4
Interfaces para Provadores de Teoremas

Como vimos no primeiro capítulo, provadores de teoremas são usados atualmente em várias áreas de conhecimento, com destaque para o projeto de circuitos digitais, projeto de protocolos de comunicação em rede e na especificação e verificação formal de sistemas. Muitos autores, como por exemplo Bertot e Théry (BT98), esperam que este tipo de ferramenta possa se tornar um componente cada vez mais importante em outras áreas, onde a especificação formal de problemas seja viável. Até pouco tempo, porém, o uso de provadores de teoremas estava restrito a um pequeno grupo de especialistas, capaz de investir esforço para aprender o comportamento destes sistemas e suas complexas notações. Para que provadores de teoremas sejam usados por uma maior variedade de usuários, de matemáticos a desenvolvedores de sistemas, em suas atividades diárias, é necessário que estas ferramentas evoluam, tornem-se mais amigáveis e mais próximas desse novo tipo de usuário.

Como justificado no capítulo anterior, a prova de teoremas assistida por computador é uma atividade altamente interativa, baseada na cooperação entre usuário e máquina de prova. O diálogo entre usuário e sistema envolve a manipulação de elementos complexos que representam, por exemplo, fórmulas, estratégias e provas. A maneira como o usuário interage com o sistema pode ser decisiva em sua busca por uma prova para um possível teorema. Interfaces amigáveis, mais fáceis de usar e com os recursos apropriados, podem facilitar em muito esse processo e tornar a cooperação entre as duas partes mais eficiente.

Por outro lado, as interfaces existentes para os provadores atuais apresentam diversos tipos de limitações: geralmente, são interfaces baseadas em linha de comandos, diretamente acopladas ao provador, em ambientes, tipicamente, monousuário. As técnicas modernas de projeto e construção de interfaces com o usuário, bem como as tecnologias de implementação disponíveis atualmente, podem ser exploradas e aplicadas para evitar esses problemas e ajudar a projetar melhores ambientes de prova.

Neste capítulo discutiremos os princípios de design mais difundidos na construção de interfaces para provadores interativos de teoremas. Veremos

PUC-Rio - Certificação Digital Nº 0711291/CA

algumas das interfaces existentes e como elas diferem umas das outras. O funcionamento básico, genérico, destas ferramentas será identificado para servir de base na construção de nossa própria solução para o problema. As características da interação do usuário com a máquina de prova serão exploradas e será feita uma análise resumida dos requisitos fundamentais que tais interfaces devem implementar.

4.1 Premissa Básica: Separação Interface-Provador

A característica mais comum, encontrada nas interfaces para provadores interativos, sem dúvida, diz respeito a *separação entre a interface com o usuário e o provador de teoremas*. Até o início dos anos 90, interfaces para provadores eram sistemas fortemente acoplados à máquina de prova em si. Fossem sistemas completamente monolíticos, onde interface e provador constituíam um único produto, ou mesmo sistemas distintos, com a interface construída no topo da máquina de prova, mas ainda assim diretamente conectada à mesma.

Théry, Bertot e Kahn foram os primeiros a propor a completa separação entre provadores e interfaces em (TBK92). Segundo eles, construir interfaces apropriadas com o usuário e construir provadores de teoremas são atividades bem diferentes. Envolvem decisões de design distintas; ferramentas apropriadas para uma podem não ser apropriadas para a outra; lidam com domínios de problema diferentes. Em seu trabalho, eles destacam a ideia de se construir interfaces genéricas para provadores de teoremas, beneficiando-se dos conceitos de modularização e de reuso.

A proposta é construir a interface com o usuário como uma entidade separada, comunicando com uma máquina de prova através de algum tipo de *protocolo de comunicação* estabelecido entre os dois. Este modelo apresenta claras vantagens: a interface pode ser implementada com linguagens e ferramentas mais apropriadas para essa tarefa, independentemente daquelas usadas na construção do provador; interface e provador podem ser executados em processos ou máquinas distintas; uma interface pode suportar diversos provadores, permitindo que o usuário escolha o sistema que mais lhe interesse de acordo com a situação.

Essas ideias foram, aos poucos, sendo seguidas por outros autores no projeto de novas interfaces para provadores e hoje se constitui um princípio fundamental no design destes artefatos.

4.2

PUC-Rio - Certificação Digital Nº 0711291/CA

Representação de Provas

No capítulo anterior, apresentamos diversas características dos provadores de teoremas. Também os classificamos segundo diferentes critérios. Destacamos um tipo particular de provadores, aqueles que conduzem o processo de prova através de uma busca orientada a objetivos (conforme apresentado na seção 3.2). Concentraremos nossa discussão sobre o design de interfaces para provadores naqueles que funcionam desta maneira. Primeiro por ser a abordagem mais comum empregada nas ferramentas existentes e, segundo, por ser essa a forma de operar do provador que usamos como base no projeto de software que valida esta dissertação.

Para provadores que implementam a busca orientada a objetivo, a representação mais básica deste processo de prova é através de uma estrutura em árvore, chamada de *árvore de estado da prova* (*proof state tree*), ou simplesmente árvore de prova. Esta representação indica, a cada instante, o estágio corrente do processo de prova, mostrando a história de decomposição de uma conjectura (objetivo inicial) em subobjetivos até o momento atual com todos os subojetivos que ainda não foram provados. Para cada etapa de decomposição, a árvore de prova registra a regra que foi aplicada, no estilo *backward*, para obtê-la. Quando nenhum subobjetivo existir, isto é, quando os nós folha da árvore puderem ser provados diretamente, a prova está finalizada. A figura 4.1 exemplifica esse tipo de representação.

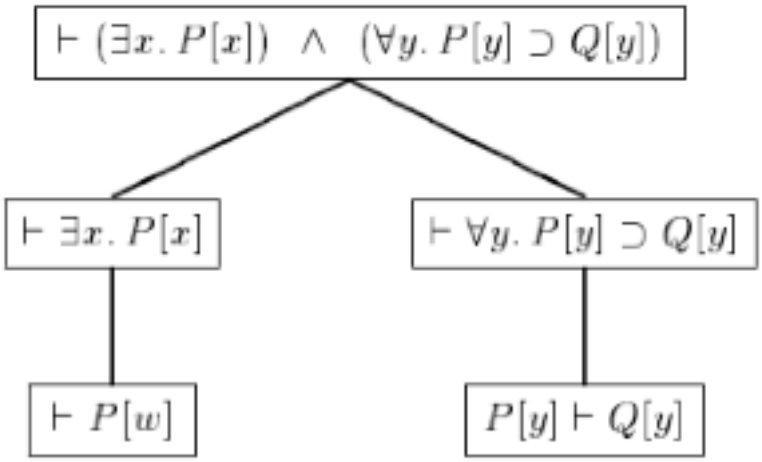

Figura 4.1: O processo de prova representado como uma árvore

A sequência de regras aplicadas de forma *backward* para produzir a prova pode ser submetida a provadores que trabalham no estilo *forward*, funcionando como uma espécie de *script* que conduz o provador ao resultado final.

PUC-Rio - Certificação Digital Nº 0711291/CA

Uma árvore de prova é um conceito abstrato, não necessariamente implementado internamente nos provadores (como é o caso de provadores LCF). Esta forma de ver o processo de prova está diretamente ligada a maneira como o usuário interage com o sistema, e é, portanto, uma responsabilidade das interfaces com o usuário lidar com esta forma de representação, considerando questões sobre como criar, modificar e inspecionar estes elementos.

