Geração de interfaces para captura de dados sobre desenhos

Luiz Henrique de Figueiredo[1,2]
Clarisse Sieckenius de Souza[2]
Marcelo Gattass[2]
Luiz Cristovão Gomes Coelho[2]

[1]IMPA–Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina, 110
22460 Rio de Janeiro, RJ, Brasil
lhf@impa.br

[2]TeCGraf–Grupo de Tecnologia em Computação Gráfica
Departamento de Informática, PUC–Rio
Rua Marquês de São Vicente, 225
22453 Rio de Janeiro, RJ, Brasil
tecgraf@icad.puc-rio.br

Abstract. Input data for engineering programs that perform numerical simulation can be best prepared if the data entry process uses graphical interfaces based on engineering drawings. Two aspects of this process are considered here: the user interface language and the software tools required to implement it.

Introdução

A preparação de dados para programas de simulação e otimização em engenharia tem sido tradicionalmente feita usando editores de texto, com os quais o usuário cria arquivos texto contendo os valores necessários para o programa aplicativo funcionar. Este processo é tedioso e portanto sujeito a erros principalmente para programas que requerem muitos dados ou que definem uma formatação fixa para o arquivo de entrada. Muitos dos programas usados em produção hoje estão nesta categoria, tendo sido desenvolvidos ainda na época do cartão perfurado.

Uma solução comum para esse problema é a adoção de *formulários*: o usuário é apresentado a uma sequência de telas formatadas com nomes de campos e espa cos a serem preenchidos interativamente, em um terminal. A metáfora neste esquema são formulários reais, em papel.

Vários programas gerenciadores de banco de dados fornecem ferramentas que constroem formulários automaticamente a partir da especificação dos campos de um banco de dados. Estas ferramentas permitem a rápida produção de aplicações de entrada de dados no banco. Mais recentemente, foram propostas ferramentas análogas para preparação de dados para programas de engenharia, nos quais o volume de dados é grande e a formatação é fixa [Rajeev–Vaideeswaran–Krishnamoorthy (1989)].

Os dados a serem preparados correspondem a atributos de entidades abstratas da aplicação. Um formulário textual solicita ao usuário o valor de cada atributo usando um texto como *prompt*, possivelmente agrupando os atributos de uma mesma entidade numa mesma tela. Como consequência, o usuário não tem uma visão global das entidades envolvidas.

Frequentemente, entretanto, as entidades abstratas da aplicação podem ser descritas com um diagrama geométrico. Neste caso, a entrada de dados pode ser feita por manipulação direta da representação gráfica das entidades [Schneiderman (1983)]. Entende-se que esta representação gráfica, baseada nos desenhos comumente utilizados nas atividades de projeto, contém os símbolos mais naturais para a composição de uma linguagem usuário-computador.

O desenvolvimento de sistemas de interface com o usuário deve considerar de maneira cuidadosa duas interfaces importantes: a do usuário e a do programador de aplicação.

Na interface com o usuário, concentram-se as questões de produtividade (agilização), qualidade (verificação) e linguagem de representação. Estas questões são todas relativas à tarefa de um projetista (usuário) fornecendo dados para um programa.

Na interface com o programador de aplicação, a questão tratada diz respeito a qual o conhecimento exigido deste programador e quais as ferramentas disponíveis para ele desenvolver o programa que faz a interface com o usuário. Dada a natureza complexa das atividades de engenharia e geologia, espera-se que o programador de interface tenha conhecimentos da área onde seu programa deve atuar. Por isto, não é razoável exigir que esta pessoa seja profunda conhecedora de sistemas gráficos (GKS, PHIGS, Xlib) e *toolkits* de interface com o usuário (Motif, XView, Windows).

Procurando aprofundar o estudo dos problemas associados às duas interfaces mencionadas acima, este trabalho discute inicialmente os problemas de interface com o usuário. Nas seções seguintes, descrevemos e propomos um arquitetura para geração automática de ferramentas de entrada de dados por manipulação direta de diagramas geométricos.

Problemas de interface com o usuário

O processo de preparação de dados através de arquivos texto apresenta uma série de inconvenientes, dentre os quais destacam-se três de particular interesse:

- é tedioso;
- permite que o arquivo de entrada esteja incompleto ou contenha erros;
- a entrada parametrizada através de arquivos é uma representação dos objetos da aplicação muito distante dos esquemas gráficos preferencialmente utilizados por técnicos e engenheiros.

