6
Conclusões e trabalhos futuros

O objetivo deste trabalho era tornar mais fácil o desenvolvimento de aplicações NCL. Com isso, uma ferramenta para inspeção de código NCL foi proposta e implementada: o NCL-Inspector. Além disso, uma pesquisa considerável foi feita na literatura a fim de catalogar problemas de código NCL, chegando a identificar quase uma centena de problemas diferentes.

O propósito inicial era criar um sistema de críticas abrangente, contendo todas as características propostas na Seção 4.1 (página 29). Além disso, o sistema deveria inspecionar os problemas de código, apresentados no Apêndice H (página 108). Estudos também seriam feitos na direção de tentar adicionar ao NCL-Inspector, a capacidade de detectar problemas de Interação Humano-Computador.

A metodologia de trabalho envolveu uma ampla análise de sistemas similares, apresentados no Capítulo 3 (página 21). A análise de sistemas similares e a pesquisa sobre o estado da arte dos sistemas de críticas, resultaram no levantamento dos requisitos do sistema, apresentados na Seção 4.1 (página 29).

Ao finalizar esta etapa, foi identificado que o escopo do problema era muito grande para o tempo disponível para finalização do trabalho. Implementar todos os requisitos do sistema e todas as regras levantadas, Seção 4.1 (página 29) e Capítulo 3 (página 21) respectivamente, era um trabalho inviável para o tempo disponível. Consequentemente, decidimos priorizar a criação do *framework* que serviria de base para o sistema.

Dos requisitos apresentados na Seção 4.1 (página 29), os que receberam prioridade foram: "possibilidade de adicionar regras em tempo de carga" e "possibilidade de escrever regras seguindo tanto o paradigma imperativo ou declarativo". Acreditamos com isso que estaremos proporcionando uma boa

infraestrutura, assim, outros desenvolvedores poderão adicionar regras no NCL-Inspector, somando esforços para o desenvolvimento da ferramenta. A Tabela 6.1 apresenta resumidamente quais requisitos foram implementados no NCL-Inspector, comparando-o com as ferramentas apresentadas como trabalhos relacionados.

Requisito	NCL-Inspector	JDT	PMD	Check-Style	Find Bugs	NCL-Validator
Sistema de críticas baseado em regras	**Sim**	Não	**Sim**	**Sim**	**Sim**	Não
Especificação de regras imperativamente e declarativamente	**Sim**	Não	**Sim**	Não	Não	Não
Seleção de regras em tempo de execução	**Sim**	Não	**Sim**	**Sim**	**Sim**	Não
Possibilidade de adicionar regras em tempo de carga	**Sim**	Não	**Sim**	**Sim**	**Sim**	Não
Integração com o ambiente de desenvolvimento	Não	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
Explicação do problema e sugestão de correção	Parc.	Parc.	Parc.	Parc.	**Sim**	Não
Correção de problemas automaticamente	Não	**Sim**	**Sim**	Não	Não	Não
Repositório central de regras	Não	Não	Não	Não	Não	Não

Tabela 6.1: Resumo dos requisitos implementados

6.1 Trabalhos futuros

É possível enumerar alguns trabalhos que podem ser sugeridos para complementar as lacunas deixadas por este:

- Melhorias na explicação dos problemas e sugestões de correção.
- Modificações no mini-*framework* de teste.
- Melhorias na notificação de violação do XSL.
- Inclusão de novas regras.
- Esquema taxonômico baseado nas instâncias de problemas NCL.

- Validação via Schematron.
- Eliminação do Arquivo de Declaração do Inspetor.
- Generalização do *framework* para validação de qualquer linguagem baseada em XML.
- Integração com o NCL-Eclipse.
- Ampliação do suporte para o perfil estendido da NCL.
- Avaliação de problemas de IHC.
- Criação de um repositório central de regras (consulte a Seção 4.1.8, na página 33, para mais detalhes).
- Correção automática de problemas (consulte a Seção 4.1.7, na página 32, para mais detalhes).
- Criação de uma Linguagem de Domínio Específico (consulte a Seção 5.2.8, na página 75 para mais detalhes).

