Alexandre Manuel Tavares Bragança

Linguagens Específicas de Domínio no Desenvolvimento de *Sofware* de Gestão

Tese a desenvolver na Escola de Engenharia da Universidade do Minho para a obtenção do grau académico de Doutor em Tecnologias e Sistemas de Informação, Área de conhecimento em Engenharia da Programação e dos Sistemas Informáticos, sob a orientação científica do Prof. Doutor Eng.º Ricardo Jorge Silvério de Magalhães Machado

Plano de Trabalho

Universidade do Minho
Escola de Engenharia
Departamento de Sistemas de Informação
Julho de 2002

Índice

1. Introdução

1.1 Motivação

Um dos grandes problemas que afectam o sucesso dos projectos de engenharia de *sofware* é a mudança [Pressman, 1994]. As metodologias de desenvolvimento de *software*, embora abordem este problema com especial importância tentando flexibilizar o processo de engenharia de *software*, ainda estão bastante limitadas a este nível. Uma das principais limitações é o facto de ser necessário envolver o engenheiro de *software* de cada vez que os requisitos mudam. Por muito optimizado que esteja este processo, este facto limita potencialmente o grau de adaptação à mudança do *software*, para além de poder haver discrepâncias entre a linguagem do utilizador e a do engenheiro e, por consequência, os requisitos poderem não ser precisos. Os próprios métodos formais aplicados à engenharia de *software* que, teoricamente, permitem diminuir esta falta de precisão, pecam a este nível pois obrigam à sua utilização por especialistas de engenharia de *software*. Uma técnica que recentemente tem recebido a atenção dos meios científicos é a adopção de *linguagens especificas de domínio* (DSL – *Domain Specific Languages*) [Deursen e Klint, 1998].

1.2 Enquadramento

Tal como foi referido, um dos problemas que afectam o desenvolvimento de *software* é o suporte à mudança. Os requisitos iniciais estão em constante mudança o que implica a alteração do *software*. Nos primeiros desenvolvimentos de soluções informáticas, a disciplina da engenharia de *software* concebia o processo de desenvolvimento dos sistemas através de metodologias pouco flexíveis e orientadas fundamentalmente para projectos novos em que o principal esforço e preocupação se encontrava, nas fases que precediam a instalação de uma primeira versão. Rapidamente se observou que os sistemas informáticos têm um período de vida bastante extenso, sendo necessário adaptar os referidos sistemas às diversas evoluções, tanto funcionais como tecnológicas. Mais recentemente, outras variáveis vieram trazer mais instabilidade e complexidade ao *software*, nomeadamente a rápida evolução dos mercados imposta pela globalização da economia, assim como o impacto do comercio electrónico e das muito rápidas evoluções tecnológicas.

A engenharia de *software* tem respondido a estas necessidades adaptando e promovendo metodologias e processos que fundamentalmente visam o desenvolvimento de soluções informáticas num processo em "espiral" [Boehm, 1988], tentando aumentar a rapidez com que são incorporados novos requisitos (ou alterações de requisitos) nas soluções informáticas. Alguns modelos têm, de facto, conseguido aceitação generalizada nesta área, como por exemplo a orientação a objectos [Booch, 1986].

No caso particular do *software* tipo ERP[1], vulgarmente designado por *software de gestão*, este tipo de metodologias tem sido fortemente adoptado, tendo-se, no entanto, continuado a sentir os problemas referenciados, nomeadamente alguma dificuldade e

[1] *Enterprise Resource Planning*

demora na adequação do *software* às continuas mudanças que se fazem sentir. De facto, segundo [Lientz e Swanson, 1980], 70% do esforço de programação concentra-se na manutenção. No contexto local da realidade portuguesa pensa-se que tal situação possa ser ainda mais aguda, pois, se retirarmos algum *software* do tipo ERP mais recente que tem um caracter genérico, a outra vertente é composta por soluções com alguma idade (pelo menos em termos informáticos) cujos domínios de aplicação já são mais restritos (determinados ramos de actividade ou tipos de industria), e que, portanto, sendo mais específicos poderão potencialmente estar mais expostos à mudança do que domínios mais genéricos.

1.3 Proposta

As abordagens clássicas que são feitas aos problemas apresentados propõem intervenções ao nível das metodologias e das ferramentas de forma a suportarem uma mais fácil e rápida incorporação de alterações, contudo continuam a implicar o recurso à equipa de análise e desenvolvimento. O mesmo acontece com métodos mais recentes que efectuam abordagens mais pragmáticas a estes problemas, como é o caso da programação extrema [Beck, 1999]. Este tipo de abordagem é cada vez menos aceitável pois recorre ao engenheiro de *software*, ou seja, a um especialista tecnológico da ferramenta(s) utilizada(s) no desenvolvimento da solução informática. Desta forma a gestão da mudança passa, em grande parte das vezes, pelo engenheiro o que limita grandemente um dos factores mais importantes para o sucesso de um *software* (para além do tempo de mudança): a flexibilidade da solução informática em "aceitar" mudanças, ou seja, o seu grau de "amigabilidade" face ao utilizador [Thibault, 1998].

Para colmatar esta deficiência, vários exemplos de abordagens baseadas na incorporação de linguagens de utilizador final têm surgido [Deursen *et al.*, 2000]. A ideia básica é permitir que o utilizador final intervenha no processo de adequação do *software* às suas necessidades sem passar pela equipa original que o desenvolveu. Exemplos bastante conhecidos desta situação são as linguagens de *script* que podemos encontrar em diversos pacotes de aplicações de escritório. Basicamente, a ideia é dar a possibilidade ao utilizador final para que este possa adaptar a aplicação à evolução das suas necessidades. Como a aplicação é desenvolvida para resolver um determinado problema, o utilizador final apenas pode adaptar a aplicação no contexto do referido problema, daí a designação de linguagem específica de domínio.

