4
Teste Funcional

O teste funcional de um sistema procura identificar as faltas existentes na sua programação por meio da observação da estrutura e do comportamento definidos pela sua interface pública. Por não depender da compreensão da organização interna do sistema avaliado e por desconhecer o conteúdo de seu código fonte, este tipo de teste é considerado caixa-preta (*Black-box Testing*).

O escopo dos testes é definido pelo engenheiro de software responsável (testador) ou pela equipe de testes do sistema de acordo com a demanda de seu usuário e de sua equipe de desenvolvimento. O teste pode ser tão abrangente como específico de acordo com o interesse destes atores. Muitas vezes, em virtude do cronograma de desenvolvimento, é preciso selecionar as partes do sistema que serão priorizadas nos testes. O ideal é que todas elas fossem avaliadas, mas quando isso não é possível, pelo menos as partes críticas de processamento deveriam ser testadas. Com a condução das atividades de teste de forma sistemática, os testes podem ser automatizados. No caso da automação, a proximidade do ideal fica cada vez maior visto que um maior número de testes é executado num menor intervalo de tempo (Hartman, 2002).

Para conduzir um teste funcional de forma sistemática, é importante definir um processo de teste. Esta pesquisa propõe um processo de teste funcional baseado em modelos gramaticais (capítulo 3). Este processo é dividido em fases compostas de atividades claramente definidas de modo a orientar a modelagem, a geração e a execução dos testes. Este processo pode ser manual, semi-automático ou automático, sendo assim classificado segundo a menor ou maior participação de testadores humanos nas fases e atividades de teste.

Este capítulo inicialmente descreve o processo de teste e suas fases em linhas gerais e introduz um exemplo que será utilizado por todo o capítulo (seção 4.1). Posteriormente, detalha cada uma das fases (seções 4.2, 4.3, 4.4, 4.5) e as atividades realizadas em cada fase (subseções).

4.1.
Processo de Teste Funcional

O termo processo é definido pelo dicionário *Merriam-Webster* como um "***conjunto de ações ou operações conduzindo a um fim***". Esta definição traduz bem a idéia central de processo adotada por esta pesquisa. Neste contexto, as ações ou operações correspondem às atividades e o fim, no sentido de objetivo, é o teste funcional de um sistema.

Ao planejar e elaborar o processo de teste descrito neste capítulo, além dos trabalhos relacionados (Whittaker, 2000; El-Far, 2001) em capítulos anteriores, algumas questões (Bertolino, 2007) foram consideradas. Tais questões (Tabela 10) serviram como orientação para a definição das atividades do processo.

Questão	Conceito
Por quê?	Motivo do Teste
Como?	Seleção do Teste
Quanto?	Adequação do Teste
Qual?	Foco do Teste
Onde?	Ambiente de Teste
Quando?	Momento de Teste

Tabela 10 – Principais Questões do Processo de Teste.

Uma questão está relacionada ao motivo do teste. ***Por que*** determinada estrutura ou certo comportamento do software está sendo analisado? É preciso identificar claramente o objetivo do teste para justificar a sua necessidade e o esforço dedicado para gerá-lo e executá-lo.

Outro fator relevante é a seleção do teste. ***Como*** definimos o critério de teste a ser empregado e selecionamos os casos de teste representativos deste critério? A escolha do critério tem influência direta na eficácia dos testes (Bertolino, 2007).

A adequação do teste é um fator complementar à seleção de teste. ***Quanto*** é necessário testar para garantir a observação de um critério de teste? A análise de cobertura e as medidas de confiabilidade são mostras claras deste ponto e auxiliam na decisão de quando parar a execução dos testes.

Adotando a terminologia de teste (seção 2.1), o termo sistema em teste pode corresponder à íntegra deste sistema ou as suas partes, representadas por subsistemas e por unidades de programação. Isto conduz a uma nova pergunta. ***Qual*** é o foco do processo de teste funcional? Em virtude dos níveis de composição de um sistema, o foco do processo de teste funcional varia. Quando o foco é a unidade, um módulo, uma classe ou um componente são exercitados. Quando o foco é a integração, o relacionamento e o comportamento das unidades são avaliados em conjunto. Quando o foco é o sistema completo, sua estrutura, seu comportamento e suas interações com o ambiente são verificados.

O contexto de execução dos casos de teste também tem impacto nos seus resultados. ***Onde*** são executados os casos de teste? Isto se reflete na geração do ambiente de teste. A configuração inicial para realização de um conjunto de testes e a capacidade de reprodução de testes são diretamente afetadas por esta questão.

Finalmente, a integração dos testes ao ciclo de desenvolvimento de um sistema não só influencia nos custos dos testes como na qualidade do produto entregue ao usuário final do sistema. ***Quando*** os testes podem ser realizados? Os momentos de interação entre equipe de teste e desenvolvimento são tão importantes quanto os seus trabalhos isolados.

4.1.1. Fases e Atividades

O processo de teste funcional proposto divide as suas atividades em fases. Cada fase equivale a um subconjunto de atividades que compartilham um mesmo aspecto do problema de teste de sistema. Os fluxos de atividades de uma fase estão organizados de forma a solucionar este aspecto em comum oferecendo, quando possível, alternativas.

Os aspectos principais a serem abordados pelo processo de teste funcional são a ***modelagem do sistema em teste***, a ***seleção de um cenário de teste***, a ***execução e avaliação deste cenário***. Cada um destes aspectos é tratado por uma fase do processo (Figura 26). Além destas fases, foi introduzida uma fase de preparação, denominada de ***preliminares***, cujo aspecto abordado é a compreensão do sistema em análise por meio dos requisitos presentes em sua especificação.

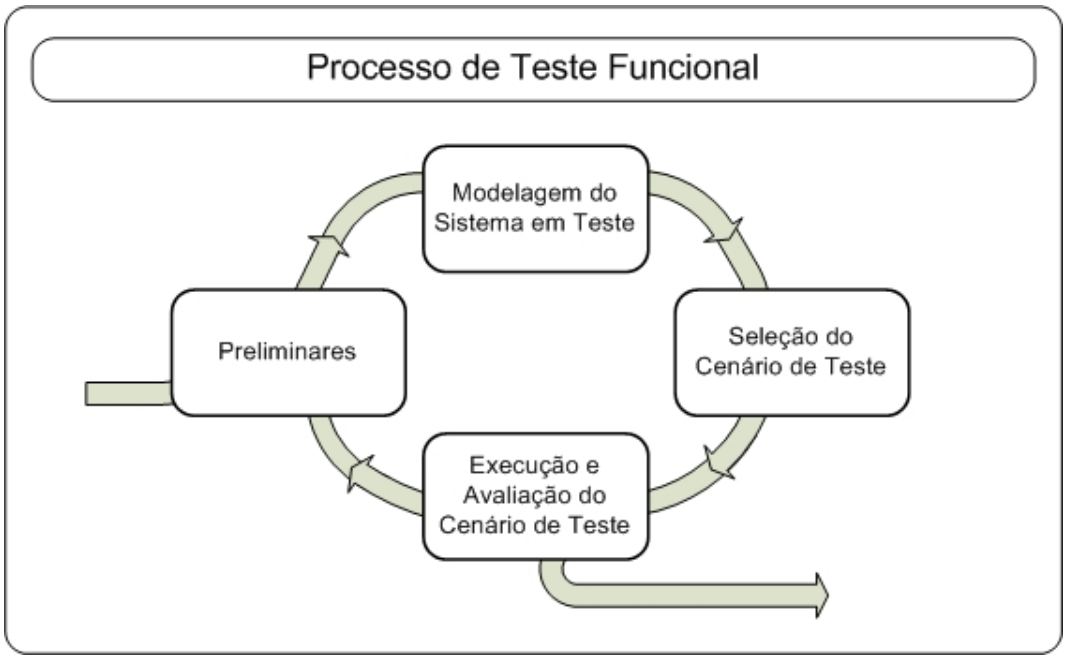

Figura 26 – Processo de Teste Funcional

As fases e as atividades do processo de teste descritas em mais detalhes nas próximas seções foram inspiradas, respectivamente, nos passos do metaprocesso de teste (seção 2.2) e nos passos do processo de teste baseado em modelos (seção 2.6.3) anteriormente descritos. O formato proposto foi concebido e elaborado a partir de composição e extensão destes passos somados a acréscimos e mudanças promovidos por idéias próprias resultantes de experiências práticas.

A descrição de cada fase apresenta ***implicitamente*** uma estrutura comum para facilitar a sua compreensão: introdução; problema; solução; conseqüências; referência. A ***introdução*** identifica sua motivação e seu objetivo. O ***problema*** revela o escopo da fase em termos de conceitos e abstrações tratados (análise). Indica quando a fase é aplicada e o contexto de aplicação. A ***solução*** define como o problema é resolvido, apresentando um projeto (*design*). As ***conseqüências*** descrevem os produtos obtidos e as decisões de projeto tomadas mediante as alternativas de projeto existentes. Apontam os custos e os benefícios da solução. A ***referência*** aponta os trabalhos relacionados que serviram como base para caracterização do problema ou que influenciaram na solução proposta.

Após a caracterização das fases, suas atividades são detalhadas. Neste ponto, o ***propósito***, os ***papéis*** envolvidos, a ***entrada*** (insumos) e a ***saída*** (resultados) das atividades são descritos. Ao final de cada atividade, um ***exemplo***

mostra em termos práticos como são realizadas. Este exemplo é contínuo e permeia todas as atividades de todas as fases.

Em cada atividade é apresentada a arquitetura da solução de software desenvolvida para resolver ou auxiliar na resolução do problema abordado pela atividade. Apresenta-se ainda a dinâmica dos participantes desta arquitetura. Os papéis desempenhados por estes participantes poderiam ser realizados num processo manual de teste por seres humanos. A nomenclatura de papéis usada pelo processo foi concebida para atender as duas realidades, humana e computacional.

A representação da arquitetura é feita com uso do diagrama de classes da UML (*Unified Modeling Language*) e a dinâmica é retratada por meio do diagrama de seqüências desta mesma linguagem (Blaha & Rumbaugh, 2005). Em cada solução proposta, destacam-se sempre os pontos fixos e os pontos de extensão da arquitetura. Além disso, quando apropriado, são tratados como aspectos arquiteturais os diferentes níveis de concretude dos testes e a dependência ou não da plataforma de execução.

4.1.2. Exemplo

Com o intuito de descrever de forma prática o processo de teste e de transformar este capítulo também num guia para sua utilização, um exemplo será desenvolvido ao longo de todas as suas fases e suas atividades. Com um exemplo em comum é possível compreender o escopo de cada fase, as necessidades (entradas) e os resultados (saídas) de cada atividade.

O exemplo escolhido foi o mesmo utilizado na descrição de modelos gramaticais (capítulo 3). Trata-se de um sistema simples para manipulação do saldo de contas bancárias de pessoas físicas ou pessoas jurídicas. O processo de teste será conduzido de forma a abordar dois cenários: o teste de uma unidade do sistema e o teste de integração de várias unidades deste sistema. A idéia é mostrar as particularidades do processo tendo como base este exemplo. Um cenário mais complexo envolvendo o teste de todo um sistema será desenvolvido somente no estudo de caso (capítulo 5) apresentado nesta dissertação.

4.2. Preliminares

Para testar um sistema é preciso saber quais são os requisitos a satisfazer segundo as perspectivas de seus usuários e seus desenvolvedores. Nesta fase, o objetivo é coletar todas as informações disponíveis sobre o sistema e selecionar dentre estas as que auxiliarão na sua modelagem para teste e na seleção, execução e avaliação de um cenário de teste.

A coleta de informações normalmente é feita a partir da documentação disponível sobre o sistema. Eventualmente, o conhecimento de usuários e de desenvolvedores também pode ser capturado. Quando não há documentação ou participantes capazes de prover informações, é preciso utilizar técnicas como a engenharia reversa para tentar identificar possíveis requisitos a partir do próprio sistema. A seleção destas informações está diretamente vinculada ao escopo de testes que precisa ser definido. Estes procedimentos devem ocorrer no início do processo de teste antes da modelagem do sistema avaliado.

Para atacar esses pontos, a fase de preliminares define duas atividades executadas em série (Figura 27). A primeira delas tem por finalidade identificar os documentos de especificação de requisitos. A segunda procura definir o escopo de avaliação. Ambas são atividades realizadas pelo testador do sistema a princípio sem auxílio de uma ferramenta de automação.

