Universidade Federal de Pernambuco

Graduação em Ciência da Computação
Centro de Informática

Ferramenta composição de circuitos em proposições lógicas

Trabalho de Graduação

Aluno: Glerter Alcantara Sabiá (gas2@cin.ufpe.br)

Orientador: Alexandre Cabral Mota (acm@cin.ufpe.br)

Recife, 02 de novembro de 2008.

Agradecimentos

Eu gostaria sinceramente de agradecer ao Centro de Informática da UFPE por toda minha graça de titulação universitária alcançada. É sem dúvida uma escola dos sonhos com toda sua pompa de aparatos técnicos e gestão eficiente. Talvez nunca seja aluno de alguma outra comparável. Mas não há nisso uma perda considerável. Há coisas que não se aprendem em escolas. E elas me interessam muito.

Obrigado todos pertencentes a melhor idade, em especial ao juiz do trabalho André Machado, que embora não me conheça mostrou-me que não seria ruim viver muito quando se pode planejar tornar-se senil e sábio.

Obrigado Alexandre Mota por ter me feito extrair um último fôlego para constituição desse trabalho e conseqüente conclusão do curso.

Obrigado Auristela pelas aulas de francês, de humildade e coerência.

Obrigado d'Artagnan por ter me sido amigo sincero, creio que houvera exceção em meio a toda a mortandade ética persistente em nossa área.

Obrigado à minhas amigas que tornaram minha vida no atroz Recife mais feliz. Neidinha, Luana e Amanda.

Obrigado Samuel por ter me permeado o intelecto como só gente como a gente sabe fazer.

Obrigado ao GRITA por co-existir.

Obrigado à VonTrapp pelo amor vulgo incondicional.

Embora me sinta resignado ou repeso nunca é tempo perdido.

Sumário

1. Introdução

No processo de engenharia de requisitos utilizado pela EMBRAER e mais especificamente no processamento de sensores aéreos, existe uma descrição gráfica de circuitos lógicos a qual é encarada como um requisito de baixo nível. Então, pode se dizer que, essa descrição é também parte de uma descrição de como os sensores devem ser processados a fim de que se extraia uma informação relevante e de mais alto nível. Por conseguinte, e embora não usual na engenharia de sistemas tradicional, faz sentido que essa descrição gráfica seja tratada como um requisito.

Sabe-se que o suporte a mudanças nos requisitos exige um esquema robusto para controlar a integridade [4] de cada requisito e do conjunto como um todo. Foi essa exigência, que é consideravelmente relevante e não estava sendo eficientemente atendida, que viabilizou esse trabalho. Até então, no processo de requisitos da EMBRAER o projeto de um circuito lógico dava-se através de ferramentas capazes de desenhá-lo. Por conseguinte, essa imagem fazia parte do documento de requisitos. Através desses passos, é notável o engessamento do controle de mudanças dos requisitos. Engessamento esse, devido principalmente ao re-trabalho de re-projetar todo o circuito lógico.

Sem desconsiderar todos os procedimentos necessários dentro de um ambiente de engenharia formal de sistemas como é o caso da indústria aeronáutica, vale salientar que o re-projeto de um circuito lógico (que estaria gravado arquivado como imagem) exige o trabalho de analistas que o projetem, o calculem, o validem e que revisem esse trabalho. Embora o circuito em discussão seja uma linguagem visual, a dificuldade de calculá-lo (extrair dele a proposição lógica correspondente) cresce exponencialmente em função do seu tamanho. Por fim, se todo esse re-projeto for necessário para cada mudança, mesmo as menos significativas, teremos um ambiente dispendioso e pouco eficiente.

1.1 Contribuições

Esse trabalho se propôs a construir uma ferramenta que mudasse as formas de anexação de um circuito lógico num corpo de requisitos utilizando um formato de descrição a partir da XML. Em outras palavras, concebeu-se um esquema XML capaz de representar um circuito lógico e uma ferramenta que manipule esse XML e construa e altere graficamente o circuito lógico. Embora a ferramenta seja ainda um protótipo, (veja sugestões de trabalhos futuros) ela surge com a esperança de mais agilidade dentro do processo de construção e refinamento de tais circuitos lógicos e, por conseguinte de mais entrosamento e confiança nas relações entre EMBRAER e CIn-UFPE. Em síntese produziu-se:

- Um editor gráfico de Circuitos Lógicos
- Um gerador da proposição lógica correspondente a um circuito lógico.
- Um esquema de documento XML para representar Circuitos Lógicos.
- Um módulo de importação e exportação do documento XML correspondente a um circuito no editor gráfico.
- Um formato independente para persistência das posições dos elementos de um circuito lógico.

1.1 Organização deste trabalho

Esse documento está apresentado em quatro partes (que compreendem as seções 2, 3, 4, 5). A seção 2 discorrerá sobre a tecnologia XML focando apenas nos aspectos que influenciaram o projeto da ferramenta de maneira a justificar seu uso. A seção 3 enfoca como se deu o mapeamento de circuitos em XML. Na seção 4 apresentamos a ferramenta em si. E na Seção 5 apresentamos o estudo de caso na EMBRAER.

2. Background

Antes de qualquer coisa, faz-se necessário esse preâmbulo sobre XML a fim de justificar o uso dessa tecnologia no presente trabalho. Essa seção também abordará a biblioteca que a ferramenta (também fruto desse trabalho) utiliza para manipular XML.

Isso se faz necessário, a fim de manter o foco na compreensão exatamente dos aspectos da XML que foram determinantes para a sua escolha como solução. Em outras palavras não se procurou conceituar a linguagem profundamente, uma vez que, embora se veja que o uso da XML fora decisivo para esse trabalho, a possível aplicação da XML tem um conjunto de vantagens consideravelmente amplo, e aquilo que não tange o escopo do presente trabalho naturalmente não foi abordado.

2.1 XML e Extensibiliade

XML significa: linguagem de marcadores extensível. Sua sintaxe se assemelha à popular HTML uma vez que ambas tiveram sua origem a partir da meta-linguagem SGML (criada pela IBM e atualmente descrita na ISO 8879:1986).

Enquanto a HTML é apenas uma adaptação da SGML, XML (embora também o seja) foi pensada como um subconjunto da SGML, mantendo um pouco de sua capacidade de extensão e uma gramática mais enxuta para tornar-se mais simples de ser analisada. Na comparação entre as três temos que:

- A HTML (que não é extensível) tem uma gramática pequena o suficiente para ser (e vem sendo como é o caso de XHTML) encarada como apenas um possível esquema XML, já que possui não só um propósito específico (marcação de hipertexto na *web*), mas também todos os seus marcadores predefinidos;
- Enquanto a SGML é mais extensível que a XML até mesmo em matéria de sintaxe, onde, embora sejam padrões, os caracteres ‘<’ e ‘>’ não são obrigatórios.

A XML é extensível porque, mesmo a sintaxe sendo inflexível (caracteres para marcação, tipos, etc.) é possível definir os nomes de marcadores bem como o modo como ficarão aninhados. Além disso, vale também não desconsiderar as características evidentes de qualquer linguagem de marcação que permitem a construção de estruturas hierárquicas e também referenciá-las semanticamente umas com as outras num mesmo documento.

Continuando, a XML não é apenas uma linguagem capaz de descrever quaisquer documentos ou dados, (devido a sua extensibilidade inerente), mas também, por utilizar-se de uma descrição estruturada, é capaz de ser facilmente: transportada, validada, transformada e processada. Dessa maneira, é presumível falar em esquema de dados e nos próprios dados que embora sejam conceitos diferentes, ambos os elementos são descritos em XML. Essa capacidade da XML para tratar instâncias e esquemas igualmente torna o seu uso uma revolução comparável à iniciada por Von Neumann quando protocolou o armazenamento de programas de maneira lógica assim como os dados eram armazenados [1]. Para concluir, essa capacidade da XML também é responsável por tornar a validação (bem como as transformações) uma simples interação entre documentos XML.

A XML é hoje o maior facilitador de integração de dados na dinâmica da tecnologia da informação atual. Atua como uma cola dentro das soluções de integração de dados ou ainda, como um mínimo denominador comum das diferentes plataformas à medida que ela transporta e converte os dados.

2.2 Modelagem de Objetos em XML

Se em XML pode-se descrever tanto a instância como o esquema de dados, a tarefa de armazenar objetos complexos (aqueles que podem conter (referenciar) outros objetos) em XML torna-se no mínimo não trivial. Como pode ser percebido, existem inúmeras maneiras de codificar o objeto em XML.

De fato as decisões do que deve ou não ser hierarquizado ou ainda o que deve ou não ser mapeado como atributo, têm repercussões que devem ser antevistas e trabalhadas de modo a garantir a eficiência do processo pelo qual essa representação XML do objeto passará. Pois na grande maioria dos casos (e esse trabalho não faz exceção), a representação em XML não surge para

substituir o objeto dentro da linguagem de programação, pelo contrário é uma extensão, é uma representação dele para um propósito especifico.

Essa responsabilidade foi salientada aqui a fim de que posteriormente seja possível explicar as decisões de projeto a cerca de como deveria ser o esquema do documento XML utilizado. Na realidade, esse é não é um tema irrelevante e, já existem trabalhos que auxiliam quem se depara com essa dificuldade. Como pode ser constatado [2] foi um artigo pioneiro nesse tema e contribuiu para esse trabalho.

