UNIVERSIDADE ESTADUAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE LICENCIATURA EM COMPUTAÇAO

SÉRGIO ADRIANO OLIVEIRA SANTOS

DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES: UMA VISÃO PRÁTICA.

CAMPINA GRANDE – PB
2015

SÉRGIO ADRIANO OLIVEIRA SANTOS

DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES: UMA VISÃO PRÁTICA.

Trabalho de Conclusão de Curso apresentada ao Programa de Graduação de Licenciatura em Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Graduado.

Área de concentração: Engenharia de Software.

Orientador: Prof. Frederico Moreira Bublitz.

CAMPINA GRANDE – PARAÍBA

2015

S237d Santos, Sérgio Adriano Oliveira.
Desenvolvimento de software baseado em componentes [manuscrito] : uma visão prática / Sergio Adriano Oliveira Santos. - 2015.
38 p.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2015.
"Orientação: Prof. Dr. Frederico Moreira Bublitz, Departamento de Computação".

1. Reuso. 2. Componentes. 3. Desenvolvimento Baseado em Componente. I. Título.

21. ed. CDD 005.3

SÉRGIO ADRIANO OLIVEIRA SANTOS

DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES: UMA VISÃO PRÁTICA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Licenciatura plena em Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação.

Aprovada em 30 de abril de 2015.

Prof. Dr. Frederico Moreira Bublitz
Orientador(a)

Prof. Me. Edson Holanda Cavalvante Jr.
Examinador(a)

Prof. Dr. José Carlos Mota
Examinador(a)

RESUMO

A qualidade dos produtos de software depende dos processos de desenvolvimento utilizados. Definir processos de software não é uma atividade trivial, necessitando conhecimento e experiências para adotar abordagens de apoio para redução de esforço na execução das tarefas. Nesse sentido, o Desenvolvimento Baseado em Componentes (DBC) pode ser usado com o objetivo de reduzir custos e tempo de desenvolvimento em certos domínios de aplicação. Enquanto metodologias tradicionais constroem sistemas do zero, nessa abordagem, o desenvolvimento de software ocorre por meio da composição de componentes pré-existentes, desenvolvidos ou adquiridos de terceiros e é controlado por processos definidos na Engenharia de Software Baseado em Componentes (ESBC). Desse modo, o DBC favorece o desenvolvimento em alguns domínios de aplicação que possuem requisitos complexos e exigem equipes especializadas no conhecimento dos requisitos, centralizando a implementação desses requisitos nos métodos dos componentes. Nesse trabalho, aborda-se a temática do desenvolvimento baseado em componentes demonstrando suas características e principais vantagens. Para demostrar como o Desenvolvimento Baseado em Componentes pode ser aplicado, desenvolveu-se um projeto de um componente de software que pode ser utilizado em muitas aplicações. Para justificar o desenvolvimento do projeto foi percebido grande potencial de reuso no domínio de aplicações de gestão comercial (ERP), na implementação de regras de negócio relacionadas à emissão de documentos fiscais.

Palavras-Chave: Reuso, Componentes, Desenvolvimento Baseado em Componente

ABSTRACT

The quality of software products depends on the development processes used. Define software processes is not a trivial activity, requiring knowledge and experiences to adopt approaches to support for reduction effort in performing the tasks. Therefore, Component Based Development (CBD) aproach can be used in order to reduce costs and time to market in certain application areas. While traditional methodologies build systems from the ground, in the CBD approach the software development occurs through the composition of pre-existing components, developed or acquired from third parties and it is controlled by processes defined in Component Based Software Engineering (CBSE). Thus, the DBC favors the development in some areas of applications that have complex requirements and require expert teams in the knowledgement of the requirements, centralizing the implementation of these requirements in the methods of the component. This paper deals with the theme of component-based development demonstrating their features and key advantages of the approach. To demonstrate how the Component Based Development can be applied, was developed a project of a software component that can be used in many applications. To justify the development of the project was noticed great potential of reuse in the domain of business management applications (ERP) implementation of business rules related to the transmission of some documents. This paper brings as the general objectives, verify the significant advantages of reuse of components and how the software market has supported the use of this approach. Specific objectives to this paper are bringing out the project of a component to meet certain demand of complex requirements in the domain of business and be reused by several applications (ERP) and showing the benefits achieved in productivity in project of applications that reuse this component.

Keywords: Reuse, Component, component based Development

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
DBC	Desenvolvimento Baseado em Componentes
OO	Orientadas a Objetos
COTS	Commercial off the shelf
SGBD	Sistemas gerenciadores de bancos de dados
ESBC	Engenharia de Software Baseada em Componentes
SOAP	Simple Object Access Protocol

LISTA DE FIGURAS

SUMÁRIO

1- INTRODUÇÃO

Devido às exigências do mercado e o aumento da competitividade, as empresas precisam desenvolver sistemas cada vez mais complexos com prazos cada vez mais curtos. A qualidade e flexibilidade dos sistemas de informação também se tornaram aspectos críticos para a sobrevivência das organizações. Nesse cenário, a reutilização de software surge como uma forma de obter os benefícios de menor custo, menor tempo de desenvolvimento (maior produtividade), e consequentemente, maior agilidade para entregar produtos de qualidade em curtos prazos (Mann e Kaur, 2010, p. 105).

A possibilidade de reuso de partes é bem aceita nos projetos de engenharia em geral a fim de alcançar a produtividade e a padronização dos processos (Sommerville, 2004, p. 260). Na engenharia de software não é diferente, a utilização da tecnologia de componentes no processo de desenvolvimento materializa claramente o reuso e o reaproveitamento, facilitando o desenvolvimento de sistemas com maior qualidade e menor complexidade. Maior qualidade porque componentes desenvolvidos, e previamente utilizados, tendem a estar com suas funcionalidades experimentadas e testadas atingindo um alto nível de maturidade (Szyperski, 2002, p. 4). Menor complexidade porque o processo de integração do componente ao sistema em desenvolvimento esconde a complexidade da implementação no domínio do componente (SOFTEX, 2007, p. 10).

O reuso de componentes de software também é eficiente em termos de custos para o desenvolvimento à medida que são distribuídos para mais aplicações (Szyperski, 2002, p. 19). Assim muitos engenheiros de software têm adotado o Desenvolvimento Baseado em Componentes (DBC) para controlar a complexidade e os riscos no desenvolvimento, utilizando uma abordagem centrada na arquitetura e no reuso, identificando funcionalidades e serviços comuns a várias aplicações para desenvolver ou adquirir componentes de software para integração com seus sistemas (Mann e Kaur, 2010, p. 105).

Um projeto desenvolvido com objetivo de reuso surge do pensamento de que funcionalidades implementadas são muitas vezes similares ou até mesmo idênticas em inúmeras aplicações (Pressman, 2006, p.665). Enquanto o desenvolvimento de software baseado em componentes pode ser visto como uma abordagem que permite reduzir custo, tempo de desenvolvimento e aumentar a qualidade final do produto, a construção

de componentes requer alguns cuidados, pois seus requisitos devem ser especificados e compreendidos de maneira isolada do contexto de um sistema que irá reutilizá-los. Além disso, componentes devem possuir uma interface clara e documentada para que qualquer desenvolvedor possa utilizá-lo sem conhecer sua implementação. Esses fatores desestimulam a adoção da abordagem DBC pelas empresas de software, pois requer um processo de desenvolvimento paralelo ao processo de desenvolvimento de aplicações, demandando equipes especializadas, ferramentas e processos diferenciados.