4.3 Modelos de Interação Humano-Computador

Antes de aprofundarmos nos aspectos específicos sobre a construção de interfaces para provadores de teoremas, devemos explorar, mais detalhadamente, a forma como o usuário interage com provadores, na busca de um modelo conceitual adequado que possa ser usado como referência para projetar interfaces mais apropriadas. Estamos falando, justamente, do campo de estudo da área de IHC ou Interação Humano-Computador, isto é, a área que se dedica a estudar a interação entre pessoas e artefatos de base computacional. IHC é uma matéria interdisciplinar, que envolve conhecimentos de várias áreas diferentes como Computação, Psicologia, Sociologia, Semiótica, Ergonomia, Linguística e outras.

São poucas as referências científicas sobre a aplicação das técnicas de IHC no contexto específico de interfaces para provadores de teoremas. Aqui, seguiremos o modelo proposto em Aitken e outros em (AGMT98) para guiar nosso trabalho. Eles utilizam uma visão em diferentes níveis de abstração para descrever a forma como o usuário interage com um ambiente de prova. Esse tipo de abordagem já foi empregado em outras ocasiões para descrever a interação homem-máquina e constitui uma importante técnica de análise em IHC, mas foi pela primeira vez utilizado para o caso específico de prova de teoremas no trabalho citado.

O relacionamento interativo, cooperativo entre usuário e ambiente de prova pode ser visto em diferentes níveis de abstração. Uma mesma ação, durante o processo de prova, pode ser descrita de diferentes formas, por exemplo:

- Escolher uma regra de inferência para ser aplicada.
- Selecionar um item de uma lista.
- Clicar o botão do mouse com o cursor em uma determinada localização da tela.

Todas são descrições completas da mesma ação do usuário, mas cada uma focando um determinado modelo (conjunto de objetos de operações aplicáveis)

PUC-Rio - Certificação Digital Nº 0711291/CA

de atividade. Essas diferentes perspectivas da interação se relacionam de duas formas importantes:

- Cada nível pode ser visto como uma explicação do nível mais baixo. Usando nosso exemplo acima, temos que o usuário clica com o mouse em certa posição para selecionar um elemento de uma lista com o objetivo de escolher uma regra de inferência a ser aplicada. O nível mais alto, neste caso, serve como motivação para realização da atividade.
- Um determinado nível de abstração contém a representação escolhida pelo projetista para os objetos e operações no nível imediatamente acima deste. No nosso exemplo, se escolher uma regra a ser aplicada é uma ação do usuário a ser suportada, então selecionar um item de uma lista, é uma das possíveis maneiras de se fazer isso. Do mesmo modo, clicar o botão do mouse sobre certa posição é uma forma de efetivar a seleção de menu.

Os autores citados acima, propõem ainda que, no caso específico de interfaces para provadores de teoremas, é suficiente usar três níveis de abstração para caracterizar a interação:

Nível Lógico A descrição da interação apenas em termos dos conceitos lógicos.

Nível Abstrato de Interação Neste nível estão objetos e operações que descrevem como a informação é comunicada do usuário ao sistema e vice-versa. São objetos e operações visualizáveis, mas suficientemente abstraídos de detalhes de implementação física. Exemplos de objetos possíveis nesse nível são diagramas, textos estruturados, listas, etc.

Nível Concreto de Interação Neste nível estão as ações sobre dispositivos de entrada e as características perceptíveis dos objetos apresentados.

Cada nível de abstração é auto-contido, isto é, uma atividade pode ser totalmente descrita apenas em termos do vocabulário específico do nível em questão. Mas uma descrição completa de uma atividade pode incluir operações além daquelas associadas ao fluxo de informação entre os níveis. Estamos falando de aspectos cognitivos subentendidos na relação do usuário com o sistema, como por exemplo as motivações que o levam a escolher pela aplicação de uma determinada estratégia ao invés de outra no estado corrente da prova. Embora não diretamente representado no modelo sugerido acima, essas características merecem atenção e devem ser consideradas na análise da interação.

PUC-Rio - Certificação Digital Nº 0711291/CA

O design de interfaces para provadores tem explorado, ao longo dos anos, particularmente a relação entre o nível concreto de interação e o nível abstrato. Esta relação, chamada de *estilo ou técnica de interação*, preocupa-se com considerações como a representação visual de estruturas em árvore, técnicas de drag-and-drop, menus de contexto, etc.

A relação entre o nível abstrato de interação e o nível lógico, por outro lado, é menos explorada e pouco levada em consideração. Aitken e outros focam nesse relacionamento, chamado por eles de *visões*. Visões representam o que do domínio lógico será representado, como será representado e identifica as possíveis restrições presentes neste relacionamento. Como a ênfase é dada para a relação do nível lógico com o nível abstrato, e considerando o nível abstrato como um nível de representação compartilhado entre usuário e sistema, visões formam uma importante ferramenta de apoio na concepção de interfaces para provadores. Os autores catalogam as principais visões presentes nos ambientes de prova existentes até então. Ainda hoje, estas visões formam a base conceitual fundamental para analisar e projetar interfaces para provadores de teoremas.

Seguimos nas próximas seções explorando essas visões, porém, vale observar que elas não são as únicas possíveis quando pensamos nas formas de interação entre usuário e sistema em ambientes de prova de teoremas, entretanto são as mais comuns encontradas nas interfaces existentes atualmente. Além disso, uma interface não precisa se basear apenas em uma visão. Muitas interfaces para provadores atuais utilizam diferentes visões, buscando o que cada uma tem de melhor de acordo com as funções que o usuário precisa executar.

PUC-Rio - Certificação Digital Nº 0711291/CA

4.3.1 Prova como Programação (Proof as Programming)

A relação entre prova de teoremas e programação é, talvez, a característica mais explorada como base para concepção de ambientes de prova assistidos por computador. Discutimos um pouco sobre esta relação quando falamos de provadores LCF, na seção 3.2, do capítulo 3.

Essa relação fica evidente quando olhamos para o cenário da construção de provas, de maneira *forward*, usando provadores LCF, por exemplo. O usuário escreve funções na linguagem destes provadores, as táticas, que funcionam como algoritmos de tomada de decisão, que instruem o provador sobre a sequência a seguir para conseguir provar o teorema. Bem semelhante ao que acontece quando escrevemos programas de computador.

Mesmo quando empregamos uma busca de prova orientada a objetivos, com o método *backward*, uma vez que a prova tenha sido encontrada, temos

uma sequência de aplicação de regras que desempenha o mesmo papel descrito acima. O produto final do processo de prova é uma espécie de programa, submetido a uma espécie de compilador ou interpretador, a máquina de prova. Neste sentido, prova de teoremas pode ser vista como um tipo específico de programação.

Muitas das interfaces já desenvolvidas para provadores de teoremas utilizam fortemente essa correspondência. Em muitos casos as interfaces se parecem com verdadeiros IDEs (*Integrated Development Environments*) de programação. Um exemplo é a interface genérica (que suporta múltiplos provadores) Proof General (Asp03), construída sobre a plataforma Eclipse (Eclipse), um framework para construção de ambientes, tipicamente empregado na produção de ferramentas de apoio à tarefas de programação.

Portanto, a visão de prova como programação reorganiza, remodela o problema de provar teoremas, de sua origem no domínio da lógica, para um problema de programação. A figura 4.2 mostra uma possível configuração para o nosso modelo de níveis de abstração, levando em conta a visão de prova como programação.