Esses trabalhos serão detalhados a seguir, separados em subseções conforme seu grau de complexidade.

6.1.1 Trabalhos Simples

Melhorias na explicação dos problemas e sugestões de correção

Essa melhoria está fortemente ligada à integração do NCL-Inspector com um ambiente de desenvolvimento, como, por exemplo, o NCL-Eclipse. Como atualmente o NCL-Inspector possui somente uma interface de linha de comando, seu poder de expressão fica um pouco limitado. Quando a integração com um ambiente de desenvolvimento for implementada, é importante observar as oportunidades de melhorias das explicações e sugestões que surgem, devido à adoção de uma nova interface.

Como sugestão de implementação desta melhoria, podemos nos basear na solução dada pelo FindBugs, ilustrada na página 26, Figura 3.8. Observe que o editor de texto possui um componente ancorado na parte inferior contendo um texto. Esse texto possui uma explicação minuciosa das causas do problema, outros exemplos ilustrando outras instâncias dessa violação e sugestões de como solucioná-lo. Talvez por questões de usabilidade, no nosso caso, essas explicações poderiam ser apresentadas através de uma *popup*, isto é, da mesma

forma que o Eclipse apresenta o Javadoc de uma classe ou método, quando estamos utilizando o JDT.

Uma alternativa de implementação que não necessita de uma interface gráfica consistiria em exibir mensagens mais pobres na tela, da mesma forma como é feito atualmente. Entretanto, através de um argumento passado na linha de comando, um arquivo HTML pudesse ser gerado contendo essas mensagens mais ricas.

Modificações no mini-*framework* de teste

O mini-*framework* de teste torna o teste de uma regra quase trivial, porém existem alguns pequenos detalhes que poderiam ser melhorados, embora não comprometam o seu funcionamento. Observando o caso de teste exemplificado na Listagem 4.5, na página 42, é possível identificar os seguintes problemas:

- a necessidade da classe de teste estar acoplada através de herança com a classe `InspectorTestCase` diminui a flexibilidade do framework;
- a passagem do identificador da regra para a classe pai, através da chamada `super()` diminui bastante a legibilidade do código e ainda diminui a redigibilidade do código;
- ainda sobre a passagem do identificador, isso reduz a redigibilidade do código, uma vez que é necessário criar um construtor apenas para isso.

Uma sugestão simples para tornar o código mais legível é utilizar programação orientada a atributos. Em Java, atributos são chamados de anotações[1]. Utilizando essa técnica, o caso de teste discutido nessa seção ficaria como é mostrado na Listagem 6.1.

```
1  package coreinspectors.layout;
2
3  import java.io.IOException;
4
5  @RunWith(PowerMockRunner.class)
6  @InspectorTestCase("coreinspectors.layout.
       InconsistentRegionSize")
7  public class InconsistentRegionSizeTest {
8
9    @Test
10   @InspectFiles("TestCaseError.ncl")
```

[1]Do Inglês: *annotations*

```
11    public void testInconsistentRegionSizeFail() throws
          IOException,
12         URISyntaxException {
13
14      expectWarning().inLine(8).inColumn(5);
15      expectWarning().inLine(10).inColumn(6);
16      expectWarning().inLine(11).inColumn(7);
17
18      expectNoMoreErrors();
19    }
20
21    @Test
22    @InspectFiles("TestCaseSuccess.ncl")
23    public void testInconsistentRegionSizeSuccess() throws
          URISyntaxException,
24         IOException {
25
26      expectNoMoreErrors();
27
28    }
29  }
```

Listagem 6.1: InconsistentRegionSizeTestAnnotated.java – Implementação de um caso de teste com *annotations*

Melhorias na notificação de violação do XSL

Essa melhoria foi descoberta durante o experimento realizado com usuários. A Seção 5.2.7 (página 74) detalha o propósito dessa melhoria.