2.3 A biblioteca XStream®

Com todas essas possibilidades inéditas, era de se esperar que as linguagens de programação oferecessem suportes à modelagem e processamento de documentos XML sem maiores esforços para o programador. Na criação dessa ferramenta utilizou-se a linguagem Java bem como uma biblioteca um tanto conhecida para tratamento XML, a XStream® [3].

A biblioteca XStream implementa um conceito bem similar a serialização de objetos porém em formato XML. Assim tornou-se quase que trivial, a geração de XML a partir de um objeto Java.

A biblioteca permite controles parametrizados para definir a estrutura do documento (os mapeamentos: o que seria mapeado em marcador o que seria em atributo, o que seria aninhado etc.). Dessa maneira o seu uso oferece a possibilidade de projetar todo o esquema do documento através de uma interface única.

Percebe-se então, que a biblioteca possui pretensões bem definidas e pode até não ser considerada robusta quando comparada aos analisadores XML complexos como o Apache Xerces® ou às outras implementações que seguem os padrões SAX e DOM, uma vez que estes são de propósitos mais gerais.

Apenas para que se torne mais concreto o entendimento do funcionamento da biblioteca a tabela abaixo ilustra os métodos principais da fachada que protocola a conversão parametrizada de XML em objeto.

new XStream()	Construtor da fachada.
alias(m,c)	Associa um nome de marcador a uma classe.

toXML(o)	Converte o objeto em XML.
fromXML(x)	Reconstrói o objeto a partir do XML.
useAttributeFor(c,n)	Mapeia como atributo XML uma classe e um nome de atributo java.

Existe uma série de outros controles relevantes, por exemplo, para lidar com coleções (limitadas, e ilimitadas) e herança, que foram utilizados no projeto. O completo conhecimento dos recursos oferecidos pode ser adquirido através da análise de [3].

Como será visto em seguida, para a resolução do problema proposto nesse trabalho era preciso escolher um formato para persistir um objeto (um circuito lógico) de modo a lidar com seus estados inconsistentes e consistentes. Por fim também era necessário que o formato fosse convertível para outras ferramentas (ou formatos) dentro do processo de engenharia de requisitos. Então, utilizar a XML na concepção de um esquema para esse tipo de objeto, foi uma escolha natural.

3. Circuitos em XML

O processo de engenharia de requisitos da EMBRAER é auxiliado por uma série de ferramentas e não faz parte do escopo desse trabalho detalha-las, mas se faz importante entender que a codificação do circuito através da XML se deu também no intuito de propiciar a integração com essas outras ferramentas (Em especial o Telelogic Dooors®[6]). De fato, a representação em XML do circuito deu condições não só para a nossa ferramenta processá-lo (e assim diminuir o esforço para remodelá-lo) como também para quaisquer outras já coadjuvantes do processo o fazerem. Em conclusão o uso de XML nesse caso tenta diminuir os impedimentos e assim garantir a aderência de nossa ferramenta ao processo.

Antes de alguma explicação a cerca da concepção do modelo XML do circuito é importante como foi mencionado anteriormente, que se compreenda qual o propósito dessa representação XML. Para tanto a seguir introduz-se alguns requisitos da ferramenta que estão ligados a esse propósito.

3.1 O Propósito do XML

A ferramenta fruto desse trabalho proveria um ambiente para a construção visual do circuito, onde ele pudesse ser calculado automaticamente. E também manipularia um formato (no caso XML) para persistir o circuito de maneira a permitir que o trabalho de construção ou refinamento circuitos fosse produtivo.

Não há grandes embaraços quando se fala de calcular a fórmula do circuito, pois a capacidade de analisar a árvore de marcadores existe para qualquer XML. E o cálculo em questão seria possível através dessa operação. Porém a persistência de um circuito envolve problemas de inconsistência que têm de ser levados em consideração. Ou seja, deve-se que lidar com os estados (consistentes e inconsistentes) de um circuito de modo que a representação XML deva tentar impossibilitar a composição de inconsistências e ao mesmo tempo propiciar a persistência dos estados não finalizados. Entenda-se para circuito não finalizado: Um circuito consistente, mas ainda incompleto.

3.2 O Circuito

Um circuito pode ser considerado como: um conjunto de portas lógicas, um conjunto de entradas, uma saída e um conjunto de ligações ou alimentações entre as portas ou entradas. Um circuito não pode estar com uma de suas portas alimentando a si mesma diretamente ou recursivamente, em outras palavras, não pode existir um ciclo no circuito. E ainda um circuito poderá estar persistido de maneira não finalizada (isso é necessário visto que alguns circuitos são demasiado grandes para serem desenhados numa sessão de trabalho): ou seja, ainda se faltarem portas ou ligações (o que impossibilita o cálculo da proposição lógica) o circuito pode ser considerado consistente.

Esses aspectos, necessários à manutenção da integridade da entidade circuito na representação XML, deixam claro que se faz necessário:

- Ou, um processamento para detecção de ciclos;
- Ou, formatos diferentes para circuitos finalizados e não finalizados.

A princípio, pode se pensar que a alternativa de utilização de dois formatos traga simplicidade através da fuga do problema da percepção de ciclos, infelizmente não é bem verdade e não se optou por essa alternativa por duas razões.

Em primeiro, não haveria como fugir do problema de ciclos, ele continuaria a existir no formato para circuitos não finalizados. Pois, é possível a existência de ciclos num circuito não finalizado. Em segundo, pensou-se que o uso de dois formatos estaria semanticamente incorreto. Os circuitos, finalizados ou não, cumprem o mesmo propósito, na verdade (não esqueça) ambos são requisitos de um sistema, e por definição são incompletos, podendo ser remodelados a cada iteração. Representações diferentes não fariam sentido para o mesmo propósito. Em conclusão os possíveis remodelamentos (refinamentos) que ocorrem naturalmente no conjunto de requisitos tornam efêmeros: o conceito de “finalizado” e, por fim, a diferença entre circuitos finalizados e não finalizados.

Isto posto, durante a fase de concepção de um modelo XML que contemplasse essas necessidades, primeiro pensou-se no melhor desempenho

para a verificação de ciclos e de finalização e para a geração da proposição lógica. Segue um exemplo dessa versão inicial:

```
<circuit>
  <ports>
    <input-port out="1" />
    <input-port out="2" />
    <and-port out="3" inp1="-1" inp2="-1" />
    <or-port out="4" inp1="1" inp2="3" />
    <and-port out="5" inp1="2" inp2="4" />
    <not-port out="6" inp1="2" />
  </ports>
  <output port="5" />
</circuit>
```

Figura 1

Notavelmente essa representação pouco se preocupou com a compreensão humana (diga-se legibilidade). A marcação “ports” compreende a coleção de componentes do circuito (portas e entradas). Para cada tipo de porta existe um tipo de marcação e todas elas possuem como atributo no mínimo o “out” representando um número único para a saída da porta. Como cada componente possui apenas uma saída o “out” acaba por ser também um identificador único da porta ou da entrada. Existem também, dependendo do componente, atributos que determinam os pontos de alimentação (como “inp1”, “inp2” etc.). Eles representam números que fazem referência ao “out” de outras portas.

Então por exemplo, se uma porta tem o atributo “inp2” é igual a 3, o componente que possua o “out” valor 3 está a alimentar uma das entradas da porta em questão. Nota-se ainda, que existem atributos associados a valores negativos, essa é a notação utilizada para dizer que nenhuma porta está ligada ao ponto de alimentação, ainda. Ao se compreender que os atributos numéricos guardam essa relação entre si, percebe-se que além de englobar as portas lógicas

e as entradas, a marcação “ports” também engloba o conjunto de ligações entre esses componentes (os fios do circuito).

Por fim a marcação “output” tem um atributo “port” que indica qual a saída de porta que seria a saída do circuito. Existiria uma e apenas uma marcação desse tipo. E também, vale a convenção: se o atributo for negativo significa que nenhuma porta ainda, foi indicada como a saída do circuito.

A seguir serão elucidadas as impropriedades dessa versão inicial, a fim de explicar tanto o refinamento ocorrido como a versão final do modelo resultante desse refinamento.

3.2 O Refinamento do Modelo de Representação XML

A Figura 1 apresenta um modelo de XML visivelmente inapropriado, ele foi concebido sem auxílio de alguma metodologia mais precisa, e manteve-se com o foco no processamento de ciclos e cálculo da proposição lógica. À medida que suas limitações vieram à tona tornou-se evidente a necessidade de pesquisar metodologias para a concepção de modelos XML. Antes de citar essa pesquisa não é demais apontar as limitações nesse modelo inicial.

Tem-se um modelo bastante matemático que de fato, provê um algoritmo de baixa ordem para processamento de ciclos (através do feixe de transitividade aplicado aos conjuntos discretos de pares: quem alimenta quem). Mas, ademais disso, não haveria mais vantagens evidentes pelo contrário.

O principal problema é o entrelaçamento das coleções de entidades diferentes. O entrelaçamento denigre a capacidade estrutural da XML diminuindo a expressividade. E no caso modelo inicial todas as três coleções (portas, entradas, ligações) estão armazenadas, como foi visto, na mesma marcação. Na prática, isso encarece o processamento de atividades como introduzir ou retirar elementos das coleções, ora, esse tipo de atividades é bastante comum no ambiente de criação e refinamento dos circuitos. O desempenho que se ganharia com uma percepção eficiente de ciclos seria suplantado facilmente.

