# 5
# Um *Framework* de Aplicações para a *Web* Semântica (SWAPpFW - *Semantic Web Application Framework*)

A principal influência para definir o *framework* de aplicações para a *Web* Semântica (SWAPpFW) vem de uma abordagem de engenharia de sistemas definida em (Sommerville, 2000). Conseqüentemente, neste capítulo, define-se quais sãos os requisitos e a arquitetura do *framework*. Também discute-se como fazer a transição da arquitetura para o *design* do *framework*.

## 5.1. Requisitos

Como explicado anteriormente, o *framework* está no contexto do SWC. Assim, a principal fonte dos requisitos origina-se da definição do que é uma SWAPp para o concurso (Capítulo 3). Assim, na próxima seção, classifica-se a definição de uma SWAPp em termos de requisitos funcionais e não funcionais.

Durante a revisão das aplicações (Cunha, 2007) ficou claro que algumas aplicações seguiram um processo “comum” para tratar metadados. Este processo ajuda a tornar mais claras quais são as principais fases que podem ser seguidas pelas aplicações para tal. Considera-se também este processo como uma fonte para a definição dos requisitos e ele é definido na Seção 5.1.2

A análise de domínio do SWC também é uma importante fonte de requisitos pois identifica as funcionalidades das aplicações e como algumas destas funcionalidades podem ser agrupadas e identificadas como um tipo de aplicação (Capítulo 4). Outra fonte de requisitos é a discussão sobre a controvérsia sobre a *Semantic Web stack* (Seção 2.3). Apesar da análise de domínio e a controvérsia serem importantes fontes de requisitos, preferiu-se não incluí-las nos requisitos já que estão mais relacionadas com um ponto de vista de arquitetural. Logo, estas discussões estão dispersas nas sub-seções da seção 5.3.

### 5.1.1.
### Os requisitos do SWC

Como definido na Seção 3.1, o SWC define uma SWAPp em função de um conjunto de requisitos e qualidades desejáveis. Nesta seção, propomos a classificação destes requisitos e qualidades desejáveis em requisitos funcionais e não-funcionais.

Requisitos funcionais descrevem as funções ou serviços oferecidos por um sistema. Os requisitos dependem to tipo de software, seus usuários e do tipo de sistema onde o software será utilizado. Requisitos não funcionais são restrições dos serviços ou funções oferecidos pelo sistema. Exemplos destas restrições são restrições de tempo, restrições no processo de desenvolvimento e padrões (*standards*) (Sommerville, 2000). Em uma abordagem mais pragmática dever-se-iam classificar todos os requisitos e qualidades desejáveis do SWC como requisitos não-funcionais. Entretanto, adotou-se uma posição de classificar algum destes como requisitos funcionais já que eles representam serviços que SWAPps deveriam oferecer.

**Requisitos funcionais:**

- R1. Considerando as fontes de informação, elas devem:
  - R1.1. ser geograficamente distribuídas;
  - R1.2. ser de diferentes proprietários - não há controle de evolução;
  - R1.3. ser heterogêneas (sintática, estrutural e semanticamente);
- R2. Considerando a opção entre mundo aberto ou fechado (*open/close world*): a aplicação precisa considerar um mundo aberto, ou seja, que a informação nunca está completa;
- R3. Considerando a descrição do significado dos dados: a aplicação deve usar alguma descrição formal.
- R4. Considerando as fontes de dados, elas deveriam:
  - R4.2. explorar tanto conhecimento estático como dinâmico - por exemplo uma combinação de ontologias estáticas e *workflows* dinâmicos;
  - R4.3. usar o conteúdo de documentos multimídia.

- R5. Considerando acesso do usuário:
    - R5.1. acesso em diferentes idiomas deveria ser oferecido;
    - R5.2. acesso a partir de outros dispositivos, além de um computador pessoal, deveria ser oferecido.

**Requisitos não-funcionais:**

- R1. Considerando as fontes de informação das aplicações, elas precisam:
    - R1.4. conter dados reais (*real-world data*) - isto é, as fontes precisam ser mais que *toy examples*.
- R4. Considerando as fontes de dados, elas deveriam:
    - R4.1. ser usadas para outros propósitos ou de outra forma que não a originalmente projetada;
- R6. Considerando a escalabilidade: as aplicações deveriam ser escaláveis em termos
    - R6.1. da quantidade de dados utilizada;
    - R6.2. do número de componentes distribuídos trabalhando em conjunto.

Como mencionado anteriormente, não se está usando um ponto de vista "tradicional" a respeito dos requisitos; desta forma, poder-se-ia classificar os requisitos R2, R3 e R4.2 tanto como funcionais ou como não-funcionais já que eles poderiam ser serviços oferecidos por uma SWAPp ou restrições nas funções ou serviços de uma SWAPp.

Como apresentado em (Cunha, 2007), em cada edição do concurso havia um objetivo adicional definido pelo comitê de avaliação. Os objetivos adicionais para as três edições foram:

2003: Aplicações deveriam integrar pelo menos duas fontes heterogêneas de dados ou informações em XML não gerenciadas pelo autor da aplicação e que permitissem diferentes pontos de vista.

2004: Mostrar os benefícios da capacidade de inferência das linguagens da *Web* Semântica usadas pelas aplicações.

2005: Mostrar os benefícios da re-utilização de ontologias, esquemas ou modelos.

2005: Adicionalmente, um objetivo informal era como você (o autor da aplicação submetida) explicaria *Web* Semântica para seus avós.

Para este trabalho, não se considera os objetivos de cada ano pois deseja-se que o *framework* seja tão geral quanto possível. Adicionalmente, ao lidar com

um *framework*, será possível personalizá-lo para representar novos objetivos que podem ser incluídos nos anos subseqüentes.

### 5.1.2. O Processo de Manipulação de Metadados (*Metadata Handling Process*)

Durante a revisão das aplicações, ficou claro que um tipo de arquitetura era comum a muitas aplicações: havia componentes ou camadas responsáveis por atividades específicas do uso de metadados (Hartmann & Sure, 2004) (Takeda & Ohmukai, 2005) (Shadbolt *et al.*, 2004) (Tummarello *et al.*, 2005) (Keller *et al.*, 2004) (Hyvönen *et al.*, 2005) (Mika, 2005a) (Haase *et al.*, 2004) (Beneventano & Bergamaschi, 2004) (Baumgartner *et al.*, 2005) (Baker *et al.*, 2006).

Por exemplo o *extended* SEAL *framework* (Hartmann & Sure, 2004) tem cinco camadas conceituais que podem ser consideradas como *workflows* de conhecimento. Cada camada tem uma tarefa específica para lidar com os dados. A camadas de "*Integration*" e "*Process and Publication*" são responsáveis pela entrada de dados na camada de "*Representation*". Nesta camada há uma atividade de "*knowledge evolution*" que pode ser considerada como uma forma de inferência (*reasoning*) aplicada ao repositório de conhecimento. As camadas de "*Representation*" e "*Organization*" podem ser consideradas como responsáveis por armazenar o conhecimento, "desenvolvê-lo (*evolve it*)" e fornecer funcionalidades de indexação e busca a este conhecimento. Finalmente, a camada "Access" é responsável pela manipulação de dados a fim de gerar uma representação conveniente para o usuário final.

Outro exemplo é a organização do SemanticOrganizer. Keller *et al.* (Keller *et al.*, 2004) apresentam-no como um conjunto de componentes arquiteturais organizados em camadas. A camada de "*Representation & Reasoning*" contem a maior parte dos componentes para a entrada de dados na aplicação ("*Email-ingestor*", "*Semantic Annotation*" etc.). Na mesma camada, encontra-se o "*Semantic Repository*" que armazena classes e instâncias; e o "*Inference Engine*" para inferência (*reasoning*) sobre os dados. Na camada "*Interface*", encontram-se diferentes formas de uso e acesso aos dados no repositório.

Acredita-se que estas camadas ou componentes representem fases de um processo de manipulação de dados em aplicações da *Web* Semântica Assim, uma generalização das camadas ou componentes em fases pelas quais

metadados tem que passar antes de ser oferecidos aos usuários finais é proposta a seguir.

