Estratégias de reuso de *software* no TeCGraf

Waldemar Celes Filho Luiz Henrique de Figueiredo
Marcelo Gattass Roberto Ierusalimschy

TeCGraf–Grupo de Tecnologia em Computação Gráfica, I^{T}S, PUC-Rio
Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brasil
{celes,lhf,gattass,roberto}@icad.puc-rio.br

1 Introdução

Uma das principais preocupações da Engenharia de Software diz respeito ao reuso de código. Devido ao alto custo de desenvolvimento, faz-se necessário prover mecanismos que possibilitem o reuso de partes de programas anteriormente desenvolvidos e já testados.

O TeCGraf é um grupo interdisciplinar da PUC-Rio que atua apoiado por diversos parceiros industriais (notadamente o CENPES/PETROBRÁS) para desenvolvimento tecnológico em computação gráfica e interface com o usuário. Desde seu início, em 1985, o sucesso tecnológico do TeCGraf tem se apoiado primordialmente na reutilização de *software*. Uma parte de sua equipe está permanentemente voltada para a construção de bibliotecas de uso geral, e não para o desenvolvimento de aplicações finais. Além disto, estas bibliotecas têm um compromisso explícito com portabilidade. Entre as bibliotecas desenvolvidas pelo grupo, podemos citar a implementação do padrão gráfico GKS [8], o sistema de interface IUP/LED [1], e a *embedded language* Lua [2]. Estas bibliotecas rodam em diversas plataformas (e.g., Unix, DOS, e MS-Windows) e já foram utilizadas em produção na construção de dezenas de aplicações entregues à indústria.

Neste artigo, discutimos as estratégias de reuso adotadas pelo TeCGraf, com ênfase no projeto da linguagem Lua. Na próxima seção, relatamos a evolução das técnicas de reuso no TeCGraf. Em seguida, descrevemos o desenvolvimento da linguagem de configuração Lua, mostrando também as motivações por trás deste passo. Na seção 4, discutimos a arquitetura de várias aplicações desenvolvidas segundo esta linha. Finalmente, na última seção, apresentamos uma avaliação das estratégias adotadas com base na experiência de desenvolvimento interno e de terceiros.

2 Evolução de reuso no TeCGraf

2.1 Sistema gráfico

A primeira forma de reuso em grande escala no TeCGraf foi através da implementação GKS/puc do padrão gráfico GKS [8]. Um dos principais objetivos no projeto do GKS foi prover portabilidade às aplicações gráficas através do conceito de dispositivo virtual (*workstation*), implementado com *drivers*. Esta arquitetura (sistema gráfico padrão + *drivers*) permite duas formas de reuso, que se mostraram bastante produtivas. Por um lado, temos o reuso tradicional das rotinas gráficas, através de bibliotecas: qualquer aplicação que necessite serviços gráficos pode usar o sistema padrão. Por outro lado, toda aplicação gráfica desenvolvida sobre este padrão já nasce com uma alta portabilidade, que se traduz em reuso de código fonte da aplicação em diferentes plataformas (o TeCGraf dispõe atualmente de implementações do padrão GKS para dezenas de plataformas diferentes). Desde então, o TeCGraf tem usado a alta portabilidade de seus produtos como um fator constante de reutilização. Além disso, numa mesma plataforma, a aplicação tem acesso simultâneo a diversos dispositivos, o que caracteriza um reuso diferente de portabilidade. A padronização de uma mesma metáfora de sistema gráfico induziu ainda um grande reaproveitamento de código fonte entre aplicações. Os próprios cursos da área procuraram desenvolver "esqueletos" de aplicações que foram reaproveitados em diversos programas feitos no CENPES.

2.2 Sistema de menus

Apesar das facilidades gráficas do GKS, os seus mecanismos para construção de interfaces são precários. Por este motivo, o TeCGraf iniciou, em 1989, o desenvolvimento de um sistema de interface chamado IntGraf, visando suportar aplicações com maior grau de interatividade [4]. As interfaces criadas sobre o IntGraf eram compostas por menus hierárquicos, descritos através de uma linguagem de configuração auxiliar, e janelas de diálogos exclusivamente modais. O sistema já incorporava o conceito de programação orientada a eventos, não sendo portanto uma biblioteca no sentido mais tradicional, mas sim o que chamamos hoje em dia de *toolkit*. O IntGraf foi construído sobre o GKS/puc, herdando portanto sua portabilidade, o que levou sistemas de menu a ambientes sem suporte nativo para interfaces gráficas.

Dezenas de programas de teses e produtos para clientes foram desenvolvidas baseados em IntGraf e GKS/puc.