Em seguida tem-se que a pouco expressiva capacidade estrutural restante no modelo ainda deixa a desejar nas restrições de integridade. O modelo proposto é muito frouxo, permitindo que muitos estados inconsistentes possam ser

persistidos (além da inconsistência por ciclos que se concluiu ser de fuga inevitável). Não há como estabelecer restrições para os valores numéricos desses atributos. Por exemplo, é possível que uma porta seja alimentada por outra que não existe. A XML consegue oferecer um controle de unicidade para certo atributo dentro de um escopo de marcação. Mas o que se fez foi trazer a responsabilidade do mapeamento da entidade “Alimentação” para o campo semântico. Deixando de aproveitar vantagens da XML caso “Alimentação” fosse mapeada como uma estrutura (marcação).

Tem-se ainda, que as entidades estão modeladas inapropriadamente dentro do XML. Num circuito as portas lógicas têm a propriedade comutativa isto é, não há diferenças entre os pontos de alimentação de uma porta. Mas o modelo diferencia os pontos de alimentação e, portanto, armazena uma informação irrelevante. Embora o armazenamento de informações a mais possa aparentar-se inofensivo, ele gera a possibilidade de que dois documentos semanticamente diferentes (na semântica de XML) representem o mesmo circuito. Existe uma contrapartida nesses casos, pois mesmo a informação sendo irrelevante, não se exaure a responsabilidade de mantê-la consistente. Na prática, fica mais trabalhoso encontrar todas as portas que são alimentadas por determinado componente. Ratificando, esse problema é fruto do descompasso semântico entre a linguagem XML e o esquema XML do circuito. A se determinar um esquema XML, deve-se buscar mapear cada representação de instância em documentos com semântica XML diferentes.

Por fim, tem-se que as coleções estão também modeladas inapropriadamente, como analogia, é como se não houvesse herança em dentro da modelagem orientada a objetos. Cada tipo de porta lógica é mapeado em uma marcação diferente, mas elas devem ser todas processadas de maneira semelhante. Vale se ressaltar que, cada marcação (tipo de porta) ainda contém seus próprios atributos (e todos são obrigatórios) eles indicam a quantidade de entradas daquele tipo de porta. Esse problema além de constituir um mau aproveitamento é também um engessamento para extensões. Por exemplo, ao surgir necessidade de mapear um novo tipo porta lógica (um AND com três

entradas) o esquema deverá ser alterado para prever uma marcação, que signifique essa porta e possua os respectivos atributos de suas três entradas. E quando o esquema tem de ser alterado para introduzir um conceito que faz parte da representação (circuito), é sinal de que o modelo utilizado não está satisfazendo as necessidades. O modelo tem que comportar qualquer circuito lógico.

Essas incapacidades do modelo, descritas da maneira acima, só foram enumeradas definitivamente quando se decidiu pesquisar como se dá a concepção de um modelo XML mais aderente ao nosso trabalho. E para isso, a contribuição de [5] foi imprescindível de modo que, sem ela esse trabalho estaria comprometido. Seguiu-se então, algumas indicações dessa contribuição para remodelar a representação e com isso, se propuseram alguns modelos intermediários até que se chegasse ao modelo final que apresentamos abaixo.

```
<circuit>
   <inputs>
      <input>
          <name>in_4</name>
      </input>
      <input>
          <name>in_5</name>
      </input>
      <input>
          <name>in_6</name>
      </input>
   </inputs>
   <ports>
      <port class="and2">
          <name>and_1</name>
      </port>
      <port class="not">
          <name>not_3</name>
      </port>
      <port class="or3">
          <name>or_2</name>
      </port>
   </ports>
```

```
  <links>
    <association>
        <device>and_1</device>
        <feeder>not_3</feeder>
        <feeder>or_2</feeder>
    </association>
    <association>
        <device>not_3</device>
        <feeder>in_4</feeder>
    </association>
    <association>
        <device>or_2</device>
        <feeder>in_4</feeder>
        <feeder>in_5</feeder>
        <feeder>in_6</feeder>
    </association>
  </links>
  <output>and_1</output>
</circuit>
```

Figura 2

3.3 O Modelo de Representação XML

O modelo final, que pode ser visto na Figura 2, é completamente estruturado, melhor inteligível, e ainda permite o uso do feixe transitivo na verificação de ciclos apenas acessando a marcação “links”.

Nessa representação, têm-se marcações para:

- O conjunto de entradas: onde cada entrada tem um nome;
- O conjunto de portas: onde cada porta também tem um nome e tem um atributo que indica qual o tipo de porta;
- O conjunto de ligações (ou alimentações): onde cada elemento (association) engloba o nome do componente e uma lista de alimentadores desse componente;
- A saída: a qual indica o nome da porta, cuja saída aponta para o fim do circuito.

Além de apresentar sanados os problemas da seção anterior, o modelo final possui uma capacidade de adaptabilidade considerável. Em primeiro lugar, o tipo

da porta é especificado como um atributo e fica independente do esquema. O tipo de porta indica também a proposição lógica dessa porta, que será usada na geração da proposição lógica do circuito. Em segundo, não existem limites sintáticos para o número de alimentadores que uma porta pode ter (entradas). Então, agora existe suporte para o surgimento de portas lógicas complexas fruto até mesmo de outros circuitos. Quando se compara os dois extremos (o modelo inicial e o final) vê-se que o modelo final está muito mais adaptado à situação dos circuitos incompletos sem necessitar de convenções forçadas.

É importante que fique patente que cada modelo de representação tem suas peculiaridades e não convém, por exemplo, classificar a versão final como melhor que a versão inicial. O que se tem é que a versão final é mais condizente com as necessidades desse projeto. É como se cada modelo tivesse algumas limitações e vantagens. Por exemplo, se, listar todas as portas alimentadas por determinada porta, fosse uma atividade crítica o modelo final não seria a melhor escolha, pois essa informação não é disponibilizada facilmente.

A seguir tratar-se-á mais a cerca da ferramenta que por agora pode ser entendida como um sistema que desenha graficamente um circuito permitindo sua construção e alteração visual. E essa construção e alterações estão sincronizadas com o documento XML correspondente ao circuito. Isto é, um editor visual para esse modelo de XML. Quando o circuito está finalizado a ferramenta gera a proposição lógica correspondente em formato textual.

```
!in_4 & (in_4 | in_5 | in_6)
```

Figura 3

Como exemplo de proposição textual tem-se a Figura 3. Ela é a proposição lógica textual corresponde justamente ao circuito lógico em XML na Figura 2 gerada pela ferramenta.

4. A Ferramenta

A ferramenta será descrita em duas partes: a primeira lida com a manipulação do XML e a segunda com o desenho gráfico. Essa divisão não se deu apenas para organização desse texto, ela guarda algumas relações com a engenharia do sistema. O diagrama abaixo (Figura 4) concebido no início do projeto da ferramenta abaixo elucida a divisão praticada nessa seção.

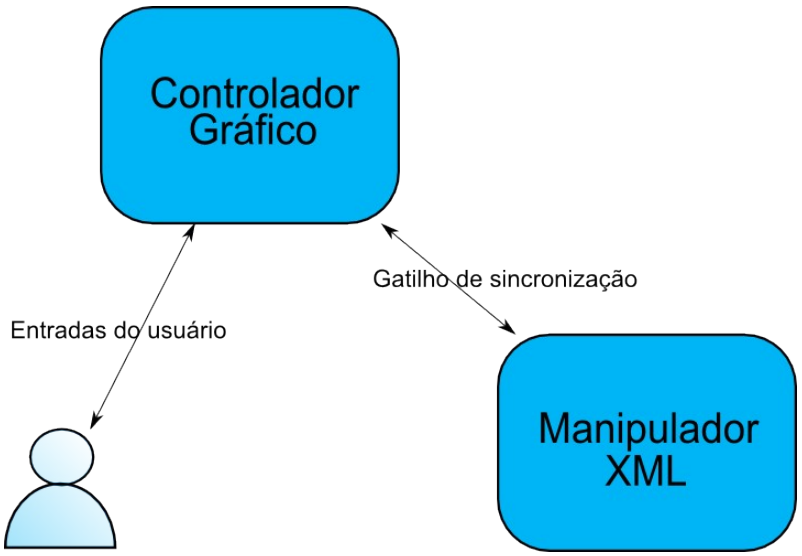


Figura 4

O sistema funciona a partir da interação com o usuário. Ele capta as estradas do usuário e devolve as saídas através de um controlador gráfico. O manipulador do modelo XML atua como um ouvinte de eventos do controlador gráfico. Toda vez que ocorrer uma alteração significativa na representação gráfica (e isso representa um evento captado pelo manipulador) um gatilho de sincronização é disparado. Esse gatilho é o cliente do subsistema manipulador de XML e solicita as alterações correspondentes.

4.1 O manipulador XML

O manipulador do modelo XML foi desenvolvido com auxílio da biblioteca XStream. Como visto anteriormente a biblioteca provê uma fachada capaz de transformar um objeto Java num texto XML. Portanto, existe uma série de classes-

base que descrevem: cada tipo de porta, a entrada, as alimentações de uma porta e a saída. Há também classes para a coleção de portas, coleção de entradas e para a coleção de ligações. Por fim, uma classe que representa o circuito. Essa classe tem como atributos as coleções citadas anteriormente e uma saída. E, é um objeto dessa classe que será transformado em XML. Portanto essas classes chamar-se-ão doravante classes de geração.

