5
Uma Interface Web para Provadores Interativos

No capítulo anterior estudamos os diferentes modelos de interação entre usuários e ambientes de prova, como estes modelos afetam a concepção de boas interfaces para provadores e os componentes e características essenciais destas interfaces. No presente capítulo, apresentamos como todos estes conceitos foram utilizados na definição, projeto e construção de uma interface web para provadores, desenvolvida no Laboratório de Métodos Formais (TecMF) do Departamento de Informática da PUC-Rio. O laboratório possui uma linha de pesquisa destinada ao desenvolvimento e aplicação de provadores de teoremas automatizados e, entre seus projetos já desenvolvidos está o provador *Hemera*, uma máquina de prova construída com o objetivo de suportar diferentes linguagens lógicas e múltiplas estratégias de prova. Em seu atual estágio, o Hemera suporta provas em dois sistemas dedutivos, Cálculo de Sequentes e Tableaux para a Lógica Proposicional, aplicando a busca por prova no estilo *backward*.

Em sua primeira versão, o Hemera fornecia uma interface textual, bastante limitada, para interação com o usuário. Nela, o usuário do sistema podia apenas digitar fórmulas e comandos e submetê-los para o processamento imediato do provador, que apresentava os resultados de volta ao usuário. Estas respostas podiam ser mensagens referentes ao processamento dos comandos ou representações textuais do estado da prova. O ambiente funcionava de forma monousuária, isto é, uma instância do provador atendia às solicitações de um único usuário, não havia persistência das informações entre diferentes sessões de uso e a aplicação precisava ser instalada localmente na máquina cliente.

Estes aspectos compunham alguns dos motivos que justificaram a iniciativa de se produzir uma interface mais elaborada para o provador Hemera. Este interesse originou esta dissertação e produziu a interface *HemeraWeb*, nosso ambiente de provas, baseado na web, construído para explorar as técnicas de apresentação e visualização discutidas anteriormente. A experiência inicial com o provador Hemera serviu como base para identificarmos uma infraestrutura básica para provadores na web, a qual acreditamos ser capaz de suportar nosso objetivo de suportar diferentes máquinas de prova no futuro.

PUC-Rio - Certificação Digital Nº 0711291/CA

Neste capítulo, discutimos as razões que nos levaram à escolha da web como plataforma de desenvolvimento e porque acreditamos que este será um cenário comum para implementação destas ferramentas no futuro. Depois, seguimos descrevendo nossa solução, seu funcionamento básico, principais componentes, funcionalidades e como os conceitos discutidos no capítulo anterior influenciaram sua elaboração.

Inicialmente, para fornecer uma visão geral da aplicação proposta, apresentamos, passo a passo, como executar um processo de prova usando a interface protótipo desenvolvida.

5.1 O Processo de Prova

Para realizar uma prova o usuário começa estabelendo uma fórmula que representa a conjectura que deseja provar. Esta fórmula constitui o objetivo inicial do processo de prova. As figuras abaixo exemplificam, passo a passo, o processo de provar a sentença $A \wedge (A \rightarrow B) \vdash B$, a conhecida regra de *modus ponens*, usando o provador Hemera configurado para o sistema de cálculo de sequentes.

Inicialmente (figura 5.1), o usuário cria um novo script de prova, informando a fórmula inicial. Uma vez garantida que a fórmula está sintaticamente correta, o processo de prova pode ser iniciado.

Ao iniciar o processo de prova (figura 5.2), o objetivo inicial é carregado no provador. Na figura, a elipse amarela indica que se trata de um objetivo ainda em aberto. Ao clicar na elipse que representa este objetivo o usuário pode obter a lista de regras possíveis neste ponto (figura 5.3), numa adaptação ao modelo de prova por apontamento. No exemplo em questão a regra possível é o $(\wedge - esq)$

O usuário aplica a regra sugerida, neste caso, a única disponível. O sistema, então, avança um passo de prova, mudando o estado do provador e reproduzindo esta mudança na interface, visualmente para o usuário (figura 5.4. Neste ponto o sistema apresenta o novo objetivo em aberto e a regra aplicada que o produziu. O objetivo anterior deixa de ser um objetivo a ser provado e é representado como um retângulo cinza, indicando que o mesmo já foi resolvido.

O usuário verifica novamente as regras disponíveis no novo objetivo em aberto (5.4) e aplica a única possível para o nosso exemplo, o $(\rightarrow -esq)$, que como sabemos, tem duas possibilidades. A aplicação da regra cria dois novos ramos na árvore que representa a prova, cada um com o novo objetivo correspondente. Como estes novos objetivos são axiomas, eles não precisam ser

PUC-Rio - Certificação Digital Nº 0711291/CA

provados e, por isto, são representados como objetivos resolvidos. Como neste estado, não existe nenhum novo objetivo a ser provado, o sistema finaliza o processo de prova, mudando o estado do script para *Provado*.

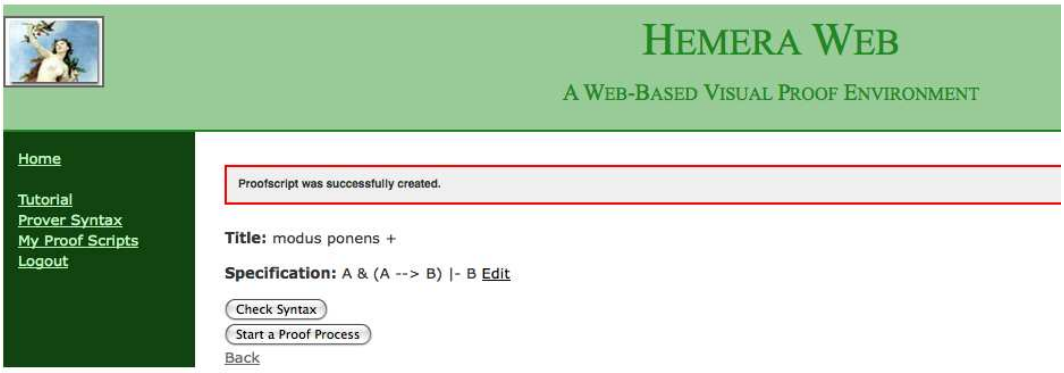

Figura 5.1: Etapa 1 - Informar o Objetivo Inicial

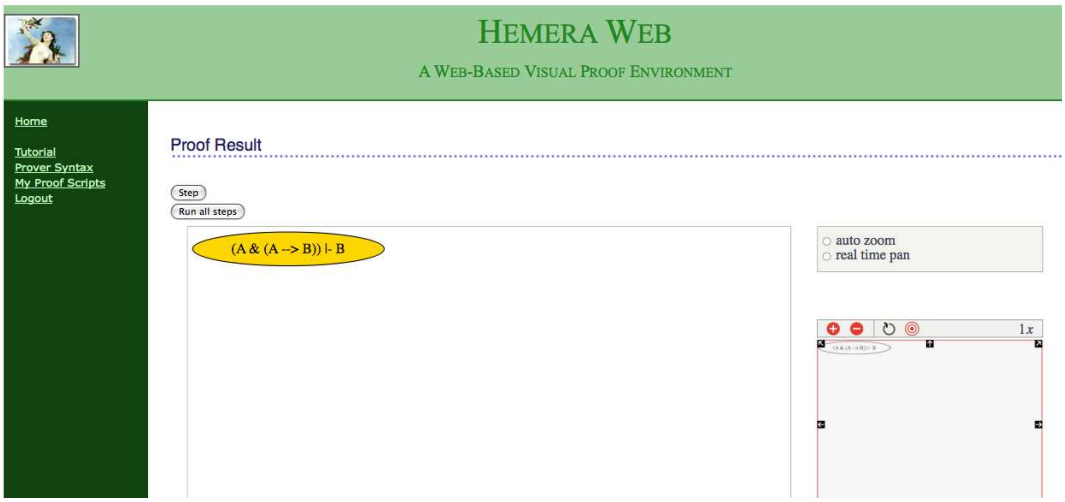

Figura 5.2: Etapa 2 - Provador iniciado com o objetivo inicial

Nas próximas seções discutimos em detalhes os conceitos por trás dessa ferramenta.