Apesar de ser cientificamente uma área recente, alguns dos princípios das DSL já têm uma longa história. Embora os princípios sejam prometedores, bastantes questões ainda se levantam na adopção das DSL, nomeadamente:

- a dificuldade em encontrar o âmbito correcto de uma DSL;
- o custo do desenho, implementação e manutenção das DSL;
- a dificuldade no balanceamento entre as construções específicas de domínio e as de programação genérica da linguagem.

O trabalho que se propõe efectuar neste projecto de doutoramento consistirá em definir um modelo para a utilização das DSL no contexto da engenharia de *software*, em particular no que respeita o desenvolvimento de *software* do tipo ERP. A ideia é a incorporação de

DSL no *software* de gestão, permitindo, desta forma, que aspectos que seriam abordados em termos de engenharia de *software* "clássica" passem a poder ser embebidos no *software* final e utilizados pelo utilizador final para adaptar e evoluir o *software*. Esta abordagem implica que na própria concepção do *software* se consiga determinar o âmbito do domínio da aplicação de forma adequada e extensível. O utilizador final deverá moldar o *software* de forma a torná-lo utilizável no contexto do domínio aplicacional. Desta forma, as eventuais alterações de evolução ou correcção do *software* poderão ser incorporadas pelo próprio utilizador final.

Assim, vai ser explorada a possibilidade da adopção de linguagens específicas de domínio como forma de aumentar o grau de adaptação e evolução do *software*. Os objectivos são:

- definir um modelo de desenvolvimento de *software* de gestão, baseado no conceito de DSL
- desenvolver a tecnologia e ferramentas DSL de suporte ao modelo
- validar o modelo e a tecnologia propostos através de casos de estudo baseados em soluções reais.

Pretende-se que os casos de estudo sejam provenientes de projectos reais executados pela I2S S.A.. A I2S S.A. é líder no mercado nacional de soluções informáticas para a actividade seguradora e tem vindo a incorporar, nas suas soluções, tecnologias que se enquadram no âmbito das linguagens específicas de domínio. Esta parceria permitirá, por um lado, a validação do trabalho desenvolvido com base em casos de estudo de implementações concretas e, por outro lado, o desenvolvimento previsto de tecnologias que suportem o modelo a propor terá como base a evolução de uma linguagem de domínio que a empresa desenvolveu no inicio dos anos 90 para suportar a especificação de calculo actuarial. Com esta abordagem ao trabalho a desenvolver nesta tese pretende-se atingir uma forte vertente de validação baseada em casos concretos.

Na secção 2 deste documento apresenta-se um levantamento sobre o estado actual das linguagens específicas de domínio, por forma a que se entenda melhor o trabalho que se propõe desenvolver. Na secção 3 apresenta-se a proposta de caso de estudo com a qual se pretende experimentar e validar o trabalho a desenvolver. Finalmente, dedica-se uma secção à explanação do plano de trabalhos a desenvolver ao longo dos três anos e conclui-se com um resumo dos objectivos da tese e sua justificação.

2. Linguagens Específicas de Domínio

2.1 O Conceito de DSL

Segundo [Hudak, 1998] e [Thibault, 1998] uma linguagem específica de domínio é uma linguagem de programação adequada para um domínio de aplicação particular. Uma DSL deve possuir a capacidade de desenvolver aplicações completas para um domínio específico. Uma DSL deve capturar precisamente a semântica de um domínio de aplicação. Bentley [Bentley, 1986] também sugere que as DSL são pequenas linguagens (*little languages*). Assim, uma DSL não é necessariamente genérica. De facto, as linguagens específicas de domínio são linguagens de programação que sacrificam a generalidade contra a aproximação a uma determinada área de problemas [UTCAT].

Alguns autores aprofundam um pouco mais este conceito defendendo que as DSL são linguagens que permitem especificar estereótipos de tarefas computacionais quer em domínios de aplicação especializados quer em terminologia familiar aos peritos de um domínio [Kieburtz, 2001]. Em particular, esta característica de especificidade é normalmente conseguida através de notações e abstracções apropriadas a um domínio particular de problemas. Esta visão de especialização implica que as linguagens específicas de domínio sejam normalmente declarativas e, por consequência, possam ser vistas como linguagens de especificação, assim como linguagens de programação [Deursen *et al.*, 2000].

Embora requeira um investimento inicial na análise e desenho do domínio, logo que uma DSL é criada, esta pode ser usada e reutilizada para endereçar uma diversidade de problemas no domínio. De um certo ponto de vista, uma DSL é um meio para capturar, representar e reforçar o desenho, num formato que encoraja a reutilização.

Muitas DSL são suportadas por um compilador de DSL que gera aplicações com base em programas DSL. Neste caso, o compilador DSL é referido como gerador de aplicações [Cleveland, 1988], e a DSL como linguagem específica de aplicação. Outras DSL, como o YACC [Bentley, 1986] ou o ASDL [Wang *et al.*, 1997], não têm como objectivo a programação (especificação) completa de aplicações, mas apenas a geração de bibliotecas ou componentes. Existem também DSL cuja execução consiste na geração de documentos (TEX), ou figuras (PIC) [Bentley, 1986]. Um termo comum para DSL orientadas para a construção de sistemas de processamento de dados de negócio é o de linguagens de 4ª geração (4GL) . Um dos exemplos mais comuns deste tipo de linguagem é o SQL [Pressman, 1994].

Relacionado com a programação específica de domínio está a programação de utilizador final (*end-user programming*), que acontece quando os utilizadores finais executam tarefas simples de programação, utilizando linguagens de macros ou *scripting*. Um exemplo típico é a programação de folhas de calculo, através da linguagem de macro Excel.