PUC-Rio - Certificação Digital Nº 0711291/CA

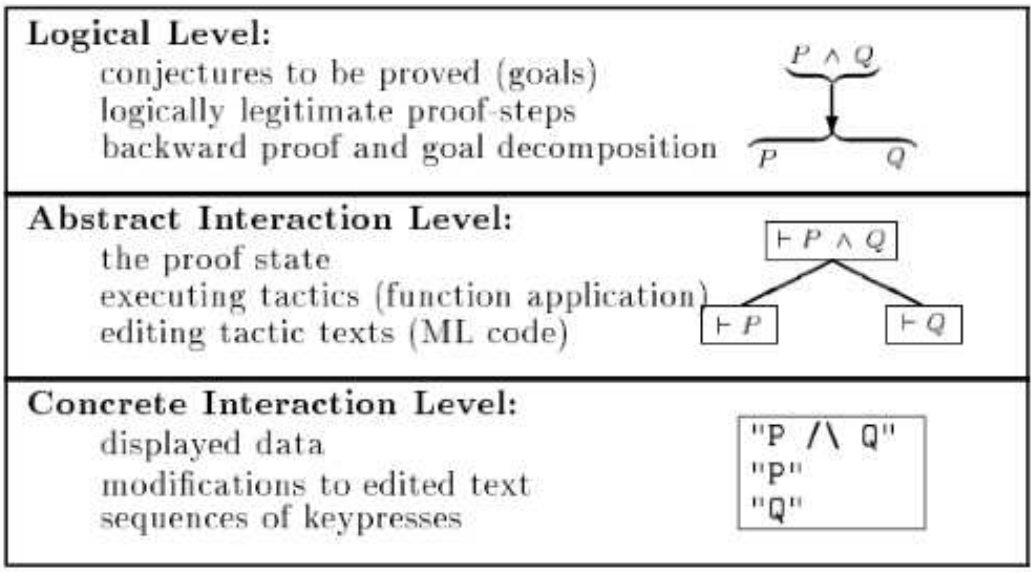

Figura 4.2: A visão de prova como programação

Apesar de tudo isso, como observado em (AGMT98), existem problemas com esse tipo de visão. Conceber que o usuário pense na construção de provas como uma atividade de programação, embora seja uma abordagem seguida em diversas implementações de interfaces para provadores, leva a interfaces projetadas para programadores, distanciando outros tipos de usuários do sistema. Usando as palavras dos próprios autores acima:

> *Programming is not a metaphor for proof construction; instead, programming is the medium through which proofs are constructed and expressed.*

Portanto, a questão que deve ser analisada é o quanto a visão de prova como programação explica o que realmente ocorre em um processo de prova e, o quanto interfaces projetadas segundo esta ótica aumentam, efetivamente, a produtividade na interação do usuário com o sistema.

4.3.2 Prova por Apontamento (Proof by Pointing)

Prova por apontamento ou *prova por seleção* é uma forma de enviar comandos da interface para o provador, apenas selecionando com o mouse uma subexpressão de um determinado objetivo (*goal*) em aberto, isto é, ainda não provado. A técnica de prova por apontamento foi proposta por Bertot, Kahn e Théry em (BKT94) e é uma característica comum em muitos provadores orientados a objetivo.

Os autores demonstraram, no trabalho em questão, que, no Cálculo de Sequentes, uma sequência adequada de regras de inferência pode ser obtida identificando uma subexpressão à esquerda ou à direita do símbolo de sequente ($\vdash$). Por exemplo, considerando a fórmula $P(a) \vee Q(a) \vdash R$, clicar com o mouse sobre $P(a)$, na subexpressão da fórmula à esquerda de $\vdash$, pode ser interpretado como um desejo de realizar uma análise de casos, devido ao fato de $P(a)$ ocorrer dentro de uma disjunção ($\vee$). Consequentemente, o estado da prova é alterado, introduzindo dois novos objetivos, um para cada caso da análise da disjunção. Os autores mostram que seguindo esse processo, sucessivamente, pode-se resolver todos os objetivos em aberto, finalizando a prova.

Em termos de interface, a ideia intuitiva na técnica de provar por apontamento, é que o ambiente de prova apresenta os objetivos correntes de um processo de prova em uma janela gráfica e o usuário pode guiar o processo simplesmente indicando as subexpressões desses objetivos que são importantes. O ambiente de prova, então, envia os comandos inferidos pela parte da fórmula selecionada ao provador, que traz essas subexpressões para o contexto atual.

Prova por apontamento não é a única maneira de permitir ao usuário interagir com a máquina de prova. É possível utilizar outras formas, como por exemplo, permitir que os usuários escrevam procedimentos de decisão ou estratégias. Um ambiente de prova pode, inclusive, fornecer os diferentes métodos como opção para o usuário.

Para lógicas simples, como a lógica proposicional, o método de prova por apontamento constitui uma técnica de prova completa, isto é, seguindo

PUC-Rio - Certificação Digital Nº 0711291/CA

esta abordagem o usuário conseguirá obter a prova procurada, caso exista. Para outras lógicas, incluindo aquelas da maioria dos provadores existentes (lógicas de mais alta ordem), a prova por apontamento só consegue substituir alguns comandos. Para outros comandos, formas diferentes de entrada são necessárias, como por exemplo, selecionar uma ação de uma lista ou digitar texto diretamente. A prova por apontamento pode ser estendida para associar ao clique a escolha de possíveis ações. Quando o usuário clica sobre uma posição, deve escolher também uma ação a ser executada que pode ser uma regra fundamental ou alguma estratégia mais complexa. Essa variação na técnica de prova por apontamento é conhecida como *apontar e atirar* (em inglês, *point and shoot*)

Considerando nosso modelo de níveis de abstração para interação entre usuário e ambiente de prova, temos no nível lógico, a mesma configuração usada na descrição da visão de prova como programação (ver figura 4.2). Entretanto, no nível abstrato de interação, não temos a edição de textos de táticas (que correspondem a programas), nem a execução destas táticas (novamente, semelhante ao funcionamento de programas). Temos apenas o estado da prova, transformado por ações de seleção, isto é, a escolha de uma operação a ser executada no objetivo corrente.

PUC-Rio - Certificação Digital Nº 0711291/CA

4.3.3 Proof como Edição de Estrutura (Proof as Structure Editing)

A Lógica Intuicionista é a Lógica da Matemática Construtiva, uma tradição matemática que defende que a demonstração da existência de certo objeto matemático deve ocorrer através da sua construção, isto é, uma demonstração construtiva fornece um algoritmo para se obter o objeto em questão. O já citado Isomorfismo de Curry-Howard nos dá uma conexão entre provas na Lógica Intuicionista e λ-termos tipados. Martin-Löf estendeu esses ideias, criando uma *teoria de tipos* fundamentada no construtivismo matemático. Na teoria de tipos construtiva de Martin-Löf , por exemplo, proposições são vistas como tipos, onde, por exemplo, um tipo A é uma proposição e um termo tipado $a :: A$ é visto como uma prova de A. Uma introdução à teria de tipos de Martin-Löf pode ser encontrada em (NPS90).