Todo sistema de informação têm que coletar dados, armazená-los e processá-los para oferecê-los aos usuários finais. As SWAPps, de certa forma, são diferentes dos sistemas de informações usuais a medida que lidam com metadados que tem sua semântica bem definida. Desta forma, em qualquer das fases (coleta, armazenamento e uso) a manipulação de metadados pode ocasionar perda da semântica. E para SWAPps, a preservação da semântica é de grande importância ao longo do processo.

Na Figura 6, são apresentadas as 3 fases do processo de manipulação de metadados. As fases são *Metadata Gathering*, *Metadata Storage* e *Metadata Usage*. Entre todas as fases há pelo menos um fluxo de metadados. O fluxo de metadados indica que os metadados que são saída de uma fase servem como entrada para outra fase.

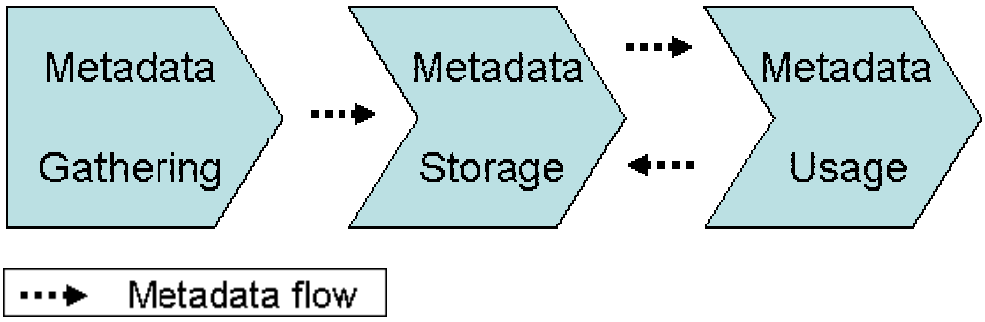


Figura 6 - O Processo de Manipulação de Metadados (*Metadata Handling Process*)

Na representação do fluxo de metadados da fase de *Metadata Usage* para a *Metadata Storage*, considera-se que o uso de metadados pode gerar novos metadados que podem vir a ter que ser armazenados. Algumas aplicações consideram este fluxo de metadados como um “auto-fluxo” da fase *Metadata Storage* (Hartmann & Sure, 2004) (Keller *et al.*, 2004) (Mika, 2005a). Aqui prefere-se separá-los porque esta separação dependerá da heurística escolhida para inferência e atualização dos metadados já coletados e armazenados.

Considerando-se o último parágrafo, poder-se-ia simplificar o processo de manipulação de metadados como tendo apenas duas fases: *Metadata Storage* e *Metadata Usage*. Entretanto a existência da fase *Metadata Gathering* é importante porque a utilidade da aplicação está diretamente relacionada a quantidade de informação disponível para o usuário. Além disso, o desenvolvimento de componentes que são capazes de anotar dados e dar os primeiros passos para o usuário não são apenas uma qualidade desejável das SWAPps mas também um incentivo ao uso deste tipo de aplicações.

Conseqüentemente, espera-se que as SWAPps geradas pelo *framework* proposto neste trabalho sejam capazes de ter pelo menos um componente para cada fase do processo de manipulação de metadados. Também espera-se que as aplicações utilizem uma ontologia (ou modelo) comum durante o processo de manipulação dos metadados a fim de não perder semântica. A perda de semântica pode ser evitada com outras abordagens, como por exemplo, se os dados necessitam ser convertidos de uma ontologia para outra, ou se a linguagem de representação tem que ser alterada, então algum tipo de anotação pode ser usada para manter a semântica que pode ser perdida na conversão ou tradução.

Os requisitos do SWC apresentados na seção anterior e o processo de manipulação de metadados servem como uma entrada para a definição da arquitetura do *framework* na próxima seção. Como poderá ser visto, outras considerações também são importantes para a definição da arquitetura. Elas são a análise de domínio das aplicações do SWC apresentada no capítulo 4, e a controvérsia sobre a *Semantic Web stack*, apresentada na seção 2.3.

## 5.2. Arquitetura do SWAPpFW

Uma arquitetura de software é, freqüentemente, representada por apenas um diagrama ou documento. Entretanto, este diagrama ou documento não é capaz de comunicar todas as preocupações que um engenheiro de software tem que ter a fim de mapear os requisitos para os modelos de *design* ou desenvolvimento. Neste trabalho, inspirado por Conallen (Conallen, 1999), usa-se uma abordagem de múltiplos pontos de vista, ou visões, a fim de definir a arquitetura do *framework*. A introdução de Conallen à visualização de uma arquitetura através de visões deve-se ao *The "4+1" View Model of Software Architecture* (Kruchten, 1995). Na próxima seção, é apresentado o trabalho de Kruchten a fim de utilizá-lo como base para a definição da arquitetura do *framework* nas seções seguintes.

### 5.2.1. O *"4+1" View Model of Software Architecture*

O *"4+1" View Model of Software Architecture* é uma tentativa de abstrair, decompor e compor com estilo e estética o *design* e implementação de uma estrutura alto-nível de um software. Para descrever uma arquitetura de software,

o *“4+1” View Model* fornece um modelo composto por múltiplos pontos de vista, visões ou perspectivas conforme apresentado na Figura 7 (Kruchten, 1995).

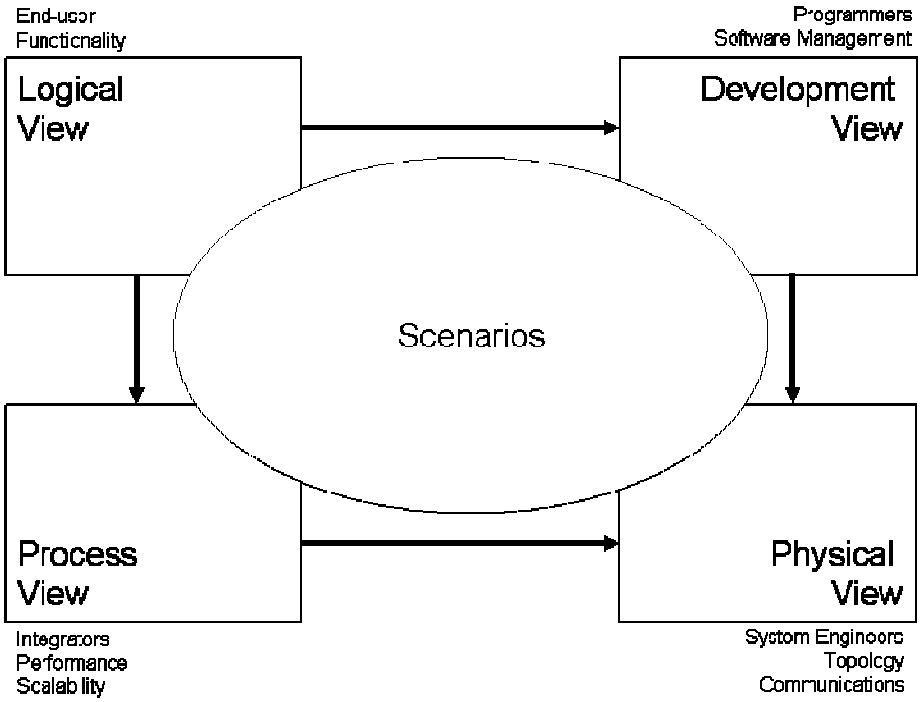


Figura 7 - *The "4+1" View Model* (Kruchten, 1995)

As visões ou perspectivas na Figura 7 são (Kruchten, 1995):

- A visão lógica (*logical view*), que é primordialmente a representação dos requisitos funcionais do software;
- A visão de processos (*process view*), que captura aspectos de concorrência e sincronização do *design*;
- A visão física (*physical view*), que descreve os mapeamentos do software no hardware e reflete seu aspecto distribuído;
- A visão de desenvolvimento (*development view*), que descreve a organização estática do software no seu ambiente de desenvolvimento;
- Alguns casos de uso, ou cenários descrevendo a arquitetura

Kruchten (Kruchten, 1995) afirma que o *“4+1” View Model* bastante genérico: ele define notações para cada um dos pontos de vista (ou visões), mas outras notações podem ser utilizadas. Adicionalmente, é possível ajustar o modelo, já que nem todos os softwares precisam de todos os pontos de vista. Um ponto de vista pode ser omitido e outros podem ser criados se o contexto do software requerer. Isto é mostrado na próxima seção: como foi feito o ajuste do *“4+1” View Model* para trabalhar com diagramas UML e representar a arquitetura de um *framework* de aplicações para a *Web* Semântica.

