**Capítulo 2**

**Conceitos Básicos**

Com o objetivo de firmar o embasamento teórico do projeto e do aprimoramento do InteropFrame, será elucidado neste capítulo os conceitos que foram utilizados no decorrer da pesquisa. Alguns deles foram utilizados no aprimoramento do *framework*, como também na sua construção. Conceitos como Modularização, Invocação Remota de Métodos, Geração Automática de Código, Componentes Distribuídos, entre outros. Por fim, um breve relacionamento entre esses conceitos e o aprimoramento do *framework* proposto nesse trabalho.

2.1 Modularização

Modularização significa a concepção de um sistema completo formado por módulos logicamente independentes {Hall2011}. Ela é capaz de reduzir a complexidade do problema, dividindo-o em subproblemas mais simples, ou seja, dividindo-o em módulos {Knuth1996}. Um módulo define um limite lógico executável. Dessa forma os detalhes internos dele não são visíveis a outros módulos ou sistemas. Os únicos detalhes visíveis são aqueles que ele expõe explicitamente, ou seja, a API pública {Hall2011}. Com isso é mais fácil de detectar problemas e resolvê-los, pois os módulos são, em princípio, independentes. Em sua característica um módulo é implementável, gerenciável, reutilizável, combinável e é uma unidade independente de software que provê interfaces a outros módulos ou sistemas {Kirk2012}.

É confuso o conceito de Modularização e Orientação a Objeto. As duas suportam a especialização, ou seja, elas quebram o sistema em partes pequenas dando a cada uma delas a sua devida responsabilidade. Entretanto elas atuam de formas diferentes. Com a Orientação a Objeto é possível modularizar de forma lógica, ou seja, essa modularização se referencia a visibilidade do código. Dessa forma a Orientação a Objeto utiliza parte da Modularização em seu contexto. Entretanto a Modularização abrange mais que isso. Ela pode ser utilizada tanto da forma física, onde é possível subdividir o código em vários arquivos, entretanto mantendo as mesmas dependências e comunicação entre eles, quanto da forma lógica, como na Orientação a objeto {Hall2011}.

A modularização ganhou a popularidade no início da década de 70. Entretanto é algo que ainda hoje não está tão presente nos requisitos não funcionais do desenvolvimento de software. Apesar disso é algo tão importante que traz grandes benefícios para a aplicação. Para isso é preciso aplicar os princípios de modularização para obter bons resultados. Princípios esses como alta coesão e baixo acoplamento. Com eles, uma das grandes vantagens que a modularização oferece é a reutilização. Isso se torna fácil quando um módulo é responsável por aquilo que realmente deve ser e não tem um grande número de dependências. A depender do *framework* utilizado, para a aplicar a modularização, a declaração das dependências pode ser feita de forma explícita como no OSGI. O que traz um ganho para a manutenção no código e para o melhor entendimento do mesmo {Hall2011}.

2.1.1 Componentes e Modelos de Componentes

A engenharia de software baseada em componentes surgiu como uma abordagem para softwares de desenvolvimento de sistemas com base no reúso de componentes de softwares {sommerville2011}. Não existe consenso sobre um componente ser uma unidade independente de software que pode ser composta com outros componentes {sommerville2011}. Segundo {councill2001 apud sommerville2011}, componente é um elemento de software que está de acordo com um modelo de componente padrão e pode ser independentemente implantado e composto de acordo com um padrão de composição. Entretanto, {szyperski2002 apud sommerville2011} menciona que um componente de software é uma unidade de composição de interfaces contratualmente especificadas e pode ser implantado de forma independente, além de estar sujeito a ser composto por parte de terceiros.

Um componente funciona como um provedor de um ou mais serviços. Dessa forma, quando um sistema precisa de um serviço, ele chama um componente para fornecer esse serviço sem se preocupar sobre onde esse componente está sendo executado, nem mesmo de características como linguagem de programação que o componente foi desenvolvido {sommerville2011}. Para isso, os componentes possuem duas interfaces relacionadas que refletem os serviços que o componente fornece (interface *provides*) e os serviços de que o componente necessita (interface *requires*), como mostra a Figura {figuras/001 – Interfaces de componentes.png} {sommerville2011}.

