Componentes de Software Baseados em Engenharia de Domínio

Leonardo Ciocari, Rafael Cancian

[1]Centro de Ciências Tecnológicas da Terra e do Mar (CTTMar)
Universidade do Vale do Itajaí (UNIVALI)
Email:{leonardociocari,cancian}@univali.br

Abstract. *Embedded systems are usually designed using a repository of reusable components. To be reusable, however, components must be designed for that. A common technique used to identify and design reusable components is the domain engineering. This paper presents the development of a software tool that starts from a domain model and generates a repository of components in a semi-automatic way. The repository includes the source-code and testers structure of components and the meta data for their characterization. As a study case a domain model for embedded operating system was used to generate a repository. Results show that a well done domain model with an adequate detail level allows the code generation and characterization of components and families of components that can really be used for automatically component selection and for designing several applications.*

Resumo. *Sistemas embarcados são comumente desenvolvidos a partir de um conjunto ou repositório de componentes reusáveis. Entretanto, para serem reusáveis, esses componentes precisam ser projetados com esse objetivo. Uma técnica utilizada para identificar e projetar componentes reusáveis é a engenharia de domínio. Este artigo apresenta o desenvolvimento de uma ferramenta que, a partir de um modelo de domínio gerado pelo usuário, realiza a geração semi-automática de um repositório de componentes, incluindo a infra-estrutura dos componentes, testadores e seus metadados para caracterização dos componentes e posterior seleção. Como estudo de caso utilizou-se um modelo de domínio que representa sistemas operacionais embarcados. Os resultados demonstram que uma boa modelagem do domínio com o nível de detalhamento adequado permite a geração de código e a caracterização de componentes e famílias de componentes que, efetivamente, podem ser utilizados por uma ferramenta para seleção automática de componentes e geração de diferentes aplicações embarcadas.*

1. Introdução

Por integrarem hardware e software, desenvolvedores de aplicações embarcadas costumam se preocupar não apenas com as próprias aplicações, mas também com o suporte de execução de sua aplicação e com a arquitetura de hardware na qual a aplicação executará. Entretanto, essa necessidade costuma levar a menos reusabilidade e portabilidade e ainda maior tempo de desenvolvimento. Algumas técnicas auxiliam no aumento da reusabildiade de artefatos de software, como o Projeto Baseado em Componentes (CbD), em que se desenvolvem componentes que possam ser reaproveitados em outros projetos. Entretanto para que um componente seja reusável, técnicas como a Engenharia de Domínio

devem preceder seu desenvolvimento, pois é nessa etapa que se faz o mapeamento das características do domínio da aplicação e a extração de componentes reusáveis.

Com um repositório de componentes reusáveis, desenvolvedores de aplicações podem utilizar os componentes como infra-estrutura básica para o desenvolvimento de novas aplicações. Além disso, projetistas precisam que os componentes sejam anotados com metadados que permitam sua seleção adequada e a estimativa de características como energia consumida, área de memória e tempo de execução.

Este artigo apresenta o desenvolvimento de uma ferramenta capaz de fazer a geração semi-automática de um repositório de componentes a partir de um modelo de domínio. Assim, o desenvolvedor de aplicação terá à sua disposição um repositório de componentes comuns ao modelo de domínio (e, portanto, reusáveis), preocupando-se apenas com a aplicação, e o desenvolvedor da infra-estrutura terá a estrutura básica dos componentes. Esta estrutura básica inclui arquivos com metadados, a estrutura em C++ do código dos componentes e de seus testadores.

Como estudo de caso foi utilizado um modelo de domínio de sistemas operacionais embarcados para a geração de um repositório de componentes. A escolha do domínio de sistemas operacionais embarcados justifica-se pois sistemas embarcados estão se tornando mais complexos e é necessário que haja o suporte adequado a aplicações complexas, como gerenciamento de processos, temporização, sincronização, gerenciamento de dispositivos e outros aspectos tipicamente encontrados nos sistemas operacionais.

Este artigo está organizado da seguinte forma: na Seção 2 é apresentada a fundamentação teórica, focando técnicas de projeto de sistemas embarcados. Na Seção 3 é apresentada a ferramenta a ser desenvolvida, sua arquitetura, funcionamento básico, requisitos e um estudo de caso. A seção 4 apresenta as considerações finais, incluindo as próximas etapas do desenvolvimento.

2. Fundamentação Teórica

O **Desenvolvimento Baseado em Componentes** (Component-based Design - CbD) melhora a produtividade e qualidade no desenvolvimento de um software, pois visa usar blocos de códigos ou serviços já existentes. Isto também facilita a atualização do software, pois basta substituir os componentes existentes por atualizados [YAU and DONG 2000]. Os componentes devem ser claramente caracterizados para permitir posterior seleção e exploração do espaço de projeto. Utilizando o CbD é possível criar componentes adequados à aplicação e disponibilizá-los num repositório de componentes.

