Um *Framework* para Criação de Linguagens de Domínio Específico

Sérgio Roberto P. da Silva, Josiane Mechiori Pinheiro

Departamento de Informática — Universidade Estadual de Maringá (UEM)
Av. Colombo, 5790, zona 07 – 87020-900 – Maringá – PR – Brasil

{srsilva,jmpinhei}@din.uem.br

Abstract*: In this paper, we propose a framework based on the use of a controlled natural language as a type-language for the instantiation of domain specific languages. We explain the advantages of introducing linguistic mechanisms from natural languages into programming languages to facilitate the learning and use of this language. We also show how to limit the set of linguistic mechanisms accepted by the language so that it could be made computationally efficient.*

Resumo*: Neste artigo, propomos um framework baseado no uso de uma linguagem natural controlada como uma linguagem-tipo para a instanciação de linguagens de domínio específico. Explicamos as vantagens de se introduzir mecanismos lingüísticos provenientes das linguagens naturais nas linguagens de programação para facilitar sua aprendizagem e uso. Também mostramos como limitar o conjunto de mecanismos lingüísticos aceitos pela linguagem de forma a torná-la computacionalmente eficiente.*

1. Motivação

Devido à complexidade das aplicações atuais, vem tornando-se prática entre os desenvolvedores de software estender linguagens de programação de amplo espectro para apoiar o desenvolvimento de aplicações em domínios específicos. Como resultado as linguagens de programação atuais estão tornando-se cada vez mais difíceis de serem utilizadas, devido à grande quantidade de comandos e bibliotecas que lhe são acrescidas—vide a linguagem Java [Sun 2004]. Numa tentativa de recuperar a usabilidade das linguagens de programação tem sido disseminada a idéia da criação de **Linguagens de Domínio Específico** (LDEs) [Deursen 2000]. Uma LDE é uma linguagem de programação ou linguagem de especificação executável que oferece, por meio de notações e abstrações apropriadas, poder expressivo focado e usualmente restrito a um domínio particular. Frechot (2003) acrescenta ainda que estas linguagens são geralmente pequenas, mais declarativas do que imperativas, e menos expressivas do que uma linguagem de propósito geral.

Em geral, as LDEs apresentam as seguintes vantagens, em relação as linguagens de programação convencionais [Deursen 2000]: a incorporação de conhecimento do domínio (permitindo a conservação e a reutilização deste conhecimento); a possibilidade da expressão de soluções no idioma e no nível de abstração do domínio do problema (habilitando os próprios *experts* do domínio a entender, validar, modificar e, até mesmo, desenvolver programas em uma LDE); a possibilidade de validar e otimizar os programas no nível de domínio; e o fato de que, em geral, seus programas são concisos, auto-documentados e podem ser reusados para diferentes propósitos,

resultando em uma melhoria da produtividade, da confiabilidade, da portabilidade e da manutenibilidade dos programas. No entanto, as LDEs também apresentam algumas desvantagens, tais como: sua disponibilidade limitada; a dificuldade de encontrar seu escopo próprio; o custo de projetá-las, implementá-las e mantê-las; e o custo de adaptação dos usuários à LDE.

Este artigo propõe um *framework* baseado em uma **linguagem-tipo**—a especificação de uma classe de linguagens—para o desenvolvimento de LDEs. Neste *framework* um projetista de linguagens disporá de um *parser* e um processador para uma sintaxe fixa de controle e de referenciação a objetos, e adaptará esta sintaxe instanciando o léxico específico do domínio, por meio da definição de uma ontologia para este domínio. Esta ontologia dará apoio à resolução de questões relativas às referências a objetos e à explicitação de *loopings* nos programas da LDE, conforme será descrito nas seções a seguir.

Nossa proposta visa atenuar as desvantagens inerentes às LDEs tornando seu ciclo de desenvolvimento e aprendizado mais curto. Primeiramente, procuramos facilitar o desenvolvimento de uma LDE, reduzindo esta etapa ao processo de instanciação de uma linguagem-tipo, por meio da criação de uma ontologia do domínio. Depois procuramos reduzir o custo de adaptação do usuário, baseando nossa linguagem-tipo em uma **linguagem natural controlada** (LNC)—um subconjunto da linguagem natural, cujo léxico, sintaxe e semântica são restritos a um domínio específico [Altwarg 2000]. Recentemente as LNCs têm sido propostas e usadas com relativo sucesso em várias tarefas [CLAW 2003,2000], mostrando que, em um domínio específico, esta pode ser uma boa alternativa.

Assim, na seção 2 deste artigo, apresentamos os motivos pelos quais resolvemos usar uma LNC como linguagem-tipo. Na seção 3, descrevemos a estrutura do *framework* proposto e o processo de instanciação de uma LDE. A seguir, na seção 4, descrevemos a linguagem-tipo que forma a base de nosso *framework*. Na seção 5, discutimos como tornar este *framework* computacionalmente eficiente. Finalmente, na seção 6, resumimos nossa proposta e comentamos sobre alguns caminhos futuros.

2. Porque usar uma linguagem natural controlada para gerar LDEs

O processo de aprendizagem de uma linguagem de programação é uma tarefa árdua e demorada. A maioria destas linguagens é projetada visando somente à eficiência computacional, deixando de lado os aspectos comunicativos necessários a qualquer linguagem para comunicação com pessoas. Assim, seus usuários têm que aprender uma linguagem de comunicação com a máquina totalmente diferente daquela a que estão acostumados a usar no dia-a-dia. Este aspecto se agrava ainda mais quando os usuários não têm nenhum tipo de experiência com programação, ou seja, são *experts* no domínio, porém leigos na arte de programar. No entanto, frente à capacidade computacional disponível nas máquinas atuais, esta estratégia de implementação não tem mais razão de ser, uma vez que o custo de criação e manutenção de um programa já suplanta em muito o custo do *hardware*.