Estes inconvenientes podem ser formulados como problemas de:

- *agilização* da entrada (no tocante ao tédio);
- *verificação* da entrada (no tocante aos esquecimentos e enganos);
- *linguagem de representação* dos objetos conceituais da aplicação (no tocante à distância entre grupos de parâmetros, de um lado, e entidades e atributos, de outro).

Estes problemas são bastante conhecidos na área de interfaces entre usuários e sistemas; a solução clássica foi evoluir do processo de entrada de dados em *batch* para um processo interativo.

A solução interativa para interfaces de usuários traz consigo todos os desafios envolvidos no diálogo entre seres humanos e sistemas computacionais. Norman (1986) caracteriza este diálogo como uma travessia de dois golfos que separam o homem da máquina: o *golfo da execução* e o *golfo da avaliação*.

No primeiro caso, faz-se necessária uma ponte que ligue as metas que o usuário tem para realizar determinada tarefa com a representação física dos procedimentos executáveis pela aplicação no sentido de atender às intenções do usuário. Um exemplo típico ocorre em editores gráficos como o Canvas [Deneba (1986)]: A intenção do usuário é, digamos, traçar uma reta; esta é uma formulação mental, em uma da margens do golfo de execução. Na outra margem, está o procedimento necessário para realizar esta tarefa: o usuário deve pressionar o *mouse* sobre o ícone de uma reta na moldura da tela da aplicação e marcar dois pontos no espaço de desenho. Quanto mais óbvia é a relação entre o que é preciso fazer na interface e o que o usuário pretende realizar, mais estreito é o golfo, e mais fácil é a interação [Hutchins–Hollan–Norman (1986)].

No segundo caso, faz-se necessária uma ponte que ligue os sinais físicos perceptíveis nos dispositivos de saída da interface com uma interpretação sobre o sucesso ou insucesso da tentativa de interação. No exemplo acima, após ter marcado o primeiro ponto, o usuário percebe uma reta móvel (*rubberband*), com uma extremidade fixa no ponto inicial e a outra variando conforme o movimento do *mouse* sobre a tela. Quando o usuário marca o segundo ponto, a reta se fixa visualmente na tela. O golfo de avaliação desta etapa final no Canvas é do tamanho da distância entre a reta que o usuário vê na tela e a reta que o usuário tinha em mente. Nos editores gráficos do tipo WYSIWYG (*what-you-see-is-what-you-get*), esta distância é mínima; este é um argumento comum em favor de interfaces WYSIWYG.

Captura de dados sobre desenhos

É notável que as aplicações com interfaces interativas têm em seu protocolo de comunicação entre usuário e sistema—mais especificamente, na linguagem ou código desta comunicação—um ponto crítico de sucesso.

No caso em questão, a solução apresentada aqui é uma interface para captura de dados na qual se oferece ao usuário uma representação gráfica do objeto a ser manipulado pela aplicação. Neste diagrama estão explicitamente identificadas todas as entidades da aplicação e suas inter-relações, ainda que de forma esquemática. A cada entidade está associado um conjunto de parâmetros ou *atributos* que precisam ser especificados; estes atributos vão constituir exatamente o arquivo de entrada original.

Em contraste com formulários textuais, a requisição dos atributos de cada entidade não é feita sequencialmente, e sim voluntária e assincronamente pelo usuário. A interação necessária para preencher os atributos de uma entidade pode ser híbrida, contendo formulários textuais e outros protocolos gráficos.

Uma arquitetura em dois níveis

A aplicação de preenchimento precisa criar o diagrama que representa as entidades da aplicação final. Como mencionado na Introdução, o objetivo é a geração automática de aplicações de preenchimento. Assim sendo, a programação deste diagrama diretamente na aplicação não é uma alternativa, pois exigiria do programador de aplicação conhecimentos profundos de sistemas gráficos e de sistemas de interface com o usuário.

Descreveremos a seguir uma arquitetura que objetiva a geração automática de aplicações de preenchimento. Espera-se que esta arquitetura seja eficiente (i.e., aumente a produtividade) e simples (i.e., fácil de aprender e utilizar).

