Geração de Sistemas de Gestão de Conteúdo com Softwares Livres

Fernando Silva Parreiras
Universidade Federal de Minas Gerais, Escola de Ciência da Informação,
Belo Horizonte, Brasil, 31270-010
fparreiras@ufmg.br

Marcello Peixoto Bax
Universidade Federal de Minas Gerais, Escola de Ciência da Informação,
Belo Horizonte, Brasil, 31270-010
bax@ufmg.br

Abstract

This article handles the content management systems generation using open source softwares. It asks if it is possible, based on the main existing theoretical contributions and in the available frameworks, to implement a process that allows reflecting a knowledge domain, represented by models, in a content management system. It is intended to establish the steps necessary for implementation of this approach and to raise framework of content management system development. It was made case study using a software development process as scope. As result, a content management system was generated and a process made up of five activities: (1) platform independent modeling, (2) platform specific modelling, (3) source code generation, (4) source code maintenance and (5) installation. This process uses 50% of the time planed for of the system development, but it presents some restrictions. It concludes that (1) the UML is a good candidate to a knowledge representation language, (2) modelling using levels of abstraction is a crucial necessity for the evolution of the area of information systems, and (3) the used framework presents advantages in prototyping or application in reduced knowledge domains.

Keywords: Domain Modeling, Software Engineering, Knowledge representation, Content Management.

Resumo

Este artigo trata a geração de sistemas de gestão de conteúdos, usando softwares livres. Pergunta-se se é possível, com base nas principais contribuições teóricas existentes e nos arcabouços disponíveis, implementar um processo que permita refletir um domínio de conhecimento, representado por meio de modelos, em um sistema de gestão de conteúdos. Pretende-se estabelecer os passos necessários para implementação desta abordagem e levantar um arcabouço de desenvolvimento de sistemas de gestão de conteúdos. Foi realizado um estudo de caso tendo como escopo o domínio de conhecimento de um processo de desenvolvimento software. Como resultado, tem-se um processo de geração, composto de cinco atividades: (1) Modelagem Independente de Plataforma, (2) Modelagem Específica de Plataforma, (3) Geração do código-fonte, (4) Manutenção Evolutiva do código-fonte, e (5) Instalação. Este processo de geração diminui em mais de 50% o tempo estimado para desenvolvimento do sistema, mas apresenta algumas restrições. Conclui-se que: (1) a UML é uma boa candidata à linguagem de representação do conhecimento, (2) a modelagem em vários níveis é uma necessidade crucial para a evolução da área de sistemas de informação, e (3) o arcabouço utilizado apresenta vantagens na prototipagem ou aplicação em domínios de conhecimento reduzidos.

Palavras-chave: Modelagem de domínio, Engenharia de Software, Representação do conhecimento, Gestão de conteúdo.

1. INTRODUÇÃO

Nos últimos anos, com a rápida depreciação do valor investido no código-fonte de sistemas, pesquisas sobre modelagem em camadas emergem da engenharia de software ([1], [2]), tentando separar em distintos níveis de abstração a representação de um domínio de conhecimento, de forma que o especialista de domínio seja capaz de modelar em nível diferente do analista de sistemas..

Por outro lado, a explosão das comunidades de desenvolvimento de softwares livres, a criação de repositórios de projetos e adoção deste tipo de software pelas organizações e governos fomenta o desenvolvimento de um cenário onde o conhecimento sobre que ferramenta resolve um dado problema é bastante valioso.
Uma terceira questão, mais explorada, é a grande quantidade de conteúdo digital gerado nas organizações. Com a necessidade de compartilhar documentos de maneira rápida e fácil utilizando a web, surgem os sistemas de Gestão de Conteúdo (SGCs). Gestão de Conteúdo é uma abordagem que surge em função da explosão de conteúdo multimídia na web e em Intranets e visa a permitir a gerência de todas as etapas, desde a criação até a publicação de conteúdo [3].
Estas três questões incentivam o desenvolvimento de pesquisas que envolvam a geração de sistemas de gestão de conteúdo baseados de softwares livres. Este artigo tem como problema de pesquisa verificar até que ponto é possível, com base em contribuições teóricas e em arcabouços de ferramentas disponíveis em código aberto, implementar um processo que permita refletir um domínio de conhecimento modelado em um sistema de gestão de conteúdo (SGC).
Com o aumento do número de sistemas de informação utilizados nas organizações, a possibilidade de gerá-los automaticamente deve ser considerada e alguns fatores que contribuem para viabilizar esta geração são:

- A existência de uma linguagem de modelagem como a UML [4], padrão de fato, reconhecido e usado por grande parte da indústria de software.
- A noção de perfis UML que permite a criação de diversos níveis de abstração sucessivos durante a modelagem, separando o conhecimento em diversas camadas ([5], [1], e [2]).
- A existência de uma linguagem-padrão para representação de dados e metadados, a linguagem de marcação estendida XML [6], e o padrão de intercâmbio de metadados, XML Metadata Interchange ou XMI [7].
- Melhor conhecimento sobre padrões e arquiteturas de software de qualidade, como resultado de anos de experiências, reflexão e reengenharia das melhores práticas, em numerosos projetos ([8], [9], [10] e [2]).