Assim, buscando ampliar a usabilidade das linguagens de programação e tornar o processo de aprendizagem o mais natural possível, direcionamos nossa pesquisa no sentido de ampliar o tratamento de aspectos comunicativos nestas linguagens. Deste ponto de vista, a candidata mais apropriada para uma nova linguagem de programação

seria a linguagem natural, uma vez que os usuários já a conhecem e seus aspectos comunicativos são indiscutíveis. Entretanto, além da ambigüidade inerente a tal linguagem, o processamento de sua forma irrestrita ainda não é viável, apesar dos enormes avanços alcançados [CoLing 2002]. Contudo, o uso de uma linguagem natural controlada tem sido explorado em tarefas como a especificação de requisitos [Fuchs 1999] e a representação de conhecimento [Pulman 1996] entre outras, com relativo sucesso.

A grande semelhança de objetivos entre as LNCs e as LDEs nos levaram a pensar em utilizar as primeiras como base para as últimas. Outra grande vantagem de se usar uma LNC está no fato de que todas as sentenças usadas nesta LNC estão corretas na linguagem natural, facilitando aos usuários sua aprendizagem e uso. Porém, nem todas as sentenças de uma linguagem natural têm de ser permitidas em uma LNC, o que possibilita a eliminação das sentenças ambíguas. Assim, um usuário do domínio terá somente que aprender quais os tipos de sentenças ele pode e quais ele não pode usar durante a programação, o que se torna uma tarefa bem mais simples do que aprender uma linguagem inteiramente nova.

No entanto, é importante salientar que na definição de uma LNC não basta introduzir ricos mecanismos de comunicação a uma linguagem de programação. É necessário que estes sejam os mecanismos realmente empregados pelos usuários do domínio em questão, ou seja, uma LNC, em geral, é especificamente definida para a aplicação na qual será empregada. Além disso, é extremamente importante que o conjunto de mecanismos comunicativos introduzidos permita manter o processamento computacional da LNC dentro de uma faixa de eficiência razoável, de forma a não inviabilizar sua implementação.

Em trabalho anterior [Da Silva 2001], apresentamos uma análise da linguagem utilizada por usuários finais para expressar planos no dia-a-dia como, por exemplo, receitas e manuais de faça-você-mesmo. Naquele trabalho visávamos, dentre outras coisas, a obtenção de uma linguagem-tipo na forma de uma LNC para programação por usuários finais. Contudo, a função final desta linguagem era a de atualizar uma base de conhecimento do design da aplicação, contendo a ontologia do domínio que foi definida pelo design da aplicação e as extensões criadas pelos usuários. Este processo é muito semelhante ao processo de programação, visto que os usuários terão que definir objetos, referenciá-los e manipulá-los como em um programa.

Um ponto relevante, que notamos naquele trabalho, é que a estrutura dos planos descritos pelas pessoas no dia-a-dia permite expressar dois tipos de conhecimento: o procedimental, quando especificam o processo a ser executado sobre os recursos (que é semelhante à definição de comandos utilizados em uma linguagem de programação); e o declarativo, quando estão introduzindo os recursos a serem usados no processo (que é semelhante ao tipo de declaração feita na etapa de definição das variáveis usadas em um programa, ou da ontologia de um domínio).

O grande diferencial entre as linguagens de programação atuais e as linguagens comumente empregadas pelos usuários finais para a especificação de processos está no mecanismo empregado para a referenciação a objetos. A linguagem empregada por estes usuários apresenta uma forma muito natural de operar sobre objetos estruturados, usando **conhecimento de senso comum** do domínio e mecanismos lingüísticos como **quantificadores** (do tipo "*all*", "*every*" e "*each*" e plurais), **qualificadores**, **seletores** e

figuras de linguagem (anáforas, elipses, metáforas e metonímias) para facilitar a referenciação a estes objetos. Tais mecanismos ajudam as pessoas a fazer referência a características de objetos complexos de uma forma direta e simples, uma vez que eles ocultam muitos detalhes sobre a estrutura usada na implementação do objeto.

Além disso, é importante observar que os substantivos, que descrevem objetos no domínio, geralmente são substituídos por **pronomes** nas referências subseqüentes a sua introdução no texto, um caso de **anáfora**. Também é comum o uso de **elipses**, limitadas às formas básicas, para ocultar objetos das sentenças. Neste caso, os nomes ocultados sempre se referem a objetos já mencionados no contexto formado pelas sentenças anteriores, podendo ser omitidos sem prejuízo à sua interpretação. Além disso, em relação à estrutura do texto da linguagem de planos analisada, é interessante ressaltar que um parágrafo sempre realiza um passo de um plano, seja ele de transformação (execução de comandos no nosso caso) ou de declaração, e este passo sempre declara ou modifica um único objeto. Este objeto é o **foco do discurso** que, em geral, é bem marcado no texto e por isto muitas vezes é usado de forma implícita, para resolver anáforas presentes no texto, por exemplo.

Estes tipos de mecanismos comunicativos são muito importantes porque tornam o processo de descrever algo através de uma linguagem muito mais natural para os usuários. Além disso, previnem a necessidade de escrever e, portanto, interpretar sentenças muito longas, o que é particularmente difícil (para as pessoas). Por outro lado, devido às suas características, estas referências são de simples resolução, facilitando em muito sua implementação. Deste modo, o custo computacional exigido para a resolução destes mecanismos se torna baixo, quando comparado aos lucros obtidos na comunicação com o emprego dos mesmos, justificando, assim, nossa escolha pelo uso de uma LNC na definição da linguagem-tipo a ser empregada em nosso *framework* para a criação de LDEs.

É importante ressaltar que, apesar do *corpus* empregado naquela análise referir-se à língua inglesa, os comentários lá realizados também são válidos para outras línguas (feitos os devidos ajustes), uma vez que foram analisados elementos que pertencem aos "mecanismos lingüísticos universais" [Lions 1981] como, por exemplo, a estrutura frasal das diferentes formas de se referenciar um objeto nas linguagens naturais.

