Desenvolvimento de Software Baseado em Componentes Usando IDL

Yury A. Lima[1], Giliardi Schmidt[1], Elder Rodrigues[1], Maicon Bernardino[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

yuryalencar19@gmail.com, gili.schmidt@hotmail.com
elderrodrigues@unipampa.edu.br, bernardino@acm.org

Abstract. *This article adresses the problems encountered in the characterization of software components, and proposes and improvment using a description language, IDL (Interface Description Language). It is a technology that has as its main objective the description of interfaces at the programing level, making it easier for developers to understant. Its application togheter with Component Based Development significantly impacts development time, plus assisting in the implementation of the components, as well as in the creation of a Middleware for the communication of these components, even providing the means for a hybrid software.*

Resumo. *Este artigo aborda os problemas encontrados na caracterização de componentes de software, e propõe uma melhoria nesta descrição utilizando uma linguagem de descrição. IDL (Interface Description Language) é uma tecnologia que tem como o maior objetivo a descrição de interfaces a nível de programação, facilitando o entendimento do desenvolvedor. Sua aplicação conjunta com o Desenvolvimento de Software Baseado em Componentes impacta significativamente no tempo de desenvolvimento, além de auxiliar tanto na implementação dos componentes, quanto na criação de middlewares para comunicação dos mesmos, possibilitando até a confecção de um software híbrido.*

1. Introdução

O desenvolvimento de software tem como intuito informatizar processos, provendo soluções práticas que possam ser realizadas em um pequeno espaço de tempo. Para isso, várias formas de otimizar o desenvolvimento foram criadas com o objetivo de diminuir os custos, tempo de produção e complexidade, além de colaborar na manutenibilidade, segurança e reúso dos artefatos implementados. O Desenvolvimento de Software Baseado em Componentes (DSBC) [DeRemer and H. Kron 1976] é uma alternativa fundamentada no princípio da divisão e conquista, onde é feita uma separação das responsabilidades na forma de componentes independentes de software. Essa abordagem proporciona um melhor entendimento de cada responsabilidade/funcionalidade e diminuindo o esforço na programação. Além disso, a adoção de DSBC aumentando a qualidade do software já que os componentes depois de desenvolvidos e testados podem ser reutilizados e evoluídos em outros projetos.

Durante o projeto desses componentes deve ser realizado um detalhamento por meio de Modelos de Componentes e uma especificação das *interfaces* a serem providas,

sendo estas etapas cruciais para o entendimento e qualidade dos componentes a serem desenvolvidos. Entretanto, esta especificação pode ser extensa e acarretar dificuldades ao desenvolvedor como por exemplo, atraso no desenvolvimento decorrente a ambiguidade nas descrições dos requisitos dos serviços do componente [Paldês et al. 2016]. Assim, impactando no entendimento e implementação, comprometendo o artefato final.

Para se mitigar estes problemas, a utilização de uma Linguagem de Descrição de Interfaces (*Interface Description Language* - IDL) [OMG 2017] pode ser uma solução viável. A adoção de IDL fornece várias vantagens ao programador, como por exemplo, descrever a interface de modo que a mesma fique mais próxima da sua implementação final e representar a arquitetura e estrutura dos pacotes intensificando também a interoperabilidade entre componentes.

Com o objetivo de explorar as vantagens apresentadas, este trabalho propõe uma abordagem para a especificação das interfaces de componentes utilizando IDL. Destaca-se que em nosso trabalho a caracterização pode descrever interfaces de qualquer componente do sistema, não estando restrita apenas a utilização em arquiteturas de sistemas distribuídos. Esta abordagem apresenta benefícios tanto para o desenvolvedor de componentes, quanto para a criação de *middlewares* responsáveis pela comunicabilidade entre os mesmos. Estas vantagens derivam da simplicidade do uso de IDL e de sua especificação livre de ambiguidade.