5.2 A Web como Plataforma para Interfaces para Provadores

Na introdução desta dissertação, citamos alguns dos cenários de uso que claramente demonstram como a web se tornou parte da vida diária das pessoas, apoiando-as nas mais variadas tarefas, profissionais e pessoais. Esta disseminação popularizou o estilo de interação da web, baseado em *hyperlinks*, tornando-o amplamente conhecido entre os mais variados perfis de usuários.

PUC-Rio - Certificação Digital Nº 0711291/CA

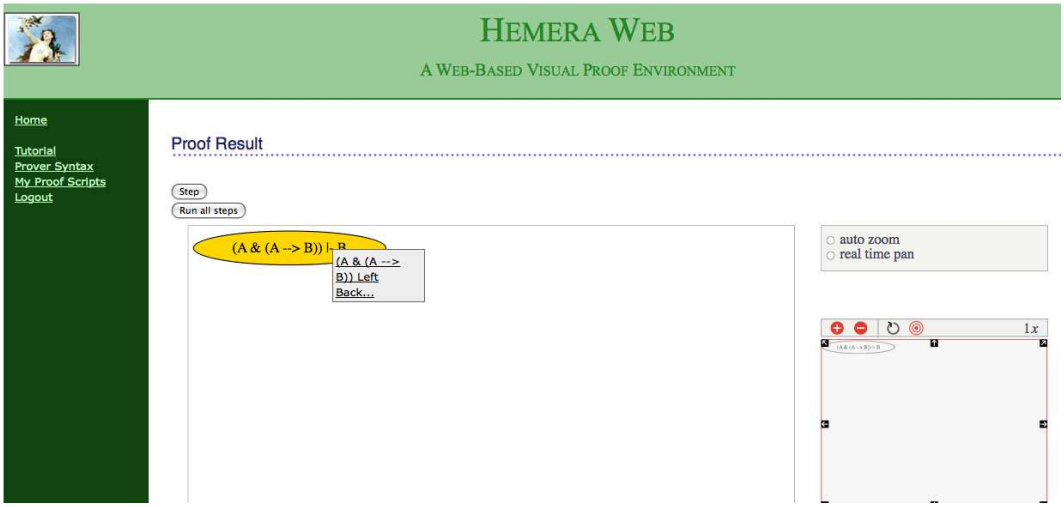

Figura 5.3: Etapa 3 - Lista de regras disponíveis no objetivo escolhido

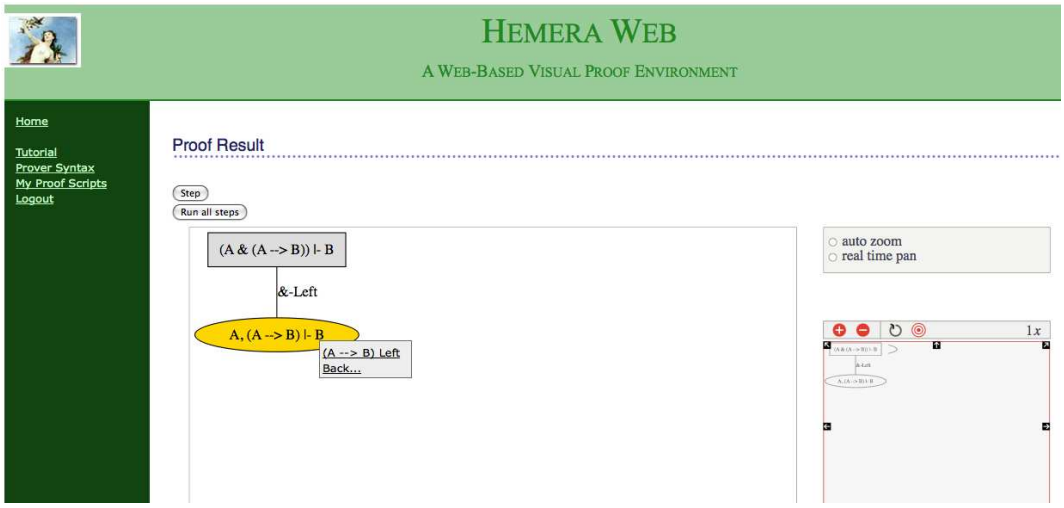

Figura 5.4: Etapa 4 - Regra aplicada e apresentação de nova lista de regras

Com o aumento da disponibilidade do acesso a Internet e com o crescimento das conexões de alta velocidade, mais e mais serviços são oferecidos através da rede, em diferentes domínios de conhecimento.

Apesar disso, encontramos poucas referências sobre o uso de ambientes baseados na web para prova de teoremas. Em um dos primeiros trabalhos sobre o tema, Goguen destaca algumas vantagens desta abordagem em (Gog99), onde também apresenta uma ferramenta nestes moldes, construída com tecnologias disponíveis na época. A evolução rápida das tecnologias existentes e o surgimento de novos recursos para facilitar o desenvolvimento de aplicações para a web, tornaram desatualizadas muitas das descrições narradas por Goguen sobre a construção de seu ambiente. Em uma referência mais recente (Kal07), Kaliszyk apresenta uma interface web genérica denominada