A criação de desenhos é feita com um editor gráfico genérico, cuja função básica é gerar uma descrição textual dos desenhos (*metafile*) que será lida e interpretada pela aplicação de preenchimento. A associação de desenhos do diagrama a entidades da aplicação final é feita agrupando primitivas gráficas e rotulando estes grupos com nomes de entidades (*tags*). Deste modo, o editor gráfico não precisa conhecer os atributos de cada entidade, mas somente os nomes das entidades.

As entidades e seus atributos estão descritos em um arquivo separado, chamado *template*, que é usado pela aplicação de preenchimento para controlar tanto a forma da interação quanto o formato do arquivo a ser gerado para a aplicação final. O *template* pode ser alterado sem consequências para o editor gráfico ou para a aplicação de preenchimento. Neste sentido, a aplicação de preenchimento é, na verdade, formada por um *template*, um ou mais *metafiles*, e um interpretador genérico de *templates* e *tagged metafiles* (veja Figura 1).

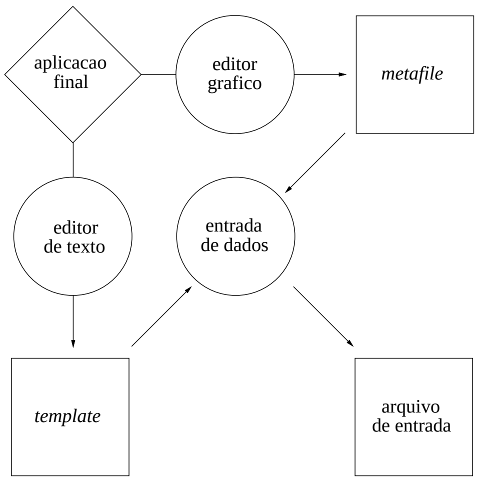

Figure 1: A arquitetura proposta.

Na arquitetura descrita acima, a criação do *template* é responsabilidade de alguém que entenda bem a aplicação final, de modo a poder identificar as entidades fundamentais. Idealmente, esta pessoa seria o programador responsável pela aplicação final, mas poderia ser um usuário experiente. A criação do diagrama que representa as entidades necessita não só de conhecimento das entidades abstratas da aplicação final, mas também de técnicas de programação visual e projeto de interface para que o diálogo seja bem definido e a interação seja efetiva (veja Figura 2).

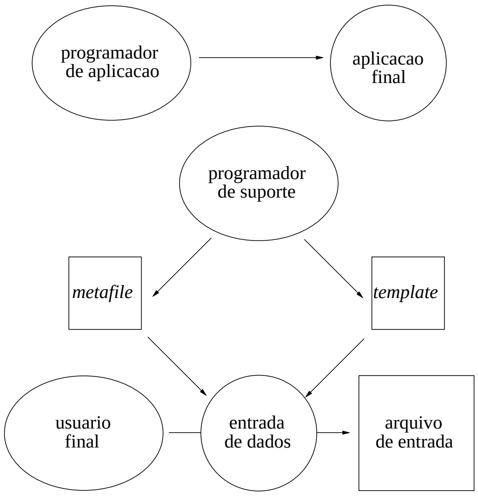

Figure 2: Agentes e ações.

Extensões

A arquitetura descrita na seção anterior é suficiente para interações simples, nas quais os atributos fornecidos pelo usuário não requerem verificação, nem dependem uns dos outros.

Para que a aplicação de preenchimento possa fazer verificação de validade, ou a validade de cada atributo está descrita na biblioteca de entidades, ou o usuário inclui código para fazer esta validade no momento da captura do atributo. Em ambos os casos, o esforço necessário à implementação da verificação de validade não é muito grande e pode ser feito ou pelo usuário ou por programadores pouco sofisticados.

Um problema mais sério ocorre quando os atributos são *acoplados*, isto é, os seus valores dependem uns dos outros. Neste caso, o esforço de programação pode ser substancial, requerendo sofisticação do programador para o projeto e implementação do diálogo. Uma solução possível para este problema é a avaliação de predicados associados a pré-condições para a aplicação final (veja a Discussão abaixo).

Uma implementação

A arquitetura proposta foi implementada. O desenho das primitivas é feito com um editor gráfico implementado sobre o GKS/puc, que é uma implementação GKS de nível 2b desenvolvida pelo TeCGraf (1990). A interação com o usuário é feita com o IntGraf, um *toolkit* também desenvoldido pelo TeCGraf, e que roda sobre o GKS/puc. O uso do GKS permitiu que a mesma implementação funcione em plataformas diversas, desde PC's a *workstations*.