Neste trabalho, a Seção 2 apresenta a arquitetura dirigida por modelos, descrevendo os tipos de modelos e suas principais características, a Seção 3 apresenta o processo de geração de sistemas de gestão de conteúdo proposto, sua estrutura, suas atividades e o arcabouço utilizado. A Seção 4 traz um estudo de caso levantando os aspectos positivos e as restrições do processo. Finalmente, conclui-se o artigo com a Seção 5.

2. ARQUITETURA DIRIGIDA POR MODELOS

Em 2001, o Object Management Group (OMG) definiu uma abordagem de modelagem chamada Arquitetura Dirigida por Modelos – Model Driven Architecture (MDA). A abordagem consiste na adoção de modelos em diferentes níveis de abstração, com características distintas, a fim de isolar os aspectos computacionais do domínio de conhecimento.
A questão-chave da arquitetura MDA é a transformação de modelos [2] (Figura 1). O modelo inicial é um modelo independente de plataforma – Platform Independent Model (PIM) –, que possui as seguintes características:

- Representa o domínio de conhecimento sem a distorção de aspectos tecnológicos;
- É completamente independente da pilha de plataforma;
- É um modelo detalhado e, geralmente, representado em UML;
- É a base para o modelo específico de plataforma – Platform Specific Model (PSM).

O modelo PIM é transformado em modelo específico de plataforma – Platform Specific Model (PSM), por intermédio de mapeamentos. Um modelo PSM:

- Contém informações do domínio e da plataforma;
- É criado a partir do mapeamento de um PIM para uma plataforma específica;
- É um modelo detalhado e sempre representado em UML;
- É a base para o código-fonte.

O PSM é, então, convertido em código-fonte e outros artefatos como documentos, roteiros, etc.

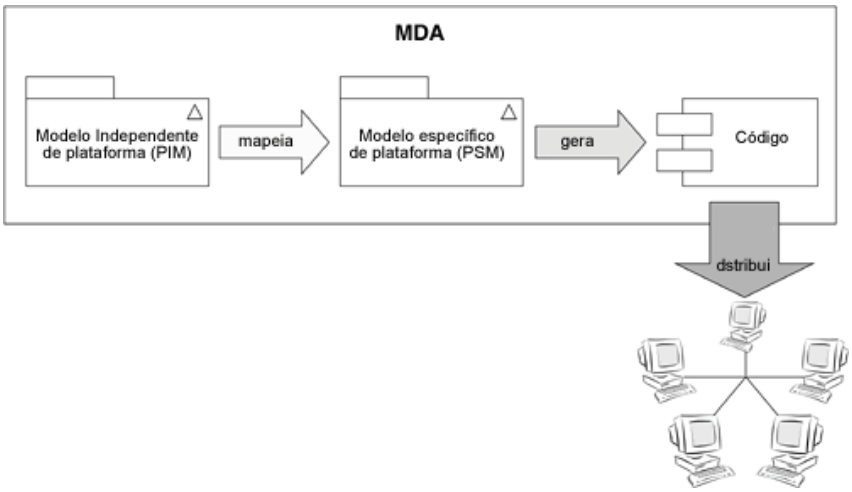

Figura 1. Transformação de modelos [5]

O valor investido em código-fonte tende a depreciar-se rapidamente, juntamente com a linguagem de programação e plataforma ao qual ele pertence. A vantagem desta abordagem é que o domínio de conhecimento fica independente do código-fonte. Na arquitetura MDA, o valor é investido em modelos abstratos. Assim, o valor do modelo aumenta de forma diretamente proporcional ao nível de abstração. Ainda nesta abordagem, o especialista de domínio trabalha em nível de abstração diferente do analista ou programador.
A arquitetura MDA pode trazer contribuições importantes, a saber: permite a separação do conhecimento em níveis de abstração, isolando diferentes aspectos deste; habilita o especialista de domínio a lidar com ferramentas e conceitos mais próximos de sua realidade; permite ao analista ou programador se libertar da responsabilidade do conhecimento de domínio; e abre espaço para o surgimento de ferramentas de geração automática de sistemas, uma vez que, para tanto, basta concentrar esforços no mapeamento entre os diversos níveis de abstração.