Com o objetivo de solucionar tais problemas decorrentes a especificação, o presente artigo foi estruturado como segue. A Seção 2 apresenta o estado da arte em que descreve sobre os componentes de software e suas vantagens, a seguir é exposto sobre o tipo de desenvolvimento baseado em componentes, quando surgiu a proposta e o que esta forma de desenvolver proporciona, bem como conceitua IDL com o objetivo de explicar sobre a linguagem e suas vantagens. Além disso, descreve o impacto de sua utilização em arquiteturas. A Seção 3 é o segmento responsável pela comparação entre como é feita a especificação tradicional e a proposta para utilização de IDL dentro do processo. A Seção 4 apresenta um exemplo de especificação de componente usando DSL. Finalizando, a Seção 5 descreve as considerações finais do estudo.

2. Estado da Arte

2.1. Componentes de Software

O conceito de componente não é unanimemente determinado e existem várias definições do termo. Um conceito geral de componentes reconhecido é que um componente de software deve ser um pacote coerente e independente que pode ser entregue de forma unitária, onde o mesmo pode ser utilizado em conjunto com outros para construir algo maior sem que haja a necessidade de mudanças em sua implementação [D'Souza and Wills 1998].

Componentes podem ser: qualificados, adaptados, aglutinados e atualizados e implementam um interesse específico do sistema, ocultando seus detalhes de confecção e provendo seus serviços por meio de *interfaces*. De acordo com sua especificidade os componentes podem ser mais reutilizáveis do que outros. Sabendo disto estes artefatos podem ser divididos em três grupos (ver Figura 1). Sendo o primeiro os `Genéricos` em que se localizam os componentes que possuem maior reusabilidade, um exemplo destes são módulos que manipulam a interface. Outro grupo é definido pelos que provêm

`Serviços` como impressão de nota fiscal, possuindo uma reusabilidade intermediária dentre os conjuntos. O último conjunto é definido pelos componentes de `Domínio`, estes retratam módulos específicos que possuem a menor reusabilidade como os que são utilizados em clínicas médicas.

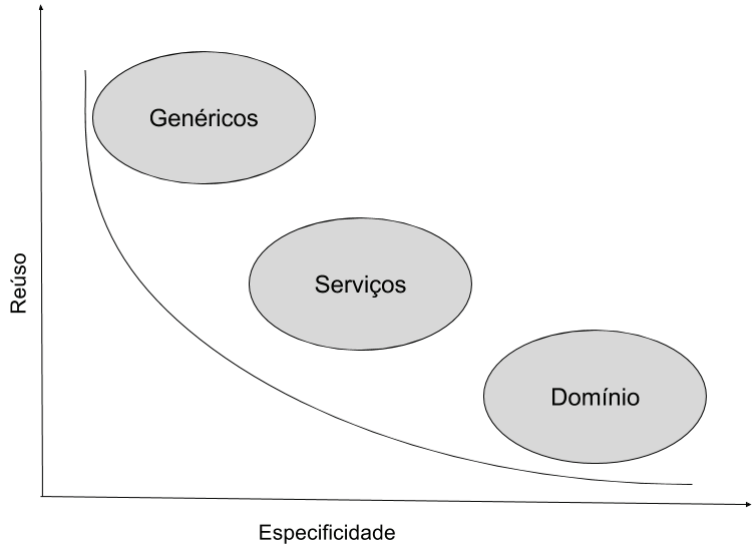

Figura 1. Níveis de reúso dos componentes

Estes componentes tem como principal objetivo aumentar o reúso, e com isso garantir uma maior confiabilidade, qualidade, além de uma redução no tempo de desenvolvimento, custos e complexidade dos sistemas. A gerência de mudanças em um software confeccionado por componentes também é facilitada, tendo em vista que um componente pode ser facilmente adicionado ou substituído, desde que sua interface seja mantida fornecendo novas funcionalidades.

2.2. Desenvolvimento de Software Baseado em Componentes (DSBC)

A utilização de componentes para o desenvolvimento de software foi exposta pela primeira vez em 1968 durante uma conferência da OTAN sobre Engenharia de Software em que foi apresentada a ideia de componentização e reúso por meio do artigo *Mass Produced Software Components* [McIlroy 1968]. Entretanto, o paradigma de desenvolvimento só foi proposto em 1976 por intermédio de uma comparação entre duas formas de implementação de software, em que mencionava a programação utilizando somente componentes e os conectando entre si [DeRemer and H. Kron 1976].

