Engenharia de Domínio no Suporte ao Aumento de Flexibilidade nos Sistemas de Software

Alexandre Bragança e Ricardo J. Machado

Resumo — A flexibilidade é uma das principais qualidades que são requeridas actualmente às aplicações. As técnicas de implementação de variabilidade fornecem meios para atingir essa flexibilidade. A indústria de software adopta correntemente diversas técnicas de implementação de variabilidade. Estas técnicas fornecem diversos graus de variabilidade. As técnicas que fornecem níveis mais elevados de variabilidade também implicam processos mais complexos de engenharia. Este artigo aborda esta problemática. É proposto um método simples para medir o grau de variabilidade. Mostra-se também como a adopção de métodos de engenharia do domínio, aplicados em paralelo com o método de engenharia da aplicação, pode fornecer um suporte adequado para gerir a complexidade resultante da adopção de algumas técnicas de implementação de variabilidade.

Palavras-chave — Engenharia de Domínio, Extensibilidade, Flexibilidade, Variabilidade.

1 Introdução

A flexibilidade de comportamento é actualmente uma característica necessária em sistemas de software complexos. Esta flexibilidade pode ser conseguida introduzindo pontos de variabilidade no software. Estes pontos de variabilidade podem ser introduzidos em diversas fases do ciclo de desenvolvimento do software e podem adoptar diversas técnicas para implementar a variabilidade. Com o objectivo de atingir níveis mais elevados de flexibilidade, a indústria de software tem vindo a adoptar técnicas de implementação de variabilidade suportadas em tempo de execução das aplicações. Os padrões da indústria para componentes de software como o COM, XPCOM, Java2, CORBA e o .Net suportam este tipo de implementação de variabilidade. Estes permitem que diversas implementações possam ser utilizadas para uma determinada interface funcional. Assim, diversos componentes de software podem ser desenvolvidos para uma determinada interface funcional. Estes diversos componentes podem ser instalados e dinamicamente utilizados nos pontos de variabilidade da aplicação.

As plataformas de software como o Java2 e o .Net começam também a suportar a geração em tempo de execução de componentes de software e a sua instalação. Exemplos dessas novas capacidades das plataformas de software são os mecanismos de introspecção e a capacidade de compilação de código em tempo de execução. Algumas aplicações e sistemas, como o JBoss, aplicam já estas técnicas [1].

A adopção de técnicas de implementação de variabilidade, particularmente técnicas suportadas em tempo de execução, promete níveis de flexibilidade e evolução mais elevados para as aplicações. Apesar de tais promessas, para se atingirem as vantagens inerentes às técnicas de implementação de variabilidade, a sua simples adopção não é suficiente.

De facto, a adopção de técnicas de implementação de variabilidade também eleva o nível da complexidade das aplicações e, desta forma, pode aumentar a dificuldade do desenvolvimento e da manutenção. Neste artigo, na secção 2 e 3, apresenta-se uma visão geral sobre as técnicas de implementação de variabilidade. Com base nesta análise geral, na secção 4 discute-se como o grau de variabilidade é influenciado pela adopção das técnicas mais comuns de implementação de variabilidade nas diversas fases do processo de desenvolvimento. Apresentam-se também razões para o facto da adopção de técnicas de implementação de variabilidade (principalmente aquelas que são suportadas em tempo de execução) no processo de desenvolvimento de software poder falhar. Na secção 5 apresenta-se uma abordagem centrada na adopção de métodos de desenvolvimento de software, baseados em engenharia de domínio, como forma de abordar o desenvolvimento de aplicações complexas com elevado grau de variabilidade. A secção 6 conclui o artigo.

2 Técnicas de Implementação de Variabilidade

Um dos conceitos mais importantes quando se aborda a variabilidade no software, em particular no processo de desenvolvimento, corresponde ao momento em que se selecciona uma das variantes do ponto de variabilidade, ou seja, o momento em que o ponto de variabilidade é fechado (momento usualmente designado por *binding time*). Esta acção significa, normalmente, que a partir desse momento o software executará a opção seleccionada para o ponto de variabilidade, ou seja, esse 'local' do software deixa de ser um ponto de variabilidade. É possível seleccionar uma opção de um ponto de variabilidade em diversas fases do

- *Alexandre Bragança é investigador na I2S Informática – Sistemas e Serviços, S.A. e docente do Instituto Superior de Engenharia do Porto. E-mail: alexandre.braganca@i2s.pt; alex@dei.isep.ipp.pt.*
- *Ricardo J. Machado é Prof. Auxiliar do Departamento de Sistemas de Informação da Universidade do Minho. E-mail: rmac@dsi.uminho.pt.*

processo de desenvolvimento: derivação da arquitectura do produto (aplicação), compilação, edição de ligações (*linking*) e execução. Este artigo baseia-se nas últimas três fases porque estas dizem respeito a técnicas que podem ser utilizadas nas linguagens de programação e ambientes de desenvolvimento mais comuns. Não obstante, é possível conceber outros momentos de *binding* como, por exemplo, o tempo de depuração.