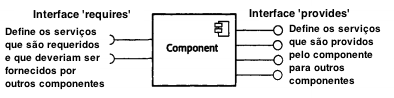


Figura {figuras/001 – Interfaces de componentes.png} – Interfaces de componentes {sommerville2011}

Um modelo de componente é uma definição de normas para implementação, documentação e implantação de componentes que garantem a interoperabilidade deles {sommerville2011}. Existem diversos modelos de componentes, entretanto neste trabalho serão abordados os modelos OpenCOM, Fractal e OSGI.

2.1.1.1 Modelo OpenCOM

O OpenCOM é um modelo de componentes de baixo peso projetado para o desenvolvimento de middlewares em dispositivos de computação com poucos recursos (processamento, memória, armazenamento). Além de ser um modelo de baixo peso, o OpenCOM provê a capacidade de reconfiguração dinâmica de middlewares tanto no domínio estrutural quanto no comportamental {rocha2008}.

O OpenCOM é fundamentado em três tecnologias {nascimento2013}:

- Componentes - O modelo permite a especificação da estrutura de sistemas através do uso de componentes e conexões entre componentes (ROCHA, 2008).

Os conceitos fundamentais no OpenCOM são interfaces, receptáculos e conexões. Uma interface representa uma unidade de provisão de serviços, enquanto que um receptáculo representa uma unidade de requerimento de serviços e é usado para tornar explícita a dependência de uma interface de um componente com outra. Uma conexão representa uma ligação entre um serviço fornecido por uma interface de um componente e um serviço requerido por um receptáculo de outro componente {clarke2001};

- Reflexão computacional - O modelo OpenCOM foi projetado para suportar a reflexão computacional - que é a capacidade que um sistema tem de observar sua própria representação/estrutura e modifica-la em tempo de execução;

- Frameworks de componentes – Uma característica chave do OpenCOM é o uso da noção de frameworks de componentes. Um framework de componentes é definido no OpenCOM como um conjunto fortemente acoplado de componentes que coopera para resolver alguma área de interesse. O OpenCOM também fornece um protocolo de extensão bem definido para a aceitação de componentes adicionais que modificam ou estendem o comportamento do framework de componentes, além de restringir o modo como os componentes são organizados {coulson2008}.

Cada componente OpenCOM implementa quatro interfaces, além de interfaces personalizadas, como mostra a Figura {002opencom.png} {grace2007 apud nascimento2013}:

- ILifeCycle - fornece as operações para a inicialização e a finalização do ciclo de vida de um componente;

- IConnections (opcional) - oferece os métodos para modificar as interfaces ligadas aos receptáculos de um componente. Esta interface deve ser implementada por todos os componentes que possuem receptáculo;

- IMetaInterface – suporta a inspeção dos tipos de interfaces e receptáculos declarados pelo componente;

- IUnknown - é equivalente à interface do mesmo nome no Microsoft COM, isto é, é usada para obter a referência para a interface solicitada na instância do componente.

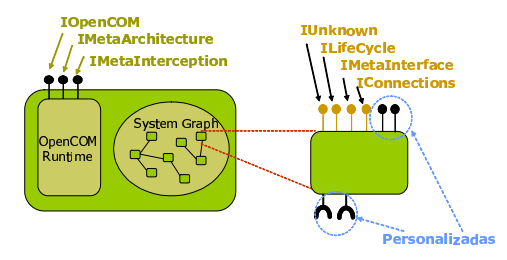


Figura {figuras/002opencom.png} – Interfaces dos metamodelos do OpenCOM {rocha2008}

O OpenCOM implanta um substrato padrão em *runtime* que está disponível em todo o espaço de endereçamento. Insto é implementado através de um componente *singleton* chamado "OpenCOM” que exporta uma interface chamada IOpenCOM. O papel do *runtime* do OpenCOM é o de gerir um repositório de componentes disponíveis e, assim, permitir a criação e exclusão de componentes. Além disso, a interface IOpenCOM serve como um ponto centralizado para a submissão de todas as solicitações de conexão ou desconexão entre receptáculos e interfaces no seu espaço de endereçamento. Para facilitar a reconfiguração, o *runtime* registra cada criação e exclusão de cada componente ou conexão em um espaço de meta-estrutura chamado de *system graph* (grafo do sistema). Isto permite que o OpenCOM suporte consultas que, através de um identificador de conexão, fornece detalhes sobre o receptáculo e as interfaces participantes da conexão, juntamente com detalhes de seus componentes que os implementam {clarke2001}.