A proposta se fortaleceu em 1980 com a orientação a objetos que proporcionou maior reutilização de código. Este desenvolvimento possui cinco etapas essenciais de acordo com [Brown and Short 1997], são elas: (1) Procurar componentes que possam ser utilizados no desenvolvimento da aplicação; (2) Selecionar os componentes que suprem as necessidades dos requisitos da aplicação a ser desenvolvida; (3) Realizar as adaptações necessárias nos componentes selecionados; (4) Efetuar a composição dos componentes entre si; (5) Atualizar os componentes necessários que demandem modificações.

Com a aplicação destas etapas é possível melhorar a qualidade do software desenvolvido, pois os componentes já foram testados, além do software ser liberado em um

tempo reduzido para o mercado. Outra possibilidade decorrente da utilização de componentes, além do reúso, é a manutenção facilitada do software, pois os componentes podem facilmente serem substituídos, removidos ou adicionados ao produto. De acordo com [Cheesman and Daniels 2000] o princípio adotado para este desenvolvimento é o de divisão e conquista, simplificando a implementação e entendimento do sistema.

A implementação de componentes é uma das perspectivas do desenvolvimento de software baseado em componentes. Estes artefatos devem ser criados de maneira que possam ser reutilizados em outras aplicações, podendo estas pertencerem ao mesmo domínio ou não. Para isto, existem técnicas de criação que podem ser utilizadas visando a melhoria da qualidade dos artefatos criados.

2.3. Interface Description Language (IDL)

A IDL é uma linguagem declarativa que tem como foco a descrição das interfaces de objetos [OMG 2017]. Com a mesma é possível especificar parâmetros de entrada, atributos e operações a serem realizadas pelo componente. É uma linguagem fortemente tipada, baseada em C++, onde se pode estipular constantes tipos e até as exceções que podem ocorrer no módulo durante a execução além de possibilitar especificação dos retornos. Provê também suporte para o pré-processamento, substituição de macros e herança múltipla. A adição de documentação também é possível dentro do mesmo arquivo, como divisão dos módulos e comentários sobre a especificação. É amplamente utilizada para a definição de funções em sistemas distribuídos ou baseados em componentes. Tecnologias como a arquitetura CORBA (*Common Object Request Broker Architecture*) [OMG 2012], juntamente com o DCOM [Redmond 1997], o utilizam para chamada de processos remotos, possibilitando uma maior interoperabilidade entre os componentes além da viabilidade de criação de softwares híbridos.

Atualmente, existem especificações diferentes para IDL, tendo como padrão a elaborada pela OMG que é direcionada para utilização em conjunto com o CORBA. Uma das alternativas de uso disponíveis é o MIDL (*Microsoft Interface Definition Language*), que também é utilizado juntamente com o RPC (*Remote Procedure Call*) para chamada de procedimentos remotos e interoperabilidade entre componentes [Myerson 2002]. É possível encontrar compiladores disponibilizados pelos fornecedores, os quais tem como objetivo a conversão do arquivo IDL para uma estrutura em uma das linguagens de programação suportadas pela especificação escolhida.

2.4. Impacto da IDL na Arquitetura

Em sistemas distribuídos os componentes podem ser escritos em diferentes linguagens de programação e utilizar estruturas de dados incompatíveis. Neste caso, o uso de *middlewares* se torna necessário para confecção dos mesmos. Com a utilização de arquivos IDL para especificação dos componentes, viabiliza uma clara definição dos módulos, possibilitando sua implementação de forma simultânea com os *middlewares* assegurando sua integração. A importância da utilização de IDL é ainda maior quando o *middleware* é orientado a objetos, pois a interação entre componentes é realizada pela invocação de métodos, que também podem ser descritos utilizando IDL, e podem ser traduzidos para qualquer outra linguagem mapeada pela especificação escolhida.

A arquitetura CORBA é baseada em componentes distribuídos que utiliza o modelo cliente-servidor. Neste tipo de arquitetura cada componente pode atuar como cliente

e/ou servidor. Para permitir a comunicação entre estes componentes é utilizado *middlewares* orientado a objetos. Dentro do CORBA estes *middlewares* são denominados ORB (*Object Request Broker*), e têm como fator crucial efetuar chamadas de métodos remotos dentro do sistema [Link et al. 2013]. Com esta finalidade o ORB utiliza das especificações das interfaces dos componentes em IDL no formato padrão definido pela OMG. Assim, não necessitam conhecer a tecnologia sob a qual o módulo foi implementado para realizar sua invocação.

