capturado pelo *ContextAgregator*, que implementa a interface *AgregatorEventListener*. Após o evento ser capturado é analisado de que fonte é que provêm os dados, sendo executado um código específico para cada tipo de sensor. Por exemplo, o Código 6.17 mostra as instruções que fazem a análise do identificador do sensor, após a captação do evento lançado pelo *AquisitionEngine*. Se os dados forem do serviço/sensor *iCas.GPSSensor* então é actualizada a entrada da tabela de *hash* da própria classe cuja chave é o *username* do utilizador e cujo par é (localização, coordenadas de GPS). Caso os dados sejam do serviços/sensor *iCas.ActivitySensor*, tal como acontece no caso anterior, também é adicionada/actualizada a tabela de *hash* com os novos dados, mas neste caso também é actualizada a base de conhecimento contextual.

```
1: ...
2: if (evt.getManage().getSensorData().sensorID == iCas.GPSSensor) {
3:
4:   this.updateItem(s.user, new Pair(iCas.locatedAt, s.geoCoordinates));
5:   if (counter.get(iCas.GPSSensor) == iCas.CounterSnapshot) {
6:     Location loc = new Location(ontModel);
7:     loc.updateLocatedAt(s.user, s.geoCoordinates);
8:   } else {
9:     int count = counter.get(iCas.GPSSensor) + 1;
```

```
10:     counter.put(iCas.GPSSensor, count);
11:   }
12: }
13: if(evt.getManage().getSensorData().sensorID ==iCas.RelactiveLocaSensor){
14:   Location loc = new Location(ontModel);
15:   loc.updateLocatedAt(s.user, s.relativePosition);
16: }
17: if (evt.getManage().getSensorData().sensorID == iCas.ActivitySensor) {
18:   Activity act = new Activity(ontModel);
19:   act.addIsEngageIn(s.user, s.activity);
20: }
21: ...
```

Código 6.17 – Tratamento de eventos pelo agregador de contexto.

É possível observar que no primeiro caso optou-se por não actualizar a base de conhecimento contextual por cada actualização dessa variável, por se considerar uma variável de elevado dinamismo, sendo feito o *snapshot* do valor segundo um contador configurável, por forma a não subcarregar a base de dados que mantém a base de conhecimento e o histórico de acções.

O componente *aprendizagem de perfis e preferências* é implementado recorrendo a um algoritmo que lê regras de um ficheiro e procura na base de dados *histórico de acções* acções realizadas por um recurso, e se para uma determinado tipo de acção o número de ocorrências for superior a um valor *threshold* então a propriedade de um determinado recurso é alterado. Por exemplo, a seguinte regra (ADD, 5, *actor*, *isEngagedIn* -> *actor*, *hasInterestIn*) indica que se na base de dados histórico de acções houver mais de cinco ocorrências sobre uma actividade em que utilizador esteja inscrito então o perfil do utilizador será actualizado indicando que o mesmo utilizador tem interesse nessa actividade. Este algoritmo é implementado pela classe *ProfileAndPreferencesLearning*, que em intervalos de tempo regulares lê as regras do ficheiro de configuração de regras através do método *readRules()*. Para cada uma das regras o método *learn()* vai procurar ocorrências de acções e propriedades especificadas na regra e caso o número de ocorrências seja superior ao valor *threshold* altera o par (indivíduo, propriedade) na base de conhecimento persistente.

```
1:  ...
2:   for (Iterator iter = allUsers.getResultVars().iterator();
3:                                        iter.hasNext();) {
4:      String var = (String) iter.next();
5:      Object val = sol.get(var);
6:      individual = sol.getResource(var).getLocalName();
7:
8:      Thread.sleep(5000);
9:
10:     java.sql.ResultSet rs = searchIndividualProperties(individual);
11: ...
12:     occur =  countIndividualActions(individual,action);
13:     if (occur > threshold) {
14:       manage.updateProperty(model, individual, property, newValue);
```

```
15:     }
16: ...
```

Código 6.18 – Parte do algoritmo de aprendizagem de perfis e preferências.

6.4 Motor de aquisição

O motor de aquisição é composto pelas seguintes classes: *AquisitionEngineCore*, *AquisitionEvent*, e pela interface *Sensor*, ilustradas na Figura 6.8.

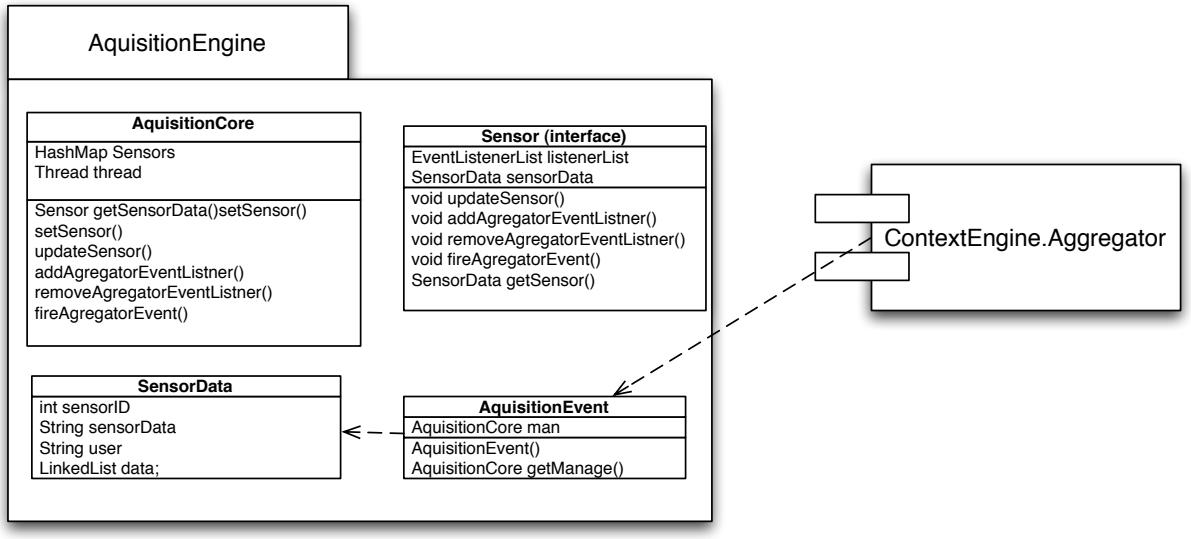


Figura 6.8 – Classes e interface do motor de aquisição de dados contextuais.

De modo a poder adicionar dinamicamente novos sensores/serviços à plataforma de forma dinâmica, a classe *AquisitionEngineCore* recorre a uma funcionalidade do Java que permite o carregamento de classes em *runtime*. Deste modo, para adicionar um sensor à plataforma, este deverá implementar os métodos definidos pela interface *Sensor*. No seu construtor deverá ser implementado o código que irá carregar os dados, ler os dados, do sensor/serviço e disparar o evento para o agregador de contexto, sempre que tiver novos dados para serem passados para a plataforma. O Código 6.19 mostra a utilização da classe *Loader* para invocar uma instância da classe *Sensor2*, carregando-a para a memória e executando o código que está no construtor da classe *Sensor2*. Para adicionar um sensor à arquitectura, a classe que implementa a captação desses dados deverá ser colocada numa directoria específica onde a classe *AquisitionEngineCore* estará periodicamente a verificar se existem novas classes. Quando for detectada uma nova classe ela instancia-a e carrega-a em memória.

```
1: ...
2:    try {
```

```
3:     java.io.File dir = new java.io.File(loadDirectory);
4:       java.net.URL url = dir.toURL();
5:       urls = new java.net.URL(url);
6:    } catch (Exception e) {
7:       System.out.println(" Directory not found!");
8:    }
9:
10: ...
11:
12:   java.lang.ClassLoader cl = new java.net.URLClassLoader(urls);
13:   java.lang.Class cls = cl.loadClass("icas.AquisitionEngine.Sensor2");
14:   java.lang.Object m = cls.newInstance();
15: ...
```

Código 6.19 – Carregamento de sensores em *runtime* na classe *AquisitionEngineCore.*

No Código 6.20 é apresentada uma parte das instruções do construtor da classe *Sensor2*, fazendo uma leitura de dados em intervalos de 10 segundos e disparando o evento para notificar a classe *ContextAgregator* quando existem novos dados.

```
1:  ...
2:  for (;;) {
3:    try {
4:      Thread.sleep(10000); //sleep for 10000 ms
5:    } catch (InterruptedException ex) {
6:      Logger.getLogger(LocationAquisition.class.getName()).log(
7:         Level.SEVERE, null, ex);
8:    }
9:    System.out.println(" Collecting new data from sensor 2... ");
10:   LinkedList data = (LinkedList) getData();
11:   ContextAgregator agregator = new ContextAgregator();
12:   addAgregatorEventListner(agregator);
13:   updateSensor(2, data);
14:   fireAgregatorEvent(new AquisitionEvent(this));
15: }
16: ..
```

Código 6.20 – Exemplo do construtor de um sensor.

6.5 Motor de adaptação de conteúdos

Como foi especificado na secção 5.6.4, este componente deverá receber os resultados dos serviços que são executados e adaptar o seu conteúdo ao terminal do utilizador. A complexidade deste motor pode aumentar consideravelmente, à medida que se pretende abranger os diversos tipos de dados que os serviços podem retornar, características dos clientes, dispositivos e outros parâmetros relevantes na utilização de um serviço. Deste modo, na implementação deste motor pretende-se exemplificar a sua funcionalidade considerando um cenário relativo à execução de serviços que retornam informação de localização (por exemplo: coordenadas GPS ou uma morada). Assim, com base na análise das propriedades do

software do cliente que ordenou a execução do serviço, os dados poderão ser devolvidos em diferentes formatos.

O *motor de adaptação* é composto pela classe principal *ContentAdaptation* que fornece as funcionalidade de adaptação de conteúdos ao *motor de composição* e ao *motor de contexto*, e por uma classe auxiliar *GeocodeProcessor* adaptada de (Google, 2008).

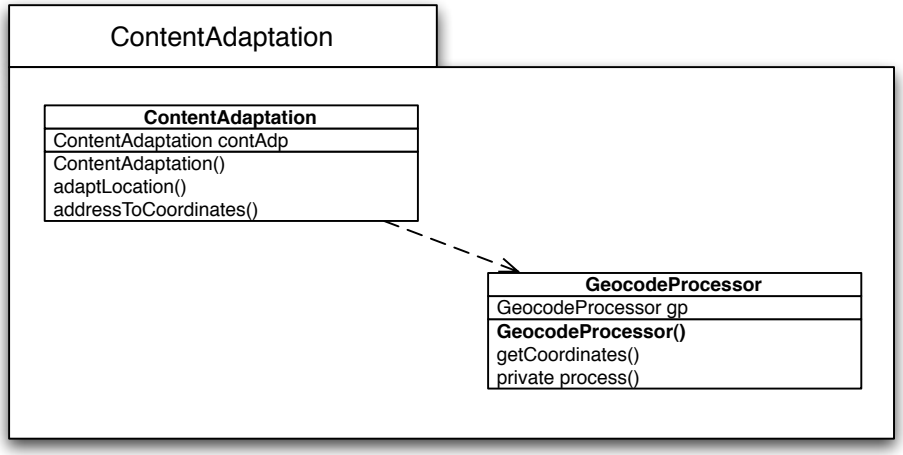


Figura 6.9 – Classes do *motor de adaptação de conteúdos*.

A classe *ContentAdaptation* implementa um método *adaptLocation*, que analisa através da classe *Queries* qual o cliente de software que é usado pelo utilizador, através de uma *query* SPARQL à ontologia *Device* (classe *UserAgent*, *datatype http://icas.ipb.pt/secom/device#softwareName*) e, dependendo do cliente utilizado pode retornar diferentes tipo de dados *GeoCoordinates* (ex. para o cliente *iCas*), um URL para um navegador *Web* recorrendo aos serviços do *Google Maps* ou uma localização no formato KML (OGC, 2008). Outra das funcionalidades é a transformação de um endereço em coordenadas GPS, através da classe *GeocodeProcessor* que recorre a um *Web Service* da Google API e que pode ser usado caso o cliente não suporte o processamento de endereços em mapas, podendo o resultado do serviço ser adaptado para coordenadas GPS.

Embora implementado com algumas funcionalidades simples de adaptação, este motor não foi integrado e usado pelos outros componentes da arquitectura, devido a restrições temporais.

6.6 Administração de informação contextual

Para que o administrador pudesse gerir dados contextuais de forma explícita (por exemplo,

adicionar eventos, actividades, utilizadores e outras informações) foi implementado um componente que fornece uma interface gráfica que permite as funções de adicionar, alterar e remover dados do modelo contextual.

A Figura 6.5 ilustra o diagrama de classes do componente de administração de contexto. A classe *AdministrationPanelGui* implementa a interface gráfica que permite a administração da informação de contexto definida pelo modelo contextual. Esta classe utiliza por sua vez a classe *PersistentOntology*, que lhe dá acesso ao modelo contextual e permite gerir de forma explícita informação contextual definida por cada uma das ontologias: *actor, activity, spatial, spatialevent, temporalevent, time, device, relationship, knowledge, role, contact, project, document.* Os formulários para gerir os dados são construídos através das classes, das propriedades dos objectos e dos tipos dos dados de cada umas das ontologias carregadas pela aplicação cliente.

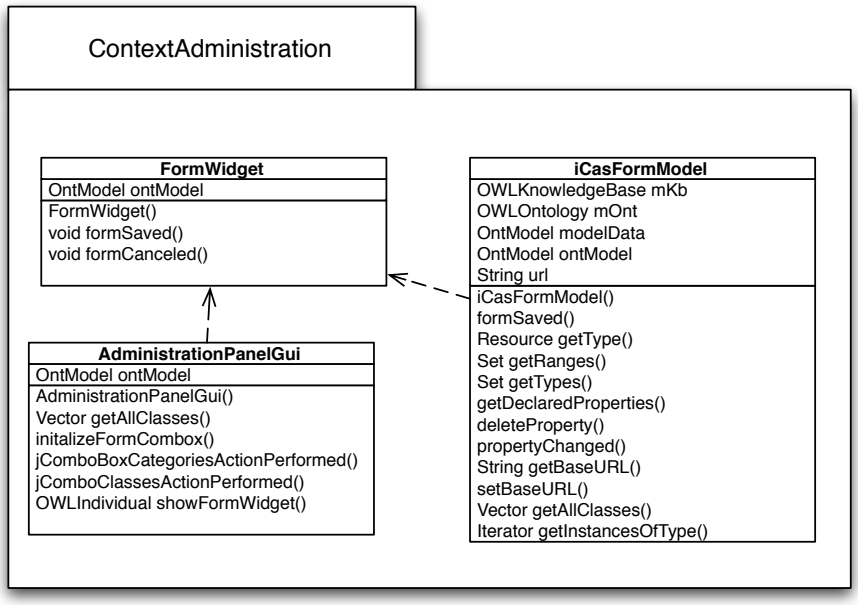


Figura 6.10 - Diagrama de classes do componente de administração de contexto.

6.7 Cliente

Para testar a implementação do modelo apresentado no capítulo 5, desenvolveu-se um protótipo de uma aplicação que permite aos utilizadores comporem serviços dinamicamente de uma forma *ad-hoc* e receberem informações contextuais da plataforma *iCas*.

O protótipo da aplicação cliente é bastante simples em relação às funcionalidades suportadas,

pois o objectivo não era desenvolver uma aplicação profissional, mas validar as soluções propostas no modelo apresentado. Deste modo, as funcionalidades de composição implementadas são a composição sequencial de serviços auxiliada por informações contextuais do utilizador, a gestão de informação contextual e o visualizador de informação.

A Figura 6.11 ilustra a arquitectura interna do protótipo da aplicação de composição dinâmica de serviços sensível ao contexto (cliente *iCas*). O componente de interface de composição fornece as funcionalidades de composição, execução, armazenamento e partilha de serviços, e faz uso dos serviços fornecidos pelo *iCas* através do proxy respectivo. O componente gestor de perfil fornece as funções de adicionar e actualizar informações do perfil do utilizador através do serviço de gestão de perfil. O gestor de administração, embora apareça na arquitectura deste cliente é disponibilizado localmente como aplicação única e apenas ao administrador para que ele possa gerir qualquer tipo de informação do modelo contextual. O visualizador de informação lê os dados no formato RSS para apresentar informação actualizada ao utilizador.

Figura 6.11 – Arquitectura interna da aplicação do cliente *iCas*.

A Figura 6.12 apresenta o diagrama de classes do cliente *iCas*. A classe *iCasComposer* é a classe principal, que implementa a interface gráfica com o utilizador e as funcionalidades de composição de serviços. A classe *NewsNeedProxy* é utilizada para ler as notícias disponibilizadas pela plataforma *iCas*. A classe *Map* permite apresentar o mapa no protótipo para visualizar os resultados relativo à localização. A classe *FormFrame* implementa os formulários que permitem editar as informações de perfil do utilizador.