2.3 Sistema de interface

Com a disseminação de ambientes de janelas, os requisitos sobre a interface de programas tornaram-se bem mais críticos. Vários *toolkits* para construção de interfaces surgiram no mercado, como o Motif e o SDK do MS-Windows. Entretanto, nenhum padrão de fato emergiu. Para atacar as deficiências do IntGraf, mas preservando o seu "patrimônio" de portabilidade, o TeCGraf optou pela construção de um novo sistema de interface, denominado IUP/LED [1, 7]. Este sistema é composto por um *toolkit* virtual IUP, e por uma linguagem de configuração de diálogos, chamada LED. O *toolkit* IUP forma uma camada de abstração sobre o sistema de interface nativo de modo a assegurar portabilidade; neste sentido, o IUP é análogo a um sistema gráfico padrão, implementando sua abstração através de *drivers*. Para plataformas sem sistemas de interface nativos, como o MS-DOS, implementamos um sistema de janelas próprio, que dá suporte ao IUP. Este sistema de janelas é portátil, o que permite que aplicações desenvolvidas com IUP possam ter um *look-and-feel* fixo além do *look-and-feel* nativo (e.g., Motif e MS-Windows).

A linguagem de descrição de diálogos LED permite a configuração de parte da interface *pelo usuário* sem necessidade de recompilação da aplicação: especificações LED são convertidas para chamadas do *toolkit* IUP em tempo de execução. Assim, as aplicações desenvolvidas sobre IUP/LED são configuráveis, pelo menos no que diz respeito à interface com o usuário.

Atualmente, praticamente todas as aplicações desenvolvidas pelo TeCGraf usam como plataforma o sistema IUP/LED; a maioria das aplicações que usavam o sistema IntGraf estão atualmente migrando para o sistema IUP/LED.

2.4 Linguagens de configuração

Paralelamente ao desenvolvimento destes sistemas de interface, com amplo domínio de aplicação, havia uma crescente percepção de que outras formas de reuso poderiam ser usadas com sucesso em domínios mais específicos. Um exemplo desta situação foi o sistema de entrada de dados ED [3]. Em 1992, o CENPES encomendou ao TeCGraf vários programas para captura de dados. Devido à alta similaridade entre estas aplicações, o grupo responsável pelo projeto decidiu pela implementação de uma ferramenta genérica para captura de dados sobre desenhos. Os desenhos de fundo eram feitos com um editor gráfico, e os vários diálogos para entrada dos dados eram especificados em uma pequena linguagem especialmente projetada para este fim, chamada DEL (*Data Entry Language*). A intenção original do grupo era que o usuário final usasse a linguagem DEL para configurar as capturas de dados. Entretanto, logo percebeu-se que o nível de complexidade para este tipo de "programação" ainda era muito alto para usuários finais. Com isto, surgia a figura do *expert user*, um nome mais elegante para o que poderíamos chamar de *configurador* (veja a seção 4). Utilizando o sistema ED, era possível a construção (configuração) de um novo programa de entrada de dados em pouco tempo, sem programação convencional.

Outro exemplo importante de programação pelo usuário surgiu no projeto GMestre. Neste projeto, o próprio cliente encomendou uma "aplicação" configurável. O programa GMestre é um programa para visualização de perfis verticais para acompanhamento litológico, onde diferentes aspectos do perfil, como escalas, cores, *grids*, podem ser configurados. Era necessária alguma linguagem para descrição de um perfil. Entretanto, devido a este desejo do cliente por uma aplicação altamente configurável, a especificação do sistema era bastante fluida. Como abordagem, o grupo optou então por uma linguagem mais geral para descrição de objetos, denominada SOL. Esta linguagem foi fortemente influenciada pela linguagem de descrição de bibliografias do programa BibTeX [6]; um exemplo de sua sintaxe segue abaixo:

```
trilha1 = @trilha{ nome="Trilha 1", largura=50, grid=grid_default }
```

No início de 1993, o TeCGraf contava com várias linguagens de configuração para aplicações específicas, entre elas LED, para o sistema IUP/LED, DEL, para o programa ED, e SOL, para o programa GMestre.

3 O desenvolvimento de Lua

Assim que o sistema ED entrou em fase de testes, ficou claro que os mecanismos puramente declarativos de DEL não seriam suficientes para atender às aplicações encomendadas: tornou-se necessário incluir em DEL algumas construções procedurais. É importante notar que essas deficiências foram identificadas pelos usuários ao ver o potencial do sistema; a especificação inicial não previa construções procedurais. As extensões procedurais em DEL foram feitas mantendo a sintaxe declarativa, o que se mostrou pouco natural e complicado de usar.