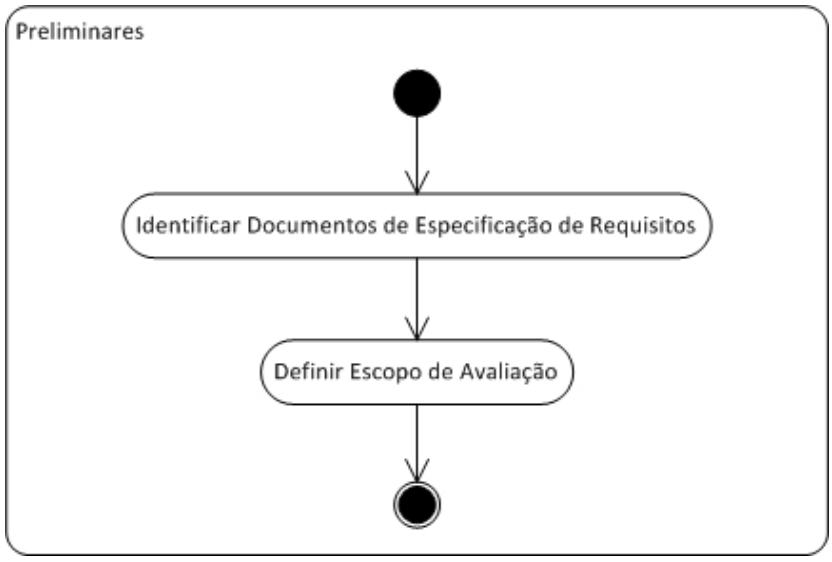

Figura 27 – Fase: Preliminares

O resultado final destas atividades é a definição de um conjunto de requisitos que serão utilizados como base para a construção dos modelos gramaticais das próximas fases.

4.2.1. Identificar Requisitos e Documentos de Especificação

O propósito desta atividade (Figura 28) é fazer o levantamento das possíveis fontes de informação provedoras de requisitos funcionais do sistema. Esta atividade é desempenhada pelo testador do sistema com auxílio do usuário e da equipe de desenvolvimento.

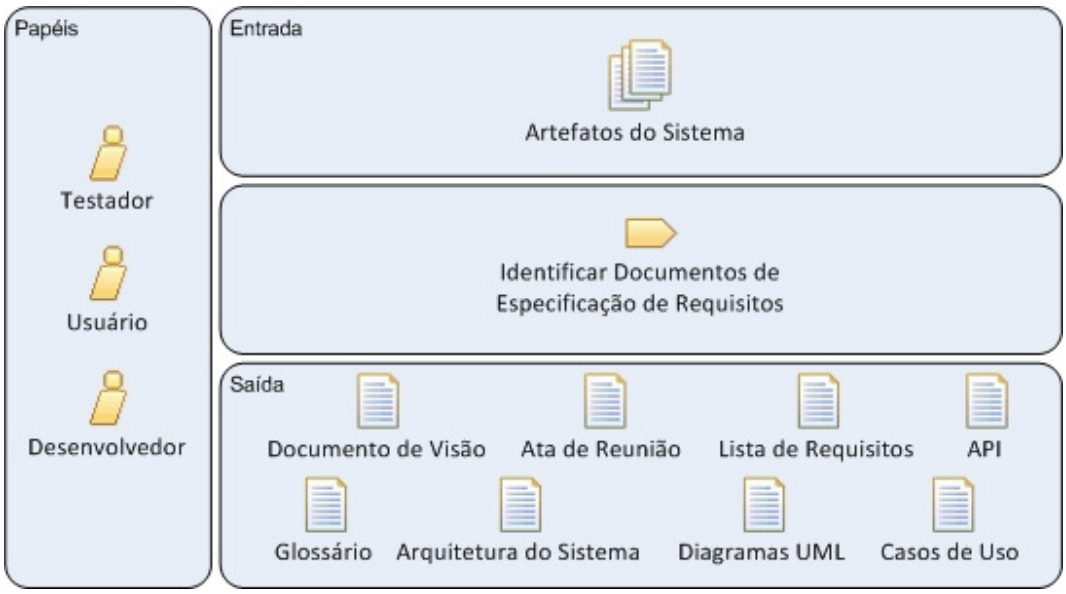

Figura 28 – Atividade: Identificar Documentos de Especificação de Requisitos

O resultado deste levantamento normalmente é uma coleção de documentos. Dentre estes documentos estão documentos de visão, atas de reunião, glossários, requisitos, descrição de casos de uso, diagramas com notação UML (*Unified Modeling Language*), documentos sobre a arquitetura do sistema, documento sobre a interface de programação.

No exemplo do sistema de contas bancárias, que foi desenvolvido utilizando a linguagem Java, os diagramas de classe, os diagramas de seqüência de casos de uso e a especificação da interface de programação (*javadoc*) foram coletados (Figura 29).

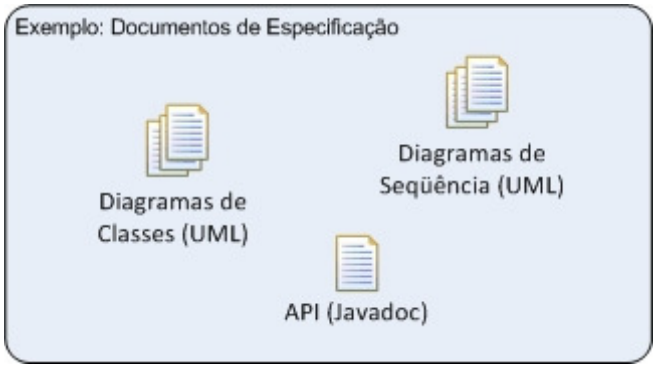

Figura 29 – Exemplo: Documentos do Sistema de Contas Bancárias

4.2.2. Definir Escopo de Avaliação

O propósito desta atividade (Figura 30) é definir que partes do sistema serão testadas e selecionar quais dos requisitos presentes nos documentos de especificação coletados são pertinentes para avaliação destas partes. Esta atividade é desempenhada pelo testador. A definição do escopo é normalmente pautada pela indicação de usuários e desenvolvedores sobre o que é mais crítico em termos de funcionalidade ou o que causa maior impacto em termos de controle e integração.

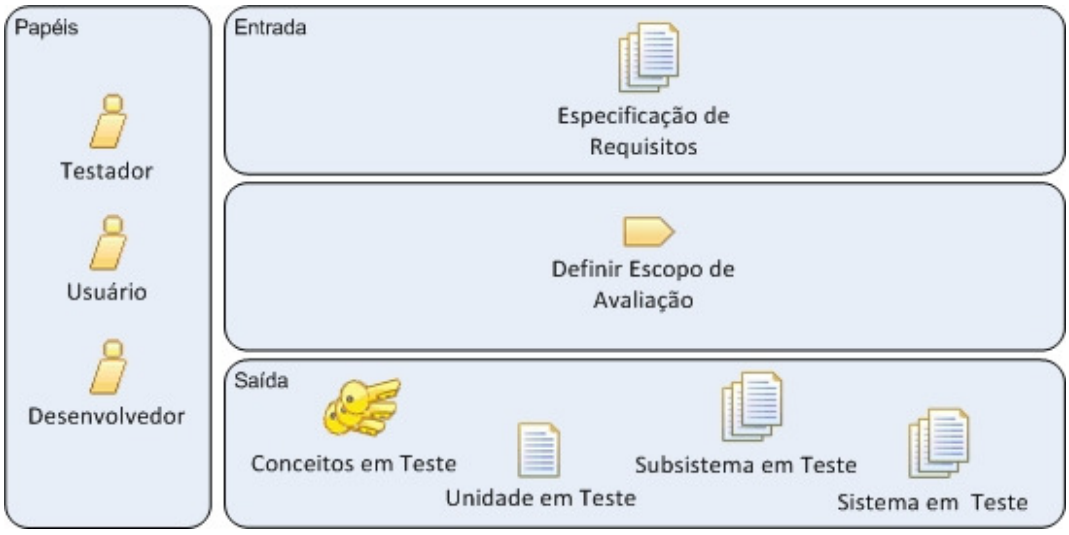

Figura 30 – Atividade: Definir Escopo de Avaliação

No caso da avaliação de uma unidade de programação de forma isolada, somente a documentação relacionada a esta unidade será considerada. O mesmo é

válido para um subsistema. No caso mais extremo correspondente ao teste de todo o sistema, toda documentação disponível será considerada.

No exemplo do sistema de contas bancárias, dois testes serão realizados. O teste de unidade de uma conta bancária e o teste de integração que envolve o serviço bancário (classe *Bank Service*), a conta bancária (classe *Account*) e o registro de transferências (classe *Account Transfer*). Para atender a este escopo de teste, o diagrama de seqüência (Figura 31) do caso de uso de ***transferência entre contas bancárias*** e a documentação de suas interfaces públicas (Figura 32) foram coletados. Como tais elementos dependem também de conceitos que não são o foco do teste, o diagrama de classe (Figura 33) do sistema também foi requerido.