Esse trabalho tem como objetivos gerais analisar o panorama do processo de desenvolvimento de software baseado em componentes, detalhando as vantagens e desvantagens dessa abordagem e as etapas do processo para se obter os benefícios do DBC. Como objetivos específicos será apresentado o projeto de um componente de software desenvolvido com objetivo de atender certa demanda de requisitos para emissão de NF-e (nota fiscal eletrônica), verificando na prática os benefícios alcançados em termos de produtividade e redução de custos em projetos de aplicações de gestão comercial (ERP) que fazem reuso desse componente.

2- REVISÃO DA LITERATURA

2.1. ASPECTOS DO REÚSO DE SOFTWARE

SOFTEX (2007, p. 11) apud Szyperski (2002) define o reuso de software em quatro categorias:

1. Reuso de Código Fonte: Durante a fase de desenvolvimento de um novo projeto de software, trechos de códigos podem ser reutilizados. As sub-rotinas já eram uma forma de reaproveitamento de código nas linguagens procedurais. Com o surgimento das linguagens Orientadas a Objetos (OO), o conceito de reuso de código evoluiu com a criação de classes especializadas que podem ser reutilizadas em diversas aplicações escritas em uma plataforma OO.

2. Reuso de Partes de Software: Reuso de arquitetura e implementação de fragmentos de software em diferentes projetos. Nesse caso o reuso ocorre durante o design da arquitetura da aplicação e da implementação do código. Assim como no caso anterior, nessa modalidade de reuso não existe componente como parte independente da aplicação final a ponto de ser substituível por outro componente que realiza as mesmas tarefas.

3. Integração Dinâmica: O reuso ocorre em tempo de execução. A aplicação já está desenvolvida e em produção, e novas funcionalidades são acrescentadas através de plugins desenvolvidos independentemente do desenvolvimento da aplicação e geralmente são desenvolvidos posteriormente para acrescentar funcionalidades requisitadas. Um exemplo típico desse tipo de reuso são os plugins[1] adicionados aos browsers para que estes possam ler arquivos PDF.

4. Componentização: Tipo de reuso onde pacotes de artefatos de software que foram desenvolvidos independentemente como uma unidade, são integrados com outros componentes para construir um sistema. Esses pacotes implementam regras de negócio comuns a vários sistemas dentro de um determinado domínio de aplicação. Teoricamente é possível substituir um componente em um sistema por outro com maior desempenho sem alterações consideráveis no sistema. Esse tipo de reuso é o objeto principal desse trabalho.

Outra forma importante de reuso de código é a utilização de frameworks Orientados a Objetos, que fornecem uma infraestrutura completa de classes (arcabouço) e pode ser reutilizado em vários contextos, reduzindo assim o esforço para se construir uma aplicação (Gurp & Bosch, 2001, p. 278). Na literatura também se enfatiza o reuso de boas práticas de design por meio de padrões de projeto, nesse caso, reuso de experiência na solução de problemas recorrentes em projetos de software (Gamma et al, 1997, p. 12).

2.2 DESENVOLVIMENTO BASEADO EM COMPONENTES

Metodologias tradicionais de desenvolvimento consistem em desenvolver sistemas a partir do zero, estruturados em forma de blocos monolíticos. Essa abordagem faz com que uma alteração possa propagar efeitos colaterais por todo o código, o que dificulta a manutenção e substituição de partes de código (Werner & Braga, 2005, pg 66). A abordagem baseada em componentes surgiu como uma nova perspectiva para solucionar essa e outras questões. Na abordagem baseada em componentes, o software passa a ser composto de partes relativamente independentes, que foram concebidas para serem substituíveis, reutilizáveis e interoperáveis.

2.3 REÚSO DE COMPONENTES DE SOFTWARE

O reuso é uma técnica utilizada já há algum tempo na comunidade de desenvolvimento. Essa técnica consiste em incorporar em um novo produto artefatos já utilizados em outros projetos como código, especificações de requisitos, produtos gerados anteriormente, ideias e conhecimentos em geral, afim de se obter melhores índices de produtividade, produtos de melhor qualidade, mais confiáveis, mais consistentes e com menor custo.

As linguagens procedurais permitem que se crie funções ou subrotinas que podem ser chamadas de qualquer ponto do sistema e também podem ser reutilizadas em outros sistemas. Já as linguagens orientadas a objetos vão ainda mais além quando permitem o reaproveitamento de classes que foram escritas anteriormente e já tem seu funcionamento comprovado e testado. Essa técnica diminui a necessidade de se escrever novas classes e métodos e comprovadamente diminui também o trabalho do programador e a ocorrência de erros (Santos, 2013, pg 113). Em Java, por exemplo, existem dois mecanismos básicos de reuso de classes - composição e herança. Com a composição pode-se criar novas classes que tenham como campos (propriedades) instancias de outras classes já existentes. Já a reutilização de classes por meio de herança permite que se escrevam novas classes utilizando como base classes criadas anteriormente de modo que essas novas classes estendem as classes já existentes e essas são mais especializadas do que as primeiras. As classes a serem reutilizadas geralmente são disponibilizadas por meio de pacotes de classes que são importados em novos projetos para reuso e geralmente suas assinaturas são expostas por meio de uma documentação (API) que auxilia os desenvolvedores a fazerem reúso dessas classes escritas por outros desenvolvedores. Para desenvolver pacotes de classes para reúso deve-se ter boas abstrações que poderão ser úteis em vários contextos de software só assim é justificável o reúso dessas classes.

No domínio da orientação a objetos também existem frameworks formados por interfaces, classes abstratas e concretas que compõem uma estrutura genérica para servir de base para várias aplicações (Gurp & Bosch, 2001, p. 278). São implementadas funcionalidades comuns a diversas aplicações (acesso a dados, autenticação, segurança, componentes de interfaces gráficas, validação, chamadas remotas a procedimentos, etc). Geralmente esses frameworks são desenvolvidos e mantidos por comunidades de desenvolvimento. A utilização dessa estrutura se dá pelo desenvolvimento de classes

que estendem as classes do framework (herança) de modo a adaptar o seu funcionamento às regras de negócio do projeto em particular. Temos alguns frameworks conhecidos no mercado de desenvolvimento como o Struts, Spring, Hibernate (persistência) para Java e MFC para plataformas da Microsoft.

2.4 O MERCADO DE COMPONENTES

O relatório da SOFTEX mostrou uma tendência cada vez maior de as empresas de software apoiarem suas atividades na engenharia de software baseada em componentes e o DBC surge como um novo segmento no mercado de software (SOFTEX, 2007). Estes estudos indicam alterações no mercado, como o estabelecimento de um novo setor, vinculado ao conceito de componentes. Algumas empresas atuam neste mercado, que inclui desde o desenvolvimento independente de componentes, a intermediação de compra-venda dos mesmos e até sua certificação de qualidade.

O SEI-CMU (Bass et al, 2001) caracteriza o mercado da seguinte forma:

1. Mercado de componentes individuais: Desenvolvimento e comercialização de componentes para o mercado de varejo.

2. Mercado de linhas de produtos: Desenvolvimento e comercialização de linhas de componentes dedicadas a um mercado vertical em certo domínio de negócio (por exemplo, uma suíte de componentes de negócios voltada para aplicações financeiras).

3.Mercado de infra-estrutura: Compreende um conjunto de padrões, plataformas e ferramentas de apoio ao desenvolvimento baseado em componentes. Este mercado está dividido em:

a) Padrões de infra-estrutura: Envolvem organismos e empresas interessadas em promover e definir os padrões que formarão a infra-estrutura do desenvolvimento baseado em componentes. Exemplos: OMG (Corba), Microsoft (COM, .NET) e Sun (EJB).

b) Plataformas e servidores de aplicação: A disponibilidade de plataformas e servidores de aplicações (que implementam os padrões de infra-estrutura) são fundamentais em todos os ambientes em que se deseja utilizar software baseado em componentes.

c) Mercado de ferramentas para componentes: Envolve o desenvolvimento e comercialização de ferramentas para apoio ao desenvolvimento baseado em componentes e gerenciamento de componentes.