Apesar destas escolhas, a arquitetura proposta independe do sistema gráfico ou de interface escolhido. Em particular, o *metafile* usado não é GKS nem CGM [ISO (1987)], pois estes padrões não possuem hierarquia para classificação das primitivas graficas.

O problema de verificação qualitativa foi tratado incluindo no *template* predicados associados a pré-condições. Estes predicados são avaliados pelo programa de entrada de dados e servem também para controlar a interação.

Exemplo de ***template***

Um exemplo típico de captura de dados é o programa tampo, desenvolvido no CENPES/Petrobrás, que faz a captura de dados para um programa de otimização do projeto mecânico do cabeçote flutuante de um permutador de calor [Matos (1990)]. A descrição abaixo corresponde a uma versão preliminar do programa tampo, que foi implementada usando a arquitetura proposta acima.

A captura baseia-se em um esquema da geometria do cabeçote em perfil, onde estão presentes todas as variáveis associadas a esta geometria. As variáveis que não possuem representação gráfica são associadas a ícones gráficos (veja Figura 3).

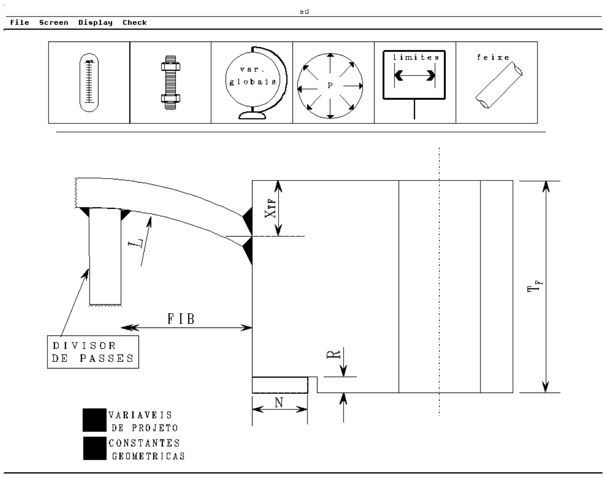

Figure 3: Diagrama geométrico do programa tampo.

As variáveis ou entidades que já foram visitadas e se encontram com valores válidos, aparecem marcadas com uma *bounding box* tracejada, como ilustra a Figura 4. Conforme sugerido anteriormente, a verificação de validade é feita avaliando predicados após cada captura.

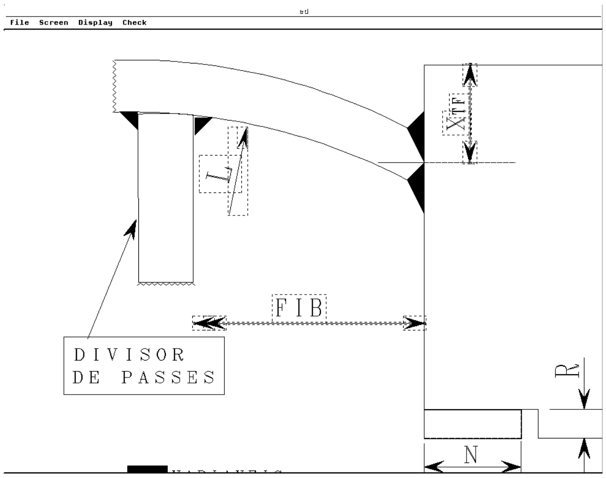

Figure 4: Visualização de uma entidade preenchida.

O trecho do *template* que descreve a captura de dados para a entidade gasket (características da junta) é o seguinte:

```
:e      gasket  "junta"
mat     s          # material
m       f       0  # fator m
y       f       0  # tensao de assentamento
t       i       1  # tipo de faceamento
:p
m>30
m<3000
y>335.8
y<2576.8
:t  "material"  "gasket"  "%43s %s %s"  42
mat
m
y
:f      "material novo"         mat="new"
m       "fator m                ="
y       "tensao de assentamento="
:m      "tipo"
"gasket.g"      GROUP_1a        GROUP_2
t               1               2
```

O rótulo :e identifica uma entidade gráfica selecionável de nome gasket, que possui as seguintes variáveis (atributos): mat de tipo string, m de tipo float, y de tipo float, e t de tipo integer.