2.1 Trabalhos relacionados

Identificaram-se na literatura algumas iniciativas relacionadas à geração de sistemas de informação, apoiadas na arquitetura dirigida por modelos. Sem exaurir o tema, apresentam-se aqui algumas propostas.
Utilizando uma implementação de MDA, [11] mapeiam um MOF (Meta Object Facility) para a linguagem Java. Um MOF [12] define uma linguagem abstrata e uma estrutura para especificação, construção e gerenciamento de metamodelos independentes de tecnologia de implementação. Entretanto, esta implementação não conta com as funcionalidades pré-construídas de um ambiente de Gestão de Conteúdo e demanda grande quantidade de esforço na implementação de serviços como personalização, segurança, etc.
Outro projeto que se baseia na arquitetura MDA é o projeto "XIS" [13]. "XIS" (Desenvolvimento de Sistemas de Informação baseado em Modelos e Arquiteturas de Software) é um projeto de Investigação e desenvolvimento realizado no contexto do Grupo de Sistemas de Informação do LAVC/INESC-ID[1]. O principal objetivo deste projeto é o estudo, desenvolvimento e avaliação de mecanismos de produção de sistemas de informação de forma mais eficiente. O projeto XIS é influenciado pelo modelo de referência MDA e baseia-se fortemente em um conjunto de práticas emergentes, seguindo uma abordagem (1) baseada em modelos, (2) centrada em arquiteturas de software, e (3) baseada em técnicas de geração automática. Contudo, este projeto também não usufrui as características já presentes nos ambientes de Gestão de Conteúdo, e nem da facilidade de incorporação de novas funcionalidades.
[14] descrevem o Modelo de Objeto Adaptável (Adaptative Object-Model) tratado também como arquitetura reflexiva ou meta-arquitetura. Um Modelo de Objeto Adaptável (MOA) é um sistema que representa classes, atributos e relacionamentos como metadados. Ele é um modelo baseado em instâncias, ao invés de classes. Usuários mudam o metadado (modelo de objeto) para refletir alterações no domínio. Estas alterações modificam o comportamento do sistema. MOA é uma arquitetura que pode, dinamicamente, adaptar-se, em tempo de execução, aos novos requisitos do usuário. Outras nomenclaturas para esta arquitetura são "modelo de objeto ativo" [15] e "modelo de objeto dinâmico" [16]. Esta abordagem concentra-se no aspecto evolutivo do domínio de conhecimento. Em contrapartida, sistemas de informação que utilizam o MOA são mais difíceis de construir e de serem compreendidas pelos atores envolvidos, além de não especificar como se dá a interface com o usuário.

[1] http://berlin.inesc-id.pt

Uma outra implementação em camadas é realizada pelo projeto Hércules [17]. O projeto Hércules consiste em um arcabouço para desenvolvimento de sistemas de informação, visando possibilitar a geração automática de código a partir de diagramas UML, e estabelece um conjunto de descrições de alto nível de abstração, que os analistas ou programadores utilizam para especificar a apresentação, o desenvolvimento operacional e a persistência do sistema a ser construído. Este modelo é dividido em três camadas: domínio controle e apresentação. Cada uma destas camadas possui diagramas UML associados. Embora esta abordagem envolva a geração automática de sistemas, ela não se preocupa em tratar separadamente o conhecimento de domínio do conhecimento tecnológico..

3. PROCESSO DE GERAÇÃO DE SISTEMAS DE GESTÃO DE CONTEÚDO

O Processo de geração proposto é composto de um arcabouço tecnológico, baseado em ferramentas de código aberto, e um conjunto de atividades, que produzem e consomem resultados. O arcabouço e as atividades são detalhados a seguir.

3.1 Arcabouço

O arcabouço utilizado é baseado em softwares de código aberto. A partir de uma busca no sítio web sourceforge[2], identificou-se sistemas que apoiassem às atividades necessárias à realização do processo. A partir de descritores como "Content Management System" e "UML", foram encontradas ferramentas que atendem a fins de Gestão de Conteúdo e edição em UML. Foram escolhidas ferramentas que possuíam a maior comunidade de desenvolvedores e se encontravam no estágio de maturidade estável (5 - stable).

O arcabouço utilizado no experimento é formado pelo conjunto de ferramentas listadas abaixo e descritas na Tabela 1.

- Editor UML: Gentleware Poseidon UML (GENTLEWARE);
- Servidor de aplicações, de banco de dados e web: Zope [18];
- Plataforma de Gestão de Conteúdo (SGC): Plone [19];
- Componentes pré-construídos: Plone Archetypes [20];
- Gerador de código-Fonte: ArchGenXML [21].