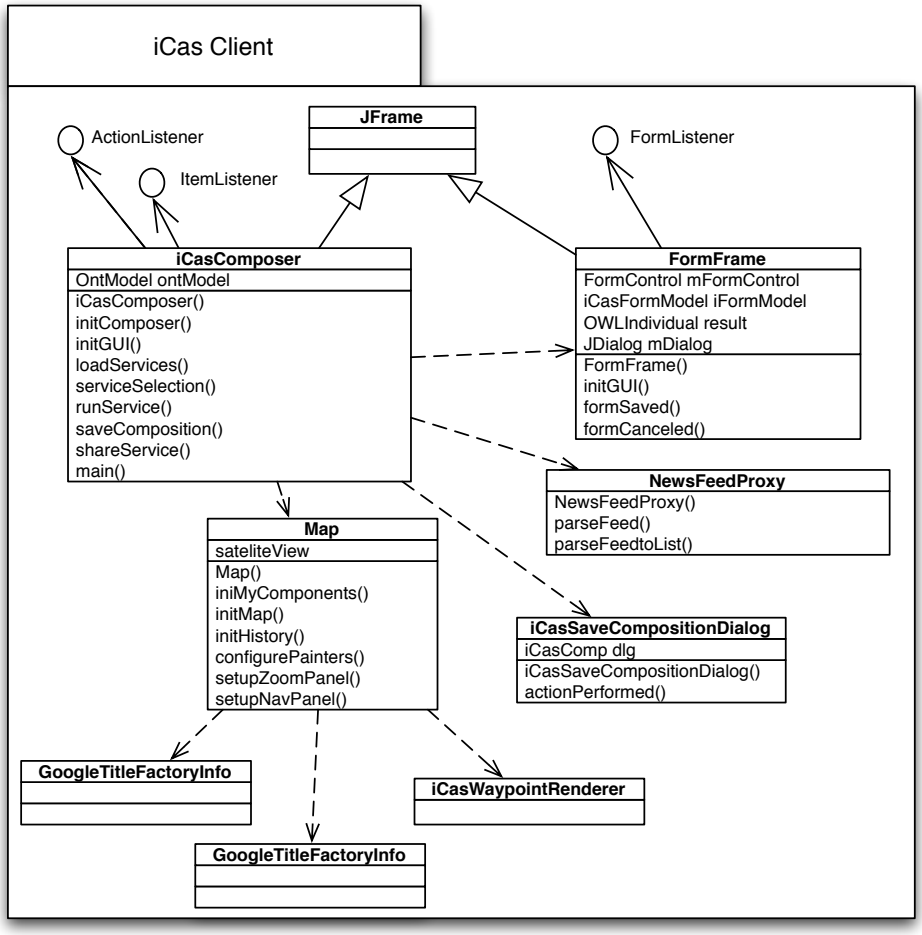


Figura 6.12 – Diagrama de classes do cliente *iCas*.

A Figura 6.13 ilustra o painel de composição de serviços, no qual o utilizador inicia a composição escolhendo um dos serviço das várias classes de serviços disponíveis no sistema.

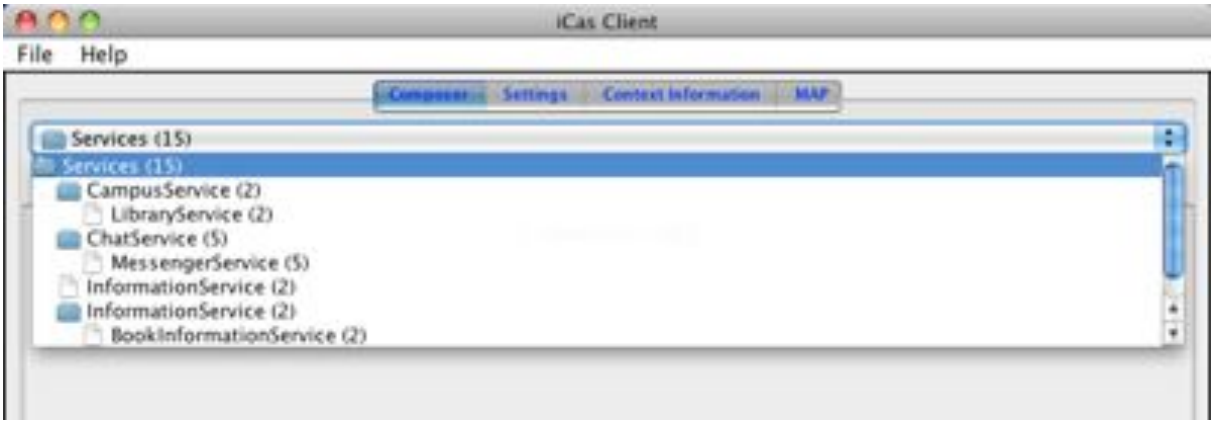


Figura 6.13 - Selecção de serviços por categorias.

Na Figura 6.14 pode-se ver o *debug* do mecanismo de *score* de serviços, após a selecção do serviço *reserveBook* para a composição representada na Figura 6.15. Os resultados apresentados têm em consideração o utilizador (*jpaulo*) e o último serviço seleccionado

(*reserveBook*), e mostram os serviços que podem ser combinados com este serviço por ordem de importância segundo o mecanismo de *scoring* anteriormente apresentado. Os serviços que apresentam *score* zero são serviços cujo *score* dos parâmetros no perfil respectivo é nulo ou que não têm os parâmetros de avaliação.

```
- Expressivity: SHOIN(D), Classes: 200 Properties: 294 Individuals: 559 Strategy: SHOINStrategy
- ABox consistency for 0 individuals
- Consistent: true Tree depth: 2 Tree size: 1256 Time: 165
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/searchBook.owl Scoring : 56.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/reserveBook.owl Scoring : 38.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/OMS2GetUserList.owl Scoring : 27.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/OMS2_AddUserToChat_simple.owl Scoring : 22.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/OMS2_SetStatus_simple.owl Scoring : 22.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/OMS2_SendMessageToChat_simple.owl Scoring : 21.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/OMS2_CreateChat_simple.owl Scoring : 19.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/getItemPrice.owl Scoring : 19.0
User: jpaulo Serviço: http://icas.ipb.pt/icas/owl-s/GetAllCountryCurrencies.owl Scoring : 17.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/BabelFishTranslator.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/FindCheaperBook.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/FindLatLong.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/GetTemperature.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/GetWeather.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/GooglePhoto.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/GoogleSearch.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/Hurricane.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/WeatherInfo.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/ZipCodeFinder.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/ZipCodeSupplier.owl Scoring : 0.0
User: jpaulo Serviço: http://icas.ipb.pt/2004/owl-s/1.1/ZipCodeTemp.owl Scoring : 0.0
```

Figura 6.14 – *Debug* do *score* dos serviços no servidor.

A Figura 6.15 ilustra a interface gráfica de composição de serviços do cliente *iCas* adaptado do trabalho desenvolvido em (Services, 2004) e permite a composição de um serviço através da selecção de serviços que são mostrados em *combobox,* e que termina quando o utilizador selecciona como última entrada um input do serviço ao invés de um novo serviço. A composição ilustrada na Figura 6.15 mostra a criação de um serviço, que permite ao utilizador reservar um livro. Como só é possível reservar um livro dado o seu ISBN, que ele desconhece, o utilizador procura um serviço que lhe possa dar essa informação de uma forma mais simples. Neste caso o *iCas* propôs uma lista de serviços, ordenada segundo o seu contexto e preferência, que podem ser combinados com esse serviço. O utilizador seleccionou *searchBook*, que neste caso tem como entrada o título do livro e retorna o respectivo ISBN e o resultado dessa saída será a entrada do *reserveBook*. Por fim, o utilizador introduziu o título do livro e através do botão *run* ordena a execução do serviço invocando o método *runService* do *Web Service CompositionEngineCore.*

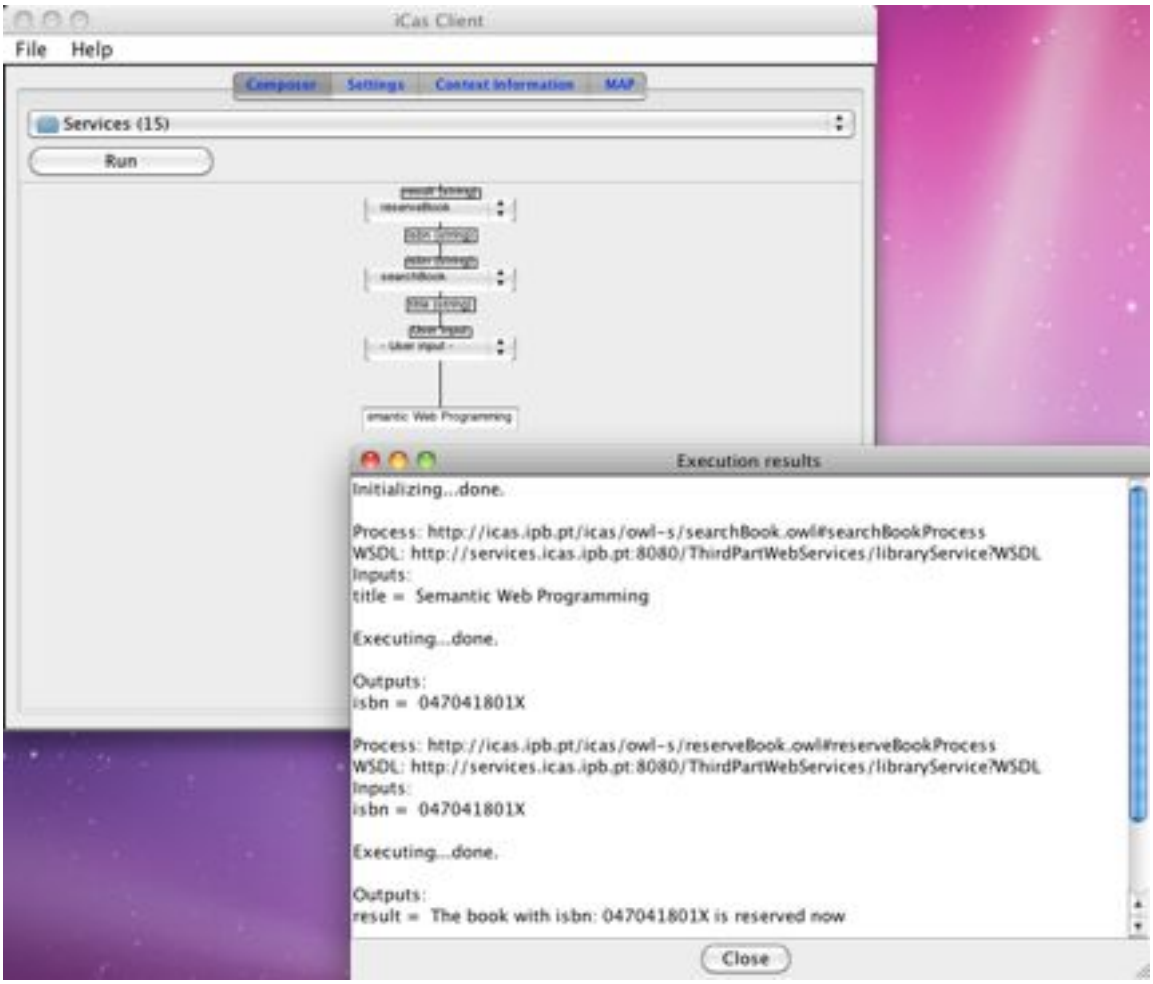


Figura 6.15 – Composição e execução de um serviço.

A Figura 6.16 ilustra o separador *Settings* onde é possível editar o perfil e preferências do utilizador. Para este protótipo foi implementada a funcionalidade edição de informações do perfil do utilizador através de formulários. Estes formulários utilizam o *Web Service ContextEngineWS* para apresentarem e alterarem as informações do perfil do utilizador.

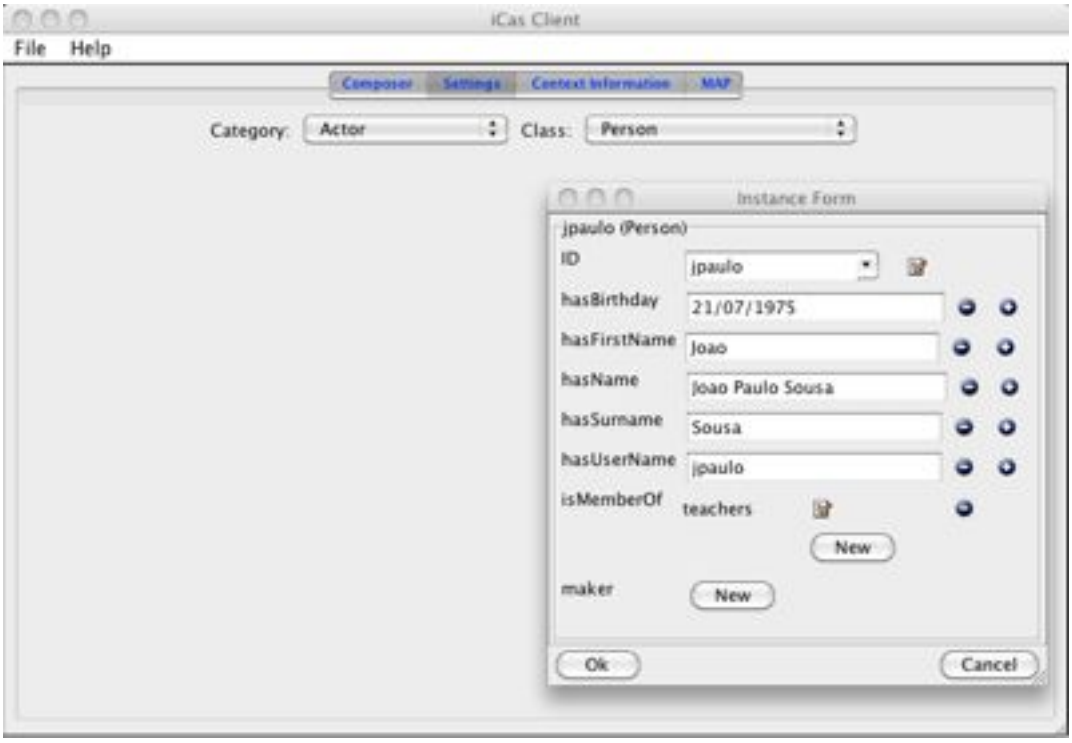


Figura 6.16 – Edição de informações de perfil do utilizador.

Através do separador *Context Information*, ilustrado na Figura 6.17, o utilizador poderá consultar informações disponibilizadas pela plataforma *iCas* sob a forma de notícias. Embora esta funcionalidade não estivesse inicialmente prevista, optou-se por implementa-la, por dois motivos: o primeiro é que parte do mecanismo estava já desenvolvido através do modelo ontológico persistente e das funcionalidades de consulta e inferência disponibilizadas pelas classes *Queries* e *Inference*, respectivamente; o segundo motivo é que através deste mecanismo facilmente se disponibilizam informações úteis a um utilizador que estejam armazenadas num modelo ontológico persistente (ex. poder visualizar os eventos agendados dentro do campus). A parte de *RSS feeds* foi adaptada utilizando código de diversos exemplos disponíveis em (Johnson, 2006), de modo a fornecer as informações de forma autónoma. Para isso, procura indivíduos da classe *SpatialEvent* e escreve esses eventos em formato RSS num ficheiro, que poderá ser depois lido por qualquer cliente de RSS que suporte o formato RSS 2.0.

Figura 6.17 – Painel de visualização de informações.

O quarto separador do cliente *iCas* (Figura 6.18) mostra um mapa para visualização de resultados georreferenciados. Este visualizador é baseado no projecto (Marinacci, 2007) e foi adicionado ao cliente *iCas* para que os utilizadores pudessem visualizar de forma mais simples dados georreferenciados resultantes da execução de serviços ou de serviços de informações da arquitectura *iCas*. Na Figura 6.18, é ilustrado um exemplo do resultado da execução do serviço *userActivitiesAndLocation*, que retorna a actividade de um utilizador e a sua localização. Este painel também poderá ser usado para mostrar resultados das funções avançadas de consulta de dados disponibilizadas pelo *motor de contexto*, como por exemplo saber quais as actividades dos amigos do utilizador e a sua localização, implementada pelo método *getFriendsActivitiesAndLocation* do *Web Service ContextEngineWS* e que faz uso das *queries* implementadas pelos métodos *getNeighborsFriends* e *getCurrentActivityByNode* da classe *queries*.

Figura 6.18 – Visualizador de resultados georreferenciados.

6.8 Criação de serviços semânticos

Todos os serviços suportados por esta arquitectura são *Web Services Semânticos* ou serviços descritos em OWL-S. No entanto, a maior parte dos serviços oferecidos pelas diversas plataformas são desenvolvidos como *Web Services* com interface WSDL, havendo neste caso a necessidade de os transformar em serviços OWL-S.

A descrição de um serviço semântico é dividida em três partes: a descrição do perfil do serviço, do modelo de processo e do *grounding*. Para facilitar a criação destas três componentes descritivas existem ferramentas (apresentadas na secção 3.3.3.2) que fazem a transformação da descrição de um serviço em WSDL para OWL-S. Estas ferramentas ajudam a criar em OWL um esqueleto descritivo semântico do serviço, que pode depois ser enriquecido semanticamente (principalmente o perfil do serviço) com a edição através de uma ferramenta de edição de ontologias. No entanto, todas estas ferramentas apresentam problemas quando é necessário converter tipos de dados complexos dos parâmetros de entrada ou saída que um serviço WSDL possa ter. Isto é, quando as entradas e saídas OWL são do tipo XSD (Peterson, Shudi et al., 2008) (ex. *strings*, inteiros, etc.) esse mapeamento é simples de conseguir e é realizado por algumas ferramentas que transformam serviços WSDL para

serviços OWL-S. Mas existem situações em que os serviços OWL-S contém parâmetros OWL descritos através de classes OWL. Também nestes casos esses parâmetros têm que ser transformados em tipos de dados XSD e muitas vezes em tipos complexos (tipos de dados que descrevem elementos ou com possíveis atributos ou estruturas de sub-elementos).

Como já foi abordado na secção 4.4.1.3, o *Grounding* de um serviço OWL-S descreve como executar esse serviço. Caso esse serviço seja fornecido por *Web Services* padrão contém descrições WSDL. Nestes casos, a informação do *Grounding* vai ser constituída por apontadores para essas descrições WSDL, capazes de invocar directamente esse *Web Service*. Deste modo o *Grounding* vai funcionar como mecanismo de *binding* que mapeia entradas e saídas OWL em tipos de dados XML usados nas mensagens de entrada e saída WSDL e vice-versa. Na Figura 6.19, é possível ver o funcionamento do *Groundig*, principalmente as várias formas de relacionamento de entradas e saídas de um serviço atómico OWL-S com partes de mensagens WSDL.

Figura 6.19 – Possibilidades de mapeamento entre parâmetros de um serviço OWL-S e as partes das mensagens WSDL, adaptado de (Saadati and Denker, 2005).