4. Mercado de integração/consultoria: grandes conhecedores dos componentes, das plataformas e ferramentas disponíveis, estes desenvolvedores podem construir aplicações de forma eficiente e rápida para seus clientes, tornando este domínio sua vantagem competitiva. Também é o caso de fornecedores que oferecem uma linha de componentes e têm seu modelo de negócio baseado no uso desta linha juntamente com seu know-how para a construção de soluções.

5. Mercado de intermediação (brokerage): dedicados a oferecer uma área de compra-venda para componentes de software desenvolvidos por terceiros, criando uma marca consolidada e facilidades para busca e aquisição de componentes.

6. Mercado de certificação: que oferece aos consumidores de componentes garantias que os mesmos atendem determinados padrões, efetivamente oferecem determinados serviços e/ou apresentam determinados atributos de qualidade.

O mercado de componentes origina-se na interação de empresas especializadas no desenvolvimento, manutenção e atualização de componentes com seus clientes imediatos que, na maior parte dos casos, também são desenvolvedores de software. Os clientes têm a opção de adquirir componentes de diversos fornecedores, desde que sigam um mesmo modelo de componentes, um mesmo contrato (assinatura de métodos) e uma mesma interface, assim, um produto que utiliza componentes beneficia-se da produtividade, flexibilidade e inovação de todos os fornecedores presentes no mercado de componentes.

2.5 QUANDO APLICAR O REÚSO DE COMPONENTES

Antes de se fazer reuso de componentes, é fortemente recomendado ter-se um plano de reutilização baseado em decisões organizacionais com relação aos custos e benefícios envolvidos. Para adotar um processo de reuso de componente em um projeto, numa fase inicial devem ser definidos os objetivos de se fazer reuso, como a produtividade ou a qualidade, e analisadas as oportunidades de reutilização. Um determinado projeto de software pode estar incluído em diversos domínios de aplicação, devendo o potencial de reutilização de cada domínio ser avaliado. Enquanto certos domínios que exibem grande estabilidade e já foram extensivamente estudados oferecem índices de reutilização potencial elevados, outros existem onde o potencial de reutilização é baixo (Pressman, 2006, p.668). Contudo, não existe uma forma simples que permita avaliar qual o potencial de reutilização dentro de um determinado domínio de aplicação sem o fazer repetidamente e analisar os resultados obtidos.

Com base em experiências obtidas diretamente ou indiretamente através dos resultados de terceiros, é necessário definir os domínios e subdomínios que oferecem maiores índices de reutilização potencial e menores riscos técnicos e financeiros para o projeto (Pressman, 2006, p.669). Uma vez definidos os domínios onde será aplicado o reúso, deve-se definir os objetivos a serem atingidos dadas as limitações técnicas e financeiras.

Jasmine et al (2010), cita alguns obstáculos que devem ser identificados e removidos para se ter um alto nível de reuso na produção de software:

- Cultural: Fatores culturais podem influenciar nas decisões de se reutilizar componentes. Geralmente as equipes envolvidas resistem ao fato de componentes não terem sido implementados por eles (síndrome do “Não foi inventado aqui”).
- Institucional: A necessidade de identificar produtos com alto potencial de reuso deve ser compartilhada por todos os integrantes da empresa.
- Financeiro: O custo-benefício deve ser levado em conta ao pensar em desenvolver produtos com a filosofia de reuso. Componentes reusáveis devem ser vistos como ativos fixos da empresa.
- Técnico: São necessários mecanismos adequados para garantir que diretrizes, técnicas e padrões sejam desenvolvidos e seguidos.
- Legais: Negociações devem ser realizadas para determinar a forma de reter os

direitos autorais sobre os componentes desenvolvidos e distribuídos para recuperar os custos em um contexto de reutilização. São necessários mecanismos de cobrança de royalties pelo uso e reutilização na área comercial.

A natureza dos componentes desenvolvidos também constitui um dos maiores desafios no projeto de sistemas baseados em componentes, quanto mais um componente é independente de determinado contexto mais facilmente ele pode ser reutilizado em outros contextos. No entanto, um componente totalmente independente de qualquer contexto não pode ser reutilizado, uma vez que necessita de um contexto para poder ser integrado.

A principal motivação para a reutilização de software é a redução do esforço necessário para o seu desenvolvimento. Quanto mais vezes um componente for reutilizado, menor é o esforço total de desenvolvimento. Por outro lado, quanto mais genéricos forem os componentes, menos componentes são necessários para cobrir um dado domínio. Tal fato facilita a sua posterior seleção e aumenta a possibilidade de se utilizar a abordagem baseada em componentes no desenvolvimento.

3- COMPONENTES DE SOFTWARE

O termo componente de software está bastante difundido na engenharia de software e alguns autores utilizam esse termo para se referir a pedaços do código fonte (funções), estruturas de dados e classes. Outros chamam de componentes instâncias de classes (objetos) de um programa que podem ser utilizadas por outras instâncias.

Pode-se dizer que um componente de software é uma entidade arquitetural que :

1. Encapsula um subconjunto de funcionalidades do sistema e/ou dados;
2. Restringe acesso para este subconjunto via uma interface explicitamente definida;
3. Tem dependências explicitamente definidas em seu contexto de execução.

Assim, componentes de software podem ser vistos como pequenas “partes”, extremamente bem-definidas, que podem ser combinadas em diferentes “sistemas” mais complexos. Por isso muitos autores fazem analogia de componentes de software com “blocos de LEGO”.

A ideia de reuso por meio de componentes de software é baseada no reuso de classes no desenvolvimento orientado a objetos, mas cumpre mais efetivamente o seu papel no reuso, pois os objetos e classes possuem comportamentos de encapsulamento, mas não traz noção de independência como a definição de componentes (Szyperski, 2002, p. 11).

Uma característica notável de um componente de software é que esse deve possuir projeto e implementação próprios e oferecer interfaces bem definidas para o meio externo, de modo que, uma vez que as tarefas a serem realizadas estejam implementadas em um componente, não é necessário que se conheça os detalhes dessa implementação para poder utilizá-lo. Basta saber quais dados devem ser passados para o componente para sua execução e qual o retorno esperado na chamada ao componente. Somerville (2004, p. 263) define um componente como uma entidade executável independente em que o código-fonte não está disponível e não é compilado junto com o restante do sistema.

Mann e Kaur (2010, p. 105) definem um componente como um artefato de software escrito e compilado com objetivo de ser reutilizado em vários projetos. Quanto mais esse for reutilizado, mais rápido atingirá a maturidade, pois isso faz com que seja mais testado e tenha suas falhas corrigidas. Um componente deve ser um elemento independente no sistema podendo ser substituído por outro componente com a mesma interface e com a mesma função definida no contexto em que foi inserido. Em geral componentes são projetados para fornecer certo nível de serviço em uma aplicação. São desenvolvidos por especialistas no domínio ao qual o componente é proposto, levando a uma vantagem significativa ao fazer reuso dessa solução em vários projetos (Sommerville, 2004, p. 261).

3.1 COMPONENTES COMO SUBSISTEMAS

Componentes podem ser descritos como pedaços pré-definidos de software com interfaces (entrada e saída) e comportamento bem definidos e documentados, que podem ser reutilizados muitas vezes em diferentes aplicações (Stojanović, 2005, p. 22). Essencialmente, são objetos em um alto nível de abstração que podem se comportar efetivamente como soluções “caixa preta” (subsistemas independentes), para as

necessidades de um sistema específico, fornecendo serviços para uma arquitetura de aplicações bem definida (Szyperski, 2002, p. 39). Assim é possível implementar partes de um sistema em um componente e este, uma vez invocado, retornará algum valor ou realizará algum trabalho com autonomia própria.