### 5.2.2.
### O Ajuste do *“4+1” View Model*

Na Figura 8, é apresentado um *“4+1” View Model* ajustado. Inicialmente a visão lógica foi renomeada para visão de análise (*analysis view*). Isto foi feito para evitar ambigüidades já que diversos aspectos da *Web* Semântica são baseados em Lógica. Entretanto, a função da visão de análise permanece a mesma da visão de lógica que é representar os requisitos funcionais do software em um diagrama de classes.

Na visão de processos, alguns requisitos não-funcionais, como desempenho e disponibilidade, são considerados. Entende-se que estes requisitos não funcionais poderiam ser representados por um diagrama de interação como um diagrama de seqüência com raias mostrando quais são as fases de um processo que está sendo executado.

Um diagrama de classes representará a visão de desenvolvimento e focará na modularização do software assim como nos requisitos internos relacionados ao software.

Um diagrama de implantação (*deployment diagram*) representará a visão física focando na representação da escalabilidade da distribuição da arquitetura e, eventualmente, outros requisitos não-funcionais.

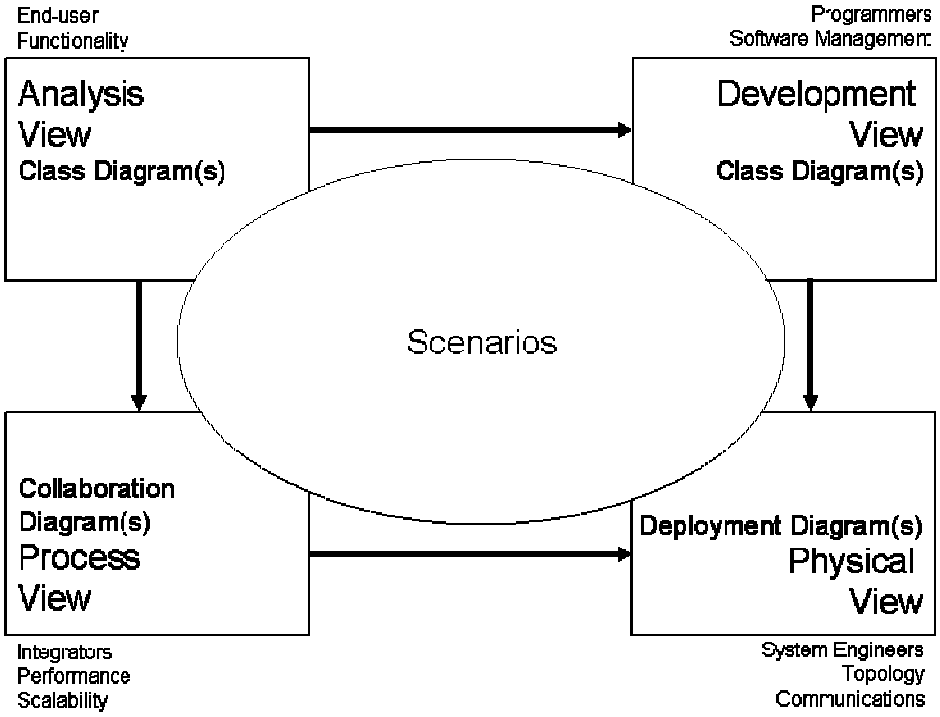


Figura 8 - O *"4+1" View Model* ajustado

Finalmente, os cenários serão representados como diagramas de caso de uso. Esses cenários representarão os requisitos elicitados até o momento e outros que venham a ser considerados necessários. Na próxima seção, são apresentadas as visões do SWAPpFW, ou seja, a instanciação do *"4+1" View Model* que representa a arquitetura do *framework* de aplicações para a *Web* Semântica.

## 5.3. As Visões do SWAPpFW

Nas próximas sub-seções, as visões do SWAPpFW são apresentadas e brevemente discutidas. Estas visões são diagramas UML e representam a arquitetura do SWAPpFW. Primeiro os cenários são definidos, seguidos pelas apresentações das outras visões definidas na seção 5.2.2.

### 5.3.1. Os Cenários (*Scenarios*)

Uma das características mais atrativas do *"4+1" View Model* é que diferentes tópicos (*concerns)* não são representados, necessariamente, em apenas uma visão, mas eles são "reconciliados" por cenários. Na Figura 9, são apresentados os elementos que compõe o cenário geral do SWAPpFW:

- Os requisitos do SWC (seção 5.1.1);
- A análise de domínio das aplicações do SWC (capítulo 4);
- O Processo de Manipulação de Metadados (seção 5.1.2); e
- A controvérsia sobre a *Semantic Web stack* (seção 2.3).

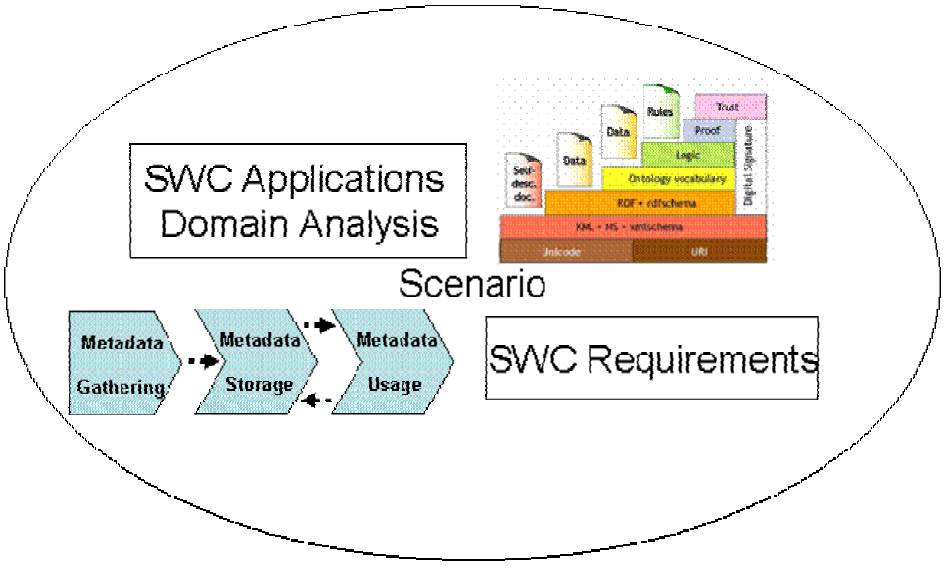


Figura 9 - O Cenário Geral do SWAPpFW

Todos os elementos que compõem o cenário geral não puderam ser facilmente mapeados em um único diagrama de caso de uso. Assim, no restante desta seção é apresentado um conjunto de diagramas de caso de uso que representa estes elementos.

O primeiro elemento a ser mapeado em um diagrama de caso de uso foram os requisitos do SWC. Estes requisitos são bastante amplos. Assim, obteve-se um diagrama de caso de uso bastante genérico representado na Figura 10.

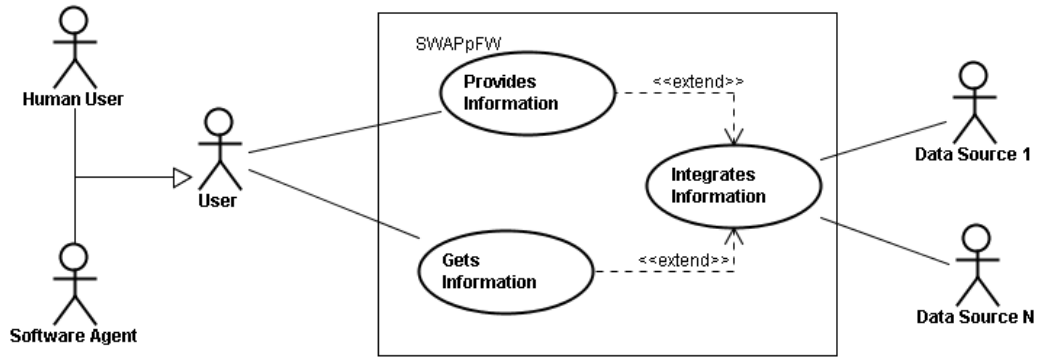


Figura 10 - Cenário 0: Os Requisitos do SWC

No Cenário 0 há, basicamente, dois tipos de atores: usuários e fontes de dados. O usuário pode ser um agente de software ou uma pessoa. Adicionalmente, há pelo menos duas fontes de dados, que são necessariamente heterogêneas e geograficamente distribuídas

Os casos de uso do Cenário 0 são bastante genéricos assim como os requisitos do SWC. Eles oferecem ao usuário a opção de fazer uso de informações que requerem que o *framework* integre dados de pelo menos duas fontes de dados diferentes.