PUC-Rio - Certificação Digital Nº 0711291/CA

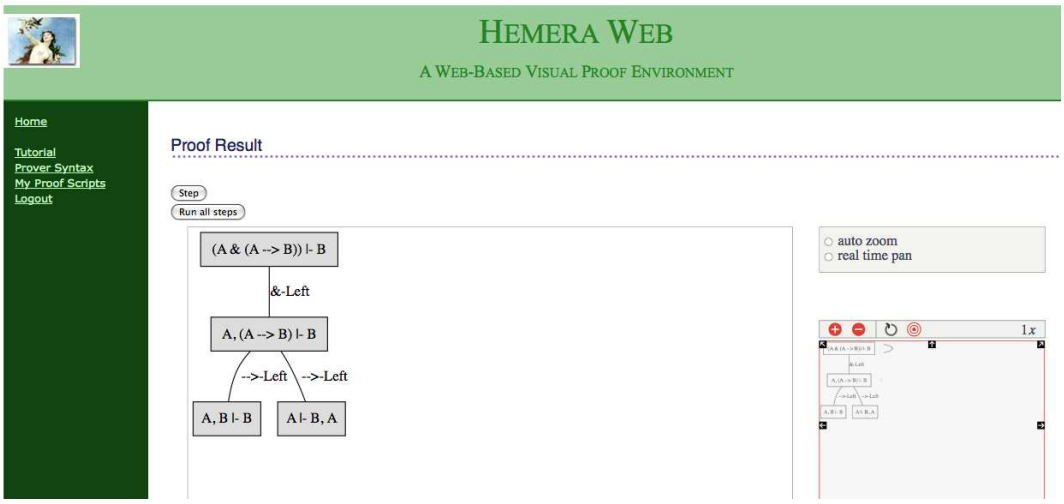

Figura 5.5: Etapa 5 - Prova concluída

ProofWeb (ProofWeb). Ele destaca como principal vantagem da web o fato desta solução evitar que o usuário tenha que se preocupar com os passos de instalação do ambiente de prova, normalmente tarefa não trivial para a maioria dos provadores existentes. Contra o argumento de que interfaces de desktop são superiores em performance e em recursos, Kaliszyk contrapõe que, com as tecnologias disponíveis atualmente para desenvolvimento de aplicações para web, interfaces nesta plataforma podem se comportar de forma muito próxima a interfaces locais e ainda, com as demais vantagens próprias deste ambiente. JavaScript, a API DOM (DOM) e XmlHttp (XmlHttp) formam a combinação conhecida como AJAX, acrônimo do termo em inglês, *Asynchronous JavaScript And XML*, que permite desenvolver estas facilidades. O termo AJAX foi cunhado por Jesse James Garrett no artigo (Gar05) publicado em seu blog em 2005.

Uma das principais aplicações da web tem sido como um meio para colaboração global entre as pessoas. Aplicativos como *wikis*, redes de relacionamento, *blogs*, entre outros, permitem que usuários na rede troquem informações e compartilhem conhecimento de uma forma nunca antes realizada. Um exemplo deste tipo de colaboração está nas enciclopédias globais, baseadas na tecnologia de *wikis*, como a *Wikipedia*, onde cada pessoa pode participar da criação e atualização do conteúdo, tornando o conhecimento efetivamente público. Achamos que a prova de teoremas pode ser muito beneficiada por recursos de colaboração semelhantes aos descritos acima. Usuários poderiam compartilhar teorias e o processo de prova de teoremas complexos poderia ser conduzido por diferentes participantes, sugerindo caminhos, estratégias e passos de inferência. Existem poucos exemplos do uso destas tecnologias no

PUC-Rio - Certificação Digital Nº 0711291/CA

campo matemático, porém. Merece destaque a iniciativa MathWiki (CK07), ainda experimental, que pretende funcionar como um wiki matemático, com suporte inclusive à prova de teoremas.

Além da dificuldade de instalação dos provadores existentes, temos o problema de que a maioria destes softwares são construídos para um sistema operacional específico, o que limita o número de possíveis usuários da ferramenta. Poucos provadores possuem versões para diferentes plataformas, pois costumam utilizar recursos próprios do ambiente para fornecer seus serviços. Usuários que necessitam usar o provador em uma plataforma diferente daquela em que este foi concebido, precisariam fazer uso de alguma ferramenta de *virtualização*, o que também não é uma tarefa trivial para usuários não técnicos. Se o provador possui uma interface baseada na web, não importa para o cliente o sistema operacional e bibliotecas que estão sendo usadas pelo servidor, bastando que o usuário possua, em seu ambiente, um browser de navegação.

Nossa abordagem se assemelha à proposta por Kaliszyk: desejamos criar um ambiente de prova leve, com recursos e facilidades semelhantes às encontradas nas interfaces para desktop, sem a necessidade de instalação de nenhum software adicional, tornando o processo mais fácil para o usuário final. Acreditamos que desta maneira poderemos alcançar o público não técnico, formado por matemáticos, lógicos e alunos em cursos de lógica, afastados normalmente deste tipo de aplicativo devido a complexidade para colocá-los em funcionamento.

Descrevemos nossa solução nas próximas seções. Apresentamos a arquitetura geral empregada e explicamos cada componente da mesma, confrontando com os conceitos apresentados no capítulo 4.

PUC-Rio - Certificação Digital Nº 0711291/CA

5.3 Arquitetura Geral da Solução

Vimos como as novas tecnologias para desenvolvimento de software na web podem facilitar a criação de aplicações com vastos recursos de interatividade. Não apenas isto, mas vimos como a web se tornou uma plataforma para comunicação de múltiplas mídias, ou seja, texto, gráficos, imagens, vídeos, áudio, etc. É também uma plataforma de colaboração, permitindo que pessoas de diferentes partes do planeta se relacionem, troquem informações e construam conhecimento juntas. Todos estes pontos fazem da web uma plataforma adequada para hospedar um ambiente de provas colaborativo, onde usuários espalhados pelo mundo possam compartilhar e auxiliar uns aos outros no desenvolvimento de suas provas.

Nosso trabalho está baseado na arquitetura genérica para interfaces com provadores proposta por Bertot e Théry em (BT98), como apresentamos no capítulo anterior. A visão dos autores de uma arquitetura de componentes distintos, distribuídos, comunicando-se através da troca de mensagens pré-estabelecidas, sobre algum meio de comunicação em rede, tornou-se o padrão para muitas das principais interfaces para provadores existentes atualmente. Este é o modelo por trás de ferramentas como PCoq, PVS (Pvs) e ProofGeneral, por exemplo. Inclusive, dentre estes exemplos, a interface ProofGeneral, principalmente em sua versão mais nova para a plataforma Eclipse, também inspirou muito nosso trabalho, principalmente no que diz respeito a utilização de um *broker*, um componente da arquitetura usado para mediar a comunicação entre a interface e a máquina de prova.

Nosso ambiente de prova, o HemeraWeb é, portanto, um conjunto de componentes, distribuídos, comunicando-se através de protocolos padrões da Internet e da web, como HTTP (Http), SOAP (Soap) e XML (XML). A figura 5.6 apresenta os componentes fundamentais de nossa arquitetura e seguimos com uma breve descrição de como eles interagem entre si.