3. O processo de uso do *framework* proposto

A seção anterior procura apresentar as vantagens do uso de uma LNC na geração de LDEs. Ao mesmo tempo ela proporciona uma visão do perfil da LDE que esperamos obter de nosso *framework*, que será detalhado na seção seguinte. Nesta seção procuramos mostrar como funciona o processo de criação e uso de uma LDE a partir do *framework* proposto.

Visando obter um sistema modular, decidimos empregar uma sintaxe de controle, de referenciação a objetos e de metalinguagem fixa, como ocorre nas linguagens naturais, separando os elementos léxicos que são específicos do domínio. Esta decisão torna possível desenvolvermos um ambiente de programação composto de um *parser* e um processador para esta sintaxe antes mesmo dela ser instanciada pelo projetista da linguagem. Uma vez fornecido este ambiente, o trabalho do projetista de linguagens será resumido a instanciar o léxico específico da LDE por meio da definição de uma ontologia do domínio, e alimentar uma base de conhecimento do design da

aplicação com os objetos específicos da aplicação. O resultado deste processo será uma instância da linguagem-tipo, isto é, a LDE desejada. O *parser* e o processador da linguagem usarão a ontologia e a base de conhecimento definidas pelo designer da linguagem na resolução de referências a objetos (no caso do uso de anáforas e elipses), e na resolução de *loopings* nos programas (no caso do uso dos quantificadores "*all*", "*every*" e "*each*" ou de plurais). A Figura 1 abaixo ilustra este processo.

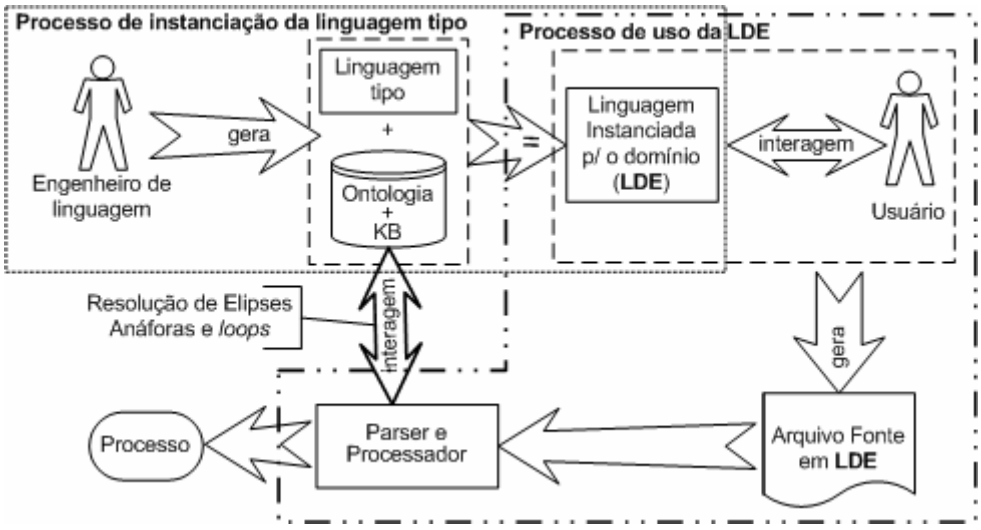

Figura1. Processo de instanciação da linguagem-tipo e uso da LDE gerada

É importante salientar que esta linguagem-tipo não é necessariamente única, podendo apresentar outros mecanismos de controle e mecanismos de referenciação mais ricos, conforme as necessidades do domínio em questão. Por exemplo, no caso de querermos utilizá-la como um mecanismo de criação de ontologias para *web* baseadas na linguagem de representação *OWL* [W3C 2004], é necessário que sua sintaxe seja alterada para aceitar as relações básicas previstas na *OWL*.

4. A linguagem-tipo proposta para instanciação de LDEs

Em [Da Silva 2001] identificamos que, de fato, a linguagem de planos usada pelas pessoas no dia-a-dia pode ser dividida em três, sendo composta por uma sublinguagem para os mecanismos de:

- **Referenciação a objetos**, que incorporam os mecanismos comunicativos anteriormente citados, atuando de forma ortogonal a outras sublinguagens, para aumentar a coesão textual e, portanto, facilitar a interpretação por parte do usuário final;
- **Controle**, que permitem a criação e definição do conhecimento procedimental e/ou do controle inferencial em alguns casos; e
- **Metalinguagem**, que permitem a criação e extensão da ontologia de um domínio por meio da definição de conhecimento declarativo.

Esta divisão é bastante interessante, sobretudo por tornar a linguagem modular, facilitando sua implementação, e será adotada em nosso *framework* como estrutura central na definição da linguagem-tipo. A seguir, vamos detalhar cada uma destas sublinguagens, procurando destacar suas características mais interessantes.

4.1. A sublinguagem de referenciação de objetos

Nossa análise em [Da Silva 2001] mostrou que, em geral, o elemento crítico das linguagens de programação atuais é o fraco mecanismo de referenciação a objetos por elas disponibilizado. Assim, definimos esta sublinguagem de forma a permitir a referenciação a objetos simples e compostos empregando uma gramática restrita para sentenças nominais, semelhante à encontrada nas linguagens naturais. Entre outras coisas, ela admite o uso de determinantes para os objetos e um conjunto restrito de **figuras de linguagem** (anáforas e elipses). Dentre os determinantes é possível especificar **quantificadores** (como "*all*", "*every*", "*each*" e "*an*"), **qualificadores** (pré-modificadores, frases preposicionais e adjetivos como, por exemplo, "*red*" *car*) e **seletores** (ordinais que atuam na escolha de um elemento de um conjunto, tais como "*first*", "*next*", etc.). A Tabela 1 apresenta um conjunto de exemplos da classe de referências possíveis.

Tabela 1: Exemplos de referências válidas na LNC proposta.

