6
Aplicação da Ferramenta

6.1.
Projeto Alvo

A fim de se verificar a eficácia da linguagem de máquinas de estado criada, tanto como linguagem de especificação para sistemas predominantemente reativos e com requisitos de tempo real, a ferramenta criada foi utilizada em um projeto real. O projeto escolhido consiste na criação de um sistema que tem como finalidade dar suporte à gestão de frotas de veículos. Seu objetivo principal é controlar a operação de frotas de transporte coletivo, permitindo às empresas controlarem seus veículos fora das garagens e terminais. O posicionamento dos veículos é controlado através de um aparelho GPS que transmite suas coordenadas para o sistema através da internet, como pode ser visto na figura 20.

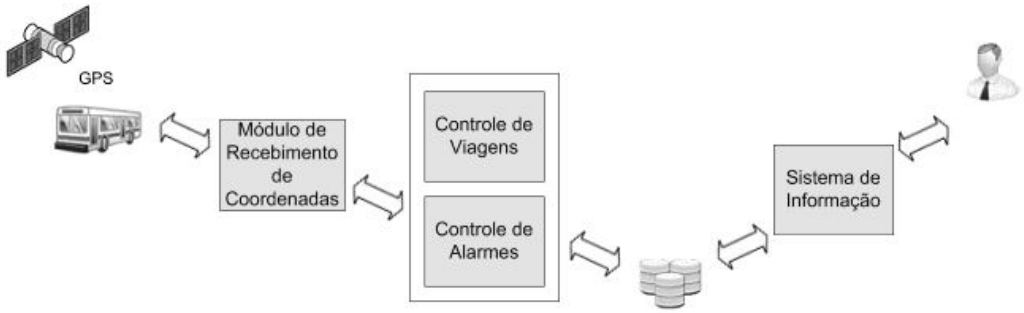

Figura 22: Arquitetura do sistema alvo

O sistema foi, na sua maior parte, desenvolvido na linguagem de programação Java. Devido à necessidade de fácil acessibilidade e funcionamento em múltiplas plataformas, foi adotada a arquitetura cliente-servidor em plataforma web, utilizando a tecnologia J2EE [24]. Na construção das interfaces com o usuário, parte considerável do código de exibição foi escrito em JavaScript [25], utilizando as técnicas de Ajax [26] para a construção de aplicações web ricas [27], através do *framework* DWR [28]. Para a persistência dos dados foi utilizado o *framework* Hibernate [29], que cria um modelo de persistência objeto-relacional,

PUC-Rio - Certificação Digital Nº 0721317/CA

fornecendo um nível mais alto de abstração para seus usuários. O *framework* Log4J [30] foi empregado para o registro das operações do sistema, que possui vários níveis de mensagens e permite que seus registros de sejam ativados e desativados através de um arquivo de configuração, podendo esta configuração ser feita de acordo com classes, pacotes, nível de mensagem, entre outras.

6.1.1. A Arquitetura

A arquitetura básica do sistema possui quatro componentes principais: o módulo de recebimento de coordenadas, o controle de alarmes, o controle de viagens e o sistema de informação.

6.1.1.1. Módulo de Recebimento de coordenadas

O módulo de recebimento de coordenadas é a parte do sistema responsável por receber as informações de posicionamento dos veículos, persistí-las e notificar os demais componentes do sistema de que novas coordenadas estão disponíveis. As coordenadas são enviadas por equipamentos de GPS [31]disponíveis nos veículos, e que devem se comunicar com o sistema utilizando algum protocolo, através da internet. Como muitas vezes equipamentos de posicionamento de algum tipo já existiam nos veículos previamente à implantação desse sistema, esse componente foi construído de forma flexível o suficiente para que a maneira como o sistema recebe e interpreta as informações de posicionamento possa ser modificada com facilidade. Esses módulos são, portanto, adaptados para diferentes aparelhos, desde que eles sejam capazes de fornecer as informações necessárias ao sistema. Diversos tipos de aparelhos também podem ser utilizados simultaneamente, através da utilização de diversos módulos de recebimento de coordenadas em conjunto.

6.1.1.2. Controle de Viagens

O controle de viagens é responsável por acompanhar as viagens dos diversos veículos monitorados, tornando-as ativas no horário agendado e registrando as informações relevantes oriundas das coordenadas recebidas. Cada viagem possui um estado, e esse estado pode assumir os seguintes valores:

PUC-Rio - Certificação Digital Nº 0721317/CA

- Agendada, quando uma viagem cadastrada no sistema deverá iniciar em algum momento no futuro.
- Em andamento, quando a viagem está sendo ativamente acompanhada pelo sistema no momento.
- Cancelada, quando uma viagem agendada ou em andamento é cancelada, pelo sistema, de forma automática, ou por um operador através da interface.
- Concluída, quando uma viagem é concluída corretamente.