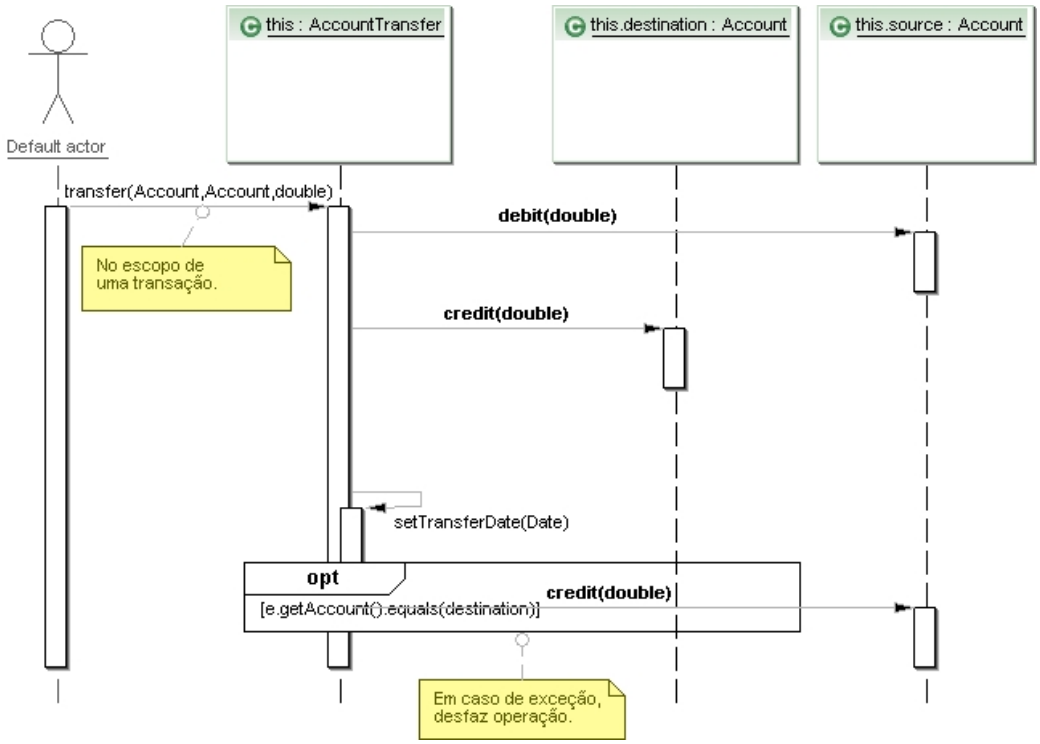

Figura 31 – Exemplo: Diagrama de Seqüência da Transferência entre Contas

Figura 32 – Exemplo: Documentação da Interface Pública de Conta Bancária

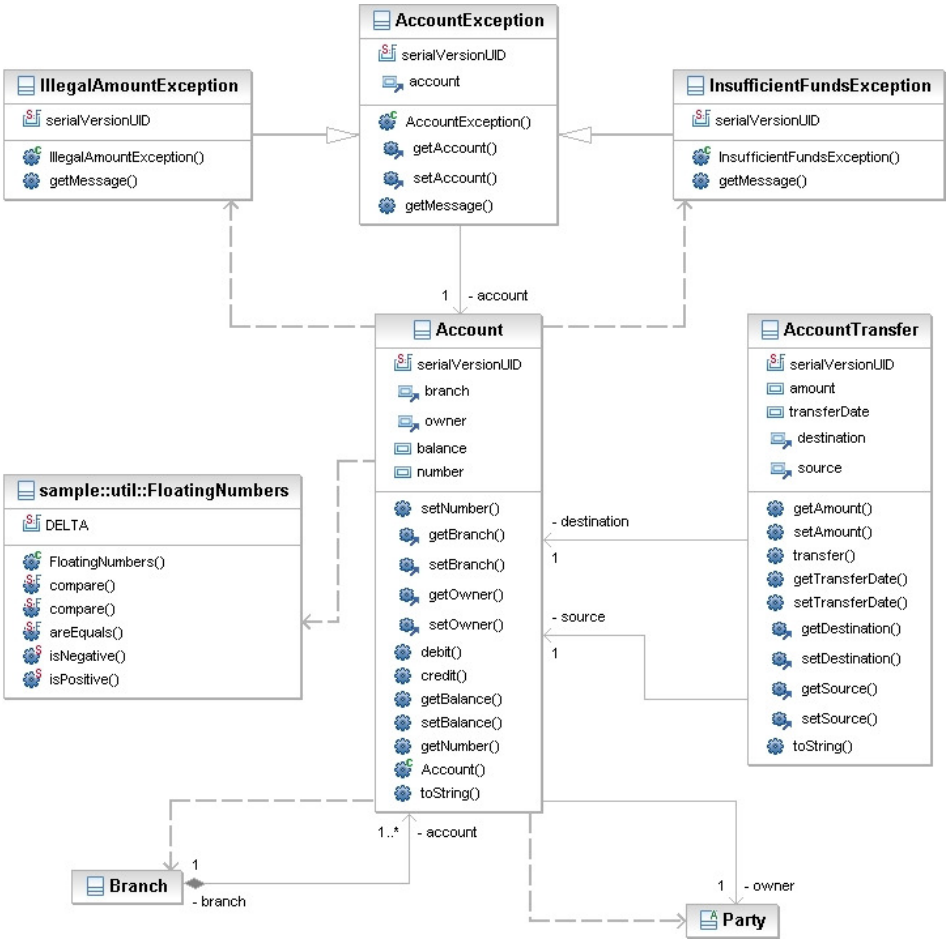

Figura 33 – Exemplo: Diagrama de Classes com foco na Conta Bancária

4.3. Modelagem do Sistema em Teste

Definido o escopo de teste de um sistema, inicia-se a etapa de sua modelagem. Neste momento, é importante compreender as responsabilidades das unidades de programação, como elas estão relacionadas e como interagem para cumprir sua função ou papel no sistema. O objetivo desta fase é construir modelos do material em análise que capturem estes aspectos e permitam a geração de casos de teste a partir de suas representações.

Nesta fase, a forma de representação dos conceitos do sistema, seus relacionamentos e seu comportamento é a maior preocupação do responsável pela condução dos testes. A representação escolhida também deve ser capaz de capturar todos esses itens.

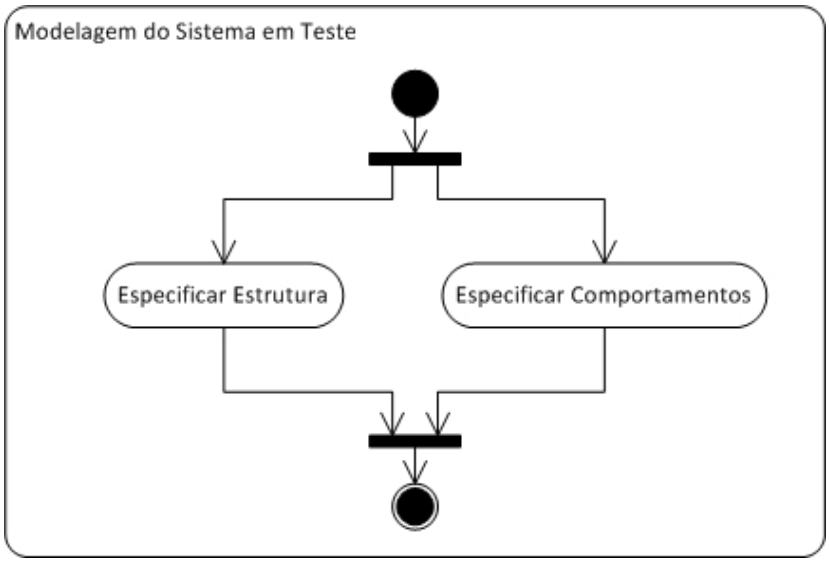

Figura 34 – Fase: Modelagem do Sistema em Teste

A solução proposta é o uso de modelos gramaticais como forma de representação tanto da parte estática, estrutura, quanto da parte dinâmica, comportamento, do sistema em teste. Duas atividades (Figura 34), que podem ser concorrentes, são planejadas e executadas nesta fase visando construir um modelo estrutural e um modelo comportamental. De posse da especificação de requisitos do sistema em teste, é possível elaborar estes modelos e construir a partir deles um modelo orientado a objeto que os represente num contexto computacional.

A automação da elaboração destes modelos gramaticais pode ser alcançada com o emprego de técnicas capazes de converter representações comumente empregadas na especificação de seus requisitos. A literatura de referência deste trabalho destaca duas destas técnicas. Uma gera uma gramática a partir de uma documentação descrita em linguagem natural (Riebisch & Hübner, 2005), outra usa diagramas UML de classe (Javed et al., 2005) como base.

Como conseqüência da solução proposta, o sistema passa a ser avaliado sob uma nova perspectiva: uma linguagem gramatical. Por ter uma forma genérica, a análise foca a especificação e é inicialmente independente de plataforma. O custo relacionado à descrição dos sistemas por modelos gramaticais é compensado pelo benefício obtido com seu formalismo que prepara a base para a automação do teste.

4.3.1. Especificar Estrutura

O propósito desta atividade é descrever os tipos abstratos de dados usados pelo sistema em teste de forma que possam ser aplicados posteriormente por um gerador de casos de teste. O diagrama de classes de um sistema é um exemplo de fonte de extração destas estruturas.

Esta atividade (Figura 35) é desempenhada pelo testador do sistema (`System Tester`), responsável pela geração do modelo estrutural, e em parte pelo interpretador da gramática (`Structural Grammar Parser`), responsável pela sua conversão num modelo orientado a objeto (seção 3.5.1).

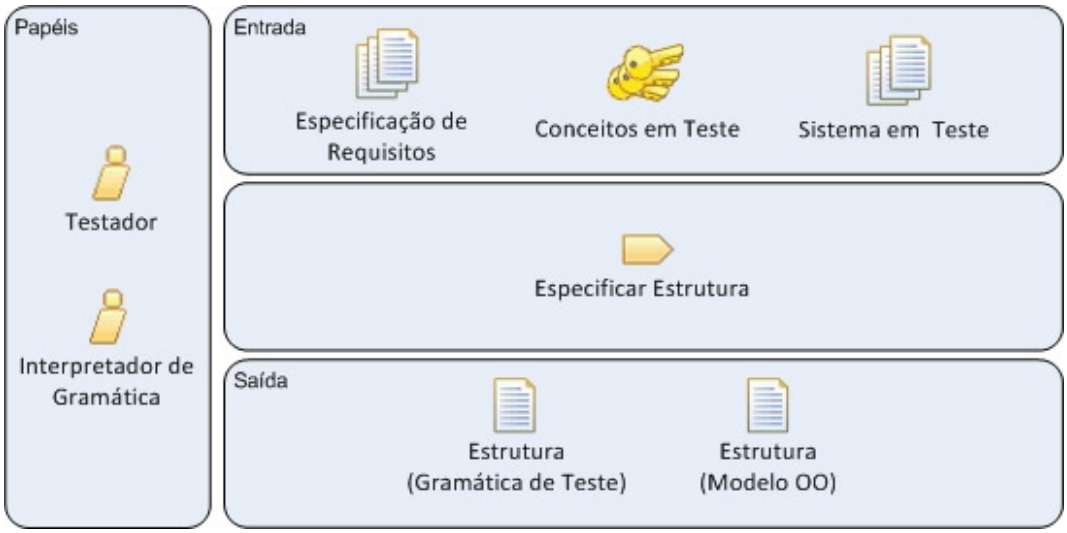

Figura 35 – Atividade: Especificar Estrutura

Neste modelo (Figura 36 e Figura 37), cada estrutura de dados é representada por um tipo (`Type`), que pode ser uma unidade de programação (`Program Unit`), um tipo primitivo (`Primitive Type`) ou uma enumeração (`Enumeration`). Este modelo ainda captura a relação de generalização entre unidades de programação (`superunits e subunits`) e a definição das propriedades (`Properties`) de uma unidade considerando a sua multiplicidade. Os relacionamentos de associação entre as unidades de programação são materializados nas propriedades das unidades relacionadas.

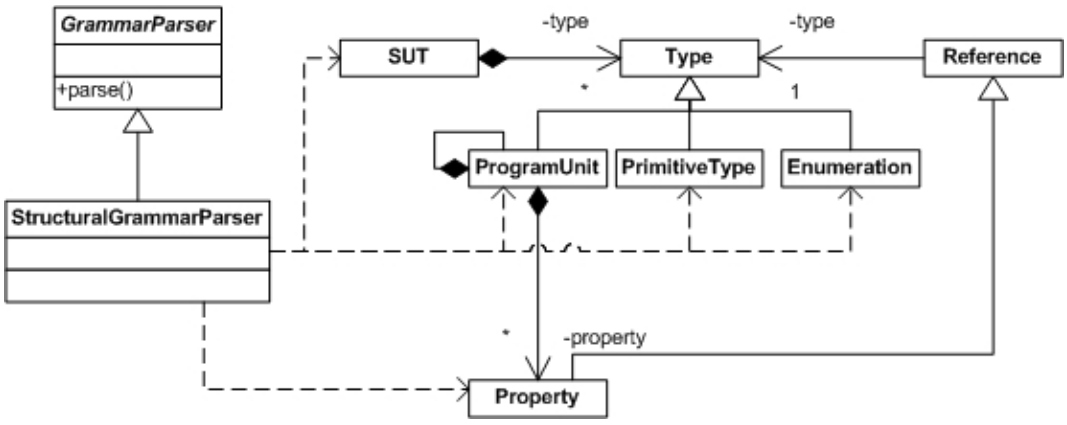

Figura 36 – Arquitetura de Representação da Estrutura do Sistema

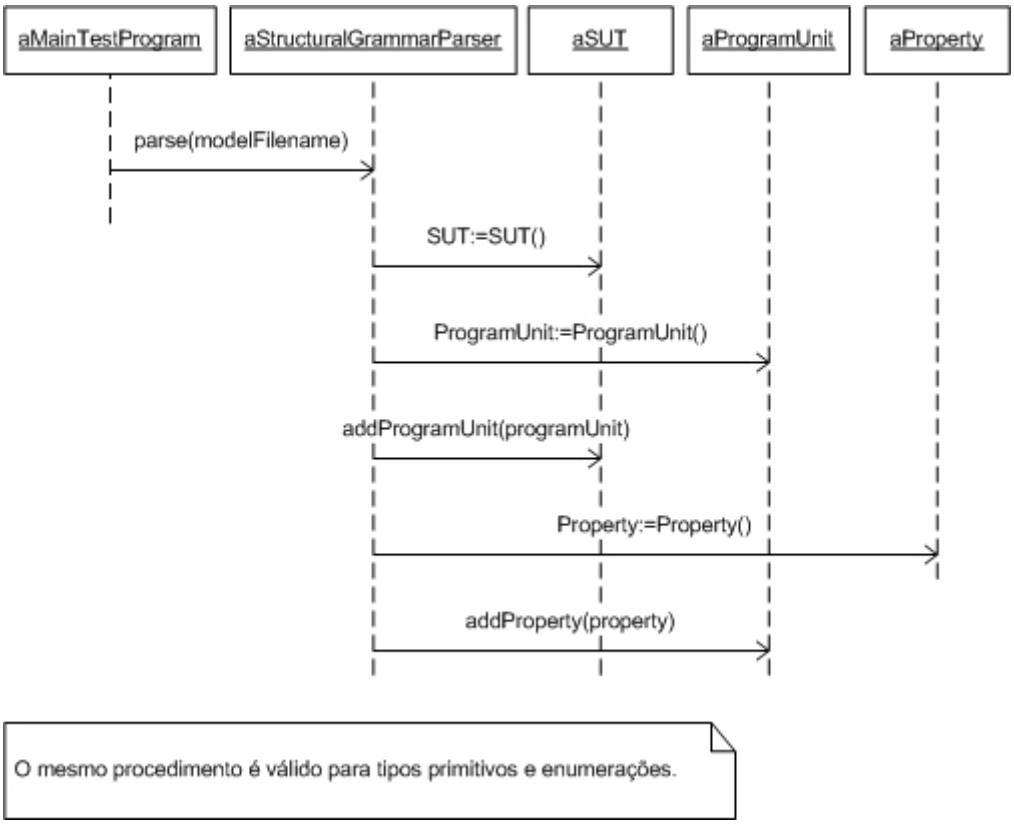

Figura 37 – A Dinâmica de Representação da Estrutura do Sistema

No exemplo do sistema de contas bancárias, a estrutura avaliada está definida pelo modelo estrutural exibido a seguir (Tabela 11).