A empresa desenvolvedora de componentes poderá oferecer, como solução, implementações parciais de sistemas, distribuídas em pacotes para atender as necessidades parciais de um sistema maior, e esses pacotes de componentes deverão passar por um processo de "integração" para ser incorporado no sistema final. Dessa forma componente de software atuarão como soluções semi-prontas para atender a uma necessidade de negócio, sem a necessidade de se investir na construção completa de uma solução de software.

4- PROCESSO DE SOFTWARE COM REÚSO DE COMPONENTES

Para fazer reuso de componentes em um projeto deve ser possível encontrar ou desenvolver componentes em domínios específicos do sistema e certificar-se de que esses são confiáveis. Esses componentes também devem ser bem documentados para que os desenvolvedores integradores possam compreender seu funcionamento, bem como conhecer detalhes sobre a trajetória de reuso e evolução desse componente (Sommerville, 2004, p. 261). Por outro lado, quando se utiliza um componente para fornecer um serviço a um sistema, os desenvolvedores integradores não precisam se preocupar com detalhes de implementação, como esse componente executa tal serviço, ou em que linguagem de programação ele foi desenvolvido (Sommerville, 2004, p. 263). Quando um sistema deve fornecer algum serviço, basta chamar a funcionalidade do componente por meio de sua interface externa para que este execute o serviço solicitado.

Sommerville (2004, p. 267, 268) descreve duas formas de aquisição de componentes para reuso:

1. A utilização de componentes COTS (commercial off-the-shelf) que são componentes desenvolvidos por terceiros, geralmente por equipes

especializadas no domínio específico desses componentes. Os sistemas gerenciadores de bancos de dados (SGBD) são um exemplo típico desse reuso, são reutilizados em praticamente todos os projetos de software. Os SGBD se comunicam com os sistemas por meio de sua interface, a linguagem SQL.

2. O desenvolvimento de componentes localmente para reuso. Nesse caso podem-se construir componentes confiáveis com base em componentes COTS já existentes no mercado, com a vantagem de estar de posse do código fonte do componente para facilitar futuras alterações e manutenções nesse componente para adaptar-se a novos requisitos que possam aparecer.

O uso de componentes COTS traz grandes vantagens com relação ao tempo para integração bastando apenas entender a documentação, enquanto que desenvolver componentes demanda tempo e equipe especializada para implementação do componente. Em contrapartida, alguns engenheiros de software preferem desenvolver seus próprios componentes por não confiarem em software desenvolvido por outros e por prever futuros problemas de manutenção no sistema com a falta de código fonte e falta de suporte técnico do desenvolvedor de COTS e também devido a problemas de interoperabilidade entre plataformas de desenvolvimento e sistemas operacionais. Essa escolha deve levar em conta requisitos de qualidade, produtividade e custo final do projeto (Pressman, 2006, p. 675).

O processo de Engenharia de Software Baseada em Componentes (ESBC) é similar à engenharia de software convencional sendo incluídas no processo algumas atividades específicas que apoiam o reuso de componentes. Entre a análise de requisitos e o projeto arquitetural da aplicação, existem as atividades de procurar componentes reutilizáveis, desenvolver partes não atendidas por componentes e integrar componentes ao sistema.

A tabela abaixo mostra um comparativo de atividades na ESBC, tomando como base um modelo em convencional em cascata (waterfall):

Engenharia de Software	ESBC
• Análise dos requisitos. • Especificação. • Aprovação da especificação pelo cliente.	• Análise dos requisitos. • Especificação. • Aprovação da especificação pelo cliente.
• Projeto de Arquitetura do Software. • Implementação.	• Busca e Seleciona de Componentes. • Desenvolvimento de partes não atendidas por componentes. • Integração.
• Testes.	• Testes de integração.
• Implantação	• Implantação

Tabela 1 - Modelo de Processo para a CBSE

5- ESTUDO DE CASO

Para exemplificar um cenário de reuso, foi desenvolvido um componente que implementa métodos para serem consumidos por aplicações comerciais. Para isso, será abordado o projeto desse componente que possui alguns métodos para acesso aos serviços disponíveis nos servidores da SEFAZ (secretaria da fazenda) para emissão de NF-e.

5.1 METODOLOGIA

A proposta de desenvolvimento do componente partiu da visão de que esses métodos eram frequentemente implementados em diversas aplicações comerciais e sistemas ERP em muitas empresas de desenvolvimento de software, e devido a complexidade dos requisitos, demandavam grandes custos de desenvolvimento e envolviam equipes especializadas nesse domínio tornando inviável alguns projetos. Um componente capaz de encapsular e esconder essa complexidade e que possa ser integrado facilmente a um sistema em desenvolvimento, teria alto grau de reuso nesse

cenário.

Para definir uma arquitetura e plataforma de desenvolvimento para o projeto foi realizado levantamento das tecnologias mais utilizadas por algumas empresas para o desenvolvimento de aplicações comerciais e sistemas ERP para que o componente proposto possa ter maior grau de reuso, assim poderá ser distribuído mais vezes, justificando o seu desenvolvimento.

5.2 ANÁLISE DE REQUISITOS

A análise de requisitos foi realizada por meio da documentação fornecida pela SEFAZ para utilização do serviço e Manual de Integração do Contribuinte disponível no site nacional do projeto (www.nfe.fazenda.gov.br). Estes documentos definem as especificações e critérios técnicos necessários para a integração entre os portais das SEFAZ dos Estados e os sistemas de informações das empresas emissoras de NF-e.

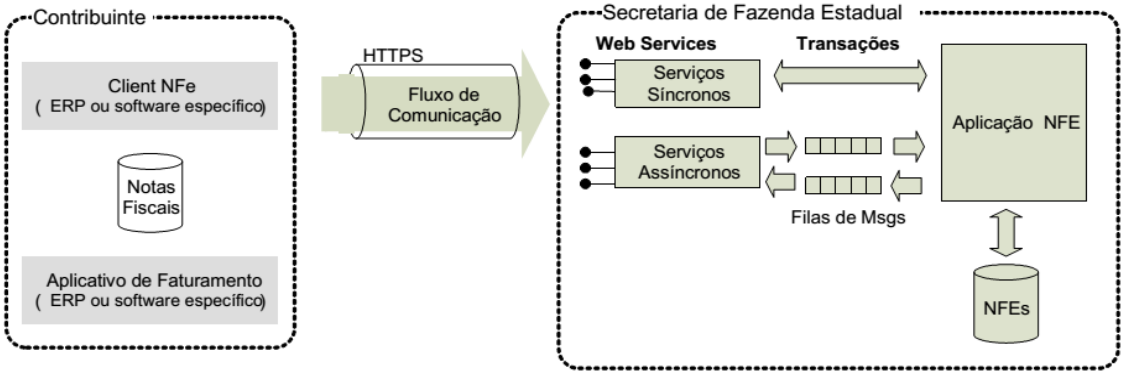
De maneira simplificada, os métodos do componente criado deverá gerar um arquivo xml contendo as informações fiscais de uma operação comercial, assinará digitalmente esse arquivo, para garantir a integridade dos dados e a autoria do emissor. Este arquivo, que corresponderá à NF-e, será então transmitido após uma pré-validação da estrutura do arquivo, para os servidores da SEFAZ do Estado do contribuinte emitente, que receberá o arquivo e devolverá um código de recibo. Com esse código de recibo o componente deve aguardar o processamento pelos servidores da SEFAZ (alguns segundos) e realizar a consulta do processamento do documento. Caso receba um retorno de “Autorizado o Uso”, irá executar o método de impressão do DANFE (documento auxiliar da NF-e) e só então poderá haver o trânsito de mercadoria.