PUC-Rio - Certificação Digital Nº 0711291/CA

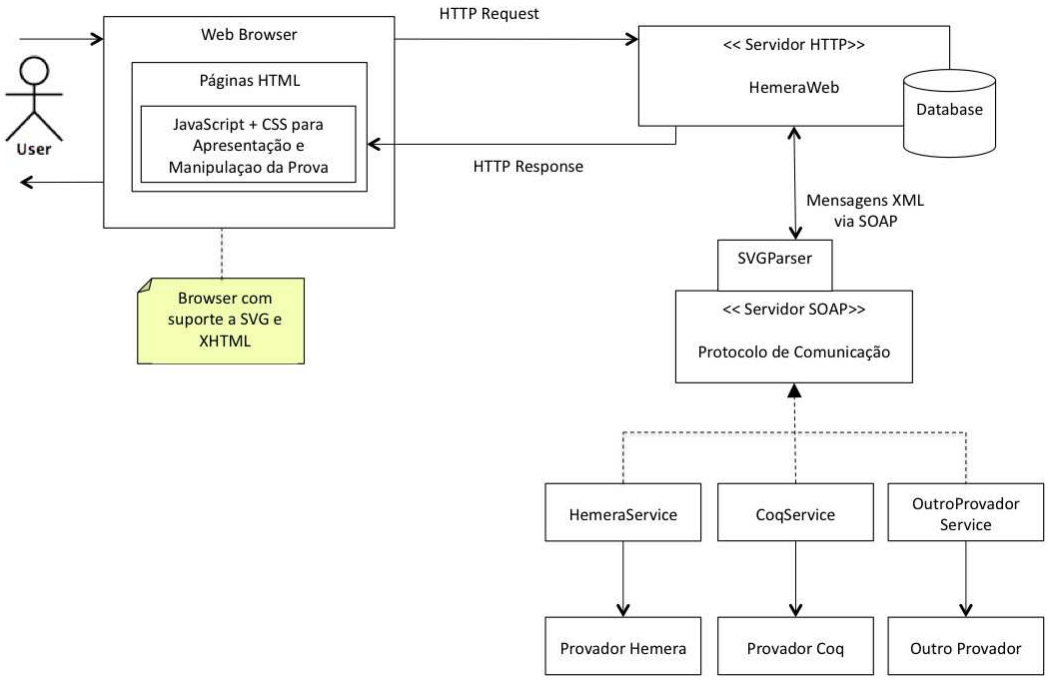

Figura 5.6: Arquitetura do HemeraWeb

O usuário se conecta ao sistema através de um *browser web*, acessando a página inicial do ambiente de prova. Isto é, a interface do nosso ambiente de prova é implementada como uma *aplicação web*, executando em um servidor HTTP, que recebe solicitações neste formato, realiza seu processamento e

devolve a resposta para o browser cliente, que renderiza e apresenta os resultados visualmente para o usuário. A aplicação web gerencia as sessões do usuário com o ambiente, mantém seu conjunto de scripts de prova e o estado dos processos de prova em aberto. Estas informações são mantidas em um *banco de dados* auxiliar. A aplicação web é composta também de uma biblioteca JavaScript e arquivos CSS (*Cascading Style Sheet*) que são usados no browser para exibir a prova e permitir a manipulação direta dos objetos gráficos que a representam, pelo usuário.

Os comandos do usuário são interpretados pela interface e repassados para processamento pelo provador, na forma de chamadas ao *web service* (WebService) que implementa o protocolo de comunicação entre a interface e o provador. Esta comunicação é feita via SOAP, o protocolo padrão para troca de mensagens através de web services, do W3C. Neste padrão as mensagens trocadas entre o consumidor (interface) e o provedor do serviço (máquina de prova) estão em um formato XML pré-definido. Uma alternativa à essa solução usando SOAP, seria a implementação do *web service* seguindo o modelo REST (Fie00). REST significa *Representational State Transfer* e define um estilo de arquitetura onde a web é vista como um conjunto de recursos, cada um identificado por uma URI (*Uniform Resource Identifier*) que podem ser manipulados através de ações definidas pelos verbos fundamentais do protocolo HTTP: GET, POST, DELETE, etc. REST não determina XML como formato para codificação das mensagens, sendo aberto a outras opções. *Web services* que seguem o estilo REST são chamados de *RESTFul web services*. Escolhemos usar SOAP, por ser um padrão W3C e pela disponibilidade de bibliotecas bem evoluídas em Python, que simplificam bastante a implementação seguindo esse padrão.

Em nossa solução, as mensagens são enviadas ao servidor SOAP, que traduz o comando encapsulado na mensagem em uma ação real na instância da máquina de prova servindo a sessão do usuário. O servidor SOAP, à cada sessão de usuário estabelecida com o ambiente, cria uma instância do provador para servir as solicitações seguintes. A arquitetura foi planejada para suportar diferentes provadores, embora tenhamos construído o ambiente inicialmente para o provador Hemera, do laboratório TecMF, como descrito acima.

Os comandos processados pelo provador mudam o estado atual da prova. Este estado é devolvido ao web service, que é responsável por traduzir a representação de estado do formato usado pelo provador para o formato SVG (*Scalable Vectorial Graphics*, ver (Svg)), uma linguagem derivada de XML, padronizada pelo W3C para exibição e manipulação de gráficos vetoriais na web. A representação em SVG é encapsulada como uma mensagem XML-

PUC-Rio - Certificação Digital Nº 0711291/CA

SOAP e envida de volta à aplicação web. Esta, por sua vez, extrai da mensagem recebida a representação do estado da prova em SVG e embute o resultado na página que será devolvida ao browser. Desta forma o estado do processo de prova pode ser exibido graficamente no cliente.

Nas próximas seções exploramos estes componentes em detalhes, apresentamos e justificamos as tecnologias usadas em seu desenvolvimento e discutimos as principais características implementadas.

5.4 Implementando o Protocolo de Comunicação como Web Service

Um dos principais argumentos apresentados no capítulo 4 para justificar a separação entre a interface e a máquina de prova foi que ferramentas mais apropriadas na construção de um podem não ser as melhores na construção do outro. Por exemplo, muitos provadores são escritos em linguagens funcionais como LISP, Prolog e ML, bastante adequadas para este fim devido a relação deste tipo de linguagem com prova formal, como vimos no capítulo 3. Por outro lado, elas oferecem suporte limitado para o desenvolvimento de interfaces sofisticadas, principalmente quando comparadas com outras linguagens com mais recursos deste tipo (a recém criada linguagem F# (FSharp) da Microsoft promete ser um caso contrário a este argumento, pois é uma linguagem funcional totalmente integrada a plataforma .Net e a suas bibliotecas para desenvolvimento de interfaces como WebForms, WPF, ASP.NET e Silverlight). Atualmente, o desenvolvimento de software para a web, por exemplo, está repleto de *toolkits* e *frameworks* que facilitam a construção e aceleram o tempo em que aplicações completamente funcionais podem ser produzidas. Bibliotecas de funções em JavaScript e frameworks baseados em linguagens dinâmicas como Ruby, Python e PHP são exemplos de combinações que propiciam esta produtividade. Por estes motivos, escolhemos adotar ferramentas como estas para o desenvolvimento de nossa interface para provadores, confiando na arquitetura distribuída proposta para integrar a aplicação produzida com o provador utilizado.

Entretanto, fazer com que componentes comuniquem-se em uma arquitetura distribuída envolve lidar com um grande número de questões complexas como tratamento de transações, concorrência, segurança, *pooling* de objetos, tratamento de erros, etc. Tipicamente, são empregados para este fim os chamados *frameworks de componentes orientados a objeto* ou *middlewares*, como CORBA, COM+ e Enterprise Java Beans (EJBs), que escondem grande parte desta complexidade do programador, permitindo que ele possa se concentrar na aplicação que está sendo desenvolvida. Porém, quando não são dependentes

PUC-Rio - Certificação Digital Nº 0711291/CA

de linguagens de programação (ex.: EJB com Java) ou de sistemas operacionais específicos (ex.: COM+ para Windows) são plataformas muito complexas (ex. CORBA), justamente para suportar esta heterogeneidade, oferecendo muitas opções e funcionalidades para o desenvolvedor, nem sempre usadas em sua totalidade.

