Resolvedores SAT para Verificação de Consistência em Modelos de Características

Thieres Nardy Dias[1], Eduardo Figueiredo[1], Rodrigo Geraldo Ribeiro[2]

[1]Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
31.270-010 – Belo Horizonte – MG – Brasil

[2]Instituto de Ciências Exatas e Aplicadas (ICEA)
Universidade Federal de Ouro Preto (UFOP)
35.931-008 – João Monlevade – MG – Brasil

thieresnard@yahoo.com.br,

figueiredo@dcc.ufmg.br, rodrigogribeiro@decea.ufop.br

Resumo. *Linhas de produtos de software é uma nova abordagem de desenvolvimento de sistemas voltada ao reuso de software. As principais razões para se criar uma linha de produtos são a redução dos custos do processo, software mais confiável e de qualidade. Os modelos de características são uma forma diagramática de se modelar uma linha de produtos de software. Como os modelos estão crescendo em termos de suas características, um forte apoio ferramental se faz necessário para suportar o processo de modelagem, como por exemplo, a verificação de consistência dos mesmos. Assim, este trabalho tem como objetivo propor um algoritmo resolvedor SAT especializado para depurar modelos de características. O algoritmo proposto possui pontos de extensão para o desenvolvimento de novos algoritmos e para integração com ferramentas de modelagem de características. É esperado contribuir, desta forma, com a comunidade de SPL incluindo desenvolvedores de ferramentas de modelagem e usuários destas ferramentas.*

1. Introdução

Linhas de produtos de software (LPS) é uma abordagem emergente no campo da Engenharia de Software, com o intuito de permitir a reutilização em larga escala [12]. O desenvolvimento de software baseado em LPS tem como objetivo explorar as semelhanças e diferenças dentro de uma família de sistemas, a fim de proporcionar uma infraestrutura comum para derivar produtos de software em tempo hábil, com padrões de alta qualidade e com os custos do processo reduzidos [12]. No que tange o desenvolvimento de software para LPS, a modelagem da variabilidade se apresenta como uma técnica central para colocar este conceito em prática [2]. Ou seja, a modelagem de variabilidade e/ou modelos de variabilidade estão presentes em quase todas as fases e atividades do ciclo de vida de uma LPS como: análise do domínio da aplicação, escopo, seleção do produto, derivação do produto, tempo de execução e evolução [11].

Os benefícios de uma LPS são alcançados por meio de uma arquitetura de software destinada a promover a reutilização das características obrigatórias e opcionais em produtos diversos [12]. As características opcionais definem os pontos de variabilidade e

o seu papel é permitir a instanciação de diferentes produtos, ao habilitar ou desabilitar características específicas na LPS [10]. Desta forma, os modelos de características desempenham um papel vital em LPS, fornecendo uma visão diagramática para representar as semelhanças e os pontos de variabilidade dentro de uma família de sistemas. Assim, eles permitem membros individuais da família possam ser configurados e instanciados de forma segura [10].

É notável no campo de LPS, que os modelos de características estão aumentando cada vez mais em tamanho e complexidade, necessitando assim de um forte apoio ferramental automatizado para suporte à configuração de produtos e verificação de consistência de seus modelos [4]. Atualmente, diversas ferramentas automatizadas de configuração de produtos estão disponíveis [10] e o raciocínio orientado por características podem tirar proveito dos amadurecidos sistemas baseados na lógica de primeira ordem. Exemplos destes sistemas são resolvedores SAT [11], *Binary Decision Diagrams* (BDD) [10] e *Constraint Satisfaction Problems* (CSP) [4], dentre outros [10].

Muitos estudos e pesquisas exploram a ligação entre os modelos de características com a lógica proposicional. Estes estudos fazem intenso uso dos motores de inferência baseados na lógica, como BDD, resolvedores SAT e CSP. O objetivo é raciocinar em cima dos modelos de características e obter conclusões acerca da consistência destes.