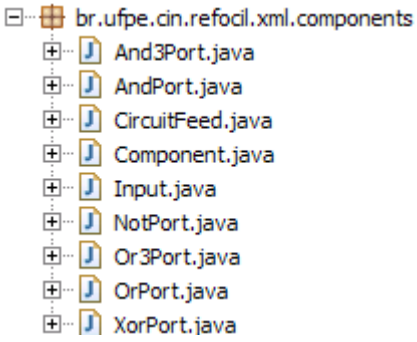


Figura 5

Na Figura 5 pode-se conferir as classes básicas de geração. Todas as classes listadas são filhas de "Component.java" com exceção de "CircuitFeed.java" que representa o conjunto de alimentações de uma porta.

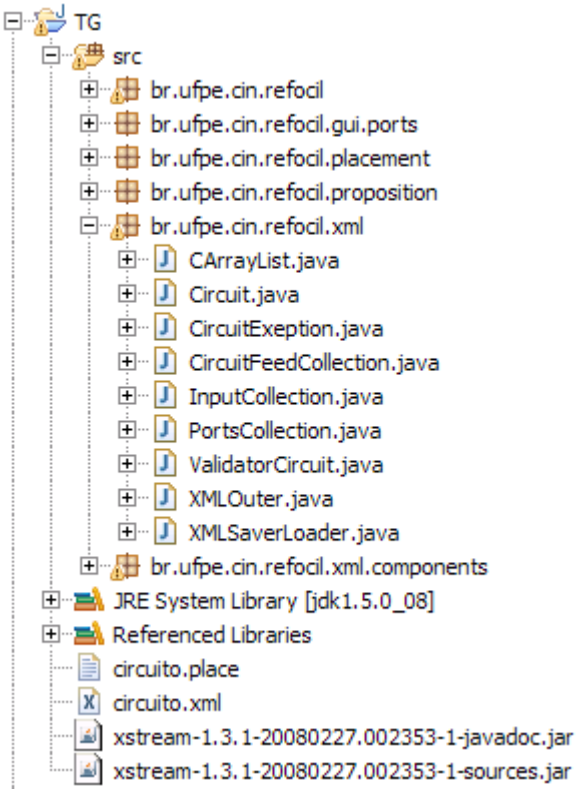


Figura 6

A Figura 6 mostra as demais classes de geração no pacote XML: As classes que representam coleções de entidades e a classe Circuito. As classes restantes são de suma importância:

- Uma classe de exceção para o caso de tentativa de criação de ciclo
- Uma classe para a validação, que engloba os algoritmos de verificação de ciclo e de finalização de circuito.
- Duas classes (últimas classes a contar de cima para baixo) são responsáveis diretas pela geração do documento XML.

Seguindo o raciocínio, vamos transcrever um pouco do código de parametrização que existe nas classes responsáveis pela geração do documento XML.

```
this.xmlBulder = new XStream();
this.xmlBulder.addImplicitCollection(InputCollection.class,"list","input",Input.class);
this.xmlBulder.addImplicitCollection(CircuitFeedCollection.class,"list","association", CircuitFeed.class);
this.xmlBulder.addImplicitCollection(PortsCollection.class,"list", "port", Component.class);
this.xmlBulder.alias("circuit", Circuit.class);
this.xmlBulder.alias("xor2", XorPort.class);
this.xmlBulder.alias("and2", AndPort.class);
this.xmlBulder.alias("not", NotPort.class);
this.xmlBulder.alias("or2", OrPort.class);
this.xmlBulder.alias("or3", Or3Port.class);
this.xmlBulder.alias("and3", And3Port.class);
this.xmlBulder.useAttributeFor("name",Component.class);
this.xmlBulder.addImplicitCollection(CircuitFeed.class,"feeders","feeder", String.class);
```

O código nessa transcrição é, juntamente com as classes geradoras, o projeto do esquema de XML do circuito. A primeira linha inicia a fachada da biblioteca criando o objeto que gera o XML e as linhas subseqüentes configuram como se dará essa geração. Na segunda linha temos uma chamada a um método que dentro da classe “InputCollection.class” mapeia o atributo com o nome “list” (que é a lista de objetos entrada, objetos da classe “Input.class”) como uma lista ilimitada de marcadores, cada um deles com o nome “input:”. A terceira e quarta linhas fazem o mesmo para as outras coleções. Já da quinta à décima segunda linha, ocorrem chamadas a um método que indica o nome da marcação para determinada classe. Na penúltima linha diz-se que o atributo Java chamado

“name” dentro da classe “Component” será mapeado como um atributo XML, sem essa chamada ele seria mapeado aninhadamente a marcação que representa a classe, pois é esse o comportamento padrão. Na última linha ocorre algo semelhante às linhas iniciais do mapeamento das coleções de circuitos, uma vez que o objeto “CircuitFeed” é na realidade uma coleção das alimentações de determinada porta.

O resto da configuração é parte do comportamento padrão da biblioteca. Por exemplo, nessas linhas não está evidente o nome de marcação “links”, que existe no esquema final (Veja novamente a Figura 2). Então como a biblioteca aprende que a coleção de ligações (mais precisamente a classe “CircuitFeedCollection”) deve ser mapeada numa marcação com o nome “links”? Isso acontece porque na classe “Circuit” o atributo Java que é essa coleção (e, portanto esse atributo é um objeto “CircuitFeedCollection”) tem esse nome. E é exatamente isso o comportamento padrão: A biblioteca coloca marcações aninhadamente com o nome usado como atributo Java.

4.2 O motor XML de posicionamento

Uma informação que não está contemplada no XML do circuito (e nem deveria) é o posicionamento de cada porta lógica no desenho. De início, a essa preocupação foi deixada de lado e o posicionamento de cada porta não era persistido. Isso fazia com que a ferramenta esquecesse a diagramação do circuito. Embora a diagramação (o formato do desenho) não interfira no processamento do circuito pela ferramenta, é preciso persistir a posição gráfica de cada porta lógica para garantir uma coerência visual durante o uso da ferramenta.

Seguindo o exemplo de ferramentas de desenho UML que tem um formato para os diagramas e outro formato para o posicionamento gráfico dos elementos dentro desses diagramas, pensou-se em um formato separado para gravar o posicionamento. A escolha por dois formatos tem seu fundamento também na pesquisa sobre modelagem XML de objetos. Pensou-se que a entidade circuito não se alteraria semanticamente caso algumas portas apenas mudassem de lugar. Além disso, o propósito do modelo não incluía o armazenamento das posições.

O subsistema posicionamento foi desenvolvido rapidamente a partir da reutilização do manipulador XML. O arquivo que armazena as posições é também um XML. Essa funcionalidade, embora irrisória dentro dos requisitos essenciais da ferramenta (protótipo), é a constatação de que o manipulador XML possui uma arquitetura inteligente e reutilizável.

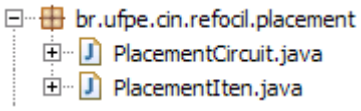


Figura 7

Esse pacote na Figura 7 contém o objeto paródia do circuito, que é uma representação do circuito (mas dessa vez não em XML e sim em Java) apenas para o propósito de posicionamento. O posicionamento de um circuito é composto pelo posicionamento de cada item.

No Apêndice C tem-se um exemplo de um documento XML de posição. É possível analisar sua sintaxe e perceber que ele se trata de mais uma representação XML de um circuito apenas para o propósito da posição dos componentes.

4.3 O controlador gráfico

O controlador gráfico engloba principalmente a interface com o usuário. Mas como esse trabalho apresenta a ferramenta como um protótipo, não se dirigiu maiores esforços na engenharia de usabilidade, como veremos. Logo no início desse trabalho prescreveu-se que a interface gráfica agiria integrada a outras ferramentas gráficas já de comum uso dentro do ambiente EMBRAER. Ou seja, a ferramenta funcionaria como uma extensão da funcionalidade de outra. Posteriormente viu-se que essa escolha era por demais, complexa e não fazia jus ao escopo do trabalho.

Então o controlador foi pensado apenas como um simples acessório (um cliente) do manipulador XML, que posteriormente poderia ser modificado, seja por questões de usabilidade ou de acoplamento a outras ferramentas. Veja a parte de trabalhos futuros.

Posto isto, para o seu desenvolvimento, utilizou-se a tecnologia Java2D, a qual permite a criação de efeitos geométricos dentro de janelas. Esses efeitos foram necessários para o desenho das portas lógicas e fios alimentadores dentro da janela, ainda que modestamente. Trata-se de um subsistema conciso, porém bastante simplificado.

As classes básicas desse subsistema gerenciam o desenho de cada porta lógica. Essas classes também indicam os pontos geométricos de entradas e saída de cada porta. Então um controlador desenho liga os pontos necessários para elaborar o circuito, através de retas. Não há muita sofisticação, a não ser pelas cores. Java2D permitiu o desenho de arcos, retas, e quadriláteros preenchidos, que juntos representam cada porta lógica.

4.4 A ferramenta em uso

Pode-se usar a ferramenta a partir do acesso ao arquivo executável Java. Ele foi nomeado de REFOCIL (um acrônimo que significa requisitos formais de circuitos lógicos). A Figura 8 abaixo mostra a ferramenta em uso.