Relativamente às técnicas de implementação de variabilidade, é muito útil fazer a distinção entre técnicas em que o *binding time* se realiza antes do momento da distribuição das técnicas nas quais o *binding time* é após a distribuição do software. Esta distinção é muito importante porque define uma linha que separe o ciclo de vida tradicional de desenvolvimento do software onde é possível mudar quase tudo da instalação e execução do software no cliente e, no qual a mudança e a variabilidade são quase impossíveis de introduzir. As técnicas de implementação de variabilidade com tempos de *binding* antes da distribuição, usadas vulgarmente pela indústria, têm por base [2]:
- Pré-processamento de código de fonte (por exemplo através de directivas do pré-processador de C);
- Directivas do editor de ligações (por exemplo ligando objectos e/ou bibliotecas diversas);
- Parâmetros (por exemplo *templates* do C++);
- Herança;
- Composição de código.

De acordo com a nossa experiência, as técnicas de composição de código não são usadas extensivamente na indústria. A maioria destas técnicas são resultado da investigação recente desenvolvida nos meios académicos e, como tal, na grande maioria dos casos, estas técnicas ainda não são adoptadas pela indústria. Exemplos destas técnicas são a programação orientada pelos aspectos [3], a programação orientada pelos sujeitos [4], a programação generativa [5] e o GenVoca [6].

As técnicas de composição de código são baseadas na suposição que o modelo de programação orientado pelos objectos tem limitações relativamente à composição de classes, tal como o suporte para assuntos transversais (*crosscutting concerns*) [7]. No GenVoca, por exemplo, é possível compor o código de diversas classes como uma unidade ou camada. A combinação de diversas camadas, como forma de atingir um determinado comportamento, tem algumas similaridades com o mecanismo da herança do modelo orientado pelos objectos. No modelo de programação orientado pelos objectos, a herança é utilizada para adicionar comportamento novo a uma classe através da especialização das suas características originais. As camadas do GenVoca funcionam de uma forma similar à herança mas com uma abrangência maior. Estas aproximações à composição de componentes são geralmente baseadas em técnicas de transformação de programas [8].

O desenvolvimento de software com o GenVoca é baseado na composição dos componentes através de camadas de tipos de dados. Estas camadas de tipos de dados representam características dos componentes. Desta forma, as composições das camadas podem ser descritas por expressões. Segue-se um exemplo de uma expressão de composição de um componente em GenVoca [6]:

```
Jak = Sm ( Templ ( Java ) )
```

Esta expressão está a definir um dialecto de Java que amplia a linguagem Java com suporte para máquinas de estados e *templates*. Cada característica é representada pelos tipos de dados de uma camada. Cada camada contribui para a composição final do componente com classes, atributos e métodos que implementam a característica correspondente. Empilhando as diversas camadas podem-se combinar as características necessárias ao componente.

A principal limitação destes mecanismos de composição de código semelhantes ao GenVoca, no que respeita a implementação de pontos de variabilidade, é que eles dependem da selecção e composição dos componentes em fases do processo de desenvolvimento que precedem a distribuição do software. A secção 4 descreve como o grau de variabilidade do software pode ser mais elevado quando os pontos de variabilidade são fechados em fases posteriores à distribuição do software.