O rótulo :p descreve os predicados associados às variáveis da entidade em questão. Neste exemplo, os predicados são simples comparações. Em geral, os predicados são expressões complexas, refletindo acoplamento com variáveis de outras entidades.

O rótulo :t aciona o método de captura relativo a tabelas. Nesta linha, o programador informa ainda o título da captura (material), que é combinado com o título da entidade e usado como título do *pop-up* no momento da captura, o nome do arquivo que contém a seleção (gasket), o formato de conversão de uma linha (%43s %s %s), e o número de colunas que o usuário quer visualizar (42).

O rótulo :f aciona o método de captura com formulários textuais. No exemplo acima, este método tem subtítulo material novo e só será acionado se o valor da variável mat for "new", ou seja, se o usuário tiver selecionado a linha com o conteúdo new na captura anterior. Vemos aqui um exemplo de predicados associados a métodos de captura de dados. Em contraste com os predicados de *validação*, que correspondem a pré-condições para a aplicação final, estes predicados são de *decisão*, e servem para controlar o fluxo da captura de dados (veja a Discussão a seguir).

Continuando o exemplo, se os métodos anteriores tiverem tido sucesso, é acionado o último método de captura, que é um menu gráfico identificado pelo rótulo :m. Na captura de dados por menu gráfico, o usuário escolhe uma dentre várias entidades gráficas apresentadas. Os valores atribuídos às variáveis associadas a este método dependem da entidade selecionada. No exemplo acima, se a entidade selecionada for GROUP_1a, então t recebe o valor 1; se a entidade selecionada for GROUP_2, então t recebe o valor 2. A Figura 5 ilustra a interação com o usuário para a captura do tipo de faceamento da junta, usando o menu gráfico representado pelo *metafile* gasket.g.

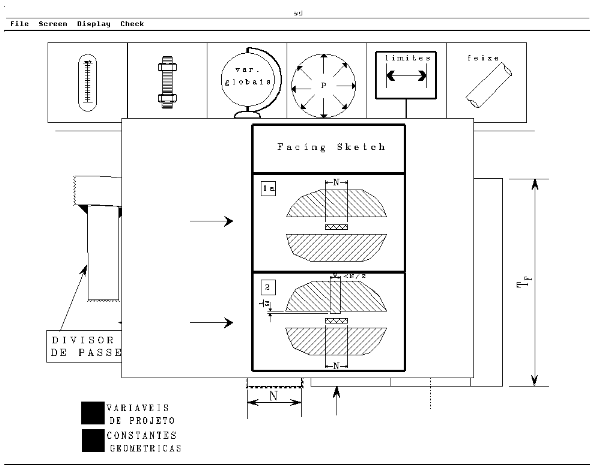

Figure 5: Captura com menu gráfico

O trecho apresentado acima é apenas uma pequena parte do *template* completo para o programa tampo. Um *template* típico, como o do tampo, contém mais rótulos que os discutidos no exemplo. A lista de rótulos disponíveis na programação de *templates* é parte de uma linguagem para captura de dados, cuja sintaxe e semântica estão completamente descritas em outro trabalho [Figueiredo (1992)].

Discussão

O valor de soluções interativas está em seu comportamento na relação custo/benefício envolvida nos fatores críticos que a motivam. No caso apresentado neste trabalho, os fatores de agilização, verificação e representação requerem inicialmente uma priorização de uns relativamente aos outros.

O problema de agilização é comumente medido em função do tempo gasto na solução *batch* versus o tempo gasto na solução interativa. Dificilmente o fator *tempo* é independente de outros, visto que a sua dissociação do fator *verificação*, por exemplo, pode levar a erros frequentes que terminam por anular os ganhos horários obtidos na atividade estrita de captura de dados. Da mesma forma, o fator *tempo* muitas vezes tem de ser desdobrado em *tempo de aprendizado e fixação* mais *tempo de execução*. Isto se explica facilmente pelos casos de interfaces extremamente ágeis (que oferecem *power keys*, ou macros) para os usuários que memorizaram todos os recursos disponíveis, mas bem menos ágeis e subutilizadas para aqueles que não tiveram tempo ou possibilidade de efetuar tal memorização. Neste sentido, verifica-se uma dependência entre o fator *tempo* e o fator *representação*, no sentido do estreitamento dos golfos mencionados por Norman (1986). Se a representação do que se pode fazer e do que foi feito é facilmente inteligível para o usuário, então o tempo de aprendizado é mínimo; caso contrário, claramente há um impacto negativo sobre a agilização da interface para usuários quaisquer.