2.1.1.2 Modelo Fractal

O Fractal é definido em {bruneton2006} como um modelo de componentes geral e extensível, projetado para implementar, implantar e gerenciar sistemas de *software* complexos, incluindo, em particular, sistemas operacionais e *middlewares*. As principais motivações do modelo são: (i) Composição de componentes, onde um componente pode conter outros componentes, permitindo uma visão uniforme das aplicações em vários níveis de abstração; (ii) Compartilhamento de componentes entre estruturas compostas de componentes, como forma de compartilhar recursos enquanto se mantém o encapsulamento de um componente; (iii) Capacidades reflexivas, para monitorar e controlar um sistema em execução; (iv) Capacidades de reconfiguração, como forma de implantar e configurar dinamicamente um sistema.

De acordo com {coupaye2007}, o modelo de componentes Fractal suporta várias linguagens de programação, como por exemplo, Java e C, e de forma experimental .NET, SmallTalk, Python e C++.

No contexto do Fractal os **componentes** são entidades de *runtime* que estão em conformidade com o modelo, estão encapsuladas, possuem identificações únicas e suportam uma ou mais interfaces. As **interfaces** são os pontos únicos de interação entre os componentes e expressam a dependência desses em termos de interfaces requeridas e providas. Os ***bindings*** são os canais de comunicação entre as interfaces dos componentes {coupaye2007} {bruneton2006}.

Um componente Fractal é a composição de uma **membrana** e um **conteúdo**, como pode ser observado na Figura {figuras/004fractal.png}. A membrana tem o papel de fornecer interfaces para um controle reflexivo sobre o conteúdo. O conteúdo consiste num conjunto finito de outros subcomponentes. A membrana de um componente pode ter interfaces internas, acessíveis somente pelos subcomponentes internos, e externas, acessíveis de fora do componente. Além disso, uma membrana possui diversas interfaces de controle, que atuam como interceptadores entre as operações de chamada que entram e saem do componente, e adicionam comportamentos aos manipuladores de tais operações {bruneton2007}.

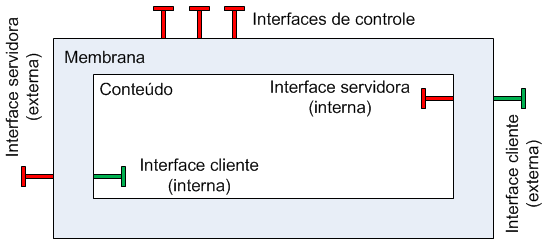


Figura {figuras/004fractal.png} – Estrutura de um componente Fractal {bruneton2006 apud nascimento2013}.

2.1.1.3 Modelo OSGI

Segundo {osgi}, o OSGi é um conjunto de especificações que definem um sistema de componentes dinâmico para o Java. Com essas especificações é possível criar um sistema composto dinamicamente por diversos componentes reusáveis. O OSGi permite que os componentes escondam suas implementações de outros componentes enquanto se comunicam através de **serviços**. Os serviços são objetos compartilhados de maneira especifica entre componentes.

A arquitetura do OSGi é composta por camadas como mostra a Figura {figuras/005osgi.png}. Elas são brevemente descritas a seguir {osgi}:

- *Bundles* – OS *bundles* são os componentes OSGi implementados pelos desenvolvedores.

- *Services* – A camada de Serviços conecta os *bundles* de maneira dinâmica. Os serviços são publicados pelos *bundles*, e são passíveis de busca e conexão posteriormente.

- *Life-Cycle* – Parte da API do OSGi que permite a instalar, desinstalar, executar, parar, e atualizar *bundles*.

- *Modules* – Camada que define como um *bundle* pode importar e exportar código.

- *Security* – Camada que manipula os aspectos de segurança do OSGi.