Por norma a OWL-S permite a utilização da XSL *Transformations* – XSLT[5] (Clark, 1999) para mostrar como cada entrada WSDL é derivada de uma ou mais entradas OWL-S e como cada saída OWL-S é derivada de uma ou mais partes das saídas WSDL (Martin, Burstein et al., 2004). Um dos métodos para realizar essa transformação é utilizar a propriedade *xsltTransformationString* e folhas de estilo XSLT (XSLT *stylesheet*) para transformar parâmetros OWL em mensagens WSDL e vice-versa. Essas folhas de estilo inseridas na propriedade *xsltTransformationString* contêm funções de transformação que vão ser reconhecidas pelo executor OWL-S e que irão forçar a transformação de WSDL para indivíduos/instâncias OWL e vice versa. Por exemplo, o Código 6.21 mostra uma folha de estilo XSLT inserida no *Grounding* de um parâmetro de saída OWL-S, para interpretar uma resposta XML (Código 6.22) de um parâmetro WSDL e convertê-lo num grafo RDF serializado em sintaxe RDF/XML. Neste caso, é feita a transformação de uma estrutura complexa para uma classe OWL chamada *iCasOutput*, que contém quatro *datatypes* do tipo *string* (*person*, *activity*, *latitude* e *longitude*).

[5] A linguagem XSLT é usada para transformar documentos XML. Esta transformação é normalmente utilizada em dois casos: transformar dados de um formato para outro, utilizado por exemplo em transacções B2B; e para a publicação, transformando dados para uma formato visivelmente amigável aos humanos (ex. HTML e PDF).

```
<grounding:wsdlOutput>
  <grounding:WsdlOutputMessageMap>
    <grounding:owlsParameter rdf:resource="#&owlsOutput;"/>
     <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">&wsdl;#&jparaReturn;</grounding:wsdlMessagePart>
      <grounding:xsltTransformationString>
       <![CDATA[
        <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns2="http://icas.ipb.pt/schemas">
        <xsl:template match="/">
          <xsl:variable name="person" select="/S:Envelope/S:Body/ns2:getActvLocaFrindResponse/infloc/person"/>
          <xsl:variable name="activity" select="/S:Envelope/S:Body/ns2:getActvLocaFrindResponse/infloc/activity"/>
          <xsl:variable name="latitude" select="/S:Envelope/S:Body/ns2:getActvLocaFrindResponse/infloc/latitude"/>
           <xsl:variable name="longitude" select="/S:Envelope/S:Body/ns2:getActvLocaFrindResponse/infloc/longitude"/>
            <rdf:RDF
              xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
              xmlns:icasout="http://icas.ipb.pt/icas/owl-s/iCasOutput.owl#"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
              xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
              xmlns:owl="http://www.w3.org/2002/07/owl#">
                <icasout:iCasOutput>
                   <icasout:person rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select="$person"/>
                   </icasout:person>
                   <icasout:activity rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select="$activity"/>
                   </icasout:activity>
                   <icasout:latitude rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select="$latitude"/>
                   </icasout:latitude>
                   <icasout:longitude rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                     <xsl:value-of select="$longitude"/>
                   </icasout:longitude>
                </icasout:iCasOutput>
            </rdf:RDF>
        </xsl:template>
        </xsl:stylesheet>
       ]]>
      </grounding:xsltTransformationString>
  </grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
```