2.2 Exemplos

Segundo [Hudak, 1998], exemplos comuns de DSL incluem o Lexx e o Yacc para programas de análise léxica e *parse*, PERL para manipulação de texto, VHDL para descrição de hardware, TeX and LaTex para preparação de documentos, HTML e SGML para documentos com *links*, Tcl/Tk para *scripting* de interface gráfico, VRML e OpenGL para gráficos 3D, Mathematica e Maple para computação simbólica, AutoLisp para desenho assistido por computador. Algumas linguagens do tipo genérico também se podem classificar de DSL, como por exemplo o Prolog e as linguagens funcionais como o Haskell e o ML.

Um dos problemas das DSL reside precisamente na sua especificidade. De facto, é bastante provável que apenas tenhamos um conhecimento limitado das DSL existentes, visto que uma parte significativa destas mantém-se escondida do público precisamente devido à sua especificidade as tornar proprietárias das aplicações ou empresas que as desenvolveram. Segundo [Deursen *et al.*, 2000], existem literalmente centenas de DSL em utilização. Destas, apenas um subconjunto é descrito na bibliografia da engenharia de *software* ou das linguagens de programação.

[Deursen *et al.*, 2000] agrupa as DSL nas seguintes áreas:

- Engenharia de Software
  Produtos financeiros [Brand *et al.*, 1996] [Deursen, 1997] [Deursen e Klint, 1998], controlo e coordenação de comportamento [Bergstra e Lint, 1998] [Bertrand e Augeraud, 1999], arquitecturas de *software* [Medvidovic e Rosenblum, 1997] e bases de dados [Horowitz *et al.*, 1985].

- Sistemas de Software
  Descrição e análise de árvores de sintaxe abstractas [Wang *et al.*, 1997] [Crew, 1997] [Leijen e Meijer, 1999], especificação de *drivers* de dispositivos de vídeo [Thibault *et al.*, 1999], protocolos de coerência de *cache* [Chandra *et al.*, 1999], estruturas de dados em C [Smaragdakis e Batory, 1997] e especialização de sistemas operativos [Pu *et al.*, 1987].

- Multimedia
  Computação para a Internet [Cardelli e Davies, 1999] [Fuchs, 1997] [Atkins *et al.*, 1999] [Fernandez *et al.*, 1999], manipulação de imagem [Stevenson e Fleck, 1997], animação 3D [Elliott, 1999] e desenho [Kamin e Hyatt, 1997].

- Telecomunicações
  Linguagens para verificação de modelos [Klarlund e Schwartzbach, 1999], protocolos de comunicação [Basu *et al.*, 1997], *switches* de telecomunicação [Ladd e Ramming, 1994], e *signature computing* [Bonachea *et al.*, 1999].

- Diversos
  Simulação [Antoniotti e Göllü, 1997] [Bruce, 1997], agentes móveis [Gray, 1995], controlo de autómatos [Peterson et al., 1999], resolução de equações

diferenciais [Dinesh et al., 1998] e desenho de *hardware* digital [Jennings e Beuscher, 1999].

2.3 Pontos Fortes

Através da redução da distância conceptual entre o espaço do problema e a linguagem utilizada para exprimir o problema, a programação torna-se mais simples, fácil e de maior confiança [UTCAT]. A quantidade de código que é necessário escrever é reduzida, aumentando a produtividade e decrementando o custos de manutenção. A adopção de uma aproximação baseada em DSL à engenharia de *software* envolve oportunidades, mas também riscos [Deursen *et al.*, 2000]. Uma DSL bem desenhada (linguagem de qualidade) tenta encontrar um balanceamento equilibrado entre os referidos factores. As DSL permitem que as soluções sejam expressas no idioma e no nível de abstracção do problema de domínio. Por consequência, os peritos de domínio podem compreender, validar, alterar e, eventualmente desenvolver programas em DSL.

De seguida, apresenta-se um apanhado das vantagens mais significativas apontadas às DSL:

- Os programas em DSL são normalmente concisos, auto-documentados e podem ser reutilizados para diferentes propósitos [Ladd e Ramming, 1994];
- As DSL podem melhorar a produtividade, clareza, manutenção [Deursen e Klint, 1998] [Kieburtz *et al.*, 1996] e portabilidade [Herndon e Berzins, 1988];
- As DSL encorporam conhecimento de domínio, e desta forma, permitem a conservação e reutilização deste conhecimento [Deursen *et al.*, 2000];
- As DSL permitem a validação e optimização ao nível do domínio (e não da programação genérica) [Basu *et al.*, 1997] [Bruce, 1997] [Menon e Pingali, 1999];
- As DSL melhoram o grau de testabilidade, através de aproximações do género das descritas em [Sirer e Bershad, 1999];
- Permitem a reutilização no desenho do *software*, através da reutilização de geração automática de programas [Kieburtz, 2001].

2.4 Pontos Fracos

Para além dos pontos fortes e vantagens da adopção das DSL, é possível identificar alguns aspectos que devem ser ponderados. Segundo a [UTCAT]**,** os maiores problemas das DSL concentram-se na sua criação. De facto, a criação de uma DSL envolve um investimento em compreensão da área do problema: – *Quais são as características importantes, partilhadas dos problemas no domínio*? *Como podem estas características*

ser abstraídas? *Quais as características que tendem a mudar entre problemas*? Estas questões devem ser muito bem abordadas sob pena de se desenvolver uma DSL que não corresponde claramente ao domínio do problema.

Outros autores, como [Deursen *et al.*, 2000], defendem também que as grandes questões relativamente às DSL se encontram na fase da sua concepção. Segundo este, as DSL podem sofrer das seguintes desvantagens:

- o custo do desenho, implementação e manutenção das DSL;
- a dificuldade em encontrar o âmbito correcto de uma DSL;
- a dificuldade no balanceamento entre as construções específicas de domínio e as de programação genérica da linguagem;
- a potencial diminuição de eficiência quando comparada com *software* escrito manualmente numa linguagem de programação genérica.