Esta parte do sistema possui um comportamento bastante ativo. Em intervalos de tempo regulares, usualmente da ordem de um minuto, é executada uma tarefa do sistema responsável por:

- Requisitar ao banco de dados viagens agendadas que devem ser iniciadas e passá-las para o estado em andamento; e
- Verificar a existência de viagens em estados que possam ser indícios de falhas (como viagens que se encontram em andamento ou agendadas muito tempo após seu horário previsto de término, ou no segundo caso também de início, evitando que sejam acompanhadas corretamente no devido horário). Tais viagens devem receber o estado cancelada.

Cada viagem é criada para acompanhar o deslocamento do veículo por uma linha. A linha é o elemento do sistema responsável por descrever os possíveis trajetos dos veículos. Uma linha é composta por uma seqüência de pontos que descrevem seu trajeto. Alguns desses pontos estão atrelados a pontos de referência, que são marcos no trajeto da linha perceptíveis aos operadores, como praças, universidades, museus, construções que se destaquem das demais, entre outros. Os pontos inicial e final da linha são atrelados a pontos de referência especiais, chamados de terminais. Ao acompanhar o deslocamento de um veículo por uma linha o sistema deverá registrar na viagem os horários em que o veículo passou por cada um dos seus pontos de referência. Os terminais se diferem dos demais pontos de referência porque quando o sistema percebe que um veículo em viagem chegou ao terminal final da sua linha, além de registrar o horário de ocorrência, ele deve encerrar a viagem, atribuindo a ela o estado concluída.

PUC-Rio - Certificação Digital Nº 0721317/CA

6.1.1.3. Controle de Alarmes

O controle de alarmes é responsável por verificar automaticamente diversas situações de não conformidade durante as viagens, gerando alarmes para os operadores do sistema. As não conformidades verificadas pelo sistema são:

- Desvios da rota pré-estabelecida, situação na qual o sistema deve perceber através das posições recebidas do veículo que este se encontra a uma distância além da permitida do traçado original da rota.
- Excesso de velocidade, quando o veículo se encontra em velocidade maior que o mínimo entre a velocidade máxima permitida pela companhia e a velocidade máxima permitida para a via em que está trafegando.
- Atrasos e adiantamentos, quando o veículo passa por algum dos pontos de referência com uma diferença em relação ao horário previsto maior que a tolerância estabelecida.
- Não recebimento de coordenadas, quando um veículo não envia seu posicionamento por longos períodos de tempo, dando indícios de um possível mau funcionamento do equipamento.
- Veículo parado, quando por um longo período de tempo o veículo se mantém a menos que uma distância de tolerância de uma posição. Esta situação só é verificada quando o veículo deveria estar em trânsito.
- Pânico, quando um equipamento no veículo é acionado pela tripulação para comunicar ao sistema a existência de uma situação de emergência.
- Aviso do operador, quando um operador comunica através do sistema uma situação de não conformidade detectada, informando os detalhes e motivos que levaram a este alarme.
- Comboios e buracos, situação na qual dois veículos que cumprem uma mesma rota se encontram ou muito próximos (situação de comboio) ou muito distantes (situação de buraco) de acordo com tolerâncias pré-estabelecidas.

PUC-Rio - Certificação Digital Nº 0721317/CA

Algumas dessas situações de não conformidade só fazem sentido no contexto de uma viagem. Por exemplo, a verificação de desvios da rota pré-estabelecida não faz sentido se não houver uma rota, o que acontece somente durante uma viagem. Em contra partida, a verificação do não recebimento de informações de posicionamento de algum veículo deve ser sempre realizada, pois pode indicar, por exemplo, um mau funcionamento do equipamento ou algum tipo de ataque ao sistema. Portanto, o controle de alarmes deve levar em consideração se o veículo está em viagem ou não para verificar algumas situações de exceção. As situações de exceção dependentes de alarmes são: desvios da rota pré-estabelecida; atrasos e adiantamentos; veículo parado; e comboios e buracos. Os demais controles são realizados mesmo que o veículo não esteja realizando uma viagem.

Esta parte do sistema também possui características bastante peculiares, pois é necessário manter informações relevantes sobre o estado de cada veículo monitorado com relação a cada tipo de situação de exceção, o que exige modelos bastante complexos dos eventos monitorados.

6.1.1.4. Sistema de Informação

No sistema de informação está disponível o cadastro dos diversos elementos do sistema, bem como as consultas a todos as informações adquiridas durante as tarefas de controle como situações de não conformidade, andamento das viagens ativas, informações das viagens já concluídas, entre outras. Esta é a parte do sistema que apresenta o fluxo de dados mais convencional em comparação com aplicações *web* tradicionais, onde um servidor recebe requisições, as processa e retorna uma resposta ao cliente.