Tabela 1. Arcabouço Utilizado

Nome da ferramenta:	Tipo de licença	Versão	Descrição da função	No. de envolvidos
Poseidon UML CE	Livre para uso não comercial.	3.0.1	Editor UML. Utilizado para modelagem OO na notação UML.	21
Plone	GPL	2.0.4	Sistema de Gestão de Conteúdos.	88
Archetypes	GPL	1.3.1.	Ferramenta de geração automática de tipos de conteúdo plone, orientada a esquemas.	65
ArchGenXML	GPL	1.1	Utilitário de linha de comando que gera aplicações Plone baseado no framework Archetypes, a partir de um modelo UML representado em XMI ou XMLSchema.	65
strip-o-gram	GPL	1.4	Converte a documentação HTML disponível no arquivo xmi para texto puro.	6
oi18ndude	GPL	0.2.2	Permite a geração de rótulos traduzíveis.	81
Zope	GPL	2.7	Servidor de aplicações, banco de dados e web.	-

A interação entre estas ferramentas e os atores envolvidos pode ser ilustrada na Figura 2. O especialista de domínio gera o Modelo Independente de Plataforma (PIM), que é mapeado em um Modelo Específico de Plataforma (PSM) com a ajuda de um analista ou programador. O PSM é o insumo do gerador de código-fonte (ArchGenXML), produzindo o código-fonte (CF). Este, juntamente com componentes pré-construídos são os insumos da plataforma de gestão de conteúdo que produz o sistema de gestão de conteúdo (SGC) para um domínio de conhecimento específico.

[2] http://www.sourceforge.net

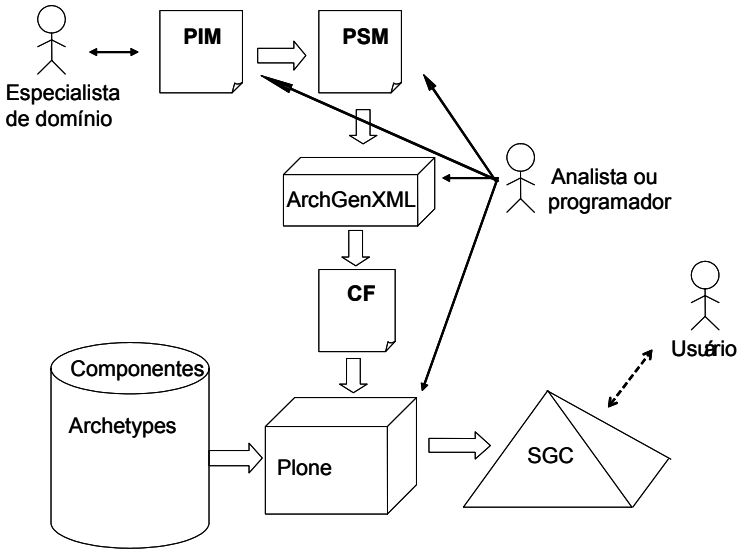

Figura 2. Arcabouço utilizado

3.2 Estrutura do Processo

O processo utiliza um arcabouço composto de ferramentas, que possuem como requisito outras ferramentas. O processo é composto de atividades, que assumem compromissos. Estas atividades produzem resultados e consomem resultados produzidos nas atividades anteriores (Figura 3).

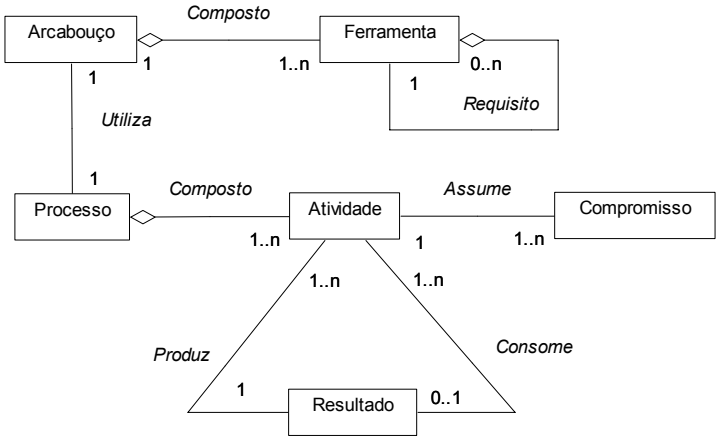

Figura 3. Estrutura do processo

3.3 Atividades do processo

As cinco atividades do processo são: Modelagem Independente de Plataforma; Modelagem Específica de Plataforma; Geração do código-fonte; Manutenção Evolutiva do código-fonte; e Instalação (Tabela 2). O fluxo das atividades é ilustrado na Figura 4, onde é são apresentados os insumos e produtos de cada atividade, assim como o ator relacionado.