Em uma arquitetura cliente/servidor para invocar alguma operação em outro componente do sistema é necessário conhecer detalhes do servidor, como os parâmetros de entrada, retorno e as operações a serem realizadas. Para isto, dentro do CORBA é utilizado o ORB que tem como função ser responsável por todos os mecanismos necessários para a localização da implementação do objeto invocado, garantindo que assim que o mesmo possa receber a requisição e transferir os resultados de volta ao cliente [Link et al. 2013].

3. Uma Proposta de Especificação de Componentes Utilizando IDL

No DSBC é realizado a especificação dos componentes no repositório, isto é relevante para que seja possível ter um melhor controle dos mesmos. É importante conter uma quantidade de informações suficientes, pois assim é possível recuperar esses artefatos por meio de mecanismos de busca [Redolfi et al. 2004]. Com o objetivo de suprir essa necessidade a especificação das interfaces podem ser realizadas com arquivos IDL, facilitando a busca como acontece na arquitetura CORBA com um mecanismo semelhante as ORB's, além de também garantir a interoperabilidade e portabilidade dos componentes quando desenvolvidos. Caso aconteça a utilização de alguma arquitetura que faça seu uso dentro das comunicações entre os componentes, sua utilização acaba sendo potencializada no desenvolvimento.

O modo tradicional de se caracterizar e especificar as interfaces de componentes é realizado por meio de diagramas e tabelas contendo informações sobre o artefato. Essa abordagem é interessante por conter um detalhamento em linguagem natural sobre os serviços que os componentes oferecem. Esta abordagem, no entanto, pode acarretar problemas como ambiguidade e omissão de informações a nível de implementação, ocasionando diversos problemas para a equipe de desenvolvimento, tais como: aumento do tempo gasto no desenvolvimento e diminuição na qualidade do componente criado. A utilização de IDL é uma solução para os problemas presentes no modo tradicional, além de proporcionar um melhor entendimento do funcionamento dos componentes também contém o necessário para a comunicação com os mesmos, impactando tanto no desenvolvimento de componentes quanto na implementação do sistema que os utiliza.

Com a especificação das interfaces definidas em IDL é possível saber quais são as entradas e saídas dos métodos de forma simplificada e rápida, impactando no tempo de desenvolvimento das conexões entre os componentes. Tendo em vista que desta forma a especificação é realizada de forma independente da linguagem de programação, é possível representar um componente implementado em qualquer tecnologia. A adoção de arquiteturas que utilizam a linguagem juntamente com mecanismos de chamada remota possibilita a criação de softwares híbridos e melhorias relacionadas a portabilidade e comunicabilidade entre os componentes. Com a utilização de IDL é viável o gerenciamento dos módulos e adicionar comentários proporciona uma especificação ainda mais facilitada

para o desenvolvedor do sistema.

Como IDL é uma linguagem baseada em C++, o entendimento da notação pela equipe de desenvolvimento não é afetado drasticamente com a adoção da tecnologia, sabendo que esta linguagem é amplamente conhecida e utilizada também como base para o desenvolvimento de outras como Java e C#, que por sua vez também são tecnologias aplicadas na implementação de sistemas baseados em componentes [Sebesta 2009]. No caso do desenvolvimento de componentes a especificação se torna ainda mais eficaz, já que possui compiladores que geram código não funcional a partir da tecnologia, definindo assim uma estrutura para o componente. Dependendo do compilador utilizado, o código gerado é baseado na arquitetura, usado comumente para a arquitetura CORBA.

4. Exemplo de Componente Especificado em IDL

Com a finalidade de ilustrar um componente especificado utilizando IDL, a ferramenta da linha de produtos PLeTs [Rodrigues 2013] foi escolhida como exemplo, já que seu desenvolvimento ocorreu baseado no uso de componentes. Este software tem como funcionalidade a criação de casos de teste para sistemas web baseado em modelos formais, *e.g.* Máquinas de Estados Finitos. Seu diagrama de componentes pode ser visualizado na Figura 2.