Uma sugestão para a implementação poderia ser encapsular o XML de erro dentro de uma macro XSL. Neste caso, seria a macro que iria permitir que fosse omitido algum argumento. Então, na ocorrência dessa omissão, seria considerado que os atributos `line` e `column` são o elemento corrente, representado por XSL como um ponto (.).

Inclusão de novas regras

Apenas um conjunto pequeno das regras apresentadas no Apêndice H foram implementadas. Para tornar o NCL-Inspector realmente útil é necessário que todas as regras que verificam a validade do código NCL sejam implementadas.

Esquema taxonômico baseado nas instâncias de problemas NCL

Baseado na norma e em outras fontes de consulta, foram catalogados no Apêndice H quase uma centena de problemas de código NCL. Muitos desses problemas são, na verdade, instâncias do mesmo problema. Considere, por exemplo, os seguintes problemas de código NCL, retirados do Apêndice H:

1. O atributo `role` do elemento `<simpleCondition>` deve ser único para o conjunto de roles de um elemento `<causalConnector>`.
2. O atributo `role` do elemento `<simpleAction>` deve ser único para o conjunto de roles de um elemento `<causalConnector>`.
3. O atributo `role` do elemento `<attributeAssessment>` deve ser único para o conjunto de roles de um elemento `<causalConnector>`.
4. O atributo `name` do elemento `<connectorParam>` deve ser único para o conjunto de names de um elemento `<causalConnector>`

Observe que todos os seis problemas enumerados são na verdade instâncias de um mesmo problema. Poderíamos escrever cada um dos problemas usando o modelo: "O atributo x de um elemento y deve ser único para um conjunto de x's de um elemento z".

Identificar e classificar os problemas de mesma instância irá certamente facilitar o desenvolvimento das regras de inspeção. Ao criar uma regra para inspecionar um dos problemas mencionados anteriormente, podemos parametrizar essa regra para que inspecione também as outras instâncias desse problema.

Validação via Schematron

É interesse dos projetistas da NCL que o XML Schema da linguagem possua restrições também escritas em Schematron, para restrições instrínsecas à linguagem, isto é, que não dependem de boas práticas ou padrões de codificação. Se isso ocorrer, seria interessante que o processo de inspeção do NCL-Inspector incorporasse uma validação baseada no Schematron. Como foi apresentado na Figura 4.2 da página 35, no passo 4, o *parser* realiza uma validação baseada no XML Schema. Além dessa validação baseada no XML Schema, poderia, sem muito esforço, ser adicionada uma validação baseada no Schematron.

6.1.2 Trabalhos de Complexidade Moderada

Eliminação do Arquivo de Declaração do Inspetor

O princípio DRY (do Inglês Don't Repeat Yourself – Não se Repita) é bastante difundido no meio de profissionais que utilizam técnicas de Desenvolvimento Ágil. Esse princípio é descrito em detalhes em (Hunt et al. 1999). Violar esse princípio certamente irá ocasionar em problemas de manutenção. O autor descreve a essência deste princípio com a seguinte afirmação:

> Todo fragmento de conhecimento deve haver uma única, não-ambígua e confiável representação dentro do sistema[2]. (Hunt et al. 1999)

O Arquivo de Declaração do Inspetor viola esse princípio, o que o torna prejudicial. Outro agravante que pesa contra, é que ao longo do desenvolvimento do sistema, ele demostrou ser também dispensável. Para isso, é necessário algumas modificações no algoritmo que faz a leitura das regras, que está no módulo do Construtor do Mecanismo de Regras. Esse algoritmo não deverá mais procurar pelo arquivo de declaração de inspetor. Em vez disso, ele deverá efetuar uma busca por todos os arquivos Java e XSL dentro da biblioteca de regras.

Os metadados do Arquivo de Declaração do Inspetor, podem ser incluídos diretamente dentro do arquivo Java, através de anotações. Os arquivos XSL não precisam de metadados, pois o valor do elemento `<parent>` é sempre o mesmo, logo, os metadados não tem relevância para regras em XSL.