Pelo fato de SAT ser bem conhecido na literatura como um problema $\mathcal{NP}$-Completo, sua utilização na verificação dos modelos de características tem atraído bastante atenção dos pesquisadores. Contudo, ela se mostrou como uma opção viável para verificação de modelos, mesmo os modelos em larga escala realistas não representam dificuldades significativas para resolvedores SAT [11]. Assim, os resolvedores SAT podem ser utilizados para suportar a execução de tarefas relacionadas à depuração em modelos de características, tais como a verificação da consistência e a detecção de características comuns e mortas [10]. A utilização de SAT para verificação formal na modelagem de características é considerada uma alternativa promissora, pois devido ao grande esforço dos pesquisadores em criar métodos sofisticados para SAT, os resolvedores se tornam cada vez mais eficientes com o passar do tempo [1].

Este trabalho contribui para a disseminação do uso de SAT para verificação de consistência em modelos de características, já que o emprego desta técnica se mostrou eficiente em detectar inconsistências de modelos. Além disso, o trabalho tem como objetivo fornecer uma infraestrutura (teórica e ferramental) adequada para vindouras ferramentas de modelagem de características, contribuindo de forma direta aos modeladores de software, programadores e/ou fornecedores das ferramentas de edição, enriquecimento do campo de SPL, pois este emerge como uma nova abordagem de desenvolvimento de software.

Este trabalho consiste na proposta de um resolvedor SAT projetado especificamente para ser utilizado por ferramentas de modelagem de SPL. Na seção seguinte, são apresentados os referenciais teóricos do trabalho, fornecendo o embasamento necessário para contextualizar o leitor ao problema aqui tratado. Na Seção 3 é apresentado o mecanismo utilizado para reduzir modelos de características em fórmulas proposicionais, por fim chegando na tradução para SAT (na CNF). Na sequência, é discutido o algoritmo proposto para permitir a construção de resolvedores

SAT especializados ao problema da modelagem de características. Já na Seção 4 temos os trabalhos relacionados. E por fim, na Seção 5 as conclusões deste artigo são apresentadas e possibilidades em trabalhos futuros são discutidas.

2. Modelos de características

A abordagem de desenvolvimento em LPS tem sido cada vez mais utilizada como forma de reusar componentes de software entre aplicações semelhantes. Nesta abordagem, diversas aplicações passam a ser implementadas sobre uma mesma plataforma e diferem sobre determinados aspectos chamados características. Modelos de características descrevem como estas características podem ser combinadas a fim de implementar um produto desta linha. Assim, um modelo de características é uma representação gráfica hierárquica em forma de árvore [8]. As relações entre uma característica pai com suas características filhas (ou subcaracterísticas) são categorizadas da seguinte maneira: mandatória, opcional, ou não exclusivo e alternativa [2].

Detalhadamente, cada característica *alternativa* faz parte de um grupo de características alternativas sob um determinado pai, significando exatamente que, apenas uma destas subcaracterísticas tem de ser selecionada sempre que o pai é selecionado. Já as características sob a relação *ou não exclusivas*, sempre que seu pai for instanciado, uma ou mais subcaracterísticas do grupo podem ser selecionadas. Uma característica *opcional* pode ou não ser incluída na configuração quando o seu pai é selecionado. Uma característica *mandatória* tem de ser selecionada sempre que o seu pai é selecionado. Para efeito de ilustração, ver Figura 1 mostrando um diagrama de características do sistema chamado MobileMedia [6].

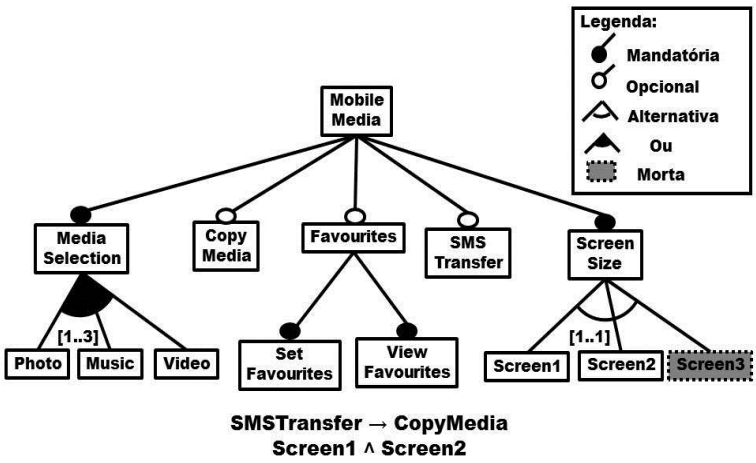

Figura 1. Modelo de características MobileMedia.