3 Técnicas de Suporte à Variabilidade em Tempo de Execução

Uma técnica muito frequente de implementação de variabilidade em tempo de execução, que é utilizada por quase todas as aplicações, é a execução condicional de código baseada no valor de uma variável ou expressão. Neste caso, todos as opções ou variantes do ponto de variabilidade estão incluídas na aplicação e não é possível aumentar o conjunto de variantes do ponto de variabilidade após a construção (compilação) da aplicação. Outras técnicas, como o uso de apontadores para funções ou o uso do padrão de desenho *template method* [9], são muito similares em efeito, pois todas as variantes possíveis são definidas em tempo de compilação. Uma forma de alargar as variantes de um ponto de variabilidade, no contexto das técnicas referenciadas, consiste em substituir o módulo com o ponto de variabilidade por um módulo novo e compatível que contenha as novas variantes. Esta técnica é usada normalmente quando uma empresa necessita de corrigir problemas com um módulo de uma aplicação. Para tal, substitui o binário do módulo por uma versão nova que corrija o problema. No caso de variabilidade suportada em tempo de execução, o objectivo não é corrigir um problema, como um *bug*, mas aumentar um ponto de variabilidade com novas variantes. Existem outras técnicas de variabilidade em tempo de execução que são mais flexíveis. Nestas, é possível que o ponto de variabilidade esteja aberto a novas variantes depois da produção do software. As principais técnicas que suportam variabilidade pós-distribuição, adoptadas pela indústria e com pontos de variabilidade abertos a novas variantes em tempo de execução, são:
- Carregamento dinâmico de bibliotecas;
- Infra-estruturas de componentes;
- Linguagens de *scripting*;
- Introspecção.

Uma forma de alargar um conjunto de variantes em tempo de execução consiste em utilizar a técnica de carregamento dinâmico de bibliotecas. Esta técnica é bastante usada pela indústria, sendo um exemplo muito conhecido de aplicação desta técnica a arquitectura dos controladores

(*drivers*) de ODBC. A capacidade de dinamicamente carregar uma biblioteca e usar os seus pontos de entrada possibilita distribuir apenas as bibliotecas que executam as variantes que são necessárias para um determinado ponto de variabilidade. Com esta técnica, é também possível adicionar novas variantes após a distribuição e instalação inicial da aplicação.

A técnica de carregamento dinâmico de bibliotecas é uma das bases das infra-estruturas de componentes de software como o COM ou o XPCOM. Estas infra-estruturas baseiam-se no conceito de componente de software suportado em bibliotecas de código que são carregadas dinamicamente em memória e também na noção de interfaces como forma de aceder aos serviços prestados pelos componentes de software. Uma vez que as aplicações acedem aos componentes através das interfaces, é possível substituir o componente que a aplicação está a usar se o novo componente suportar a interface que a aplicação necessita. Para além desta funcionalidade básica, as infra-estruturas de componentes podem oferecer outras funcionalidades, como a invocação de serviços remotos ou o *pooling* de componentes. As infra-estruturas de componentes são uma técnica de implementação de variabilidade em tempo de execução que é bastante usada pela indústria, como por exemplo no projecto Mozzila e no pacote de aplicações de escritório da Microsoft. Estes dois exemplos usam também uma outra técnica de implementação de variabilidade: linguagens de *scripting*.

As línguagens de *scripting* permitem um grau muito elevado de variabilidade e de flexibilidade visto que a funcionalidade de uma aplicação pode ser totalmente alterada em tempo de execução. Não obstante, estas são usadas, na maior parte dos casos, para expandir e adaptar aplicações e não explicitamente como mecanismo de suporte à variabilidade. A grande vantagem das linguagens de *scripting* é que elas não necessitam de processos de construção muito pesados (envolvendo compilação e edição de ligações), usuais em linguagens de programação comuns. No entanto, o seu uso é limitado, nomeadamente, devido a requerem algum treino por parte do utilizador final. Este é um factor muito importante em técnicas que suportam variabilidade em tempo de execução, já que a sua utilização em fases posteriores à distribuição do software pode, em determinados casos, implicar a participação do utilizador final. É o que acontece normalmente no caso das linguagens de *scripting*.

A introspecção é também uma técnica que pode ser usada para suportar variabilidade em tempo de execução. As plataformas de software mais utilizadas na indústria, como o Java2 ou o .Net, suportam esta técnica. Como estas plataformas lideram o mercado relativamente a ambientes de desenvolvimento de software, é muito provável que a introspecção se torne também uma técnica utilizada no suporte à implementação de variabilidade em tempo de execução. No mínimo, esta técnica permite o carregamento dinâmico de tipos, a instanciação dinâmica de objectos e também a execução dinâmica de métodos. Algumas plataformas fornecem ainda funcionalidades mais avançadas como, por exemplo, a geração dinâmica de código. Estas características de introspecção são muito poderosas pois podem trazer quase todas as possibilidades existentes nos ambientes de desenvolvimento durante as fases anteriores à distribuição do software para fases posteriores.