Outros autores referem também problemas ao nível da utilização das DSL, nomeadamente a disponibilidade limitada das DSL [Krueger, 1992]. O custo da educação dos utilizadores das DSL pode ser um factor significativo quando é requerido que os utilizadores passem a utilizar uma sintaxe menos familiar nas suas especificações [Deursen *et al.*, 2000].

2.5 Desenho e Implementação

A realização de linguagens específicas de domínio difere de forma fundamental da realização de linguagens tradicionais de programação [Spinellis, 2001]. Embora o conceito DSL esteja já maduro [Landin, 1966], o seu papel na arquitectura, desenho e implementação de sistemas de *software* só recentemente tem sido reconhecido [Ramming, 1997]. Algumas DSL têm sido desenhadas como linguagens genéricas de programação [Wirth, 1974], e implementadas como interpretadores ou compiladores usando técnicas tradicionais de implementação de linguagens de programação [Aho *et al.*, 1985]. No entanto, o processo e a economia na realização de uma DSL são, de forma mais comum que se possa imaginar, bastante diferentes daqueles que conduzem a implementação de linguagens de programação tradicionais. Em particular, as DSL são, por definição, parte de uma sistema mais abrangente e, muitas vezes, implementadas para um domínio de aplicação muito restrito. Os recursos disponíveis para o seu desenho e implementação são, por isso, constrangidos a uma pequena percentagem daqueles que estão disponíveis para o sistema em que estão inseridas e dificilmente podem ser amortizados com base num grupo alargado de utilizadores. Os constrangimentos no esforço e talento que podem ser atribuídos ao desenho e implementação de uma DSL fizeram aparecer várias estratégias distintas de reutilização. Estas estratégias de realização de DSL resolvem problemas específicos do desenho e podem ser aplicadas a muitos problemas similares. A descrição

destes desenhos reutilizáveis, muitas vezes referidos como padrões[2] [Alexander *et al.*, 1977] [Coplien e Schmidt, 1995] [Gamma *et al.*, 1995], permite a sua disseminação e utilização conscienciosa por parte dos engenheiros de DSL.

[Deursen *et al.*, 2000] sugere um ciclo de vida típico para uma DSL

- Análise
  (1) Identificar o domínio do problema.
  (2) Recolher todo o conhecimento relevante neste domínio.
  (3) Agregar este conhecimento num grupo de noções semânticas e operações sobre estas.
  (4) Desenhar uma DSL que descreva com precisão aplicações nesse domínio.

- Implementação
  (5) Construir uma biblioteca que implemente as noções semânticas.
  (6) Desenhar e implementar um compilador que traduza programas DSL numa sequência de chamadas à biblioteca.

- Utilização
  (7) Escrever programas DSL para todas as aplicações desejadas e compilá-los.

Os passos (5) e (6) de implementação podem ser executados através de diferentes aproximações.

Uma aproximação clássica ao desenvolvimento de DSL é a interpretação ou compilação. Podem ser utilizadas ferramentas vulgares de implementação de compiladores [Aho *et al.*, 1986] [Bentley, 1986] ou ferramentas dedicadas à implementação de DSL, como o Draco [Neighbors, 1984], ASF+SDF [Deursen *et al.*, 1996], Kephera [Faith *et al.*, 1997], Kodiyak [Herndon e Berzins, 1988] ou o InfoWiz [Nakatani e Jones, 1997].

A principal vantagem em construir um compilador ou interpretador consiste no facto de que a implementação é completamente dirigida para a DSL e não é necessário efectuar concessões relativamente à notação ou primitivas da linguagem. Estas concessões estão quase sempre presentes quando se baseia uma DSL numa linguagem de programação existente. Também a detecção de erros, análise estática e as optimizações podem ser efectuadas ao nível do domínio quando se constrói um compilador ou interpretador de base para a DSL.

É claro que um problema importante é o custo de construir um compilador ou interpretador de base (normalmente) sem reutilizar partes de outras implementações de DSL (embora algumas ferramentas tenham funcionalidades apropriadas para ajudar a ultrapassar esses problemas - [Nakatani e Jones, 1997]).

Uma alternativa à implementação completa de uma DSL é a sua implementação com base na extensão de uma linguagem base. Por exemplo, [Basu *et al.*, 1997] descreve a extensão de uma versão restrita de uma linguagem de uso geral com construções

[2] *patterns*

específicas de domínio. A grande vantagem desta aproximação é que todas as características da linguagem base continuam disponíveis e não necessitam de ser implementadas.

Quando se implementam extensões específicas de domínio numa linguagem base, a implementação da linguagem base pode ser reutilizada de três forma distintas:

- Linguagem embebida (bibliotecas específicas de domínio)
  Nesta aproximação, os mecanismos existentes, tais como as definições de funções ou de operadores com sintaxe definida pelo utilizador, são utilizadas para construir uma biblioteca de operações específicas de domínio. Os mecanismos sintácticos da linguagem base são utilizados para exprimir o idioma do domínio.
  Uma vantagem desta aproximação consiste no facto de que o compilador ou interpretador da linguagem base é reutilizado na DSL (sem nenhuma alteração). A maior limitação é o grau de expressividade dos mecanismos sintácticos da linguagem base. Em muitos casos, a notação óptima do domínio fica comprometida pelas limitações da linguagem base. Exemplos típicos desta aproximação são a linguagem de controlo de *robots* embebida em Haskell descrita em [Peterson *et al.*, 1999] e uma linguagem de desenho embebida em ML [Kamin e Hyatt, 1997]. O conceito de *domain-specific embedded language* foi *criado* por Hudak [Hudak, 1996].