6.1.2. Características de Escalabilidade e Balanceamento de Carga

Esses componentes, embora muitas vezes sejam executados em um mesmo servidor, foram criados de forma bastante independente, para que seja possível criar instâncias do sistema onde cada um desses componentes execute em uma máquina diferente, por questões de escalabilidade e balanceamento de carga. Pode ser tomada como exemplo a seguinte configuração, envolvendo o controle de uma frota de 100 veículos e utilizando 4 servidores:

PUC-Rio - Certificação Digital Nº 0721317/CA

- Servidor A: Executa um módulo de recebimento de coordenadas, recebendo informações dos veículos 1 até 50.
- Servidor B: Executa outro módulo de recebimento de coordenadas, recebendo informações dos veículos 51 até 100.
- Servidor C: Executa o controle de viagens e alarmes para todos os veículos.
- Servidor D: Executa apenas o sistema de informações.

Todos os servidores devem compartilhar um mesmo banco de dados, que se necessário também deve utilizar recursos de distribuição de carga.

6.2. Contexto da Aplicação da Ferramenta

O projeto em questão já possui algumas versões entregues, é realizado de forma contínua, e seu desenvolvimento se iniciou em abril de 2007. Trabalharam na equipe de desenvolvimento do sistema ao longo de todo o projeto 6 pessoas, entre as quais estão mestres em informática e engenheiros da computação.

6.2.1. Oportunidade para a Aplicação

A oportunidade para a realização deste estudo se originou da necessidade de uma manutenção evolutiva no sistema alvo, mais especificamente no controle de alarmes. A evolução em questão foi a inclusão de novos controles para dois tipos de situações de não conformidade, comboios e buracos. Antes da existência desses alarmes, todo o controle de não conformidades do sistema era realizado levando-se em consideração as informações de posicionamento de cada veículo individualmente, sem que houvesse a necessidade de cruzamento de dados de diferentes veículos em diferentes viagens. Esses novos requisitos tornaram necessário um processo de reengenharia dos módulos do controle de alarmes, a fim de adequá-lo a estas novas necessidades. Portanto, possuir uma forma de realizar os testes de forma simples tornou-se importante para garantir a qualidade dos artefatos que seriam modificados nesse processo. Havia também a necessidade de se especificar não somente os novos alarmes, mas também os antigos, por conta de algumas modificações requisitadas em decorrência da experiência de uso do sistema em situações reais.

PUC-Rio - Certificação Digital Nº 0721317/CA

6.2.2.
O Processo de Desenvolvimento

O processo de desenvolvimento adotado nesse projeto seguia a linha das metodologias ágeis, mais especificamente semelhante a *scrum* [32]. No processo, é utilizado um sistema de controle de tarefas, o JIRA [33], no qual todo o trabalho a ser feito é cadastrado na forma de tarefas, que podem ser sub-divididas em tarefas menores e organizadas por versão do produto na qual devem constar, prioridade e complexidade de implementação. Este sistema funciona como uma espécie de *backlog* da metodologia *scrum* [32].

As entregas de partes funcionais do sistema para o cliente eram realizadas em iterações de aproximadamente duas a três semanas. Ao início de cada iteração um representante da equipe do projeto é responsável por negociar com um representante do cliente as funcionalidades que deverão integrar a próxima entrega, com base nas dificuldades de implementação e nas necessidades do cliente. Cada uma das tarefas necessárias para o desenvolvimento das funcionalidades da próxima iteração é atribuída a um integrante do projeto, que passa a ser responsável pela sua conclusão no prazo estabelecido. Portanto, qualquer indício de que alguma dificuldade impedirá a conclusão de uma tarefa no prazo deverá ser comunicado ao representante da equipe o mais rápido possível para que seja possível renegociar com o representante do cliente a entrega da funcionalidade.

Antes da aplicação da ferramenta o processo de desenvolvimento não possuía nenhum tipo de automatização de testes. Os testes eram realizados de forma manual, seguindo roteiros de testes definidos após a codificação de cada funcionalidade. A realização dos testes de forma manual muitas vezes fazia com que após ocorridas alterações em módulo, apenas o módulo alterado fosse testado, o que culminava em muitos defeitos após a integração dos diversos módulos. Após as entregas, ao final de cada iteração, o cliente era responsável por realizar testes de aceitação e reportar à equipe de desenvolvimento as modificações necessárias por conta de falhas detectadas ou mudanças nas especificações. Como o processo de controle de qualidade era falho nas etapas iniciais, uma quantidade elevada de defeitos chegava a etapa de testes de aceitação e muitas falhas eram reportadas.

PUC-Rio - Certificação Digital Nº 0721317/CA

A ferramenta de controle de tarefas era utilizada como única documentação formal dos requisitos do sistema, e a maioria da documentação das funcionalidades era feita na forma de descrições textuais. A ferramenta de controle permite que documentos sejam anexados às tarefas, mas a funcionalidade não era utilizada com freqüência para vincular especificações às tarefas.

O projeto utiliza também um sistema de controle de versão, o CVS [34], e sempre que novas versões de arquivos são enviadas os comentários relacionados ao envio desses arquivos devem conter o código da tarefa que motivou a criação da nova versão, de modo que seja possível manter a rastreabilidade entre o código no controle de versão e as tarefas no sistema de controle de tarefas. O sistema de controle de tarefas possui uma funcionalidade de integração com o controle de versão que permite a visualização desse relacionamento.