4 A Variabilidade no Processo de Desenvolvimento de Software

Todas as técnicas de implementação de variabilidade que foram apresentadas podem ser usadas para aumentar o grau de variabilidade das aplicações. Existem dois conceitos nucleares relativamente à variabilidade e ao ciclo de vida do software: o momento da introdução da variabilidade (*introduction time*) e o momento da selecção da variante (*binding time*). O momento da introdução diz respeito à fase do processo de desenvolvimento de software na qual é introduzido o ponto de variabilidade. Por exemplo, na fase de análise de requisitos é possível introduzir um ponto de variabilidade respeitante a uma característica da base de dados da aplicação que especifica que motor de base de dados deve ser suportado: orientado para o objecto ou relacional. No momento da introdução, o ponto de variabilidade é aberto para as diversas variantes possíveis. Numa fase posterior do processo de desenvolvimento de software, o ponto de variabilidade é fechado através da selecção de uma das possíveis variantes. Este é o momento da selecção da variante ou *binding time*. No exemplo apresentado, nalguma fase posterior do processo de desenvolvimento de software, uma das possibilidades de motor de base de dados será seleccionada.

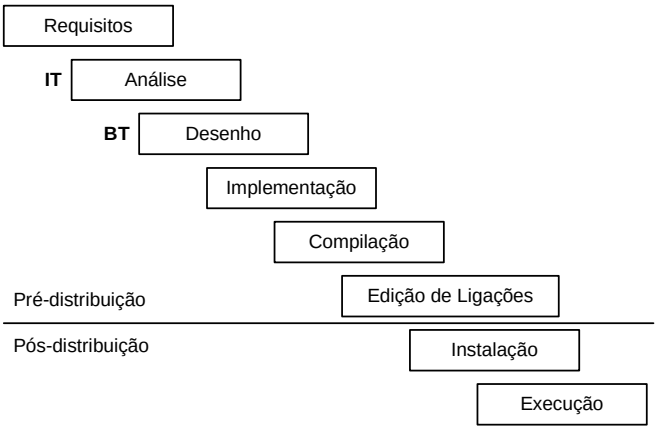

Fig. 1. Exemplo de *Introduction time* (IT) e *binding time* (BT) para um ponto de variabilidade no ciclo de vida do software.

Alguns autores desenvolvem estes conceitos simples de variabilidade introduzindo o conceito de *binding site* que difere do *binding time* porque abrange a dimensão espaço para além da dimensão tempo [5]. Basicamente, o *binding site* representa *quando* e *onde* o ponto de variabilidade é fechado através da selecção de uma das variantes. Por questões de simplicidade, adopta-se o conceito de *binding time* durante o artigo. Um ponto de variabilidade pode também permanecer aberto após o *binding time*. Isto significa que se podem continuar a seleccionar novas variantes no ponto de variabilidade mesmo após se ter seleccionado uma variante, ou seja, o ponto de variabilidade permanece aberto.

Uma outra característica importante a respeito de um ponto de variabilidade é o número de variantes que este

suporta. No exemplo anterior, relativo ao motor de base de dados, o ponto de variabilidade apenas suporta duas variantes.

A fig. 1 apresenta um cenário possível para o *introduction time* e *binding time* do exemplo do motor de base de dados relativamente às fases mais comuns do ciclo de vida do software. Neste caso, o *introduction time* (IT) e o *binding time* (BT) correspondem a duas fases consecutivas do ciclo de vida do software. Isto significa que o ponto de variabilidade permanece aberto apenas entre as fases de análise e de desenho. Depois da fase de desenho, o ponto de variabilidade é fechado pois é seleccionada uma das variantes relativas ao motor de base de dados da aplicação. Assim, a partir dessa fase, pode-se afirmar que o grau de variabilidade da aplicação, relativamente ao suporte de motores de base de dados, é nulo. Isto não significa que globalmente a aplicação não tenha variabilidade. De facto, a aplicação apresenta características de variabilidade, no entanto, neste exemplo, o momento de introdução e o *binding time* acontecem em fases inicias do ciclo de vida do software. Se o ponto de variabilidade permanecer aberto até fases mais posteriores, por exemplo até à fase da compilação, então o grau de variabilidade da aplicação aumenta. Isto acontece pois quando se atrasa a introdução e o *binding time* de um ponto de variabilidade está-se a aumentar o seu grau de variabilidade e, por consequência, o grau de variabilidade da aplicação. Ou seja, quanto mais tarde se aplicar a variabilidade maior será o grau de variabilidade de uma aplicação. A adopção de pontos de variabilidade em fases finais do ciclo de vida do software pode possibilitar uma mais rápida mudança de variantes pois não é necessário regressar às fases iniciais de desenvolvimento. No exemplo da fig. 1, se for necessário mudar de motor de base de dados, essa operação é executada na fase de desenho. Assim, o caso da fig. 1 apresenta um grau de variabilidade menor do que um caso similar que tenha o *binding time* numa fase posterior como, por exemplo, a fase de compilação.