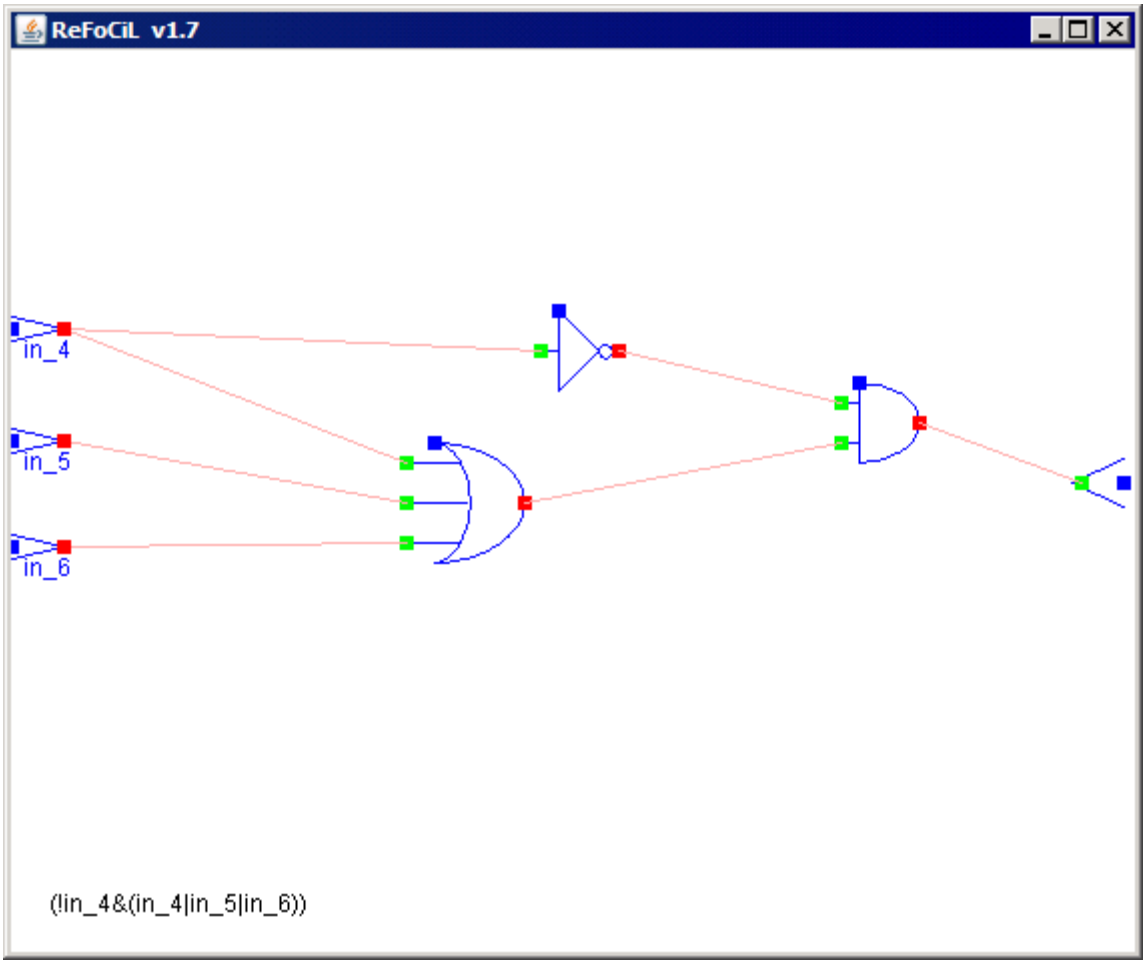


Figura 8

As entradas podem ser vistas à esquerda da janela, elas são todas nomeadas. Os fios de alimentação são cor-de-rosa e ligam um ponto vermelho a um verde. Os pontos azuis servem para interagir com cada porta ou entrada. No canto inferior direito encontra-se a proposição correspondente ao circuito, esse campo só aparece se o circuito estiver finalizado, é lógico.

A interação com o usuário se dá primordialmente através do mouse. É possível mover cada componente utilizando o botão esquerdo nos pontos azuis. As entradas também podem ser movidas, mas somente na dimensão vertical de modo que elas sempre permanecem na esquerda da janela. Com o botão esquerdo é possível construir uma ligação entre a partir de um ponto vermelho até um ponto verde, mas apenas se essa nova alimentação não constituir um ciclo dentro do circuito.

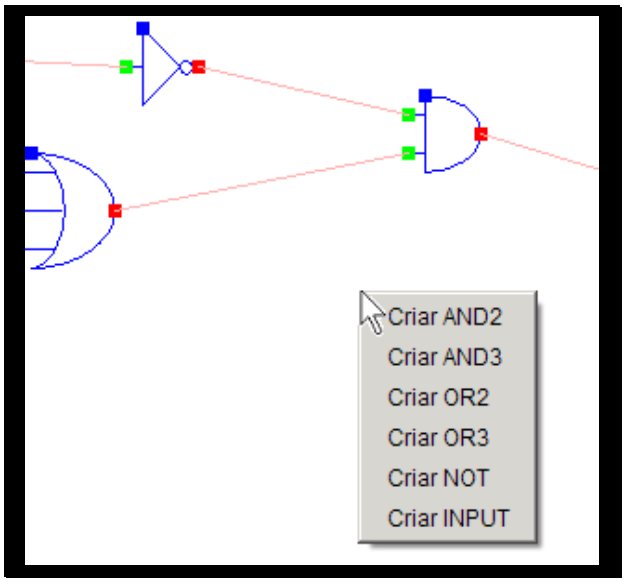


Figura 9

A Figura 9 mostra que quando utilizamos o botão esquerdo numa área livre surgem opções que permitem a criação de elementos do circuito. Quando utilizamos o botão direito num ponto azul surge uma outra lista de opções, associadas ao componente em questão. Como pode ser bem percebido na Figura 10 logo abaixo.

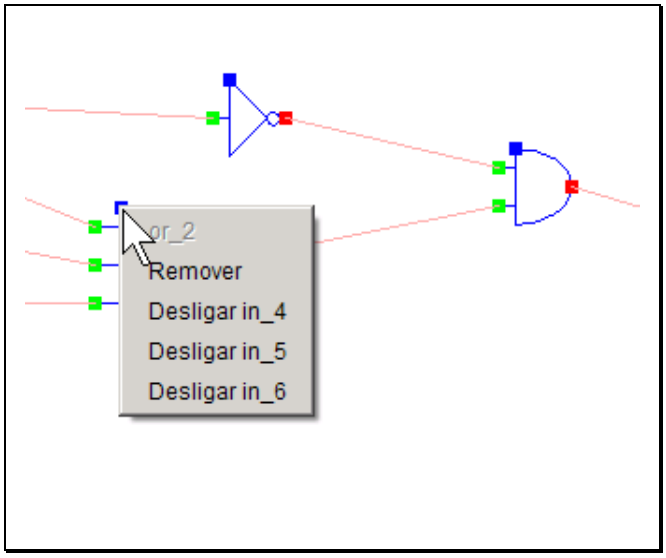


Figura 10

As opções que surgem ao clicar no ponto azul de uma porta permitem desligar uma alimentação ou remover a própria porta.

Como se vê, não há barra de ferramentas e o salvamento ocorre automaticamente quando o aplicativo é fechado. O arquivo XML do circuito é chamado “circuito.xml” e o arquivo XML de posicionamento é chamado “cicuito.place”. Esses arquivos são salvos na mesmo pasta do executável. Veja a Figura 11.

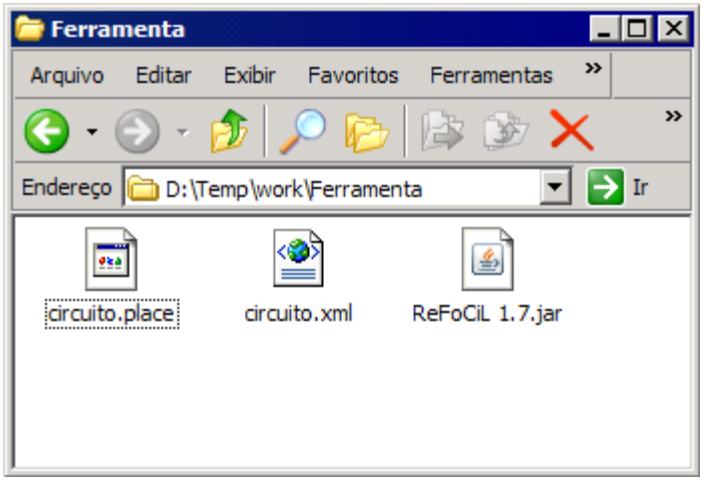


Figura 11

Se esses arquivos ainda não existem a ferramenta os cria de maneira a iniciar o circuito vazio. Na prática uma tela branca onde só é possível visualizar a saída do circuito.

5. Estudo de Caso

Durante o desenvolvimento da ferramenta técnicos da EMBRAER cederam alguns exemplos de circuitos lógicos do seu ambiente de desenvolvimento. A constante análise desses materiais permitiu que as funcionalidades da ferramenta não se distanciassem do que era esperado dentro de um ambiente de produção.