O único motivo que talvez justifique a manutenção do Arquivo de Declaração do Inspetor, é a maior flexibilidade que ele proporciona. Se uma nova tecnologia for usada para implementar regras além de Java e XSL, essa tecnologia deve possuir formas de incluir metadados, como Java permite através de anotações, caso contrário, isso pode inviabilizar o uso dessa tecnologia. É provável que a melhor forma para solucionar esse problema é tornar o Arquivo de Declaração do Inspetor opcional.

[2]Do Inglês: "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." (Hunt et al. 1999)

Generalização do *framework* para validação de qualquer linguagem baseada em XML

Um tempo considerável foi investido na criação de uma arquitetura para o NCL-Inspector. O objetivo era tornar o software mais flexível possível. O retorno desse investimento foi que, com algumas pequenas modificações, é possível tornar o NCL-Inspector um validador genérico de XSL.

Devido à estrutura baseada em componentes do NCL-Inspector, as modificações podem ser feitas sem necessidade de mexer na forma com são escritas ou carregadas as regras. Para implementação dessa melhoria é preciso: (1) criar um mecanismo para trocar o *parser* sem a necessidade de recompilar todo o código; (2) gerar um *parser* através do XML Beans para a linguagem em que deseja fazer a inspeção.

6.1.3 Trabalhos Complexos

Integração com o NCL-Eclipse

Usar o NCL-Inspector como validador dentro do ambiente NCL-Eclipse, proporcionaria uma melhoria mútua para ambos os sistemas. O NCL-Eclipse poderia incorporar um editor e um assistente para criação de regras, tornando mais fácil a criação das mesmas. Além disso, poderia automaticamente fazer a implantação da regra no NCL-Inspector. O NCL-Inspector por sua vez, tornaria o NCL-Eclipse uma ferramenta mais flexível e poderosa, no que diz respeito a detecção de problemas.

A Seção 4.1.5 (página 31) apresenta um ponto geral que deve ser observado, em relação a integração de sistemas de críticas com ambientes de desenvolvimento.

Ampliação do suporte para o perfil estendido da NCL

A intenção inicial do NCL-Inspector era suportar ambos os perfis da linguagem NCL. Como foi apresentado, o *parser* usado pelo NCL-Inspector foi gerado automaticamente pelo XML Beans, através do XML Schema da linguagem. Entretanto, alguns problemas ocorreram durante esse processo:

- o XML Beans não conseguiu gerar o *parser* para o XML Schema no NCL. O gerador de código do XML Beans parecia se perder em meio a grande utilização do `<xsd:substitutionGroup>`, por parte do XML Schema da NCL. Isso aconteceu também com outros geradores de *parsers* XML para Java.
- O XML Schema possuía alguns erros e discrepâncias com o texto da Norma ABNT da NCL (vide Apêndice G).
- O XML Schema possuía erros na referência ao SMIL.

Para progredir com o trabalho, foi pego o XML Schema do perfil básico da NCL e reescrito de forma a retirar os `<xsd:substitutionGroup>`. Com essa modificação, o XML Beans conseguiu fazer a geração de código corretamente.

Antes de efetuar qualquer esforço de implementação no código NCL, é preciso verificar que os erros do XML Schema tenham sido corrigidos. Além disso, o XML Schema também precisa ser reescrito, de forma que, consiga ser interpretado pelos XML Beans e os principais geradores de *parsers* do mercado. Caso contrário, é preciso modificar o XML Schema do Perfil Estendido da NCL, da mesma forma como feito com o Perfil Básico. Somente então poderá ser gerado um novo *parser* e adicionado ao NCL-Inspector.

Avaliação de problemas de IHC

É desejável que o NCL-Inspector possua regras que avaliem problemas relacionados a qualidade do uso das aplicações de TV Digital. Existem estudos relacionados a usabilidade em iTV. A partir desses estudos, deve ser feita uma classificação dos problemas que são viáveis ou não de serem detectados automaticamente, então, um próximo passo é implementar no NCL-Inspector essas regras de inspeção.