Outros serviços também são disponibilizados pelas SEFAZ como cancelamento de NF-e, inutilização de numeração, consulta cadastro do emitente e consulta status do serviço, sendo para cada serviço um web service específico. Esses métodos também foram implementados no componente proposto.

O fluxo de comunicação para utilização dos serviços será sempre iniciada pelo componente através do envio de uma mensagem xml com a solicitação do serviço desejado e a recepção de uma mensagem de resposta do web service pelo componente. Essa comunicação deverá ser feita no padrão SOAP via protocolo SSL para garantir um canal seguro entre aplicação e servidor, com autenticação mútua.

Para assinar as mensagens enviadas para o servidor, o componente utilizará certificado digital contendo CNPJ do emitente autorizado a emitir NF-e.

A figura abaixo ilustra a arquitetura de comunicação entre uma aplicação que utiliza o componente desenvolvido (aplicativo de faturamento) e os servidores da SEFAZ:



Fonte: *Manual de Integração do Contribuinte da NF-e*

Figura 1 - Arquitetura de comunicação entre aplicação e servidores.

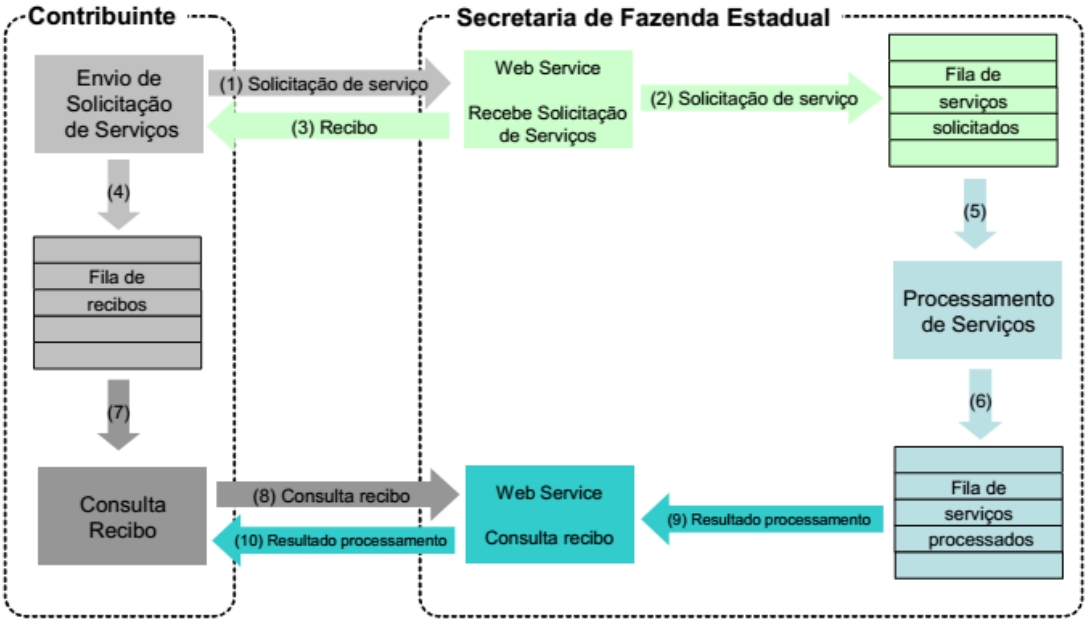
Entre os serviços fornecidos pelos web services da SEFAZ existem serviços síncronos e assíncronos como mostra a imagem acima. Nos serviços síncronos a resposta com o resultado do processamento é devolvida na mesma conexão da solicitação realizada pelo método do componente, enquanto que nos serviços assíncronos a resposta com o resultado do processamento deve ser consultada pelo estabelecimento de uma nova conexão pelo componente. Assim para consumir os serviços assíncronos, a aplicação deve chamar um método do componente para solicitação do serviço e logo após deve fazer outra chamada ao método de consulta processamento da solicitação.

A figura abaixo representa o fluxo simplificado de solicitação a um serviço síncrono onde existe apenas uma conexão com o web service para completar a solicitação do serviço:

Fonte: *Manual de Integração do Contribuinte da NF-e*

Figura 2 - Fluxo de solicitação de serviço síncrono.

Abaixo a ilustração do fluxo de solicitação a um serviço assíncrono, onde a aplicação cliente deverá chamar o método de solicitação do serviço, onde é retornado um recibo com um código numérico de 8 dígitos. Esse recibo deve ser armazenado e colocado em fila para enviar como parâmetro do método de consulta de processamento da solicitação numa segunda conexão com o servidor, esse recibo irá identificar a solicitação na chamada anterior.



Fonte: *Manual de Integração do Contribuinte da NF-e*

Figura 3 - Fluxo de solicitação de serviço assíncrono.

O componente desenvolvido deverá implementar todos os métodos de acesso a esses web services e mais alguns métodos adicionais para execução de tarefas relacionadas a emissão de NF-e. O objetivo é implementar uma biblioteca de funções no componente que poderá ser usado em diversas aplicações que emitem esse documento

fiscal, escondendo assim a complexidade dessa implementação dos desenvolvedores de aplicações.

A tabela abaixo mostra a relação de métodos implementados no componente desenvolvido:

Método	Serviço
GeraNFe	Gera arquivo xml com os dados na NF-e a ser transmitido
AssinarXML	Assina arquivo xml NF-e digitalmente
ValidarXML	Realiza pré-validação do arquivo assinado
NfeRecepcao	Transmite arquivo xml assinado (serviço assíncrono)
NfeRetRecepcao	Consulta processamento do método NfeRetRecepcao
NfeConsulta	Consulta situação de uma NF-e (serviço síncrono)
NfeStatusServico	Verifica status dos servidores do sefaz (serviço síncrono)
NfeInutilizacao	Inutiliza numeração de NF-e (serviço síncrono)
NfeCancelamento	Cancela NF-e emitida anteriormente (serviço síncrono)
DistribuicaoNfe	Gera arquivo xml com dados para impressão
EnviaEmail	Envia email para o destinatário da NF-e
ValidaAssinatura	Verifica a assinatura realizada pelo método AssinarXML
GeraCCe	Gera arquivo xml com carta de correção eletrônica para uma NF-e emitida anteriormente com erro nos dados
NFeRecepcaoEvento	Transmite o arquivo xml da CC-e (serviço síncrono)

Tabela 2 – Relação de métodos do componente.

5.3 IMPLEMENTAÇÃO

Esse componente foi desenvolvido na plataforma **COM+** (Component Object Model) em **.NET** e poderá ser utilizado por outros sistemas independente da linguagem de programação e arquitetura utilizada, pois na plataforma **COM+** o **.NET** permite interoperabilidade com outros sistemas desenvolvidos em outras plataformas (como ASP, VB, VBScript, Delphi ou até mesmo Ole Automation do SQL Server) que podem reutilizar objetos sem a necessidade de se conhecer sua implementação interna.

Para a implementação do componente foi utilizado o MS Visual Studio® 2010

para criar um projeto do tipo **Class Library** escrito em **C#**, para dar origem a uma DLL e, consequentemente, ser hospedada dentro do **COM+**. Os métodos implementados realizarão algumas rotinas da camada de negócio em aplicações, reduzindo assim o esforço em desenvolver várias aplicações que suportem essa estrutura.