Código 6.21 – Transformação de parâmetros WSDL para OWL.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
3:   <S:Body>
4:    <ns2:getActvLocaFrindResponse xmlns:ns2="http://icas.ipb.pt/schemas">
5:       <infloc>
6:          <person>Maria</person>
7:          <activity>Studying botany for the next exam</activity>
8:          <latitude>-7.738622</latitude>
9:          <longitude>41.286975</longitude>
10:      </infloc>
11:    </ns2:getActvLocaFrindResponse>
12:  </S:Body>
13:</S:Envelope>
```

Código 6.22 – Exemplo de resposta SOAP da invocação de um *Web Service*.

Infelizmente escrever estas funções de transformação não é fácil. Estas funções podem ser fontes de erros e requerem conhecimento aprofundado sobre OWL-S, mensagens WSDL e sintaxe RDF. No entanto, esta operação não afecta o modo como os parâmetros são vistos por outros serviços OWL, pois estes parâmetros serão sempre descritos em OWL, suportando o mecanismo de comparação que é utilizado no *motor de composição*.

6.9 Resumo

Este capítulo apresentou as características de implementação e análise mais relevantes do protótipo do sistema proposto. Inicialmente, foi descrita a arquitectura do sistemas, fazendo o resumo de cada uma das partes que a compõem. Seguidamente foram enunciadas e justificadas as escolhas das tecnologias, bibliotecas e ferramentas utilizadas para o desenvolvimento do protótipo da arquitectura *iCas*. O desenvolvimento do capítulo consistiu na descrição pormenorizada das partes mais relevantes de cada um dos motores que compõem a arquitectura *iCas*. Posteriormente, foi abordada a parte da criação dos serviços semânticos, principalmente o problema dos parâmetros dos serviços que são de tipos de dados complexos.

Capítulo 7

ANÁLISE DO DESEMPENHO DO SISTEMA

Neste capítulo é feita uma análise do comportamento do sistema que pretende validar o modelo proposto nesta tese, através da verificação do correcto funcionamento da arquitectura e do seu desempenho. Para isso, foram realizados conjuntos de testes, cujos resultados são ilustrados através de tabelas e gráficos, acompanhados de explicações e da discussão dos mesmos. Finalmente, é feito um resumo dos resultados obtidos para o desempenho do sistema nos diversos testes que foram realizados.

7.1.1 Cenário de testes

No capítulo anterior foi apresentado um protótipo de uma aplicação cliente que pode ser testado por utilizadores. No entanto, devido às dificuldades de simular condições reais que forneçam dados sobre o contexto do utilizador necessários para fazer uma avaliação heurística (Nielsen and Molich, 1990), optou-se por avaliar o aspecto funcional e o desempenho dos principais componentes do sistema. Para isso, foi implementado um cenário para a realização de testes e foram desenvolvidas aplicações cliente que fazem diversos pedidos para serem executados pela *iCas*.

No cenário de testes foram usados dois computadores ligados à rede sem fios "*eduroam*" do campus universitário (IEEE 802.11g). O computador 1 (C1) é um Intel Core 2 Duo 7400 (2.12GHz) 3GB DDR2 com o sistema operativo OS X 10.5.5, que executa a arquitectura *iCas*. Um servidor de aplicações, que contém *Web Services* de terceiros está instalado no computador 2 (C2), um Intel Core 2 Duo T8600 (2.4GHz) 4GB DDR2 e com o sistema

operativo Linux (kernel 2.6.24). Esta máquina pretende simular um servidor de aplicações *Web* que fornece serviços ao campus universitário, tais como serviços da biblioteca, pedagógicos e administrativos.

A análise do desempenho do sistema incide fundamentalmente sobre os *motores de contexto*, composição e *motor de execução*, que contêm algumas das contribuições mais importantes deste trabalho, que são as partes mais críticas do sistema, devido às suas exigências de recursos computacionais, memória principal e o subsistema de entrada e saída de dados.

7.1.2 Testes ao motor de contexto

Com os testes ao *motor de contexto*, pretende-se analisar o seu comportamento relativamente ao processo de escrita e consulta de dados no modelo ontológico persistente.

Como já for referido anteriormente a utilização de um modelo ontológico suportado por base de dados relacionais além de fornecer a característica inerente de persistência dos dados permite:

- trabalhar com modelos de dados significativamente maiores que a memória principal, situação que se prevê no nosso cenário de aplicação;
- resolver problemas de concorrência no acesso às fontes de dados, permitindo explorar as transacções das bases de dados controladas pelo sistema de gestão de base de dados;
- reduzir o tempo de carregamento do modelo ontológico cada vez que a arquitectura é iniciada, porque o modelo não é carregado para a memória. Tem ainda a vantagem de permitir trabalhar com um grande volume de dados.

Não obstante as vantagens anteriormente enunciadas, existe a preocupação de conhecer o comportamento deste motor quando sujeito a uma carga de pedidos de introdução e actualização de novos dados e consulta de dados existentes no modelo contextual.

Para realizar este teste foi criado uma aplicação cliente que faz a introdução de diversos tipos de dados do *motor de contexto*. Para isso é realizado um ciclo sem pausas, que gera dados aleatórios para serem guardados no modelo contextual da arquitectura *iCas*. Como descrito na secção 6.3, quando existe uma operação sobre o modelo contextual é lançado um evento que é depois tratado por uma entidade própria que regista as informações relativas a essa operação

na *base de dados do histórico*.

O Gráfico 7.1 apresenta os tempos consumidos para inserir dados no modelo contextual através do *motor de contexto*, medidos no computador 1 que executa a plataforma *iCas*. Para cada teste foram feitas duas medidas e a tabela mostra a média das duas medidas em milissegundos. Foram escolhidas quatro ontologias que supostamente poderão estar entre as mais solicitadas (*actor*, *places*, *activities* e *devices*) e fizeram inserções de dados às quais correspondem as linhas "adicionar pessoas", "adicionar lugares", "adicionar dispositivos" e "adicionar tarefas". Dos resultados dos testes observou-se que todos os eventos lançados pelo *motor de contexto* foram capturados e as acções foram registadas na base de dados do histórico de acções. Do Gráfico 7.1, é ainda possível observar que o *motor de contexto*, para as operações de inserção, mantém um desempenho relativamente constante e que estas são executadas em tempos bastante satisfatórios.

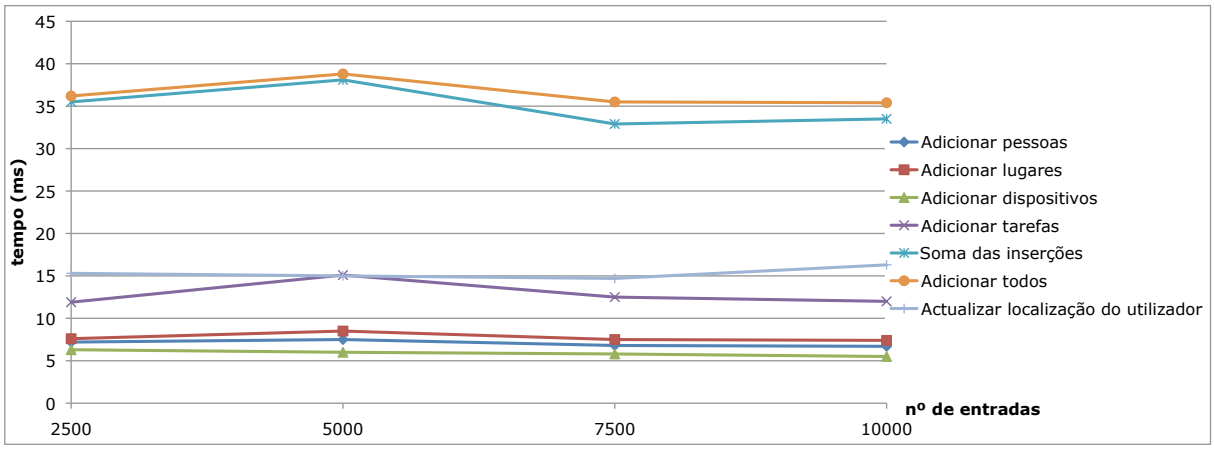


Gráfico 7.1 - Tempos para inserir dados contextuais através do *motor de contexto*.

Também é possível observar que o tempo necessário para inserir dados relativos uma única ontologia ou dados pertencentes a várias ontologias não apresenta variações relevantes.

O teste de actualizar a localização do utilizador, como era esperado, apresenta tempos mais elevados. Isto acontece porque nessa actualização é feita uma *query* para retornar a localização de um determinado utilizador e proceder à sua actualização.

Durante estes testes a base de dados iCasDB chegou a conter mais de 500 mil entradas, correspondentes a aproximadamente 2GB de espaço em disco.

Destes testes é possível retirar diversas conclusões:

- o *motor de contexto* mostrou-se robusto, capaz de suportar cargas intensivas de

inserção e actualização de dados;

- a utilização de um modelo contextual persistente é viável, com tempos de inserção e actualização de dados bastante bons;
- a utilização de base de dados relacionais para guardar o modelo contextual persistente mostrou-se fiável e consistente, não evidenciando corrupção de dados.

Outra das funcionalidades do *motor de contexto* é permitir a consulta de informações contextuais e a inferência de novas informações. Deste modo, foi feito um teste ao desempenho do *motor de contexto*, para analisar o seu comportamento relativamente à consulta de informações e de inferência, que será apresentada em seguida.

A realização do teste da operação de consulta consistiu em realizar um conjunto de consultas, ilustradas no Gráfico 7.2. Este teste difere do apresentado anteriormente porque não sujeita o *motor de contexto* a uma carga de pedidos de consultas, devido ao aumento de complexidade no desenvolvimento do cliente. No entanto, é possível analisar o tempo de resposta de diversos tipos de consultas de informação. Tal como na inserção de dados contextuais, a consulta de contexto está fortemente relacionada com o modelo contextual persistente e as tecnologias por ele utilizado, neste caso o motor de gestão de base de dados *PostgresSQL*.

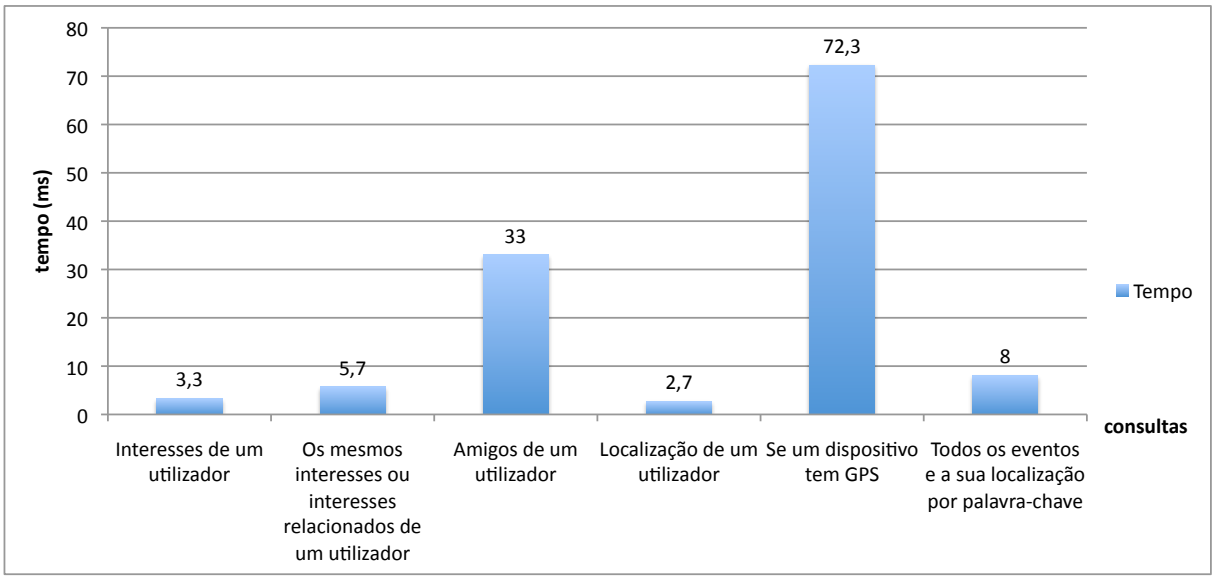


Gráfico 7.2 – Tempos para consultas de dados contextuais através do *motor de contexto*.

As consultas de informação executadas correspondem a diferentes tipos de *queries* SPARQL, mais ou menos complexas e com diferentes propriedades: conjunções, disjunções, construtores que afectam os resultados, grafos temporários, etc. O Gráfico 7.2 apresenta o

tempo consumido por cada consulta em milissegundos. A consulta que consumiu mais tempo é relativa a uma *query* SPARQL do tipo ASK (SPARQL), as restantes consultas correspondem a *queries* do tipo SELECT. Segundo (Iqbal, Ott et al., 2009) o tempo consumido por uma query do tipo ASK, é normalmente menor e no pior caso igual ao tempo consumido por uma *query* do tipo SELECT, no entanto, para este teste a *query* do tipo ASK é feita usando um grafo padrão diferente a um conjunto de dados diferentes, não podendo ser comparada às restantes *queries* do tipo SELECT.

Destes resultados é possível concluir que o *motor de contexto* permite tempos de consulta de dados bastante satisfatórios e que também para este caso a utilização de um modelo contextual de forma persistente se revelou uma opção acertada.

Acredita-se ainda que, através da optimização de parâmetros do sistema de gestão de base de dados e através de componentes de hardware dos subsistemas de entrada/saída de dados mais evoluídos, será possível melhorar o desempenho do *motor de contexto* neste tipo de operações.

Relativamente ao desempenho dos processos de inferência do *motor de contexto*, este está fortemente dependente de motor de inferência, neste caso particular o *Pellet*. A métrica mais utilizada para avaliar o desempenho dos processos de inferência baseados em ontologias é o tempo total de resposta, assim como os tempos associados a tarefas de raciocínio ou sub-processos de inferência, como os tempo de carregamento, verificação de consistência, classificação e realização (Tempich and Volz, 2003; Wang, Guo et al., 2005). Alguns dos conceitos destas tarefas de raciocínio foram abordados na secção 3.2.6, mas segue-se uma breve descrição dos tempos dessas tarefas:

- tempo de carregamento: é o tempo que o motor de inferência consome para ler, analisar e carregar as descrições ontológicas relativas a um ficheiro e as triplas RDF. Este processo ocorre antes do processo de inferência e está fortemente dependente do número de indivíduos e de as descrições estarem em ficheiros remotos ou ficheiros locais;
- tempo de consistência: é o tempo consumido pelo motor de inferência para verificar se a definição de uma ontologia é consistente, isto é, verificar se contém axiomas ou factos inconsistentes ou contraditórios com a sua definição;
- tempo de classificação: é o tempo gasto pelo motor de inferência para construir o

grafo completo envolvendo todas as classes da ontologia, mais o tempo de criação da hierarquia de classes da ontologia. Permite encontrar todas as instâncias que pertencem a uma classe;

- tempo de realização: é o tempo consumido pelo motor de inferência, para encontrar as classes específicas a que um individuo pertence, isto é, consiste em encontrar as classes de que um determinado indivíduo é membro;
- tempo total: corresponde à soma dos tempos de carregamento, consistência, classificação e realização.

A Tabela 7.1 mostra os tempos médios, em segundos, para as diversas tarefas de raciocínio desempenhadas pelo *motor de contexto* e executadas para cada uma das ontologias. De todas as ontologias do modelo ontológico, foram escolhidas estas sete por se achar que serão as mais utilizadas e que é entre estas que se encontram as ontologias com maior complexidade computacional. Para cada uma das ontologias foi executada a inferência três vezes, sendo o tempo final a média dos tempos consumidos nessas execuções.

Tabela 7.1 – Propriedades e tempos médios (segundos) dos processos de inferência de cada uma das ontologias.

Ontologias	Dialecto	Triplas	Classes	Propriedade	Indivíduos	Carregar	Consistência	Classificar	Realizar	Total
Actor	Lite	65906	9	10	10787	4,13	1,03	0,24	0,69	5,81
Activity	DL	7976	89	189	3375	0,39	0,67	2,72	671,28	674,99
Spatial	DL	64772	21	14	834	0,52	0,57	0,80	0,58	2,47
Knowledge	Lite	59950	10	13	10795	2,19	0,36	0,01	0,23	2,76
Contact	DL	16051	15	21	6794	1,03	0,15	0,01	0,20	1,37
Device	DL	162715	44	103	7331	3,36	1,62	3,73	12,63	21,30
Relationshi	DL	54000	9	21	8787	1,74	0,30	0,00	0,19	2,21

As ontologias *Activity*, *Spatial*, *Contact*, *Device*, *Relationship* são as que apresentam maior expressividade, porque utilizam construtores do dialecto *OWL DL*, enquanto que as ontologias *Actor* e *Knowledge* são descritas com construtores do dialecto *OWL Lite*. Da análise estrutural (classe, propriedades, triplas, indivíduos) é possível observar que as ontologias com maior expressividade apresentam um maior número de classes e propriedades e como consequência um maior número de triplas para o mesmo número de indivíduos.

Dos testes realizados observa-se que o número de indivíduos tem um impacto significativo sobre o tempo total. Exceptuando as ontologias *Activity* e *Device*, o carregamento da ontologia é o processo que consume mais tempo nas diversas tarefas de inferência. Por

exemplo, no Gráfico 7.3 é possível observar a variação do tempo de carregamento com o número de indivíduos, verificando-se que o aumento de indivíduos afecta essencialmente o tempo de carregamento. Quantos mais indivíduos uma ontologia possuir maior será o seu tempo de carregamento. Para a mesma ontologia, o processo de carregamento representou 71,2% do tempo total.

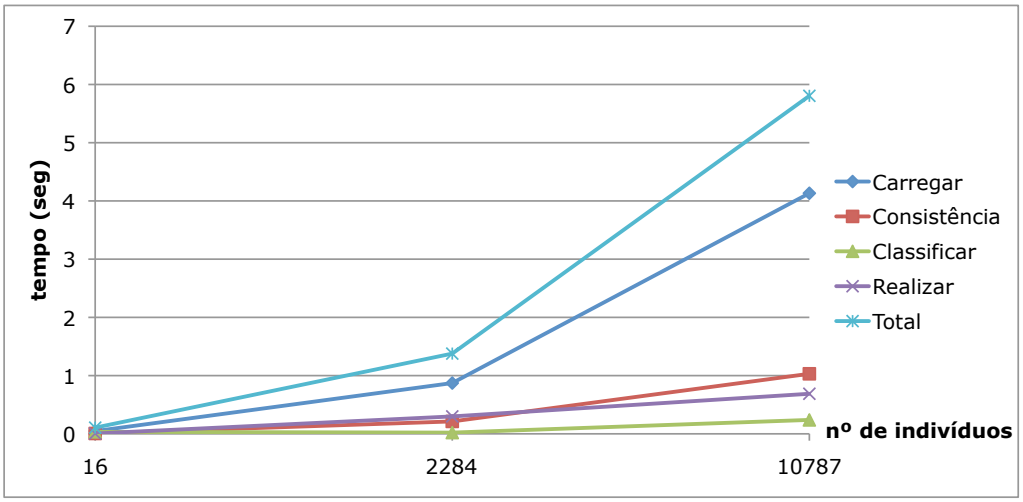


Gráfico 7.3 – Análise dos processos de inferência com o aumento do número de indivíduos para a ontologia *Actor*.

Na Tabela 7.1 é também possível observar que as ontologias *Device* e *Activity* apresentam tempos que se destacam dos restantes. Para estes dois casos, a tarefa que consome mais tempo é o processo de realização que corresponde a 99,4% do tempo total para a ontologia *Activity* e 59,2% para a ontologia *Device*. Este aumento está relacionado com o nível de complexidade da ontologia, número de classes e indivíduos; no entanto neste trabalho não se pretende analisar com detalhe as razões destes resultados. No Gráfico 7.4 (eixo do y com escala logarítmica), é possível observar o comportamento dos diversos processos de inferência com o aumento do número de indivíduos na ontologia Activity. Para as três amostragens verifica-se o aumento do impacto do processo de realização com o incremento do número de indivíduos, que para 142 indivíduos é de 63% do tempo total e subindo para 95,5% para 1216 indivíduos e 99,5% para 3375 indivíduos.

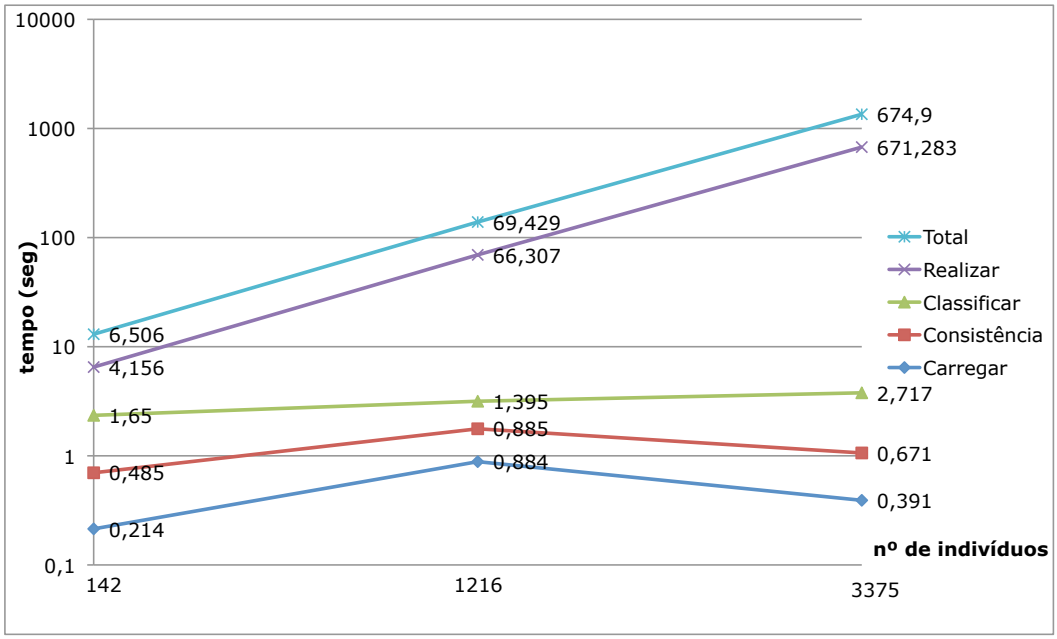


Gráfico 7.4 – Análise dos processos de inferência com o aumento do número de indivíduos para a ontologia *Activity*.

Dos valores da Tabela 7.1 é ainda possível observar que o tempo do processo de realização principalmente para a ontologia *Activity*, é bastante mais elevado do que para as restantes ontologias. Embora este processo seja apenas executado uma vez (quando é iniciada a plataforma) mantendo-se os resultados em memória, são apontadas algumas soluções possíveis para minimizar este tipo de problema, entre as quais se destacam:

- a vigilância do administrador do número de instâncias das ontologias que apresentam os valores mais elevados;
- optimização dos parâmetros de configuração do motor de inferência;
- utilizar uma versão mais recente do motor de inferência, que segundo indicação da equipa de desenvolvimento, está mais optimizado para este processo. No entanto, esta solução implica alteração do código fonte da plataforma *iCas*.

7.1.3 Testes ao *motor de composição*

Os testes ao *motor de composição* pretendem analisar o seu comportamento, especialmente o seu correcto funcionamento e a seu desempenho quando sujeito a uma elevada carga de pedidos para composição e execução de serviços em simultâneo. Os valores considerados nos diversos testes pretendem simular o cenário de aplicação da arquitectura *iCas* descrita na secção 5.7.

A Tabela 7.2 apresenta os resultados do teste do mecanismo de selecção de serviços, descrito na secção 6.2.1. A segunda coluna mostra o tempo para ler a descrição dos serviços e para verificar a sua consistência, para diversos números de serviços, especificados na primeira coluna. De referir que o carregamento e verificação de consistência das ontologias apenas ocorrem uma vez, quando a arquitectura é inicializada e é feita a leitura dos serviços para a memória principal. Quando um novo serviço é adicionado ou removido no *repositório de serviços* apenas esse serviço é adicionado ou retirado da base de conhecimento e é uma operação rápida, pelo facto de se tratar de carregar/libertar uma ontologia (normalmente sem grande complexidade) da base de conhecimento. A terceira coluna mostra o tempo consumido para seleccionar os serviços segundo o contexto do utilizador e que lhe serão entregues.

Tabela 7.2 – Tempos médios para o carregamento e verificações de consistência de serviços.

Nº. de Serviços	Carregar e verificar a consistência das ontologias (seg)	Selecção de serviços (ms)
10	8,96	104,56
20	31,02	180,72
30	80,73	378,38

Para os testes realizados ao *motor de composição*, foram utilizados dois tipos de serviços semânticos (atómicos e compostos) e apenas é dada importância à descrição semântica do seu perfil, visto que a parte lógica do serviço do serviço WSDL não influencia nenhuma das tarefas do *motor de composição*.

Todas as descrições dos serviços utilizados na plataforma *iCas* podem ser carregadas por HTTP ou directamente do sistema de ficheiros. Para a realização destes testes, todos os serviços foram carregados por HTTP; no entanto, estavam localizados no mesmo servidor que executa a plataforma *iCas* e, como tal, o tempo de carregamento foi bastante rápido (Gráfico 7.5). Se os serviços semânticos se encontrarem num servidor Web externo, é previsível que o tempo de carregamento aumente pois estará principalmente dependente da velocidade da ligação. Como consequência irá provocar um aumento do tempo de carregamento, passando a este a ser o tempo que mais influencia o processo de carregamento e verificação, que são normalmente executados em conjunto.

O Gráfico 7.5 mostra o tempo médio necessário para carregar e verificar a consistência de cada serviço. O processo de carregamento e de verificação de consistência executado é semelhante ao descrito na secção anterior, já que um serviço consiste numa ontologia que

descreve os vários aspectos de um serviço semântico. No Gráfico 7.5 é possível verificar que o tempo de carregamento médio de um serviço aumenta com o número de serviços a carregar. Neste teste observa-se que o tempo médio de carregamento de um serviço aumentou da primeira amostragem (10 serviços) para a terceira amostragem (38 serviços) 235%. Embora seja um incremento significativo, considera-se que não é preocupante para um processo que irá ocorrer na iniciação da plataforma. No entanto, também para este caso podem ser adoptadas as soluções apontadas no final da secção 7.1.2.

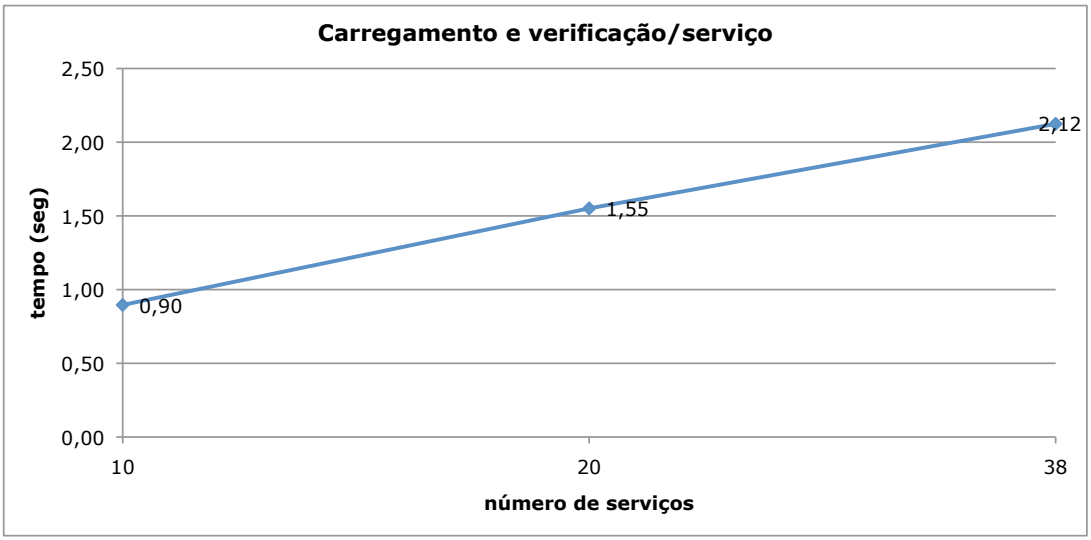


Gráfico 7.5 – Tempos do processo de carregamento e verificação dos serviços semânticos.

O Gráfico 7.6 ilustra o tempo médio para fazer a selecção de um serviço semântico pelo mecanismo de selecção de serviços com base no contexto e preferências do utilizador, explicado na secção 6.3. No mesmo gráfico é possível observar que o tempo necessário para a selecção de um serviço se manteve relativamente estável com tempos inferior à dezena de milésimas de segundos.

Gráfico 7.6 – Tempos do processo de selecção de serviços.

Para testar a composição e a execução de serviços, foi desenvolvida uma pequena aplicação cliente que lança pedidos concorrentes de composição e execução à plataforma *iCas*. Este teste pretendia analisar como este componente responde a diversas cargas de diferentes composições e execuções de serviços. O teste consistiu na variação de três parâmetros: o número de serviços utilizados na composição, o número de pedidos lançados pelo cliente e o tempo de execução dos serviços WSDL que são invocados pelos serviços semânticos que fazem parte da composição. Cada *thread* simula uma ordem de composição e a execução de um novo serviço semântico, que quando executado invoca *Web Services* tradicionais descritos no seu *Grounding* que estão num servidor de aplicações (computador C2).

As composições realizadas incluem 2, 4, 6, e 8 serviços atómicos associados sequencialmente. O número de pedidos feitos pelos clientes varia de 50 a 300 *threads* em incrementos de 50 e são lançados com intervalos de 100 milissegundos. Foram criados dois serviços semânticos, cada um descreve um *Web Service* tradicional configurado para consumir tempo (*Tempo de Consumo do Web Service* – TCWS): 20 milissegundos, 0,5 segundos e 4 segundos. Isto é, quando é dito que um serviço contém um TCWS de 0,5, significa que o *Web Service* contém um temporizador que consome 0,5 segundos. Deste modo, garante-se o tempo consumido para executar uma determinada tarefa, isto porque para a realização destes testes não importa a funcionalidade do serviço, mas sim o tempo que ele consome. Todas as composições e execuções solicitadas foram executadas com sucesso. Em seguida é apresentada para discussão uma das tabelas e um dos gráficos dos diversos testes realizados ao *motor de composição*. Os restantes resultados podem ser consultados no anexo III.

A Tabela 7.3 mostra o tempo total em segundos para a composição e execução de serviços, para o caso em que os *Web Services* tradicionais têm um TCWS de 0,5 segundos. É possível observar que a tarefa de executar 300 pedidos de composições lançados com intervalos de 100 milissegundos, em que cada pedido solicita uma composição de 8 serviços, demorou aproximadamente 120 segundos para ser concluída.

Tabela 7.3 – Tempos (segundos) para a composição e execução de serviços (TCWS de 0.5 seg).

Nº de serviço na composição	50 th	100 th	150 th	200 th	250 th	300 th
2	19,06	38,49	52,97	75,32	92,35	108,48
4	18,20	38,40	55,39	77,38	93,32	113,56
6	22,14	38,64	57,22	73,64	93,08	113,35
8	20,21	39,61	55,22	77,51	93,87	120,03

O Gráfico 7.7 é construído com os valores apresentados na Tabela 7.3, sendo possível observar que o tempo para a execução dos diversos pedidos aumenta, mas de um forma bastante uniforme, indicando que este componente responde de forma gradual à carga imposta.

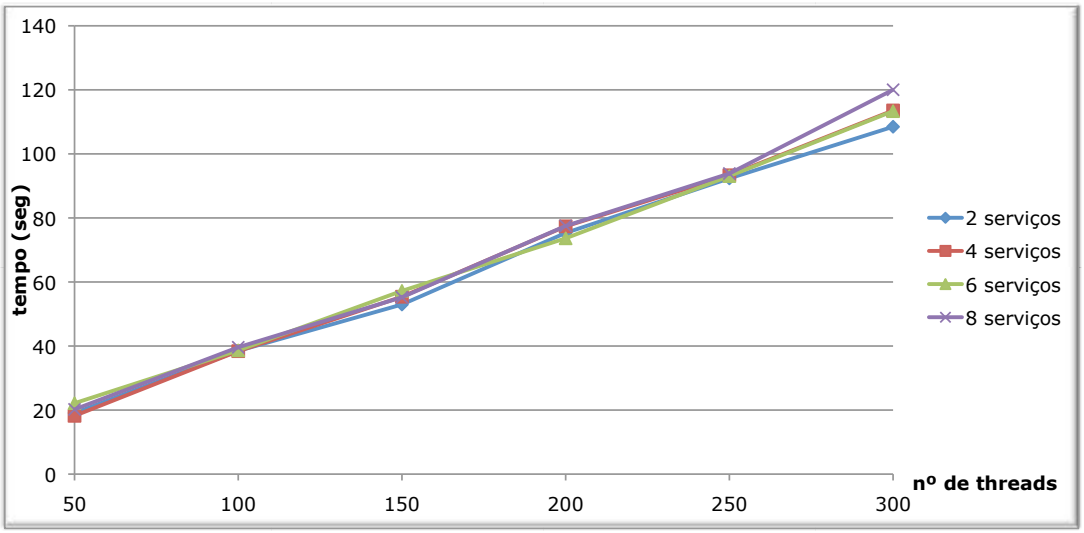


Gráfico 7.7 – Variação do tempo consumido pelas diversas tarefas (TCWS de 0.5 seg).

Gráfico 7.8 apresenta o tempo médio consumido para cada composição e execução de serviços pedida. É possível observar que, para a composição e execução de 8 serviços, o tempo consumido para as diversas cargas é de aproximadamente 5 segundos e que se mantém relativamente estável com o aumento do número de pedidos.

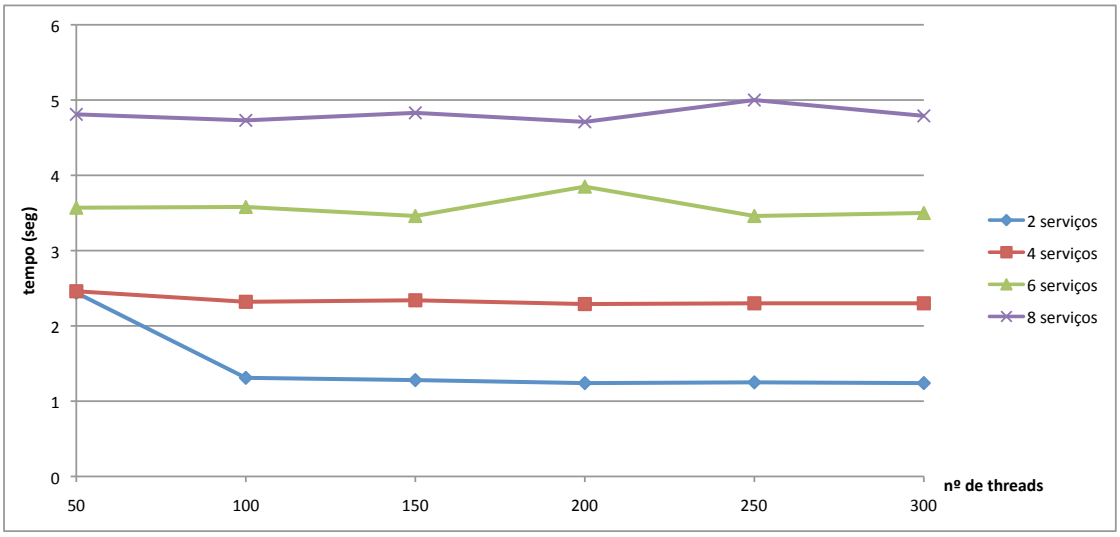


Gráfico 7.8 – Tempo médio para a composição e execução de serviços (TCWS de 0.5 seg).

Para se ter a noção do tempo que o *motor de composição* de serviços consome para compor um serviço, é necessário subtrair o tempo que cada serviço pertencente à composição despende para ser executado. Assim, o tempo consumido para compor um novo serviço é calculado da seguinte forma $T_{composição} = T_{composição\ e\ execução} - (N_{serviços\ na\ composição} \times TWCS)$. No Gráfico 7.9 é possível visualizar o tempo médio consumido pelo *motor de composição* para criar um novo serviço para cada pedido. No mesmo gráfico observa-se um pico inicial para a primeira carga de 50 threads que ordena uma composição de dois serviços, mas que depois estabiliza. Verificou-se que este pico ocorria em alguns testes quando eram feitos os primeiros pedidos por parte da aplicação cliente, pressupondo-se que a plataforma demora mais tempo para processar os novos pedidos. Este fenómeno poderá estar relacionado com tarefas de alocação de memória e utilização de mecanismos de cache.

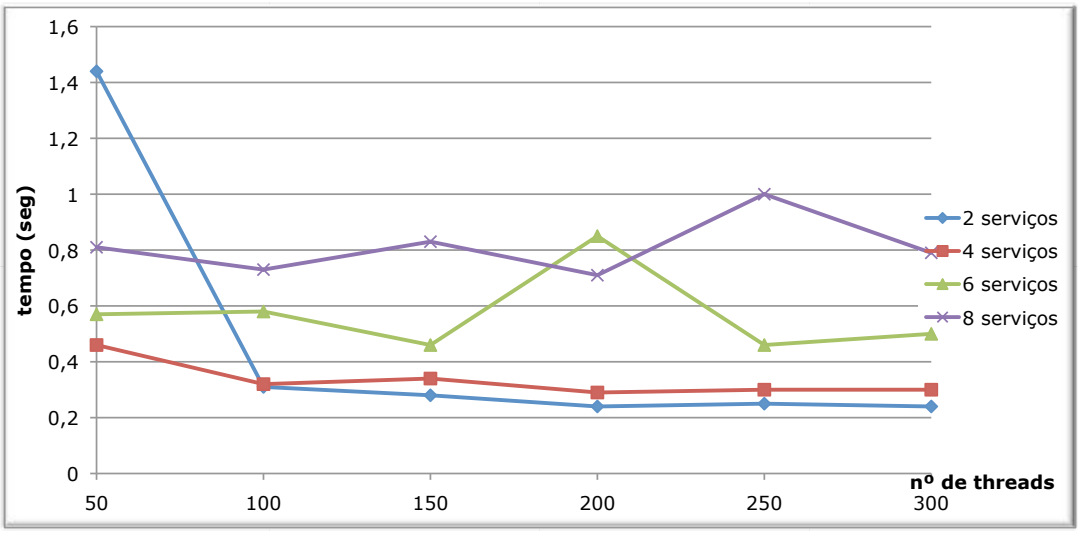