Portanto, julgando-se prioritária a agilização, não se pode negligenciar aspectos relativos aos demais fatores. Neste sentido, a solução apresentada aqui oferece respostas satisfatórias para aspectos ligados à verificação e à representação dos dados.

No tocante à verificação, a aplicação estudada necessitaria de dois tipos de operação: uma verificação que chamamos de *quantitativa* e outra que chamamos de *qualitativa*. A verificação quantitativa é aquela que confere se todos os parâmetros necessários para a aplicação final foram fornecidos. A verificação qualitativa confere, adicionalmente, se os valores associados aos atributos satisfazem os predicados que definem as pré-condições necessárias para a aplicação final. Estas pré-condições podem ser simples testes de faixa de validade (*range checking*) ou predicados refletindo acoplamentos mais complexos entre os atributos.

O protótipo construído trata satisfatoriamente a verificação quantitativa, pois, quando é selecionada uma área do diagrama representando uma entidade, aparece junto a ela uma *caixa de diálogo* com todos os atributos desta entidade. Do mesmo modo, o protótipo implementa a verificação qualitativa por meio de predicados, conforme descrito na seção anterior. Os predicados são avaliados após cada interação com uma entidade, e antes de escrever o arquivo para a aplicação final. Isto garante que as pré-condições estão satisfeitas e que, portanto, o arquivo de dados gerado está correto.

A comparação com a entrada original em *batch* revela que a prática anterior era que os usuários utilizavam um arquivo de entrada de uma execução anterior da aplicação e o editavam com os dados referentes ao novo objeto da execução atual. Os perigos de edição de texto são bastante conhecidos—haja vista o problema corriqueiro de cartas escritas sobre outras cartas "para aproveitar cabeçalho", onde datas e por vezes mesmo os destinatários aparecem com nomes trocados por 'èsquecimento" de correção das especificidades do cabecalho aproveitado.

No tocante à representação, entre o arquivo texto contendo parâmetros linguisticamente codificados e a figura esquemática do objeto a ser manipulado (veja-se o exemplo do protótipo), não parece haver dúvidas de que a representação gráfica é intuitivamente superior. Contudo, não se trata apenas de representar o objeto, mas também de representar *a operação* sobre o objeto—no nosso caso, a captura de dados paramétricos.

O desafio de representação da interface interativa em questão é deixar claro para o usuário:

1. o que ele tem de fazer;
2. o que ele pode fazer e
3. o que ele já fez.

Os itens (1) e (3)—o a fazer e o feito—estão sendo tratados através de um código de visualização que associa às regiões ÿem que há dados a fornecer um código gráfico específico (tom forte de cor) e às regiões ÿem que os dados já estão fornecidos um outro código (tom esmaecido de cor). Nestes dois casos não-ambíguos, a figura presente na tela acusa claramente em que regiões ÿo usuário deve trabalhar ainda, bem como aquelas em que ele já não precisa trabalhar.

O item (2)—o que o usuário pode fazer—oferece-nos, entre outras, uma oportunidade interessante de resolução de interface. Trata-se do problema dos valores *default*: um dado *default* foi fornecido ou não? Sob certo aspecto, a resposta é positiva, no sentido que o sistema provê um valor aceitável e dispensa assim o usuário de explicitamente fornecê-lo. Sob outro aspecto, porém, a resposta é negativa, justamente porque o usuário de fato *não forneceu* nenhum valor. Para ilustrar em que medida isto pode ser um problema, pode-se voltar ao ponto do *esquecimento*, levantado relativamente à questão da verificação no início do trabalho. Se um usuário *não queria* aceitar um valor default provido pelo sistema, e a interface se restringe a mostrar o que há para fazer e o que já foi feito através do código tonal, então o tom associado a valores *default* é o de dado fornecido, o que tira a visibilidade de que naquela região há um valor *default* que é ao mesmo tempo fornecido e não-fornecido. Neste tipo de interação, na verdade, um usuário não tem visibilidade de valores *default*.