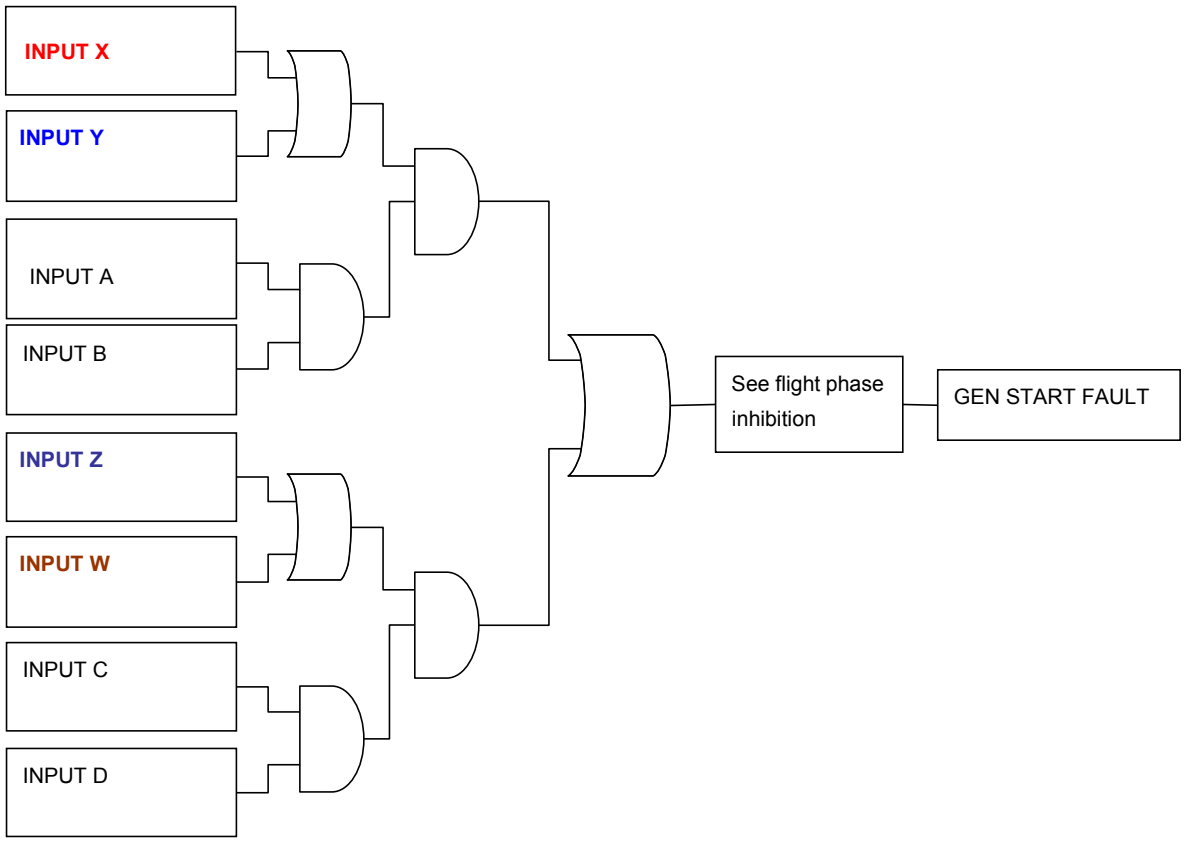
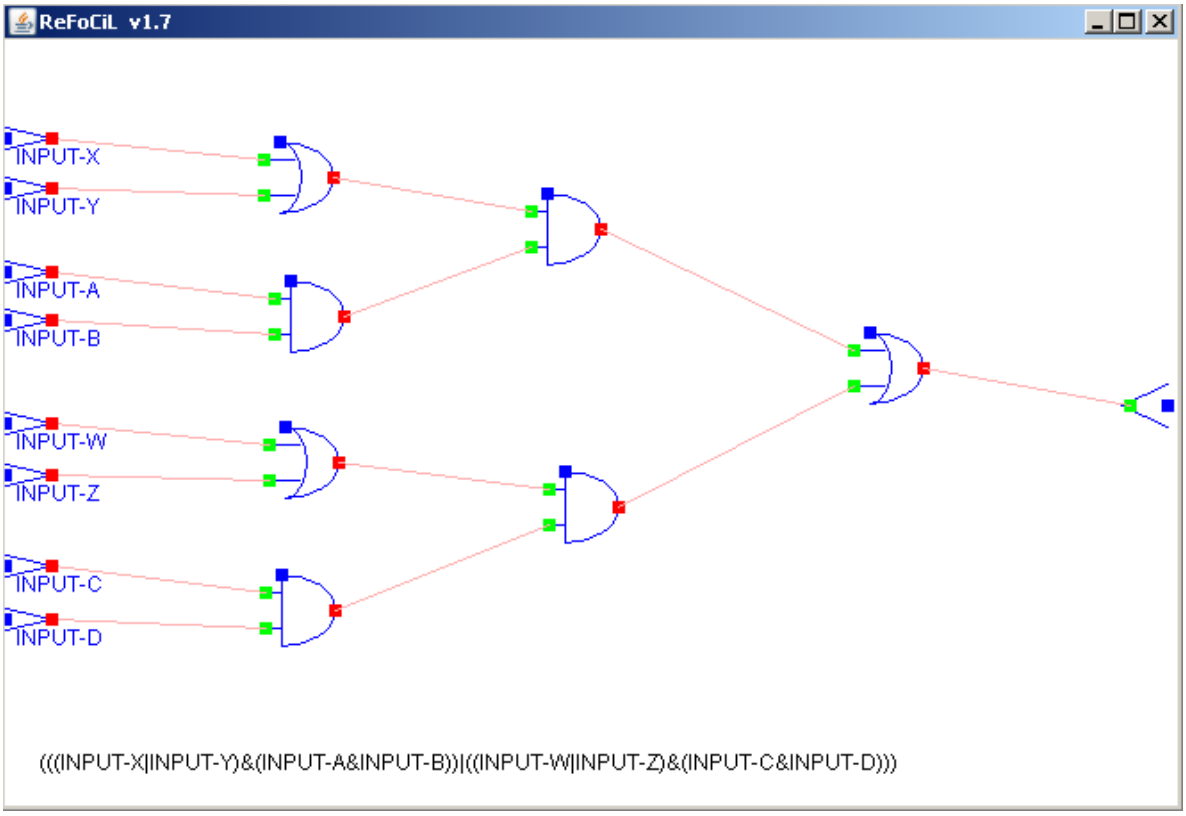


Figura 12

A Figura 12 mostra um dos exemplos enviados circuito associado à pressão de ar em pneumáticos. A Figura 14, outro associado à temperatura do combustível. Todos os exemplos enviados foram projetados na ferramenta onde se extraíram as proposições lógicas correspondentes. As proposições e XML gerados foram apresentados para os analistas de EMBRAER que validaram o material. Na Figura 13 mostra o circuito representado na Figura 12 re-projetado na ferramenta. Atente para a proposição lógica correspondente.




```
(((INPUT_X|INPUT_Y)&(INPUT_A&INPUT_B))|((INPUT_W|INPUT_Z)&(INPUT_C&INPUT_D)))
```

Figura 13

Ainda no circuito da Figura 14, podemos ver uma porta com quatro entradas, embora o modelo comporte qualquer tipo de porta a ferramenta não está pronta para desenhá-la. Outra característica para a qual a ferramenta também não está pronta é um circuito definido pela própria ferramenta sendo utilizado como uma porta lógica de outro circuito. Embora nenhum dos exemplos cedidos aponte essa característica, o modelo a comportaria. O Apêndice B apresenta transcrito o documento XML que corresponde ao circuito da Figura 12.

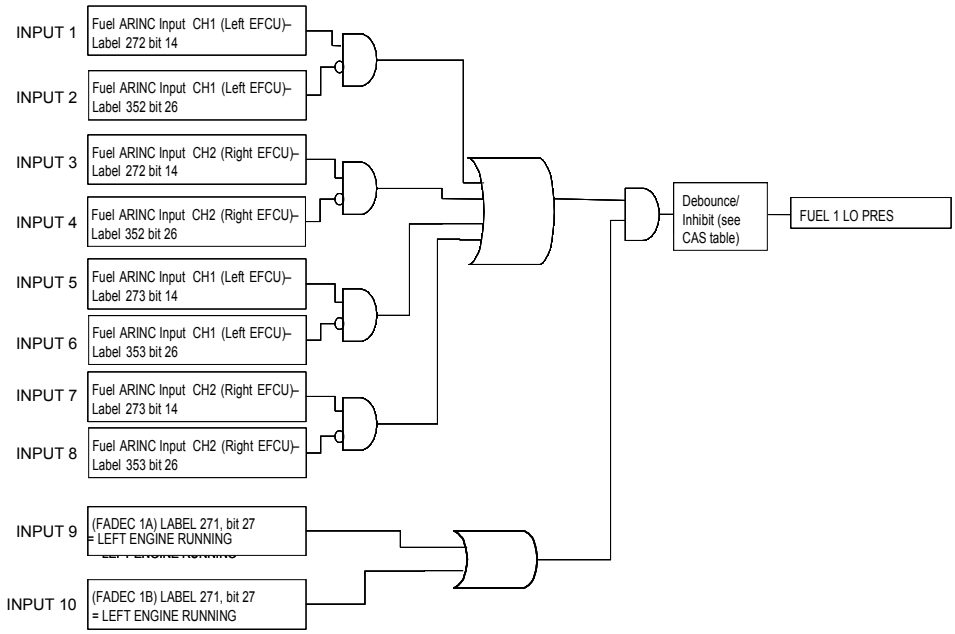


Figura 14

Posteriormente, a versão do executável produzido nesse trabalho foi apresentada aos técnicos da EMBRAER. Na apresentação, que foi recebida com considerável entusiasmo, projetaram-se circuitos do ambiente de produção. E com essa oportunidade foram discutidos:

- As funcionalidades da ferramenta
- A arquitetura da ferramenta
- As decisões de projeto do modelo XML

Por fim cedemos o executável produzido nesse trabalho aos técnicos, para que eles pudessem usufruir de uma experiência mais duradoura e produzissem um relatório de uso mais detalhado.