Tipo da referência	Exemplo
"noun" (um nome próprio)	"personal"
[***a***/***an***] noun	[***a***] message
noun(s)	Messages
cardinal noun(s)	7 messages
the/***this*** noun	***the*** message
a [***set*** | ***list*** | ***sequence***] ***of*** noun(s)	***A set of*** messages
the nounip ***of*** [***the***] noun	***the*** sender ***of the*** message
pronouns (restrito a ***it*** e ***them***)	***It belongs to*** … / … ***and*** copy ***them***
the ordinal noun (onde ordinal = ***next***/***last***/***first***/***previous***)	***the last*** message
all [***the***] noun(s)	***all the*** messages
each/***every*** noun	***each*** message

É importante observar que o uso de anáforas traz um ganho comunicativo significativo a esta sublinguagem, pois elas ampliam a **coesão textual** e reduzem o número de elementos a serem escritos e interpretados pelos usuários. A linguagem-tipo proposta aceita uma classe restrita de anáforas intra e inter-sentenciais. A principal restrição imposta está relacionada à classe de pronomes que podem ser empregados na criação destas anáforas. Em particular, como os domínios esperados para sua aplicação não denotam gênero, restringimos o uso de pronomes ao pronome neutro ("***it***") e seu plural ("***them***"), reduzindo, assim, a complexidade do algoritmo a ser utilizado.

Complementando o uso de anáforas, o uso de quantificadores associado a uma ontologia do domínio facilita em muito a expressão de *loopings* implícitos, que são os mais comumente usados na linguagem natural, reduzindo ainda mais a quantidade de texto a se escrever, quando comparado a outras linguagens de programação. A Figura 2 abaixo exemplifica este fenômeno, onde na sentença 1 (escrita usando a LNC proposta) está implicitamente especificado que um grupo (em um sistema de *email*) tem membros, os quais têm um endereço de *email*. Esta mesma especificação é apresentada na sentença 2 escrita na linguagem C, fazendo-se o uso de iteradores.

1	`send the message to the group.`
2	`for (i=first(group.member); last(group.member); next(group.member)` `  send(message, group.member[i].address);`

Figura 2. Comparação entre um texto em LDE (1) e um texto equivalente na linguagem C usando iteradores (2).

4.2. A sublinguagem de controle

Em [Da Silva 2001] chamamos a atenção para o fato de a forma natural de expressar ações em linguagens naturais ser bem diferente da forma usada em uma linguagem de programação. Como principais diferenças, podemos citar os fatos de os mecanismos de controle mais comumente empregados nas linguagens naturais serem expressos no estilo de uma linguagem de eventos ou de regras de produção e de as interações sobre múltiplos objetos serem comumente expressas implicitamente, através da operação sobre conjunto de objetos (por meio do uso de quantificadores nas próprias referencias).

Partindo destas observações, definimos uma sintaxe para esta sublinguagem que é ligeiramente diferente dos mecanismos semelhantes empregados nas linguagens de programação convencionais. Esta sintaxe emprega as estruturas encontradas no subconjunto das sentenças imperativas da linguagem natural que analisamos, tornando-a mais compreensível aos usuários. A Tabela 2 mostra alguns tipos de sentenças que são possíveis na sublinguagem de controle proposta.

Tabela 2: Tipos de sentenças permitidas na sublinguagem de controle

Sintaxe da sentença na linguagem-tipo	Exemplo
actions → action (‘**,**’ | ‘***and***’) actions ‘**.**’	replay to ***the*** message ***and*** mark ***it*** as read.
action → verb [preposition] object_ref [preposition object_ref [preposition object_ref [prepositon object_ref]]] ‘**.**’	send ***the*** message to ***all*** addresses from ***the*** receiver ***of the*** message.
action → (‘***if’*** | ‘***when’***) object_ref logic_expr ‘**,**’ actions ‘**.**’ [(‘***if not***’ | ‘***otherwise’***) ‘,’ actions ‘**.**’]	***if the*** group ***belongs to the*** address-book**,** send ***the***… . ***if not,*** show ***the*** message… **.**
action → ‘***repeat***’ actions (‘***until***’ | ‘***up to***’ | ‘while’) logic_expr ‘**.**’	***repeat*** ***if the*** color ***of*** folder ***is*** blue, go to ***its*** places ***and*** pick ***it*** up ***to the last***…
action → ‘***for’*** object_ref ‘**,**’ actions ‘**.**’	***for all*** important appointments ***of the*** agenda**,** set ***its*** day’s color to red.
action → ‘***from’*** object_ref (‘at’ | ‘***on’*** | ‘***to’*** | ‘***in’***) object_ref ‘**,**’ actions ‘**.**’	***from the first*** mail to ***the last***, change ***their*** status to ‘read’**.**

O primeiro tipo de sentença simplesmente mostra como fazer a aglutinação de ações. O segundo permite ao usuário utilizar uma ação previamente declarada no domínio e, portanto, diz respeito ao mecanismo de aplicação de funções nas linguagens de programação convencionais. Este tipo de ação, juntamente com a ação condicional—na forma de regras de produção—, descrita na terceira linha da tabela, e o uso de iteradores implícitos são as ações mais comumente utilizadas pelos usuários. Isto ocorre porque elas são mais naturalmente compreendidas.

No entanto, a linguagem também contempla os iteradores explícitos, que somente terão uso prático quando se desejar repetir um conjunto de ações sobre determinado objeto. Estes iteradores podem ser de três tipos: os iteradores do tipo “*repeat*”, que testam uma condição de repetição após realizar um conjunto de tarefas; os iteradores sobre estruturas, que operam como os iteradores implícitos, porém realizam

um conjunto de ações sobre os elementos da estrutura em vez de uma única ação; e os iteradores que operam sobre um intervalo de elementos permitindo, assim, limitar o número de elementos da estrutura que serão afetados pelo conjunto de ações.

