**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 o módulo 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. 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 {Hall2012}.

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 {Hall2012}.

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 X {sommerville2011}.

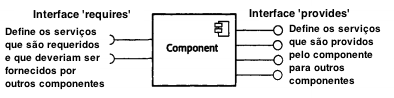


Figura X – Interfaces de componentes {sommerville2011}

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

2.1.1.2 Modelo Fractal

2.1.1.3 Modelo 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 Processo CBSE

2.2.1 CBSE para reúso

2.2.2 CBSE com reúso

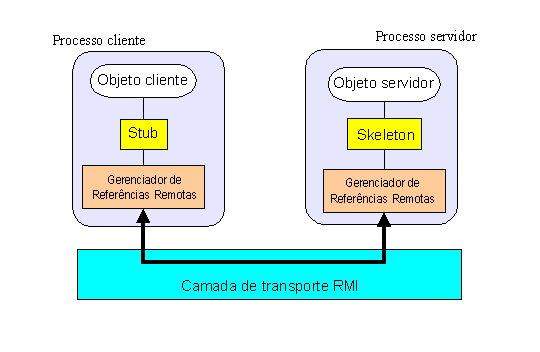
2.3 Sistemas Distribuídos

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. Dessa forma {Coulouris2007} identificam vantagens da utilização de uma abordagem distribuída de desenvolvimento de sistemas: Compartilhamento de recursos, Abertura, Concorrência, Escalabilidade e Tolerância a defeitos.

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. 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).

2.4.1 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 (COULOURIS; DOLLIMORE; KINDBERG, 2007). De acordo com Deitel e Deitel (2010), 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. Os clientes acessam as operações na interface de um serviço web por meio de requisições e respostas formatadas em XML e, normalmente, transmitidas por HTTP (COULOURIS; DOLLIMORE; KINDBERG, 2007). 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 (COULOURIS; DOLLIMORE; KINDBERG, 2007). De acordo com Deitel e Deitel (2010), 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 Sommerville (2011), 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 (DEITEL; DEITEL, 2010). De acordo com Coulouris, Dollimore e Kindberg (2007), 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 Deitel e Deitel (2010), 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 (FIELDING, 2000). Segundo Deitel e Deitel (2010), 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.

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).

2.7 Geração Automática de Código

2.8 Discussão