- *Execution Environment* – É o ambiente de execução. Define quais métodos e classes estarão disponíveis na plataforma específica.

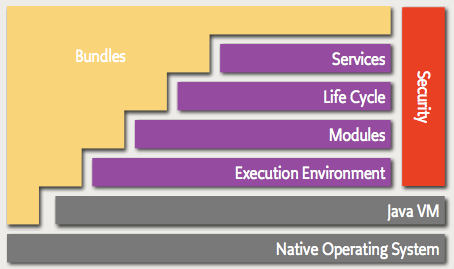


Figura {figuras/005osgi.png} – Camadas da arquitetura OSGi {osgi}.

O OSGi é apenas uma especificação de um framework para o desenvolvimento de aplicações modulares. Existem diversas implementações dessa tecnologia, sendo as mais conhecidas: Equinox, Felix e Knopflerfish. A primeira é uma implementação da especificação OSGi desenvolvida pela Eclipse Foundation. Ela é utilizada em diversas aplicações, inclusive na IDE Eclipse. Já a implementação Felix é desenvolvida e mantida pela Apache Software Foundation. A implementação Knopflerfish é desenvolvida e mantida pela Makewave.

2.2 Processo CBSE

2.2.1 CBSE para reúso

2.2.2 CBSE com reúso

2.3 Sistemas Distribuídos

Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens {Coulouris2007}. {Tanenbaum2007} definem um sistema distribuído como uma coleção de computadores independentes que aparece para o usuário como um único sistema. {Sommerville2011} menciona que os sistemas distribuídos são mais complexos que os sistemas centralizados, o que os torna mais difíceis de projetas, implementar e testar. Apesar dessa complexidade, praticamente todos os grandes sistemas computacionais são distribuídos.

{Sommerville2011 apud Coulouris2007} identifica vantagens da utilização de uma abordagem distribuída no desenvolvimento de sistemas:

* Compartilhamento de recursos: Um sistema distribuído permite o compartilhamento de recursos de hardware e software.
* Abertura: Os sistemas distribuídos são projetados para protocolos-padrão que permitem que os equipamentos e software de diferentes fornecedores sejam combinados.
* Concorrência: Em um sistema distribuído, vários processos podem operar simultaneamente em computadores separados na rede.
* Escalabilidade: Em princípio, os recursos de um sistema distribuído podem ser aumentados pela adição de novos recursos a depender da necessidade do sistema.
* Tolerância a defeitos: Um sistema distribuído pode ser tolerante a algumas falhas de hardware e software dispondo de vários computadores e replicando as informações importantes para o sistema.

2.4 Modelos de Interação

“A RMI é uma extensão da invocação a método local que permite a um objeto que está em um processo invocar os métodos de um objeto que está em outro processo.” Coulouris, Dollimore e Kindberg (2007). Ou seja, a Invocação Remota de Métodos (RMI) permite o estabelecimento de comunicação entre objetos remotos distribuídos. Objetos esses que estão dispostos em processos diferentes ou mesmo em máquinas distribuídas geograficamente. Dessa forma esses objetos implementam uma interface remota. O que pode ser considerada a porta por onde esses objetos se comunicam. Com elas é possível controlar as interações entre os módulos e especificar os procedimentos e variáveis que podem ser acessadas a partir de outro módulo. Entretanto quem os utiliza não necessita do conhecimento de como o método está implementado, muito menos onde está localizado. Apenas o utiliza por meio das interfaces. Logo nota-se que a RMI é baseada na arquitetura Cliente-Servidor. Ou seja, de um lado está a máquina Cliente que solicita ou invoca algo e do outro lá está a máquina Servidor. Sendo assim no lado Servidor são instanciados objetos remotos em uma porta e são referenciados com um nome. Dessa forma esses objetos esperam por invocações a métodos oriundos das máquinas Clientes que esperam pela resposta do solicitado.