Para componentes na web, entretanto, temos outra alternativa, mais usada neste caso e mais simples de implementar do que as soluções acima: a troca de mensagens XML através de *web services*. Um web service implementa um conjunto de funções que compõem um determinado serviço, está disponível através da Internet, utiliza um sistema de troca de mensagens baseado em XML e não está vinculado a nenhum sistema operacional, plataforma ou linguagem de programação. Web services podem ser implementados usando diferentes tecnologias como SOAP e XML-RPC (XmlRpc). São componentes de software que implementam um conjunto de funções, expostas através de uma interface pública, acessada por troca de mensagens codificadas na forma de documentos XML e, tipicamente, transmitidas sobre o protocolo de transporte padrão da Internet, o HTTP.

O uso desta abordagem para implementar a separação entre interface e provador, já foi sugerida anteriormente por Völker, como podemos observar em (Vol03), onde ele também explora as vantagens e desvantagens deste modelo em comparação com os de objetos distribuídos descritos acima. Nosso protocolo de comunicação seguiu este caminho. Ou seja, especificamos um conjunto de funções esperadas pela interface para seu funcionamento e as implementamos na forma de um web service. Este web service é responsável por traduzir as solicitações da interface em comandos reais na máquina de prova utilizada. Isso nos leva a uma arquitetura fracamente acoplada, onde interface e provador são bastante independentes um do outro.

O web service funciona como um *broker*, intermediando a comunicação entre a interface e o provador. A interface Proof General que apresentamos na seção 4.7 do capítulo 4 utiliza uma ideia semelhante, embora o broker implementado por ela não seja um web service. É um componente executável separado da interface e do provador, o que permite diferentes formas de distribuição física, usando os mesmos padrões de comunicação. Imagine um cenário onde nosso ambiente de prova precise suportar um provador que roda sobre Linux, apoiado por diversas bibliotecas específicas desta plataforma. O provador e o web service que o expõe para a interface podem ser instalados em uma máquina que atenda aos requisitos de sistema mencionados. A interface pode estar hospedada em outra máquina, com outro sistema operacional, bastando ter acesso via Internet a URI (*Universal Resource Identifier*), que

PUC-Rio - Certificação Digital Nº 0711291/CA

localiza o web service na rede. Para o usuário, toda essa complexidade fica escondida, bastando que ele acesse o sistema através do web browser da sua máquina.

O conjunto de operações oferecidas por um web service é especificado usando uma linguagem própria para isto, derivada de XML e denominada WSDL (Wsdl) ou *Web Service Description Language*, também padronizada pelo W3C. O serviço expõe sua definição no formato WSDL para que aplicações consumidoras (no nosso caso, a aplicação web que implementa a interface) entendam como se comunicar com ele. A figura 5.7 mostra uma representação gráfica do conjunto de operações definidos no arquivo WSDL que do nosso protocolo, detalhando as entradas e saídas esperadas. Abaixo apresentamos uma descrição sucinta destas operações.

Start Inicia um processo de prova no provador. Quando o usuário entra com uma fórmula para provar, a interface envia esta mensagem ao web service com um identificador para representar a sessão do usuário. Todas as mensagens trocadas entre interface e web service a partir de então serão identificadas desta forma. Na primeira vez que esta mensagem é enviada, o web service cria uma nova instância do provador que será usada para servir as solicitações do usuário, representado pelo identificador de sessão informado pela interface. O web service inicia o processo de prova no provador e retorna a representação de seu novo estado para a interface.

Step Executa um passo de prova no provador. Ao receber uma mensagem de step, o web service executa um passo de prova na instância do provador servindo a sessão do usuário informada, a partir do estado atual do processo de prova e do objetivo em aberto escolhido. Como dissemos, o Hemera executa a busca por provas no estilo *backward* e nosso protocolo espera provadores deste tipo. O novo estado é retornado.

Run A partir do ponto atual de uma prova, esta mensagem orienta o provador a executar a busca *backward* pela prova até o final. O estado final do processo é retornado, indicando se a prova foi encontrada (nenhum objetivo em aberto) ou não.

CheckSyntax Valida sintaticamente uma fórmula. A interface repassa fórmulas entradas pelo usuário ao web service que usa o provador para avaliá-la sintaticamente.

GetRules Quando o usuário escolhe um objetivo em aberto para aplicar uma regra, a interface envia a fórmula que representa o objetivo para o web service que aciona o provador para identifcar as regras que podem ser