Modelo Estrutural
`Account = ($string:number $double:balance Branch:branch Party:owner);`
`Branch = ($string:number {Account}:account Bank:bank);`
`Bank = (LegalOrganization);`
`LegalOrganization = (Organization)($string:cnpj);`
`Organization = (Party)($string:code);`
`Party = ($string name);`
`Person = ($string cpf);`
`InsufficientFundsExceptionParameters = (AccountException)();`
`IllegalAmountException = (AccountException)();`
`AccountException = ($exception)(Account:account);`

Tabela 11 – Exemplo: Modelo Estrutural

4.3.2. Especificar Comportamentos

O propósito desta atividade é descrever o comportamento do sistema a partir das funcionalidades descritas pelos documentos de sua especificação e por suas interfaces públicas. Os diagramas de seqüência UML, as descrições de casos de uso e a API do sistema são exemplos destas especificações.

Esta atividade (Figura 38) é desempenhada pelo testador do sistema (`System Tester`), responsável pela geração do modelo comportamental, e em parte pelo interpretador da gramática (`Behavior Grammar Parser`), responsável pela sua conversão num modelo orientado a objeto (seção 3.5.2) que é consolidado com o modelo gerado pela atividade anterior de especificação da estrutura de teste.

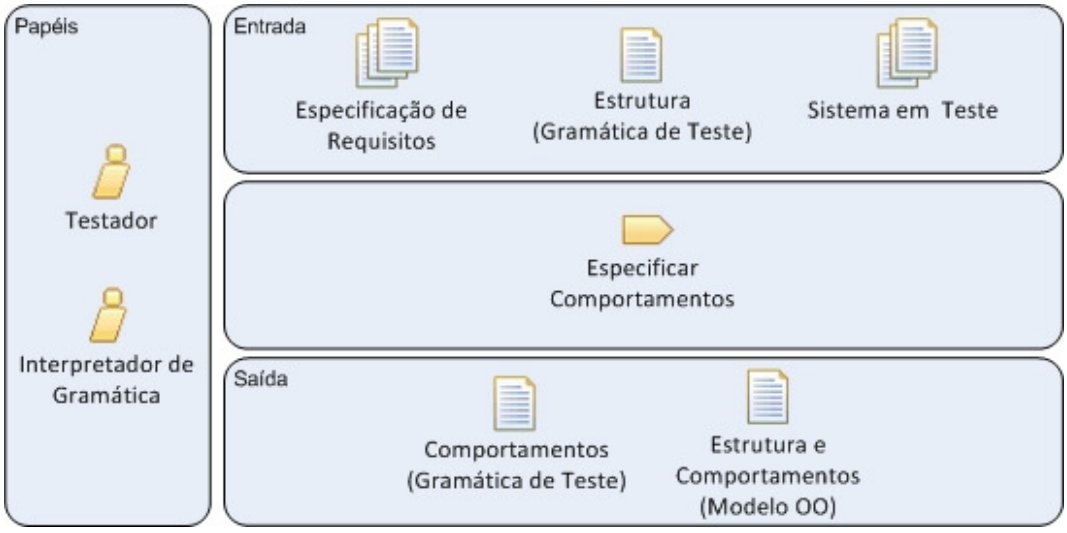

Figura 38 – Atividade: Especificar Comportamentos

Neste modelo (Figura 39 e Figura 40), cada função declarada é representada como uma operação (`Operation`) de sua correspondente unidade de programação (`ProgramUnit`). Seus parâmetros (`Parameters`), seu valor de retorno (referência a um tipo – `Type`) e possíveis exceções (conjunto de referências a um tipo – `Type`) geradas também são capturados e associados à operação.

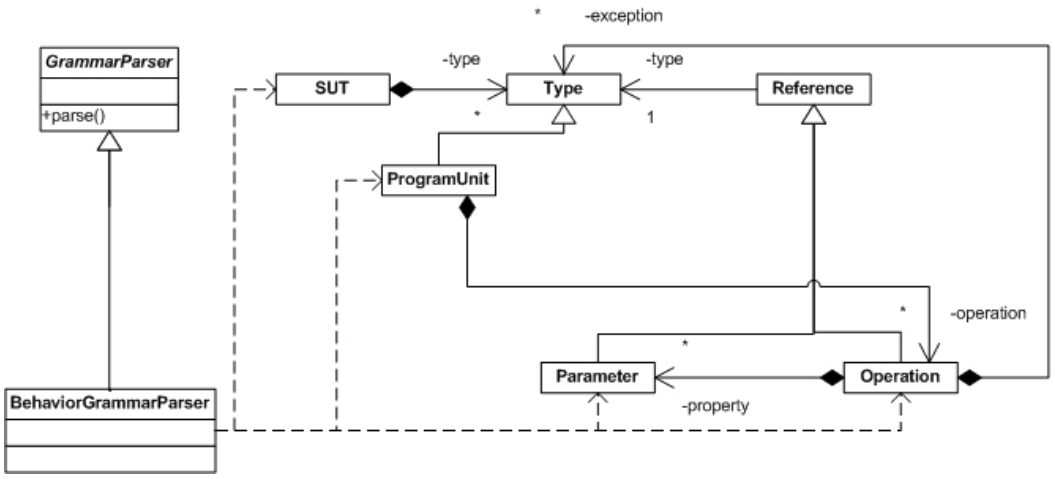

Figura 39 – Arquitetura de Representação dos Comportamentos do Sistema

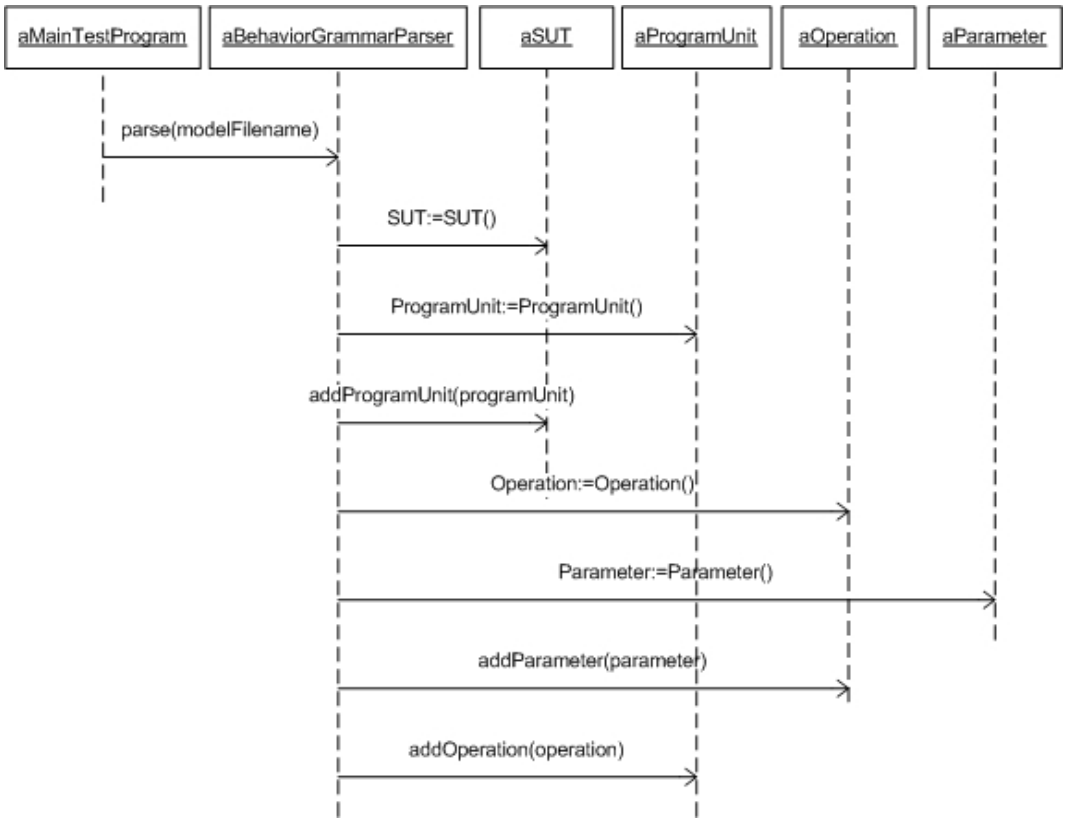

Figura 40 – A Dinâmica de Representação dos Comportamentos do Sistema

No exemplo do sistema de contas bancárias, os comportamentos avaliados estão definidos pelo modelo comportamental descrito na tabela a seguir.

Modelo Comportamental

```
Account = ( makeAccount($string:number Branch:branch
                        Party:owner $double:balance) Account;
            credit ($double:amount) $void IllegalAmountException;
            debit ($double:amount) $void
                   IllegalAmountException InsufficientFundsException;
            getBalance $void;
            getBranch branch;
            getNumber $string;
            getOwner $string;
            setBalance( $double:balance ) $void IllegalAmountException;
            setBranch( Branch:branch ) $void;
            setNumber( $string:number ) $void;
            setOwner( Party:owner ) $void; );
Branch = ( makeBranch ($string:number, Bank:bank) Branch;
           getAccount {account}; );
Bank = ( makeBank ($string:name $string:code $string:cnpj) Bank; );
Person = ( makePerson( $string:name ) Person; );
/* Só foram declaradas as classes e operações necessárias ao teste das
operações <credit> e <debit> da classe <Account>; */
```

Tabela 12 – Exemplo: Modelo Comportamental

4.4. Seleção do Cenário de Teste

Definido o escopo de avaliação e caracterizado o sistema, a próxima etapa é especificar e definir um cenário de teste. Este cenário indica que estruturas e funcionalidades serão avaliadas segundo uma estratégia de teste e contempla as particularidades do ambiente onde os testes serão executados.

O objetivo desta fase é gerar o cenário no formato de um programa que será utilizado para a execução e avaliação dos testes. Esta execução pode ser meramente conceitual, independente de plataforma, ou voltada para uma plataforma específica.

Nesta fase, é preciso definir um critério de teste que permita avaliar o sistema e que seja eficaz na detecção de faltas existentes no mesmo. Orientado por este critério, deve-se especificar os casos de teste que o atendem. Além disso, deve-se preparar um conjunto de teste, uma massa crítica que permita avaliar as funcionalidades do sistema.

O conjunto de teste é composto por casos de teste que descrevem comportamentos e usam os dados gerados para provocar mudanças nos comportamentos. É necessário estabelecer o mecanismo que permita refletir esta "sensibilidade" ao contexto. O ambiente de execução também influencia nos resultados do teste e, portanto, precisa ser modelado. O cenário de teste define um contexto representado pelo sistema, seu ambiente e um conjunto de teste.

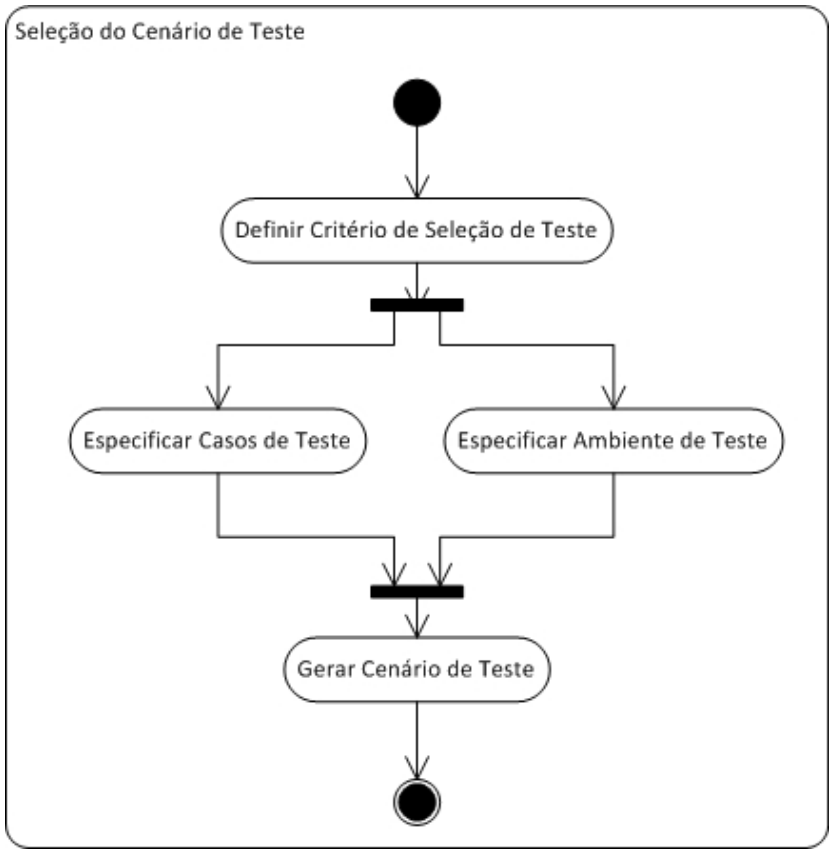

Figura 41 – Fase: Seleção do Cenário de Teste

Esta fase (Figura 41) está dividida em quatro atividades: definir critério de seleção de teste; especificar casos de teste; especificar ambiente de teste; gerar cenário de teste.