Na árvore da Figura 1, os nós representam as características e as arestas descrevem as relações entre elas. Um único nó raiz na Figura 1, nomeado MobileMedia, representa o conceito do domínio que está sendo modelado e ele está incluído em cada produto descrito pelo modelo. As características *opcionais* são desenhadas com um círculo vazio, por exemplo CopyMedia. Já as características *mandatórias*, por exemplo MediaSelection, são decoradas com os círculos preenchidos.

Em grupos de características alternativas é possível especificar relações de cardinalidade sob a forma de $[n, m]$, em que no mínimo n e no máximo n características

de um grupo, podem ser instanciadas, tal que $1 \leq n \leq m$ e m nunca é maior do que o número total de características do grupo. Por exemplo, a linha de produtos dispõe de Photo, Music e Video que formam um grupo *ou não exclusivo* com cardinalidade $[1, 3]$, em que no mínimo uma das opções deve estar presente em qualquer produto que contenha MediaSelection. Por outro lado, as características Screen1, Screen2 e Screen3 formam um grupo de *alternativas* com cardinalidade $[1..1]$, para que apenas uma opção de tamanho de tela possa ser incluída no produto final. Restrições adicionais são frequentemente adicionadas diretamente na forma proposicional em um modelo de características para complementar as relações da árvore de características [11]. Tais restrições adicionais são nomeadas como restrições entre os ramos da árvore.

3. Utilização dos resolvedores SAT em modelagem de características

Está seção tem por finalidade discutir mecanismos de redução dos modelos de características em fórmulas na CNF (padrão de codificação das fórmulas de entrada para resolvedores SAT) e apresentar o algoritmo proposto para resolver instâncias do problema da verificação dos modelos. Na Seção 3.1 é detalhado o método de tradução dos modelos em um formato conveniente de fórmulas para que um resolvedor SAT possa recebê-las como entrada. Na Seção 3.2 é mostrado o algoritmo resolvedor SAT proposto.

3.1. Redução dos modelos de características para SAT

A verificação de consistência de modelos, foco deste trabalho, tem por objetivo encontrar características mortas ou comuns, ver Figura 1. Uma característica é identificada como morta quando esta não pertencer a qualquer produto válido [11]. Já as características comuns, são compartilhadas por todos os produtos da LPS e sua presença nos modelos são às vezes indesejáveis [10]. Por exemplo, conforme Figura 1 a característica Screen3 é marcada como característica morta decorrente da restrição extra $Screen1 \wedge Screen2$, o que impede a inclusão de Screen3 em qualquer produto válido.

Análises automáticas são também utilizadas para suportar a derivação de produtos, também conhecido como configuração do produto. Dois tipos de configurações são [10]: (i) configuração interativa, um usuário faz decisões de configuração para derivar um produto, enquanto que as ferramentas de edição vão guiando o modelador em fazer escolhas consistentes dos produtos, a fim de certificar o usuário que sempre faça configurações legais; (ii) configuração offline, o sistema de edição dos modelos completam uma verificação parcial a cada etapa (de forma induzida pelo modelador).

Um passo importante neste cenário foi a exploração de regras para a tradução dos modelos de características em fórmulas da lógica proposicional, o que abriu muitas oportunidades interessantes de investigação da utilização dos sistemas baseados em lógica matemática [2]. Considerando as análises automáticas realizadas por sistemas motores de inferência, duas etapas são necessárias: (i) o modelo de característica é traduzido em uma representação na lógica proposicional e (ii) resolvedores embasados na lógica proposicional são utilizados para extrair informações úteis a partir do resultado da tradução anterior, tais como o número de produtos válidos e verificação de consistência.