Com base nos princípios apresentados, é possível quantificar o grau de variabilidade de um ponto de variabilidade. Assim, os conceitos mais significativos a ter em conta são o momento da introdução, o *binding time* e o número de variantes. Para um determinado ponto de variabilidade, quanto mais tarde ele for introduzido ou seleccionada uma das variantes, maior será o seu grau de variabilidade. A variabilidade também aumenta com o aumento do número de variantes associado ao ponto de variabilidade. Assim, associando valores crescentes às fases do ciclo de vida do software (por exemplo, 1-Requisitos; 2-Análise; 3-Desenho, etc.) consegue-se uma medida muito simples para o grau de variabilidade através da seguinte expressão:

```
IT * BT * nV
```

Nela, `IT` representa o momento da introdução, `BT` representa o *binding time* e `nV` representa o número das variantes possíveis. Com base nesta expressão, o grau de variabilidade para o caso da fig. 1 é 2*3*2=12. Alterando o *binding time* para a fase de compilação obtém-se um grau de variabilidade de 2*5*2=20. Outros autores desenvolvem mais profundamente esta aproximação para quantificar o grau de variabilidade e propõem uma representação normalizada para a variabilidade [10]. A expressão simplificada que foi apresentada para quantificar a variabilidade é adoptada pois está mais de acordo com os objectivos do artigo.

Torna-se claro que um grau elevado de variabilidade pode ser atingido quando se adopta a variabilidade em fases finais do ciclo de vida do software, particularmente nas fases que se seguem à distribuição. Esta é uma das razões pelas quais as técnicas que suportam variabilidade em tempo de execução, tais como as que foram apresentadas na secção anterior, são cada vez mais adoptadas na indústria.

Como o grau de variabilidade quantifica a capacidade para uma aplicação suportar alterações, este constitui também uma medida da flexibilidade da aplicação. No exemplo do motor de base de dados, a aplicação torna-se mais flexível ao conter um ponto de variabilidade que permite seleccionar entre dois motores de base de dados possíveis. Ou seja, mudar o motor de base de dados da aplicação requer menos esforço do que num caso em que esse ponto de variabilidade não exista. Assim, a introdução de pontos de variabilidade no ciclo de vida do software aumenta a flexibilidade das aplicações. Um tópico bastante interessante é como identificar possíveis pontos de variabilidade. Este artigo não aborda esse tópico. Contudo, a variabilidade é um dos tópicos nucleares das linhas de produtos e das famílias de produtos, e muitos autores têm proposto métodos e técnicas relativamente à identificação e representação da variabilidade [11], [12], [13]. Quando o objectivo é desenvolver uma única aplicação ou sistema, a adopção de técnicas de variabilidade não visa suportar a possibilidade de construção de múltiplas aplicações a partir de uma base comum, mas acrescentar flexibilidade e capacidade de evolução a uma única aplicação, isto é, facilitar a possibilidade de alterações na aplicação.

De forma geral, quando se considera que a implementação de uma característica da aplicação pode vir a ser alterada no futuro, então deve-se adoptar uma técnica de implementação de variabilidade. Estas decisões devem ser efectuadas com especial cuidado. Se, por exemplo, a implementação de uma característica desse género já implicar algum tipo de variabilidade, então poder-se-á justificar adoptar uma técnica de implementação de variabilidade que suporte mais facilmente a possível futura adição de novas variantes. Desta forma, é possível aumentar a flexibilidade de uma aplicação com pouco esforço de desenvolvimento uma vez que o mecanismo de variabilidade já era necessário. No caso do exemplo do motor de base de dados, em vez de adoptar uma técnica de suporte à variabilidade numa fase antes da distribuição, como o uso de directivas de compilação para suportar os dois motores da base de dados, era possível adoptar uma técnica que permitisse maior flexibilidade e capacidade de evolução. Duas possibilidades seriam a adopção de um padrão de desenho do tipo *fábrica* [9], com *binding time* das variantes em tempo de execução, ou ainda, uma técnica de implementação de variabilidade baseada em bibliotecas de carregamento dinâmico, que suportam a introdução de novas variantes em tempo de execução da aplicação.