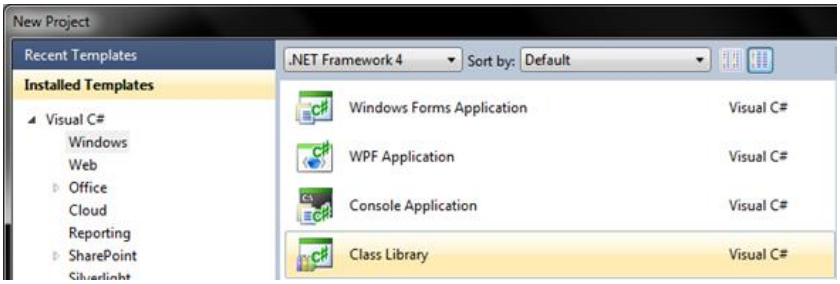


Figura 4 - Criação de Projeto tipo Class Library no MS Visual Studio 2010

Adicionamos a referência ao Assembly System.EnterpriseServices.dll no projeto e, consequentemente, esse componente suportará funcionalidades COM+.

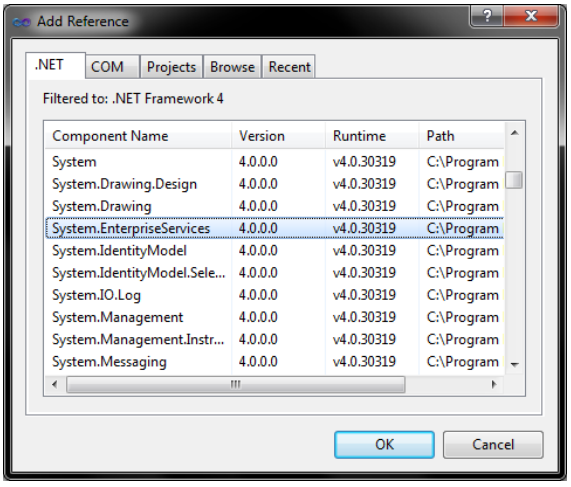


Figura 5 - Adicionando referência para que o componente Suporte funcionalidades COM+

A esse projeto foram adicionadas algumas referências para acessar todos os

métodos disponíveis no web service via protocolo http. Esses métodos enviam e recebem mensagens em forma de strings xml empacotadas em protocolo SOAP para a troca de informações entre componente e Web Service.

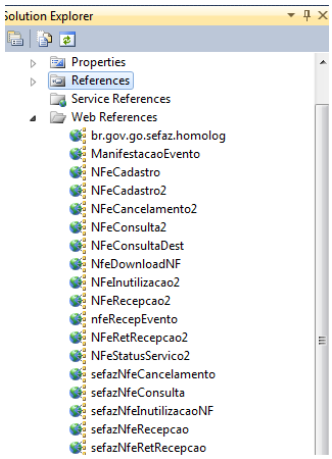


Figura 6 - Adicionando referências web específicas para o projeto

Nas propriedades do projeto, deve estar selecionada a propriedade "Assembly Information", especificando o Assembly como COM-Visible para que seja visível a aplicações COM+:

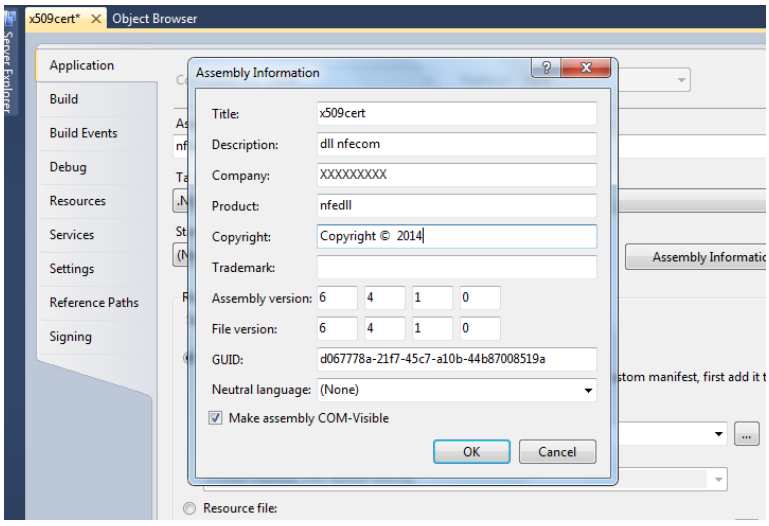


Figura 7 - Configurando o assembly como COM-Visible

.

Para disponibilizar o componente para aplicações **COM+**, foram explicitamente implementadas interfaces com as assinaturas dos métodos pois componentes **COM+** não podem expor membros diretamente e devem ter interfaces implementadas para chamadas externas aos métodos. Assim foi escrita uma classe chamada **Iutil** que possui uma interface para todos os métodos que serão implementados :

```
using System;
namespace x509cert
{
    interface Iutil
    {

        string AssinarXML(string pathArquivo, string noSerieCert, string tagUri);
        bool ValidarXML(string pathXML, string pathSchema);
        string NfeRecepcao(string noSerieCert, string hostWS, string pathLoteXML, bool trace);
        string NfeStatusServico(string cUF, string tpAmb, string noSerieCert, string hostWS, bool trace);
        string NfeInutilizacao(string noSerieCert, string hostWS, string pathXML, bool trace);
        string NfeCancelamento(string tpAmb, string noSerieCert, string hostWS, string pathXML, bool trace);
        string NfeConsulta(string tpAmb, string noSerieCert, string hostWS, string chNFe, bool trace);
        string NfeRetRecepcao(string tbAmb, string noSerieCert, string hostWS, string nRec, bool trace);
        string DistribuicaoNFe(string chNFe);
        bool EnviaEmail(string emailDest, string assunto, string msg, string pathFile);
        string NfeCadastro(string UF, string cnpj, string noSerieCert, bool trace);
        bool ValidaAssinatura(string fileNFe);
        string GeraCCe(string chnfe, string seq, string tpEvento, string dhEvento, string xCorrecao);
    }
}
```

Figura 8 - Interface com declaração de métodos para chamadas externas

Após a definição de uma interface, é necessário ter uma classe que implementa todos os métodos da interface. Foi criada uma classe chamada **Util** para comportar esses métodos. Nessa classe devem ser adicionadas algumas referências da plataforma .NET que serão utilizadas pelos métodos e por fim todos os métodos definidos na interface **Iutil** foram implementados.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Xml.Schema;
using System.Security.Cryptography.Xml;
using System.Security.Cryptography.X509Certificates;
using System.IO;
using System.Runtime.InteropServices;
using System.Windows.Forms;
```