PUC-Rio - Certificação Digital Nº 0711291/CA

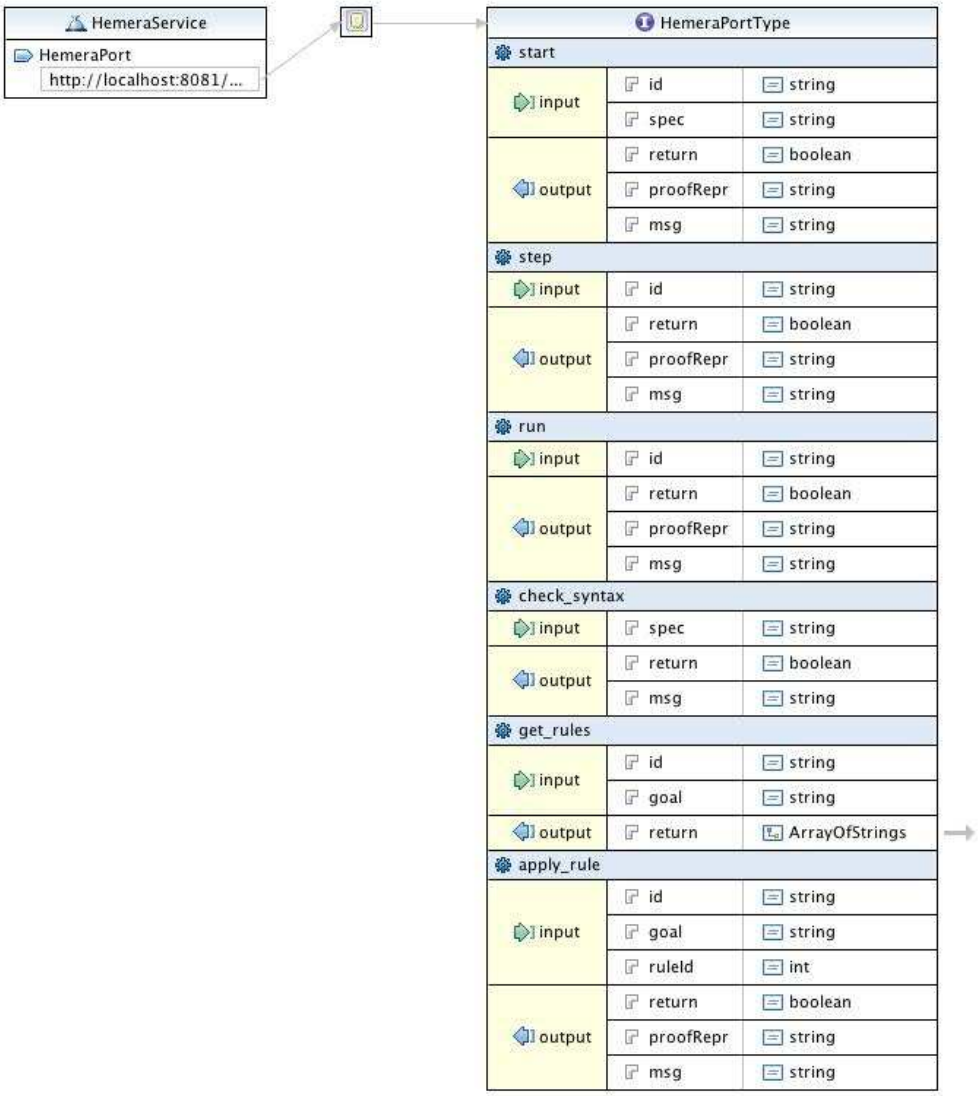

Figura 5.7: Operações suportadas pelo Web Service

PUC-Rio - Certificação Digital Nº 0711291/CA

aplicadas neste ponto. A resposta é tratada pela interface e apresentada para escolha do usuário.

ApplyRule Aplica uma regra. Após o usuário escolher uma regra para ser aplicada em um objetivo em aberto, a interface repassa a regra escolhida e o ponto atual da prova para o web service que aciona o provador para atualizar seu estado, aplicando a regra.

Existem duas formas possíveis de adaptar um provador para funcionar com nossa arquitetura. Primeiro, se o provador provê uma API para interação, basta implementar um web service, de acordo com o protocolo acima, usando as funções da API para manipular o provador de acordo com cada operação. Muitos provadores não fornecem estas bibliotecas para manipulação, apenas interfaces textuais usadas para entrada de dados e comandos diretamente pelo usuário. Esta situação já foi registrada anteriormente por Théry, Bertot e Kahn em (TBK92). Neste caso, é preciso alterar diretamente o código-fonte do provador em alguns pontos para que ele possa funcionar com o web service.

Esta era a situação do provador Hemera, que usamos para desenvolver nosso protótipo. Portanto, para implementar o protocolo acima no Hemera foi preciso modificar parte de seu código. Essas mudanças ocorreram basicamente de duas formas:

- O tratamento de excessão foi alterado. As mensagens de erro do Hemera eram simplesmente apresentadas ao usuário final via console. Para funcionar na arquitetura que propomos os pontos de excessão do provador foram alterados para repassar ao web service um objeto descrevendo o erro ocorrido. Esse objeto é tratado pelo web service e repassado para a aplicação web de forma que essa possa identificar o que ocorreu e escolher a melhor forma de apresentar o problema ao usuário.
- Foi necessário adicionar um controle de identificação de chamada (um *token*) para diferenciar as diferentes sessões de usuário gerenciadas pela aplicação web e pelo web service, tornando o provador, inicialmente monousuário, capaz de atender múltiplas solicitações simultâneas.

Para facilitar essa integração decidimos usar Python, a linguagem do provador, na construção do web service.

PUC-Rio - Certificação Digital Nº 0711291/CA

5.5 Desenvolvimento da Interface Web

Uma vez discutido como implementamos a separação entre interface e máquina de prova através do uso de web services, apresentamos, nesta seção, os detalhes da implementação de nossa interface de prova propriamente dita. Como dissemos, nossa interface foi desenvolvida como uma aplicação web. De forma geral, uma aplicação web é um tipo de aplicação onde o usuário acessa o sistema através de um browser conectado à Internet (ou a uma rede privada baseada nos padrões da Internet, denominadas intranets). O browser renderiza as páginas fornecidas pelo servidor web (ou servidor HTTP), um componente da aplicação que recebe solicitações e as processa, enviando respostas através do protocolo HTTP para o browser cliente. As páginas recebidas pelo cliente são arquivos no formato HTML, que são renderizados pelo browser, exibindo o conteúdo gráfico correspondente para o usuário.

A arquitetura que descrevemos acima para nosso ambiente nos permite implementar a interface de prova com linguagens e frameworks independentes daqueles usados na construção do provador. Atualmente, são muitos os frameworks existentes para o desenvolvimento de software para a web como, por exemplo, ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails, Struts2, entre outros. Frameworks como estes ajudam a diminuir os erros de programação, automatizando tarefas repetitivas, tornando o código escrito mais simples, oferecendo tratamento padronizado para questões como segurança, autenticação, tratamento de sessões, etc., permitindo, desta forma, que o programador concentre seus esforços na resolução dos problemas do domínio e menos nos aspectos técnicos da infra-estrutura. Além de promoverem a qualidade do código produzido implementando boas práticas de programação.

Neste sentido, escolhemos desenvolver nossa interface usando Ruby on Rails (RTH09), um framework de código aberto para desenvolvimento web, baseado na linguagem Ruby, bastante aderente às características descritas no parágrafo anterior. Rails promove o uso do padrão arquitetural MVC (acrônimo das palavras em inglês *Model*, *View* e *Controller*), originalmente proposto no final dos anos 70, na elaboração da interface gráfica do ambiente de programação da linguagem Smalltalk (ver (Bur92)). Embora as primeiras aplicações para web tenham sido desenvolvidas sem um modelo de desenvolvimento predominante, o padrão MVC logo foi redescoberto e, um pouco adaptado, tornou-se a principal forma de se projetar este tipo de aplicação. O MVC promove a separação de conceitos, dividindo a aplicação em três grupos de componentes: os modelos, as vistas e os controladores.

PUC-Rio - Certificação Digital Nº 0711291/CA

Modelo Representa a lógica de negócio da aplicação, implementando dados, regras e restrições do domínio do problema e é responsável por manter o estado da aplicação, persistindo as informações necessárias, tipicamente em bancos de dados.

Vista É responsável por gerar a interface com o usuário, normalmente baseada nos dados do modelo.

Controlador Orquestra a aplicação como um todo: recebem eventos externos representando ações do usuário, interagem com o modelo para processá-las e determinam a vista apropriada que deve ser apresentada ao usuário para exibir o resultado.

A figura 5.8 apresenta como as diferentes partes da arquitetura MVC interagem entre si para implementar uma aplicação web. As ações do usuário são enviadas do browser para o servidor web que as redirecionam para um determinado controlador (1) que interpreta a solicitação e interage com o modelo (2), alterando o estado geral da aplicação. O controlador, então, invoca a vista (3) que deve ser apresentada ao usuário, que produz o código HTML (4) que é devolvido ao browser, onde é renderizado e exibido.