A arquitetura da RMI é subdividida em quatro camadas: Aplicação, Stub e Skeleton, Referência Remota e Transporte. A camada de aplicação localiza-se a implementação do módulo. Seja ele cliente ou servidor é nessa camada que encontra-se o objeto remoto. Logo após está a camada Stub e Skeleton. Nela é implementada as interfaces que os objetos remotos utilizam para interagir entre si. É nela que o lado Servidor e o lado Cliente são devidamente representados. Em seguida encontra-se a camada Referência Remota. Ela é considerada um *middleware* entre a camada de Stub/Skeleton e a camada de Transporte. Nela são criadas e gerenciadas as referências remotas aos objetos. Por fim encontra-se a camada de Transporte. Essa oferece o protocolo que envia solicitações aos objetos remoto pela rede.

2.4.1 Middleware

2.5 Arquitetura de Componentes Distribuídos

2.6 Serviços Web

Um serviço web (web service) fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores web {Coulouris2007}. De acordo com {Deitel2010}, um serviço web é um componente de software armazenado em um computador que pode ser acessado por um aplicativo(ou outro componente de software) em outro computador por uma rede. Um serviço web possui uma interface descrita em um formato processável por máquina, especificamente a WSDL (Web Services Definition Language). Outros sistemas interagem com um *Web Service* utilizando mensagens de acordo com um padrão, tipicamente utilizando HTTP com uma serialização de XML, além de outros padrões relacionados a *Web*.

Os clientes acessam as operações na interface de um serviço web por meio de requisições e respostas formatadas em XML (Extensible Markup Language) e, normalmente, transmitidas por HTTP (HyperText Transfer Protocol) {W3c2004}. A XML é uma representação textual que, embora mais volumosa do que as representações alternativas, foi adotada por sua legibilidade e pela consequente facilidade de depuração {Coulouris2007} De acordo com {Deitel2010}, o serviço web pode ser apoiado em duas arquiteturas. A primeira é baseada no Simple Object Access Protocol (SOAP) e a segunda é baseada no Representational State Transfer (REST).

Segundo {Sommerville2011}, SOAP é um padrão de trocas de mensagem que oferece suporte à comunicação entre serviços. O SOAP é um protocolo independente de plataforma que utiliza a XML para fazer chamadas de procedimento remoto, geralmente sobre o HTTP {Deitel2010}. De acordo com {Coulouris2007}, originalmente o protocolo SOAP era baseado apenas em HTTP, mas a versão atual é projetada para usar uma variedade de protocolos de transporte, incluindo o SMTP, TCP ou UDP.

Segundo {Deitel2010}, o REST refere-se a um estilo arquitetônico de implementar serviços Web. Rest é uma estratégia com um estilo de operações muito restrito, no qual os clientes usam URLs e as operações HTTP, GET, PUT, DELETE e POST para manipular recursos representados em XML {Fielding2000}. Segundo {Deitel2010}, o REST também não está limitado a retornar dados no formato XML. Ele pode utilizar vários formatos, como XML, JSON, HTML, texto sem formatação e arquivos de mídia.

2.7 Geração Automática de Código

Um gerador de código é um sistema desenvolvido para criar automaticamente código fonte de alto nível em linguagens de programação como .NET, C++, C#, Java e outros (ADAMATII, 2006). A geração automática de código ajuda a aumentar a eficácia da produção de software complexo, reduzindo o custo e tempo associado com o esforço de codificação (KORNECKI; JOHRI, 2006).

Algumas plataformas utilizam mecanismos ou ferramentas para automação de geração de código para melhorar a produtividade e eficiência nos processos de desenvolvimento de software, como por exemplo, O MDA (Model-driven Architecture) (OMG, 2012) - que é uma abordagem para desenvolvimento de sistemas dirigido a modelos e o Acceleo (ACCELEO, 2012) - que é um plugin do Eclipse que baseado em MDA permite a geração automática de código a partir de modelos, como por exemplo, um modelo UML ou um metamodelo definido pelo usuário.

2.8 Discussão

A proposta de extensão e modularização do InteropFrame exige conhecimentos diversos que foram elencados nesse capítulo. Mais especificamente foi observado o conceito de modularização, como forma de reconstruir o InteropFrame de forma modular utilizando a distribuição Equinox do OSGi. Outro conceito a ser utilizado neste trabalho é o da implementação do Web Service do tipo REST no módulo de configuração distribuída do InteropFrame, que será abordado na seção 3.