```
using System.Text.RegularExpressions;
using Microsoft.Win32;
using System.Net.Mail;
using System.Net;
using System.Net.Sockets;
using System.Collections;
using System.IO.Compression;
namespace x509cert
{
      [ClassInterface(ClassInterfaceType.AutoDual)]
      [ProgId("x509cert.util")]
      [ComVisible(true)]
      public class util : x509cert.Iutil //implementa a interface
Iutil
      {
        //os  métodos  da  interface  Iutil  foram  implementados
aqui...}
 }
```

Figura 9 – Estrutura da classe Util para implementar os métodos do componente

Ao compilar esse projeto, o MS Visual Studio® gera um arquivo .dll (biblioteca de funções) e poderá ser reutilizada por várias aplicações que necessitam invocar os métodos implementados. Para a distribuição, devem ser incluídos alguns artefatos (estrutura de pastas, arquivos de configuração, arquivos utilitários, documentação, exemplos de uso) e estes devem ser empacotados, versionados e catalogados para que possa ser facilmente encontrado e distribuído para reuso (SOFTEX, 2007, p. 19).

Alguns autores sugerem o uso de ferramentas de apoio ao reuso para auxiliar no armazenamento e manutenção de componentes desenvolvidos ou adquiridos com o objetivo de formar uma biblioteca de reutilização. Essas ferramentas fornecem mecanismos para gerenciar os componentes organizando, armazenando e permitindo a existência de múltiplas versões do mesmo componente tornando-os disponíveis para os projetistas de sistemas (Rossi, 2004, p. 38).

5.4 REUTILIZANDO O COMPONENTE

Com o componente desenvolvido, uma aplicação cliente irá fazer chamadas a esses métodos externos da mesma forma como se esses métodos fossem implementados dentro do código da aplicação. Essa tarefa às vezes não é trivial, mas sempre exigirá menos esforço do que desenvolver todo o código do componente dentro da aplicação. É possível fazer uso desses métodos analisando a documentação e exemplos fornecidos no pacote do componente. Assim o desenvolvedor integrador saberá quais parâmetros passar, tipos de dados de envio e de retorno, mensagens de erro devolvidas pelo componente, e até debugar o código de sua aplicação fazendo chamadas ao componente e recebendo dados de retorno.

A seguir temos uma aplicação de exemplo desenvolvida para testar os métodos do componente. Essa foi distribuída dentro do pacote do componente para demonstrar a utilização para os desenvolvedores de aplicações (integradores). Essa aplicação foi desenvolvida em C# mas esse componente pode ser utilizado por aplicações desenvolvidas em várias linguagens de programação e até mesmo por sistemas legados desenvolvidos em Delphi ou Visual Basic.

A tela inicial conterá comandos para cada método do componente, instruções de uso e um console que exibe as mensagens de retorno com os resultados da execução (status).

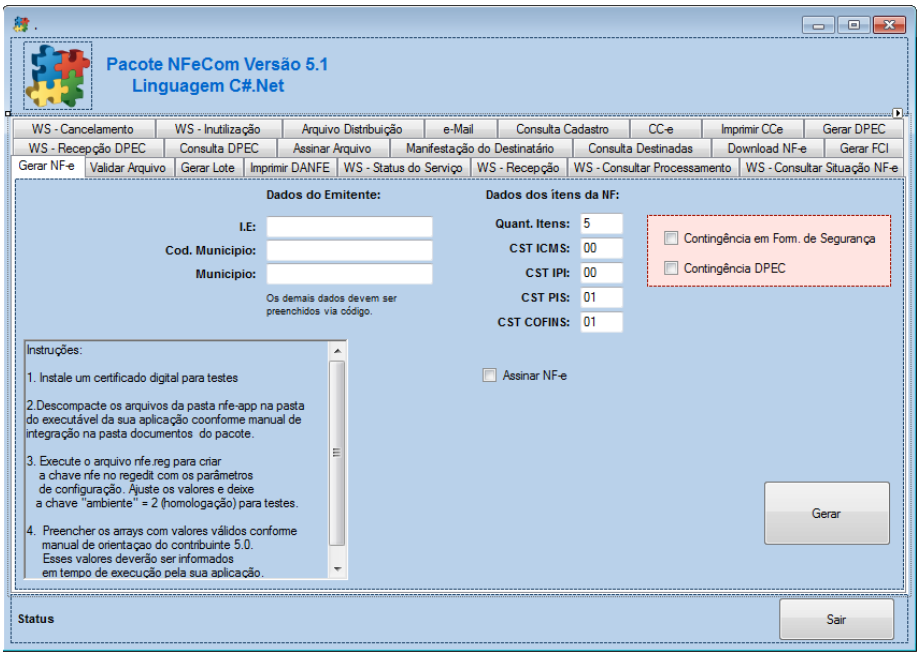


Figura 2 - Aplicação cliente de Exemplo

Nessa aplicação exemplo, o usuário deve preencher todos os dados seguindo as instruções do pacote, e clicando no botão correspondente irá invocar cada método do componente. Acima temos a tela inicial que mostra o botão “Gerar” que irá gerar um arquivo xml em determinada pasta para posteriormente ser usado pelos método do componente.

A seguir exemplos de código para invocar alguns métodos do componente (assinaturas descritas na fig. 6) a partir da aplicação cliente:

1. Assinar arquivo xml gerado pela aplicação para que seja transmitido posteriormente.

```
private void assinar_Click(object sender, EventArgs e)
    {
// caminho do arquivo e numero de série do certificado digital (deve existir um certificado digital instalado na maquina para execução do método)
String caminhoXml = Application.StartupPath + "\\nfe\\arquivos";
String numCertificado = ""

// informa ao usuário a operação que está sendo realizada
 lblStatus.Text = "Assinando arquivo, aguarde...";

// instancia um objeto com os métodos do componente
    nfec.nfecsharp nfe = new nfec.nfecsharp();

// faz a chamada ao método AssinarArquivoXML passando os 3 parâmetros necessários. O método em questão retorna uma string, usamos esse valor para exibir uma mensagem ao usuário.

lblStatus.Text = nfe.AssinarArquivoXML(caminhoXml, numCertificado, "infNFe");
}
```

2. Realizar pré-validação de arquivo xml (verifica estrutura das tags e dados)

```
private void validar_Click(object sender, EventArgs e)
    {
//caminho do arquivo
String caminhoXml = Application.StartupPath + "\\nfe\\arquivos\\assinado";
// informa ao usuário a operação que está sendo realizada
    lblStatus.Text = "Validando arquivo, aguarde...";

// instancia um objeto com os métodos do componente
nfec.nfecsharp nfe = new nfec.nfecsharp();

// faz a chamada ao método ValidarArquivoXML passando os 3 parâmetros necessários. O método em questão retorna um valor booleano, assim utilizamos esse valor de retorno para mostrar uma mensagem ao usuário.

 if (nfe.ValidarArquivoXML(caminhoXml, caminhoXsd, true))
      lblStatus.Text = "Validação concluída, nenhum erro identificado.";
else
      lblStatus.Text = "Problemas identificados na validação";
}
```

3. Transmitir arquivo validado e assinado para o webservice de recepção

```
private void enviar_Click(object sender, EventArgs e)
    {
      lblStatus.Text = "Aguarde...";

      //caminho do arquivo assinado e validado
```

```
            string caminhoXmlAssinado = Application.StartupPath +
"\\nfe\\lotes";
            //endereço do host do web service de recepção
            String                    hostWs                    =
"https://nfe.sefazvirtual.rs.gov.br/ws/Nferecepcao/NFeRecepcao2.
asmx";
            String numCertificado = "";

            //instancia um objeto com os métodos do componente
            nfec.nfecsharp nfe = new nfec.nfecsharp();

            // faz a chamada ao método NfeRecepcao passando 4
            parâmetros para o componente. O método em questão
            retorna uma string com o número do protocolo de
            recebimento do arquivo. Guardamos esse valor em uma
            variável para em seguida consultar o processamento do
            arquivo no servidor remoto.

            txtRecepcao.text   =   nfe.NfeRecepcao(numCertificado,
hostWs, caminhoXmlAssinado, false);
          }
```

4. Consultar o processamento do arquivo no servidor remoto

```
      private void consultaProc_Click(object sender, EventArgs e)
          {
            lblStatus.Text = "Aguarde...";

            //instancia um objeto com os métodos do componente
            nfec.nfecsharp nfe = new nfec.nfecsharp();

            // faz a chamada ao método NfeRetRecepcao passando 5
            parâmetros para o componente. O método em questão
            retorna uma string com o número do protocolo de
```

```
        recebimento do arquivo. Guardamos esse valor em uma
        variável para em seguida consultar o processamento do
        arquivo no servidor remoto.

        txtConsProces.Text      =      nfe.NfeRetRecepcao("1",
        numCertificado, hostWs, txtRecepcao.text,     false);