O repositório de componentes é basicamente uma biblioteca de componentes de software/hardware reusáveis que devem ser suficientes às necessidades dos desenvolvedores de aplicações. Para que o repositório de componentes seja realmente útil na criação de aplicações, é preciso caracterizar seus componentes com informações (metadados) que incluem dependências entre componentes e a descrição de suas interfaces. Para sistemas baseados em componentes, o fluxo de projeto de sistemas embarcados pode ser dividido em fluxo de projeto da infra-estrutura e fluxo de projeto da aplicação (que utiliza a infra-estrutura já desenvolvida). No fluxo de projeto da infra-estrutura faz-se a modelagem do domínio, e para isso utilizam-se metodologias como a Engenharia de Domínio.

A **Engenharia de Domínio** é a base para o desenvolvimento de sistemas de soft-

ware reusáveis, pois identifica e documenta o que há em comum entre várias aplicações de um mesmo domínio, permitindo um curto ciclo de desenvolvimento para futuras aplicações [Subramanian 1997]. Trata-se de uma técnica que se concentra em produzir uma série de artefatos que possam ser reaproveitados em outros projetos [Pressman 2000], e recomenda-se que essa engenharia seja realizada em pelo menos três etapas: análise de domínio, especificação da infra-estrutura e a implementação desta infra-estrutura [Arango and Prieto-Diaz 1989]. A Engenharia de Domínio permite extrair a interface dos componentes, e estes, então, podem ser representados adequadamente utilizando o CbD. Por fim, os componentes irão popular um repositório, formando assim o repositório de componentes que será utilizado na geração de aplicações.

Sistemas operacionais embarcados fornecem o suporte de execução a aplicações. Eles abstraem a arquitetura de hardware em que estão executando e permitem que aplicações possam executar utilizando a interface provida pelo sistema operacional. De modo geral, sistemas operacionais projetados adequadamente permitem que aplicações possam acompanhar a evolução do hardware sem necessidade de grande esforço de desenvolvimento [Polpeta and Fröhlich 2004]. Para se atingir reusabilidade e portabilidade em um sistema embarcado, a separação em camadas com interfaces bem definidas é essencial.

3. Desenvolvimento

A figura 1 apresenta a arquitetura geral da ferramenta. Um modelo de domínio é representado utilizando um ou mais diagramas UML, que são lidos e analisados pela ferramenta para extrair seus componentes e disponibilizá-los no repositório de componentes, que é formado por componentes, seus testadores e metadados.

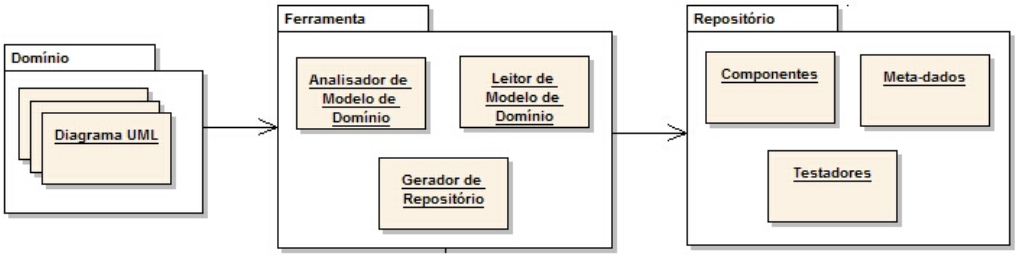

Figura 1. Arquitetura geral da ferramenta

Como estudo de caso, utilizou-se um modelo de domínio correspondente ao domínio dos sistemas operacionais embarcados para a geração de um repositório de componentes. A modelagem do domínio utilizada como entrada para a ferramenta foi feita com base no código-fonte do sistema operacional EPOS (Embedded and Parallel Operating System) [Fröhlich 2001], que não possuía repositório formal com metadados. Um modelo composto por diagramas de classe foi gerado através do software *Enterprise Architecture* (EA), usando a função de engenharia reversa sobre o código de componentes do sistema operacional EPOS. Esse modelo inicial foi adaptado e estruturado para tornar-se um modelo de domínio, representando as famílias de componentes e os níveis de abstração apresentados anteriormente. Esse modelo final foi exportado em XML e utilizado como parâmetro de entrada na ferramenta de geração de repositório. O repositório gerado inclui cerca de 50 componentes de software, dispostos em 21 pastas e em 71 arquivos.

A estrutura dos arquivos de metadados corresponde à maior parte da estrutura de componentes de software. Os arquivos basicamente contêm informações sobre componentes de software, como: família, nome da família, sua interface pública, quais são seus membros, seus métodos públicos e arquivo que contêm o código-fonte, entre outras. São gerados dois tipos de arquivos de metadados: (I) um arquivo principal, sempre denominado `metadata.xml` e armazenado na pasta raiz do repositório de componentes, e que contém informações sobre a estrutura do repositório dos outros arquivos de metadados; e (II) arquivos de metadados de cada família de componentes, e que contêm informações sobre ela e seus membros.