Como mencionado, a identificação de pontos de variabi-

lidade não está no âmbito deste artigo. No entanto, se um ponto de variabilidade for identificado, isto significa que pelo menos existem duas variantes para esse ponto de variabilidade. Neste contexto, e se o objectivo for desenvolver uma aplicação flexível, deve-se dar muita atenção no que diz respeito à técnica adoptada para suportar essa variabilidade. Foi apresentada uma expressão simples que permite ter uma ideia do grau de variabilidade que se pode obter quando se adopta uma técnica de implementação de variabilidade. Usando esta aproximação simples, um engenheiro de software pode mais facilmente prever as implicações em termos de flexibilidade das suas opções de desenvolvimento. Com base na nossa experiência, podemos afirmar que, com a eventual excepção das técnicas de introspecção e geração dinâmica de código, o esforço e a complexidade acrescidas, resultantes da adopção de técnicas de implementação de variabilidade nas fases finais do ciclo de vida do software, não têm implicações muito significativas no custo de projectos médios e grandes [14].

Como se verificou nesta secção, a adopção de técnicas de implementação de variabilidade fornece um suporte para aumentar a flexibilidade das aplicações. Com as técnicas referidas, torna-se mais fácil a introdução de variantes novas nas aplicações. Isto verifica-se pois técnicas de suporte de variabilidade em tempo de execução, como bibliotecas de carregamento dinâmico, tornam possível a introdução de novas variantes sem ser necessária a alteração das aplicações. Um exemplo bastante conhecido desta aproximação é a arquitectura do ODBC que suporta a mudança do motor da base de dados de uma aplicação em tempo de execução.

Se o suporte de variabilidade for adoptado numa aplicação com o objectivo de aumentar a sua flexibilidade, é possível que mudanças nos requisitos possam ser suportadas pela adição de novas variantes. No entanto, esta possibilidade apenas é exequível se não for necessário alterar a estrutura, ou interface, do ponto de variabilidade. Ou seja, é possível que o ODBC suporte um novo motor de base de dados se o controlador respectivo for compatível com a interface do ODBC. Se tal não for possível, então é necessário alterar a interface do ODBC para acomodar o novo controlador, e isso pode ter várias implicações, como a actualização dos controladores antigos ou o suporte de duas versões. Neste caso, será necessário regressar às fases mais iniciais do ciclo de vida do software, possivelmente à fase de introdução do ponto de variabilidade, de forma a alterar a estrutura do ponto de variabilidade. Quando isto acontece, o processo do desenvolvimento do software pode-se tornar mais complexo e, normalmente, outros problemas de desenvolvimento de software aparecem. Uma vez que este tipo de problema pode ocorrer com alguma frequência, propõe-se, na secção seguinte, uma aproximação baseada na engenharia do domínio que tem por objectivo controlar o nível de complexidade que as técnicas de implementação de variabilidade podem impor ao processo de desenvolvimento de software.

5 Engenharia de Domínio como Forma de Implementação e Gestão de Variabilidade

As linhas de produtos e as famílias de produtos visam promover a reutilização entre aplicações [15]. Para conseguir esta reutilização entre as diversas aplicações de uma linha de produtos, as mesmas devem partilhar características. Isso geralmente só é conseguido se as aplicações partilham algum domínio. Assim, é possível conseguir algum nível de reutilização entre um processador de texto e uma aplicação de apresentação pois estas partilham o domínio das aplicações de escritório.

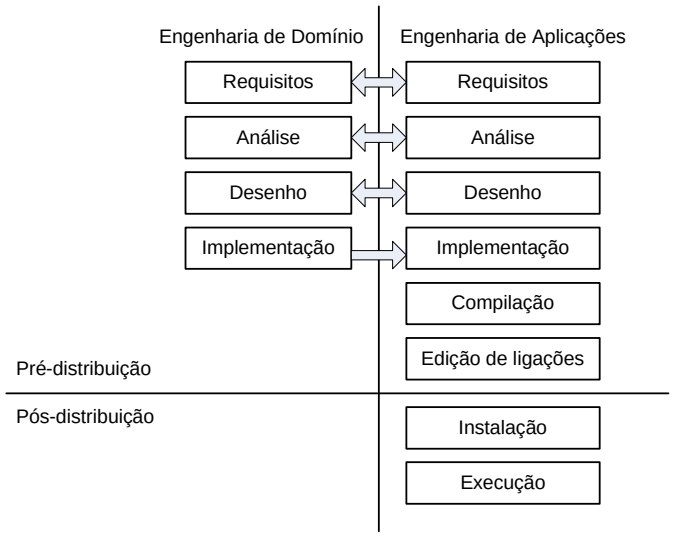

Fig. 2. Engenharia de Domínio vs. Engenharia de Aplicação.