- Pré-processamento
  Nesta abordagem,os novos construtores da DSL são traduzidos para instruções na linguagem base através de um pré-processador. A grande vantagem desta aproximação é a simplicidade. Como desvantagens temos que a validação estática e a optimização não são efectuadas ao nível do domínio. Por consequência, o código gerado é susceptível de erros e o utilizador é notificado destes possíveis erros ao nível da linguagem base ou apenas em tempo de execução.

- Compilador ou interpretador estendido
  Esta abordagem é similar à anterior mas a fase de pré-processamento é integrada no compilador. A vantagem é que torna-se possível efectuar uma maior validação de tipos e uma melhor optimização de código. Esta é efectuada em [Engler, 1999] [Stichnoth e Gross, 1997]. O interpretador Tcl [Ousterhout, 1998] é também um exemplo desta aproximação e foi estendido para dezenas de domínios.

Para além da construção de um compilador ou interpretador dedicado de DSL, ou da reutilização de uma linguagem base, podem ser utilizadas outras formas de implementação nomeadamente através da combinação das técnicas descritas.

3. O Caso de Estudo

3.1 Apresentação da Empresa

A I2S é uma empresa nacional de desenvolvimento de *software* que iniciou a sua actividade em 1985. Actualmente, desenvolve a sua actividade principalmente na área financeira e em particular na indústria seguradora. Os produtos e serviços que comercializa suportam todas as funcionalidades da actividade seguradora, assim como todos os seus intervenientes. Esta especialização levou ao desenvolvimento, no inicio da década de 90 do século passado, do embrião da linguagem LPS[3] que hoje é utilizada de forma alargada nas soluções da empresa. Esta linguagem é descrita em [Bragança, 1998], sendo já referida como uma linguagem específica de negócio.

3.2 Caso de Estudo

O trabalho desta tese enquadra-se no contexto das linguagens de domínio utilizadas em *software* de gestão. Pretende-se demonstrar que a utilização de DSL pode melhorar o grau de flexibilidade, manutenção e evolução das aplicações. Em particular, pretende-se validar esta tese em aplicações concretas da família das aplicações de gestão. Na realidade portuguesa, a empresa I2S revela-se um caso exemplar, pois desenvolve há alguns anos aplicações de gestão para uma área de domínio particular que é a actividade seguradora. Pretende-se validar o modelo a desenvolver de arquitectura de *software* baseado em DSL através da sua aplicação em casos concretos de *software* da I2S. A área de actividade seguradora revela-se bastante dinâmica em termos de mercado, com um elevado grau de particularização em cada implementação concreta de utilizador final e bastante exposta à incorporação de funcionalidades *eBusiness* sendo por isso um caso bastante adequado para a validação do trabalho a desenvolver nesta tese.

[3] Linguagem para Seguros

4. Plano de Trabalho

Para atingir os objectivos enunciados nesta proposta de doutoramento prevê-se um plano de trabalho que inclui as seguintes fases que são apresentadas por ordem cronológica.

Fase 1: Estudo do estado da arte das linguagens específicas de domínio

Inicio: Setembro de 2002

Duração: 4 meses

Esta fase consistirá num levantamento dos desenvolvimentos mais recentes e significativos no contexto das DSL. Este levantamento terá por base o que foi exposto na secção 2 deste documento. Após o levantamento inicial do estado da arte segue-se o estudo detalhado das técnicas e tecnologias mais significativas encontradas. Pretende-se seleccionar as tecnologias que mais se aproximarem conceptualmente do âmbito deste trabalho.

O material resultante desta primeira fase permitirá a elaboração de um capítulo sobre o estado da arte das DSL.

Fase 2: Estudo das características do *software* de gestão e metodologias de desenvolvimento associadas

Inicio: Janeiro de 2003

Duração: 4 meses

Nesta fase, pretende-se efectuar um estudo sobre as características fundamentais do *software* de gestão. Assim, prevê-se, numa primeira etapa, efectuar um levantamento das características funcionais e tecnológicas deste tipo de *software*, assim como das metodologias e ferramentas mais comuns que são utilizadas para o seu desenvolvimento. Este estudo terá uma vertente bibliográfica assim como um levantamento dos dados referidos com base em entrevistas e contactos com equipas de desenvolvimento de soluções de *software* deste tipo. Prevê-se que esta primeira etapa necessitará de 3 meses para estar completa. Numa segunda etapa desta fase, desenvolver-se-á o estudo das eventuais especificidades do *software* de gestão para a actividade seguradora, visto ser neste âmbito que se pretende validar o trabalho desenvolvido nesta tese. Este estudo incluirá as aplicações da I2S S.A. (que serão a base dos casos de estudo) assim como as metodologias, técnicas e tecnologias que suportam actualmente o desenvolvimento das referidas aplicações.

Esta fase permitirá, por um lado, efectuar o levantamento das características do *software* de gestão e das metodologias utilizadas no seu desenvolvimento (incluindo manutenção e evolução), por outro lado, tipificar a manutenção e evolução que normalmente ocorre neste *software*, a forma como é efectuada, os tempos de resposta obtidos e o grau de suporte à mudança das referidas aplicações.

O material resultante desta segunda fase permitirá a elaboração de um capítulo sobre *software* de gestão que será elaborado tendo como princípio orientador o suporte à manutenção e evolução deste tipo de *software*.

Fase 3: Desenvolvimento da tese

Inicio: Maio de 2003

Duração: 3+6+12 meses

Esta fase consiste no núcleo da elaboração do trabalho da tese. Durante esta fase pretende-se desenvolver o princípio de que a incorporação de linguagens específicas de domínio no *software* de gestão permite aumentar o suporte à manutenção e evolução deste tipo de *software*.