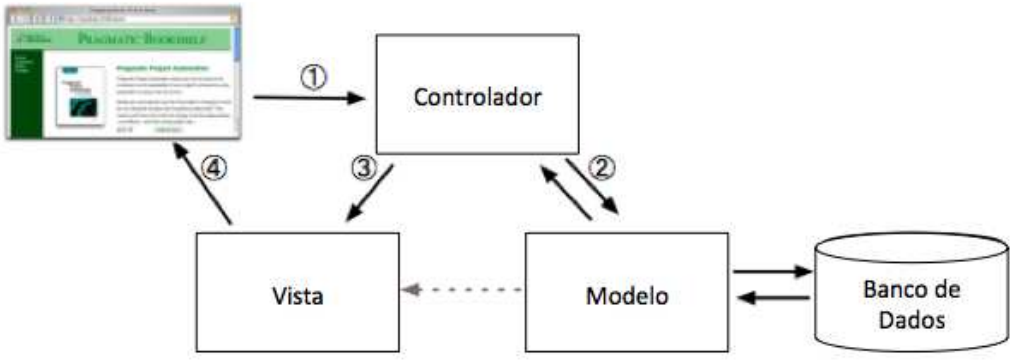

Figura 5.8: A arquitetura MVC

5.6 Modelos de Interação na Interface Web

Uma das principais decisões de design que precisou ser tomada na concepção de nossa interface de prova diz respeito aos modelos de interação que seriam suportados. Como vimos na seção 4.3, embora normalmente enfatizem uma visão mais do que outras, uma característica esperada é que interfaces

PUC-Rio - Certificação Digital Nº 0711291/CA

para provadores ofereçam diferentes formas de interação, ampliando as possibilidades de escolha do usuário.

Achamos que a visão de *prova como programação*, apesar de amplamente difundida na comunidade e de ser a visão predominante nas interfaces para provadores atualmente, como já discutimos antes, dificulta o uso de interfaces de prova por usuários não programadores, menos acostumados com ambientes de escrita de programas e suas funcionalidades. As interfaces PCoq e ProofGeneral, como já apresentamos, são exemplos do uso desta visão.

A visão de *edição de estrutura* também tem limitações. Nesta visão o processo de prova é conduzido pela manipulação direta do objeto de prova, normalmente alguma representação em árvore. Muitos argumentam que, apesar de interessantes para provas pequenas, principalmente do ponto de vista didático, esta visão não é adequada quando lidamos com provas grandes. Entretanto, os aspectos cognitivos realçados por esta visão a tornam uma solução promissora para ambientes que pretendam valorizar um público mais amplo de usuários, além daqueles ligados às áreas de computação. Acreditamos que esta desvantagem no trato de provas grandes pode ser contornada se a interface propiciar maneiras de apoiar o usuário na manipulação da prova. As interfaces Jape e PVS são exemplos de interfaces que seguem este modelo de interação.

A visão de *prova por apontamento* pode ser combinada bem com as duas visões anteriores.

Na versão atual do HemeraWeb priorizamos o modelo de prova como edição de estrutura. Um de nossos objetivos secundários com a interface é utilizá-la como ferramenta de apoio em aulas de Lógica, cenário no qual tal visão se justifica mais. Fornecemos também características da visão de prova por apontamento, com algumas adaptações, integrada com a maneira como implementamos a edição de estrutura. Em nosso caso, o usuário conduz o processo de prova escolhendo fórmulas e decidindo qual regra aplicar a elas, através de uma lista de regras possíveis inferida a partir da fórmula escolhida. Na implementação original deste modelo de interação, proposta por Bertot, Kahn e Théry, a regra a ser aplicada é escolhida automaticamente pelo processo de provar por apontamento, de acordo com a parte da fórmula (que representa um objetivo em aberto) clicada, conforme o mecanismo de decisão apresentado em (BKT94). Apenas inferindo as regras possíveis em um ponto da prova e deixando a escolha da regra para o usuário, permite que o usuário experimente diferentes caminhos na condução da prova, o que apóia o processo de aprendizado, principalmente de usuários iniciantes em Lógica. Em versões futuras desejamos implantar o suporte à visão de prova como programação, para que sirva como opção para os usuários, principalmente os

PUC-Rio - Certificação Digital Nº 0711291/CA

mais experientes.

A figura 5.9 mostra a área principal do editor de estrutura do HemeraWeb. Na imagem vemos o resultado final de um processo de prova, representado por sua árvore de estados. Em nossa interface, a cada etapa da prova, o usuário manipula diretamente esta representação da maneira descrita no parágrafo anterior. A árvore de estado da prova segue o padrão descrito na seção 4.2.

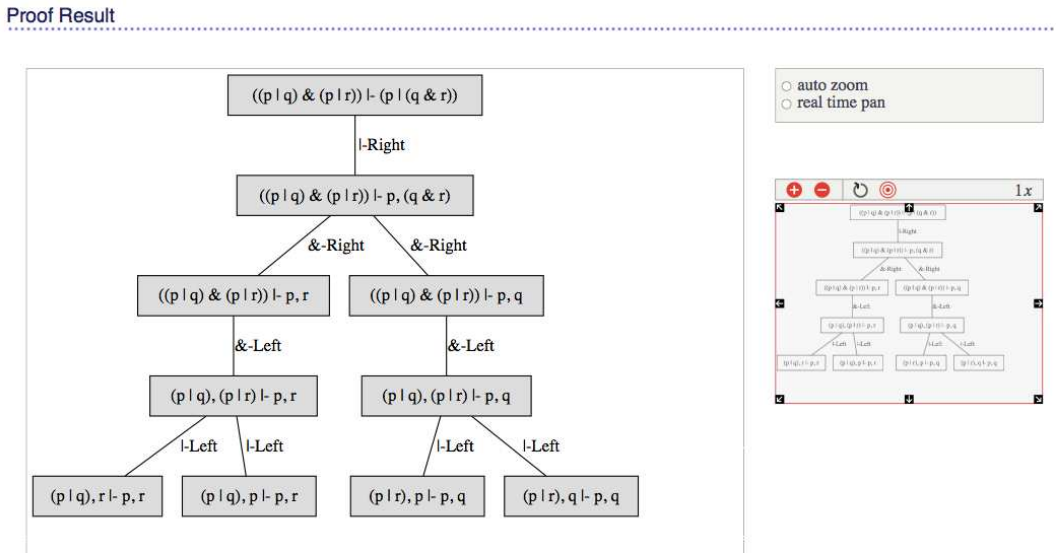

Figura 5.9: Árvore de Estado da Prova no HemeraWeb

Como descrevemos na seção sobre arquitetura, no início deste capítulo, para prover esta forma de representação em uma interface web utilizamos o formato SVG. Como vimos, o provador devolve o estado da prova para o web service em um determinado formato particular de representação usado. Um componente no web service, traduz este formato para SVG, que é, na verdade, uma linguagem derivada de XML. O web service, então devolve para a aplicação web a representação da prova no formato SVG. Na aplicação web, seguindo o mecanismo MVC, um controlador prepara a vista apropriada que vai exibir a imagem do estado da prova no browser cliente.

Como SVG é um formato XML, é possível manipulá-lo no browser através de código JavaScript utilizando a API DOM, permitindo programar recursos de interação com a imagem. É usando esta técnica que permitimos ao usuário clicar em partes da prova e aplicar ações de acordo com a parte selecionada. Arquivos CSS completam as tecnologias usadas para implementar nosso editor de estrutura. Definições CSS associadas a funções JavaScript permitem mudar a aparência da representação de acordo com ações executadas. As páginas devolvidas pela aplicação web para o browser referenciam estes arquivos que

PUC-Rio - Certificação Digital Nº 0711291/CA

são baixados pelo browser e usados para dar ao usuário a experiência de interação que projetamos.