Não foi possível representar alguns requisitos do SWC no Cenário 0. Por exemplo, o uso de alguma descrição formal para o significado dos dados manipulados pelo *framework*. No entanto, os requisitos faltantes deverão aparecer em outras visões da arquitetura. No caso do exemplo fornecido, o uso de descrição formal para o significado dos dados aparecerá na visão de análise na seção 5.3.2.

Na Figura 11, Cenário 1, foi incorporado um novo elemento do cenário geral ao Cenário 0: o Processo de Manipulação de Metadados. Continua-se com os mesmo tipo de atores e foram incluídos novos casos de uso: *Metadata Gathering*, *Metadata Storage* Access e *Metadata Usage*. Além disso, relacionaram-se os novos casos de uso aos já definidos no Cenário 0.

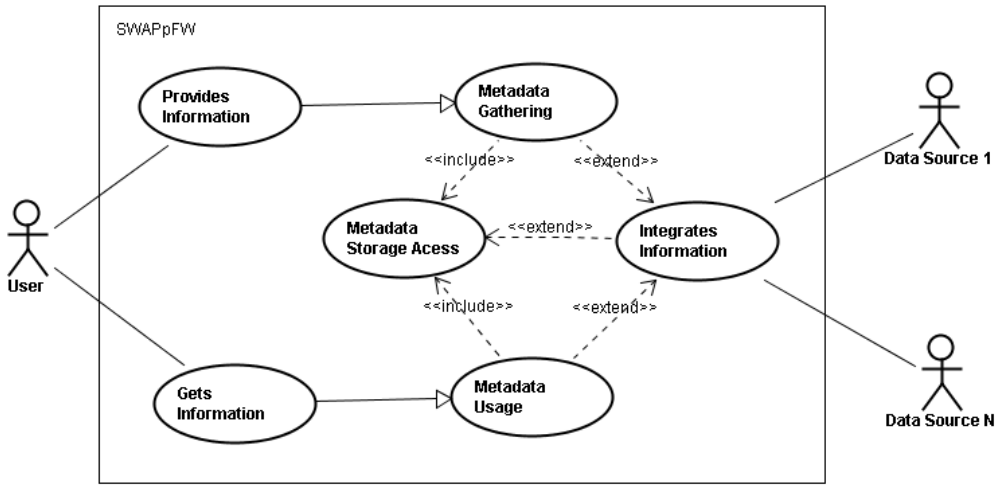


Figura 11 - Cenário 1: Requisitos do SWC + Processo de Manipulação de Metadados

Os relacionamentos de generalização introduzidos no Cenário 1 significam que os casos de uso "*Provides Information*" e "*Gets Information*" são respectivamente um tipo de "*Metadata Gathering*" e "*Metadata Usage*". Adicionalmente, a representação do relacionamento *include* entre os casos de uso "*Metadata Gathering*" e "*Metadata Usage*" e o caso de uso "*Metadata Storage* Access" significa que os primeiros casos de uso incorporarão o comportamento do caso de uso "*Metadata Storage* Access".

O relacionamento *extend* representado no Cenário 1 significa que: os casos de uso "*Metadata Gathering*" e "*Metadata Usage*", em certas condições, incorporarão o comportamento do caso de uso "*Integrates Information*"; e o caso de uso "*Integrates Information*", também sob certas condições, terá que incorporar o comportamento do caso de uso "*Metadata Storage* Access".

Mesmo que um caso de uso bem estruturado não seja um caso excessivamente generalizado (Booch *et al.*, 1998). Foi escolhido não mapear os dois elementos remanescentes do cenário geral (*SWC Applications Domain Analysis* e *Controversialism about the Semantic Web stack*) em diagramas de casos de uso. Esta decisão foi tomada porque esses dois elementos estão mais relacionados a interação de sub-sistemas e devem ser transparentes aos usuários. Adicionalmente, os dois elementos parecem ser mais importantes para a definição do *design* do *framework*. De qualquer forma, eles devem aparecer nas visões nas quais suas influências são mais percebidas (visão de análise e visão de desenvolvimento).

Agora que o cenário do SWAPpFW foi definido, pode-se avançar e descrever cada uma das outras visões propostas pelo *"4+1" View Model* ajustado. Isto é feito nas próximas sub-seções.

### 5.3.2.
### A Visão de Análise (*Analysis View*)

Na visão de análise, deve-se manter o usuário final em mente e tentar mostrar-lhe quais sãos as funcionalidades do software. Isto posto, apresenta-se o diagrama de classe na Figura 12 que representa o SWAPpFW.

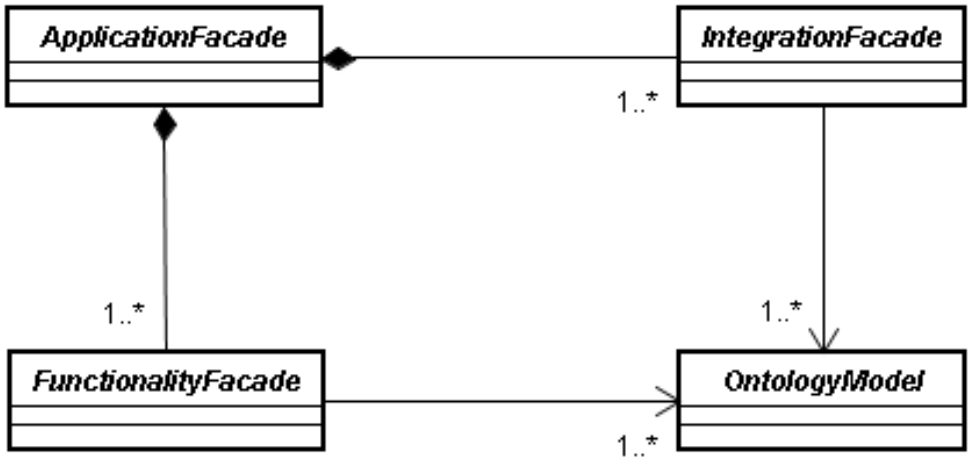


Figura 12 – O modelo de visão de análise do SWAPpFW

No diagrama da Figura 12, considerou-se que os tipos de aplicação (ApplicationFacade) gerados com o SWAPpFW são compostos de uma ou mais funcionalidades (FunctionalityFacade) e usam pelo menos um tipo de integração de dados (IntegrationFacade). Também foi considerado que as funcionalidades e o método de integração usam pelo menos um modelo de ontologia (OntologyModel).

Fica claro que a principal influência para o modelo de visão de análise do SWAPpFW vem dos requisitos do SWC. Entretanto, ele também é influenciado pela análise de domínio das aplicações do SWC, o que pode ser visto pelo uso da classe FunctionalityFacade. Na visão de desenvolvimento, avança-se na definição e especialização dos classes apresentadas nesta visão.

### 5.3.3.
### A Visão de Desenvolvimento (*Development View*)

A visão de desenvolvimento serve como um artefato que ajuda os desenvolvedores a entender e gerenciar melhor a organização estática dos conceitos, definidos na visão de análise, no ambiente de desenvolvimento do software. Para atingir isto, descreve-se e define-se cada um dos conceitos do modelo de visão de análise.

Inicialmente, apresenta-se um diagrama de classes para a ApplicationFacade na Figura 13. A ApplicationFacade é o *Facade design pattern* (Gamma *et al.*, 1995) aplicado aos tipos mais usuais de aplicações encontrados na análise de domínio das aplicações do SWC.