As regras para redução das relações do modelo de características em fórmulas proposicionais são expressas na Tabela 1. As características do modelo são representadas pelas variáveis booleanas, as relações da árvore de características e suas restrições

Relações nos modelos de características	Fórmula proposicional correspondente	Fórmula na CNF (SAT)
Opcional	$b \rightarrow a$	$\neg B \vee A$
Mandatória	$b \leftrightarrow a$	$(\neg A \vee B) \wedge (\neg B \vee A)$
Ou não exclusivo	$a \leftrightarrow (b_1 \vee ... \vee b_n)$	$(\neg A \vee B_1 \vee B_2 \vee ... \vee B_n) \wedge$ $(\neg B_1 \vee A) \wedge$ $(\neg B_2 \vee A) \wedge ... \wedge (\neg B_n \vee A)$
Alternativa	$a \leftrightarrow ((b_1 \wedge \neg b_2 \wedge \neg b_n) \vee$ $(b_2 \wedge \neg b_1 \wedge \neg b_n) \vee$ $... \vee$ $(b_n \wedge \neg b_1 \wedge ... \wedge \neg b_{n-1}))$	$(B_1 \vee B_2 \vee ... \vee B_n \vee \neg A) \wedge$ $(\neg B_1 \vee \neg B_2) \wedge ... \wedge$ $(\neg B_1 \vee \neg B_n) \wedge (\neg B_1 \vee A) \wedge$ $(\neg B_2 \vee \neg B_3) \wedge ... \wedge$ $(\neg B_2 \vee \neg B_n) \wedge (\neg B_2 \vee A) \wedge$ $(\neg B_{n-1} \vee \neg B_n) \wedge$ $(\neg B_{n-1} \vee A) \wedge (\neg B_n \vee A)$
restrições entre os ramos da árvore	direto em fórmulas proposicionais	direto na CNF (SAT)

Tabela 1. Regras para traduzir as relações dos modelos de características em fórmulas proposicionais e posteriormente para CNF (SAT).

entre os ramos da árvore representam as relações lógicas na fórmula. Como cada fórmula proposicional pode ser convertida em uma fórmula equivalente na CNF utilizando equivalências lógicas [4], os modelos de características podem ser reduzidos diretamente para o padrão CNF. Este padrão é formado por conjunções de cláusulas, em que cada cláusula é uma disjunção de literais, e um literal é uma variável ou sua negação. A fórmula $(A \vee B) \wedge (\neg A \vee \neg B)$ é uma codificação na CNF compreendendo duas cláusulas, com dois literais cada. Como via de regra, a Tabela 1 mostra como as relações entre as características podem ser reduzidas em fórmulas booleanas na CNF.

3.2. Um arcabouço resolvedor SAT

Neste ponto, de posse das fórmulas expressas na CNF, o emprego de um resolvedor SAT se faz necessário. Para cumprir a proposta deste trabalho, deve-se considerar apenas os resolvedores SAT completos, mais especificamente os algoritmos baseados no método DPLL [5], pois estes sempre conseguem determinar com exatidão se uma dada instância do problema SAT é satisfazível ou não.

Recentemente, muitos algoritmos baseados no método DPLL (método nomeado pelo acrônimo do nome dos seus criadores Davis-Putnam-Logemann-Loveland) [5] estão surgindo como os mais eficientes algoritmos para SAT [9] e pesquisas na área têm se voltado para incorporar novos algoritmos ao referido método. Neste cenário, o presente arcabouço faz uma ampla utilização do padrão de projeto *Template Method* [7] para fixar a estrutura de execução do DPLL (*Frozen-spots*) e fornece diversos *Hot-spots* para personalização deste. Cada ponto de extensão presente no algoritmo é implementado utilizando o padrão de projeto *Strategy* [7], o que permite a inclusão de novos algoritmos específicos para verificação de consistência. O resolvedor SAT proposto neste trabalho, representado no Algoritmo 1, é centrado é um núcleo DPLL.

Neste contexto de aplicação, o resolvedor SAT proposto tem como objetivo fornecer pontos de extensão para apoiar o desenvolvimento de novos métodos especializados na solução de instâncias reduzidas em SAT dos modelos de características. Outro ponto crucial de projeto contemplado pelo resolvedor SAT é o projeto do leitor de codificações SAT, em que foi aplicado o padrão de projeto *Builder* [7]. A partir deste esquema, o arcabouço se torna facilmente acoplável a qualquer ferramenta de modelagem de características. Ou seja, a classe que realiza a leitura da instância do problema pode realizar operações de *parsing* em arquivos XML. O XML é padrão geralmente adotado

nas ferramentas atuais para representar o modelo de características [10].