6. Conclusão

A produtividade dos ambientes de engenharia de sistemas passa por uma engenharia de requisitos que tenha agilidade no trato com as mudanças de requisitos. A maneira, como os circuitos lógicos eram trabalhados na engenharia de requisitos dos laboratórios da EMBRAER era dispendiosa. Além disso, trabalhar circuitos lógicos dentro da engenharia de requisitos é uma necessidade bastante desamparada de ferramentas. Provavelmente não haveria muitas escolhas para trazer celeridade à esse processo a não ser recorrer à academia.

A ferramenta desenvolvida nesse trabalho foi um esforço bem logrado à medida que proporciona mais eficiência na engenharia de requisitos de baixo nível (manipulando circuitos lógicos). A ferramenta alcançou níveis de adequação consideráveis, pois foi elaborada com o foco na problemática do ambiente da EMBRAER, que é bastante peculiar. Ela, através de um modelo XML suficientemente adequado, oferece suporte a ao conceito de circuitos lógicos como descritores de como sensores aéreos deveriam ser processados. E mais, consegue atingir essa necessidade específica agindo objetivamente no cerne da questão quando: provê um meio ágil de produzir e refinar o circuito; bem como de extrair sua proposição lógica.

6.1 Trabalhos Relacionados: Um comparativo

Seguindo uma metodologia investigativa, apresentamos também dois outros aplicativos editores de circuitos lógicos os quais faziam parte da nossa pesquisa bibliográfica inicial. O intuito era estimular uma avaliação comparativa para trazer mais confiabilidade para a validação da ferramenta. Faz sentido mostrar quais são esses aplicativos.

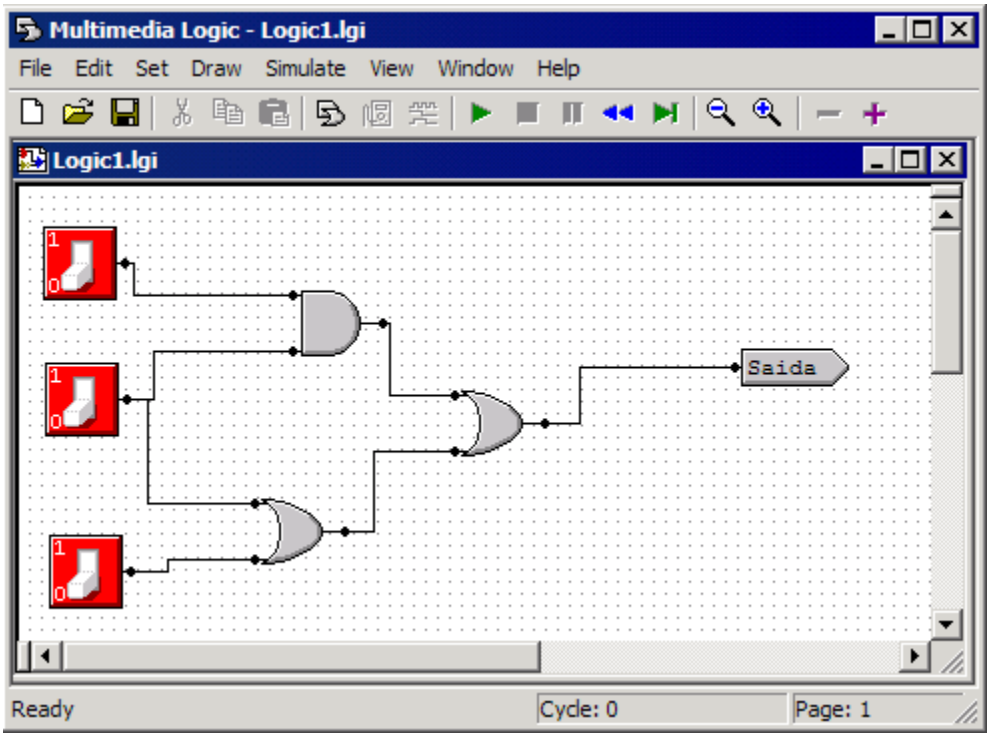


Figura 15

A Figura 15 é uma tela do Multimedia Logic® [7], que apresenta um ambiente para a criação de circuitos lógicos. Trata-se de um simulador de circuitos lógicos gratuito para uso não comercial. Ele é capaz de gerar a tabela verdade e o gráfico de oscilações de um circuito.

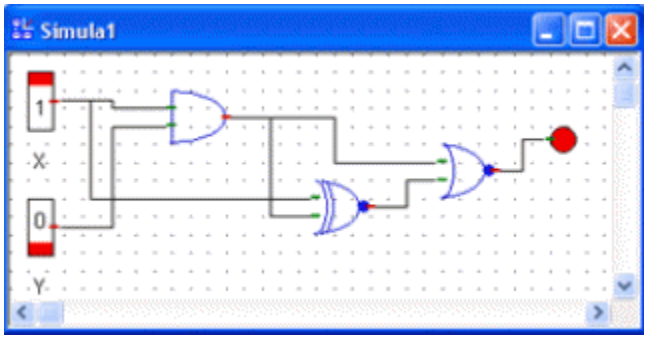


Figura 16

A Figura 16 é uma tela do Logic Circuit Simulator® [8], que revela um ambiente para a criação de circuitos lógicos bem mais simples. Trata-se de um simulador de circuitos lógicos proprietário. Ele é capaz de gerar a tabela verdade e também é capaz de gerar a proposição lógica correspondente.

Depois de apresentados, os aplicativos foram utilizados na elaboração de um circuito pelos técnicos. Sugeriu-se em conjunto uma série de características que se mostravam interessantes e foi traçado um quadro comparativo entre as ferramentas com base nessas características.

	Possibilidade de Integração	Oferecimento	Circuitos como XML	Circuito para Proposição	Proposição para Circuito
MMLogic[1]	**Não**	**Livre**	**Não**	**Não**	**Não**
LCS[2]	**Não**	**Pago**	**Não**	**Sim**	**Não**
ReFoCiL[3]	**Sim**	**Livre**	**Sim**	**Sim**	**Não**

([1]Multimedia Logic) ([2]Logic Circuits Simulator) ([3] a ferramenta desse trabalho)

- Possibilidade de Integração: Uma vez que, a ferramenta iria ser encaixada num processo onde já existem outras ferramentas aderentes, seria necessário um mínimo de interoperabilidade entre elas.
- Oferecimento: A maneira pela qual se pode adquirir a ferramenta pode constituir um empecilho.
- Circuitos em XML: Uma vez que os circuitos forem implementados em XML eles podem ser facilmente manipulados através de transformações. Além da maior possibilidade de extensão.
- Conversão de Circuito para Proposição: Essa foi uma das necessidades motivadoras desse trabalho.
- Conversão de Proposição para Circuito: Descobriu-se a partir de então que essa necessidade é interessante, porém, ela ficou de fora do escopo do trabalho. Veja o apêndice que cita contribuições futuras.

A característica de utilizar XML para representar circuitos lógicos não só não fora encontrada nos aplicativos usados na avaliação como também em nenhum outro pesquisado na análise bibliográfica. Os aplicativos que lidam com circuitos lógicos estão geralmente associados à ambientes de engenharia eletrônica ou são pedagógicos. Ou seja, se toma uma abordagem muito densa ou muito superficial. A demanda do ambiente da EMBRAER é um meio termo onde o circuito deve descrever um comportamento (um requisito). Com necessidades tão específicas foi quase dedutível que a ferramenta desse trabalho se sobressaísse.

6.2 Trabalhos futuros

A conclusão deste trabalho abre precedentes para a realização de trabalhos futuros: Que entendam as vantagens aqui desenvolvidas; E que sedimentem a postura de empresas de envergadura nacional de buscar apoio nos centros de pesquisa.

Segue então uma lista de melhoras na ferramenta com base também, em sugestões das reuniões do estudo de caso.

- Gerar XML a partir de proposição (veja o Apêndice);
- Integração com a ferramenta Telelogic Doors®;
- Estender o campo visual com barras de rolagem;
- Gerar a proposição lógica em arquivo de texto;

7. Referências

[1] DURANT, John. **XML programming bible**. Wiley Publishing, Págs. 3-27.

[2] Mani M., Lee D. & Muntz R. **Semantic Data Modeling using XML Schemas**. University of Califórnia, 1996, Los Angeles

[3] XStream. **XStream - A simple library to serialize objects to XML and back again.** Acesso em 27/11/2008. Disponível em: http://xstream.codehaus.org

[4] SOMMERVILLE, Ian. **Requiriments engineering: process and techniques**. John Wiley and Sons, 1998, Págs. 28-31.