O objetivo inicial no projeto de Lua era prover ao sistema ED uma linguagem procedural que tivesse uma sintaxe natural mas que fosse capaz de descrever os objetos de DEL de maneira simples; esta linguagem seria chamada pDEL. Neste mesmo momento (início de 1993), outras aplicações precisavam de linguagens de configuração; a principal delas era o programa GMestre, com sua linguagem descritiva SOL. Decidimos então combinar pDEL com SOL, e projetamos Lua para ser uma linguagem procedural com mecanismos para descrição de dados.

Dada a demanda crescente por programas configuráveis [2, 9], Lua foi criada para poder ser usada por qualquer aplicação que precise de uma linguagem de configuração. Por esta razão, Lua é extensível para dar suporte a diferentes domínios de aplicação. Assim, toda aplicação seria automaticamente configurável, de modo portátil, e com uma sintaxe uniforme, adaptável à aplicação.

A opção por desenvolver uma única linguagem de configuração traz os seguintes benefícios: simplifica o desenvolvimento, pois suprime a implementação de pequenos compiladores e interpretadores; evita a proliferação de linguagens incompatíveis; e simplifica o treinamento de clientes, pois todas as aplicações tornam-se configuráveis usando uma mesma base sintática.

A parte procedural de Lua é convencional; a parte descritiva usa a sintaxe de SOL. Entretanto, o poder de Lua está na combinação destes dois mecanismos: é possível associar funções a descritores de dados, criando assim *construtores de tipos* controlados pelo programador. No exemplo da seção anterior, `trilha` é uma função que é chamada após a construção da tabela que a variável `trilha1` vai receber.

Este mecanismo não convencional mostrou-se muito poderoso na prática. Os construtores de tipos podem ser usados para verificar consistência e validade dos campos, criar valores *default*, ou qualquer outro efeito colateral, como manutenção de estruturas de dados. Uma aplicação importante está na manutenção de configurações persistentes: Lua contém mecanismos reflexivos que permitem que o estado de uma configuração seja salvo usando código em Lua para criar código em Lua, que, quando executado, restaura o estado corrente [2].

4 Arquitetura de aplicações configuráveis

Krueger caracterizou diversos níveis de reuso de *software* sob os conceitos de abstração, seleção, especificação e integração [5]. A tarefa de reusar *software* está intimamente ligada ao conceito de abstração. A eficácia na reutilização de técnicas pode ser medida pela distância cognitiva para reutilizar, parcial ou integralmente, um trabalho anteriormente desenvolvido (informalmente, *distância cognitiva* é o esforço intelectual necessário para realizar uma tarefa). Assim, o reuso é facilitado se pudermos trabalhar com alto grau de abstração.

Tipicamente, a abstração em *software* pode ser dividida em três partes: a parte escondida, que engloba os detalhes de implementação e não deve ser vista pelo usuário; a parte fixa, que guarda as características invariantes da ferramenta; e a parte variável, que deve ser customizada para a aplicação. A especificação de ferramentas genéricas, que têm sido a ênfase recente no TeCGraf, consiste na parametrização da parte variável da abstração.

Portanto, aplicações complexas são melhor estruturadas se divididas em duas partes: a primeira é a *tecnologia da aplicação*, que é uma biblioteca de suporte, contendo a implementação dos serviços oferecidos pela aplicação, definindo um domínio de aplicação; a segunda corresponde ao mecanismo que oferece acesso programável aos serviços oferecidos [9]. Esta divisão desfaz a dicotomia tradicional “programador × usuário”. O acesso à tecnologia da aplicação pode ser programado por *expert users*: profissionais familiarizados com os conceitos do domínio da aplicação, mas não necessariamente programadores convencionais.

4.1 Implementação da arquitetura com Lua

A linguagem Lua contém os mecanismos necessários para prover acesso programável aos servicos contidos na tecnologia da aplicação. Além de uma sintaxe para seus próprios programas, Lua contém uma API (*application programmer interface*) que permite a comunicação entre código Lua e o código hospedeiro C. Deste modo, temos um código suporte que define a forma de acesso à tecnologia, customizando Lua para o domínio da aplicação. Este código suporte implementa a parte fixa da abstração e define quais parametrizações são válidas (Figura 1).