A engenharia de domínio é um processo que visa identificar, representar e implementar artefactos reutilizáveis de um domínio [16]. Exemplos de métodos que aplicam princípios de engenharia de domínio são o FODA [11] e o RSEB [12]. Tradicionalmente, estes métodos são aplicados apenas quando diversas aplicações partilham o mesmo domínio, como o caso das linhas de produtos. Nestes casos, a engenharia de domínio fornece métodos e técnicas adequados para suportar o desenvolvimento de artefactos reutilizáveis, como arquitecturas de software, modelos de desenho e componentes de software. A fig. 2 apresenta as interacções possíveis entre a engenharia de domínio e a engenharia de aplicação. Os artefactos produzidos pela engenharia de domínio podem ser reutilizados na engenharia de aplicação. As setas com sentido da engenharia de aplicação para a engenharia de domínio demonstram que a engenharia de aplicação pode fornecer entradas para a engenharia de domínio, usualmente sob a forma de conhecimento do domínio em causa. Cada nova aplicação que é construída dentro do domínio ganhará da reutilização dos artefactos do domínio mas também fornecerá conhecimento para refinar os mesmos artefactos do domínio ou para construir novos artefactos. Os artefactos resultantes do processo de engenharia de domínio consistem nos componentes comuns que são reutilizados nas aplicações mas também nos componentes que implementam os pontos de variabilidade e que são usados apenas em algumas aplicações. Estes

são também os tipos de artefactos que são necessários quando o objectivo é construir aplicações flexíveis e que possam evoluir mais facilmente. Assim, em vez de adoptar a engenharia de domínio como um processo que visa suportar o desenvolvimento de diversas aplicações de um domínio, este pode ser adoptado para apoiar o desenvolvimento de variantes e respectivos pontos de variabilidade na engenharia de aplicações comuns. O que se propõe é que os métodos de engenharia de domínio sejam adoptados para suportar a implementação de pontos de variabilidade na engenharia de aplicações. De facto, ao considerar-se cada ponto de variabilidade como um domínio, então faz sentido que os métodos de engenharia de domínio possam ser aplicados a cada ponto de variabilidade. Uma vez que os métodos de engenharia de domínio requerem esforço extra no processo de engenharia, deve-se ter um cuidado especial na selecção dos pontos de variabilidade que devem ser desenvolvidos através da aplicação de princípios de engenharia de domínio.

Como regra geral, somente os pontos de variabilidade que possibilitem um elevado grau de variabilidade devem ser desenvolvidos utilizando métodos e técnicas de engenharia de domínio. Este é, normalmente, o caso de pontos de variabilidade que necessitem de suporte em tempo de execução. O exemplo do motor de base de dados, introduzido na secção precedente, é um caso típico em que poderia ser adoptado um método de engenharia de domínio. Se o grau de variabilidade for elevado, por exemplo porque devem ser suportados dinamicamente diferentes motores de base de dados, então deve-se adoptar métodos de engenharia de domínio no ponto de variabilidade, mesmo se o suporte de diferentes motores de base de dados seja apenas utilizado para uma única aplicação. Neste caso, o suporte para o ponto de variabilidade deve ser promovido a um componente a reutilizar na aplicação. Desta forma, um método de engenharia de domínio poderia ser utilizado para desenvolver o componente que suporta o ponto de variabilidade, assim como as respectivas variantes.

Esta aproximação à engenharia de pontos de variabilidade, que combina engenharia de domínio e engenharia de aplicações, tem muitas vantagens face à engenharia simples de aplicações. Na nossa opinião, uma das principais vantagens é a de tornar mais simples a gestão de mudanças estruturais nos pontos de variabilidade. Como o suporte para o ponto de variabilidade consiste num componente resultante da engenharia de domínio, uma alteração estrutural num componente deste tipo consiste, de facto, numa alteração no domínio do ponto de variabilidade. Devido às suas características, este tipo de alteração é gerida no contexto do processo de engenharia de domínio adoptado e não implica maior complexidade no processo de engenharia da aplicação.

6 CONCLUSÕES

Existem exemplos documentados de adopção bem sucedida de métodos de engenharia de domínio para o desenvolvimento de artefactos de domínio. As linguagens específicas de domínio são um de muitos exemplos possíveis [17]. Noutros contextos encontram-se outros exemplos, como o desenvolvimento de linhas de produtos ou de familias de produtos [15].