Analisando o domínio de cada função disponível na interface pública do sistema em teste é possível selecionar um critério de cobertura. Após a definição do critério, é possível escolher que técnicas de teste funcional (seção 2.4.1) serão aplicadas na geração dos dados de teste. Para cada técnica, existe um algoritmo que representa uma estratégia (Gamma et al., 1994) usada para percorrer os modelos gramaticais que representam a estrutura e os comportamentos do sistema.

Os casos de teste também são modelados com base em gramáticas. Para permitir a variabilidade, cada elemento desta gramática apresenta atributos que correspondem a restrições tanto na interpretação do seu valor quanto na definição do fluxo a ser percorrido. Os valores são limitados em faixas ou conjuntos por regras definidas pelos atributos. Este fluxo corresponde a uma seqüência de chamadas de métodos descritos como elementos não-terminais da gramática. O fluxo é alterado por meio de condições a serem testadas. Com o uso de atributos, portanto, a geração torna-se sensível ao contexto.

Até então, as especificações foram feitas numa representação genérica que independe da plataforma de execução, mas o ambiente também impõe restrições. Estas precisam ser capturadas e consideradas na geração dos casos de teste. A forma de captura também ocorre pelo uso de modelo gramatical. O modelo descreverá as configurações do sistema, a maneira como o mesmo será instanciado e quais serão as restrições de recursos utilizados pela estrutura de teste. Isto caracterizará o estado inicial de teste.

Neste momento, realiza-se um mapeamento entre os conceitos genéricos e os conceitos concretos da programação. É preciso utilizar agora uma infra-estrutura de teste compatível com a plataforma em uso.

Nesta fase foram gerados modelos gramaticais de casos de teste e de ambiente de teste adotados para gerar casos de teste executáveis propriamente ditos. A construção dos modelos é manual, mas os casos de teste executáveis são gerados automaticamente a partir destes modelos e da infra-estrutura disponível para uma plataforma específica.

O uso dos modelos gramaticais facilita a concepção dos testes uma vez que se trabalha com uma representação independente da plataforma. Mesmo quando eles são derivados automaticamente de modelos (como diagramas UML) ou de textos em linguagem natural, que compõem a especificação do sistema, eles são descritos num nível abstrato. O custo de sua elaboração é compensado pelo fato da geração dos casos de teste ser sistemática. A geração é limitada apenas pela existência ou não de infra-estrutura para uma plataforma específica.

4.4.1. Definir Critério de Seleção de Teste

O propósito desta atividade é definir o critério de cobertura de teste usando como referência o escopo de avaliação (seção 4.2.2) definido na fase de preliminares e as funcionalidades do sistema em teste, sua estrutura e suas formas de interação identificadas na modelagem do sistema em teste (seção 4.3). Esta atividade é desempenhada pelo testador do sistema (`System Tester`).

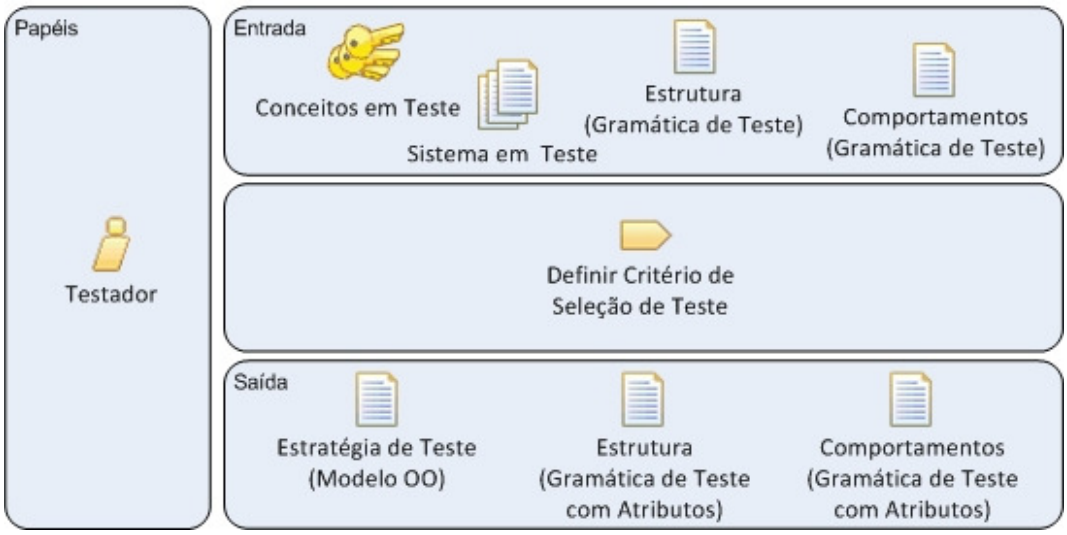

Figura 42 – Atividade: Definir Critério de Seleção de Teste

Esta atividade (Figura 42) é desempenhada pelo testador do sistema (`System Tester`). O testador do sistema avalia cada item do escopo de avaliação e identifica qual a técnica de teste funcional (seção 2.4.1) é mais adequada para avaliar a estrutura ou comportamento envolvido. Quando se trata de um teste de um sistema que já foi avaliado anteriormente, o conhecimento de um conjunto bem-definido de faltas pode auxiliar na seleção da técnica de teste. A escolha da técnica de teste funcional indicará a estratégia (`Functional Strategy`) de geração de dados de teste (Figura 43). O uso de determinadas estratégias requer a extensão da gramática utilizada na descrição dos modelos de teste. Esta extensão é representada pelo acréscimo de atributos a elementos desta gramática. Um cenário de teste será montado para cada estratégia de teste identificada.

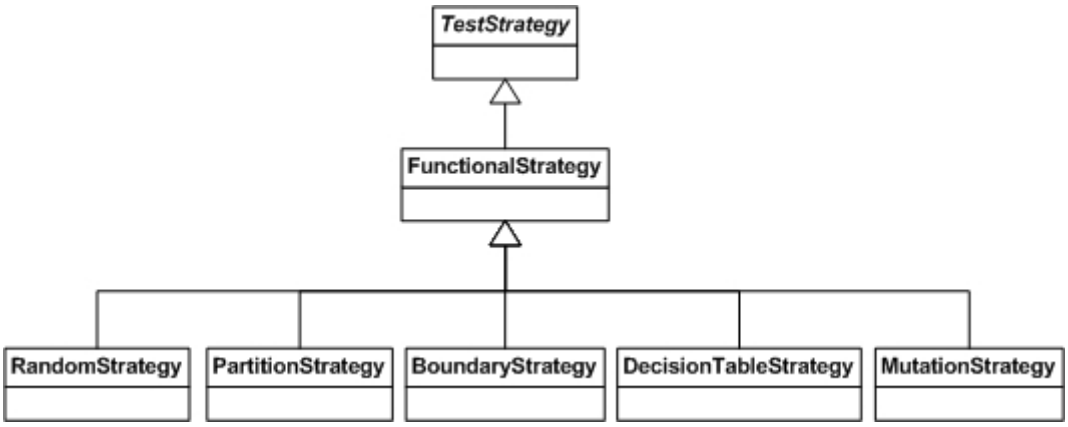

Figura 43 – Arquitetura de Estratégia de Teste

No exemplo do sistema de contas bancárias, como o escopo de avaliação são as operações de crédito e débito realizadas diretamente pela conta (`Account`) ou indiretamente pela transferência entre contas (`Account Transfer`) gerada a partir de um serviço bancário (`Bank Service`), a estratégia de valor de contorno foi selecionada. Com ela é possível avaliar se os limites das operações estão sendo contemplados corretamente, identificando a ausência de fundos e quantias inválidas, permitindo a transferência entre contas e recuperando o saldo correto.

4.4.2. Especificar Casos de Teste

O propósito desta atividade é descrever os casos de teste do sistema, formalizando o critério de teste caracterizado na atividade anterior. Esta atividade (Figura 44) é desempenhada em parte pelo testador do sistema (`System Tester`), responsável pela geração do modelo de casos de teste, e em parte pelo interpretador da gramática do modelo (`Test Case Grammar Parser`), responsável pela sua conversão num modelo orientado a objetos (seção 3.5.3).

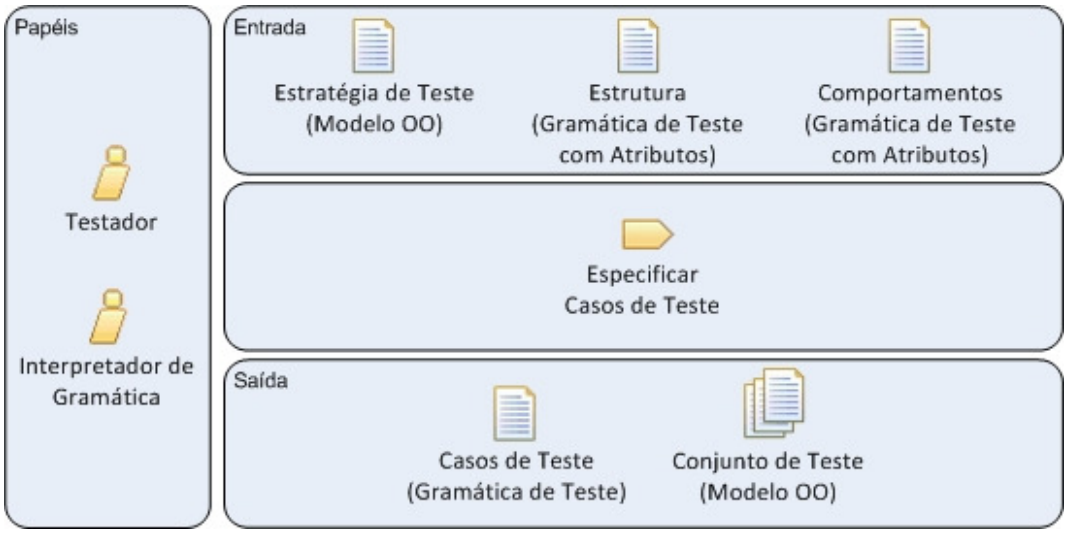

Figura 44 – Atividade: Especificar Casos de Teste

Para cada caso de teste é definido um objetivo de teste (`Test Goal`) que corresponde a uma seqüência de comandos consistentes com a descrição da estrutura e do comportamento do sistema. Estes comandos normalmente correspondem às chamadas de funções do sistema em teste.

A cada objetivo está associado um oráculo de teste (`Test Oracle`) que estabelece o resultado esperado para a execução do caso de teste. O oráculo de teste pode ser um valor calculado ou um tipo esperado, representado, respectivamente, por um elemento não-terminal da gramática do modelo. O oráculo, assim como o objetivo de teste, também pode ser descrito como uma seqüência de comandos, sendo calculado dinamicamente para revelar o resultado esperado.

Nesta atividade (Figura 45 e Figura 46), o modelo de casos de teste é analisado por um interpretador que o converte num conjunto (`Test Suite`) de casos de testes (`Test Cases`) constituídos por pares de objetivo de teste (`Test Goal`) e oráculo de teste (`Test Oracle`). Este conjunto será referenciado posteriormente pelo cenário de teste (`Test Scenario`).