Tabela 2. Atividades e resultados

Resultado	Sigla	Responsável	Ferramenta aplicável	Atividade
Modelo Independente de Plataforma	PIM	Especialista de Domínio	Editor UML	Modelagem Independente de plataforma
Modelo Específico de Plataforma	PSM	Analista ou Programador	Editor UML	Modelagem Específica de Plataforma

Código-fonte	CF	Analista ou Programador	Gerador de código	Geração do código-fonte
Código-fonte Evoluído	CF	Analista ou Programador	Editor de Código	Manutenção Evolutiva do código-fonte
Sistema de Gestão de Conteúdo	SGC	Analista ou Programador	Plataforma de gestão de conteúdo	Instalação

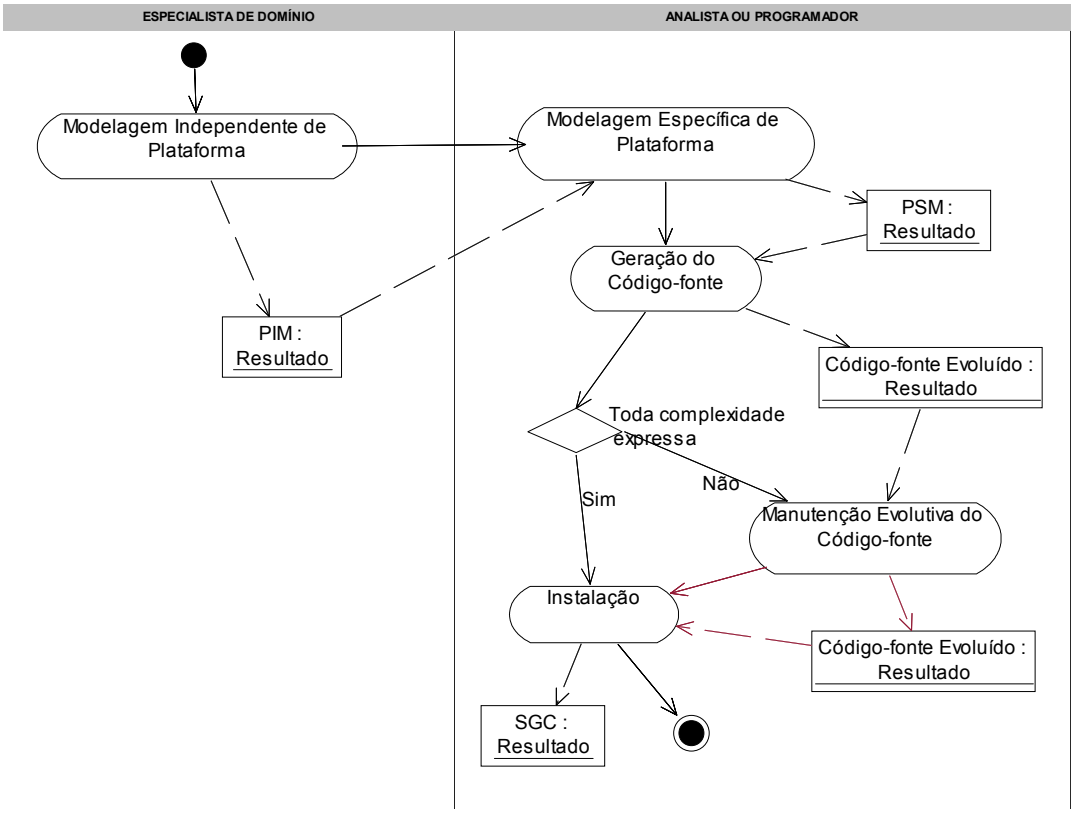

Figura 4. Atividades do Processo

Outra forma de sistematizar as transformações ocorridas neste experimento é por intermédio da equação abaixo:

SGC = Plone (MAN (CF (PSM (PIM, Perfil_UML), ArchGenXML)))

Em que:

- SGC: Sistema de Gestão de Conteúdo.
- Plone: Plataforma de gestão de conteúdo que utiliza o PraxisCMF;
- MAN = Manutenção Evolutiva do código-fonte do SGC;
- CF = Transformação do PSM, a partir da ferramenta ArchGenXML, em Código-fonte;
- PSM = Transformação do PIM e Perfil_UML em Modelo Específico de Plataforma;
- PIM: Modelo independente de plataforma;
- Perfil_UML: conjunto de mecanismos de extensão;

Abaixo são detalhadas as atividades do processo.

3.3.1 Modelagem independente de plataforma

Esta atividade concentra esforços na modelagem e representação de um determinado domínio de conhecimento. Ao modelar um domínio de conhecimento, Noy e McGuinness (2001) recomendam, inicialmente, a identificação de modelos já existentes. Utilizou-se o editor UML Poseidon para a criação do diagrama de classes. Esse editor utiliza o formato XMI [7] para armazenamento do modelo. O formato XMI é utilizado na geração automática também em [13] e [17]. A arquitetura MDA [2] também recorre ao formato XMI. O resultado desta atividade é o Modelo Independente de Plataforma (PIM), representado em UML e armazenado no formato XMI.

3.3.2 Modelagem específica de plataforma

Os aspectos dependentes de plataforma são adicionados nesta atividade, que cria um Modelo Específico de Plataforma (PSM) com menor nível de abstração, mais próximo do código-fonte. A partir de um perfil UML, é possível acrescentar aspectos inerentes a um determinado arcabouço. Utilizou-se o perfil UML ArchGenXML [21], que permite a representação de ícones, rótulos de dados, máscara de entrada, etc. A adequação tecnológica ocorre nesta atividade, em que o analista ou programador modela de acordo com a tecnologia escolhida.