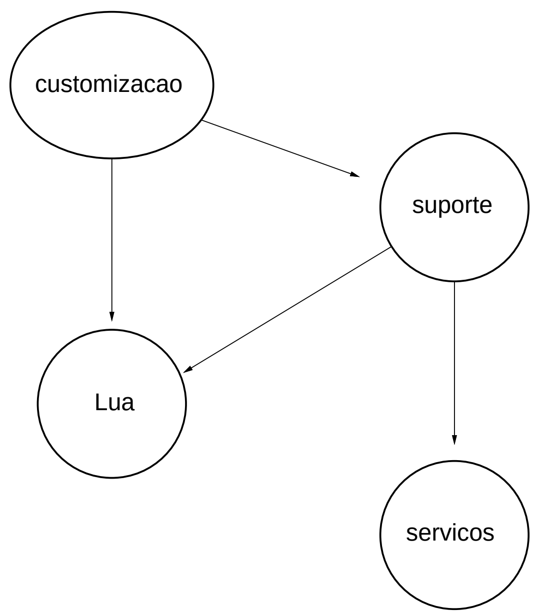

Figure 1: Arquitetura de aplicações configuráveis com Lua

Nesta arquitetura, o *expert user* tem acesso tanto às construções básicas de Lua, quanto às abstrações definidas no código suporte. Na realidade, esta mesma arquitetura pode ser estendida para incluir diversos níveis de *expert users*. Isto ocorre no programa GMestre de vizualização de perfis litológicos: temos um *expert user* definindo valores *default* para os objetos que podem compor perfis, e um outro definindo valores *default* para estes objetos.

4.2 A arquitetura de clientes de Lua

O diagrama da Figura 2 ilustra a arquitetura atual de clientes de Lua. No mais alto nível de abstração, o diagrama mostra os programas de captura de dados desenvolvidos sobre o sistema ED. Este sistema usa o código suporte iupLUA, que implementa uma abstração do *toolkit* IUP em Lua. No programa GMestre, criou-se um suporte em Lua para descrição dos objetos que compõem os perfis litológicos; iupLUA é usado como suporte do ambiente interativo para edição do perfil. O diagrama mostra, ainda, a arquitetura do programa MI, que implementa um ambiente para integração e edição de malhas de elementos finitos. Visando oferecer serviços de configuração, o MI permite que o usuário especifique os atributos associados à malha; para tanto, utiliza iupLUA para especificação dos parâmetros e da forma de captura destes atributos.