Gráfico 7.9 – Tempo médio para a composição de serviços (TCWS de 0.5 seg).

Durante os diversos testes ao *motor de composição*, o funcionamento global deste motor decorreu sem problemas, sendo capaz de executar todos os pedidos realizados pelo cliente. Na variação dos parâmetros número de pedidos, número de serviços e tempo de execução do serviço, observou-se que o motor se mantém estável executando as tarefas com tempos uniformes.

7.2 Resumo da análise do sistema

A observação do funcionamento global do protótipo implementado revelou-se satisfatória. No entanto ocorreram alguns bloqueios relativamente à verificação da consistência da base de conhecimento quanto esta continha um número elevado de triplas. Este problema resultou de um erro de alocação de memória que acontecia no motor de inferência *Pellet*, que foi identificado pela equipa de desenvolvimento da API. As versões mais recentes desta API viriam a corrigir este aspecto; no entanto, obrigava a alterações no código fonte de algumas classes e, como tal, essa nova versão não foi usada.

Neste capítulo fez-se uma análise qualitativa do desempenho dos principais componentes do sistema, onde se verificou de um modo geral um bom desempenho do sistema para cadências de utilização semelhantes às que serão esperadas no cenário de aplicação da plataforma *iCas* num campus universitário.

Capítulo 8

CONCLUSÕES E TRABALHO FUTURO

Finalizado o desenvolvimento e a análise do desempenho, é tempo de fazer uma apreciação global do trabalho realizado tendo em conta os objectivos estabelecidos. Neste capítulo recordam-se os objectivos inicialmente definidos, faz-se uma análise crítica das características do modelo proposto e dos resultados obtidos com o protótipo *iCas*, e apresentam-se os principais contributos deste trabalho para o desenvolvimento de sistemas que permitam a composição dinâmica e contextual de serviços, tendo como referência o modelo proposto.

8.1 Conclusões

A sensibilidade ao contexto tem sido vista como uma propriedade importante no futuro dos serviços móveis. No entanto, apenas algumas aplicações são sensíveis ao contexto e apresentam características limitadas. A composição de *Web Services* recebeu muito interesse da área das aplicações *business-to-business* ou aplicações empresariais, mas não tanto na área das aplicações *business-to-consumer*. O propósito principal desta dissertação foi especificar e testar um modelo que permita a criação de novos serviços de forma dinâmica e contextual para dispositivos móveis, em que o utilizador seja capaz de ter acesso a um conjunto de serviços móveis e em tempo real pudesse criar um novo serviço personalizado, através da utilização das suas preferências e do seu contexto actual, com mais funcionalidades, e o pudesse usar e partilhar quando pretendesse. O utilizador pode ainda receber informação sensível ao seu contexto actual que lhe é entregue de uma forma transparente, de modo a manter-se actualizado acerca dos seus temas favoritos. Por forma a testar o modelo proposto, desenvolveu-se uma arquitectura (*iCas*) que implementa as principais funcionalidades para a

criação de serviços dinâmica e sensível ao contexto num cenário de um campus universitário. Posteriormente implementou-se um protótipo da arquitectura proposta, o qual permitiu verificar se a solução era exequível e se o seu desempenho era satisfatório num ambiente de utilização semelhante ao de um campus universitário.

A arquitectura resultante do modelo proposto nesta dissertação apresenta as seguintes características:

- aberta: baseada em normas e recomendações abertas, maioritariamente propostas pelo W3C e pelo IETF;
- híbrida: embora forneça serviços aos clientes através do modelo cliente/servidor, tem propriedades distribuídas que fazem parte das arquitecturas orientadas aos serviços;
- modular: pelo facto dos diversos componentes estarem bem definidos é possível um desenvolvimento parcial e faseado da arquitectura;
- interoperável: porque oferece serviços ao cliente através de *Web Services* tornando-a independente da tecnologia de implementação dos clientes, e porque os serviços disponíveis para composição são também eles independentes da tecnologia de implementação;
- flexível: possui funcionalidades suficientemente genéricas para se adaptar a diversos tipos de aplicações.

Relativamente ao objectivos propostos no capítulo de introdução, cumpriram-se as seguintes partes:

- foi investigado com mais detalhe o conceito de contexto nos sistemas de computação, assim como as classes em que se dividem e diversos modelos contextuais existentes;
- foram estudados os aspectos mais importantes das arquitecturas orientadas aos serviços e os esquemas de composição de serviços;
- foi feita uma recolha de informação relativa aos principais sistemas de composição de serviços e *context-aware*, de modo a apreciar o seu estado evolutivo e identificar áreas de desenvolvimento de novos trabalhos;

- para a facilitar a compreensão da utilização da semântica na *Web*, foi estudada a *Semantic Web*. Para isso fez-se uma análise da sua arquitectura e das suas diversas camadas e de como é possível inferir novas informações através das suas descrições;
- fez-se a análise das tecnologias que serviram de suporte à implementação de um protótipo da arquitectura proposta e, como resultado desta análise, escolheram-se as tecnologias mais adequadas aos objectivos que se pretendiam atingir;
- propôs-se um modelo inovador que permite a composição de serviços dinamicamente e sensível ao contexto por parte do utilizador, através da composição de serviços existentes;
- implementou-se um protótipo da arquitectura *iCas*, não tendo sido implementadas todas as funcionalidades, mas apenas aquelas que se julgou serem relevantes para a validação do modelo, no cenário de estudo escolhido;
- realizaram-se testes ao funcionamento do protótipo desenvolvido, analisando o seu comportamento funcional e avaliando qualitativamente e quantitativamente o desempenho dos diversos componentes em termos de tempo de resposta aos pedidos efectuados pelos utilizadores.

Os objectivos definidos na capítulo da introdução foram, na generalidade, alcançados com sucesso, apresentando-se contribuições inovadoras para a área de conhecimento em que se integra o trabalho desenvolvido para esta tese, como será visto adiante.

O protótipo da arquitectura foi implementado de uma forma muito simplificada, dada a grande complexidade que comporta o desenvolvimento de uma aplicação deste tipo. No entanto, estas simplificações não impediram a validação do modelo de composição dinâmica de serviços sensível ao contexto. Foi ainda possível demonstrar que a construção da arquitectura utilizada neste modelo é viável funcionalmente e o desempenho exibido foi encorajador. A utilização do modelo cliente/servidor através de *Web Services* como tecnologia de suporte permitiu trazer para o lado do servidor o peso computacional da composição e inferência, entre outras tarefas, que seriam problemáticas ou até impossíveis de realizar em dispositivos móveis.

Nos testes elaborados foi possível observar que os principais componentes da arquitectura *iCas* mostraram capacidade para suportar cargas de invocação elevadas, sem deterioração

visível do desempenho das aplicações. Foi ainda possível observar que na maioria dos testes o protótipo desenvolvido tinha uma resposta linear com a carga aplicada.

Após reflexão sobre o trabalho desenvolvido foram identificados alguns problemas durante a elaboração deste trabalho:

- embora existam ferramentas que permitam a transformação de *Web Services* tradicionais em *Web Services Semânticos*, estas ferramentas apresentam limitações na conversão quando os *Web Services* tradicionais contêm tipos de dados complexos nos seus parâmetros. Nestes casos, existe a necessidade de recorrer a scripts de transformação que não são fáceis de escrever e são susceptíveis a erros;
- durante as fases de testes determinados sub-processos de inferência apresentaram sintomas de consumo de tempo preocupante. Embora para o cenário de aplicação escolhido para este trabalho estes sintomas não fossem inquietantes, poderá haver limitações em cenários de larga escala, como por exemplo, num cenário de aplicação do modelo numa organização de comunicações móveis;
- para extrair o máximo do potencial da utilização do modelo contextual semântico, é necessária uma descrição exaustiva da informação através de metadados, tratando-se de uma tarefa morosa e entediante; das diversas camadas que compõem a arquitectura da *Semantic Web*, muitas ainda não estão totalmente especificadas e as múltiplas tecnologias a elas associadas não as tornam uma solução simples, como acontece com as diversas arquitecturas que tiveram sucesso nas áreas dos sistemas distribuídos e Internet (ex. arquitecturas SOA, protocolos de Internet).

No âmbito da implementação da arquitectura *iCas* destacam-se os seguintes contributos mais importantes:

- a apresentação de um modelo que integra o conceito de modelo contextual semântico para auxiliar o utilizador na construção de serviços com características evolutivas. Este modelo apresenta ainda as características de permitir a composição de serviços em dispositivos móveis, adaptação dos resultados da execução de serviços ao terminal do utilizador, e um *motor de contexto* capaz de lidar com dados contextuais do utilizador de forma transparente, por forma a auxiliar em determinadas tarefas num cenário específico;
- um mecanismo de selecção de serviços, que oferece ao utilizador os serviços mais

adequados ao seu contexto actual e às suas preferências;

- um componente que oferece um conjunto de funcionalidades para gerir e interpretar informações de contexto apoiadas por um modelo contextual ontológico e semântico;
- a construção de um protótipo da arquitectura *iCas*, com características inovadoras, visto que não foram encontradas implementações com estas funcionalidades durante a revisão do estado da arte.

8.2 Trabalho futuro

O modelo proposto no âmbito do trabalho realizado nesta tese não pretende ser um produto final para a composição de serviços auxiliada pelo contexto do utilizador. Deve ser antes entendida como um proposta para uma solução satisfatória, por forma a resolver o problema em questão. Procura ainda abrir caminho para outros trabalhos e o suporte de um vasto número de iniciativas ao nível científico e académico.

Os diversos componentes que foram desenvolvidos não implementam a totalidade das funcionalidades consideradas pelo modelo proposto ou que poderão oferecer mais características, necessitando de trabalho futuro neste campo. Assim, de entre várias iniciativas ao nível do desenvolvimento da plataforma que podem ser lançadas, enumeram-se algumas:

- embora o *motor de adaptação* fosse especificado, a sua implementação oferece funcionalidade bastantes simples. Esta opção deveu-se ao facto de não se considerar um dos componentes primordiais para a validação do modelo proposto. Deste modo, poderá haver algum trabalho neste componente de modo a que se consiga alcançar um motor capaz de adaptar qualquer resultado de serviços ou conteúdos ao terminal, rede e contexto em que o utilizador se encontra;
- o *motor de composição* poderá ser também optimizado, por forma a suportar características de composição mais complexas. Por exemplo, o suporte de todos os construtores de controlo apresentados na secção 4.4.1.2;
- apesar do servidor de aplicações suportar a infra-estrutura de segurança dos *Web Services*, estes não foram incluídos na arquitectura *iCas*. No entanto, como esta arquitectura lida com informação pessoal, entre outras, será uma funcionalidade

bem-vinda;

- poderá ser realizado um estudo para se conseguir a melhor interface de composição de serviço num dispositivo móvel, através da implementação de diversas propostas que poderão ser sujeitas a uma avaliação de um conjunto de utilizadores;
- poderá ser realizado um estudo prévio sobre as necessidades reais dos utilizadores neste tipos de aplicações.

O modelo apresentado nesta tese defina funções básicas em relação à partilha de serviços. Neste modelo a partilha de um serviço é vista como a publicação desse serviço, de modo a que ele possa ser visto por qualquer utilizador. Neste sentido, o modelo poderá evoluir na forma de partilha de serviços, adicionado novas propriedades de partilha, como por exemplo: que tipo de utilização poderão fazer do serviço (ex.: execução, ou utilização em composição), quem poderá utilizar o serviço (ex.: utilizador, grupos, todos), entre outras.

Neste trabalho apenas se implementou uma única aplicação de composição dinâmica de serviços sensível ao contexto, sendo considerado um cenário de aplicação específico. Assim, outros cenários poderão ser testados por forma a obter uma validação mais completa do modelo. As próprias características das funcionalidades que o sistema deverá oferecer poderão ser também alvo de um estudo sobre quais as necessidades reais dos utilizadores para determinado cenário.

O modelo apresentado nesta tese não inclui a interacção com outras plataformas, de modo a permitir a troca de dados dos diversos domínios ontológicos e de serviços semânticos com sistemas externos. Esta capacidade de troca de dados com outras plataformas, seria outra das funcionalidades que poderiam vir a ser desenvolvidas no futuro.

Finalmente, a utilização desta plataforma num cenário de um *campus* universitário abre perspectivas a novos estudos, como, por exemplo, se a utilização das tecnologias móveis baseadas em contexto pode melhorar as condições pedagógicas e a interacção social-pedagógica dos diversos intervenientes (alunos, docentes, funcionários, visitantes, etc.) existentes num *campus* universitário. ■

REFERÊNCIAS BIBLIOGRÁFICAS

Abowd, G., A. Dey, et al. (1999). Towards a Better Understanding of Context and Context-Awareness. pp. 304-307. HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Springer-Verlag.

Abowd, G. D., E. D. Mynatt, et al. (2002). "The human experience." IEEE Pervasive Computing **1**(1): pp. 48-57. issn:1536-1268.

Adida, B., M. Birbeck, et al. (2008). "RDFa in XHTML: Syntax and Processing." W3C Recommendation, W3C. Retrieved 08/2009, from http://www.w3.org/TR/rdfa-syntax/.

Agents, T. F. f. I. P. (2002). FIPA Abstract Architecture Specification. sc00001l edition.

Akkiraju, R. (2003). "ETTK for Web Services." IBM. Retrieved 12/2009, from http://alphaworks.ibm.com/tech/ettk.

Akkiraju, R., J. Farrell, et al. (2005). "Web Service Semantics - WSDL-S." W3C. Retrieved 09/2009, from http://www.w3.org/Submission/WSDL-S/.

Akkiraju, R., J. Farrell, et al. (2005). "Web Service Semantics - WSDL-S." W3C Member Submission, W3C. Retrieved 09/2009, from http://www.w3.org/Submission/WSDL-S/.

Al-Feel, H. T., M. Koutb, et al. (2008). "Semantic Web on Scope: A New Architectural Model for the Semantic Web." Journal of Computer Science **4**(7): pp. 613-624. issn:1549-3636.

Alesso, H. P. a. S., Craig F. (2006). Thinking on the Web: Berners-Lee, Godel and Turing, iley-Interscience. isbn:0471768146.

Alonso, G., F. Casati, et al. (2003). Web Services - Concepts, Architectures and Applications, Springer. isbn:3540440089.

Andrews, T., F. Curbera, et al. (2003). Business Process Execution Language for Web Services - Version 1.1. isbn:ISBN.

Antoniou, G. and F. vanHarmelen (2004). A Semantic Web Primer, MIT Press. isbn:0262012103.

Apache. (2009). "Apache Geronimo." Retrieved 12/2009, from http://geronimo.apache.org/.

Asuncion, G.-P., C.-G. Oscar, et al. (2003). Ontological Engineering, Springer-Verlag New York, Inc. isbn:978-1-85233-551-9.

Battle, S., A. Bernstein, et al. (2005, 05/2005). "Semantic Web Services Framework (SWSF) Overview - version 1.0." W3C. Retrieved 09/2009, from http://www.daml.org/services/swsf/1.0/overview/.

Battle, S., A. Bernstein, et al. (2005). "Semantic Web Services Language (SWSL)." W3C

Member Submission, W3C. Retrieved 09/2009, from http://www.w3.org/Submission/2005/SUBM-SWSF-SWSL-20050909/.

Battle, S., A. Bernstein, et al. (2005, May). "Semantic Web Services Ontology (SWSO)." DAML.org. from http://www.daml.org/services/swsf/1.0/swso/.

Bechhofe, S. (2003). The DIG Description Logic Interface: DIG/1.1. pp. Proceedings of DL2003 Workshop, Rome.

Bechhofer, S., F. v. Harmelen, et al. (2004). "OWL Web Ontology Language - Reference." W3C Recommendation, W3C. Retrieved 08/2009, from http://www.w3.org/TR/owl-ref/.

Beckett, D. (2001). The Design and Implementation of the Redland RDF Library. pp. Tenth International World Wide Web Conference, Hong Kong isbn:1-58113-348-0/01/0005.

Beckett, D. and J. Broekstra. (2008). "SPARQL Query Results XML " W3C Recommendation W3C. Retrieved 09/2009, from http://www.w3.org/TR/rdf-sparql-XMLres/.

Beckett, D. and B. McBride. (2004). "RDF/XML Syntax Specification (Revised)." W3C Recommendation W3C. Retrieved 08/2009, from http://www.w3.org/TR/rdf-syntax-grammar/.

Berners-Lee, T. (1997). "Metadata architecture." Retrieved 11/2008, from http://www.w3.org/DesignIssues/Metadata.

Berners-Lee, T. (1998). "Notation 3." W3C. Retrieved 08/2009, from http://www.w3.org/DesignIssues/Notation3.

Berners-Lee, T. (2000, Last Update). "Sematic Web on XML." Access 2000, from http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html.

Berners-Lee, T. (2003, Last Update). "The Semantic Web and Challenges." Access 2003, from http://www.w3.org/2003/Talks/01-sweb-tbl/Overview.html.

Berners-Lee, T. (2006, Last Update). "Artificial Intelligence and the Semantic Web." Access 2006, from http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html#(1).

Berners-Lee, T., D. Connolly, et al. (2007). "W3C Semantic Web Activity." from http://www.w3.org/2001/sw/

Berners-Lee, T., J. Hendler, et al. (2002) "The Semantic Web." Retrieved Access Date, Access 2002, from http://www.scientificamerican.com/article.cfm?id=the-semantic-web.

Bizer, C. and D. Westphal. (2007). "Developers Guide to Semantic Web Toolkits for different Programming Languages." Retrieved 01/2010, from http://www4.wiwiss.fu-berlin.de/bizer/toolkits/.

Bock, J., P. Haase, et al. (2008). "Benchmarking OWL Reasoners." ARea2008 - Workshop on Advancing Reasoning on the Web: Scalability and Commonsense: pp.

Bolchini, C., C. A. Curino, et al. (2007). "A data-oriented survey of context models." SIGMOD Rec. **36**(4): pp. 19-26. issn:0163-5808.

Boley, H., G. Hallmark, et al. (2009). "RIF Core Dialect." W3C Candidate Recommendation,

W3C. Retrieved 08/2009, from http://www.w3.org/TR/rif-core/.

Boley, H. and M. Kifer. (2009). "RIF Basic Logic Dialect." W3C Candidate Recommenda, W3C. Retrieved 08/2009, from http://www.w3.org/TR/rif-bld/.

Booth, D., H. Haas, et al. (2004). "Web Services Architecture." W3C Working Group Note, W3C. from http://www.w3.org/TR/ws-arch/.

Boughanem, M. and M. Tmar (2002). Incremental adaptive filtering: profile learning and threshold calibration. pp. 640-644. SAC '02: Proceedings of the 2002 ACM symposium on Applied computing, Madrid, Spain, ACM,isbn:1-58113-445-2.

Bozsak, E., M. Ehrig, et al. (2002). KAON - Towards a large scale Semantic Web. pp. 304-313. Proceedings of the Third International Conference on E-Commerce and Web Technologies, Aix-en-Provence, France,isbn:3-540-44137-9.

Brand, A., F. Daly, et al. (2003). Metadata Demystified: A Guide for Publishers. NISO Press and The Sheridian Press. isbn:ISBN.

Bray, T., D. Hollander, et al. (2006). Namespaces in {XML} 1.0 (Second Edition). isbn:ISBN.

Bray, T., D. Hollander, et al. (2006). "XML Schema." from http://www.w3.org/TR/REC-xml-names.

Bray, T., J. Paoli, et al. (1998). Extensible markup language (XML) 1.0. isbn:ISBN.

Bray, T., J. Paoli, et al. (2006) "Extensible Markup Language (XML) 1.1 (Second Edition)." Retrieved Access Date, Access 2006, from http://www.w3.org/TR/2006/REC-xml11-20060816/.

Brickley, D. and R. V. Guha. (2004). "RDF Vocabulary Description Language 1.0: RDF Schema." W3C Recommendation, W3C. Retrieved 06/2009, from http://www.w3.org/TR/rdf-schema/.

Brickley, D. and R. V. Guha. (2004). "RDF Vocabulary Description Language 1.0: RDF Schema." Retrieved 08/2009, from http://www.w3.org/TR/rdf-schema/.

Brown, P. J. (1996). The Stick-e Document: a Framework for Creating Context-aware Applications. pp. 259-272. Proceedings of EP'96, Palo Alto, also.

Bruijn, J. d., C. Bussler, et al. (2005). "Web Service Modeling Ontology (WSMO)." W3C Member Submission 3 June 2005, W3C. from http://www.w3.org/Submission/WSMO/.

Bruijn, J. d., D. Fensel, et al. (2005). "Web Service Modeling Language (WSML)." W3C Member Submission, W3C. Retrieved 09/2009, from http://www.w3.org/Submission/WSML/.

Brussee, R. and S. Pokraev (2007). Reasoning in the Semantic Web Telematica Instituut

Bulcão Neto, R. F. (2006). A software process for ontology-based context-aware computing, ICMC-USP

Bulcão Neto, R. F. and M. G. C. Pimentel (2005). Toward a Domain-Independent Semantic Model for Context-Aware Computing. pp. 61-70. 3rd Latin American Web Congress (LA-Web'05), Argentina, IEEE Computer Society,isbn:0-7695-2471-0.

Bulcão Neto, R. F. and M. G. C. Pimentel (2006). Performance evaluation of inference services for ubiquitous computing. pp. 27-34. XII Braziliam Symposium on Multimedia and Web Systems, Brazil,isbn:85-7669-100-0.

Bultan, T., X. Fu, et al. (2003). Conversation specification: a new approach to design and analysis of e-service composition. pp. 403-410. WWW '03: Proceedings of the 12th international conference on World Wide Web, ACM.

Bussler, C., E. Cimpian, et al. (2005). "Web Service Execution Environment (WSMX)." W3C Member Submission, Retrieved 09/2009, from http://www.w3.org/Submission/WSMX/.

Carroll, J. J., I. Dickinson, et al. (2004). Jena: implementing the semantic web recommendations. pp. WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters, New York, NY, USA, ACM,isbn:1-58113-912-8.

Carvalho, J. F. (2007). Localização de Dispositivos Móveis em Redes Wi-Fi. Vila Real, Universidade de Trás-os-Montes e Alto Douro. **Mestrado:** 54-69

Chakraborty, D. and A. Joshi (2001). Dynamic service composition: State-of-the-art and research directions

Chakraborty, D., A. Joshi, et al. (2005). "Service Composition for Mobile Environments." Mobile Networks and Applications **10**(4): pp. 435-451. issn:1383-469X.

Chen, G. and D. Kotz (2000). A Survey of Context-Aware Mobile Computing Research. isbn:ISBN.

Chen, H. (2004). An Intelligent Broker Architecture for Pervasive Context-Aware Systems, University of Maryland, Baltimore County

Chen, H., T. W. Finin, et al. (2004). "Intelligent Agents Meet the Semantic Web in Smart Spaces." IEEE Internet Computing **8**(6): pp. 69-79

Chen, H., F. Perich, et al. (2004). SOUPA: standard ontology for ubiquitous and pervasive applications. pp. 258-267. Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on.

Chinnici, R., J.-J. Moreau, et al. (2007). "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language." W3C Recommendation, W3C. Retrieved 09/2009, from http://www.w3.org/TR/wsdl20/.

Christensen, E., F. Curbera, et al. (2001). Web Services Description Language ({WSDL}) 1.1, World Wide Web Consortium (W3C)

Clark, J. (1999). "XSL Transformations (XSLT) - Version 1.0." W3C Recommendation, W3C. Retrieved 12/2008, from http://www.w3.org/TR/xslt.

Connolly, D. (2007). "Gleaning Resource Descriptions from Dialects of Languages (GRDDL)." W3C. Retrieved 08/2009, from http://www.w3.org/TR/grddl/.

Connolly, D., F. v. Harmelen, et al. (2001). "DAML+OIL Reference Description." W3C. Retrieved 08/2009, from http://www.w3.org/TR/2003/PR-owl-ref-20031215/.

DAML. (2000). "DARPA Agent Markup Language - (DAML)." Retrieved 09/2009, from http://www.daml.org/.

DAML. (2005). "DAML Services - Tools." DAML. Retrieved 12/2009, from http://www.daml.org/services/owl-s/tools.html.

David Martin, M. B., Jerry Hobbs, Ora Lassila, Drew Mcdermott, Sheila Mcilraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara (2004) "OWL-S: Semantic Markup for Web Services." Retrieved Access Date, Access 2004, from http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.

DCMI. (2006). "Dublin Core Metadata Initiative (DCMI)." Retrieved 08/2009, from http://dublincore.org/.

Dey, A. (2000). "The Context Toolkit." Retrieved 09/2009, from http://www.cc.gatech.edu/fce/contexttoolkit/.

Dey, A. (2000). Providing architectural support for building context-aware applications

Dey, A. K., D. Salber, et al. (1999). The Conference Assistant: Combining Context-Awareness with Wearable Computing. Proceedings of the 3rd IEEE International Symposium on Wearable Computers, IEEE Computer Society**:** 21. isbn/issn:0-7695-0428-0

Eclipse. (2009). "Eclipse IDE." Retrieved 12/2009, from http://www.eclipse.org/.