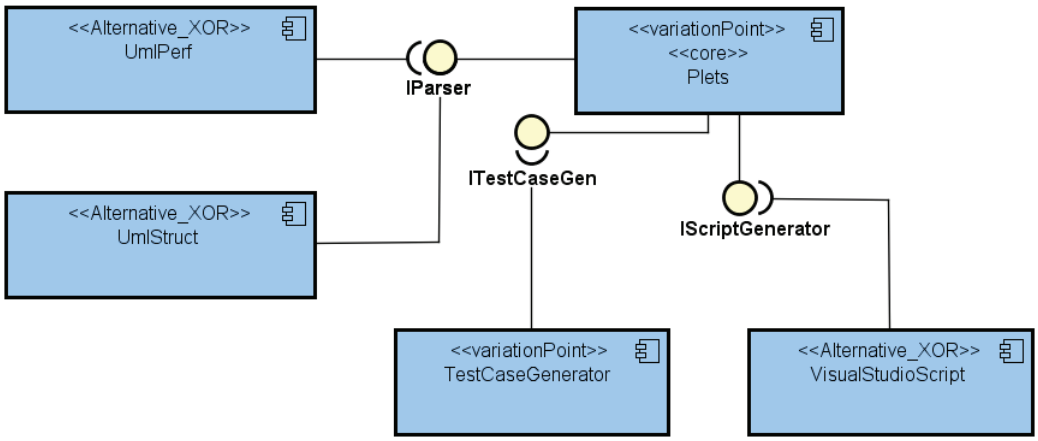

Figura 2. PLeTs: Parte do Diagrama de Componentes UML- Baseado em [Rodrigues 2013]

O módulo escolhido para a representação aplicando IDL foi o *IParser* utilizando a notação UML (*Unified Modeling Language*) [Booch et al. 2000]. Esse componente tem como função a análise de um arquivo no formato XMI, com o objetivo de retirar os dados referentes aos modelos pertencentes ao sistema onde serão executados os testes gerados. No diagrama de componentes da Figura 2 é possível visualizar alguns dados sobre o artefato, entretanto, falta informações a nível de implementação. Uma alternativa pode ser o uso de diagramas de classe, mas como a utilização de IDL acarreta também várias outras melhorias, sua aplicação pode ser mais benéfica ao desenvolvimento. O exemplo pode ser visualizado por meio do trecho de código da Figura 3, o qual retrata o componente *ParserUML* com base na interface *IParser*.

Com a especificação da interface *IParser* é perceptível uma maior quantidade de detalhes relacionadas ao funcionamento, diferente do diagrama encontrado na Figura 2.

```
#include<UML.idl>
module Project {
    module ExampleParserXMI{
        interface ParserUML{
            void managerParser(in string pathArchiveXMI);
            Uml::UMLElement createActor(in string partialArchiveXMI);
            Uml::UMLElement createUseCase(in string partialArchiveXMI);
            Uml::UMLElement createInitAction(in string partialArchiveXMI);
            Uml::UMLElement createFinalAction(in string partialArchiveXMI);
            Uml::UMLElement createAction(in string partialArchiveXMI);
            Uml::UMLElement createFork(in string partialArchiveXMI);
            Uml::UMLElement createJoin(in string partialArchiveXMI);
            Uml::UMLAssociation createAssociation(in string partialArchiveXMI);
            Uml::TypeAssociation createInclude(in string partialArchiveXMI);
            Uml::TypeAssociation createExtend(in string partialArchiveXMI);
            Uml::TypeAssociation createGeneralization(in string partialArchiveXMI);
            Uml::UMLElement createDecisionNode(in string partialArchiveXMI, in Uml::
                TypeDiagram typeDiagram);
            boolean addDiagram(in Uml::UMLDiagram newDiagram, in Uml::TypeDiagram
                typeDiagram, in string uniqueName);
            Uml::UMLDiagram getDiagram(in Uml::TypeDiagram typeDiagram, in string
                uniqueName);
        };
    };
};
```

Figura 3. Especificação IDL do componente *ParserUML* com base em *IParser*

Tendo em vista que é possível estruturar os pacotes, *interfaces*, métodos, entradas, saídas e até mesmo a inclusão de outros módulos ao projeto. É importante salientar que sua quantidade de detalhes pode superar a de um diagrama de classes comum, pois além das possibilidades descritas também é possível o uso de comentários em conjunto à especificação.