Para desenvolver este princípio, esta fase consistirá nas seguintes etapas:

- Caracterizar com grande detalhe os problemas existentes actualmente no desenvolvimento, manutenção e evolução de *software* de gestão. Justificar de que forma as DSL quando utilizadas no *software* de gestão podem diminuir significativamente (ou resolver) este tipo de problemas. Abordar casos de incorporação de linguagens de domínio em *software* de gestão. Resultará desta etapa a elaboração de um capítulo sobre a utilização de DSL em *software* de gestão. Estima-se a necessidade de 3 meses para a conclusão desta etapa.
- Estudo detalhado da linguagem de domínio LPS e da sua utilização em *software* de gestão desenvolvido pela I2S, S.A.. O estudo detalhado das características técnicas e funcionais desta linguagem de domínio servirá de suporte ao modelo a propor para a incorporação de DSL no *software* de gestão. Resultará desta etapa a elaboração de um capítulo sobre a linguagem de domínio LPS e, eventualmente, a elaboração de um artigo científico a publicar em revista ou conferência expondo a LPS como caso de estudo exemplificativo da incorporação de DSL em *software* de gestão. Estima-se a necessidade de 6 meses para a conclusão desta etapa.
- O objectivo da última etapa desta fase é a elaboração da justificação teórica da tese que se defende, ou seja, um modelo e uma arquitectura aplicacional que suportem a incorporação de DSL em *software* de gestão. O resultado desta etapa poderá servir de base para um artigo científico a publicar em revista ou conferência expondo o modelo defendido na tese. Este modelo deverá ser validado nos casos de estudo previstos para a fase seguinte. Para que tal seja possível, prevê-se a possibilidade de se efectuarem desenvolvimentos adicionais à LPS de forma à enquadrar no modelo proposto. A utilização da LPS torna possível a validação do modelo proposto com base em aplicações e casos de estudo concretos de *software* de gestão da I2S S.A.. A linguagem LPS foi objecto de referência e descrição em [Bragança, 1998], tendo sido desenvolvida num projecto liderado pelo autor desta proposta de tese, sendo portanto justificada a sua utilização. Estima-se a necessidade de 12 meses para a conclusão desta etapa.

Fase 4: Caso de estudo

Inicio: Fevereiro de 2005

Duração: 4 meses

Esta fase consistirá, fundamentalmente, na experimentação do modelo desenvolvido na fase anterior com base na utilização da LPS nas aplicações de gestão da I2S, S.A.. O objectivo será validar e demonstrar na prática os princípios que a tese defende. O resultado desta fase poderá, eventualmente, ser alvo de um artigo científico a publicar em revista ou conferência expondo as conclusões retiradas destes casos de estudo.

O material resultante desta fase permitirá a elaboração de um capítulo da tese dedicado aos casos de estudo.

Fase 5: Conclusões e revisão do texto da tese

Inicio: Junho de 2005

Duração: 3 meses

Esta fase estará dividida em duas etapas. Numa primeira etapa, com uma duração esperada de dois meses, será efectuada uma reflexão sobre o trabalho desenvolvido. Esta reflexão permitirá efectuar uma análise crítica sobre o trabalho desenvolvido, tirando conclusões e avançando com hipóteses de seguimento do trabalho. A segunda etapa desta fase será o culminar do trabalho da tese, consistindo essencialmente na revisão do texto da tese. Prevê-se uma duração de 1 mês para esta etapa.

5. Conclusão

A manutenção e evolução do *software* constituem os problemas mais importantes que se levantam hoje em dia ao engenheiro de *software*. O *software* de gestão representa uma percentagem muito significativa do total de *software* existente. Pelas suas características este tipo de *software* é bastante vulnerável aos problemas referidos. As metodologias e técnicas actuais não abordam de forma satisfatória estes graves problemas. As linguagens específicas de domínio têm merecido elevada atenção da comunidade científica. A sua aplicação na tentativa de resolução de diversos problemas tem resultado num número significativo de experiências positivas. Com esta proposta de trabalho pretende-se validar a tese de que a aplicação de linguagens específicas de domínio no *software* de gestão (suportadas por um modelo adequado) permite melhorar significativamente o grau de flexibilidade e suporte à manutenção e evolução do *software*. Com o plano de trabalho apresentado pretende-se atingir esse objectivo de forma rigorosa e credível. Acredita-se que o texto resultante da elaboração desta tese se poderá constituir numa referência original e significativa nesta área científica.

Referências

[Aho *et al.*, 1985] Alfred V. Aho, Ravi Sethi e Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, 1985.

[Aho *et al.*, 1986] A.V. Aho, R. Sethi e J.D. Ullman. *Compiler: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[Alexander *et al.*, 1977] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King e Shlomo Angel. *A Pattern Language*. Oxford University Press, 1977.

[Antoniotti e Göllü, 1997] M. Antoniotti e A. Göllü. SHIFT and SMART-AHS: A language for hybrid system engineering modeling and simulation. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 1997. USENIX Association., páginas 171-182.

[Atkins *et al.*, 1999] D. Atkins, T. Ball, G. Bruns e K. Cox. Mawl: A domain-specific language for form-based services. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 334-346.

[Basu *et al.*, 1997] A. Basu, M. Hayden, G. Morrisett e T. von Eicken. A language-based approach to protocol construction. *DSL '97 - First ACM SIGPLAN Workshop on Domain-Specific Languages, in Association with POPL '97*, Paris, França, Janeiro 1997. University of Illinois Computer Science Report., páginas 1-15.

[Beck, 1999] Kent Beck. Extreme Programming Explained, Addison-Wesley, 1999

[Bentley, 1986] J. L. Bentley, "Programming pearls: Little languages.", *Communications of the ACM*, 29(8):711-721, Agosto 1986.

[Bertrand e Augeraud, 1999] F. Bertrand, M. Augeraud. BDL: A specialized language for per-object reactive control. In Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 347-362.

[Bergstra e Lint, 1998] J.A. Bergstra, P. Klint. The discrete time TOOLBUS--a software coordination architecture. *Science of Computer Programming*, 31:205-229, 1998.