5.7 Scripts de Prova

Em ambientes baseados em prova como programação, scripts de prova são elementos de primeira importância. O usuário escreve scripts de prova que são submetidos ao provador, conduzindo-o na busca pela prova. Vimos que em provadores no estilo *backward*, o script de prova pode ser extraído do resultado final do processo de prova, podendo ser reexecutado posteriormente a qualquer instante.

Devido a ênfase inicial do projeto na visão de edição de estrutura, nossa interface não suporta scripts de prova como normalmente eles aparecem em outros ambientes. No HemeraWeb scripts são o ponto de entrada para o processo de prova. Usuários iniciam um processo de prova, criando um novo script, especificando seu objetivo inicial e iniciando a execução da prova no provador (através do web service).

A aplicação web é responsável por manter o script de prova entre sessões do usuário. Ela oferece recursos para que o usuário possa criar, editar, apagar, consultar e executar seus scripts. Para isto é usado um banco de dados MySql, onde o sistema persiste os objetos do modelo que representam o estado dos scripts do usuário. Assim o usuário pode parar um processo de prova antes de finalizá-lo, salvá-lo e voltar a ele mais tarde, quando o sistema como um todo recupera o estado de execução no provador permitindo que o usuário continue do ponto onde parou.

Nossa intenção é, no futuro, permitir extrair o script na forma tradicional feita por outras ferramentas. Esta seria uma visão complementar a visão oferecida atualmente. Isto exige editores de prova mais sofisticados, que compreendam a linguagem do provador sendo usado e ofereçam, por exemplo, recursos de *syntax highlight* e *code complete* como forma de tornar esta visão mais amigável aos usuários.

A figura 5.10 apresenta a página do HemeraWeb de onde o usuário acessa todos os seus scripts. A página apresenta o objetivo inicial que está sendo provado e o estado atual do processo de prova associado.

Um script de prova pode assumir diferentes estados durante seu ciclo de vida. A figura 5.11 mostra o diagrama de estados de scripts de prova, indicando as ações no sistema que provocam a mudança destes estados.

PUC-Rio - Certificação Digital Nº 0711291/CA

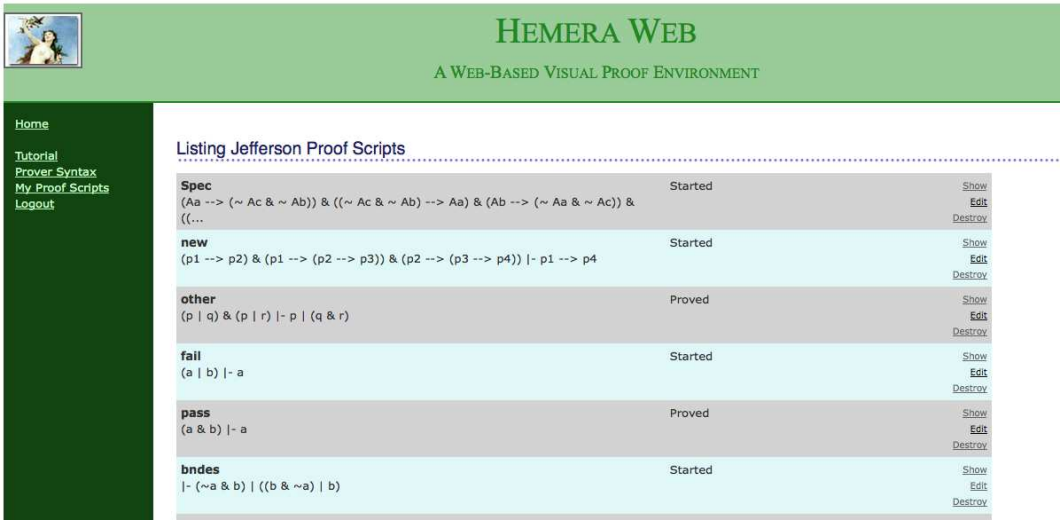

Figura 5.10: Biblioteca de Scripts de um Usuário no HemeraWeb

PUC-Rio - Certificação Digital Nº 0711291/CA

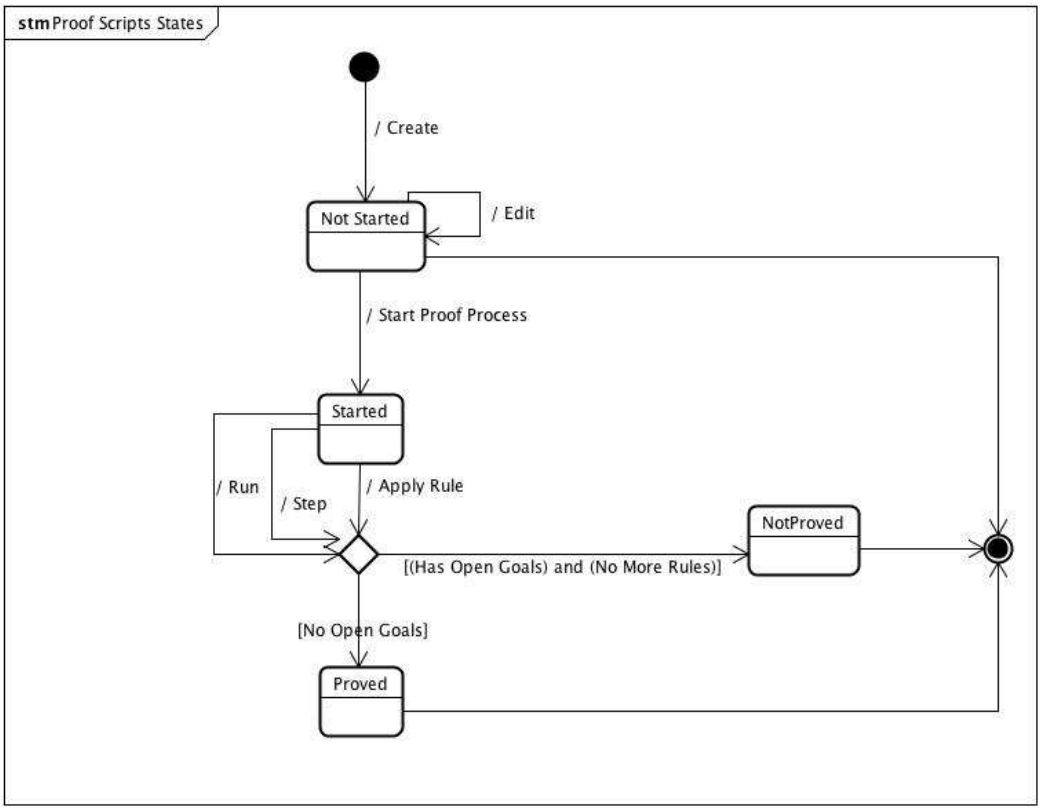

Figura 5.11: Estados dos Scripts de Prova