5. Considerações Finais

A utilização de IDL no Desenvolvimento de Software Baseado em Componentes (DSBC) pode se mostrar superior ao modelo tradicional no aspecto especificação e entendimento do componente para o desenvolvedor, impactando positivamente tanto na criação quanto na utilização. Além de possibilitar a implementação do mesmo em paralelo com o *middleware* necessário para a integração. Outro fator positivo seria a facilidade na busca do repositório, sua importância é justificada em grandes empresas decorrente da quantidade elevada de componentes, criando a necessidade de um mecanismo de busca que pode utilizar IDL com esta finalidade, proporcionando uma maior praticidade, e tornando viável a procura do componente de acordo com seu comportamento. Assim, podendo auxiliar na redução do tempo de desenvolvimento de um sistema orientado a componentes.

É importante destacar que a aplicação de IDL na especificação de componentes não é independente. Mesmo que IDL proporcione uma série de melhorias a linguagem não pode substituir toda a especificação tradicional. Tendo em vista que esta também é aplicada no entendimento do componente para as demais partes interessadas, *i.e. stakeholders*. A linguagem natural mesmo que possa conter ambiguidades é o modo de proporcionar uma melhor assimilação no momento da negociação, em especial com clientes não técnicos. Diagramas e tabelas explicativas também podem ser mais expressivas nesse cenário do que somente o uso de IDL.

Assim, o uso da tecnologia deve ser realizado de modo conjunto com a especificação tradicional, pois devido ao seu impacto no desenvolvimento é possível uma melhor busca, compreensão e melhoria do artefato gerado. Além disso, a diminuição da ambiguidade das tabelas explicativas, tendo em vista que os arquivos podem ser utilizados para verificação e validação tanto do documento de especificação quanto do código implementado. Por fim, como trabalho futuro será realizada a aplicação desta proposta dentro de um cenário real de desenvolvimento. Assim, com o propósito de avaliar o impacto que a equipe poderá ter com a adoção desta proposta na especificação para softwares baseados em componentes.

Referências

Booch, G., Rumbaugh, J., and Jacobson, I. (2000). UML, Guia do Usuário.

Brown, A. W. and Short, K. (1997). On components and objects: the foundations of component-based development. In *Assessment of Software Tools and Technologies, 1997., Proceedings Fifth International Symposium on*, pages 112–121. IEEE.

Cheesman, J. and Daniels, J. (2000). *UML components: a simple process for specifying component-based software*. Addison-Wesley Longman Publishing Co., Inc.

DeRemer, F. and H. Kron, H. (1976). Programming-in-the-large versus programming-in-the-small. SE-2:80 – 86.

D'Souza, D. F. and Wills, A. C. (1998). *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Link, E., Alexandre, E. B. P., Wolf, J. L., and Strzykalski, M. S. (2013). Uma Introdução ao CORBA.

McIlroy, M. D. (1968). Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*.

Myerson, J. M. (2002). *The complete book of middleware*. CRC Press.

OMG (2012). Information technology: Common Object Request Broker Architecture (CORBA), Interfaces. Formal/2012-05-03.

OMG (2017). Interface Definition Language. Version 4.1.

Paldês, R., Calazans, A., Mariano, A., Castro, E., and de Souza da Silva, B. (2016). A utilização da linguagem natural na especificação de requisitos: um estudo por meio das equações estruturais.

Redmond, F. (1997). *DCOM: Microsoft Distributed Component Object Model*. Professional Series. IDG Books Worldwide.

Redolfi, G., de Araujo Spagnoli, L., Bastos, R. M., Cristal, M., and Espindola, A. P. (2004). Especificando informações para componentes reutilizáveis. *Porto Alegre, Abril*.

Rodrigues, E. d. M. (2013). PLeTs: a product line of model-based testing tools.

Sebesta, R. W. (2009). *Conceitos de linguagens de programação*. Bookman Editora.