6.3. A Aplicação da Ferramenta ao Projeto

PUC-Rio - Certificação Digital Nº 0721317/CA

Antes de iniciar a utilização da ferramenta de testes, foi necessário integrá-la ao projeto e treinar os membros da equipe para a sua correta utilização.

O processo de integração da ferramenta de teste ao projeto envolve os seguintes passos:

- A inclusão da biblioteca do *framework* como dependência do projeto.
- A criação dos módulos de linguagem e de exibição de resultados.
- A criação dos arquivos de configuração que realizam a ligação dos módulos criados.

Como já se tinha uma versão bastante desenvolvida dos módulos de máquina de estados, inicialmente não houve muito esforço na etapa de criação de módulos de linguagem. O mesmo pode ser dito da criação do módulo de exibição de resultados.

O treinamento da equipe ocorreu através de uma breve explicação do funcionamento da ferramenta, com alguns exemplos práticos. Um grande facilitador do treinamento foi o fato da equipe do projeto ter acesso aos desenvolvedores da ferramenta de testes durante todo o projeto, de forma que a proximidade simplificou o contato e tornou mais rápido o aprendizado.

6.3.1. Mudanças no Processo de Desenvolvimento

A aplicação da ferramenta ao projeto tornou necessárias algumas mudanças no processo de desenvolvimento. Uma das principais mudanças foi a inserção de uma etapa de criação dos documentos de especificação de testes, anterior à codificação. Essa etapa envolve a modelagem das situações de não conformidade descritas na seção 6.1.1.3. Para a utilização da linguagem proposta, a modelagem foi feita no formato de máquinas de estado determinísticas. A modelagem ocorreu de forma simultânea ao levantamento dos novos requisitos. Após cada uma das reuniões de levantamento dos requisitos, os desenvolvedores criavam as máquinas de estado a partir do que foi compreendido, dando início desta forma a um processo de especificação do que viria a ser construído. Nesta etapa ocorreram diversas reuniões para o refinamento dos requisitos da nova funcionalidade, e este processo de modelagem permitiu que a equipe de desenvolvedores tivesse um melhor entendimento do problema, auxiliando os clientes nesta tarefa de forma mais eficiente. A modelagem foi feita utilizando os diagramas de máquinas de estado propostos pela linguagem de modelagem UML 2 [15], próprios para a modelagem de comportamentos de sistemas de software. Para tal, foi utilizada a ferramenta de modelagem JUDE Community versão 5.3 [35], que permite que se realize a modelagem de diversos diagramas de UML [15] de forma gráfica, e que é de simples utilização, através de manipulação direta dos elementos dos diagramas.

De posse de uma especificação mais evoluída, a etapa seguinte envolveu a criação dos *scripts* de teste no formato de entrada da ferramenta. Esse processo consiste apenas na transcrição da máquina de estados gráfica na linguagem textual utilizada pela ferramenta. Os documentos de especificação de testes criados foram adicionados ao controle de versão utilizado no projeto, com comentários que os ligam às tarefas relacionadas, de forma que é possível ver os documentos de especificação relativos a cada tarefa.

Imediatamente anterior à etapa de codificação, foi inserido um procedimento envolvendo a criação dos módulos específicos de teste para cada script. Na prática, essa etapa ocorreu de forma praticamente paralela ao desenvolvimento dos *scripts* e máquinas de estados, uma vez que os métodos

PUC-Rio - Certificação Digital Nº 0721317/CA

presentes no módulo correspondem de forma quase direta aos estados e eventos criados, tornando os processos de criação bastante interligados.

Ao longo do desenvolvimento, alterações nos requisitos provocaram modificações no documento de especificação e conseqüentemente nos *scripts* de teste. Tal procedimento tornou necessária a atenção por parte da equipe para co-evoluir os documentos, de forma que a máquina de estados gráfica não ficasse diferente da sua versão textual e executável.

Os procedimentos de testes foram os que sofreram as maiores modificações. A ferramenta de testes foi configurada nos ambientes de desenvolvimento, de forma que durante o desenvolvimento era possível executar os testes para verificar aquilo que estava sendo construído. Os desenvolvedores ficaram responsáveis também por executar a massa de testes antes de enviarem novas versões de arquivos para o sistema de controle de versão, como forma de garantir que o código enviado atendia aos comportamentos especificados. Esse procedimento é também uma forma de realizar testes de integração, pois como o desenvolvedor deve sempre realizar suas alterações tendo como base as versões mais atuais dos arquivos no controle de versão, ao executar os testes para enviar novas versões está se testando na verdade a versão mais atual do sistema.