Embora a gramática correspondente às expressões lógicas nas LDEs esteja intimamente ligada ao domínio, a linguagem-tipo proposta contempla um conjunto básico de expressões lógicas descrito na Tabela 3. Este conjunto é capaz de atender às necessidades básicas de muitos domínios, podendo ser ampliada pelo projetista da LDE quando necessário. Não apresentamos aqui nenhuma gramática especial para as expressões aritméticas, pois, em geral, estas variam com as necessidades do domínio.

Tabela 3: Gramática básica de expressões lógicas

logic_expr → simple_expr [rel_operator simple_expr]
simple_expr → [unary_operator] term {add_operator term}
term → factor {multi_operator factor}
factor → literal | object_ref
rel_operator → '***belongs to***' | '***is***' | '***are***' | '***is accepted***' | '***is bigger than***' | '***is smaller than***' | '***is bigger or equal than***' | '***is smaller or equal than***'
unary_operator → '***there is***' | '***there are***' | '***not***'
add_operator → '***and***'
multi_operator → '***or***'

4.3. A sublinguagem para os mecanismos de metalinguagem

Esta sublinguagem deve dispor de estruturas frasais que permitam a declaração de novas entidades ou o refinamento de entidades já existentes, admitindo, assim, a descrição de conhecimento declarativo de uma ontologia do domínio. Nossa intenção ao disponibilizar estes mecanismos foi a de possibilitar que até mesmo um usuário comum—que não seja o projetista da linguagem—possa estender o léxico da LDE, de uma forma tão natural de comunicação quanto possível.

Um tema relevante a ser avaliado, no caso da extensão de uma ontologia, é a possibilidade ou não da criação de entidades sem ligação direta com a ontologia existente. Se isto não for possível, é necessária a definição de um conjunto de entidades predefinidas que deverão fazer parte da linguagem, como acontece com os tipos primitivos nas linguagens de programação. A hipótese contrária permite ao usuário criar uma floresta de hierarquias dentro da ontologia, dificultado seu uso em um processo de inferência (consulta ou derivação de novos fatos). Nosso trabalho adota o princípio de que existirá um conjunto de entidades e relações das quais toda a ontologia deriva.

Outro tema relevante em relação ao refinamento de conhecimento é a possibilidade, ou não, de se revogar conhecimento já existente na ontologia ou na base de conhecimento. Ou seja, a ontologia ou a base terão crescimento monotônico ou não. Em geral, as ontologias são monotônicas, mantendo suas relações estáveis, mas as bases de conhecimento que as utilizam não são, possibilitando o apagamento de conhecimento caso este não seja mais válido. Este é um ponto importante, mas que independe da definição da linguagem e por isso, nesta fase do trabalho, estamos adotando a política de crescimento monotônico, que poderá ser revista em trabalhos futuros. Desta forma, o refinamento da base estará restrito à adição de atributos e relações entre as entidades.

Do ponto de vista ontológico, nossa linguagem-tipo aceitará que os usuários definam e manipulem os seguintes elementos: **entidades**—que representam as categorias existentes no domínio; **atributos**—que representam características destas entidades; **ações**—que representam as operações que uma entidade pode realizar no domínio; **partes**—que representam entidades que compõem (fazem parte ou estão agregadas a) uma outra entidade do domínio; e **relações**—que representam as inter-relações entre as entidades do domínio. Apesar da linguagem-tipo proposta poder aceitar qualquer tipo de relação, as relações de classificação ("*is a*") e partonomia ("*part of* / *has*") são predefinidas, por serem úteis na resolução de anáforas e elipses. Uma parte da gramática da classe de sentenças aceita pela sublinguagem para os mecanismos de metalinguagem é apresentada na Tabela 4. Como pode ser observado na tabela, a linguagem-tipo proposta aceita a definição de uma faixa de valores para os atributos, ou o seu tipo, e também da definição de valores *default*.

Tabela 4: Gramática para a classe de sentenças aceitas pela sublinguagem para os mecanismos de metalinguagem.

Object_Ref_1 ***'is'*** Object_ Ref _2 '**.**'
Object_Ref_1 '***is'*** Object_Ref_2 'that has' Object_Ref_3 '**.**'
Object_Ref_1 ('***is part of***' | '***are parts of***') Object_Ref_2 '**.**'
Object_Ref_1 '***has***' Object_Ref_2 '**.**'
Object_Ref_1 '***has***' Object_Ref_2 '**,**' '***with possible***' ('***value***' | '***values***') Object_Ref_3 '**.**'
Object_Ref_1 '***has***' Object_Ref_2 '**,**' '***which is***' Object_Ref_3 '**.**'
Object_Ref_1 '***has default value***' Object_Ref_2 '**.**'

4.4. Um exemplo de uso da LNC proposta

Para dar uma idéia da forma final de um código resultante de uma LDE instanciada a partir da linguagem-tipo proposta, usaremos como exemplo uma LNC para a criação de extensões em um sistema de *emailer* simples. A ontologia deste domínio, juntamente com as ações válidas para cada entidade, está apresentada na Figura 3. O código necessário para criar um *script* para enviar uma mensagem a um grupo de usuários neste sistema está apresentado na Figura 4. Este código contém somente elementos das sublinguagens de referenciação e controle.

É interessante observar que, com relação à especificação do conhecimento procedimental, a primeira vista não há a noção de variáveis nestes tipos de sentenças, conforme se vê na Figura 4. No entanto, em uma análise mais cuidadosa, podemos notar que algumas palavras referem-se a *types*—uma classe inteira de objetos—e, portanto, simbolizam variáveis como, por exemplo, "*message*" e "*group*" no código da Figura 4. Isto é possível devido ao uso de mecanismos anafóricos específicos. Deste modo, os nomes que denotam *types* e são precedidos por quantificadores representados por artigos indefinidos (ex. "*an email*"). Quando estes mesmos nomes aparecem no corpo da programação, eles geralmente denotarão um objeto único específico—um *token*—, e serão precedidos por um artigo definido (ex. "*the email*"). Esta é uma diferença crítica do ponto de vista comunicativo, por ocultar a necessidade da introdução de um novo tipo de objeto—as variáveis—, e mostra como a diferenciação *type*/*token* é sutil e sensível ao contexto neste tipo de linguagem.