[Boehm, 1988] B. Boehm, "A Spiral Model for Software Development and Enhancement", *Computer*, vol. 21, nº 5, Maio de 1988, pp. 61-72

[Bonachea *et al.*, 1999] D. Bonachea, K. Fisher, A. Rogers e F. Smith. Hancock: A language for processing very large-scale data. *Proceedings of the second USENIX*

Conference on Domain-Specific Languages. USENIX Association, Outubro 3-5 1999., páginas 163-176.

[Booch, 1986] G. Booch, "Object-Oriented Development", IEEE Trans. Software Engineering, vol. SE-12, nº 2, Fevereiro 1986, pp. 211ff

[Bragança, 1998] Alexandre Bragança, "Sistema para Gestão Operacional de Processos, Uma Visão Global na Indústria e nos Serviços no Contexto dos Sistemas de Fluxo de Trabalho"**,** Dissertação de Mestrado em Engenharia Electrotécnica e de Computadores, Especialização em Informática Industrial, FEUP, Abril de 1998

[Brand *et al.*, 1996] M. van den Brand, A. van Deursen, P. Klint, S. Klusener e E. van der Meulen, "Industrial applications of ASF+SDF", *Algebraic Methodology and Software Technology (AMAST '96)*, volume 1101 of *Lecture Notes in Computer Science*, pp 9-18. Springer-Verlag, 1996.

[Bruce, 1997] D. Bruce. What makes a good domain-specific language? APOSTLE, and its approach to parallel discrete event simulation. *DSL '97 - First ACM SIGPLAN Workshop on Domain-Specific Languages, in Association with POPL '97*, Paris, França, Janeiro 1997. University of Illinois Computer Science Report., páginas 17-35.

[Cardelli e Davies, 1999] L. Cardelli e R. Davies, Service combinators for web computing. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 309-316.

[Chandra *et al.*, 1999] S. Chandra, B. Richards e J. R. Larus. Teapot: A domain-specific language for writing cache coherence protocols. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 317-333.

[Cleveland, 1988] J. C. Cleaveland, "Building application generators", *IEEE Software*, pp. 25-33, Julho 1988.

[Coplien e Schmidt, 1995] James O. Coplien e Douglas C. Schmidt. *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[Crew, 1997] R. F. Crew. ASTLOG: A language for examining abstract syntax trees. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 15-17 1997. USENIX Association., páginas 229-242.

[Deursen *et al.*, 1996] A. van Deursen, J. Heering e P. Klint, editors. *Language Prototyping: An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing*. World Scientific Publishing Co., 1996.

[Deursen, 1997] A. Deursen, "Domain-specific languages versus object-oriented frameworks: A financial engineering case study", *Smalltalk and Java in Industry and Academia, TJA'97*, pp 35-39. Ilmenau Technical University, 1997.

[Deursen e Klint, 1998] A. Deursen, P. Klint, "Little languages: Little maintenance?", *Journal of Software Maintenance*, 10:75-92, 1998.

[Deursen *et al.*, 2000] A. van Deursen, P. Klint e J. Visser, "Domain-Specific Languages: An Annotated Bibliography", ACM SIGPLAN Notices 35(6):26-36, Junho 2000

[Dinesh *et al.*, 1998] T. B. Dinesh, M. Haveraaen e J. Heering. An algebraic programming style for numerical software and its optimization. Technical Report SEN-R9844, CWI, 1998. ACM CoRR Preprint Server cs.SE/9903002 (Março 1999). Submitted to *Scientific Programming*.

[Elliott, 1999] C. Elliott. An embedded modeling language approach to interactive 3D and multimedia animation. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 291-308.

[Engler, 1999] D. R. Engler. Interface compilation: Steps toward compiling program interfaces as languages. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 387-400.

[Faith *et al.*, 1997] R. E. Faith, L. S. Nyland e J. F. Prins. Khepera: A system for rapid implementation of domain specific languages. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 15-17 1997. USENIX Association., páginas 243-55.

[Fernandez *et al.*, 1999] M. Fernández, D. Suciu e I. Tatarinov. Declarative specification of data-intensive web sites. *Proceedings of the second USENIX Conference on Domain-Specific Languages*. USENIX Association, Outubro 3-5 1999., páginas 135-148.

[Fuchs, 1997] M. Fuchs. Domain specific languages for ad hoc distributed applications. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 15-17 1997. USENIX Association., páginas 27-36.

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

[Gray, 1995] R. Gray. Agent Tcl: A transportable agent system. In J. Mayfield and T. Finnin, editors, *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM'95)*, Dezembro 1995.

[Herndon e Berzins, 1988] R. M. Herndon e V. A. Berzins. The realizable benefits of a language prototyping language. *IEEE Transactions on Software Engineering*, SE-14:803-809, 1988.

[Horowitz *et al.*, 1985] E. Horowitz, A. Kemper e B. Narasimhan. A survey of application generators. *IEEE Software*, páginas 40-54, Janeiro 1985.

[Hudak, 1996] P. Hudak. Building domain-specific embedded languages. *ACM Computing Surveys*, 28(4es), December 1996.

[Hudak, 1998] P. Hudak. "Modular Domain Specific Languages and Tools". In Fifth International Conference on Software Reuse, páginas 134--142, Victoria, Canada, Junho 1998.

[Jennings e Beuscher, 1999] J. Jennings e E. Beuscher. Verischemelog: Verilog embedded in Scheme. *Proceedings of the second USENIX Conference on Domain-Specific Languages*. USENIX Association, Outubro 3-5 1999., páginas 123-134.

[Kamin e Hyatt, 1997] S. Kamin e D. Hyatt. A special-purpose language for picture-drawing. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 15-17 1997. USENIX Association., páginas 297-310.

[Kiczales *et al.*, 1997] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C. Lopes, C. Maeda, e A. Mendhekar. Aspect oriented programming. *DSL '97 - First ACM SIGPLAN Workshop on Domain-Specific Languages, in Association with POPL '97*, Paris, França, Janeiro 1997. University of Illinois Computer Science Report., páginas 75-88.