Esses conceitos formam a origem de uma visão no desenvolvimento de ambientes de prova onde a construção e exploração de provas consiste na edição de *objetos de prova*, usando algum tipo de *editor de estrutura*. Em provadores baseados nessa visão, por exemplo, o processo de provar uma determinada conjectura A consiste em construir um objeto de prova para A. O editor de provas ALF (MN94) é um representante desse tipo de visão. Nele, o estado da

prova possui dois componentes: um objetivo A, usando nosso exemplo anterior, e um objeto de prova incompleto, que representa o estado corrente do processo de provar A. Em tal modelo, o objeto de prova é construído por *manipulação direta*, isto é, por algum tipo de edição de estrutura. Um objeto de prova incompleto possui marcações (*placeholders*), indicando posições na estrutura para aquelas partes do objeto que ainda serão criadas. O processo de prova consiste em completar essas marcações com novos objetos até obter o objeto final, completo, representando a prova.

(AGMT98), a base para nosso modelo de níveis de abstração para a construção de interfaces para provadores, observa que na visão de prova como edição de estruturas, o nível abstrato de interação é organizado em torno de um estado de prova que inclui objetos de prova. A prova prossegue, então, pela edição da estrutura que representa tais objetos. Os autores suspeitam que esta visão leve a uma perspectiva diferente no nível lógico, mas não exploram em maiores detalhes esta ideia.

PUC-Rio - Certificação Digital Nº 0711291/CA

4.4 Componentes Básicos de Interfaces para Provadores

Até aqui, discutimos os conceitos fundamentais que envolvem o design e construção de interfaces para provadores de teoremas. Vimos o princípio fundamental de separação entre interface e máquina de prova, discutimos sobre a representação de provas e apresentamos algumas visões possíveis na interação entre usuário e ambiente de prova. A questão que surge é como efetivamente organizar um projeto de interfaces para provadores de acordo com estes conceitos. Qual a arquitetura mais adequada? Que componentes devem estar presentes?

Um dos primeiros trabalhos a tratar especificamente de uma arquitetura para o desenvolvimento de interfaces para provadores foi (BT98), já citado no começo deste capítulo. As ideias propostas pelos autores serviram de base para a construção de muitas das interfaces existentes atualmente para provadores de teoremas. A arquitetura proposta por eles foi resultado de uma série de pesquisas anteriores, onde estabeleceram os conceitos fundamentais sintetizados no trabalho em questão (ver (TBK92) e (BKT94)).

Nesta seção, exploramos os principais componentes desta arquitetura como base para o desenvolvimento de nossa própria solução. Uma breve discussão sobre estes componentes e suas principais características segue abaixo.

4.4.1

Protocolo de Comunicação

Para implementar nosso princípio fundamental de separação entre interface e máquina de prova precisamos de uma forma de permitir que estas diferentes partes de software se comuniquem. Interface e provador, são processos distintos, executando em uma mesma máquina ou mesmo em máquinas diferentes. A arquitetura proposta em (BT98) se baseia nos modelos então vigentes para estruturar sistemas distribuídos de software. Ou seja, a arquitetura para ambientes de prova como estes pode ser vista como uma coleção de componentes independentes, conectados por algum meio, comunicando através de troca de mensagens. As mensagens possíveis entre as partes envolvidas formam o *protocolo de comunicação* do sistema.

Um aspecto importante na implementação desse protocolo é como tratar os dados estruturados. Interface e provador possuem suas próprias representações de objetos tais como fórmulas, provas, sequentes, etc. Os resultados enviados do provador para a interface precisam ser, de alguma maneira, traduzidos para a representação interna da interface para que esta possa apresentar os resultados para o usuário. É necessário que o provador anote os resultados enviados à interface com informações extras, que possam ser usadas por ela para traduzir a informação para sua forma de representação. A interface também precisa converter os comandos entrados pelo usuário de forma que o provador possa compreender que operações precisa realizar. Além disso, o provador também precisa enviar algumas informações adicionais para a interface que não serão usadas para apresentação, mas sim para que a interface envie de volta ao provador em diálogos posteriores.

A grande vantagem desse modelo é promover a modularização, criando componentes fracamente acoplados através de um conjunto de operações explícito. Isto permite que cada uma das partes possa ser desenvolvida e evoluída com pouco impacto na outra. Linguagens e ferramentas apropriadas podem ser empregadas na interface enquanto outras são utilizadas no desenvolvimento do provador. Além disso, é possível, neste cenário, permitir que uma mesma interface suporte diferentes provadores, bastando cada um destes provadores conhecerem e se adequarem ao protocolo estabelecido.

O tamanho do conteúdo das mensagens trocadas deve ser considerado. Como estamos falando de comunicação interprocessos, minimizar o número de mensagens e seu tamanho é uma questão importante. Deve ser avaliado a necessidade de implementação de alguma forma de incrementalidade, isto é, de que, entre comunicações sucessivas, apenas as diferenças sejam transmitidas e não todo conteúdo novamente.

O padrão arquitetural conhecido como *Broker* (ver (BMRSS96)) estende

PUC-Rio - Certificação Digital Nº 0711291/CA

o modelo de arquitetura acima e é visto, atualmente, como um modelo apropriado para a construção de interfaces desacopladas de provadores. A principal diferença para a proposta original de Bertot e Théry é a presença de um mediador central, o Broker, para intermediar a comunicação entre as partes. Este modelo é seguido, por exemplo, no protocolo implementado pela interface Proof General, já citada acima.

É comum descrever protocolos de comunicação deste tipo em alguma linguagem independente da interface e do provador. É o correspondente à IDL (*Interface Description Language*) de ambientes baseados no padrão CORBA (Corba) e aos arquivos WSDL (*Web Service Description Language*) usados para descrição de web services.

Também é preciso levar em conta se o protocolo armazenará estado ou não e se a troca de mensagens funcionará de maneira síncrona ou assíncrona. São aspectos do design do protocolo que devem ser analisados de acordo com as ferramentas disponíveis e das características desejadas no ambiente de prova. Por exemplo, ambientes assíncronos podem ser mais fáceis de construir de forma geral, mas podem ser mais difíceis de implementar quando se deseja dotar o sistema com algum tipo de função de interrupção.

PUC-Rio - Certificação Digital Nº 0711291/CA

4.4.2 Apresentação e Manipulação

As funções principais de uma interface entre usuários e sistemas computacionais são: exibir os resultados dos processamentos do sistema computacional em um formato adequado para entendimento do usuário e permitir que este usuário possa interagir com o sistema enviando comandos e informações. Discutiremos nesta seção estes dois aspectos como responsabilidade do componente de apresentação e manipulação da interface de prova.

Quando pensamos em exibir informações em interfaces para provadores de teoremas, um primeiro problema a ser considerado está em como tratar as *convensões tipográficas* características da prova formal. Fórmulas contém símbolos especiais como, por exemplo, os quantificadores $\forall$ e $\exists$, conectivos como $\rightarrow$ e expressões matemáticas como $\sqrt[3]{a}^{\sqrt{b}}$. Ao projetar tais interfaces, o projetista deve considerar o uso de um sistema de codificação de caracteres que suporte os símbolos em questão. A interface deve não apenas exibir estes símbolos corretamente, mas também fornecer meios para o usuário do sistema informá-los durante os ciclos de interações.

Normalmente, a interface possui sua própria implementação dos objetos presentes no processo de prova (fórmulas, regras, árvores de prova, objetivos em aberto). Esses objetos são manipulados internamente para refletir o estado

atual da prova de acordo com os comando enviados e respostas obtidas da máquina de prova. Este estado interno (conjunto de objetos existentes em determinada instante) é apresentado de alguma forma gráfica para o usuário pela interface. A escolha de correspondentes visuais adequados para representá-lo é uma tarefa importante no design destas interfaces. Algum mecanismo de sincronização é necessário, de forma que ações do usuário sobre determinado objeto visual sejam reproduzidas pela interface no objeto interno correspondente.