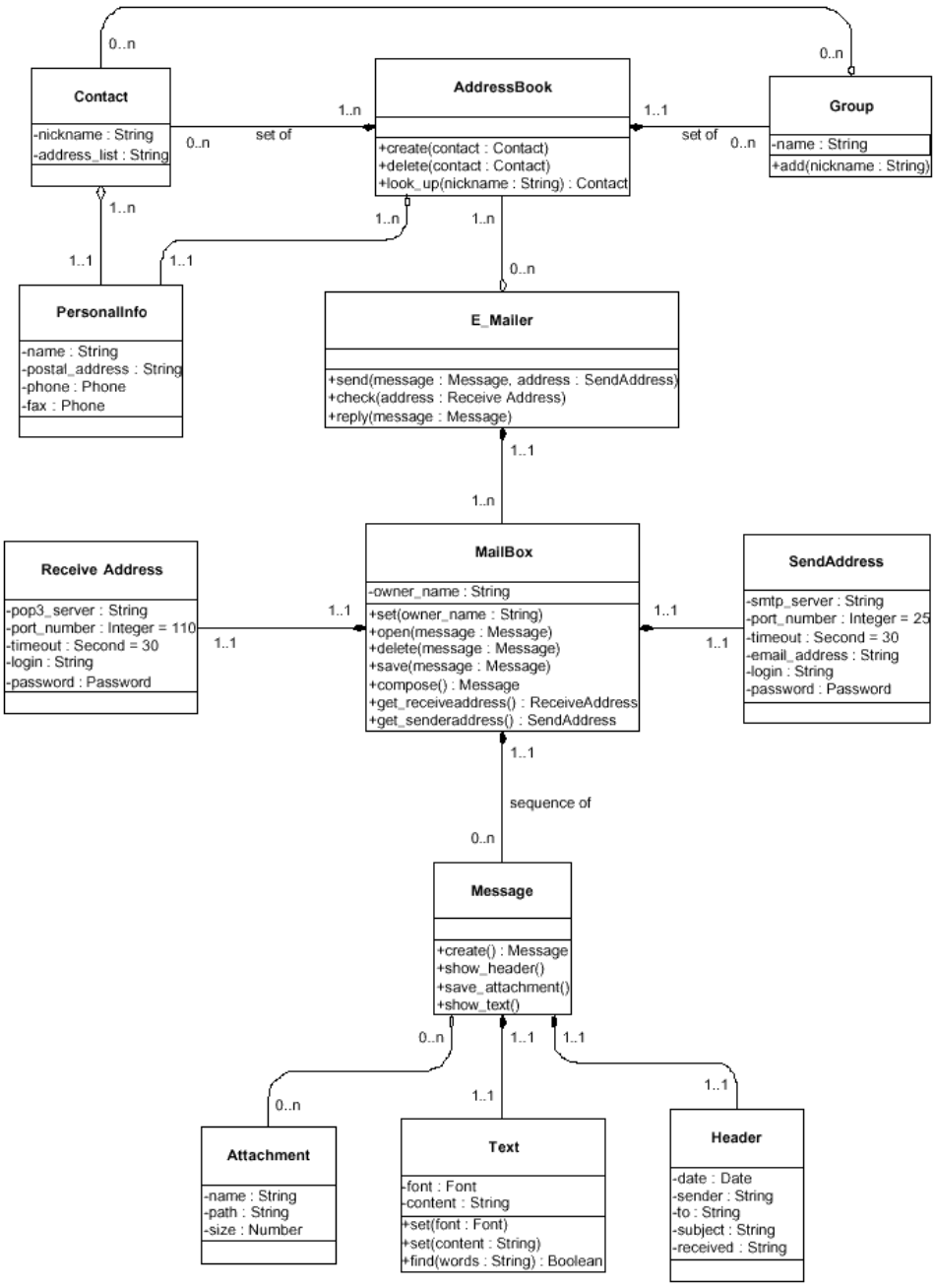

Figura 3: A ontologia de uma sistema de *emailer* simples.

Uma característica bastante conveniente que introduzimos na linguagem-tipo, mas que devido à limitação de espaço não será tratada mais profundamente neste artigo, é a possibilidade de definir elementos de interação com o usuário. Um exemplo deste mecanismo pode ser visto na segunda linha de código em que se indica que a função definida poderá ser ativada a partir do "*menu*" da aplicação. Outro exemplo é a definição do diálogo que será usado na aquisição de dados caso esta função seja ativada pela interface da aplicação, que ocorre na linha 3. Maiores detalhes sobre este mecanismo podem ser encontrado em [Da Silva 2001].

```
1  % Code for the creation of an action to send a message to a group.
2  To send a message to a group is an action of the agenda. It is
   activated by menu and follows these instructions:
3  When the message or the group are not indicated, ask for them
   using the get-data dialog.
4  If the group belongs to the address-book, send a message to all
   contacts of the group.
5  If not, show the message "The specified group does not exist.
   Please, check your data and try again.".
6  When there is an error, show the message "Some error has occurred
   during the sending of the messages. Please, check your data and
   try again." using the error-handling dialog.
```

Figura 4. Exemplo do código em LDE de uma *script* para enviar um *email* a um grupo de usuário.

5. A implementação do *framework* para a criação de LDEs

Adicionar elementos lingüísticos à sintaxe de uma linguagem de programação é simples, no entanto, é necessário que o processamento destes elementos seja eficiente. Nosso *framework* prevê o uso de uma sintaxe fixa para as sublinguagens de referenciação e controle. Isto permite que desenvolvamos um *parser* para este subconjunto de mecanismos que seja independente do léxico do domínio. O *parser* implementado usa a técnica de *top-down*, tendo sido empregado o mecanismo de *DCG*, presente na linguagem Prolog, para sua prototipação. A escolha pela linguagem Prolog se deve simplesmente à sua facilidade de uso na representação de conhecimento, o que nos possibilitou construir a ontologia e a base de conhecimento do design da aplicação na mesma linguagem.