No primeiro quadro abaixo pode ser visto um pedaço do arquivo de metadados `metadata.xml` do repositório de sistemas operacionais embarcados gerado pela ferramenta, contendo informações das famílias de componentes extraídas do domínio e sobre os outros arquivos de metadados do repositório que as representam. O segundo quadro abaixo mostra um trecho do arquivo de metadados da família `thread`.

```
<repository domain="embedded operating system" version="1.0.0" date="">
   <...>
   <family name="thread" hierarchy="abstraction/process/thread"
            filename="./abstraction/process/thread/thread.xml" />
   <family name="scheduler" hierarchy="abstraction/process/scheduler"
            filename="./abstraction/process/scheduler/scheduler.xml" />
   <...>
</repository>
```

```
<...>
<family name="thread" hierarchy="abstraction/process/thread" interface_class="incremental">
   <dependencies>
      <dependency type="requisit" dependency="abstraction/process/scheduler"/>
      <dependency type="requisit" dependency="hardware_mediator/architecture/cpu" />
   </dependencies>
   <...>
   <member name="Thread">
      <source_codes>
         <local_file author="" filename="thread.h"
                     path="./abstraction/process/thread/thread.h" type="h" />
      </source_codes>
      <public_interface>
         <virtual_interface>
         <software_interface>
            <methods>
               <method name="join" qualifier="" returnType="int" />
               <method name="resume" qualifier="" returnType="void" />
               <method name="sleep" qualifier="" returnType="void">
                  <parameter name="q" defaultvalue="" direction="" isConst="false"
                             type="Queue*" />
               </method>
               <method name="switch_threads" qualifier="" returnType="void">
                  <parameter name="prev" defaultvalue="" direction="" isConst="false"
                             type="Thread*" />
                  <parameter name="next" defaultvalue="" direction="" isConst="false"
                             type="Thread*" />
               </method>
               <...>
   </member>
</family>
<...>
```

4. Conclusões

Para reduzir o tempo de desenvolvimento e de manutenção do software embarcado é necessário que os componentes sejam reusáveis para diferentes aplicações, o que pode

ser conseguido com a prévia engenharia de domínio dessas aplicações para a geração de componentes mais adequados. Nesse sentido, o objetivo principal deste trabalho foi desenvolver uma ferramenta para a geração semi-automática de um repositório de componentes a partir de um modelo de domínio.

A versão atual desta ferramenta inclui uma infra-estrutura que permite facilmente sua evolução, devido ao uso de padrões de projeto que separam interface de possível implementações. Ela já possui geradores internos de metadados, código e testadores de componentes, é simples de ser usada e exige poucas diretrizes para que o modelo de domínio possa ser interpretado. Por fim, efetivamente pode auxiliar o desenvolvedor de aplicações gerando de forma semi-automática um repositório de componentes para qualquer aplicação que será desenvolvida. Entretanto, nem todos os metadados podem ser extraídos do modelo de domínio, simplesmente porque o modelo de domínio não possui certos tipos de informações, como o tamanho de memória ocupado por um componente de software. De qualquer modo, esse tipo de metadado é necessário apenas no caso de haver uma outra ferramenta para geração automática do sistema, seleção dos componentes ou exploração do espaço de projeto. Se não for esse o caso, esses metadados não são necessários e pode-se passar diretamente às etapas de implementação dos artefatos reusáveis e das aplicações em si.

Desenvolvimentos futuros incluem a representação completa da estrutura de componentes prevista inicialmente, o que contempla componentes de hardware sintetizável (IPs) e de hardware físico (componentes eletrônicos), o sincronismo automático entre a implementação do repositório e o modelo de domínio que o gerou e a manutenção de metadados usados para exploração do espaço de projeto.

Referências

Arango, G. and Prieto-Diaz, R. (1989). *Domain Analysis: Acquisition of Reusable Information for Software Construction*, chapter Domain analysis: Concepts and research directions. IEEE Computer Society Press.

Fröhlich, A. A. M. (2001). *Application-Oriented Operating Systems*. PhD thesis, Sankt Augustin: GMD - Forschungszentrum Informationstechnik. 200 pages.

Polpeta, F. V. and Fröhlich, A. A. (2004). Hardware mediators: a portability artifact for component-based systems. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280, Aizu, Japan. Springer.

Pressman, R. (2000). *Software Engineering: A Practitioner's Approach*. McGraw-Hill.

Subramanian, S. (1997). Software asset management and domain engineering. In *COMPSAC '97: Proceedings of the 21th IEEE Computer Software and Applications Conference*, pages 494–495.

YAU, S. S. and DONG, N. (2000). Integration in component-based software development using design patterns. In *(COMPSAC) 24ºAnnual International Computer Software and Applications Conference*.