Na exibição da árvore de prova, a interface precisa utilizar algum tipo de processamento de *layout* para fornecer uma melhor apresentação visual da prova para o usuário do sistema. O problema de arranjar visualmente figuras consiste em uma área de pesquisa e desenvolvimento própria, com suas próprias técnicas e desafios. Assim, uma prática comum é a utilização de componentes já existentes, criados para este fim e que possam ser adaptados para trabalhar em conjunto com a interface de prova. Isto permite concentrar os esforços em soluções específicas para a interface de prova. Em (BT98), os autores utilizam esta técnica no projeto da interface CtCoq. Por outro lado, aplicar funções de *layout* após cada novo estado da prova pode levar a problemas de performance, tornando a interface mais lenta cada vez que o processo de prova avança. Alguma solução para aplicação incremental das funções de *layout* se torna necessário, evitando que partes não alteradas da prova sejam refeitas a todo momento.

Recursos para facilitar a navegação do usuário pela árvore de prova são importantes. Provas muito grandes podem não caber de forma totalmente visível na área de tela disponível, tornando tais recursos indispensáveis. Poder colapsar e expandir partes da prova para facilitar a visualização, localizar trechos da prova (exibir o trecho em destaque na tela) a partir de alguma informação de busca, identificar visualmente os objetivos em aberto e as partes já finalizadas da prova (por exemplo, através de cores diferentes), funções de aproximação e afastamento (*zoom in* e *zoom out*) são algumas das capacidades desejadas para facilitar a visualização de provas.

Fórmulas muito extensas também são problemáticas. Durante o processo de prova, subobjetivos podem crescer consideravelmente, ultrapassando os limites da tela disponível. Recursos para que o usuário possa rolar pelo texto da fórmula, o uso de reticências para indicar fórmulas comprimidas, a possibilidade de comprimir e expandir o texto de fórmulas, entre outras facilidades podem ajudar a lidar com esta questão.

Cada sistema dedutivo possui uma forma de visualização típica, já consagrada na literatura científica. É interessante que as interfaces para

PUC-Rio - Certificação Digital Nº 0711291/CA

provadores possibilitem que uma prova seja visualizada em diferentes formatos, de acordo com o sistema dedutivo usado. Por exemplo, em dedução natural, além da forma tradicional de visualização da prova como uma árvore, o chamado estilo Gentzen de apresentação, temos também a visualização sob a forma de caixas, conhecido como estilo Fitch (Fit52).

As visões que discutimos acima influenciam diretamente os recursos de edição e manipulação de prova disponíveis na interface. A visão de prova como programação implica em editores de scripts de prova com suporte a *syntax highlight* [1], *code completion* [2], entre outros recursos, tipicamente presentes em ambientes de programação. Nesta visão, as fórmulas e a prova em si, são entradas textualmente e a interface precisa reproduzir a gramática do provador para garantir a construção de scripts sintaticamente corretos. A interface pode, também, enviar os textos entrados através dela para o provador validar. É preciso considerar as questões de performance e redundância do código de validação, para escolher a opção mais apropriada. Na visão de prova como edição de estrutura, a interface faz intenso uso da manipulação direta dos objetos de prova, isto é, *wizards* [3] para criação estruturada de fórmulas estão disponíveis, através de menus de contexto, que guiam e limitam a construção de objetos sintaticamente corretos. Wizards orientam também a construção da prova, exibindo seu estado atual e permitindo manipular com menus de contexto os espaços ainda em aberto. A visão de prova por apontamento sugere a possibilidade de indicar a sequência de prova através do clique com mouse em partes da fórmula do objetivo selecionado. A interface infere as regras aplicáveis pelo contexto selecionado e envia os comandos para o provador, evoluindo um passo no processo de prova.

PUC-Rio - Certificação Digital Nº 0711291/CA

4.4.3 Gerenciamento de Script de Prova

Um *script de prova* corresponde a uma sequência de comandos enviados ao provador para direcioná-lo na busca por uma prova. O componente de gerenciamento de scripts é responsável por fornecer recursos na interface para facilitar a criação, edição, armazenamento e execução destes scripts.

Nas ferramentas atuais, scripts costumam ser executados interativamente, onde cada linha (comando) do script de prova é enviado ao provador, mudando seu estado. Ou seja, cada linha do script leva diretamente o prova-

[1] Característica que destaca em um texto, usando cores e fontes diferentes, os termos de uma linguagem de acordo com a categoria semântica a que pertencem.

[2] O sistema prevê a palavra ou frase que o usuário quer digitar de acordo com o contexto no qual a palavra ou frase está inserido.

[3] Programas auxiliares que automatizam funções repetitivas do usuário e orientam passo a passo a realização de uma determinada tarefa.

dor a um novo estado e seu estado corrente sempre corresponde à linha do script que está sendo executada. Esta linha é chamada de *foco do provador*. Interfaces seguindo este modelo costumam utilizar um sistema de cores para visualmente indicar ao usuário os comandos já processados, o comando sendo processado no momento e o que ainda não foi executado. Podemos perceber quanto esta forma de enviar comandos ao provador está fundamentada em uma visão de prova como programação, assemelhando-se a maneira como programadores acompanham a execução de programas em alguns ambientes de desenvolvimento usando ferramentas de depuração (*debugging*).

Scripts armazenados no sistema devem poder ser executados novamente a partir do estado inicial do provador, produzindo sempre os mesmos resultados, sem provocar erros. O provador é levado ao mesmo estado final de prova, após executar a sequência de comandos indicada pelo script, ou seja, o histórico de comandos enviados diretamente indica o estado do provador.

Uma das tarefas mais importantes no gerenciamento de scripts pela interface deve ser fornecer capacidades para desfazer e refazer a aplicação de comandos no provador, permitindo aos usuários voltarem atrás e descartarem comandos que levem a direções erradas. Estes recursos estão normalmente disponíveis através de opções de menu e atalhos de teclado. O script de prova final, visualizado pelo usuário contém apenas os comandos confirmados pelo usuário. Mesmo ao desfazer comandos, devemos considerar a possibilidade de ocorrência de erros. Comandos enviados ao provador podem ser expressões extensas e necessitarem de um tempo considerável para serem avaliados para processamento pela máquina de prova, o que normalmente não ocorre com sistemas de desfazer e refazer em outros tipos de software. Além disso devemos considerar que estamos propondo um ambiente distribuído, onde interface e provador são processos distintos, o que aumenta ainda mais a dificuldade de suportar tais funções.