[Kieburtz *et al.*, 1996] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith e L. Walton. A software engineering experiment in software component generation. In *Proceedings of the 18th International Conference on Software Engineering ICSE-18*, páginas 542-553. IEEE, 1996.

[Kieburtz, 2001] Dick Kieburtz, "Retrospective on Formal Methods" OGI/2020, OGI School of Science and Engineering, OHSU, Setembro 2001.

[Klarlund e Schwartzbach, 1999] N. Klarlund e M. I. Schwartzbach. A domain-specific language for regular sets of strings and trees. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999., páginas 378-386.

[Krueger, 1992] C. W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131-183, Junho 1992.

[Ladd e Ramming, 1994] D. A. Ladd e J. C. Ramming. Two application languages in software production. In *USENIX Very High Level Languages Symposium Proceedings*, páginas 169-178, Outubro 1994.

[Landin, 1966] P. J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157-166, Maio 1966.

[Leijen e Meijer, 1999] D. Leijen e E. Meijer. Domain specific embedded compilers. *Proceedings of the second USENIX Conference on Domain-Specific Languages*. USENIX Association, Outubro 1999, páginas 109-122.

[Lientz e Swanson, 1980] B. Lientz e E. Swanson, *Software Maintenance Management*, Addison-Wesley, 1980

[Medvidovic e Rosenblum, 1997] N. Medvidovic e D. S. Rosenblum. Domains of concern in software architectures and architecture description languages. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 1997. USENIX Association., páginas 199-212.

[Menon e Pingali, 1999] V. Menon e K. Pingali. A case for source-level transformations in MATLAB. *Proceedings of the second USENIX Conference on Domain-Specific Languages*. USENIX Association, Outubro 1999, páginas 53-66.

[Nakatani e Jones, 1997] L. Nakatani e M. Jones. Jargons and infocentrism. *DSL '97 - First ACM SIGPLAN Workshop on Domain-Specific Languages, in Association with POPL '97*, Paris, França, Janeiro 1997. University of Illinois Computer Science Report., páginas 59-74.

[Neighbors, 1984] J. M. Neighbors. The Draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering*, SE-10(5):564-74, Setembro 1984.

[Ousterhout, 1998] J. K. Ousterhout. Scripting: Higher level programming for the 21st century. *IEEE Computer*, Março 1998.

[Peterson *et al.*, 1999] J. Peterson, P. Hudak e C. Elliott. Lambda in motion: Controlling robots with Haskell. In *PADL'99*, volume 1551 of *LNCS*, páginas 91-105, 1999.

[Pfahler e Kastens, 1997] P. Pfahler e U. Kastens. Language design and implementation by selection. *DSL '97 - First ACM SIGPLAN Workshop on Domain-Specific Languages, in Association with POPL '97*, Paris, França, Janeiro 1997. University of Illinois Computer Science Report., páginas 97-108.

[Pressman, 1994] Roger S. Pressman. “Software Engineering A Practitioner’s Approach”. McGraw Hill, 1994

[Pu *et al.*, 1987] C. Pu, A. Black, C. Cowan, J. Walpole e C. Consel. Microlanguages for operating system specialization. *DSL '97 - First ACM SIGPLAN Workshop on Domain-Specific Languages, in Association with POPL '97*, Paris, França, Janeiro 1997. University of Illinois Computer Science Report., páginas 49-57.

[Ramming, 1997] J. Christopher Ramming, editor. *USENIX Conference on Domain-Specific Languages*, Santa Monica, CA, USA, Outubro 1997. USENIX.

[Sirer e Bershad, 1999] E. G. Sirer e B. N. Bershad. Using production grammars in software testing. *Proceedings of the second USENIX Conference on Domain-Specific Languages*. USENIX Association, Outubro 1999, páginas 1-14.

[Smaragdakis e Batory, 1997] Y. Smaragdakis e D. Batory. DiSTiL: A transformation library for data structures. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 1997. USENIX Association., páginas 257-270.

[Spinellis, 2001] Diomidis Spinellis, Notable design patterns for domain specific languages. *Journal of Systems and Software*, 56(1):91-99, Fevereiro 2001.

[Stevenson e Fleck, 1997] D. E. Stevenson e M. M. Fleck. Programming language support for digitized images or, the monsters in the closet. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 1997. USENIX Association., páginas 271-284.

[Stichnoth e Gross, 1997] J. M. Stichnoth e T. Gross. Code composition as an implementation language for compilers. *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, Outubro 1997. USENIX Association., páginas 119-132.

[Thibault, 1998] S. Thibault, "Domain Specific Languages: Conception, Implementation and Application", PhD Thesis, Université de Rennes 1, 1998

[Thibault *et al.*, 1999] S. A. Thibault, R. Marlet e C. Consel. Domain-specific languages: From design to implementation application to video device drivers generation. Special issue on domain-specific languages. *IEEE Transactions on Software Engineering*, 25(3), Maio/Junho 1999, páginas 363-377.

[UTCAT] University of Texas Center for Agile Technology, http://www.cat.utexas.edu/dsl.html

[Wang *et al.*, 1997] D. C. Wang, A. W. Appel, J. L. Korn, C. S. Serra, "The Zephyr abstract syntax description language", *Proceedings of the USENIX Conference on Domain-Specific Languages*, Berkeley, CA, USENIX Association, pp 213-28, Outubro 1997

[Wirth, 1974] Niklaus Wirth. On the design of programming languages. In Jack L. Rosenfeld, editor, *Information Processing 74: Proceedings of IFIP Congress 74*, páginas 386-393, Stockholm, Sweden, Agosto 1974. International Federation for Information Processing, North-Holland Publishing Company.

Índice Remissivo