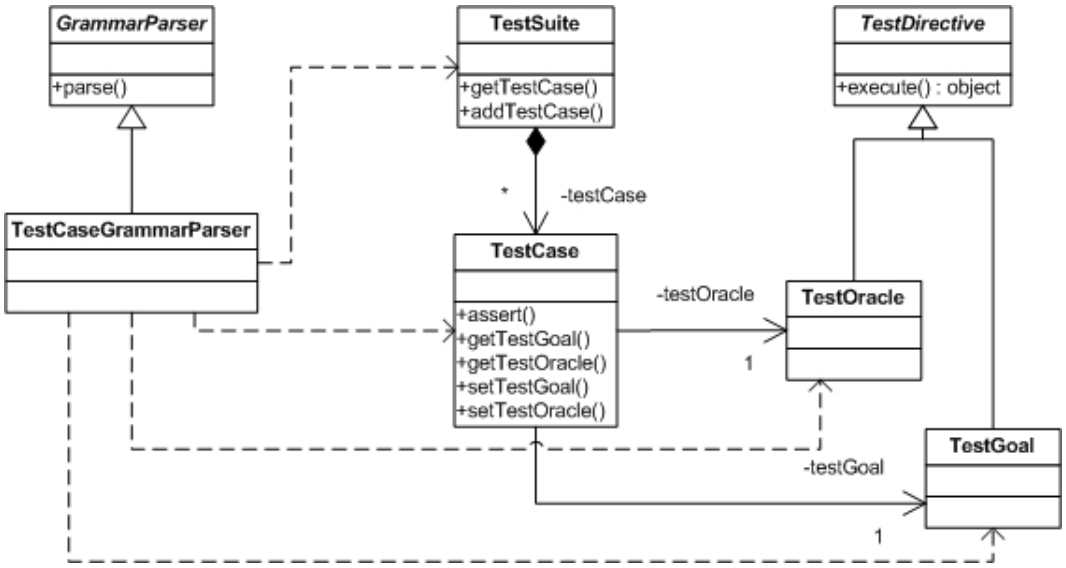

Figura 45 – Arquitetura de Representação dos Casos de Teste

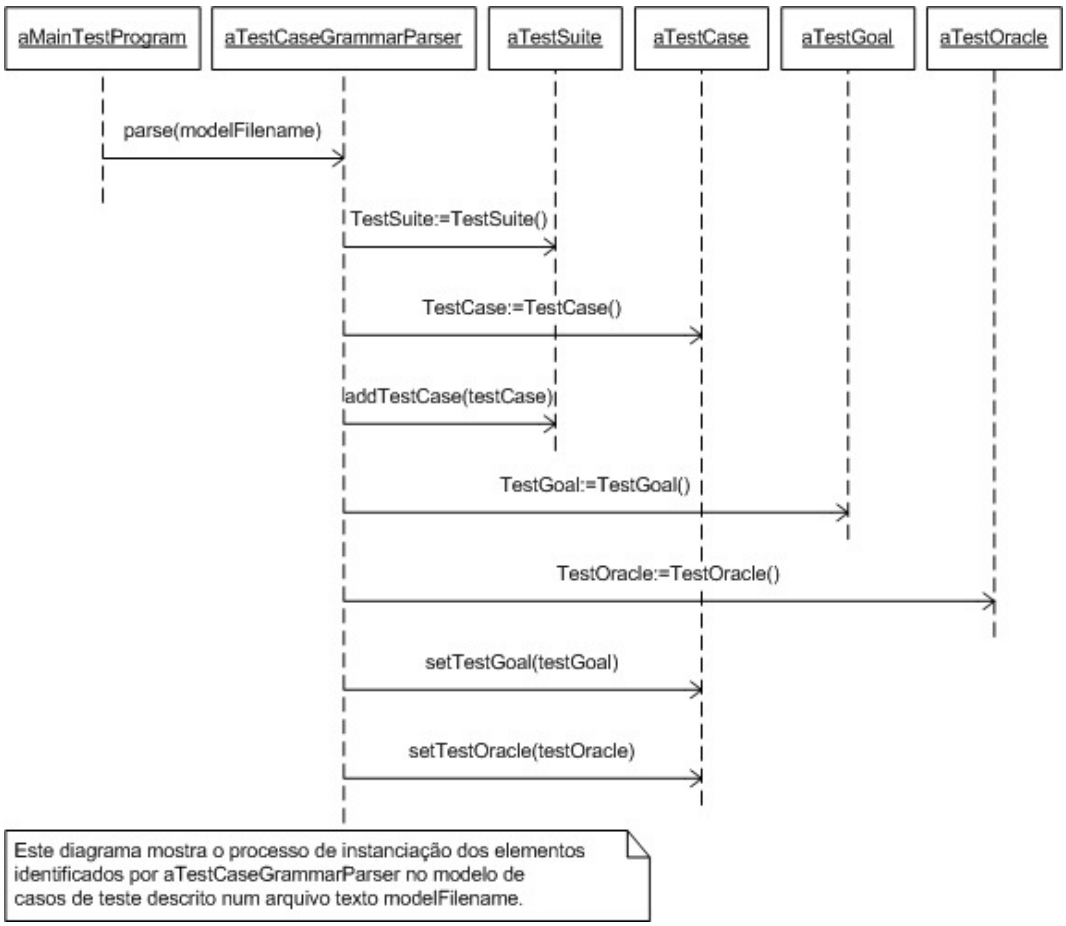

Figura 46 – A Dinâmica de Representação dos Casos de Teste

No exemplo do sistema de contas bancárias, os casos de teste estão definidos pelo modelo descrito na tabela a seguir.

Modelo de Caso de Teste
testGoal = (makeAccount(1234 aBranch aPerson 0) ?createdAccount?; ?createdAccount? credit(10); ?createdAccount? debit(20);) ?createdAccount? getBalance(); testOracle = InsufficientFundsException;

Tabela 13 – Exemplo: Modelo de Caso de Teste

4.4.3. Especificar Ambiente de Teste

O propósito desta atividade é especificar os parâmetros de configuração do ambiente de teste que serão considerados pelo cenário de teste. Quando o teste é concebido para execução numa plataforma específica, esta atividade também realiza o mapeamento entre os conceitos usados na descrição da estrutura e do comportamento do sistema e na sua representação nesta plataforma de execução. Nos casos citados, a especificação é realizada por meio de um modelo de ambiente de teste.

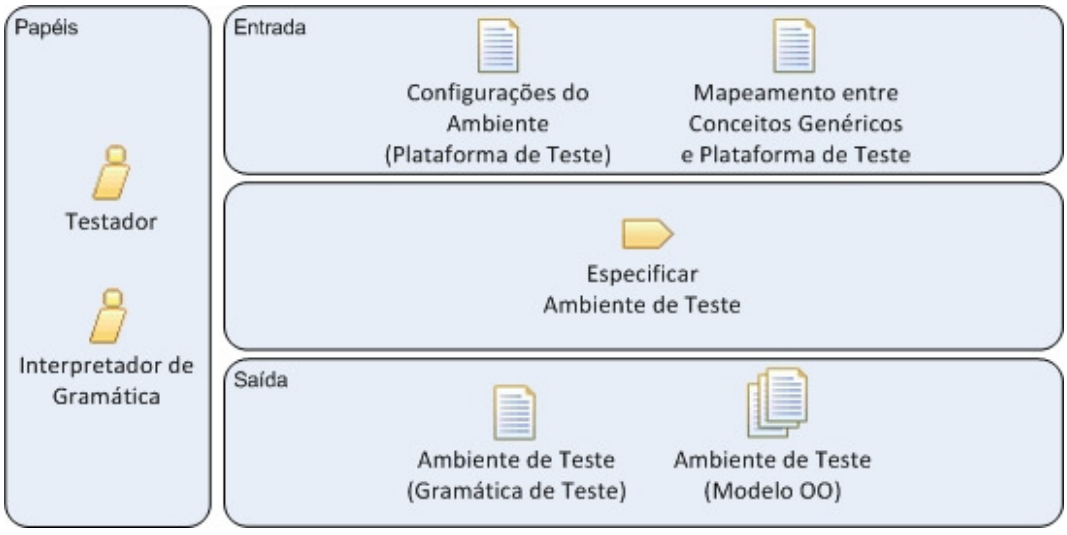

Figura 47 – Atividade: Especificar Ambiente de Teste

Esta atividade (Figura 47) é desempenhada por um interpretador de gramática de ambiente de teste (`Environment Grammar Parser`). Este interpretador converte o modelo de ambiente de teste descrito num arquivo texto

num modelo orientado a objeto (seção 3.5.4). Este modelo (Figura 48 e Figura 49) é simples e constituído por uma classe que representa o ambiente de teste (`Test Environment`) sendo as suas propriedades e valores armazenados por uma estrutura de mapa (conjunto de pares de chave-valor). Tal classe será referenciada posteriormente pelo programa que representa o cenário de teste (`Test Scenario`).

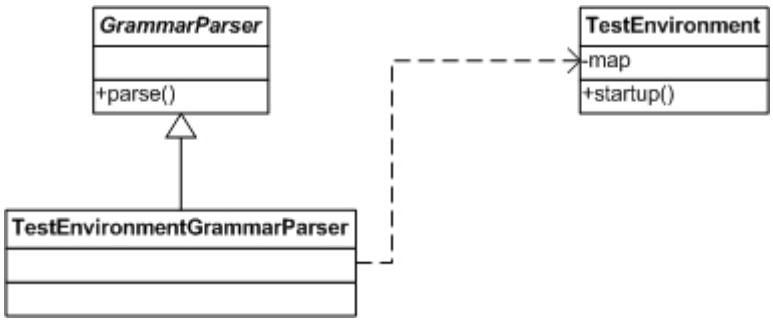

Figura 48 – Arquitetura de Representação do Ambiente de Teste

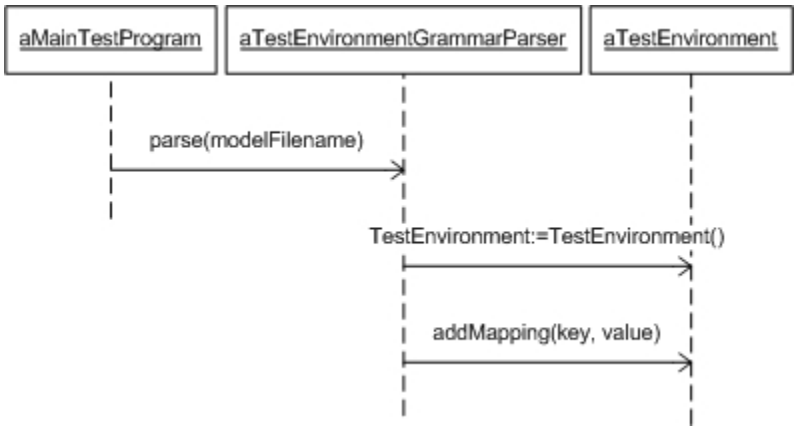

Figura 49 – A Dinâmica de Representação do Ambiente de Teste

No exemplo do sistema de contas bancárias, o ambiente de teste está definido pelo modelo gramatical descrito na tabela a seguir.

Modelo de Ambiente

`/* Configuração Inicial do Ambiente */`
`startup = ( aPerson = makePerson ('Silva' '000.000.000-00');`
`aBank = makeBank ('Banco' '001' '00.000.000/0000-00' );`
`aBranch = makeBranch ('1000' aBank) );`

```
/* Mapeamento da Estrutura para a Plataforma Java */
Person = sample.Banking.Person;
Bank = sample.Banking.Bank;
Branch = sample.Banking.Branch;s
/* ... (continua) ... */

/* Mapeamento dos Comportamentos para a Plataforma Java */
Branch.makeBank = Branch;
Bank.makeBranch = Bank;
Account.makeAccount = Account;
Person.makePerson = Person;
Account.credit = credit;
Account.debit = debit;
Account.getBalance = getBalance;
Account.setBalance = setBalance;
/* ... (continua) ... */
```

Tabela 14 – Exemplo: Modelo de Ambiente

4.4.4. Gerar Cenário de Teste

O propósito desta atividade é gerar um programa de teste que representa o cenário de teste do sistema. Esta atividade (Figura 50) é desempenhada por um construtor de programa de teste (`Test Builder`) auxiliado por interpretadores de modelos de teste (`Test Parsers`).