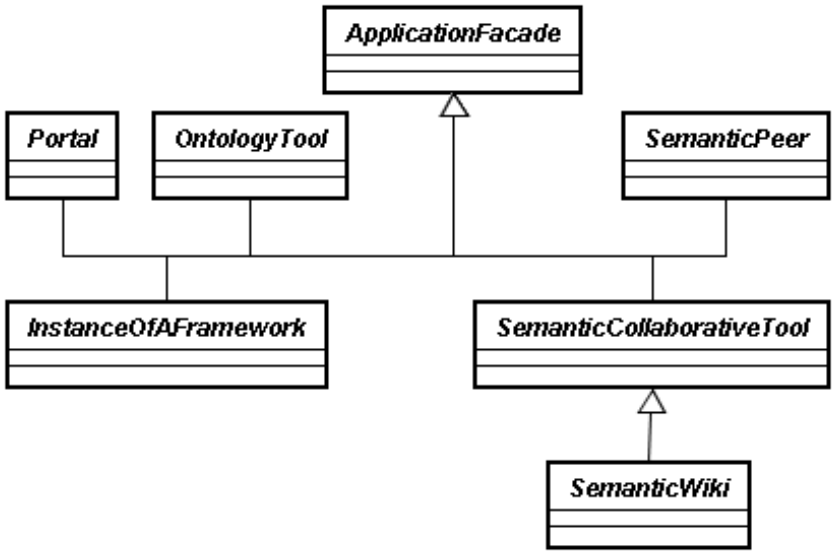


Figura 13 - Pacote *Application*

As definições de cada tipo de aplicação, mostradas na Figura 13, são dadas na seção 4.9. É possível, ortogonalmente, classificar os tipos de aplicações pela sua definição ou uso. Por exemplo:

- Portal e OntologyTool são tipos de aplicações que são compostas de funcionalidades específicas;
- InstanceOfAFramework é um tipo de aplicação onde algumas funcionalidades de um *framework* já definido são reutilizadas
- SemanticPeer é um tipo de aplicação definido por seu modelo particular de implantação (em uma rede *Peer-to-Peer*); e
- SemanticCollaborativeTool e sua especialização, SemanticWiki, são tipos de aplicações que são definidos pelo processo que eles empregam ao lidar com metadados.

Qualquer dos tipos de aplicação, mesmo quando definidos pela composição de algumas funcionalidades, podem oferecer funcionalidades “extras”. Além disso, o uso do *design pattern Facade* para representar os tipos de aplicação é uma forma de possibilitar a definição de outros tipos de aplicação. O uso do padrão também introduz a questão de que cada tipo de aplicação não é a simples implementação de uma classe.

Por outro lado, cada tipo de aplicação será a fachada que oferecerá acesso a múltiplas classe, caracterizando, desta forma, um subsistema. Esta caracterização de sub-sistemas e suas implicações serão discutidas na visão de processo na seção 5.3.4.

Na Figura 14, é apresentada a FunctionalityFacade, que é o *Facade design pattern* aplicado aos tipos de funcionalidades oferecidos pelas aplicações submetidas ao SWC. Como explicado na análise de domínio das aplicações do SWC (capítulo 4), as funcionalidades aqui representadas são aquelas que chamaram a atenção pelo seu uso de semântica ou por seu uso freqüente pelas aplicações submetidas ao concurso.

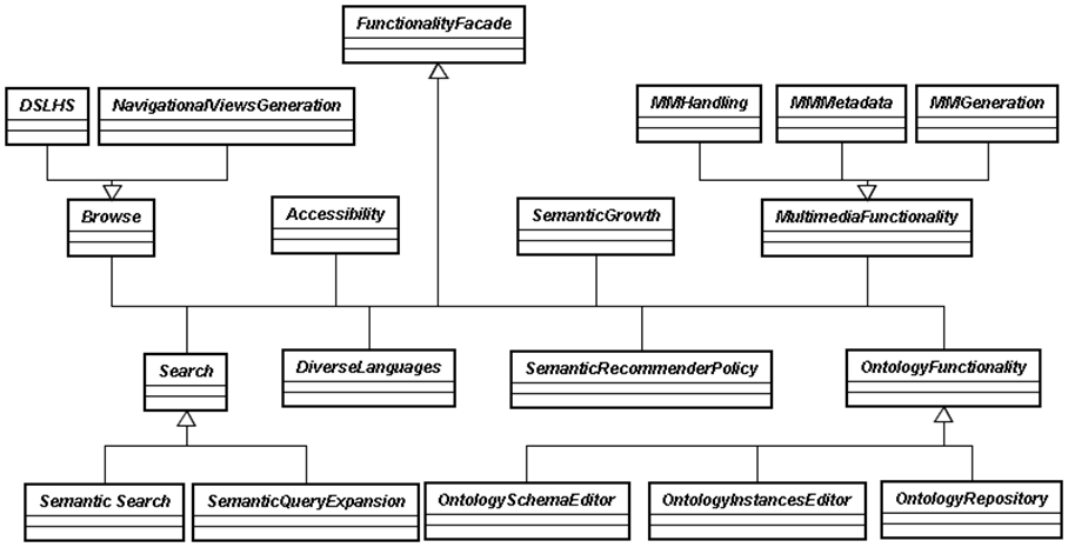


Figura 14 - Pacote *Functionality*

As definições de cada tipo de funcionalidade, mostradas na Figura 14, são feitas da seção 4.1 à seção 4.8, totalizando 18 tipos de funcionalidades. No entanto, duas funcionalidades são “muito” abstratas e nunca foram utilizadas na análise de domínio das aplicações do SWC: MultimediaFunctionality e OntologyFunctionality. Elas são apenas uma forma de organização coerente das funcionalidades que as especializam.

Características importantes deste conjunto de funcionalidades é que todas as funcionalidades usam metadados da aplicação e são bastante genéricas, de forma que poderiam ser utilizadas por qualquer aplicação. A exceção é a MultimediaFunctionality e suas especializações, que podem ou não usar metadados, e são um requisito específico do SWC. Poder-se-ia, sem perda de generalização, renomear a MultimediaFunctionality como DocumentFunctionality. Desta forma teríamos um *framework* ainda mais flexível e que poderia ser instanciado para lidar com outros tipos de documentos que não só os

multimídias, por exemplo, documentos de Bioinformática, geo-referenciados ou qualquer combinação de todos eles.

Quaisquer dos tipos de funcionalidade podem ser combinados para compor uma aplicação. Entretanto pelo menos um é necessário para caracterizar uma aplicação como uma SWAPp. Além disso, o uso isolado de alguma funcionalidade ou algumas combinações de funcionalidade podem não fazer sentido, não caracterizando uma SWAPp. Como será comentado no próximo parágrafo, esta é uma das implicações da interação entre sub-sistemas.

O uso do *design pattern Facade* para representar os tipos de funcionalidades é uma forma de possibilitar a definição de outros tipos de funcionalidade. O uso do padrão também introduz a questão de que cada tipo de funcionalidade será uma fachada que oferece acesso a várias classes, caracterizando, assim, um sub-sistema. Essa caracterização de sub-sistemas e suas implicações serão discutidas na visão de processo na seção 5.3.4.

Um dos requisitos-chave do SWC é que as aplicações integrem dados de diferentes fontes de dados. Na análise de domínio das aplicações submetidas ao concurso, observou-se que duas abordagens para atender a este requisitos eram utilizadas: a *Wrappers and Mediators Integration Functionality* (seção 4.10.1) e a *Manual Integration Functionality* (seção 4.10.2). Na Figura 15, elas são generalizada como a classe IntegrationFacade, que é o *Facade design pattern* aplicado ao tipo de integração de dados.