Assim, é necessário manter um sincronismo entre o PIM e o PSM. Propõe-se um mapeamento entre os modelos, por meio de um roteiro de transformação. Este roteiro é escrito na linguagem XML Extensible Stylesheet Language Transformation (XSLT) [22], manualmente, pelo analista ou programador, e contem todas as transformações realizadas. Um mapeamento deste tipo também é utilizado por Silva [13], e outros mapeamentos semelhantes são realizados em ontologias, conforme [23].

O resultado desta atividade é o PSM, representado em UML, estendido segundo um perfil UML e armazenado em XMI.

3.3.3 Geração do código-fonte

Nesta atividade, o PSM é transformado em código-fonte, por intermédio de um gerador de código-fonte em linguagem Python, que importa componentes pré-construídos disponíveis no arcabouço. Este código-fonte compõe o SGC. O PSM serve de entrada para o gerador de código-fonte, responsável por transformar o PSM em CF. Esta atividade resulta no código-fonte (CF) do SGC.

3.3.4 Manutenção Evolutiva do código-fonte

Se toda a complexidade do SGC puder ser expressa no PSM, esta atividade é opcional. Contudo, dado o atual desenvolvimento das ferramentas do arcabouço utilizado, esta ocorrência é rara. Assim, o analista ou programador deve fazer alterações no CF, a fim de atender os requisitos do sistema de informação que não puderam ser expressos no PIM ou PSM. O CF é, então, copiado e evoluído, tornando-se o resultado desta atividade, o Código-fonte Evoluído (CFE).

Esta alteração traz a mesma necessidade de sincronismo que a atividade de Modelagem Específica de Plataforma. Entretanto, como o CF é estruturado, mas não em uma linguagem de marcas, o mapeamento por meio de um roteiro de transformação fica mais difícil. Embora mais difícil, tal mapeamento é possível, utilizando técnicas de engenharia reversa, que consiste na leitura do CF do SGC para geração de modelos mais abstratos. No processo, se ocorre uma alteração em uma das atividades anteriores, e existe a necessidade de manutenção do código-fonte, esta alteração deve ser tratada manualmente nesta atividade. Esta atividade resulta no CFE.

3.3.5 Instalação

Nesta atividade, o código-fonte manipulado na atividade anterior é agrupado, formando o SGC. O roteiro de instalação envolve a interrupção do serviço web, cópia do código-fonte e outros arquivos, inicialização do serviço web e instalação do SGC na plataforma de gestão de conteúdo.

4. ESTUDO DE CASO

4.1 Escopo

Foi realizado um estudo de caso com o domínio de conhecimento do processo de desenvolvimento de software que, de acordo com o aumento do grau de maturidade, sofre alterações freqüentes. A motivação de uso deste domínio para o estudo de caso foi principalmente seu dinamismo. Das 30 empresas, hoje, no Brasil, que possuem algum nível de certificação CMM [24], verifica-se que 24 delas estão no nível 2 e cinco no nível 3. Há uma empresa no nível 4 e nenhuma no nível 5. [25] constata que as empresas nacionais buscam a melhoria de seu processo de software, alterando-o e adequando-o às melhores práticas internacionais. Com efeito, para controlar os indicadores necessários para os níveis 2 e 3 do CMM é indispensável o uso de sistemas de apoio. Elegeu-se um processo de desenvolvimento de software baseado no Processo Unificado [26] chamado Praxis [27].

Apenas uma parte do domínio foi utilizada. Foram considerados apenas os diagramas de classe, que contemplam itens estruturais do domínio. Esta restrição se deu devido a uma das ferramentas do arcabouço utilizado, o gerador de

código-fonte, que manipula somente diagramas de classe. Foram identificadas 45 classes e 49 relacionamentos em todos os diagramas de classe do Praxis. O experimento utilizou um modelo com 19 classes, 26 relacionamentos e 38 atributos.
A medição do tamanho do SGC deste domínio resulta em 247 pontos de função não ajustados. Ajustando este valor baseado no ambiente tecnológico obtém-se 207,48 pontos de função ajustados (PFA). Aplicando um coeficiente de produtividade razoavelmente baixo neste valor, ou seja, 1 (um), tem-se uma estimativa de 207,48 horas a serem gastas no desenvolvimento do SGC.

4.2 Resultados

Neste estudo de caso foram gastas, aproximadamente, 66 horas para o desenvolvimento do SGC, conforme descrito na Tabela 3. Sendo constantes todas as outras variáveis, observa-se uma diferença significante entre os valores obtidos pelo autor e os valores estimados conforme a métrica de análise por pontos de função, salvo as possíveis falhas na metodologia utilizada. Esta diferença pode ser atribuída principalmente ao gerador de código-fonte e aos componentes pré-construídos. O Gerador de código-fonte permite gerar automaticamente inferfaces de inclusão, alteração e exclusão dos objetos, enquanto que os componentes pré-construídos trazem funcionalidades como busca, navegação, padrões de apresentação e outros.