Elenius, D., G. Denker, et al. (2005). "The owl-s editor - a development tool for semantic Web services." Second European Semantic Web Conference: pp. issn:0302-9743.

ESSI. (2004). "European Semantic System Initiative." Retrieved 09/2009, from http://www.essi-cluster.org/.

Fitting, M. (1996). First-order logic and automated theorem proving (2nd ed.), Springer-Verlag New York, Inc. isbn:0-387-94593-8.

Fonseca, B. (2005). Um Modelo para a Criaç ão de Serviç os Cooperativos. Vila Real, Universidade de Trás-os-Montes e Alto Douro. **PhD:** 228

Foundation, F. S. (2007, 09/2009). "GNU AFFERO GENERAL PUBLIC LICENSE." Retrieved 09/2009, from http://www.gnu.org/licenses/agpl-3.0.html.

Foundation, F. S. (2007). "GNU LESSER GENERAL PUBLIC LICENSE - Version 3." Retrieved 09/2009, from http://www.gnu.org/copyleft/lesser.txt.

FreeBSD. (1992-2010). "The FreeBSD Copyright." Retrieved 09/2009, from http://www.freebsd.org/copyright/freebsd-license.html.

Fujii, K. and T. Suda (2004). Component Service Model with Semantics (CoSMoS): A New Component Model for Dynamic Service Composition. pp. SAINT-W '04: Proceedings of the 2004 Symposium on Applications and the Internet-Workshops (SAINT 2004 Workshops), IEEE.

Gardiner, T., I. Horrocks, et al. (2006). Automated Benchmarking of Description Logic Reasoners. pp. Description Logics, CEUR-WS.org.

Gearon, P., D. Wood, et al. (2006). "Mulgara Semantic Store." Retrieved 09/2009, from http://docs.mulgara.org/overview/legal.html.

Gerber, u., A. V. d. Merwe, et al. (2008). A Functional Semantic Web Architecture. pp. 273-

287. 5th European Semantic Web Conference (ESWC2008), Tenerife, Spain,isbn:978-3-540-68233-2.

Glassfish. (2009). "GlassFish Application Server ", Retrieved 12/2009, from https://glassfish.dev.java.net/.

GNU. (2007). "GNU Affero General Public License." Retrieved 09/2009, from http://www.gnu.org/licenses/agpl-3.0.html.

Google. (2008). "The Google Geocoding Web Service." Google Maps API Web Services, Google. Retrieved 03/2008, 2008, from http://code.google.com/apis/maps/documentation/geocoding/.

Grant, J. and D. Beckett. (2004). "RDF Test Cases." W3C Recommendation, W3C. Retrieved 09/2009, from http://www.w3.org/TR/rdf-testcases/.

Group, P. G. D. (2004). "PostgreSQL JDBC Driver." PostgreSQL Global Development Group. Retrieved 12/2009, from http://jdbc.postgresql.org/.

Gruber, T. R. (1993). "A translation approach to portable ontology specifications." Knowl. Acquis. **5**(2): pp. 199-220. issn:1042-8143.

Gruber, T. R. (1995). "Toward principles for the design of ontologies used for knowledge sharing." Int. J. Hum.-Comput. Stud. **43**(5-6): pp. 907-928. issn:1071-5819.

Gu, T., H. Pung, et al. (2005). "A service-oriented middleware for building context-aware services." Journal of Network and Computer Applications **28**(1): pp. 1-18. issn:1084-8045.

Gu, T., H. K. Pung, et al. (2004). "Toward an OSGi-Based Infrastructure for Context-Aware Applications." IEEE Pervasive Computing **3**(4): pp. 66-74. issn:1536-1268.

Guarino, N. (1997). Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. pp. 139 - 170 International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, Springer-Verlag,isbn:3-540-63438-X.

Guarino, N. (1998). Formal Ontology and Information Systems. Formal Ontology in Information Systems. pp. 3-15. Proc. Of the 1st International Conference, Trento, Italy, IOS Press.

Gudgin, M., M. Hadley, et al. (2003). "{SOAP} Version {1.2} Part 1: Messaging Framework. ." W3C Recommendation, W3C. Retrieved 07/2008, from http://www.w3.org/TR/soap12-part1/.

Haarslev, V. and R. Möller (2001). RACER System Description. pp. 701-705. International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy, Springer-Verlag,isbn:3-540-42254-4.

Haas, H. and A. Brown. (2004). "Web Services Glossary - Service-Oriented Architecture." from http://www.w3.org/TR/ws-gloss/

Haase, P., J. Broekstra, et al. (2004). A comparison of RDF query languages. pp. Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan,isbn:978-3-540-23798-3.

Hamadi, R. and B. Benatallah (2003). A Petri net-based model for web service composition.

pp. 191-200. ADC '03: Proceedings of the fourteenth Australasian database conference, Australian.

Harris, S. (2006). "4store." Retrieved 09/2009, from http://4store.org/.

Hashimi, S. (2003) "Service-Oriented Architecture Explained." O'Reilly Network. Retrieved Access Date, Access 2003, from http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html.

Hat, R. (2009). "Relational Persistence for Java and .NET." Red Hat. Retrieved 12/2009, from https://www.hibernate.org/.

Hawke, S. (2006). "Rule Interchange Format, Working Group Charter." W3C. Retrieved 06/2009, from http://www.w3.org/2005/rules/wg/charter.

Hendler, J. and D. L. McGuinness (2000). "The DARPA Agent Markup Language." IEEE Intelligent Systems **15**(6): pp. 67-73

Hewlett-Packard. (2009). "Jena 2 Inference support." Retrieved 12/2009, from http://jena.sourceforge.net/inference/.

Hewlett-Packard. (2009). "Jena – A Semantic Web Framework for Java (2.6.0)." Retrieved 08/2009, from http://jena.sourceforge.net/.

Hoecke, S. V., K. Steurbaut, et al. (2009). Automatic WS-BPEL Composition of Medical Support Services in the ICU. Proceedings of the 2009 International Conference on eHealth, Telemedicine, and Social Medicine, IEEE Computer Society**:** 251-256. isbn/issn:978-0-7695-3532-6

Hori, M., J. Euzenat, et al. (2003). "OWL Web Ontology Language - XML Presentation Syntax." W3C. Retrieved 08/2009, from http://www.w3.org/TR/owl-xmlsyntax/.

Horrocks, I. (2002, Last Update 08/2002). "Reasoning with Expressive Description Logics: Logical Foundations for the Semantic Web.". Access 2002, from http://www.cs.manchester.ac.uk/~horrocks/Slides/ed02.pdf.

Horrocks, I., P. F. Patel-Schneider, et al. (2004). "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." Retrieved 08/2009, from http://www.w3.org/Submission/SWRL/.

Hull, R., M. Benedikt, et al. (2003). "E-Services: A Look behind the Curtain."

Hustadt, U., B. Motik, et al. (2009). "KAON2." Retrieved 09/2009, from http://kaon2.semanticweb.org.

IBM. (2009). "WebSphere Application Server Community Edition." IBM. Retrieved 12/2009, from http://www-01.ibm.com/software/webservers/appserv/community/.

IETF (1994). Uniform Resource Locators (URL). IETF. from http://www.ietf.org/rfc/rfc1738.txt.

IETF (1997). URN Syntax. IETF. from http://www.ietf.org/rfc/rfc2141.txt.

IETF (1998). Internet Calendaring and Scheduling Core Object Specification - (iCalendar). IETF. from http://www.ietf.org/rfc/rfc2445.txt.

IETF (1998). Uniform Resource Identifiers (URI): Generic Syntax. from http://www.ietf.org/rfc/rfc2396.txt.

IETF (1998). vCard MIME Directory Profile. IETF. from http://www.ietf.org/rfc/rfc2426.txt.

IETF (2008). Dublin Core Metadata Element Set, Version 1.1. IETF. from http://dublincore.org/documents/dces/#RFC5013.

Iqbal, A. A., M. Ott, et al. (2009). Semantic information retrieval in a distributed environment. Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference. Las Vegas, NV, USA, IEEE Press**:** 786-790. isbn/issn:978-1-4244-2308-8

J.Scicluna, C.Abela, et al. (2004). Visual Modelling of OWL-S Services,. pp. ADIS International Conference WWW/Internet, Spain, IADIS.

JBoss. (2009). "JBoss Enterprise Application Platform." Retrieved 12/2009, from http://www.jboss.com/products/platforms/application/.

Jena. (2009). "Jena Ontology API." Retrieved 12/2009, from http://jena.sourceforge.net/ontology/.

Jena. (2009). "SDB - A SPARQL Database for Jena." Retrieved 09/2009, from http://jena.sourceforge.net/SDB/.

Johnson, D. (2006). "Project Blogapps." Retrieved 09/2008, from https://blogapps.dev.java.net/.

Kalyanpur, A., B. Parsia, et al. (2005). "Debugging Unsatisfiable Classes in OWL Ontologies." Journal of Web Semantics **4**(2): pp. 268-293. issn:1570-8268.

Karvounarakis, G., S. Alexaki, et al. (2002). RQL: a declarative query language for RDF. pp. 592 - 603. Proceedings of the 11th international conference on World Wide Web, Honolulu, Hawaii, USA, ACM,isbn:1-58113-449-5.

Kavantzas, N., D. Burdett, et al. (2005). "Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation 9 November 2005." Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation 9 November 2005, Retrieved 10/2008, from http://www.w3.org/TR/ws-cdl-10/.

Kolovski, V., B. Parsia, et al. (2006). Extending the SHOIQ(D) Tableaux with DL-safe Rules: First Results. Description Logics, CEUR-WS.org

Korkea-Aho, M. (2000). Context-Aware Applications Survey. isbn:ISBN.

Kowari. (2004). "Kowari." Retrieved 09/2009, from http://sourceforge.net/projects/kowari/.

Lassila, O. and R. R. Swick. (1999). "Resource Description Framework (RDF) Model and Syntax Specification." Retrieved 08/2009, from http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

Leymann, F. (2001). Web Services Flow Language (WSFL) Version 1.0 IBM. isbn:ISBN.

Lieberman, H. and T. Selker (2000). "Out of context: computer systems that adapt to, and learn from, context." IBM Syst. J. **39**(3-4): pp. 617-632. issn:0018-8670.

Lopes, C. J. and S. C. Ramalho (2005). Web Services - Aplicações Distribuídas sobre Protocolos Internet, FCA. isbn:972-722-421-0.

M. Kifer, G. Lausen, et al. (1995). "Logical Foundations of Object-Oriented and Frame-Based Languages." Journal of the ACM **42**: pp. 741-843. issn:0004-5411.

Mahmoud, Q. H. (2005) "Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)." Retrieved Access Date, Access 2005, from sun.com

http://java.sun.com/developer/technicalArticles/WebServices/soa/.

Manola, F. and E. Miller. (2004). "RDF Primer." W3C. Retrieved 06/2009, from http://www.w3.org/TR/REC-rdf-syntax/.

Marie, C. d. S., A. Paschke, et al. (2009). "RIF Production Rule Dialect." W3C. Retrieved 08/2009, from http://www.w3.org/TR/rif-prd/.

Marinacci, J. (2007). "Mapping Mashups with the JXMapViewer." Retrieved 09/2008, from http://today.java.net/pub/a/today/2007/11/13/mapping-mashups-with-jxmapviewer.html.

Martin, D., M. Burstein, et al. (2004). "OWL-S: Semantic Markup for Web Services." W3C Member Submission, W3C. Retrieved 09/2009, from http://www.w3.org/Submission/OWL-S/.

Martin, D., M. Burstein, et al. (2004). "OWL-S: Semantic Markup for Web Services." pp.

Martins, P. N., E. M. Carrapatoso, et al. (2005). O Ensino a Distância em Ambientes que suportam Mobilidade"; Conferência IADIS Ibero-Americana pp. 27-34. Conferência IADIS Ibero-Americana – WWW/Internet 2005, Lisbon, Portugal,isbn:972-8924-03-8.

McGuinness, D. L. and F. v. Harmelen. (2004). "OWL Web Ontology Language Overview." W3C Member Submission. Retrieved 08/2009, from http://www.w3.org/TR/owl-features/.

Mcilraith, S. A., T. C. Son, et al. (2001). "Semantic Web Services " IEEE Intelligent Systems **16**: pp. 46-53. issn:1541-1672.

METEOR-S (2003). METEOR-S: Semantic Web Services and Processes. Large scale distributed informa- tion systems (LSDIS) lab - University of Georgia. isbn:ISBN.

Microsoft. (1997). "DCOM Specification." Microsoft. Retrieved 09/2009, from http://msdn.microsoft.com/en-us/library/ms809311.aspx.

Milanovic, N. and M. Malek (2004). "Current Solutions for Web Service Composition." IEEE Internet Computing **8**(6): pp. 51-59

Miller, E. (2003, Last Update 20/01/2003). "Weaving Meaning : An Overview of The Semantic Web." Access 2003, from http://www.w3.org/2004/Talks/0120-semweb-umich/slide1-0.html.

Milner, R., F. L. Bauer, et al. (1993). The polyadic pi-calculus: a tutorial. Logic and Algebra of Specification, Springer-Verlag**:** pp. 203-246

Mitaim, S. and B. Kosko (1998). "Neural Fuzzy Agents for Profile Learning and Adaptive Object Matching." Presence **7**(6): pp. 617-637. issn:1054-7460.

Möller, T. (2009). "OWL-S API." University of Basel. Retrieved 12/2009, from http://on.cs.unibas.ch/owls-api/index.html.

Motik, B., B. C. Grau, et al. (2009). "OWL 2 Web Ontology Language Profiles." W3C

Candidate Recommendation 11 June 2009, W3C. Retrieved 08/2009, from http://www.w3.org/TR/owl2-profiles/.

Motik, B., B. C. Grau, et al. (2009). "OWL 2 Web Ontology Language Profiles." W3C Proposed Recommendation, W3C. Retrieved 09/2009, from http://www.w3.org/TR/owl2-profiles/.

Mozilla. (2009). "http://www.mozilla.org/MPL/MPL-1.1.html." Retrieved 09/2009, 2009, from http://www.mozilla.org/MPL/MPL-1.1.html.

Musen, M. A. (1988). Generation of Model-Based Knowledge Acquisition Tools for Clinical Tiral Advice Systems, Stanford University

NAICS. (1992). "North American Industry Classification System - NAICS." Retrieved 09/2009, from http://www.census.gov/eos/www/naics/.

Nan, L., Y. Junwei, et al. (2006). Towards Context-Aware Composition of Web Services. pp. 494–499. Fifth International Conference on Grid and Cooperative Computing, Washington, DC, USA, IEEE Computer Society,isbn:0-7695-2694-2.

Netbeans. (2009). "NetBeans IDE ", Retrieved 12/2009, from http://netbeans.org/.

Neto, R. d. F. B. and M. d. G. C. Pimentel (2005). Toward a Domain-Independent Semantic Model for Context-Aware Computing. Proceedings of the Third Latin American Web Congress, IEEE Computer Society

Nielsen, J. and R. Molich (1990). Heuristic evaluation of user interfaces. Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people. Seattle, Washington, United States, ACM**:** 249-256. isbn/issn:0-201-50932-6

Oasis (2005). "UDDI v3.0 Ratified as OASIS Standard." pp.

OASIS (2007). Web Services Business Process Execution Language Version 2.0. Organization for the Advancement of Structured Information Standards from Organization for the Advancement of Structured Information Standards

Oasis (2008) "Web Services Business Process Execution Language (WSBPEL)." Retrieved Access Date, Access 2008, from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

OGC (2008). OpenGIS® KML Encoding Standard (OGC KML). Open Geospatial Consortium, Inc. (OGC). from http://www.opengeospatial.org/standards/kml/.

Oliveira, J., R. Roque, et al. (2001). Creation of 3rd Generation Services in the Context of Virtual Home Environment. pp. 27-36. ICN '01: Proceedings of the First International Conference on Networking-Part 1, London, UK, Springer-Verlag,isbn:3-540-42302-8.

OMG. (1991). "The Common Object Request Broker: Architecture and Specification.", OMG. Retrieved 09/2009.

OMG (2005). XML Metadata Interchange. OMG. from http://www.omg.org/technology/documents/formal/xmi.htm.

OpenLink-Software. (1998). "OpenLink Virtuoso." Retrieved 09/2009, from http://virtuoso.openlinksw.com/main/featurematrix/index.htm.

OW2. (2009). "JOnAS - Java Open Application Server." Retrieved 12/2009, from

http://wiki.jonas.ow2.org/xwiki/bin/view/Main/WebHome.

Pan, Z. (2005). Benchmarking DL Reasoners Using Realistic Ontologies. pp. Proceedings of the OWL: Experiences and Directions Workshop. Galway.

Panagiotakis, S. and A. Alonistioti (2006). "Context-Aware Composition of Mobile Services." IT Professional **08**(4): pp. 38-43

Paolucci, M., T. Kawamura, et al. (2002). Semantic Matching of Web Services Capabilities. pp. 333-347. ISWC 2002: Proceedings of the First International Semantic Web Conference, Sardinia, Italy, Berlin, Springer,isbn:978-3-540-43760-4.

Paolucci, M., N. Srinivasan, et al. (2003). Towards a Semantic Choreography of Web Services: From WSDL to DAML-S. pp. 22-26. In Proceedings of the First International Conference on Web Services (ICWS'03), Las Vegas, Nevada, USA.

Paolucci, M., N. Srinivasan, et al. (2005). "WSDL2OWL-S 1.1 ", Retrieved 09/2009, from http://wsdl2owl-s.projects.semwebcentral.org/details.html.

Parsia, B. and E. Sirin (2004). Pellet: An OWL DL Reasoner. pp. In 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan.

Pascoe, J. (1998). Adding Generic Contextual Capabilities to Wearable Computers. pp. ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers, IEEE.

Pascoe, J. (1998). Adding Generic Contextual Capabilities to Wearable Computers. pp. 92. ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers, Washington, DC, USA, IEEE,isbn:0-8186-9074-7.

Patel-Schneider, P. F., P. Hayes, et al. (2004). "OWL Web Ontology Language - Semantics and Abstract Syntax." W3C. Retrieved 08/2009.

Peterson, D., Shudi, et al. (2008). "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes." W3C. from http://www.w3.org/TR/2008/WD-xmlschema11-2-20080620/.

Prud'hommeaux, E. and A. Seaborne. (2005). "SPARQL Query Language for RDF (W3C Working Draft 23 November 2005)." Retrieved 08/2009, from http://www.w3.org/TR/2005/WD-rdf-sparql-query-20051123/.

Prud'hommeaux, E. and A. Seaborne. (2008). "SPARQL Query Language for RDF." W3C Recommendation, W3C. Retrieved 06/2009, from http://www.w3.org/TR/rdf-sparql-query/.

Rao, J. and X. Su (2004). A Survey of Automated Web Service Composition Methods. pp. 43-54. SWSWPC, Springer,isbn:3-540-24328-3.

Reagle, J. (2008) "XML Encryption." Retrieved Access Date, Access 2008, from http://www.w3.org/Encryption/2001/.

RMI. (2009). "The Rule Markup Initiative." Retrieved 08/2009, 2009, from http://ruleml.org/.

Ryan, N. S., J. Pascoe, et al. (1998). Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. pp. Computer Applications in Archaeology, Tempus.

Saadati, S. and G. Denker (2005). An OWL-S Editor Tutorial Version 1.1

Schilit, B., N. Adams, et al. (1994). Context-aware computing applications. pp. 85-90. Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on.

Schilit, B. and R. Want. (1995). "The Xerox PARCTAB." Xerox. from http://sandbox.xerox.com/parctab/.

Schilit, B. N., N. Adams, et al. (1993). The ParcTab Mobile Computing System. pp. 34-39. Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV).

Schilit, W. N. (1995). A System Architecture for Context-Aware Mobile Computing, COLUMBIA UNIVERSITY. **Doctor of Philosophy:** 153

Schmidt, A., M. Beigl, et al. (1999). "There is more to context than location." Computers & Graphics **23**(6): pp. 893-901

Seaborne, A. (2004). "RDQL - A Query Language for RDF." W3C Member Submission, Retrieved 10/2009, from http://www.w3.org/Submission/RDQL/.

Services, C.-d. F. a. S. o. S. W. (2004). Composition-driven Filtering and Selection of Semantic Web Services. pp. AAAI Spring Symposium on Semantic Web Services.

Sheshagir, M., N. Sade, et al. (2004). Using Semantic Web Services for Context-Aware Mobile Applications. pp. MobiSys 2004 Workshop on Context Awareness, Boston.

Sheth, A. (2003, Last Update 20/05/2003). "Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery,Composition and Execution." Access 2003, from http://lsdis.cs.uga.edu/lib/presentations/WWW2003-ESSW-invitedTalk-Sheth.pdf.

Sheth, A., K. Verma, et al. (2006). "Semantics to energize the full services spectrum." Commun. ACM **49**(7): pp. 55-61. issn:0001-0782.

Sirin, E. (2004). "OWL-S API." Mindswap. Retrieved 12/2009, from http://www.mindswap.org/2004/owl-s/api/index.shtml.

Sirin, E. (2004). "OWL-S Validator." Retrieved 09/2009, from http://www.mindswap.org/2004/owl-s/validator/.

Sirin, E. and B. Parsia (2007). SPARQL-DL: SPARQL Query for OWL-DL. pp. 3rd OWL Experiences and Directions Workshop Innsbruck, Austria.

Sirin, E. and B. Parsia (2007). SPARQL-DL: SPARQL Query for OWL-DL. pp. OWLED, CEUR-WS.org.

Sirin, E., B. Parsia, et al. (2004). "Filtering and Selecting Semantic Web services with Interactive Composition techniques." Intelligent Systems **19**(4): pp. 42-49. issn:1541-1672.

Sousa, J. o. P., E. Carrapatoso, et al. (2009). "Composition of context aware mobile services using a semantic context model." International Journal on Advances in Software, issn 1942-2628 **2**(2&3): pp. 275 - 287. issn:1942-2628.

Sousa, J. P., E. Carrapatoso, et al. (2009). A Service-Oriented Middleware for Composing Context Aware Mobile Services. pp. 357-362. Internet and Web Applications and Services, International Conference on, Venice, Italy, IEEE Computer Society,isbn:978-0-7695-3613-2.

Sousa, J. P., B. Fonseca, et al. (2009). An Evolutionary Platform for the Collaborative Contextual Composition of Services. pp. 182-189. CRIWG - 15th Collaboration Researchers' International Workshop on Groupware, Régua, Portugal, Springer,isbn:978-3-642-04215-7.

Sowa, J. E. and S. C. Shapiro (2006). Knowledge Representation: Logical, Philosophical, and Computational Foundations (Paperback), Course Technology; 1 edition (August 17, 1999). isbn:0534949657.

Srinivasan, N. (2005). "OWL-S 1.1 API – Version 1.1 ", Retrieved 12/2009, from http://semwebcentral.org/projects/owl-s-api/.

Srivastava, B. and J. Koehler (2003). Web service composition --- current solutions and open problems. pp. ICAPS 2003.

Strang, T. and C. L. Popien (2004). A Context Modeling Survey. pp. In Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England.

SUN. (1997). "Remote Method Invocation Specification." SUN. Retrieved 09/2009, from http://v1.dione.zcu.cz/java/docs/jdk1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

Sure, Y., M. Erdmann, et al. (2002). OntoEdit: Collaborative Ontology Engineering for the Semantic Web. pp. Ffirst International Semantic Web Conference 2002 (ISWC 2002), Sardinia, Italy.

Swartout, W. and A. Tate (1999). "Ontologies." IEEE Intelligent Systems &Their Applications **14**(1): pp. 18-19. issn:1094-7167

SWSA, C. (2004). "Semantic Web Services Language - SWSL." SWSA Commitee. Retrieved 09/2009, from http://www.daml.org/services/swsa/.

SWUCSIG. (2004). "Semantic Web in UbiComp Special Interest Group." from http://pervasive.semanticweb.org.

Tebri, H., M. Boughanem, et al. (2005). Incremental profile learning based on a reinforcement method. pp. 1096-1101. SAC, ACM,isbn:1-58113-964-0.

Tempich, C. and R. Volz (2003). Towards a benchmark for Semantic Web reasoners - an analysis of the DAML ontology library. pp. 12. In Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON'03).

Thatte, S. (2001). XLANG Web services for business process design. Microsoft Corporation. isbn:ISBN.

Tim Berners-Lee, J. H. a. O. L. (2001) "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities." Retrieved Access Date, Access 2001, from http://www.scientificamerican.com/article.cfm?id=the-semantic-web.

Tsarkov, D. and I. Horrocks (2003). DL Reasoner vs. First-Order Prover. pp. 152-159. Description Logics, CEUR-WS.org.

Tsarkov, D. and I. Horrocks (2003). DL Reasoner vs. First-Order Prover. pp. Description Logics, CEUR-WS.org.

Tsarkov, D. and I. Horrocks (2006). FaCT++ Description Logic Reasoner: System

Description. Automated Reasoning**:** pp. 292-297

Unicode (2008). The Unicode Standard, Version 5.1. Unicode Consortium. from http://www.unicode.org/standard/standard.html.

UNSPSC. (1998). "United Nations Standard Products and Services Code." Retrieved 09/2009, from http://www.unspsc.org/.

Verma, K., K. Sivashanmugam, et al. (2003). METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. isbn:ISBN.

VESPER. (2000). "Virtual Home Environment for Service Personalization and Roaming Users (VESPER)." Retrieved 11/2009, from http://cordis.europa.eu/fetch?ACTION=D&CALLER=PROJ_IST&RCN=57458.

Volz, R., M. Horridge, et al. (2003). "The OWL API." Retrieved 12/2009, from http://owlapi.sourceforge.net/.

W3C. (2001). "XML Schema." W3C. Retrieved 08/2009, from http://www.w3.org/XML/Schema.

W3C. (2002). "Web Services Activity." W3C. 04/2008, from http://www.w3.org/2002/ws/.

Waldo, J. and K. Arnold (2000). The Jini Specifications, Addison-Wesley Professional; 2nd edition (December 15, 2000). isbn:978-0201726176.

Wang, S.-Y., Y. Guo, et al. (2005). The Semantic Web – ISWC 2005 - Rapid Benchmarking for Semantic Web Knowledge Base Systems, Springer Berlin / Heidelberg. isbn:978-3-540-29754-3.

Wang, X., J. S. Dong, et al. (2004). "Semantic Space: An Infrastructure for Smart Spaces." IEEE Pervasive Computing **3**(3): pp. 32-39. issn:1536-1268.

Want, R., V. Falcao, et al. (1992). "The Active Badge Location System." ACM Transactions on Information Systems **10**: pp. 91-102. issn:1046-8188.

Want, R., B. Schilit, et al. (1996). The Parctab Ubiquitous Computing Experiment. Mobile Computing, Springer US**:** pp. 45-101, isbn:978-0-7923-9697-0.

WSMO. (2004). "Web Service Modeling Ontology." Retrieved 09/2009, from http://www.wsmo.org/index.html.

Xiao, T. G., X. H. Wang, et al. (2004). An Ontology-based Context Model in Intelligent Environments. pp. 270--275. In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, CA, USA.

Yang, J. and M. Papazoglou (2002). Web Component: A Substrate for Web Service Reuse and Composition. pp. 21-36. CAiSE '02: Proceedings of the 14th International Conference on Advanced Information Systems Engineering, Springer-Verlag,isbn:3-540-43738-X.

Zolin, E. (2009, 09/2009). "Complexity of reasoning in Description Logics." Retrieved 09/2009, from http://www.cs.man.ac.uk/~ezolin/dl/.

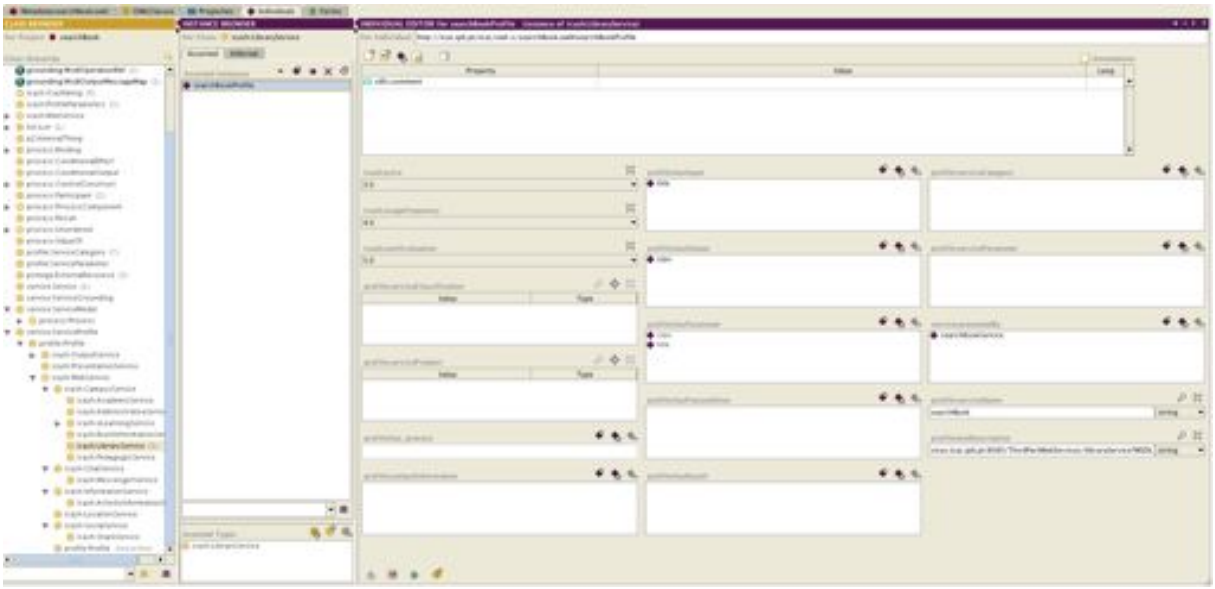
ANEXOS

Anexo I – Ontologia *Person* com algumas instâncias.