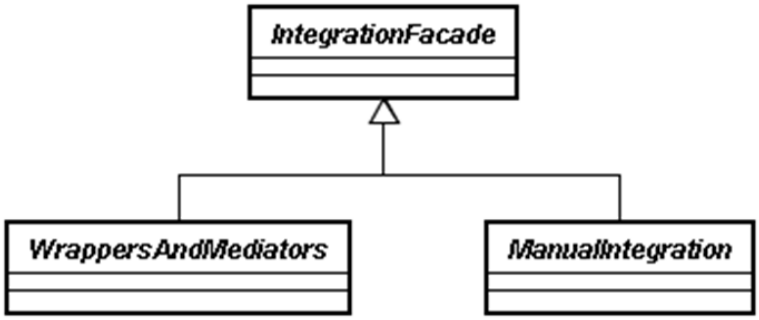


Figura 15 - Pacote *Integration*

O uso do *Facade design pattern* para representar os tipos de integração possibilita a definição de novos tipos de integração e também introduz a questão de que cada tipo de integração não é a implementação de uma única classe. Por outro lado, cada tipo de aplicação será uma fachada que oferece acesso a várias classes, caracterizando, assim, um sub-sistema. Essa caracterização de sub-sistemas e suas implicações serão discutidas na visão de processo na seção 5.3.4.

Os tipos de integração podem ser usados isoladamente ou de forma combinada na mesma aplicação. Contudo pela definição de uma SWAPp, qualquer aplicação terá pelo menos um tipo de integração implementado.

Outro requisito importante do SWC é o uso de alguma descrição formal para o significado dos dados da aplicação. Para atender a este requisito e estar de acordo com a idéia geral da área de *Web* Semântica de representar semântica utilizando ontologias, utilizou-se o *Abstract Factory design pattern* (Gamma *et al.*, 1995).

O uso deste padrão possibilita ao projetista da aplicações que as configure com uma ou mais fábricas de ontologia (OntologyFactory). O primeiro produto que, naturalmente, vêm a mente é o modelo da ontologia (OntologyModel). Contudo conforme as aplicações evoluem ou o entendimento sobre as técnicas da *Web* Semântica estejam mais claros para o projetista, outros produtos podem ser criados e associados à fábrica abstrata. Além de outras fábricas concretas que podem ser criadas. Na Figura 16, são apresentadas as classes abstratas para a fábrica e um produto (OntologyFactory e OntologyModel) assim como dois possíveis conjuntos de especializações dessas classes. Um lida com modelos RDF e o outro com modelos OWL-DL.

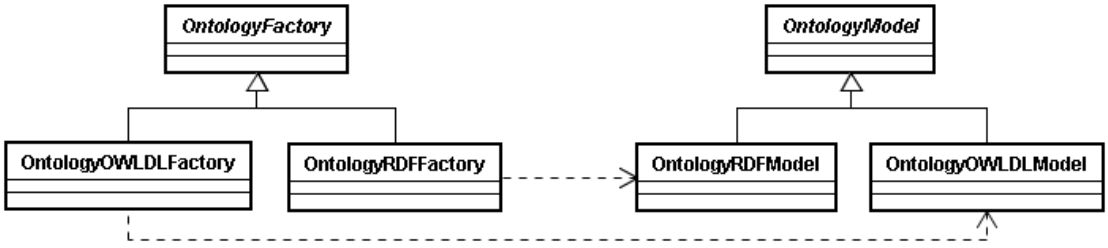


Figura 16 - Pacote *Ontology*

O uso do *Abstract Factory design pattern* para representar o modelo de ontologia aborda, de certa forma, os requisitos da controvérsia sobre a *Semantic Web stack* já que o projetista pode configurar sua aplicação para utilizar o seu entendimento e preferência quanto à versão da *Semantic Web stack*. O uso do padrão também permite que o spectrum de ontologias, apresentado na seção 2.1, seja considerado enquanto uma aplicação é desenvolvida.

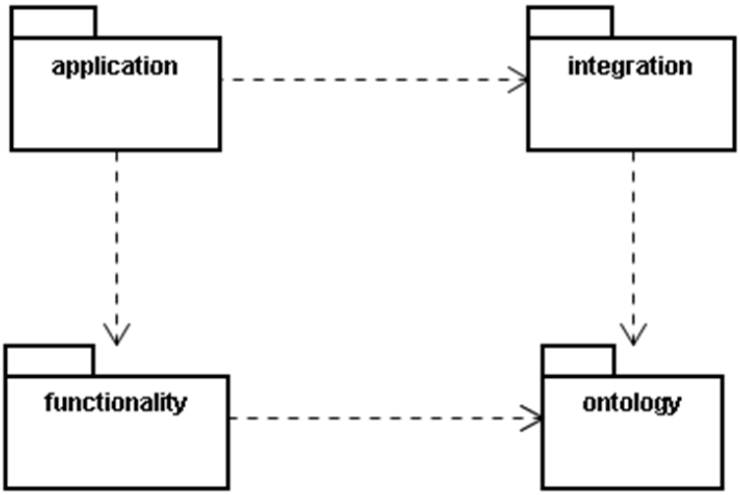


Figura 17 - Dependências entre pacotes

Finalmente, apresenta-se na Figura 17, um diagrama de classes que demonstra os pacotes que contém as classes de cada diagrama já apresentado nesta visão de desenvolvimento e como eles dependem uns dos outros. Como foi levantado nessa seção, há algumas questões a respeito das interações entre sub-sistemas que serão discutidos na próxima seção, a visão de processo.

### 5.3.4. A Visão de Processo (*Process View*)

A visão de processo pode ser descrita em diversos níveis de abstração, com cada nível tratando de uma questão ou tópico (*concern*) (Kruchten, 1995). Neste trabalho, a visão de processo trata como as principais entidades da visão de análise se encaixam no processo da arquitetura. Já foi mencionado que o Processo de Manipulação de Metadados, definido na seção 5.1.2, é importante para o *framework* e possivelmente para esta visão. Ele já foi incorporado ao *framework* através dos casos de uso incluídos no Cenário 1, apresentado na seção 5.3.1. Entretanto, conforme expressado na mesma seção, a generalidade daqueles casos de uso não é uma qualidade desejável para casos de uso bem estruturados.

Outra fonte de informação para a visão de processo são as interações entre sub-sistemas, conforme mencionado na visão de desenvolvimento. O número de possíveis interações entre sub-sistemas pode ser prolífico no nível de abstração tratado nesta seção, o arquitetural. Portanto esta discussão será adiada para o nível de *design* apresentado na seção 5.4.

### 5.3.5.
### A Visão Física (*Physical View*)

A visão física lida com questões sobre a implantação de uma SWAPp. O requisito do SWC de uma SWAPp ter ao menos duas fontes de informação distribuídas, de diferentes proprietários e heterogêneas pode, em parte, ser tratada nesta visão. Além disso, a partir da análise de domínio das submissões ao SWC, descobriu-se que um tipo específico de aplicação atende aquele requisito sem utilizar uma arquitetura cliente-servidor tradicional, a *Semantic Peer-to-Peer Application*.

Assim, dois diagramas de implantação bastante genéricos são apresentados para demonstrar uma aplicação cliente-servidor na *Web* (Figura 18) e uma rede de aplicações *Peer-to-Peer* (Figura 19).

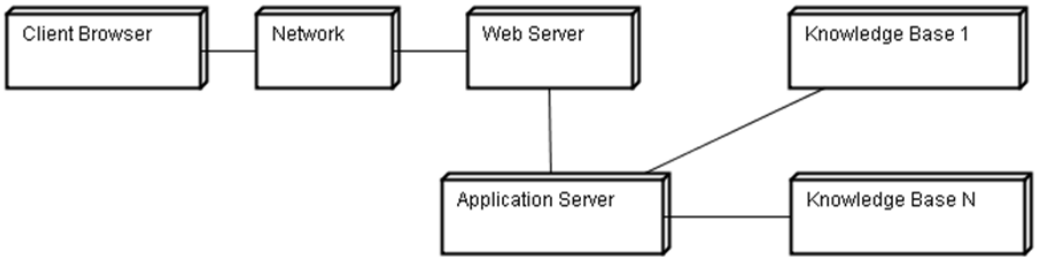


Figura 18 - Aplicação cliente-servidor na *Web*

Na Figura 18, apresenta-se um navegador cliente que é utilizado pelo usuário final para acessar através da rede um servidor *Web*. O servidor *Web* acessa um servidor de aplicação que se comunica com duas ou mais bases de conhecimento e, provavelmente, integra suas informações. O nome base de conhecimento foi escolhido para manter o diagrama genérico e, por exemplo, não restringir o desenvolvedor a usar um repositório de ontologias ou um servidor de bancos de dados.