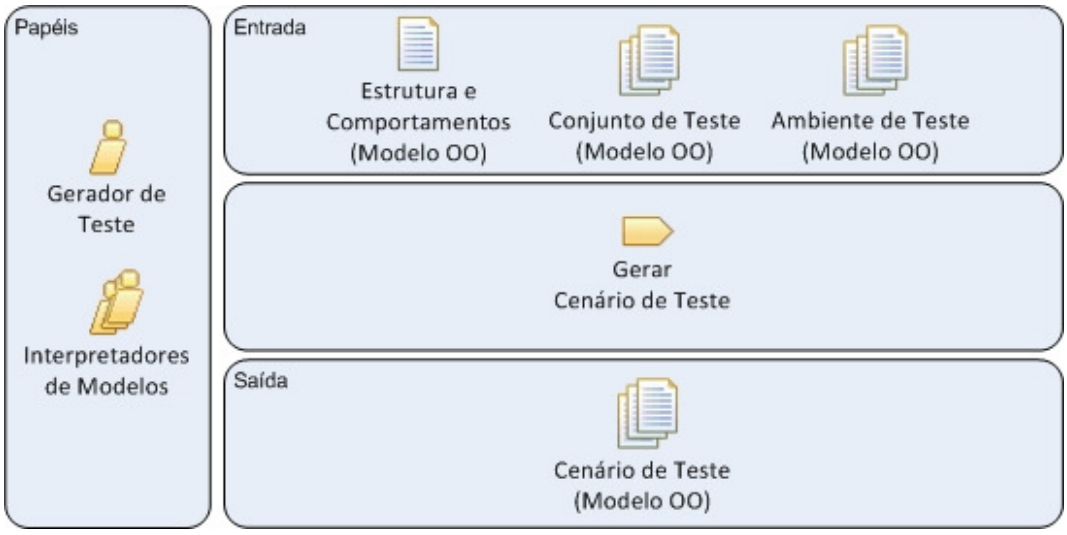

Figura 50 – Atividade: Gerar Cenário de Teste

O construtor de programa de teste coordena a ação dos interpretadores dos modelos de teste. O construtor gera o corpo do programa de teste (`Test Scenario`). Este é preenchido por métodos gerados para cada caso de teste identificado pelo interpretador de caso de teste (`Test Case Model Parser`). O corpo de cada método é descrito pela seqüência de comandos definida pelo objetivo de teste correspondente. Ao final do método, uma assertiva é gerada com base no oráculo de teste correspondente. Durante a atividade de geração do programa (Figura 51), os interpretadores de estrutura (`Structural Model Parser`), de comportamento (`Behavior Model Parser`) e de ambiente de teste (`Environment Model Parser`) são consultados para garantir a consistência do código gerado (somente estruturas ou funções definidas podem ser referenciadas) e para substituir parâmetros de configuração por seus respectivos valores.

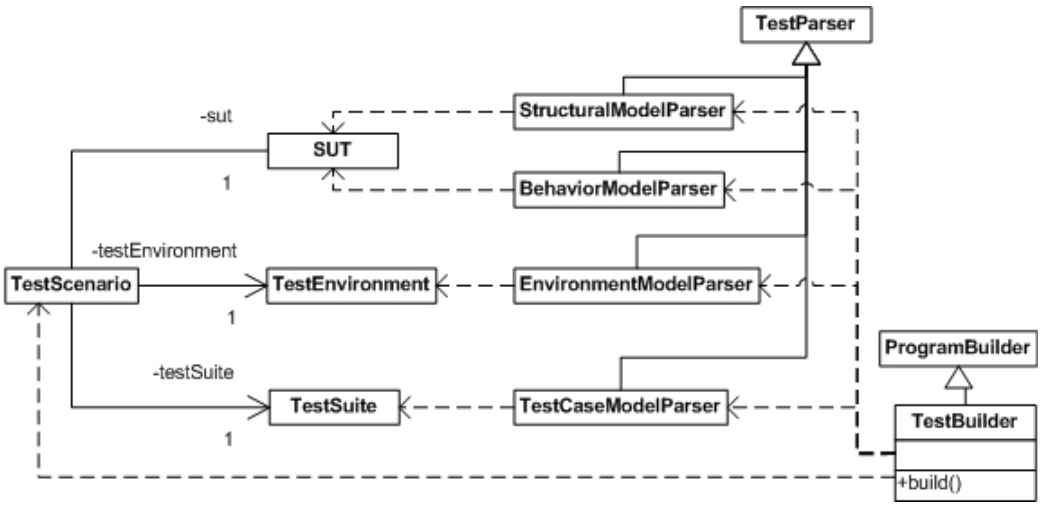

Figura 51 – Arquitetura de Geração de Cenário de Teste

4.5. Execução e Avaliação do Cenário de Teste

Para analisar a correspondência entre a codificação de um sistema e sua especificação, é preciso exercitá-lo com a finalidade de encontrar problemas funcionais. O objetivo desta fase (Figura 52) é executar os casos de teste descritos num programa de teste construído na fase de seleção de cenário (seção 4.4), observando as falhas manifestadas e identificando os possíveis erros no sistema.

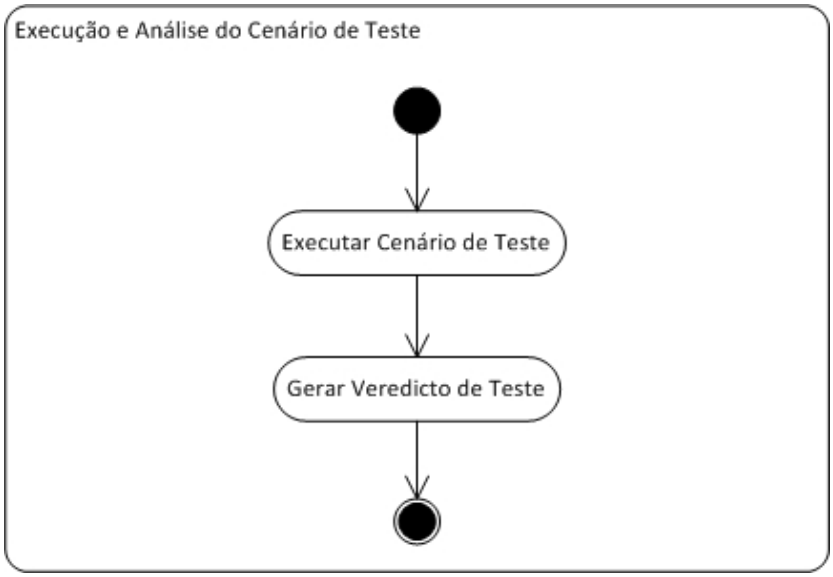

Figura 52 – Fase: Execução e Avaliação do Cenário de Teste

Uma solução de execução de teste deve conceber como é feito o controle da seqüência de execução dos casos de teste, garantindo ou não seu isolamento. Também identifica como o resultado de um objetivo de teste deve ser comparado com o resultado previsto pelo oráculo de teste. Os resultados podem apresentar níveis de abstração diferentes, sendo necessário compatibilizar as diferenças para compará-los. Suas descrições podem ser mais genéricas e independentes de plataforma ou mais especializadas e dependentes de plataforma. Em caso de falha, um laudo deve ser gerado identificando a falta e o local onde foi encontrada, facilitando a correção do problema. Em caso de erro, a arquitetura de teste deve indicar o contexto de sua execução onde foi identificada uma inconsistência. Com base nos laudos, deve ser possível atribuir um veredicto ao cenário de teste. Este deve ser um indicativo de qualidade do sistema.

A seqüência de execução dos casos de teste é feita de acordo com o que foi descrito no modelo de casos de teste. Ela está refletida no programa gerado para o cenário de teste. A princípio, o programa também define quando deve retornar ou não ao seu estado inicial após a execução de um caso de teste. O comportamento padrão é retornar ao estado inicial, pois assim se evita que o resultado de um caso de teste interfira no processamento de outro caso de teste.

Durante a execução de um caso de teste, tanto o objetivo de teste quanto o oráculo de teste são avaliados. Nos casos de teste conceituais, ambos estarão

descritos de forma genérica. Nos casos de teste dependentes de plataforma, o objetivo de teste e o oráculo de teste são concretos e equivalem ao resultado de um processamento nesta plataforma. Em virtude disto, evita-se a discrepância entre os níveis de concretude da representação. Os comandos descritos para o objetivo de teste e para o oráculo de teste nos modelos de casos de teste são convertidos por um adaptador para a sua representação na plataforma.

O resultado de toda comparação é registrado num laudo de teste dependendo da estratégia de teste empregada. Normalmente, se houve um erro ou uma falha, o laudo de teste é gerado. Este laudo indicará a diferença encontrada e marcará o local, ponto de processamento do modelo gramatical, onde ela foi identificada. No caso de sucesso, o laudo conterá apenas o veredicto. Os laudos são armazenados pelo cenário de teste.

Após a execução do cenário de teste, todos os defeitos encontrados são relacionados em laudos de teste do sistema. Estes laudos são verificados por um juiz de teste que atribui um veredicto ao cenário de teste segundo uma estratégia de julgamento. A partir desse ponto, os fatos gerados permitirão ao testador (ou equipe de testes) decidir sobre quando gerar mais testes, modificar os modelos, alterar a estratégia de teste e parar os testes. Os resultados da execução descritos nos laudos de teste podem funcionar como parâmetros para medidas de confiabilidade e qualidade do sistema em teste.

É importante notar que para averiguar a adequação dos testes, podem ser introduzidos códigos mutantes de modo a controlar se os testes estão sensíveis às mudanças. Essa percepção é conseguida pelo acompanhamento dos veredictos gerados. Se um defeito foi introduzido conscientemente e um caso de teste foi projetado corretamente para detectá-lo, então a falha deverá ser manifestada.

4.5.1. Executar Cenário de Teste

O propósito desta atividade é executar o cenário de teste selecionado para o sistema. Esta atividade (Figura 53) é desempenhada por um executor de testes (`Test Runner`). Este executor (Figura 54) aciona o programa de teste correspondente ao cenário de teste (`Test Scenario`), disparando os casos de

teste (`Test Case`) do mesmo. Os resultados dos testes são registrados em laudos de teste (`Test Logs`).