    }
```

Nesse exemplo podemos observar que com poucas linhas de código podemos executar tarefas complexas onde a complexidade é implementada no componente, poupando os desenvolvedores de aplicações de conhecer detalhes dessa implementação e tornando o desenvolvimento mais rápido e com baixo custo.

5.5 CONTROLE DE VERSÕES DO COMPONENTE

Frequentemente um componente sofre atualizações seja por mudanças de requisitos, melhorias ou correções de falhas. Quando ocorre alterações no código, uma nova versão deve ser distribuída para todas as aplicações clientes que o utilizam. Pela definição de componentes de software, essas alterações no código do componente devem implicar em poucos ou nenhum impacto na aplicação cliente, ou seja, este componente deve ser substituído sem que sejam necessárias alterações significativas no código da aplicação que o reutiliza. Além disso, um sistema pode ser composto de vários componentes e esses devem ser independentes e substituíveis na aplicação, dessa forma o reuso de componentes sugere uma melhor adaptação da aplicação a mudanças de requisitos (Rossi, 2004, p. 92).

Uma forma de atribuir versão a um componente, a exemplo do código acima, seria informar a versão a cada nova build.

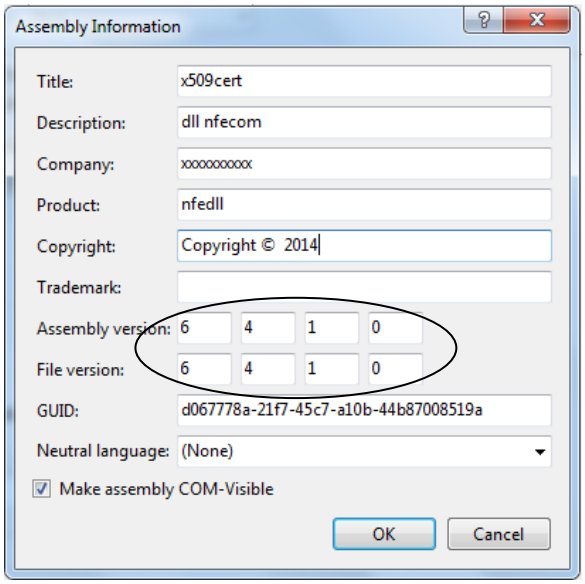


Figura 3 - Definindo versão para a build a ser compilada

Para controle de versões faz-se uso de um repositório de componentes que é uma base de dados onde informações sobre os componentes são armazenadas de forma organizada permitindo a classificação, busca e recuperação dessas informações para reuso (Rossi, 2004, p. 39).

Para controlar versões, correções e melhorias realizadas no componente desenvolvido, foi incluído na documentação do pacote um arquivo de texto (leiame.txt) que acumula todas as alterações realizadas por versões de distribuição do componente. Esse recurso possibilitou que os desenvolvedores pudessem acompanhar a evolução do componente e reconhecer até onde a sua versão está atualizada facilitando o reuso deste componente a longo prazo.

5.6 RESULTADOS E DISCUSSÃO

O projeto e implementação do componente proposto exigiu esforço concentrado no domínio das regras de negócio do componente. Devido a complexidade dos requisitos, houve necessidade dos desenvolvedores em adquirir conhecimentos específicos no domínio em questão e de se fazer um estudo detalhado de documentos disponíveis no site do projeto NF-e e da legislação tributária e fiscal vigente.

Uma vez desenvolvido e testado, o componente foi empacotado e distribuído (comercializado) para empresas de software em todos os estados. A arquitetura proposta

trouxe um alto grau de interoperabilidade entre diferentes plataformas de desenvolvimento, dentre elas, as mais utilizadas para o desenvolvimento de aplicações de gestão comercial: Delphi, Visual Basic 6, C#.net e VB.net.

Outra vantagem observada no uso desse componente é que, como os requisitos e regras de negócio mudam frequentemente, dispondo-se de uma equipe especializada para atualizar o componente com as novas regras, tornou mais fácil e rápida essa atualização, pois os desenvolvedores de aplicação apenas necessitam obter a nova versão do componente e substituir no seu sistema com poucas ou nenhuma alteração no seu código fonte, evitando assim o surgimento de defeitos decorrentes de modificações no código, e entregando rapidamente a aplicação atendendo às novas especificações.

6- CONSIDERAÇÕES FINAIS

Naturalmente, o grau de reuso de componentes deve variar em função do número de novas aplicações no mesmo domínio. Para que o reuso aconteça é necessário que existam componentes disponíveis ou uma infraestrutura técnica capaz de desenvolver e manter componentes reutilizáveis. O DBC é uma abordagem que traz grandes benefícios para soluções de problemas de custo e prazo de desenvolvimento mas deve-se fazer uso controlado desses recursos. Para que o reuso ocorra com sucesso os componentes reutilizáveis devem estar bem documentados com relação a sua interface, requisitos atendidos e sobre como integrá-los a uma aplicação.

Esse trabalho lançou luz sobre algumas vantagens relativas ao reuso de componentes no desenvolvimento de aplicações, mostrando um exemplo típico de um componente desenvolvido para reuso. Muitos aspectos sobre a componentização ainda podem ser explorados para difundir as vantagens competitivas em relação às abordagens tradicionais de desenvolvimento. Os processos definidos pela ESBC ainda são pouco utilizados na prática, embora o reuso de componentes de forma desestruturada ocorra em grande escala em organizações desenvolvedoras de software.

7- REFERÊNCIAS BIBLIOGRÁFICAS

BASS, L, et alli. Volume I: Market Assessment of Component-Based Software Engineering. Technical Note CMU/SEI-2001-TN007, Software Engineering Institute, Carnegie Mellon University, EUA, 2001, disponível em http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01tn007.pdf

Gamma, Erick.; Helm, Richard.; Johnson, Ralph.; Vissides, John.; Design Patterns: Elements of Reusable Object-Oriented Software. 416p. 1997, IEEE Software.

Gurp , J. van; Bosch, J.; Design, Implementation and Evolution of Object Oriented Frameworks: concepts and guidelines. SOFTWARE PRACTICE AND EXPERIENCE, 2001; 31:277–300.

Jasmine, K.S.; Vasantha. R.; A New Capability Maturity Model For Reuse Based Software Development process. IACS,IT- International Journal of Engineering and Technology, vol. 2, nº 1, February, 2010 ISSN: 1793-8236.

Mann, Kulvinder Singh; Kaur, Arvinder; Component Based Software Engineering. International Journal of Computer Applications (0975 – 8887), v. 2, nº 1, May 2010, p. 105.

Pressman, Roger S.; Engenharia de Software - 6ª Edição - 720p. 2006. ISBN:8586804576, 9788586804571 Proceedings ..2006.

Rossi, Ana Claudia. Representação do componente de software na Farcsoft: Ferramenta de apoio à reutilização de componentes de software. Dissertação de Mestrado. Escola Politécnica de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais – São Paulo. 2004. 236p.

Santos, Rafael. Introdução à Programação Orientada a Objetos Usando Java. - 2ª Ed. 2013. 319 p.

Softex, Perspectivas de desenvolvimento e uso de componentes na indústria brasileira de software e serviços. Campinas, 2007. 40 p.

Sommerville, Ian. Engenharia de Software. 8a Edição, 551p. 2004.

Stojanovié, Z.; A Method for Component-Based and Service-Oriented Software Systems Engineering. Doctoral Dissertation, Delft University of Technology.The Netherlands ISBN: 90-9019100-3, 2005.

Szyperski, C.; Gruntz, D.; Murer, S.; Component Software: Beyond Object-Oriented Programming. ISBN 0-201-74572-0 2nd edition, 2002.

Werner, C.M.L. & Braga, R.M.M.B (2005) "A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes", in: Desenvolvimento Baseado em Componentes, Gimenes, I.M.S. & Huzita, E.H.M. (eds), Editora Ciência Moderna, Rio deJaneiro, 2005. ISBN 85-7393-406-9, pg. 57-103.