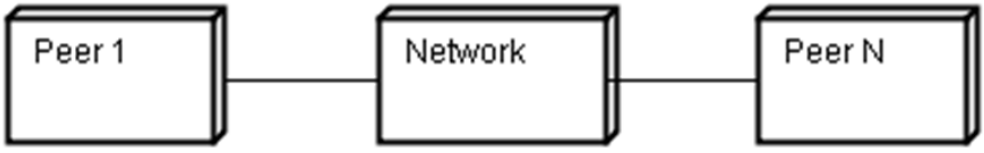


Figura 19 - Rede de aplicações P2P

Os dois diagramas de implantação não são as únicas opções de um desenvolvedor e é por isso que eles são apresentados de forma genérica neste trabalho. Outros tipos de implantação podem ser utilizados para aplicar estratégias de distribuição contanto que se garanta o requisito de a aplicação ter ao menos duas fontes de informação distribuídas, de diferentes proprietários e heterogêneas.

## 5.4.
## O *Design* do SWAPpFW

Nesta seção, discute-se como abordar a transição da arquitetura do SWAPpFW para o seu *design*. Apesar do domínio do SWAPpFW ser restrito pelos requisitos e qualidades desejáveis do SWC, como percebeu-se na definição dos cenários na arquitetura, essa restrição ainda é tênue e ampla. Portanto na próxima seção discute-se a "complexidade" da transição da arquitetura para o *design*, ou o número de possíveis configurações de *design*. Na seção seguinte, apresenta-se uma possível configuração de *design* e sua instanciação.

### 5.4.1.
### Possíveis Configurações de *Design* ("Complexidade")

Pela visão de análise, tem-se que uma aplicação oferece ao menos um tipo de funcionalidade e um tipo de integração de dados e que ambos utilizam ao menos um modelo de ontologia. Conforme apresentado na visão de desenvolvimento, 18 tipos de funcionalidades foram encontradas na análise de domínio das submissões ao SWC, entretanto 2 desses tipos são "abstratos" e não foram utilizados na análise de domínio (MultimediaFunctionality e OntologyFunctionality). Restam então 16 tipos de funcionalidades que uma aplicação pode oferecer.

Quaisquer dos tipos de funcionalidade podem ser combinados para compor uma aplicação e isso seria uma possível configuração de *design*. Contudo o uso isolado de alguma funcionalidade ou algumas combinações podem não caracterizar a aplicação como uma SWAPp. Tendo 16 tipos de funcionalidades e a redução daqueles tipos e combinações que não representam uma SWAPp, teríamos menos de $2^{16}$ (65.536) possíveis configurações de *design*. Os tipos de aplicação e de integração de dados também impõem restrições a esse número de possibilidades, mas ainda assim é uma quantidade consideravelmente alta de possibilidades.

Há portanto um número grande de possíveis configurações de *design*. Identificou-se alguns deles nas aplicações submetidas ao SWC. Contudo, o desenvolvedor não deve se restringir a eles. Já que para uma aplicação ser considerada uma SWAPp, neste trabalho, ela deve atender aos requisitos do SWC. Tentou-se tornar a definição de uma SWAPp o mais clara e precisa

possível. Mas se essa definição é tirada do contexto em que se encontra, corre-se o risco de se imaginar que apenas o agrupamento de algumas funcionalidades definidas no SWAPpFW por si geram uma SWAPp. Isto não é necessariamente verdade, o desenvolvedor pode terminar com uma aplicação que não atenda aos requisitos do SWC.

Das aplicações submetidas ao SWC, alguns tipos de aplicações eram combinações de funcionalidades. Por exemplo, definiu-se como *Portal*, as aplicações que ofereciam, no mínimo, a *Browse Functionality* e a *Search Functionality*. Na Tabela 7, apresentam-se as aplicações classificadas como *Portal* e suas funcionalidades. Escrutinando a tabela, encontra-se um subconjunto de 10 aplicações que, além das funcionalidades essenciais, oferecem também a Semantic *Search Functionality*. Outros subconjuntos ou combinações que podem ser vistos são os que oferecem a *Ontology Instances Editor Functionality*, a *Ontology Repository Functionality* e a *Multimedia Generation Functionality*.

Tabela 7 - Aplicações *Portal*

| **Application** / **Functionality - Type of Application - Integration** | SEAL | DOPE | SECO | Building Finder | CS AKTive Space | GeoShare | MusiDB | MADIERA Portal | SemanticOrganizer | Platypus Wiki | MuseumFinland | SPIA | Flink | Bibster | MOMIS | DynamicView | PPR | Oyster | CONFOTO | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Browse | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 19 |
| Generation of Navigational Views | 1 | | | | | | | | | | 1 | 1 | | | | | 1 | | | 4 |
| Dynamic and Semantic Linking Hypertext Structures | | | | | | | | | | | | | | | | | | | | 0 |
| Search | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | | 1 | 1 | 17 |
| Semantic Search | 1 | 1 | | 1 | 1 | 1 | | 1 | | 1 | 1 | 1 | | | | 1 | | | | 10 |
| Semantic Query Expansion | | 1 | | | | 1 | 1 | | | | | | | | | | 1 | | | 4 |
| Access through Diverse Devices | | | | | | | | | | | 1 | 1 | | | | | | | 1 | 3 |
| Support for Diverse Languages | 1 | | | | 1 | | | 1 | | | | 1 | | | | 1 | | | | 5 |
| Multimedia Handling | | | | | | | | | 1 | | | | | | | | | | | 1 |
| Multimedia Metadata | | | | 1 | 1 | 1 | | | | | 1 | | | | | | | | 1 | 5 |
| Multimedia Generation | | 1 | | | 1 | 1 | | | | | | 1 | 1 | | | 1 | | | | 6 |
| Semantic Growth | | | | | 1 | | | | 1 | | | | | 1 | | | | 1 | | 4 |
| Semantic Recommender Policy | | | | | | | 1 | | | | 1 | | | | | | | | | 2 |
| Ontology Schema Editor | | | | | | | | | | 1 | | | | | | | | 1 | | 2 |
| Ontology Instances Editor | | | | | | | | | 1 | 1 | | | | 1 | | | 1 | 1 | 1 | 6 |
| Ontology Repository | | | | | | | | | 1 | 1 | | | | 1 | | | 1 | 1 | 1 | 6 |
| Portal | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 19 |
| Ontology Tool | | | | | | | | | | 1 | | | | | | | | 1 | | 2 |
| Instance of a Framework | 1 | 1 | | | | | | | | | 1 | | | 1 | 1 | | 1 | 1 | | 7 |
| Semantic P2P Application | | | | | | | | | | | | | | 1 | | | | 1 | | 2 |
| Semantic Collaborative Tool | | | | | 1 | | | | | | | | | | | | | | | 1 |
| Semantic Wiki | | | | | | | | | | 1 | | | | | | | | | | 1 |
| Wrappers and Mediators Integration | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 17 |
| Manual Integration | | | | | | | | | 1 | 1 | | | | | | | | | | |

A

Tabela 8 apresenta as aplicações classificadas como *Ontology Tool*. Esse tipo de aplicação, conforme já definido, deveria oferecer a *Ontology Instances Editor Functionality*, a *Ontology Repository Functionality* e a *Ontology Schema Editor Functionality*. Contudo verifica-se, na

Tabela 8, que a *Search Functionality* está presente em quase todas as aplicações desse tipo.