6.3.2. Especificação dos Testes

Ao todo foram modeladas dez máquinas de estado, sendo uma para descrever os comportamentos dos controles de cada uma das situações de não-conformidade descritas na seção 6.1.1.3. As figuras 23 e 24 mostram dois exemplos das máquinas de estados criadas, para os controles de excesso de velocidade e atrasos respectivamente.

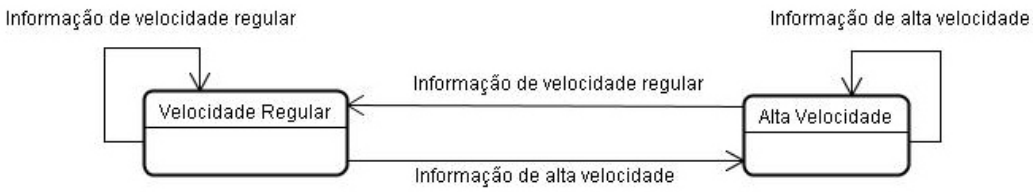

Figura 23: Exemplo de máquina de estados: Alarmes de alta velocidade

PUC-Rio - Certificação Digital Nº 0721317/CA

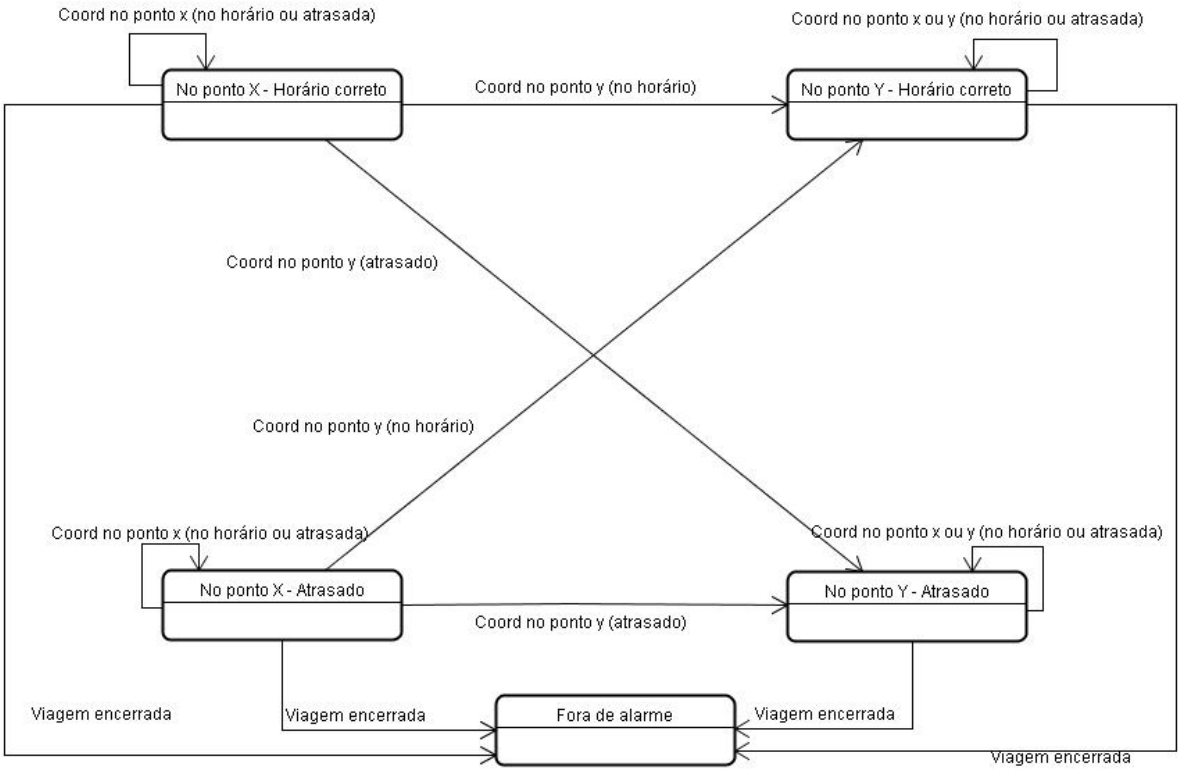

Figura 24: Exemplo de máquina de estados: Alarmes de atraso

De posse dos arquivos de modelagem das máquinas de estado de cada controle, foram criados os arquivos de *script*. Os arquivos de script possuem as definições das máquinas de estado na linguagem textual proposta nesse trabalho, de forma que um arquivo de *script* deveria ser criado para cada máquina modelada. Existiram exceções apenas nos controles de atrasos e adiantamentos e nos controles de comboios e buracos. Foi possível separar esses quatro controles em dois grupos, uma vez que atrasado e adiantado possuem apenas pequenas diferenças internamente. O mesmo vale para os controles de comboio e buraco, que possuem grande semelhança. Foram, portanto, oito arquivos de *scripts* criados.

As figuras 25 e 26 mostram dois exemplos desses *scripts*, para os testes dos alarmes de alta velocidade e atraso.