```
1:   <?xml version="1.0"?>
2:
3:
4:   <!DOCTYPE rdf:RDF [
5:       <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
6:       <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
7:       <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
8:       <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
9:   ]>
10:
11:
12:  <rdf:RDF xmlns="http://www.my-ontologies.com/person.owl#"
13:       xml:base="http://www.my-ontologies.com/person.owl"
14:       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
15:       xmlns:swrl="http://www.w3.org/2003/11/swrl#"
16:       xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
17:       xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
18:       xmlns:owl="http://www.w3.org/2002/07/owl#"
19:       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
20:       xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
21:       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
22:      <owl:Ontology rdf:about=""/>
23:
24:      <owl:Class rdf:ID="Female">
25:          <rdfs:subClassOf rdf:resource="#Gender"/>
26:          <owl:disjointWith rdf:resource="#Male"/>
27:      </owl:Class>
28:
29:      <Female rdf:ID="female"/>
30:      <owl:Class rdf:ID="Gender">
31:          <owl:equivalentClass>
32:              <owl:Class>
33:                  <owl:unionOf rdf:parseType="Collection">
34:                      <owl:Class rdf:about="#Female"/>
35:                      <owl:Class rdf:about="#Male"/>
36:                  </owl:unionOf>
37:              </owl:Class>
38:          </owl:equivalentClass>
39:      </owl:Class>
40:
41:      <owl:ObjectProperty rdf:ID="hasGender">
42:          <rdf:type rdf:resource="&owl;FunctionalProperty"/>
43:      </owl:ObjectProperty>
44:
45:      <owl:Class rdf:ID="Male">
46:          <rdfs:subClassOf rdf:resource="#Gender"/>
47:          <owl:disjointWith rdf:resource="#Female"/>
48:      </owl:Class>
49:
50:      <Male rdf:ID="male"/>
51:      <owl:Class rdf:ID="Man">
52:          <owl:equivalentClass>
53:              <owl:Class>
54:                  <owl:intersectionOf rdf:parseType="Collection">
55:                      <owl:Restriction>
56:                          <owl:onProperty rdf:resource="#hasGender"/>
57:                          <owl:allValuesFrom rdf:resource="#Male"/>
58:                      </owl:Restriction>
```