Tabela 8 - Aplicações *Ontology Tool*

| Application / Functionality - Type of Application - Integration | Platypus Wiki | Unspecified Ontology (UNSO) | Annotea Shared Bookmarks | Oyster | |
|---|---|---|---|---|---|
| Browse | 1 | | | 1 | 2 |
| Search | 1 | 1 | | 1 | 3 |
| Semantic Search | 1 | | | | 1 |
| Semantic Query Expansion | | 1 | | | 1 |
| Access through Diverse Devices | | | 1 | | 1 |
| Semantic Growth | | | | 1 | 1 |
| Ontology Schema Editor | 1 | 1 | | 1 | 3 |
| Ontology Instances Editor | 1 | 1 | 1 | 1 | 4 |
| Ontology Repository | 1 | 1 | 1 | 1 | 4 |
| Portal | 1 | | | 1 | 2 |
| Ontology Tool | 1 | 1 | 1 | 1 | 4 |
| Instance of a Framework | | | | 1 | 1 |
| Semantic P2P Application | | 1 | | 1 | 2 |
| Semantic Wiki | 1 | | | | 1 |
| Wrappers and Mediators Integration | | | | 1 | 1 |
| Manual Integration | 1 | | 1 | | 2 |

Outros tipos de aplicação não foram definidos como uma combinação de funcionalidades. Por exemplo o tipo *Instance of a Framework*, as aplicações desse tipo deveriam instanciar ou reutilizar funcionalidades oferecidas por *frameworks*. Na

Tabela 9, apresentam-se aplicações desse tipo e percebe-se que há uma "dispersão" nas funcionalidades oferecidas pelas aplicações. A maioria das aplicações são *Portal*, mas também há *Ontology Tool* e *Semantic P2P Application*.

Tabela 9 - Aplicações *Instance of a Framework*

| Application / Functionality - Type of Application - Integration | SEAL | SECO | Semblog | MuseumFinland | Bibster | MOMIS | GOHSE | PPR | Oyster | |
|---|---|---|---|---|---|---|---|---|---|---|
| Browse | 1 | 1 | | 1 | 1 | 1 | | 1 | 1 | 7 |
| Generation of Navigational Views | 1 | | | 1 | | | | 1 | | 3 |
| Dynamic and Semantic Linking Hypertext Structures | | | | | | | 1 | | | 1 |
| Search | 1 | 1 | | 1 | 1 | 1 | | | 1 | 6 |
| Semantic Search | 1 | 1 | | 1 | | | | | | 3 |
| Semantic Query Expansion | | 1 | | | | | 1 | 1 | | 3 |
| Access through Diverse Devices | | | | 1 | | | | | | 1 |
| Support for Diverse Languages | 1 | | | | | | | | | 1 |
| Multimedia Metadata | | | | 1 | | | | | | 1 |
| Multimedia Generation | | 1 | | | | | | | | 1 |
| Semantic Growth | | | | | 1 | | | | 1 | 2 |
| Semantic Recommender Policy | | | 1 | 1 | | | | | | 2 |
| Ontology Schema Editor | | | | | | | | | 1 | 1 |
| Ontology Instances Editor | | | 1 | | 1 | | | 1 | 1 | 4 |
| Ontology Repository | | | 1 | | 1 | | | 1 | 1 | 4 |
| Portal | 1 | 1 | | 1 | 1 | 1 | | 1 | 1 | 7 |
| Ontology Tool | | | | | | | | | 1 | 1 |
| Instance of a Framework | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 9 |
| Semantic P2P Application | | | | | 1 | | | | 1 | 2 |
| Semantic Collaborative Tool | | | 1 | | | | | | | 1 |
| Wrappers and Mediators Integration | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| Manual Integration | | | 1 | | | | | | | 1 |

Classificou-se 3 aplicações como *Semantic P2P Application*, mas 2 delas são instâncias de um mesmo *framework*. E isso reflete nas funcionalidades oferecidas por aquele tipo de aplicação. Entretanto, como explicado antes, o tipo *Semantic P2P Application* não é um tipo de aplicação relacionado ao agrupamento de um conjunto de funcionalidades. Ele está mais relacionado com a forma como a arquitetura (física) da aplicação foi projetada e como ela usa os metadados com o suporte de técnicas e ferramentas P2P.

Tendo tudo isso em vista, na próxima seção descreve-se uma possível configuração de *design* que gera uma SWAPp e portanto é uma combinação válida de funcionalidades, suas instâncias e uma possível implementação.

### 5.4.2.
### Uma Combinação Válida de Funcionalidades

Considere um tipo de aplicação (T1) que oferece as seguintes funcionalidades:

- *Browse Functionality*;
- *Search Functionality*;
- Semantic *Search Functionality*; e
- *Multimedia Generation Functionality*.

Classificar-se-ia esse T1 como um *Portal*. Considere também que T1 integra dados de pesquisadores e suas publicações. Voltando-se às aplicações submetidas às 3 edições do SWC, há algumas aplicações que são *Portal*, oferecem estas funcionalidades e que integram este tipo de dados:

- 2003: CS AKTive Space (Shadbolt *et al.*, 2004);
- 2004: Flink (Mika, 2005a); e
- 2005: DynamicView (Gao *et al.*, 2005).

Algumas das aplicações apresentadas oferecem mais funcionalidades. Outras, como é o caso de Flink, tem funcionalidades potenciais: como Flink é implementado usando Sesame (Broekstra *et al.*, 2002), ele poderia oferecer a *Search Functionality* e a Semantic *Search Functionality*. Apesar de algumas das aplicações apresentadas terem as suas peculiaridades, ainda assim podemos dizer que elas são instâncias de T1 e T1 seria então uma combinação válida de funcionalidades.

Uma implementação de T1 seria o Elmo[42]. Elmo é uma API Java para SWAPps. Ela é *open source* e distribuída sob a LGPL *license*[43]. Grande parte do código de Elmo apareceu originalmente na aplicação Flink, a vencedora do SWC 2004.

Elmo é implementado usando (*stands on top of*) as facilidades de armazenamento e consulta do Sesame. Ele fornece suporte para o desenvolvimento de SWAPps usando ontologias populares como FOAF, RSS 1.0 e Dublin Core. O modelo de objetos do Elmo segue uma abordagem simples: cada conceito ontológico tem uma classe Java correspondente na biblioteca (API) usando o mesmo nome. O seu modelo estático de objetos pode ser estendido para acomodar novos conceitos, relacionamentos ou ontologias. O

[42] openRDF.org - http://www.openrdf.org/ - acesso em: 28/10/2006

Elmo também oferece algumas ferramentas para trabalhar com ontologias como por exemplo um RDF *crawler* e um *smusher* para dados FOAF (Mika, 2005b).

## 5.5. Resumo

Neste capítulo, um *framework* de aplicações para a *Web* Semântica (SWAPpFW) foi apresentado. Primeiro, elicitou-se os requisitos do *framework* que foram baseados no requisitos e qualidades desejáveis do SWC para uma SWAPp. Esses requisitos foram classificados como funcionais e não-funcionais. Além disso, definiu-se o Processo de Manipulação de Metadados que foi identificado durante a revisão das aplicações submetidas ao SWC e é uma contribuição deste trabalho. É importante reafirmar que, apesar de não aparecer como um elemento do processo, inferências (*reasoning*) são parte dele. Escolheu-se representar as inferências (*reasoning*) como uma interação entre a *Metadata Storage phase* e *Metadata Usage phase*. Esta escolha foi feita por que as inferências (*reasoning*) dependerão das heurísticas escolhidas para as mesmas e para as atualizações dos metadados já coletados e armazenados.

Após os requisitos, propôs-se a arquitetura do SWAPpFW usando o *"4+1" View Model of Software Architecture* ajustado. As visões e discussões geradas durante esta atividade também são uma contribuição deste trabalho. Baseado nas visões, discutiu-se a dificuldade da transição entre a arquitetura do SWAPpFW e seu *design*. Contudo, um exemplo de solução foi apresentado mesmo não sendo uma solução de melhor-caso.

Os requisitos do *framework* e o próprio *framework* foram influenciados pela escolha do uso das submissões ao SWC. Um exemplo claro disso são a *Use of Multimedia Documents Functionality* e suas sub-funcionalidades. O uso do conteúdo de documentos multimídia é um requisito explícito do SWC mas ainda assim é amplo, como pôde ser visto pelas diferentes funcionalidades derivadas dele. O SWAPpFW e, conseqüentemente, as funcionalidades e tipos de aplicação dependem fortemente dos requisitos do SWC e da aplicações revisadas. Contudo o *framework* ainda é genérico o bastante para acomodar novas funcionalidades para lidar com outros aspectos de SWAPps, por exemplo alinhamento de ontologias, tratamento de metadados geo-referenciados ou de Bioinformática.

[43] GNU Lesser General Public License (LGPL) - http://www.gnu.org/licenses/lgpl.html - acesso em: 28/10/2006