Scripts estão baseados na linguagem de comandos do provador, em sua sintaxe e em seu conjunto de operações. Conforme (Geu09), esta linguagem pode ser de dois tipos: *declarativa* ou *procedural*. Em uma linguagem procedural, o usuário diz ao sistema *o que fazer* e *como fazer*. Em uma linguagem declarativa o sistema diz ao usuário *onde chegar*. Para deixar mais clara esta diferença comparamos dois scripts de prova para o teorema de mostrar que se dobrarmos um número e o dividirmos em seguida por 2, obteremos o mesmo número novamente. A figura 4.3 mostra um exemplo de um script de prova para o provador Coq, usando uma linguagem procedural. A figura 4.4 mostra um script, escrito em uma linguagem declarativa, proposta por Corbineau, para o mesmo provador Coq. Uma descrição desta linguagem declarativa pode

PUC-Rio - Certificação Digital Nº 0711291/CA

ser encontrada em (Cor08).

```
Theorem double_div2: forall (n : nat), div2 (double n) = n.
simple induction n; auto with arith.
intros n0 H.
rewrite double_S; pattern n0 at 2; rewrite <- H; simpl; auto.
Qed.
```

Figura 4.3:

```
Theorem double_div2: forall (n : nat), div2 (double n) = n.
proof.
    assume n:nat.
    per induction on n.
        suppose it is 0.
            thus thesis.
        suppose it is (S m) and IH:thesis for m.
            have (div2 (double (S m))= div2 (S (S (double m)))).
                 ~= (S (div2 (double m))).
            thus ~= (S m) by IH.
    end induction.
end proof.
Qed.
```

Figura 4.4:

As diferenças entre os dois estilos são visualmente verificáveis. O estilo declarativo produz scripts mais longos do que versões imperativas, o que é bom para a leitura, mas pior para digitação. A forma imperativa pode ser mais confortável para usuários com prática em programação, enquanto a forma declarativa é mais próxima de matemáticos. A primeira não se aproxima muito de uma prova formal em lógica, ao contrário da segunda. Questões como estas devem ser observadas ao se projetar o tipo de linguagem que a interface vai oferecer. É preciso avaliar, entre outras coisas, por exemplo, qual a verdadeira função do script de prova no ambiente: serve apenas como um conjunto de funções que será executado pelo provador ou também será usado como uma forma de apresentação da prova final? Parece razoável que ambientes de prova tentem combinar os dois estilos o máximo possível. Por exemplo, o usuário pode digitar o script de prova da forma procedural e o sistema fornecer recursos para expandí-lo para uma forma mais declarativa, facilitando a leitura.

Por fim, é importante projetar adequadamente os estados possíveis para o script de prova. Devemos identificar quais os comandos do usuário na interface

PUC-Rio - Certificação Digital Nº 0711291/CA

provocam a mudança de estado do script e como estes estados se relacionam com o comportamento subsequente do provador.

4.5 Requisitos de Interfaces para Provadores

Völker propõe o uso de técnicas de elicitação de requisitos para a identificação das funcionalidades que interfaces para assistentes de prova devem fornecer a seus usuários, em (Vol03). Ele defende o emprego das técnicas de *casos de uso* e *identificação de objetos*, como formas relativamente simples de conduzir um processo de elicitação de requisitos e que podem propiciar consideráveis benefícios ao design da aplicações.

Casos de uso são descrições de um conjunto de interações entre um usuário e o sistema. Casos de uso representam objetivos que um usuário pretende realizar utilizando o sistema. A análise de casos de uso (BRJ06) é uma técnica para coletar requisitos amplamente difundida na Engenharia de Software, tendo alcançada bastante sucesso ao longo dos anos.

O autor acredita que, para o desenvolvimento de ambientes para prova de teoremas, um dos maiores benefícios que podem ser alcançados com o uso destas técnicas está no fato de que elas são centradas no usuário. Um aspecto importante, considerando que, a maioria dos ambientes existentes tende a ser desenvolvidos orientados pelo ponto de vista da máquina de prova.

Aplicando a técnica de análise de casos de uso, o autor começa identificando os *atores* do sistema. Em um caso de uso, um ator representa um determinado papel através do qual um usuário interage com sistema. Os seguintes atores foram, então, identificados:

- Desenvolvedor de Teorias: que pode ser um especificador, um provador (que realiza a busca pela prova) e um leitor. Estes tipos de usuários se baseiam na classificação sugerida em (Gog99).
- Desenvolvedor de Ferramentas de Prova: que executa atividades como a programação, adaptação e teste de estratégias e procedimentos de prova.
- Administrador do Sistema: que instala, monitora e atualiza o sistema.

Völker ainda propõe duas sub-categorias de atores, de acordo com a experiência do usuário. Casos de uso podem, então, variar para usuários novatos e usuários experientes.

Como forma de facilitar a descoberta de possíveis casos de uso, o domínio do problema pode ser analisado na busca por objetos importantes para interação entre usuário e sistema. Esta identificação permite classificar elementos do domínio em categorias distintas que podem ser usadas como

PUC-Rio - Certificação Digital Nº 0711291/CA

base para o futuro desenvolvimento. Algumas heurísticas podem ser aplicadas à lista de objetos obtida nesta etapa para ajudar a encontrar possíveis casos de uso. Não vamos reproduzir aqui a lista de objetos e casos de uso apresentada em (Vol03), mas vale o registro de que a mesma foi usada para projetar as funcionalidades desejadas em nossa própria solução, conforme veremos no capítulo seguinte.

Völker baseia sua identificação de objetos e a análise de casos de uso numa descrição geral do problema de usar uma interface com provadores para provar teoremas, onde destaca os principais conceitos presentes nesta interação. Resumimos abaixo esta descrição, pois serviu de base para nosso próprio processo de elicitação.

- Normalmente, o desenvolvimento de provas consiste de algum tipo de projeto como provar um teorema ou realizar a verificação formal de um programa.
- O desenvolvimento pode ser dividido em teorias. Teorias formam uma hierarquia.
- O desenvolvimento de teorias pode ser persistido, por exemplo, como scripts de prova. O sistema deve fornecer comandos para armazenar e carregar teorias.
- O desenvolvimento de teorias ocorre sobre um contexto lógico que consiste de constantes, axiomas e teoremas que já foram provados.
- Uma prova é iniciada criando-se uma fórmula inicial (*objetivo*) que deve ser provada. A partir da fórmula inicial ocorrem sucessivos passos de prova. Em cada um, o usuário analisa o estado da prova e, então, aplica um comando, que é executado pela máquina de prova, levando o sistema ao próximo estado.
- Um estado de prova é caracterizado por um conjunto de objetivos abertos que ainda precisam ser provados. Um estado de prova pode ainda conter declarações locais e hipóteses.
- Uma prova termina quando um estado sem objetivos em aberto é alcançado.
- Processos de prova podem ser abandonados definitivamente ou temporariamente. Nesse último caso, o usuário pode querer voltar ao ponto anterior em algum momento.

PUC-Rio - Certificação Digital Nº 0711291/CA

4.6 Outros Trabalhos Relacionados

As características descritas acima correspondem ao conjunto fundamental de capacidades necessárias em um ambiente de provas interativo. Na literatura científica podemos encontrar, ainda, outras orientações que podem servir de apoio na construção destas ferramentas. Abaixo, apresentamos um pequeno resumo dos principais trabalhos na área, destacando suas ideias centrais.

Goguen (Gog99) baseia-se em conceitos originados nas Ciências Sociais, na Ciência Cognitiva e na Semiótica para definir princípios e técnicas gerais para o projeto de interfaces de provadores de teoremas. Ele destaca a possibilidade de mesclar provas formais com explicações e tutoriais informais e propõe recursos como visualização (browsing) e animação de provas.

Eastaughffe (Eas98) relaciona mais algumas funcionalidades importantes para interfaces de provadores como: suporte a múltiplas visões da construção da prova e flexibilidade com relação ao formato dos comandos emitidos pelos usuários.

Homik e Meier (HM05) apresentam os resultados de um experimento envolvendo usuários não especialistas com provadores de teoremas, o que permitiu identificar novas necessidades para estes ambientes, quando usados por este tipo de usuários.