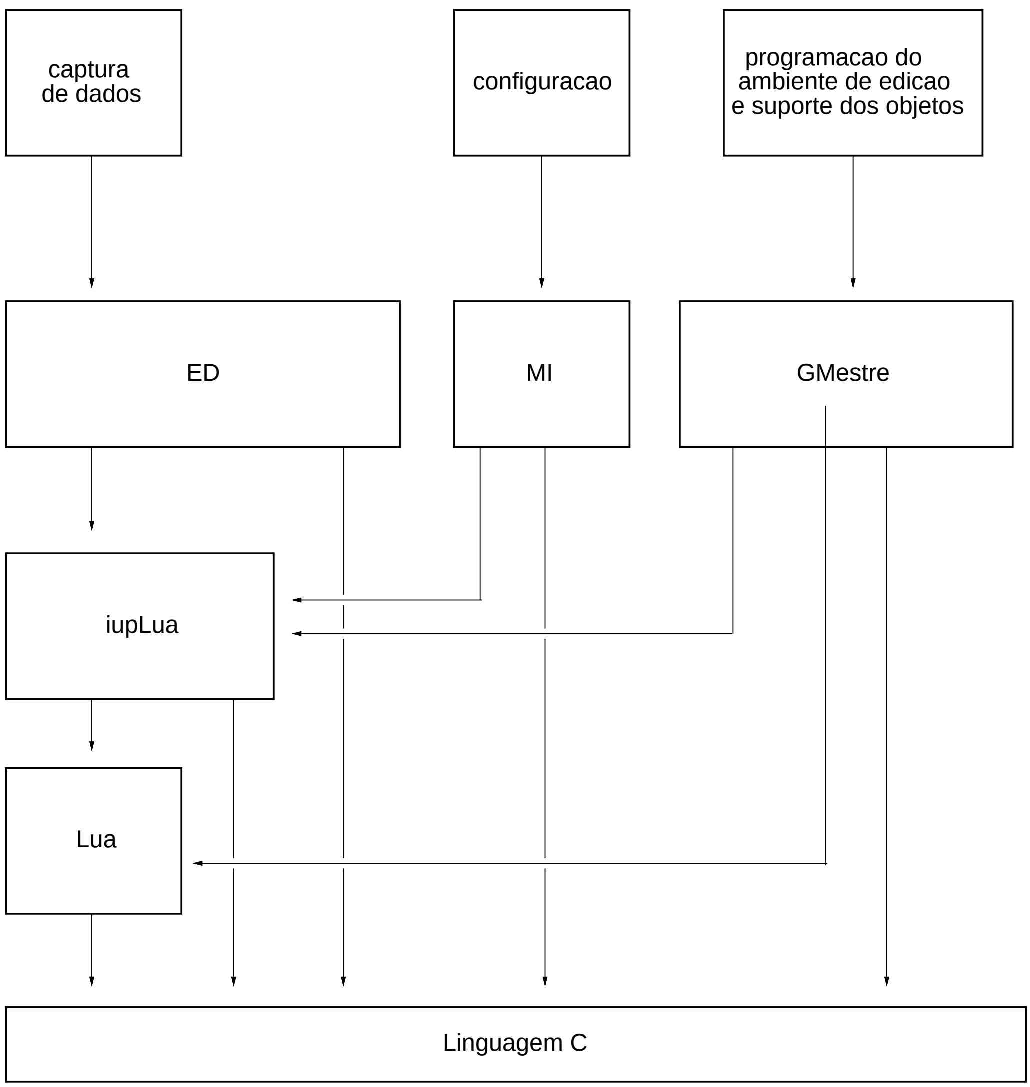

Figure 2: Arquitetura de clientes de Lua

5 Conclusões

Partindo de bibliotecas tradicionais, as formas de reuso adotadas pelo TeCGraf sofreram várias mudanças. O desenvolvimento das ferramentas apresentadas neste trabalho e a aderência a elas pelo grupo fundamentaram o desenvolvimento de aplicações gráficas interativas nos últimos anos.

Uma avaliação informal indica que o esforço de desenvolvimento de ferramentas, somados ao esforço de desenvolvimento das aplicações utilizando as ferramentas, ainda é menor que o esforço de desenvolvimento das aplicações sem as ferramentas. Esta observação justifica a existência da linha de desenvolvimento de ferramentas no TeCGraf.

Atualmente, estão disponíveis na Internet diversas ferramentas de domínio público para suporte a aplicações gráficas interativas. Entretanto, o escopo e os ambientes suportados por estas ferramentas são geralmente mais limitados que os desenvolvidos no TeCGraf, pois não há um compromisso global explícito com portabilidade.

Além disso, a existência de uma única linguagem de configuração encoraja a separação clara da tecnologia da aplicação dos problemas relativos a configuração. Como vimos, isto simplifica o desenvolvimento de aplicações configuráveis. A linguagem Lua tem sido largamente utilizada em produção para configuração pelo usuário, entrada de dados, descrição de interfaces, descrição de objetos da aplicação e armazenagem de *metafiles* gráficos estruturados. Estamos também considerando Lua com base para um sistema de programação visual.

Agradecimentos

As aplicações industriais mencionadas no texto estão sendo desenvolvidas em parceria com os centros de pesquisa da PETROBRÁS (CENPES) e da ELETROBRÁS (CEPEL).

Referências Bibliográficas

[1] L. H. Figueiredo, M. Gattass, and C. H. Levy. Uma estratégia de portabilidade para aplicações gráficas interativas. Em *VI SIBGRAPI*, páginas 203–211, 1993.

[2] L. H. Figueiredo, R. Ierusalimschy, and W. Celes Filho. The design and implementation of a language for extending applications. Em *XXI Semish*, 1994. (a ser publicado).

[3] L. H. Figueiredo, C. S. Souza, M. Gattass, and L. C. Coelho. Geração de interfaces para captura de dados sobre desenhos. Em *V SIBGRAPI*, páginas 169–175, 1992.

[4] R. Fischer. GeneSys: Sistema híbrido para modelagem de sólidos. Dissertação de mestrado, Dep. Informática, PUC-Rio, Rio de Janeiro, Brasil, 1991.

[5] C. W. Krueger. Software reuse. *ACM Computing Surveys*, páginas 131–183, 1992.

[6] L. Lamport. LaTeX*: A Document Preparation System.* Addison-Wesley, 1986.

[7] C. H. Levy. IUP/LED: Uma ferramenta portátil de interface com usuário. Dissertação de mestrado, Dep. Informática, PUC-Rio, Rio de Janeiro, Brasil, 1993.

[8] TeCGraf, PUC-Rio, Rio de Janeiro, Brasil. *Manual de Utilização do GKS/puc versão 3.0*, 1989.

[9] R. Valdés. Little languages, big questions. *Dr. Dobb's Journal*, 16(9):16–25, 1991.