Para dar visibilidade a tais valores ambíguos, ao longo do eixo que vai do feito ao não-feito, o protótipo apresentado associa a valores *default* um código tonal intermediário, transportando para a linguagem cromática a semântica intermediária destes valores na linguagem computacional aplicativa. O usuário pode desfazer esta ambiguidade através de toques no *mouse*—um toque simples aceita o valor *default* como fornecido, e um toque duplo abre a caixa de diálogo para o fornecimento de outro (ou consulta ao *default* e subsequente aceitação). Após os toques no *mouse*, a tonalidade esmaecida correspondente a entrada aceitável aparece. Desta forma, uma entrada de dados completa é aquela em que todo o desenho que representa o objeto está colorido com a tonalidade esmaecida. No caso do protótipo apresentado na seção anterior, o código tonal foi substituído por *bounding boxes* tracejadas (veja Figura 4).

Conclusão

Discutimos os problemas de interface associados à captura de dados por meio de desenhos, que antes era feita com arquivos texto, por meio de editores.

Como solução de engenharia de *software*, propussemos uma arquitetura extensível, na qual é possível programar a aplicação de entrada de dados a partir de uma especificação da entrada da aplicação final e de um diagrama gráfico representando as entidades manipuladas por esta aplicação. Esta programação é feita usando uma linguagem simples, que é interpretada.

Implementamos um sistema baseado nesta arquitetura que é útil no desenvolvimento de programas de entrada de dados na Petrobrás. Observa-se que, à medida em que as evoluções dos sistemas gráficos vão sendo incorporados à implementação, o *look and feel* do sistema melhora significativamente, sem que se seja necessário alterar o editor gráfico ou o programa de entrada de dados.

Os trabalhos futuros de pesquisa poderão envolver a sofisticação da linguagem gráfica para a captura de dados em aplicações de engenharia ao longo de dois eixos distintos:

- representar mais operações (e.g., revisão/correção de dados fornecidos, adaptação do desenho)
- explorar a possibilidade de se projetar um gerador automático de interfaces desta natureza, utilizável por programadores de aplicação.

Agradecimentos

A motivação para este estudo teve origem em uma solicitação de Pedro Paulo Lopes de Matos, no âmbito do convênio TeCGraf–CENPES/Petrobrás. No dia-a-dia do TeCGraf, Waldemar Celes Filho acompanhou de perto o desenvolvimento do protótipo e fez inúmeras sugestões.

References

[Deneba (1986)] Deneba Systems Inc., *Canvas*, Miami, Florida, 1986.

[de Figueiredo (1992)] L. H. de Figueiredo, *DEL: a data entry language*, *preprint*, TeCGraf/ICAD, PUC–Rio, 1992.

[Hutchins–Hollan–Norman (1986)] E. L. Hutchins, J. D. Hollan e D. A. Norman, Direct Manipulation Interfaces, in: [Norman–Draper (1986)], 87–124.

[ISO (1987)] ISO–International Organization for Standardization, *Information processing systems – Computer graphics – Metafile for the storage and transfer of picture description information*, **ISO 8632-1**, 1987.

[Martin (1973)] J. Martin, *Design of Man-Computer Dialogues*, Prentice-Hall, 1973.

[Matos (1990)] P. P. L. de Matos, Otimização do projeto mecânico do cabeçote flutuante de um permutador de calor, *Proceedings of 6th Brazilian Symposium on Piping and Pressure Vessels*, 1990.

[Norman (1986)] D. A. Norman, Cognitive Engineering, in: [Norman–Draper (1986)], 31–61.

[Norman–Draper (1986)] D. A. Norman e S. W. Draper (eds), *User Centered System Design*, Hillsdale NJ, Lawrence Erlbaum and Associates, Publishers, 1986.

[Rajeev–Vaideeswaran–Krishnamoorthy (1989)] S. Rajeev, S. Vaideeswaran e C. S. Krishnamoorthy, Interactive data editor: a microcomputer user interface for batch programs, *Microcomputers in Civil Engineering* **4** (1989) 75–80.

[Schneiderman (1983)] B. Schneiderman, Direct manipulation: a step beyond programming languages, *IEEE Computer* **16** (1983) 57–69.