**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 engloba muitos aspectos da programação que temos concebido. A maior experiência que temos é com projetos de sistemas. Normalmente os melhores tendem a ter a característica modular. Em resumo, modularização significa a concepção de um sistema completo formado por módulos logicamente independente.” Hall, 2011. Nesse contexto modularização nada mais é que a fragmentação do todo em partes independentes (módulos) que se relacionam entre si. É algo que está embutido no dia a dia de qualquer desenvolvedor de software que utiliza a arquitetura em camadas ou mesmo programação orientada a objeto. Podemos dizer que cada camada pode ser representada por um módulo. Assim como uma classe pode ser considerada também um módulo a depender do ponto de vista.

É confuso o conceito de Modularização e Orientação a Objeto. Os dois suportam a especialização, ou seja eles quebram o sistema em partes pequenas dando a cada uma delas a sua devida responsabilidade. Entretanto os dois atuam de formas diferentes. Com a Orientação a Objeto é possível modularizar de forma lógica, ou seja, todas as suas dependências encontram-se no mesmo arquivo ou código. Dessa forma a Orientação a Objeto utiliza parte da Modularização em seu contexto. Entretanto a Modularização abrande mais que isso. Ela pode ser utilizada tanto da forma física quanto da forma lógica. Essa é feita como na Orientação a objeto. Aquela é possível subdividir o código em vários arquivos de forma física, entretanto mantendo as mesmas dependências e comunicação entre eles.

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 trás grandes benefícios para a aplicação. Entretanto 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 trás um ganho para a manutenção no código e para o melhor entendimento do mesmo.

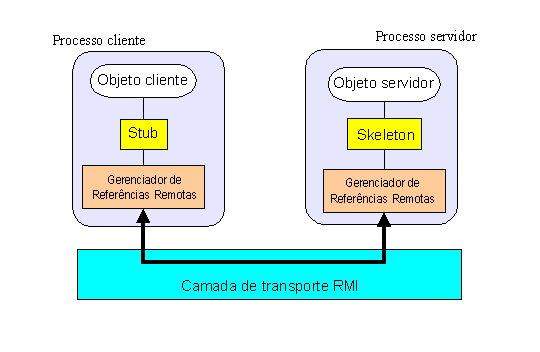
2.1.1 OSGI

Criadora da tecnologia OSGI, a OSGI Alliance foi criada por empresas como Eclipse, Apache, Red Hat, IBM, Oracle, entre diversas outras com o objetivo de criar especificações abertas que auxiliem na construção modular de um software. Dessa forma a tecnologia OSGI define um modelo de modularização para as aplicações Java. Ela facilita a modularização e garante a interoperabilidade de aplicações e serviços, como também facilita o gerenciamento remoto sobre uma ampla variedade de dispositivos. Além disso ela aumenta a produtividade do desenvolvimento de software e facilita a sua manutenção e evolução. Entretanto OSGI é apenas uma especificação. Existem diversas implementações dessa tecnologia, mas as mais conhecidas são: Equinox e Felix. A primeira é uma implementação da especificação OSGI desenvolvida pelo Eclipse. Ela é utilizada em diversas aplicações, inclusive no ambiente de desenvolvimento Eclipse. Já a outra implementação é desenvolvida pela Apache.

2.2 Invocação Remota de Métodos

“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. Conforme a figura X. 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.



Arquitetura em Camadas da RMI (http://www.dca.fee.unicamp.br/cursos/PooJava/objdist/rmiarq.html).

2.3 Serviços Web

Um *Web Service* é um sistema de *software* projetado para suportar a interação interoperável máquina a máquina sobre uma rede. Possui uma interface descrita em um formato processável por máquina, especificamente WSDL (Web Services Definition Language). Outros sistemas interagem com um *Web Service* utilizando mensagens de acordo com um padrão, tipicamente utilizando HTTP (HyperText

Transfer Protocol) com uma serialização de XML (Extensible Markup Language), além de outros padrões relacionados a *Web* (W3c, 2004). <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>.

Um serviço web fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores web (COULOURIS; DOLLIMORE; KINDBERG, 2007 apud NASCIMENTO, 2013).

2.4 Geração Automática de Código

2.5 Componentes Distribuídos

2.5.1 Middleware Baseado em Componentes

2.5.2 O Modelo OpenCOM

2.5.3 O Modelo Fractal

2.5.4 O Modelo OSGI

2.6 Discussão