6
Conclusões e Trabalhos Futuros

Neste trabalho foi visto que frameworks associados a linguagens orientadas a objetos que suportam a programação orientada a componentes, normalmente, usam abstrações das próprias linguagens OO para definir uma interface de programação para construção e uso dos componentes. Como consequência, o código fonte mistura aspectos da funcionalidade do componente com os mecanismos de implementação específicos do modelo de programação, o que impede a reutilização deste componente em outros frameworks, além de incluir uma complexidade extra no código.

Para contornar este problema, foi estudada a adoção da técnica de programação orientada a atributos juntamente com uma linguagem orientada a objetos para construção de aplicações baseadas em componentes de software. Como parte do estudo, foi desenvolvido o ASCS, um mecanismo de programação baseado em atributos para a versão Java do middleware SCS [7].

O modelo de programação ASCS permite a codificação de componentes de forma que os elementos do código que fazem referência ao modelo de componentes e à infra-estrutura utilizada se concentrem na camada de anotações. Além da codificação do componente se tornar mais fácil e intuitiva, a abordagem com anotações permite que um mesmo componente ASCS possa ser utilizado em uma outra possível implementação Java do modelo de componentes SCS, desde que dê suporte às anotações do ASCS.

Através de uma avaliação qualitativa utilizando as métricas do CDN [21, 29], comparamos o modelo de programação original do SCS com o ASCS. De forma geral, podemos concluir que um modelo de programação baseado em anotações tende a proporcionar uma maior visibilidade e uma menor viscosidade do código. Além disso, a eliminação de classes e interfaces da API permite uma redução do número de abstrações com as quais o usuário precisa interagir. Porém, um framework orientado a objetos convencional tende a proporcionar uma maior flexibilidade, já que diversas classes e métodos podem ser utilizados ou até redefinidos para contornar situações que nem sempre podem ser previstas.

Vimos que o modelo original do SCS introduz uma série de dependências

ocultas, além de uma grande quantidade de abstrações que apresentam problemas de proximidade com o domínio. O ASCS disponibiliza um mecanismo de verificação do código do componente em tempo de compilação que reduz as dependências ocultas da API. Além disso, cada uma das anotações do ASCS correspondem a um conceito do modelo de componentes, o que deixa a API mais intuitiva, evitando problemas de proximidade com o domínio.

Quando utilizamos a abordagem de anotações, é esperado que, pelo fato do processamento ser baseado em reflexão computacional, um *overhead* de desempenho seja introduzido. Através de testes de desempenho comparando os dois modelos de programação, verificamos que a operação de carga do componente oferece uma diferença de performance. Porém, deve-se considerar que a operação de carga é executada apenas uma vez, na hora da implantação do componente. Deve-se considerar também que a diferença absoluta de tempo (na casa de milisegundos) normalmente não oferecerá uma preocupação real em termos de desempenho no momento da implantação do componente.

Através de uma avaliação quantitativa, verificamos também, que um modelo de programação baseado em anotações permite o desenvolvimento de componentes através de uma menor quantidade de linhas de código.

As operações referentes à controle de ciclo de vida, hoje, ainda não são implementadas através de uma faceta SCS de fato. Ou seja, não existe uma interface definida em um arquivo IDL para agregar as operações de ciclo de vida. Por conta da arquitetura atual do SCS, as operações de ciclo de vida estão divididas nas interfaces *IComponent* e *ComponentLoader*. Futuramente, deverá ser criada uma nova interface *LifeCycle* para agregar as operações específicas de tratamento de ciclo de vida de um componente.

O suporte a herança de anotações em componentes ASCS é uma funcionalidade importante a ser incluída futuramente. É comum a necessidade do desenvolvimento de componentes abstratos, onde as subclasses fornecem a implementação de algumas facetas do componente. Para esse caso, o núcleo do ASCS deve considerar todas as anotações referentes ao ASCS contidas nos componentes de mais alto nível hierárquico. Para isso, deve ser incluído um processamento específico para verificar tais anotações nas superclasses.

Além disso, como trabalho futuro pode ser estudada uma investigação da viabilidade de retirar as dependências com o middleware (CORBA) do modelo de programação ASCS. Com uma API independente do middlware utilizado, a infra-estrutura SCS poderia ser modificada, por exemplo, para utilizar outra tecnologia de comunicação, sem que o código dos componentes precisasse ser alterado.