Merriam e Harrison (MH97) ressalvam que uso de interfaces gráficas pode desviar a atenção do usuário de seu problema principal (a prova). O usuário pode facilmente entrar em um ciclo de tentativa e erro, não investindo esforços suficientes no planejamento da prova. Embora esse argumento seja usado por eles para promover interfaces de linha de comando, suas considerações podem ser usadas na construção de interfaces gráficas melhores.

4.7 Principais Ferramentas

Abaixo apresentamos três das interfaces mais conhecidas atualmente, destacando suas principais características, vantagens e limitações.

4.7.1 Jape

Jape (Jape) é um *editor de prova* desenvolvido por Richard Bornat e outros na Universidade de Oxford, Londres. Seu principal objetivo é servir como uma ferramenta de apoio ao ensino de lógica e, talvez, seja o programa de computador mais usado para este fim, como sugerido em (KWHR07). Jape difere do modelo de ambiente que discutimos ao longo deste capítulo: não funciona

PUC-Rio - Certificação Digital Nº 0711291/CA

como uma interface conectada a alguma máquina de prova distinta. Como um editor de prova, seu principal foco é auxiliar o usuário a "escrever"provas no computador, da mesma maneira que editores e processadores de texto auxiliam na "escrita"de documentos.

O Jape não é um provador em si, mas auxilia o usuário a resolver uma prova, oferecendo conjuntos de regras de inferência que podem ser aplicados sobre fórmulas, no estilo *backward* ou *forward*. Na verdade, trata-se de esquemas de aplicação de regras. Quando o usuário solicita que o esquema seja aplicado em determinada parte da prova, a ferramenta analisa o contexto de aplicação, "casando"as informações disponíveis com os espaços abertos do esquema e solicitando ao usuário decidir ou informar aquelas necessárias para completar a aplicação da regra. Esta é a técnica conhecida como *unificação*.

O usuário pode estender a capacidade do editor, criando novos sistemas de regras através de uma linguagem própria oferecida pela ferramenta. Com esta linguagem o usuário também descreve como a interface deve funcionar para suportar o conjunto de regras em questão.

Como uma ferramenta de apoio ao ensino, Jape prioriza os usuários iniciantes, aqueles que estão começando em Lógica se encontram desorientados diante do grande número de novos conceitos que precisam enfrentar e compreender. Para isto, Jape propõe uma interface silenciosa ou *quiet interface*, usando o termo em inglês apresentado por Bornat e Sufrin em (BS96). Uma interface silenciosa é aquela que mostra ao usuário apenas o que ele quer ver, descartando os detalhes irrelevantes, evita que o usuário tenha que buscar as informações relevantes, oferece operações que o mesmo compreende e o conduz gradativamente a conclusões que inicialmente não compreendiam. Segundo os autores este tipo de interface é a ideal para suportar o grupo de usuários alvos da ferramenta e seus principais desafios.

O conceito de interface silenciosa é implementado no Jape seguindo quatro princípios de design importantes, os quais resumimos abaixo:

Flashy display helps A ideia por trás deste princípio é que, como o foco da ferramenta está em escrever provas, estas provas devem parecer com aquelas que o usuário costuma escrever no papel. Isto implica em usar as mesmas fontes tipográficas que aquelas usadas nos livros, o mesmo layout, nomear regras de inferência da mesma forma que os professores fazem em salas de aula. Esta característica é fundamental para alcançar o público-alvo do ferramenta: usuários iniciantes. O Jape utiliza fontes especiais para representar simbolos matemáticos comuns em lógica e na matemática, como, por exemplo, $\vdash$, $\rightarrow$, $\forall$ e $\bot$. Como a prova apresentada deve parecer com as feitas em papel, representações internas, usadas

PUC-Rio - Certificação Digital Nº 0711291/CA

como apoio a soluções técnicas, como identificadores, descritores de tipo, entre outros, são totalmente descartados do resultado exibido ao usuário.

Steps in a good proof are easy to find No Jape provas podem ser exibidas de duas formas: no estilo em árvore de Gentzen e no estilo em caixas de Fitch. Os projetistas da ferramenta argumentam que o estilo em caixas favorece que o usuário se localize mais facilmente na prova, contribuindo mais para uma interface silenciosa, pois são lineares e menores do que a versão correspondente no estilo em árvore. Para melhorar a percepção do usuário, ao selecionar uma linha da prova no formato em caixas, o Jape torna as fórmulas irrelevantes acinzentadas, destacando assim aquilo que é realmente importante para aquele ponto da prova. Este artifício corrige a única desvantagem da visão em caixas em relação a visão em árvore, onde essa relevância é explícita. Apesar de valorizar a visão em caixas, a ferramenta disponibiliza as duas formas de prova, principalmente porque usuários novatos tendem a preferir a forma em árvore, pelo menos no início.

Don't poke them in the eye Para manter a interface silenciosa, com o mínimo de interferência à maneira tradicional como o usuário realiza uma prova, o mecanismo de unificação para aplicação de regras no Jape permite que o usuário selecione fórmulas com o mouse, antes de aplicar a regra. Isso evita que o sistema interfira no processo, solicitando ao usuário que faça algum tipo de escolha, normalmente através de algum mecanismo de diálogo. A seleção de fórmulas *a priori* permite, inclusive, em apresentações em dedução natural, decidir se a regra *forward* ou *backward* deve ser aplicada, levando em conta se o usuário escolheu, inicialmente, uma hipótese ou uma conclusão, respectivamente. No Jape, este princípio também é aplicado na tentativa constante dos projetistas de ocultar do usuário mecanismos internos da ferramenta.

Follow the user's tradition Manter a tradição é tornar a ferramenta cada vez mais próxima do vocabulário, mecanismos e soluções que o usuário usa ao lidar com provas no dia a dia. Isto não é uma tarefa fácil, como descrito por Bornat e Sufrin em (BS96), quando relatando a dificuldade de oferecer uma versão da teoria de conjuntos na qual os operadores fossem definidos axiomaticamente. Jape não suporta diretamente definições axiomatizadas, que devem ser simuladas através do uso de regras de inferências. A ferramenta fornece um conjunto de regras já preparadas com este mapeamento para uso por usuários codificadores. Apesar disso,

PUC-Rio - Certificação Digital Nº 0711291/CA

a versão axiomatizada é aquela que o usuário encontra nos livros e textos de lógica e deveria ser suportada nativamente.

Os autores descrevem que desenvolver uma interface silenciosa é um processo gradativo. Em (BS96) eles mostram como o editor foi evoluindo ao longo do tempo, substituindo soluções invasivas ao usuário por versões mais silenciosas. Mesmo assim, a ferramenta ainda não alcançou totalmente este objetivo.

4.7.2 CtCoq e PCoq

PCoq (ABPR01) é a atual implementação de uma interface gráfica para o provador Coq, desenvolvida pelo grupo de pesquisa em métodos formais do INRIA (*Institute National de Recherche en Informatique et en Automatique*), na França. PCoq é o sucessor do ambiente de prova CtCoq, a interface para provadores que originou muitos dos conceitos propostos em referências deste capítulo, como em (BT98), (TBK92) e (BKT94). Este último foi desenvolvido usando um antigo gerador de ambientes interativos denominado Centaur e muitas de suas funcionalidades se baseavam nas ferramentas de programação disponíveis por este ambiente. A nova solução é implementada em Java e, em sua elaboração, muitas das decisões de design do produto anterior foram reavaliadas.