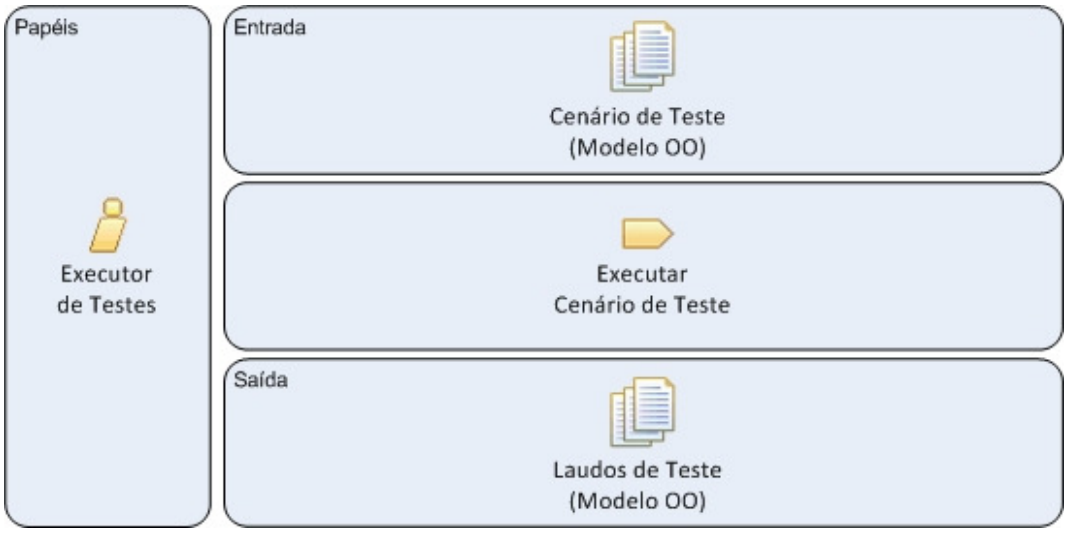

Figura 53 – Atividade: Executar Cenário de Teste

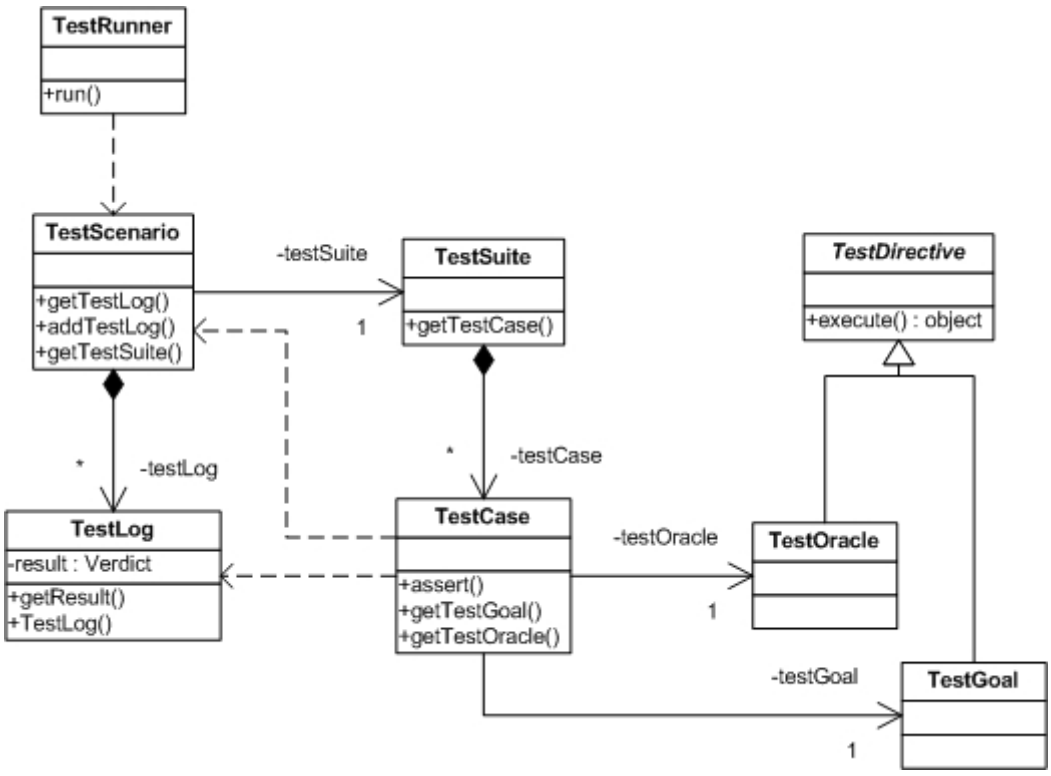

Figura 54 – Arquitetura de Execução de Teste

4.5.1.1. Executar Caso de Teste

A execução do cenário de teste (Figura 55) é composta pela execução individual de cada caso de teste de forma isolada. Isto significa que o resultado de um caso de teste não interfere no estado inicial da execução de outro. O propósito desta tarefa é avaliar uma funcionalidade do sistema de teste para um dado domínio de entrada (dados de teste) por meio da execução de um único caso de teste (`Test Case`). O caso de teste equivale semanticamente a um método do programa de teste que tenta assegurar que um objetivo de teste (`Test Goal`) seja equivalente a um oráculo de teste (`Test Oracle`). O objetivo de teste corresponde a um método com uma seqüência (bloco) de comandos disparados por um executor de testes e com um resultado (valor de retorno). Estes comandos normalmente são as chamadas de métodos do sistema em teste. O oráculo de teste corresponde à outra seqüência de comandos também com um resultado. O caso de teste confronta o resultado do processamento do objetivo de teste e do oráculo de teste por meio de uma assertiva, registra o resultado da comparação (veredicto) no laudo de teste criado por ele e adiciona este laudo ao repositório mantido pelo cenário de teste.

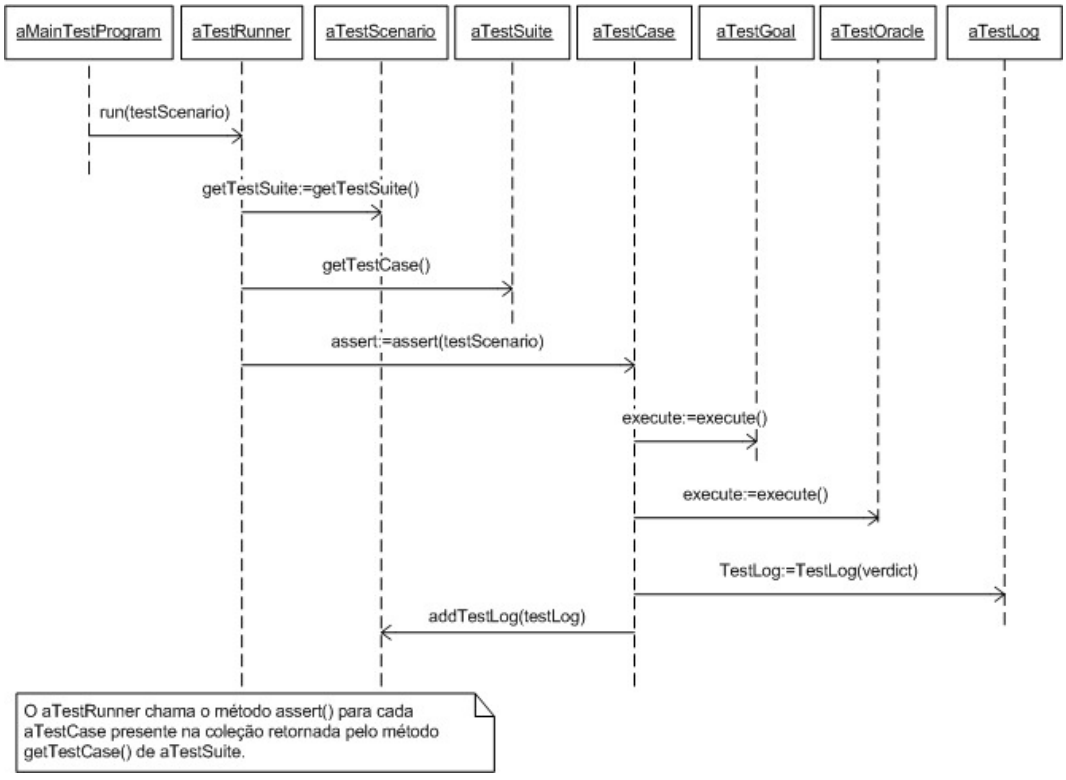

Figura 55 – A Dinâmica de Execução de Teste

4.5.2. Gerar Veredicto de Teste

O propósito desta atividade é gerar uma sentença que identifica se o critério de teste foi ou não satisfeito. Esta atividade (Figura 56) é desempenhada por um juiz de teste (`Test Judge`). Este juiz (Figura 57) avalia os laudos de teste (`Test Logs`) que contém o resultado da execução de cada caso de teste. A sentença do juiz (`Verdict`) é proferida com base numa estratégia (Gamma et al., 1994) de julgamento (`Test Judgement`).

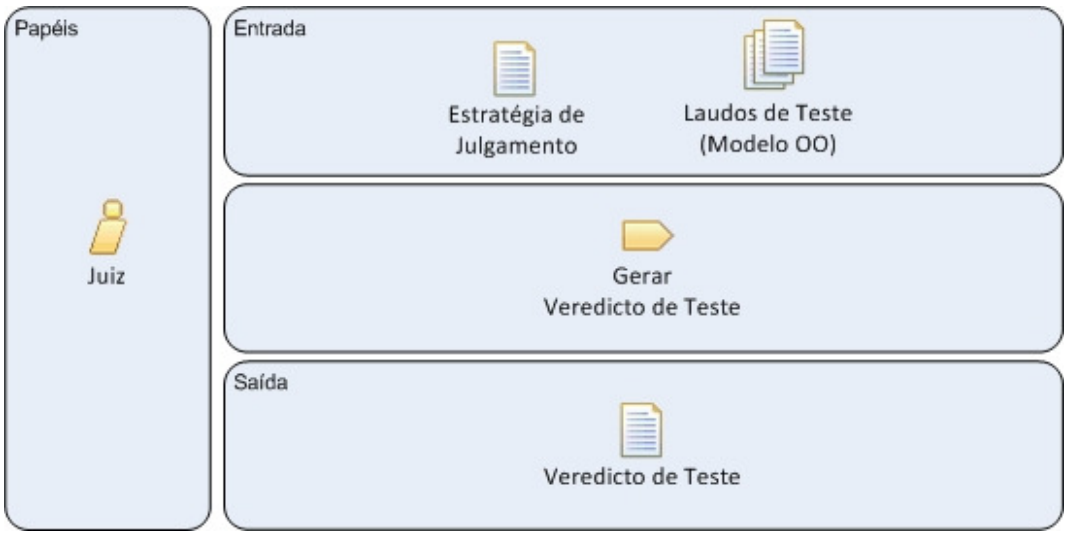

Figura 56 – Atividade: Gerar Veredicto de Teste

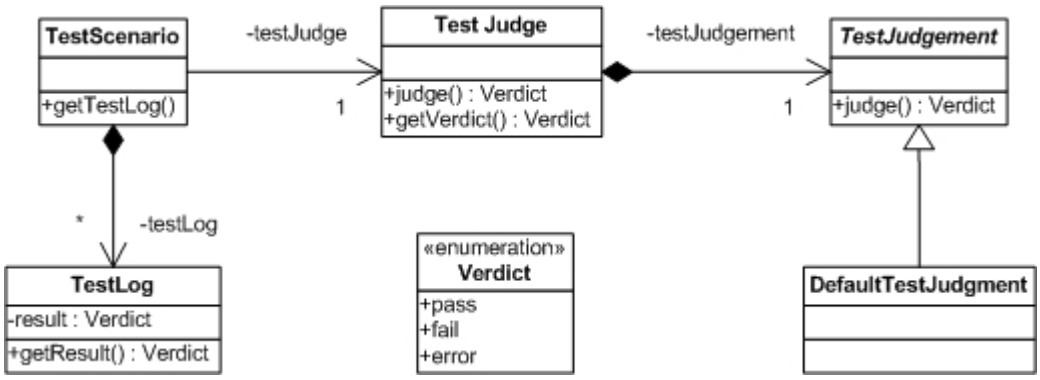

Figura 57 – Arquitetura de Julgamento de Teste

A estratégia de julgamento padrão (`Default Test Judgement`) estabelece um veredicto de falha (`Fail`) quando pelo menos um caso de teste está

inválido, situação em que a resposta obtida não equivale à resposta esperada. Estabelece um veredicto de sucesso (`Pass`) quando todos os casos de uso estão válidos, situação em que respostas obtidas equivalem a respostas esperadas. Finalmente, estabelece um veredicto de erro (`Error`), quando a estrutura de teste apresentou falha na execução (Figura 58).

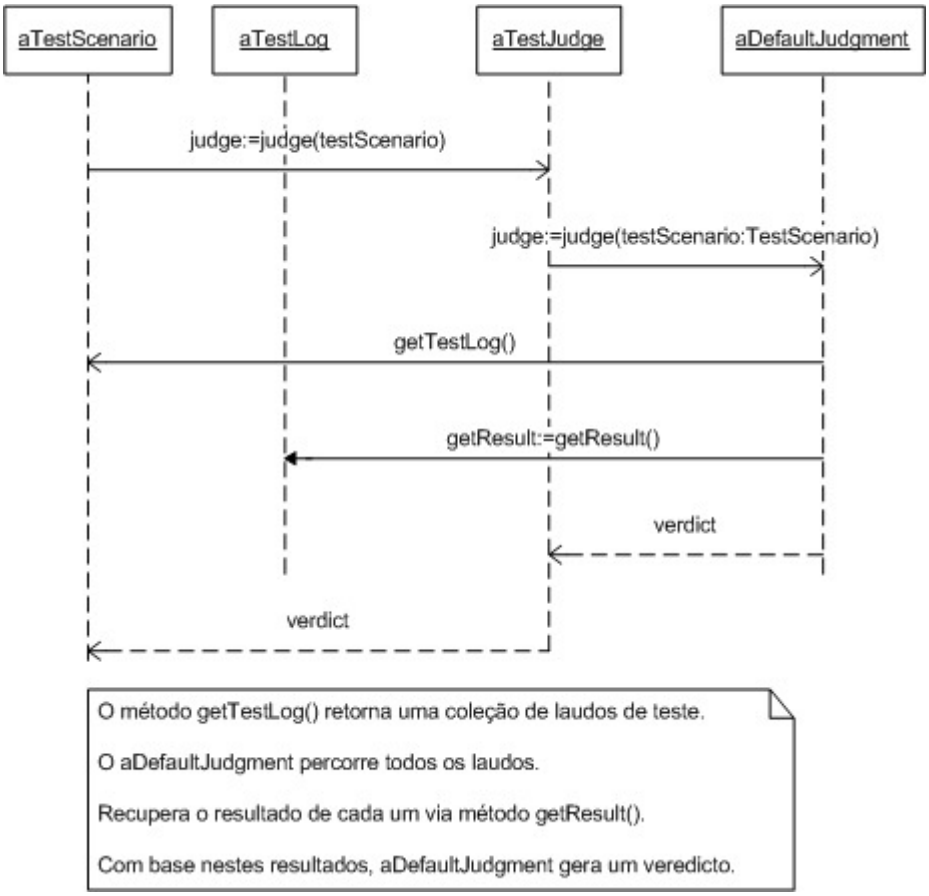

Figura 58 – A Dinâmica de Julgamento de Teste

Com base no veredicto de teste é possível definir as próximas ações. Quando falhas forem observadas, os casos de teste com problemas serão identificados e a estrutura e o comportamento relacionados serão alvo de futuras inspeções. Se todos os casos passaram, isto pode indicar uma possível parada dos testes. Em algumas circunstâncias, predizer o resultado de um teste é tão difícil, inclusive em termos de programação, que o caso de teste precisará ser simplificado. Noutros casos será descartado. Estando o problema relacionado à estrutura de teste, esta deverá ser corrigida antes que qualquer outra ação seja tomada.