PUC-Rio - Certificação Digital Nº 0721317/CA

```
1  //Definindo máquina de estados
2
3  existe transição velocidadePermitida posicaoComAltaVelocidade velocidadeAlta
4  existe transição velocidadeAlta posicaoComAltaVelocidade velocidadeAlta
5  existe transição velocidadePermitida posicaoComVelocidadeNormal velocidadePermitida
6  existe transição velocidadeAlta posicaoComVelocidadeNormal velocidadePermitida
7
8  existe transição velocidadeAlta viagemEncerrada velocidadeAlta
9  existe transição velocidadePermitida viagemEncerrada velocidadePermitida
10
11 // Teste ocorrendo alarme
12 vai para velocidadePermitida
13 acontece evento posicaoComVelocidadeNormal
14 acontece evento posicaoComAltaVelocidade
15 acontece evento posicaoComAltaVelocidade
16 acontece evento posicaoComVelocidadeNormal
17 acontece evento posicaoComVelocidadeNormal
18
19 // Teste com viagem
20 vai para velocidadePermitida
21 acontece evento posicaoComAltaVelocidade
22 acontece evento viagemEncerrada
23 vai para velocidadePermitida
24 acontece evento posicaoComVelocidadeNormal
25 acontece evento viagemEncerrada
```

Figura 25: Exemplos de script: Alarmes de alta velocidade

PUC-Rio - Certificação Digital Nº 0721317/CA

```
1  //Definindo máquina de estados
2
3  //Transições a partir de X
4  existe transição pontoXnoHorario coordenadaPontoXnoHorario pontoXnoHorario
5  existe transição pontoXnoHorario coordenadaPontoXatraso pontoXnoHorario
6  existe transição pontoXatrasado coordenadaPontoXnoHorario pontoXatrasado
7  existe transição pontoXatrasado coordenadaPontoXatraso pontoXatrasado
8
9  existe transição pontoXnoHorario coordenadaPontoYnoHorario pontoYnoHorario
10 existe transição pontoXnoHorario coordenadaPontoYatraso pontoYatrasado
11 existe transição pontoXatrasado coordenadaPontoYnoHorario pontoYnoHorario
12 existe transição pontoXatrasado coordenadaPontoYatraso pontoYatrasado
13
14 //Transições a partir de Y
15 existe transição pontoYnoHorario coordenadaPontoXnoHorario pontoYnoHorario
16 existe transição pontoYnoHorario coordenadaPontoXatraso pontoYnoHorario
17 existe transição pontoYnoHorario coordenadaPontoYnoHorario pontoYnoHorario
18 existe transição pontoYnoHorario coordenadaPontoYatraso pontoYnoHorario
19
20 existe transição pontoYatrasado coordenadaPontoXnoHorario pontoYatrasado
21 existe transição pontoYatrasado coordenadaPontoXatraso pontoYatrasado
22 existe transição pontoYatrasado coordenadaPontoYnoHorario pontoYatrasado
23 existe transição pontoYatrasado coordenadaPontoYatraso pontoYatrasado
24
25 //Transições de viagem encerrada
26 existe transição pontoXnoHorario viagemEncerrada foraDeAlarme
27 existe transição pontoYnoHorario viagemEncerrada foraDeAlarme
28 existe transição pontoXatrasado viagemEncerrada foraDeAlarme
29 existe transição pontoYatrasado viagemEncerrada foraDeAlarme
30
31
32
33 // Teste ocorrendo alarme
34 vai para pontoXnoHorario
35 acontece evento coordenadaPontoXnoHorario
36 acontece evento coordenadaPontoXatraso
37 acontece evento coordenadaPontoYatraso
38 acontece evento coordenadaPontoYnoHorario
39 acontece evento viagemEncerrada
40
41 // Teste alarme já criado
42 vai para pontoXatrasado
43 acontece evento coordenadaPontoXnoHorario
44 acontece evento coordenadaPontoXatraso
45 acontece evento coordenadaPontoYatraso
46 acontece evento coordenadaPontoYnoHorario
47 acontece evento viagemEncerrada
```

Figura 26: Exemplos de script: Alarmes de atraso

Logo nas primeiras linhas de cada um dos arquivos de *script* das figuras 25 e 26 é possível ver o resultado da transcrição das máquinas de estado gráficas para a linguagem proposta. Nos *scripts* criados, se seguiu a seguinte estrutura:

- O arquivo se inicia com a declaração da máquina de estados, na figura 25, por exemplo, da linha 1 até a linha 9.

PUC-Rio - Certificação Digital Nº 0721317/CA

- A seguir, são criados os diversos casos de teste, através de seqüências dos comandos disponíveis. Na figura 25 temos um caso de teste definido nas linhas 11 até 17 e outro nas linhas 19 até 25.

Um módulo específico de testes foi criado para cada um dos scripts de teste, uma vez que os métodos contidos nesses módulos estão diretamente ligados aos estados e eventos modelados. De posse das máquinas de estado definidas nos scripts e com os módulos específicos de testes criados, iniciou-se a criação de cada um dos casos de teste.