O *parser* implementado funciona de forma convencional, recebendo como entrada um arquivo texto em LDE, passando este arquivo pelos analisadores léxico e sintático e gerando uma árvore sintática decorada com informações relevantes à resolução de referências. Esta árvore é entregue ao analisador semântico que trabalhará ligado diretamente à ontologia e a base de conhecimento, validando o léxico utilizado no texto, verificando o correto uso de referências a objetos da base e tornando explícitos os *loopings* implícitos especificados pelos determinantes das referencias. Para realizar estas validações, este analisador trabalhará em conjunto com um algoritmo de resolução de anáforas. Ao término da análise de cada sentença, o código resultante poderá ser executado para atualizar a base de conhecimento com novos elementos, ou simplesmente produzir alguma transformação nos elementos já existentes.

A resolução de anáforas é um elemento crucial do sistema proposto. Em geral, os algoritmos de resolução de anáforas em linguagem natural são complexos e caros computacionalmente. No entanto, devido às restrições sobre a classe de anáforas aceitas, conforme citado anteriormente, em nosso *framework* a resolução de anáforas fica bastante simplificada. Desta forma, inicialmente empregamos um algoritmo simples de *history list*, que mantém o histórico de todas as referências citadas no texto em uma pilha, sendo que a última entidade referenciada se encontra no topo. Assim, um pronome é resolvido pesquisando-se a pilha do topo para a base na procura de uma referência que unifique em número e gênero com o pronome encontrado. Esta é uma

abordagem bem simples, mas resolve a maioria dos pronomes que aparecem na sublinguagem de controle.

No entanto, ao introduzirmos a sublinguagem para os mecanismos de meta-linguagem verificamos que no caso de sentenças com cópula ("*is a*")—por exemplo, "*A car is a vehicle*"—, este tipo de abordagem não funcionaria. Isto acontece porque, ao aparecer no discurso, este tipo de sentença traz o foco do discurso para a entidade que está sendo definida ("*car*"), isto é, o foco torna-se a primeira entidade da sentença e não a última ("*vehicle*") como o algoritmo de *history list* resolveria. Se colocássemos estes elementos no algoritmo de *history list* na ordem em que eles aparecem no discurso com certeza o algoritmo de resolução de anáforas não funcionaria. Como solução, incrementamos a abordagem de resolução de anáforas com características do algoritmo de *centering* [Allen 1995], um algoritmo um pouco mais complexo que leva em conta as preferências estruturais lingüísticas. Deste modo, ao terminar de analisar uma sentença (i.e., ao encontrar um ponto final) as entidades do discurso são inseridas no *history list*, respeitando uma ordem que reflete as preferências estruturais lingüísticas. Assim, será inserido primeiro o sujeito, em seguida o objeto direto, depois o objeto indireto, e então as outras entidades do discurso presentes na sentença. Esta modificação é suficiente para tratar as anáforas do conjunto de sentenças aceitas pela linguagem-tipo proposta.

A implementação do algoritmo descrito é bastante eficiente, pois ao se usar uma pilha não há a necessidade de ordenação, implicando em um menor custo de manutenção da estrutura. Além disso, seu custo de pesquisa é baixo, pois, normalmente, ele termina nas primeiras referências consultadas, visto não fazer sentido, do ponto de vista lingüístico, usar um pronome para referenciar um substantivo que foi citado há muito tempo no texto.

Em [Pinheiro 2003], apresentamos resultados da implementação do *parser* e do processador para as sublinguagens de referenciação de objetos e para os mecanismos de metalinguagem. No momento, estamos concluindo a implementação da sublinguagem de controle e de um ambiente inteligente de programação para LDEs instanciadas a partir desta linguagem-tipo.

É interessante dizer que a ontologia e a base de conhecimento utilizadas estão baseadas em um sistema de *frames*. A escolha por este tipo de linguagem de representação de conhecimento interna se deve ao fato de os sistemas de *frames* permitirem a representação tanto do conhecimento declarativo quanto do conhecimento procedimental. Na base de conhecimento empregada em nossa implementação, uma entidade é representada como um *frame*, tendo pelo menos duas relações que determinam as hierarquias de classificação e composição, que são empregadas na explicitação dos *loopings*. Maiores detalhes da linguagem de representação de conhecimento empregada podem ser encontrados em [Da Silva 2001].

6. Discussão

Neste artigo propomos um *framework* baseado no uso de uma LNC modular como uma linguagem-tipo para a instanciação de LDEs. Mostramos que a divisão da LNC em três módulos: um para a tarefa de referenciação a objetos, um para a tarefa de controle e um para a tarefa de metalinguagem, proporciona uma estrutura bastante flexível para o *framework* proposto, permitindo a fácil instanciação de uma nova LDE. Além disso, mostramos que uma vantagem direta de usar uma LNC para programação está em

permitir que os usuários usem recursos de comunicação das linguagens naturais, como figuras de linguagem e determinantes, na especificação de programas. Tais mecanismos aumentam a coesão textual, reduzindo a quantidade de texto a ser escrita e interpretada, tornando a tarefa de programação um processo muito mais natural para programadores e até mesmo para *experts* no domínio, que não tenham conhecimento de programação.

Para alcançarmos eficiência de processamento das LDEs instanciadas a partir da linguagem-tipo proposta, restringimos o conjunto de mecanismos lingüísticos permitidos na linguagem. Deste modo, limitamos as anáforas ao uso dos pronomes "***it***" e "***them***", resolvendo-as por meio de um algoritmo baseado no algoritmo de *history list*, modificado pela adição de elementos do algoritmo de *centering*. Possibilitamos também o uso de um conjunto restrito de elipses, como a omissão de verbos e sujeitos dentro do conjunto de sentenças válidas, e o uso de quatificadores na definição de *loopings* implícitos. O funcionamento destes mecanismos requer a presença de uma ontologia do domínio e uma base de conhecimento do design da aplicação presente no ambiente de programação da LDE, o que não é um preço caro frente às vantagens que eles apresentam para o aprendizado e uso de uma LDE.