Tabela 3. Tempo gasto pelo autor para implementação

Atividade	Tempo (em horas)
Modelagem independente de plataforma	24
Modelagem específica de plataforma	24
Geração do código-fonte	1
Manutenção Evolutiva do código-fonte	16
Instalação	1
TOTAL	66

Os valores utilizados para o cálculo do esforço de desenvolvimento são estimados, devido à ausência de séries históricas de projetos desenvolvidos utilizando a mesma plataforma tecnológica, o que apresenta uma fragilidade na comparação entre o esforço estimado e o esforço realizado.
Além da utilização do processo para a geração do sistema de gesta de conteúdo, realize um papel importante no desenvolvimento de software: a prototipagem. A prototipagem consiste na elaboração de um protótipo, ou seja, versão parcial e preliminar do sistema destinada à validação com o usuário ou teste. Após a atividade de modelagem, comum a qualquer processo de desenvolvimento de sistemas de informação, uma solução interessante seria utilizar esta implementação para gerar uma versão preliminar do sistema para apreciação do usuário, o que reduziria ainda mais o tempo de geração, pois o tempo gasto na manutenção do código-fonte seria menor.

4.2.1 Principais Restrições

As ferramentas de código aberto disponíveis ainda não suportam todo o poder da modelagem orientada a objeto. A impossibilidade de usufruir todos os recursos da linguagem UML restringiu o potencial de representação utilizado.
Um exemplo destas restrições é a utilização de apenas um dentre os nove diagramas da linguagem UML pela plataforma adotada no estudo de caso, o diagrama de classes. Diagramas com o poder de expressão comportamental como o diagrama de atividades, diagrama de estado e diagrama de seqüência ainda não são considerados pelas ferramentas disponíveis, o que restringe bastante o escopo de aplicação. Por exemplo, a consideração de "fluxos de trabalho" modelados ainda não pode ser integrada ao processo de geração.
A usabilidade do SGC gerado apresenta algumas restrições como, por exemplo, no que diz respeito às referências cruzadas. Ao acessar um determinado objeto, tem-se uma visualização dos outros que se relacionam com ele. Entretanto, não se pode visualizar a descrição do relacionamento, o que se torna um agravante quando um objeto se relaciona com outro de duas formas diferentes.
O mapeamento entre as camadas também é um ponto que apresenta restrições. Enquanto o mapeamento entre PIM e PSM deve ser escrito manualmente pelo analista ou programador, o mapeamento entre o PSM e os CFs fica mais complicado, pois embora os CFs sejam estruturados, ainda não há uma forma automática de realizar este mapeamento.

5. CONCLUSÃO

Este artigo apresentou um processo de geração de Sistemas de Gestão de Conteúdo a partir do uso de softwares livres. Apresentou, ainda, um estudo de caso descrevendo os aspectos positivos e restrições na aplicação do processo. Conclui-se que:

- A modelagem em vários níveis é uma necessidade crucial para a evolução da área de sistemas de informação, principalmente quando considerados domínios em que o número de conceitos é muito grande e suas propriedades e

relações muito dinâmicas, como no caso de aplicações no domínio da medicina ou mesmo da engenharia de software. Nesses domínios essa separação parece fundamental para a perenidade dos sistemas de informação;

- O arcabouço utilizado apresenta vantagens, aplicado na prototipagem ou geração de SGCs;
- Quando um sistema utiliza o paradigma da orientação a objeto, a UML é uma boa candidata à linguagem de representação do conhecimento.
- O uso de um gerador de código-fonte e de objetos pré-construídos para a geração de um SGC reduz consideravelmente o tempo de desenvolvimento do mesmo.
- A modelagem em camadas, utilizando diagramas e perfis UML, apresenta uma forma interessante de adicionar aspectos da implementação no modelo de forma isolada, sem comprometer a visão do usuário. Contribui, ainda, para a documentação do projeto, que nem sempre é concebida dentro do processo de software, ou é considerada uma atividade sacrificante por parte dos responsáveis.

Propõe-se que este trabalho continue evoluindo nas seguintes direções:

- Continuação do desenvolvimento do arcabouço utilizado, já que as ferramentas que o constituem não param de evoluir em seus princípios teóricos;
- Aplicação do arcabouço utilizado e do processo de geração utilizado no desenvolvimento de sistemas de gestão de conteúdos em outros domínios, buscando a compreensão do problema a partir da sua consideração sob outras perspectivas;

Referências

[1] Beale, T. Archetypes: Constraint-based domain models for future-proof information systems. In: OOPSLA, 17, 2002, Seatlle. Anais eletrônicos... Workshop on behavioural semantics, Seatlle: ACM. 2002.