Embora teoricamente seja possível modelar todas as situações relativas aos controles de cada situação de não conformidade em uma única máquina de estados, na prática cada controle pode necessitar de diversas máquinas de estado, uma vez que pode ser interessante criar diferentes situações para se exercitar diferentes comportamentos. Por exemplo, podemos considerar os dois grupos de comportamentos a seguir, relacionados ao controle de não conformidade de desvio da rota pré-estabelecida:

- Os comportamentos relacionados a informações de posição dentro e fora da rota, mostrando como o veículo entra e sai da situação de não conformidade, de acordo com a máquina de estados da figura 23.
- O encerramento dos alarmes ainda ativos criados por não conformidades desta natureza no momento do encerramento da viagem em que ocorreram.

Embora estes dois grupos de comportamentos possam ser modelados na mesma máquina de estados, fica mais claro separá-los em diferentes máquinas, uma vez que descrevem diferentes características do mesmo componente. Para resolver tais situações, foram criadas máquinas de estados disjuntas, com os vários componentes não conectados modelando as diversas situações necessárias.

De maneira geral, os casos de teste criados foram estruturados da seguinte forma:

- Força-se um estado inicial, através do comando **vai para** disponível na linguagem, passando o estado inicial desejado e criando dessa forma as pré-condições necessárias para a realização dos testes.

PUC-Rio - Certificação Digital Nº 0721317/CA

- Através de sucessivas chamadas do comando **acontece evento**, tendo como parâmetro o evento desejado, cria-se uma seqüência de eventos que devem forçar a máquina a percorrer os estados e transições que se quer verificar.
- Outros comandos de teste podem ser utilizados entre os eventos gerados para verificar situações específicas do módulo sob teste que não foram modeladas como estados na máquina.

Exemplos de casos de testes podem ser vistos na figura 25, nas linhas 11 até 17 e outro nas linhas 19 até 25.

As figuras 27 e 28 mostram exemplos dos relatórios gerados pelo módulo de exibição de resultados utilizado. Na figura 27 temos um exemplo de execução apontando falhas no módulo testado, enquanto na figura 28 temos um exemplo de execução bem sucedida.

PUC-Rio - Certificação Digital Nº 0721317/CA

```
1  1 - //Definindo máquina de estados
2  8 - // Teste ocorrendo alarme
3  9 - vai para naRota
4  Teste Ok!
5
6  10 - acontece evento posicaoForaDeRota
7  Exceção!
8  java.lang.reflect.InvocationTargetExceptionbr.com.utilities.AppBeanException: ↵
   java.lang.reflect.InvocationTargetException
9      at br.com.utilities.util.UtilitiesBean.invokeMethodQuietly(UtilitiesBean.java:1771)
10     at ↵
   br.pucrio.inf.jautotest.test.command.ExecuteGenerateEvent.executeElement(ExecuteGenerateEvent.java:104)
11     at ↵
   br.pucrio.inf.jautotest.test.command.ExecuteGenerateEvent.executeElement(ExecuteGenerateEvent.java:1)
12     at br.pucrio.inf.jautotest.test.DefaultTestEngine.runTests(DefaultTestEngine.java:113)
13     at br.pucrio.inf.jautotest.JAutotest.runTests(JAutotest.java:148)
14     at br.pucrio.inf.jautotest.main.Main.main(Main.java:85)
15 Caused by: java.lang.reflect.InvocationTargetException
16     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
17     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
18     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
19     at java.lang.reflect.Method.invoke(Method.java:597)
20     at br.com.utilities.util.UtilitiesBean.invokeMethodPrivate(UtilitiesBean.java:1608)
21     at br.com.utilities.util.UtilitiesBean.invokeMethodPrivate(UtilitiesBean.java:1574)
22     at br.com.utilities.util.UtilitiesBean.invokeMethod(UtilitiesBean.java:1542)
23     at br.com.utilities.util.UtilitiesBean.invokeMethod(UtilitiesBean.java:1513)
24     at br.com.utilities.util.UtilitiesBean.invokeMethodQuietly(UtilitiesBean.java:1755)
25     ... 5 more
26 .
27 .
28 .
29 Caused by: java.sql.BatchUpdateException: Unknown column 'occurrenceDate' in 'field list'
30     at com.mysql.jdbc.PreparedStatement.executeBatch(PreparedStatement.java:1492)
31     at ↵
   com.mchange.v2.c3p0.impl.NewProxyPreparedStatement.executeBatch(NewProxyPreparedStatement.java:1723)
32     at org.hibernate.jdbc.BatchingBatcher.doExecuteBatch(BatchingBatcher.java:48)
33     at org.hibernate.jdbc.AbstractBatcher.executeBatch(AbstractBatcher.java:246)
34     ... 36 more
35 O estado esperado não foi atingido.
36 Teste Falhou!
37
38 Total: 1 falhas
```