Algorithm 1 : Análise SAT do modelo de características

```
1: procedure MÉTODO DPLL(MC : Modelo de características reduzido na CNF)
2:     deduzModelo := deduzModelo(estruturaDados)        ▷ Hot-spot estrutura de dados
3:     if notPreprocessamentoModelo() then               ▷ Hot-spot pré-processamento
4:         return Incosistente
5:     end if
6:     if consistente(MC) then
7:         return Consistente
8:     end if
9:     while existeCaracteristicas() do
10:        caracteristica := ramifica()                  ▷ Hot-spot ramificação
11:        propagar(caracteristica.atribuir, caracteristica)
12:        if notdeducao() then                          ▷ Hot-spot dedução
13:            repeat
14:                nivelDeduz := analisa()               ▷ Hot-spot análise de conflitos
15:                if nivelDeduz ≡ ∅ then
16:                    return Inconsistente
17:                end if
18:                retrocesso(nivelDeduz)
19:                refaz(nivelDeduz)
20:                propaga(inverteAtribuir(caracteristicas.atribuir), nivelDeduz)
21:            until notdeducao()
22:        end if
23:        if consistente(MC) then
24:            return Consistente
25:        end if
26:    end while
27: end procedure
```

Este algoritmo recebe como entrada um modelo de características reduzido na CNF. Logo após, o motor DPLL é invocado com o objetivo de encontrar insconsistências no modelo em questão. Caso não sejam encontradas inconsistências no modelo após a execução do algoritmo núcleo DPLL, este método é finalizado fornecendo como resposta à ferramenta de modelagem a mensagem que o modelo de característica é consistente. Mais detalhadamente, a consistência de um determinado modelo pode ser decidida verificando satisfatibilidade da sua fórmula correspondente. Para detectar se uma dada característica f é morta, faz a redução da fórmula em $\delta \wedge f$ e verifica sua satisfatibilidade, em que δ é uma fórmula correspondente do modelo de características. Se $\delta \wedge f$ é insatisfazível, é dado que a característica f é morta. Um raciocínio semelhante pode ser usado para verificar se f é comum. Basta verificar a satisfatibilidade de $\delta \wedge \neg f$, inferindo que f é comum quando o resultado for insatisfazível, ou que f não é comum caso contrário. O código fonte do projeto de algoritmo desenvolvido neste trabalho encontra-se disponível para download no endereço eletrônico: `http://sourceforge.net/projects/arcaboucosat/`.

4. Trabalhos relacionados

Em um trabalho anterior [2], foi fornecida a fundamentação teórica necessária para o entendimento de como fazer a tradução dos modelos de características em fórmulas da lógica matemática e assim aplicar os motores de inferência para verificação formal dos modelos. Em outro trabalho [3] foi realizada uma análise comparativa do desempenho computacional de três abordagens baseadas na lógica para verificação que são SAT, BDD e CSP. O resultado deste trabalho mostrou que SAT, em média, obteve excelentes resultados em tempos computacionais e um bom uso de memória. Já o uso de BDD relatou um alto consumo de memória, chegando a ser exponencial ao tamanho do modelo de características. No entanto, BDD foi mais eficiente do que SAT. A abordagem via CSP perdeu para SAT e BDD nos dois quesitos comparados. Pesquisas realizadas [11], têm indicado um desempenho muito bom de resolvedores SAT para verificações em modelos de características de grande escala, contendo milhares de características. Este fato tem encorajado pesquisadores a explorar outros algoritmos SAT para análise dos modelos.

Em todos os trabalhos considerados anteriormente, usa-se o resolvedor SAT4J para verificação de consistência. Este resolvedor é fechado (caixa preta), ou seja, não possui pontos de extensão, o que impossibilita a experimentação de novos algoritmos, e ele não foi projetado especificamente para o problema em questão. Neste cenário da utilização do SAT4J, o experimento de novas abordagens e novos algoritmos é praticamente impossível. Assim, o diferencial deste trabalho é possibilitar, via o algoritmo proposto, fornecer um arcabouço resolvedor SAT no intuito de permitir o desenvolvimento de novos algoritmos, realizar experimentos diversos intercambiando heurísticas para encontrar um melhor tempo de resposta ao problema da modelagem de características.