Este trabalho não é o primeiro a propor um *framework* para a criação de LDEs. O projeto *Sprint* [Concel 1998], por exemplo, propõe um *framework* baseado em um outro *framework* formal para a definição de linguagens de propósito geral, levando em conta os aspectos de arquitetura de software e propondo uma metodologia para o desenvolvimento de LDEs. No entanto, nossa proposta se diferencia pelo tratamento dos aspectos comunicativos necessário as LDEs, que não são abordados naquele projeto. Como procuramos demonstrar, tais aspectos são de extrema relevância quando se busca uma LDE que possa ser empregada por pessoas com menor conhecimento computacional como, geralmente, é o caso dos *experts* do domínio ou de usuários finais.

É importante salientar que ainda é necessária a realização de testes de usabilidade com a linguagem-tipo proposta para que possamos ter certeza de que ela é realmente de fácil aprendizagem. No entanto, é relevante lembrar que a usabilidade de qualquer linguagem de programação está intimamente ligada à qualidade da interface do ambiente de programação no qual ela está embutida. Deste modo, dentro do âmbito do projeto APEX [Da Silva 2002], estamos desenvolvendo um ambiente inteligente de programação no qual o usuário será orientado durante o processo de criação de seus programas, como se estivesse seguindo um roteiro.

A abordagem proposta neste artigo foi primeiramente utilizada no projeto de uma linguagem de extensão para usuários finais, visando valorizar, principalmente, os aspectos de aprendizagem e comunicação da linguagem [Da Silva 2001]. Ela também foi avaliada como um mecanismo de aquisição de conhecimento em [Pinheiro 2003]. Como continuação deste trabalho, pretendemos refazer a implementação do *parser* da linguagem-tipo proposta usando o mecanismo de *Discourse Representation Theory* (*DRT*) [Blackburn 1999]. Esta mudança na linguagem intermediária do *parser* nos dará condições de ampliar seu escopo de resolução de anáforas e elipses. Como subproduto do uso do mecanismo de *DRT*, será possível traduzir diretamente o código em LDE para a linguagem Prolog, tornando possível a simulação do código—mecanismo útil, por exemplo, no caso de empregar-se uma LDE como ferramenta de especificação de requisitos no processo de desenvolvimento de software. Outro ponto que queremos abordar é a mudança da linguagem de representação de conhecimento para o uso da

linguagem *OWL*. Esta linguagem apresenta um subconjunto com uma semântica bem definida sobre a lógica descritiva [Baader 2003], um formalismo muito empregado no processamento de linguagem natural, o que poderá ampliar ainda mais o escopo dos mecanismos lingüísticos empregados. Além disso, é necessário definir uma metodologia completa para a criação de LDEs que permita uma maior flexibilidade na definição dos mecanismos de referenciação, controle e metalinguagem.

Referências

Allen, J. (1995), Natural Language Understanding, 2nd Edition. The Benjamim/Cummings Publish Company, Inc., 1995.

Altwarg, R. (2000), "Controlled Languages: An Introduction", disponível em http://www.mri.mq.edu.au/ltg/slp803D/class/Altwarg/index.html, acesso em 09/06/03.

Blackburn, P. and Bos, J. (1999), "Representation and Inference for Natural Language", Vol. II – Working with Discourse Representation Structure, Universität des Saarlandes, September 1999.

CLAW (2003), Proceedings of the Fourth International Workshop on Controlled Language Applications, 2003.

CLAW (2000), Proceedings of the Third International Workshop on Controlled Language Applications, 2000.

CoLing (2002), Proccedings of the 19th International Conference on Computational Linguistics. Taipe, 2002.

Consel, C. and Marlet, R. (1998). Architecturing software using a methodology for language development. In Proceedings of the 10th International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP/ALP '98), pp. 170-194, Pisa, Italy, September 1998.

Da Silva, S.R.P. (2001), Um Modelo Semiótico para Programação por Usuários Finais, Tese de Doutorado, Departamento de Informática PUC-Rio, Rio de Janeiro, Maio 2001.

Da Silva, S.R.P. (2002), APEX — APlicações EXtensíveis por usuários finais. Projeto aprovado pela Fundação Araucária – PR, iniciado em Julho de 2002.

Deursen, A. v., Klint, P., Visser, J. (2000), "Domain-Specific Languages: An Annotated Bibliography", disponível em http://www.cwi.nl/arie, acesso em 29/01/2004.

Baader, F.; Calvanese, D.; McGuinness, D; Nardi, D. and Patel-Schneider, P. (2003), The Description Logic Handbook: Theory, Implementation and Applications. Cambridge Press. January, 2003.

Frechot, J. (2003), "Domain-Specific Languages—An Overview", disponível em http://compose.labri.fr/documentation/dsl/dsl_overview.php3, acesso em 09/02/2004.

Fuchs, N.E., Schwitter, R. and Schwertel, U. (1999), Attempto Controlled English (ACE)—Language Manual, Institut für Informatik der Universität of Zürich, August 1999.

Lyons, J. (1981), Language and Linguistics, Cambridge University Press. London, UK, 1981.

W3C (2004), Ontology Web Language Overview. Disponível em http://www.w3.org/TR/2003/PR-owl-features-20031215/, acesso em 16/02/04.

Pinheiro, J.M. (2003), Atualização de Bases de Conhecimento pelo Usuário Final, Trabalho de Graduação, TG-18-02, UEM, Maringá, Paraná, 2003.

Pulman, S.G. (1996), "Controlled Language and Knowledge Representation", In Proceedings of the First International Workshop on Controlled Language Applications, Katholieke Universiteit Leuven, Belgium, March 1996, pg. 233-242.

Sun (2004) Java Technology. Disponível em http://java.sun.com/ , acesso em 16/02/04.