Figura 27: exemplo de relatório de execução apontando falha

Na figura 27, nas linhas 8 até 34, mostra a pilha de execução no momento em que ocorreu a falha. A falha em questão ocorreu por conta de uma consulta *HQL* [29] que utiliza uma versão de base de dados antiga. De posse da pilha de execução se torna mais simples diagnosticar o defeito responsável pela falha e corrigi-lo.

```
1  1 - //Definindo máquina de estados
2  8 - // Teste ocorrendo alarme
3  9 - vai para naRota
4  Teste Ok!
5
6  10 - acontece evento posicaoNaRota
7  Teste Ok!
8
9  11 - acontece evento posicaoForaDeRota
10 Teste Ok!
11
12 12 - acontece evento posicaoForaDeRota
13 Teste Ok!
14
15 13 - acontece evento posicaoNaRota
16 Teste Ok!
17
18 14 - acontece evento posicaoNaRota
19 Teste Ok!
20
21 15 - // Teste com viagem
22 16 - vai para naRota
23 Teste Ok!
24
25 17 - acontece evento posicaoForaDeRota
26 Teste Ok!
27
28 18 - acontece evento viagemEncerrada
29 Teste Ok!
30
31 19 - vai para naRota
32 Teste Ok!
33
34 20 - acontece evento posicaoNaRota
35 Teste Ok!
36
37 21 - acontece evento viagemEncerrada
38 Teste Ok!
39
40 Total: 0 falhas
```

Figura 28: exemplo de relatório de execução sem falhas

Na figura 28 é possível ver o resultado de uma execução bem sucedida de um *script* de testes. Na imagem é possível ver como cada comando executado é listado, com o número da linha em que apareceu no *script* à sua esquerda, e com o resultado da execução logo em seguida.

6.3.3. Resultados da Aplicação

Durante a utilização da ferramenta, foram notados problemas relacionados às informações que constavam no relatório de resultados da execução.

PUC-Rio - Certificação Digital Nº 0721317/CA

Inicialmente, na maioria dos casos, as informações disponibilizadas sobre as falhas não foram suficientes para se chegar aos defeitos que as causaram, o que tornou necessárias modificações no formato de saída. Percebeu-se que era necessário conhecer as pilhas de execução no momento de ocorrência das exceções, e estas foram então adicionadas aos relatórios. Posteriormente também se percebeu a necessidade de mais informações para responder a seguinte pergunta: “se a máquina de estados não foi para o estado esperado, em que estado ela se encontra?”. Para isso, foi adicionada à interface do módulo específico de testes comandos que devem ser implementados para informar a ferramenta em que estado a máquina se encontra. Essa informação de estado também pode ser utilizada no próprio módulo específico de testes para a criação de mensagens mais ricas em significado.

Ao todo, foram encontradas 23 falhas, o que resultou na correção de 21 defeitos. As falhas foram encontradas durante os testes realizados pelos desenvolvedores antes de enviarem novas versões de arquivos para o sistema de controle de versão. De maneira geral, o diagnóstico dos defeitos e sua posterior correção ocorreram de forma bastante rápida. Nos casos em que não houve nenhuma exceção reportada em conjunto com a falha houve uma maior dificuldade de se diagnosticar o defeito, mesmo tendo disponível a informação do estado em que a máquina se encontrava no momento do erro. Apenas 2 falhas foram detectadas após o sistema ter sido considerado pronto pela equipe de desenvolvimento, sem a utilização da ferramenta, o que ocorreu durante os testes de integração que são realizados antes das entregas ao final de cada iteração. O sistema alvo possuía ao todo 370 classes com 28224 linhas de código, das quais 12844 eram de código executável (métodos e inicializações estáticas). Os testes, entretanto, foram aplicados apenas a parte do sistema, o controle de alarmes, que possui 13 classes, com 2092 linhas de código, das quais 1364 são executáveis. O código de testes possuía ao todo 9 classes, sendo 8 concretas, para cada um dos módulos específicos de testes, e uma abstrata contendo o código comum entre os diversos módulos de testes. Ao todo foram 972 linhas de código de testes nos módulos específicos, das quais 381 executáveis. Esses dados são apresentados na tabela 1.

PUC-Rio - Certificação Digital Nº 0721317/CA

Componente	Sistema Completo	Controle de Alarmes	Código de Testes
Classes	370	13	9
Linhas de Código	28224	2092	972
Linhas de Código Executável	12844	1364	381

Tabela 1: Métricas relativas ao tamanho do sistema alvo.

As máquinas de estado modeladas graficamente foram mais utilizadas como especificação do sistema, em detrimento da utilização dos *scripts*. A linguagem textual acabou sendo de difícil compreensão para membros não técnicos do projeto, e representou certa dificuldade até mesmo para os desenvolvedores que não participaram de sua criação ou que não estavam familiarizados com os comportamentos do controle modelado.

PUC-Rio - Certificação Digital Nº 0721317/CA