```
59:                     <owl:Restriction>
60:                         <owl:onProperty rdf:resource="#hasGender"/>
61:                         <owl:someValuesFrom rdf:resource="#Male"/>
62:                     </owl:Restriction>
63:                     <owl:Class rdf:about="#Person"/>
64:                 </owl:intersectionOf>
65:             </owl:Class>
66:         </owl:equivalentClass>
67:     </owl:Class>
68:
69:     <Person rdf:ID="Maria">
70:         <hasName rdf:datatype="&xsd;string">Maria</hasName>
71:         <hasGender rdf:resource="#female"/>
72:     </Person>
73:
74:     <owl:DatatypeProperty rdf:ID="hasName">
75:         <rdf:type rdf:resource="&owl;FunctionalProperty"/>
76:     </owl:DatatypeProperty>
77:
78:     <Person rdf:ID="Pedro">
79:         <hasName rdf:datatype="&xsd;string">Ana</hasName>
80:         <hasGender rdf:resource="#female"/>
81:     </Person>
82:
83:     <owl:Class rdf:ID="Person">
84:         <rdfs:subClassOf>
85:             <owl:Restriction>
86:                 <owl:onProperty rdf:resource="#hasGender"/>
87:                 <owl:someValuesFrom rdf:resource="#Gender"/>
88:             </owl:Restriction>
89:         </rdfs:subClassOf>
90:         <rdfs:subClassOf rdf:resource="&owl;Thing"/>
91:     </owl:Class>
92:
93:     <owl:Class rdf:ID="Woman">
94:         <owl:equivalentClass>
95:             <owl:Class>
96:                 <owl:intersectionOf rdf:parseType="Collection">
97:                     <owl:Class>
98:                         <owl:complementOf rdf:resource="#Man"/>
99:                     </owl:Class>
100:                    <owl:Class rdf:about="#Person"/>
101:                </owl:intersectionOf>
102:            </owl:Class>
103:        </owl:equivalentClass>
104:    </owl:Class>
105: </rdf:RDF>
106:
```

Anexo II – Edição de uma ontologia utilizando a ferramenta Protégé.



Anexo III – Resultados de testes ao *motor de composição*.

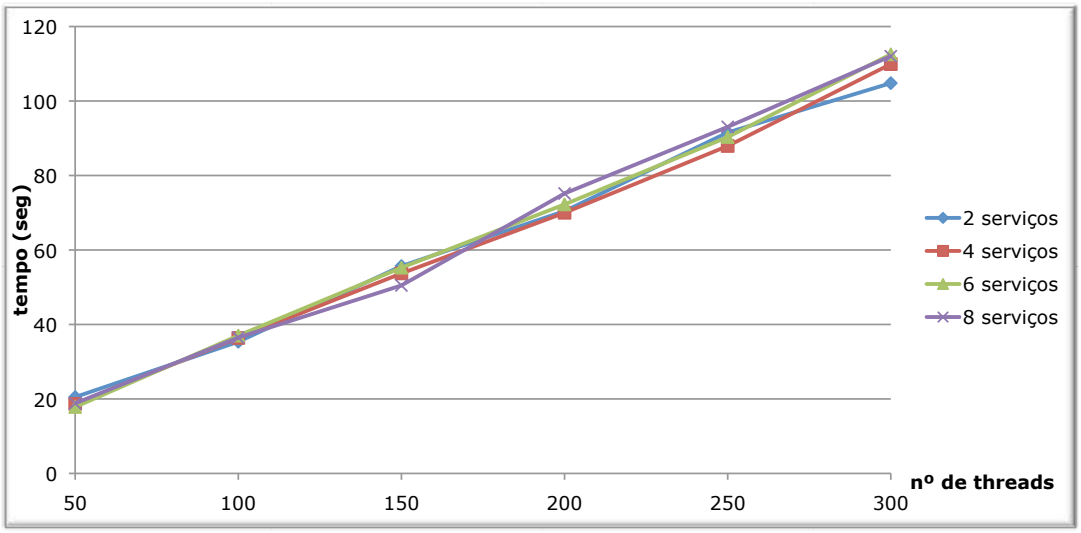


Gráfico 0.1 - Tempo total consumido para a composição e execução de serviços com TCWS de 20 ms.

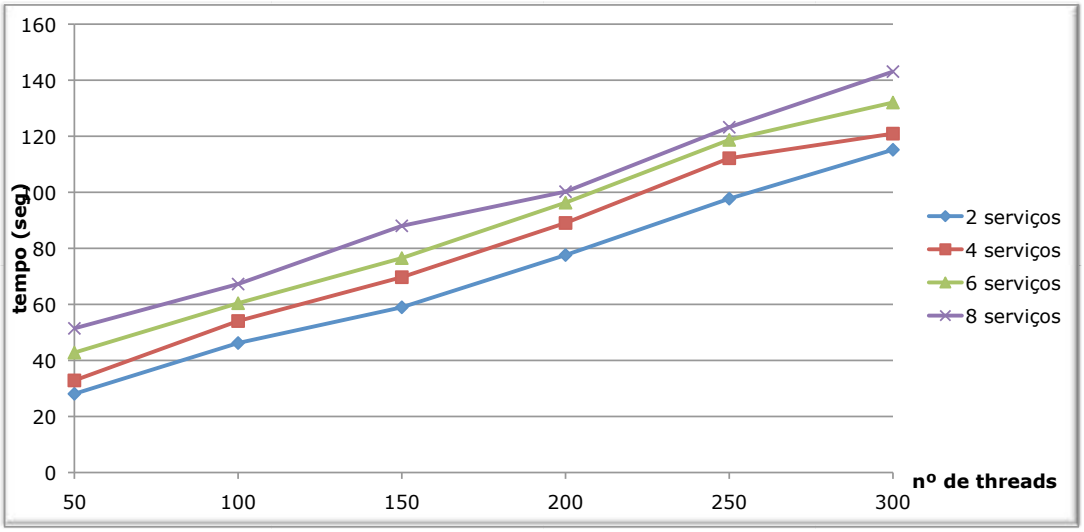


Gráfico 0.2 - Tempo total consumido para a composição e execução de serviços com TCWS de 4 seg.

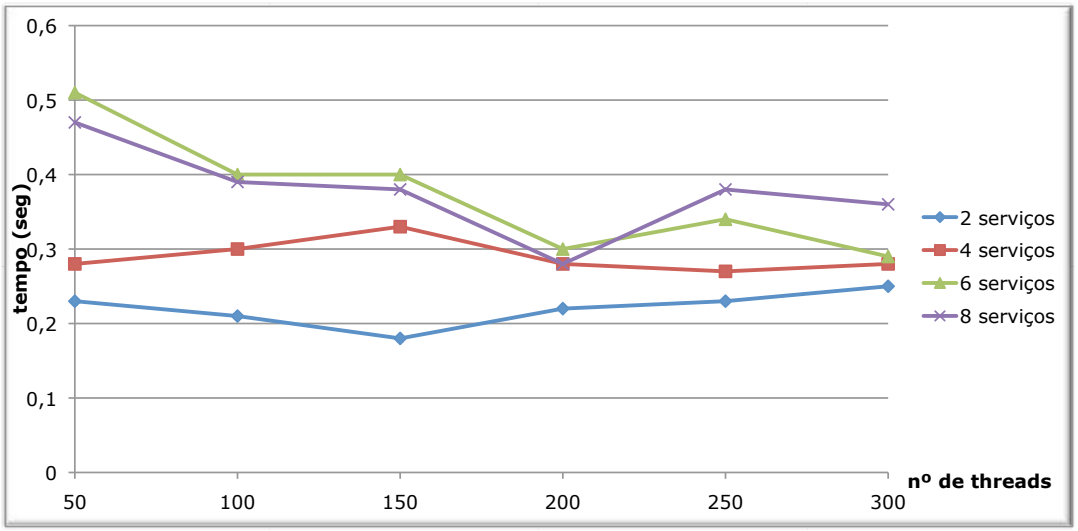


Gráfico 0.3 - Tempo médio para a execução e composição, utilizando serviços (TCWS de 20 ms).

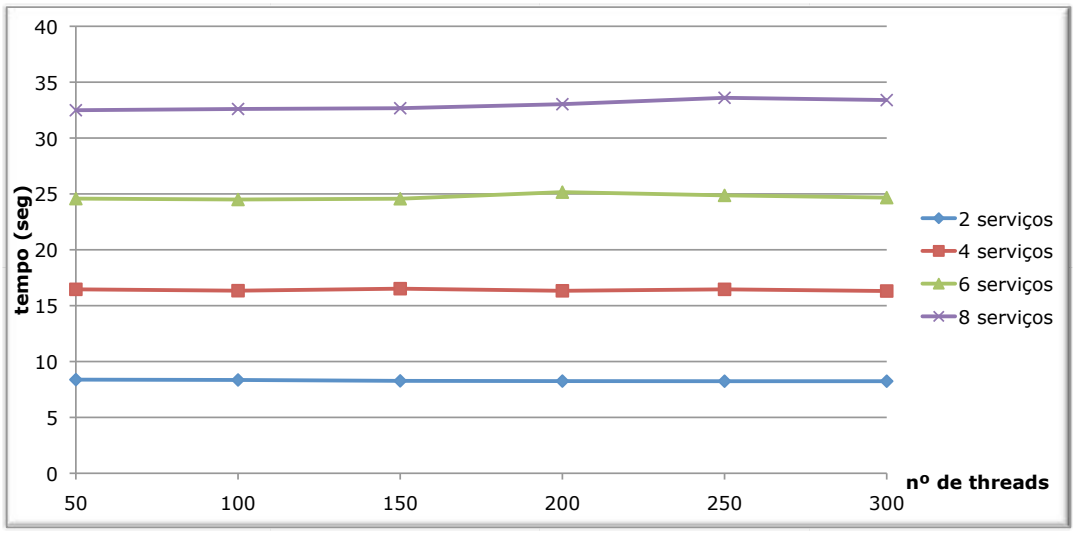


Gráfico 0.4 - Tempo médio para a execução e composição, utilizando serviços (TCWS de 4 seg).