Neste artigo foi proposta uma nova perspectiva relativamente à engenharia de domínio. Esta assenta no princípio de que a engenharia de domínio pode ser adoptada como um método bem sucedido para suportar a introdução e o desenvolvimento de pontos de variabilidade em aplicações isoladas. Os pontos de variabilidade e as técnicas que permitem a sua implementação podem suportar o aumento do grau de flexibilidade e a capacidade de evolução das aplicações. Para atingir este objectivo, diversas técnicas de implementação de variabilidade estão disponíveis. Estas técnicas, quando adoptadas, possibilitam diferentes graus de variabilidade. Este artigo apresenta uma forma muito simples para medir o grau de variabilidade que as diversas técnicas de implementação de variabilidade podem introduzir numa aplicação. Esta medida simples pode apoiar o engenheiro de software na selecção da técnica de implementação de variabilidade mais adequada ao seu caso particular. As técnicas que possibilitam um grau mais elevado de variabilidade também são geralmente aquelas que são mais complexas de implementar e manter. Para gerir esta complexidade propõe-se que a adopção de métodos de engenharia de domínio, com utilização paralela ao método de engenharia da aplicação, seja apenas efectuada no desenvolvimento dos pontos de variabilidade mais complexos.

Esta aproximação é baseada na nossa própria experiência com a adopção de técnicas de implementação de variabilidade na engenharia de aplicações como forma de aumentar a flexibilidade das aplicações [14]. Para um determinado ponto de variabilidade podem ser possíveis diversas técnicas de suporte à variabilidade. Como descrito neste artigo, é relativamente simples medir o grau de variabilidade de diferentes técnicas. O que se torna mais difícil é a identificação prévia de possíveis pontos de variabilidade e a selecção da técnica mais adequada para cada ponto de variabilidade. No futuro, pretendemos desenvolver o nosso trabalho no contexto destas preocupações.

REFERÊNCIAS

[1] M. Fleury and F. Reverbel, "The JBoss Extensible Server," *Proc. of Middleware 2003 - ACM/IFIP/USENIX International Middleware Conference*, 2003.

[2] J.v. Gurp, "On the Design & Preservation of Software Systems," PhD dissertation, Computer Science Department, University of Groningen, Groningen, 2003.

[3] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, "Aspect-Oriented Programming," *Proc.European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[4] Harold Ossher, William Harrison, Frank Budinsky, Ian Simmonds, "Subject-Oriented Programming: Supporting Decentralized Development of Objects," *Proc. 7th IBM Conference on Object-Oriented Technology*, 1994.

[5] K. Czarnecki, "Generative Programming Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models," PhD dissertation, Department of Computer Science and

Automation, Technical University of Ilmenau, 1998.

[6] Don Batory, Roberto E. Lopez-Herrejon, Jean-Philippe Marti, “Generating Product-Lines of Product-Families,“ *Proc. ASE'02*, Edinburgh, Scotland, 2002.

[7] Cristina Lopes, “D: A Language Framework For Distributed Programming,” PhD dissertation, College of Computer Science, Northeastern University, 1997.

[8] J. M. Neighbors, “The Draco approach to constructing software from reusable components,” *IEEE Transactions on Software Engineering*, vol. 10, pp. 64-74, 1984.

[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[10] Michel Jaring, Jan Bosch, “Representing Variability in Software Product Lines: A case study,“ *Proc.Second Software Product Line Conference (SPLC2)*, 2002.

[11] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, “Domain Analysis (FODA) Feasibility Study Technical Report”, Software Engineering Institute, Carnegie Mellon University, 1990.

[12] Ivar Jacobson, Martin Griss, Patrik Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*, Addison Wesley Longman, 1997.

[13] Martin Griss, John Favaro, Massimo d'Alessandro, “Integrating Feature Modeling with the RSEB,” *Proc. Fifth International Conference on Software Reuse*, Victoria, Canada, 1998.

[14] Alexandre Bragança, Ricardo J. Machado, “Run-time Variability Issues in Software Product Lines,” *ICSR8 Workshop on Implementation of Software Product Lines and Reusable Components*, Madrid, 2004.

[15] Jan Bosch, *Design and Use of Software Architectures Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[16] SEI, “Domain Engineering: A Model-Based Approach,” *Software Engineering Institute*, http://www.sei.cmu.edu/domain-engineering/, 2004.

[17] Arie v. Deursen, Paul Klint, Joost Visser, “Domain-Specific Languages: An Annotated Bibliography,” *ACM SIGPLAN Notices*, vol. 35, pp. 26-36, 2000.