[2] Frankel, D. S. Model Driven Architecture: Applying MDA to Enterprise Computing. Indianapolis: Wiley Publishing, Inc., 2003. 352p.

[3] Pereira, J.C.L., Bax, M. P. Introdução à Gestão de Conteúdos. In: KM BRASIL, 2002, São Paulo. Anais (CD-ROM)... São Paulo: [s.n.], 2002.

[4] Booch, G., Rumbaugh J., Jacobson I. UML: guia do usuário. Rio de Janeiro: Campus, 2000. 472 p.

[5] Arlow, J., Neustadt, I. Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML. Boston: Addison Wesley, 2003. 528p.

[6] Yergeau, F. et al. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, February 2004.

[7] HEATON, L. XML Metadata Interchange (XMI) Specification. Versão 1.2. Needham: OMG, jan. 2002b. 268 p. Technical report.

[8] Gamma, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison Wesley, 1995. 395p.

[9] Buschmann, F. et al. Pattern-Oriented Software Architecture: A system of patterns. 1. ed. Mississauga: John Wiley & Sons, 1996. 476 p.

[10] Hofmeister, C., Nord, R., Soni, D. Applied Software Architecture. Boston: Addison Wesley, 1999. 397p.

[11] Santos, H. L., Barros, R. S. M. Utilizando o MOF na construção de metamodelos em um ambiente MDA. In: Simpósio Brasileiro De Engenharia De Software, 18, 2004, Brasília. Anais... Brasília: Sociedade Brasileira de Computação (SBC), 2004. CD-ROM.

[12] Heaton, L. Meta Object Facility (MOF) Specification. Versão 1.4. Needham: OMG, apr. 2002a. 358 p. Technical report. Disponível em <http://www.omg.org/docs/formal/02-04-03.pdf>. Acesso em 25 jan. 2005.

[13] Silva, A. R. Abordagem XIS ao Desenvolvimento de Sistemas de Informação.In: Conferência da Associação Portuguesa de Sistemas de Informação, 4, 2003, Porto. Anais... Porto: Universidade Portucalense. 2003a.

[14] Yoder, J. W., Balaguer, F., Johnson, R. Architecture and Design of Adaptive Object Models. In: Intriguing

Technology Paper OOPSLA, 2001, Tampa Bay. Anais eletrônicos... Tampa Bay: ACM SIGPLAN Notices, ACM Press, Dez. 2001.

[15] Foote, B, Yoder, J W. Metadata and Active Object-Models. In: Collected papers from the PLoP '98 and EuroPLoP '98 Conference, 1998, Washington. Proceedings... Washington: Washington University. 1998.

[16] Riehle, D., Tilman, M., Johnson, R. Dynamic Object Model. In: PLoP2000, 7, 2000, Monticello. Proceedings... Monticello: Technical Report #wucs- 00-29, Dept. of Computer Science, Washington University, 2000. p. 1-13.

[17] Pais, A. P. V., Oliveira, C. E. T., Leite, P. H. P. M. Robustness Diagram: A Bridge Between Business Modeling And System Design . In: Proceedings of VII International Conference on Object-Oriented Information Systems - OOIS'01. Calgary, Canadá: Springer-Verlag. 2001, v. 1, p. 530-539.

[18] Latteier, A., Pelletier, M. The Zope Book. Berkeley: Pearson Education, 2001. 384p.

[19] Mckay, A. The Definitive Guide to Plone. Berkeley: Apress, 2004. 584p.

[20] Silva, S. Archetypes: An Introdution. In: ZopeMag.com. Kurfürstendamm: beehive KG. 2003b. Disponível em: <http://www.zopemag.com/Issue006/Section_Articles/article_IntroToArchteypes.html>.

[21] Klein, J. W. ArchGenXML Manual - generating Archetypes using UML. 2004. Disponível em: <http://plone.org/documentation/archetypes/archgenxml-manual>

[22] Kay, M. XSLT referência do programador. 2. ed. Rio de Janeiro: Alta Books, 2002. 667p.

[23] Handschuh, S., Staab, S., Volz, R. On deep annotation. In: International World Wide Web Conference (WWW), 12, 2003, Budapest. Proceedings… Budapest: ACM Press, 2003.

[24] Paulk, M. C. et al. Capability Maturity Model for Software, Version 1.1. Pittsburgh: Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, Feb. 1993. 82 p. Technical report.

[25] Weber, K. et al. Modelo de Referência para Melhoria de Processo de Software: uma abordagem brasileira. In: XXX Conferência Latino-Americana De Informática (CLEI2004), 30, 2004, Arequipa. Anais...Arequipa: [s.n], 2004.

[26] Jacobson, I., Rumbaugh, J., Booch, G. Unified Software Development Process. Reading - MA: Addison-Wesley, 1999.

[27] Paula Filho, Wilson de Pádua. Engenharia de software: fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: LTC, 2003. 602p.