[5] MARKOVSKI, J. **Logic circuit visualization**. Faculty of Natural Sciences and Mathematics, Ss. Cyril and Methodius University Arhimedova b.b., PO Box 162, 1000 Skopje, Macedonia. jasen@ii.edu.mk

[6] IBM TELELOGIC. **Telelogic Doors - Requirements Management for Complex Systems.** Acesso em 27/11/2008. Disponível em: http://www.telelogic.com/Products/doors/doors/index.cfm

[7] SOFTRONIX. **Multimedia Logic.** Acesso em 27/11/2008. Disponível em: http://www.softronix.com/logic.html

[8] SIMULAWORKS. **Logic Circuit Simulator.** Acesso em 27/11/2008. Disponível em: http://www.simulaworks.net/

Apêndice A: Um conversor da proposição para o XML

Como visto anteriormente, a ferramenta foi preparada como uma versão bem simplificada. Mas tão ou mais importante que apresentar qualquer melhoramento em matéria de controles de interação pelo qual se atinja um ambiente mais produtivo, é apresentar essa funcionalidade essencial, mas que ficou fora do escopo.

Durante o decorrer desse trabalho percebeu-se que seria interessante que também se automatizasse o caminho inverso entre modelo XML e proposição textual. De maneira mais direta, um compilador que transformasse uma proposição textual num circuito XML. Ainda chegou-se a despender-se algum esforço para o desenvolvimento desse compilador, mas até o momento da finalização desse escrito não foi produzida qualquer versão apresentável. Ainda assim, não seria de todo mal apresentar detalhes do que foi traçado dentro do projeto desse compilador.

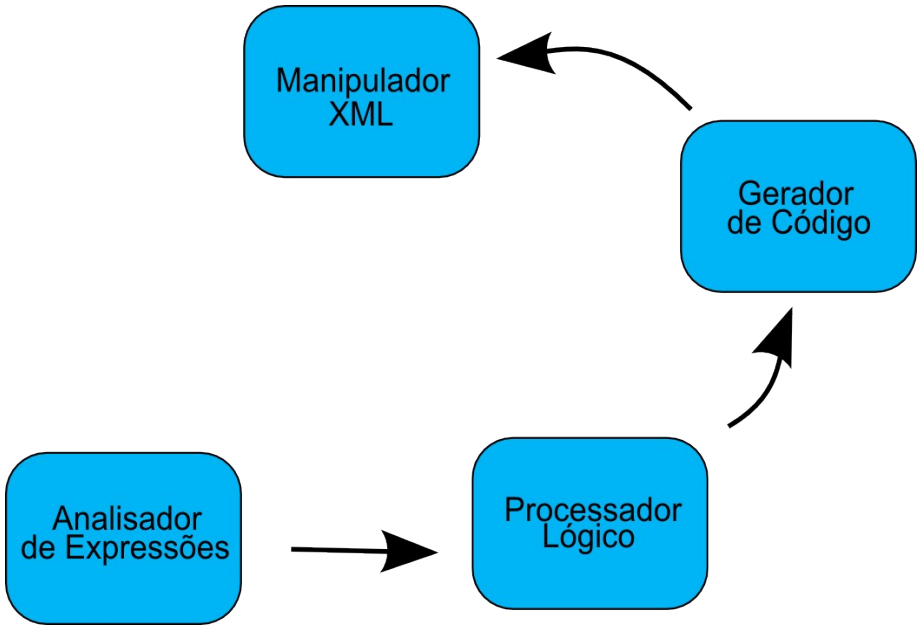


Figura A

A Figura A revela uma possível divisão de fases de processamento que o compilador compreenderia. Pode se avistar no topo da ilustração o manipulador XML. Ele seria logicamente necessário no último passo, que é escrever o XML do circuito. Esse subsistema já está escrito e sua capacidade de reuso, provada

anteriormente no subsistema de posicionamento, certamente trará alguma vantagem. E as adaptações seriam mínimas. A grosso modo, apenas mudaria o cliente do manipulador que não seria mais o gatilho (que sincronizava com o desenho gráfico) e sim o gerador de código desse compilador.

De início, a fórmula textual iria passar por um analisador de expressões para validá-la: analisar os parênteses, analisar a precedência de operadores, etc..

Para entender a razão da necessidade de um processador lógico é preciso estar ciente de que a mesma proposição pode ser escrita de formas diferentes, e mais ainda, uma proposição lógica pode corresponder a circuitos diferentes. Por exemplo, a formula da Figura 3 existe um parênteses com três termos. Podemos usar uma porta com três entradas ou duas portas com duas entradas cada, então o que escolher? E veja ainda:

```
(!in_4 & in_5) |(!in_4 & in_6)
```

Figura B

A Figura B apresenta a mesma fórmula da Figura 3, porém, escrita de outra maneira.

Embora pareça confuso decidir como ficará o circuito, muito provavelmente qualquer escolha satisfará esse requisito. Quando se tem apenas a formula o que se procura é apenas uma maneira de compreendê-la melhor visualmente.

Então a verdadeira preocupação desse processador lógico é reduzir a expressão da forma que torne o processo de construção do XML mais natural. Se o processador lógico lograr êxito nessa tarefa o gerador de código poderá reutilizar partes no analisador de expressões para ir lendo a proposição linearmente à medida que vai construindo o XML.

Espera-se que essa seção guie algum trabalho futuro no intuito do provimento dessa funcionalidade desejada, mas não contemplada aqui.

Apêndice B: Exemplo de circuito em XML do estudo de caso

```
<circuit>
  <inputs>
    <input>
      <name>INPUT-X</name>
```

```
    </input>
    <input>
      <name>INPUT-Z</name>
    </input>
    <input>
      <name>INPUT-B</name>
    </input>
    <input>
      <name>INPUT-C</name>
    </input>
    <input>
      <name>INPUT-Y</name>
    </input>
    <input>
      <name>INPUT-D</name>
    </input>
    <input>
      <name>INPUT-W</name>
    </input>
    <input>
      <name>INPUT-A</name>
    </input>
  </inputs>
  <ports>
    <port class="or2">
      <name>or_1</name>
    </port>
    <port class="and2">
      <name>and_2</name>
    </port>
    <port class="and2">
      <name>and_3</name>
    </port>
    <port class="and2">
      <name>and_4</name>
    </port>
    <port class="and2">
      <name>and_5</name>
    </port>
    <port class="or2">
      <name>or_6</name>
    </port>
    <port class="or2">
      <name>or_7</name>
    </port>
  </ports>
  <links>
    <association>
      <device>or_1</device>
      <feeder>INPUT-X</feeder>
      <feeder>INPUT-Y</feeder>
    </association>
    <association>
      <device>and_2</device>
      <feeder>INPUT-A</feeder>
      <feeder>INPUT-B</feeder>
    </association>
```

```
    <association>
      <device>or_6</device>
      <feeder>INPUT-W</feeder>
      <feeder>INPUT-Z</feeder>
    </association>
    <association>
      <device>and_4</device>
      <feeder>INPUT-C</feeder>
      <feeder>INPUT-D</feeder>
    </association>
    <association>
      <device>and_3</device>
      <feeder>or_1</feeder>
      <feeder>and_2</feeder>
    </association>
    <association>
      <device>and_5</device>
      <feeder>or_6</feeder>
      <feeder>and_4</feeder>
    </association>
    <association>
      <device>or_7</device>
      <feeder>and_3</feeder>
      <feeder>and_5</feeder>
    </association>
  </links>
  <output>or_7</output>
</circuit>
```

Apêndice C: Documento de posição do circuito do estudo de caso

```
<circuit>
  <device>
    <name>INPUT-X</name>
    <x>21.0</x>
    <y>78.0</y>
  </device>
  <device>
    <name>INPUT-Z</name>
    <x>5.0</x>
    <y>267.0</y>
  </device>
  <device>
    <name>INPUT-B</name>
    <x>6.0</x>
    <y>180.0</y>
  </device>
  <device>
    <name>INPUT-C</name>
    <x>5.0</x>
    <y>318.0</y>
  </device>
  <device>
```

```
    <name>INPUT-Y</name>
    <x>11.0</x>
    <y>106.0</y>
  </device>
  <device>
    <name>INPUT-D</name>
    <x>7.0</x>
    <y>348.0</y>
  </device>
  <device>
    <name>INPUT-W</name>
    <x>6.0</x>
    <y>238.0</y>
  </device>
  <device>
    <name>INPUT-A</name>
    <x>6.0</x>
    <y>153.0</y>
  </device>
  <device>
    <name>or_1</name>
    <x>158.0</x>
    <y>80.0</y>
  </device>
  <device>
    <name>and_2</name>
    <x>164.0</x>
    <y>158.0</y>
  </device>
  <device>
    <name>and_3</name>
    <x>308.0</x>
    <y>109.0</y>
  </device>
  <device>
    <name>and_4</name>
    <x>159.0</x>
    <y>323.0</y>
  </device>
  <device>
    <name>and_5</name>
    <x>318.0</x>
    <y>265.0</y>
  </device>
  <device>
    <name>or_6</name>
    <x>161.0</x>
    <y>240.0</y>
  </device>
  <device>
    <name>or_7</name>
    <x>489.0</x>
    <y>187.0</y>
  </device>
</circuit>
```

Assinaturas

Alexandre Cabral Mota

Orientador

Glerter Alcantara Sabiá

Aluno