5. Conclusão

O objetivo principal deste trabalho foi apresentar um resolvedor SAT para verificação dos modelos de características. Este artigo discutiu o uso desta técnica da lógica proposicional como verificadores embutidos em ferramentas de modelagem de variabilidade para LPS. É intenção deste trabalho, através do algoritmo proposto, contribuir de forma direta para os futuros desenvolvedores de ferramentas de modelagem de SPL. Contribuir também para os usuários das ferramentas de modelagem, bem como toda a comunidade de Engenharia de Software envolvida com pesquisa nesta área. Decorrente da inexistência de um *Benchmark* contendo instâncias reduzidas dos modelos de características na literatura, a análise dos resultados alcançados pelo algoritmo proposto neste trabalho foi inviabilizada.

No entanto, como pesquisa futura, nosso grupo de trabalho pretende desenvolver um *website* carregado de instâncias SAT-SPL, a fim de possibilitar a análise dos resultados do algoritmo proposto em comparação a outras abordagens. Outro trabalho futuro é a observação da utilização do algoritmo proposto a fim de intercambiar heurísticas de SAT comparando resultados frente ao resolvedor caixa preta SAT4J. É planejado a realização de testes nos mais diversos casos e registrar seus comportamentos. Novos algoritmos para este problema em particular podem ser desenvolvidos através dos pontos de extensão, realizando um estudo aprofundado do problema em questão. Pela análise de suas complexidades, espera-se otimizar tempos de resposta, devido à tendência de ferramentas online de edição para SPL.

6. Agradecimentos

Este trabalho recebeu apoio financeiro da FAPEMIG, processos APQ-02932-10 e APQ-02376-11, e do CNPq processo 485235/2011-0.

Referências

[1] Ebrahim Bagheri, Tommaso Di Noia, Azzurra Ragone, and Dragan Gasevic. Configuring Software Product Line Feature Models Based on Stakeholders' Soft and Hard Requirements. In *14th International Software Product Lines Conference (SPLC), Springer*, 2010.

[2] Don S. Batory. Feature Models, Grammars, and Propositional Formulas. In *the International Software Product Lines Conference (SPLC)*, 2005.

[3] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortes. A First Step Towards a Framework for the Automated. *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006.

[4] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortes. Automated Reasoning on Feature Models. In *17th Conference on Advanced Information Systems Engineering, CAiSE 05, Porto, Portugal*, 2005.

[5] Adnan Darwiche and Knot Pipatsrisawat. *Complete Algorithms - Chapter in Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[6] Eduardo Figueiredo, Nelio Cacho, Claudio Sant Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sergio Soares, Fabiano Ferrari, Safoora Khan, Fernando Filho, and Francisco Dantas. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. In *Proceedings of the 30th International Conference on Software Engineering (ICSE), pp. 261-270. Leipzig, Germany.*, 2008.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. Porto Alegre: Bookman, 2000.

[8] Mikolas Janota and Joseph Kiniry. Reasoning about Feature Models in Higher-Order Logic. In *Software Product Lines (SPL) 2007. Kyoto, Japan.*, 2007.

[9] Filip Maric. Flexible Implementation of SAT solvers. *Theory and Applications of Satisfiability Testing - SAT*, 2009.

[10] Marcílio Mendonça, Moises Branco, and Donald Cowan. S.P.L.O.T. - Software Product Lines Online Tools. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA Companion)*, 2009.

[11] Marcílio Mendonça, Andrzej Wasowski, and Krzysztof Czarnecki. SAT-Based Analysis of Feature Models is Easy. In *the International Software Product Lines Conference (SPLC)*, pages 231–240, 2009.

[12] Salvador Trujillo Sven Apel and Christian Kästner. Product Lines that supply other Product Lines: A Service-Oriented Approach. In *Proceedings of the SPLC Workshop on Service-Oriented Architectures and Product Lines (SOAPL), Software Engineering Institute, Carnegie Mellon University.*, 2007.