O ambiente CtCoq promoveu uma série de princípios, hoje consagrados no desenvolvimento de interfaces para provadores: a separação entre interface e máquina de prova, a edição estruturada da árvore de prova e o uso de scripts de prova, registrando a sequência comandos entrados na interface e enviados para o provador como um documento completo.

Com a interface e a máquina de prova executando como processos distintos, o ambiente baseava-se em um protocolo de comunicação para troca de dados matemáticos entre os componentes do sistema, que precederam e influenciaram os padrões atuais baseados em XML, OpenMath (OpenMath) e MathML (MathML). Além disso, CtCoq é o ambiente de origem do método de interação de *prova por apontamento*.

Como vimos, fórmulas matemáticas envolvem símbolos sofisticados como $\surd$, $\int$ e ∞, para citar mais alguns exemplos, que interfaces para provadores deveriam suportar, permitindo sua entrada pelos usuários e exibindo-os com o layout e formatação adequados. Uma das características mais interessantes deste ambiente é o tratamento destes simbolos matemáticos com alto grau de qualidade, de uma maneira bem próxima a representação tipográfica tradicional encontrada nos livros. Isto torna a leitura de fórmulas mais fácil

PUC-Rio - Certificação Digital Nº 0711291/CA

e direta para o usuário do sistema, melhorando sua interação com o sistema. Além da formatação primitiva oferecida pelo ambiente, o layout de expressões pode ser configurado pelo usuário usando uma linguagem de *pattern-matching*, originária no Centaur, denominada PPML (*Pretty Printing Markup Language*. Esta linguagem também está disponível no produto atual, o PCoq.

O ambiente CtCoq não é apenas uma ferramenta protótipo e sim, vem sendo usado como interface padrão para o provador Coq em projetos com intensivo desenvolvimento de provas formais. Atualmente, PCoq é a plataforma padrão, substituindo o anterior, mas mantendo suas principais propriedades. Por ser baseado em Java, o novo ambiente é multiplatoforma, o que facilita sua adoção por usuários de diferentes sistemas operacionais.

4.7.3 Proof General

Proof General (PG) é uma interface genérica para provadores interativos, construída com base no editor de texto customizável Emacs (Emacs). Como uma interface genérica, suporta, de forma adaptável, diferentes provadores. Em sua distribuição padrão, o Proof General vem configurado para trabalhar com os provadores Isabelle, Coq, PhoX e LEGO. Muitos outros provadores estão disponíveis em versões experimentais. O projeto é mantido pelo LFCS, o grupo de pesquisa em Teoria da Computação da Universidade de Edimburgo e liderado por David Aspinall.

A versão para Emacs do Proof General é um produto consagrado, uma das interfaces para provadores mais conhecidas e utilizadas pela comunidade. Apesar do sucesso, a versão para Emacs apresenta uma série de limitações, segundo os próprios autores, conforme (ALW07). Usuários precisam conhecer ou aprender Emacs e aceitarem seu estilo de interação próprio. Do ponto de vista técnico, desenvolvedores devem lidar com a API em LISP para Emacs, que é restrita, pouco confiável, constantemente modificada e inconsistente entre os diferentes tipos de editores Emacs existentes. Outro problema é que o ambiente Proof General surgiu através de sucessivas extensões a uma base genérica para suportar mais provadores, o que, apesar de evitar modificações nas máquinas de prova em si, tornou o design da ferramenta mais complicado e frágil.

Os motivos acima levaram a equipe do Proof General à concepção de uma nova arquitetura para seu produto. A nova solução, denominada Proof General Kit (ALW07) ou PGK, é um framework para desenvolvimento de interfaces para provadores, baseado na arquitetura genérica de componentes distribuídos proposta por Bertot e Théry (como apresentamos na seção 4.4). Assim, o

PUC-Rio - Certificação Digital Nº 0711291/CA

PGK consiste, basicamente, em um modelo genérico que busca abstrair o comportamento comum das máquinas de prova e um protocolo bem definido para comunicação entre os diversos componentes do sistema.

O framework Proof General Kit é composto de três tipos de componentes fundamentais: provadores, interfaces e um *broker* central que coordena a comunicação entre os dois primeiros. A figura 4.5 apresenta a arquitetura do PGK. Os componentes se comunicam através de troca de mensagens codificadas no formato XML e transmitidas em uma implementação de RPC (*Remote Procedure Call*). Este arranjo diminui o acoplamento entre as partes e torna a adição de novos componentes, como um novo provador ou uma nova forma de interface, mais fácil. O conjunto de mensagens possíveis e a ordem de chamada é estabelecido pelo protocolo de comunicação, denominado PGIP (*Proof General Interaction Protocol*), ou seja, o PGIP estabelece o esquema XML para a troca de mensagens. Um segundo esquema XML, chamado PGML (*Proof General Markup Language*) é usada para mostrar símbolos matemáticos de forma mais apurada.

PUC-Rio - Certificação Digital Nº 0711291/CA

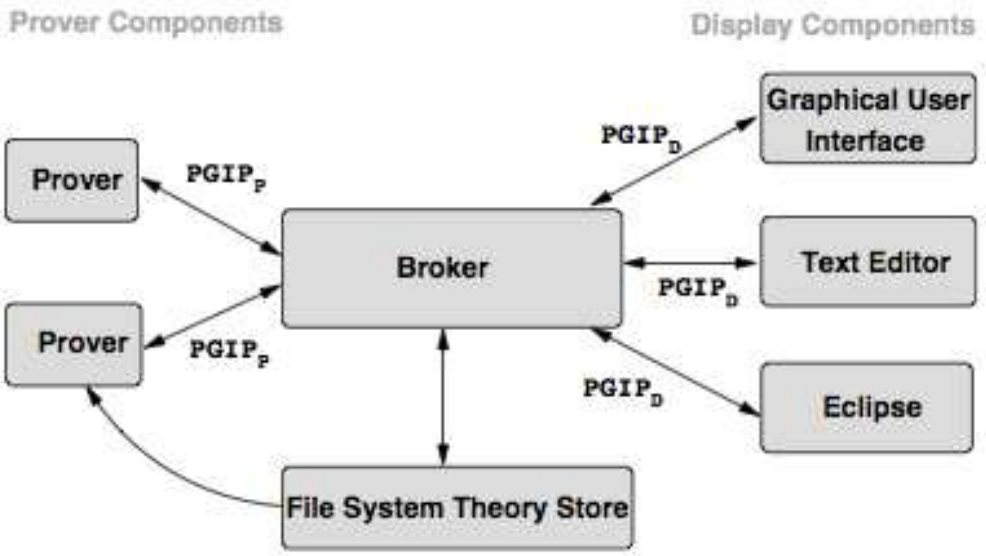

Figura 4.5:

Scripts de prova são o principal artefato do sistema. O princípio básico usado para representá-los no PGIP é usar a linguagem nativa do provador e anotar(*mark up*) seu conteúdo com comandos próprios do protocolo, que explicam a estrutura do script, de forma que possam exibidos nas interfaces. Comandos entrados pelo usuário via interface são verificados (*parse*) e inseridos no script de prova e, então, executados, levando o provador a um novo estado de prova.

Uma versão do Proof General atual para Emacs, baseado no PGK já está em desenvolvimento. Uma nova implementação do ambiente para a plataforma Eclipse também. Esta última reforça ainda mais a escolha por uma forma de interação fundamentada na visão de *prova como programação*. Os responsáveis pela ferramenta anunciam ainda implementações do framework nativas para desktop e para web.

PUC-Rio - Certificação Digital Nº 0711291/CA