Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

UM FRAMEWORK PARA AUXÍLIO NA CRIAÇÃO DE DIAGNOSTICADORES DE FALHAS EM SISTEMAS

Renata França de Pontes

Campina Grande - PB
Março de 2008

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

UM FRAMEWORK PARA AUXÍLIO NA CRIAÇÃO DE DIAGNOSTICADORES DE FALHAS EM SISTEMAS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, Campus I como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciência da Computação (MSc).

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Renata França de Pontes

Orientador: Jacques Philippe Sauvé, Phd

Campina Grande - PB

Março de 2008

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

P814u

2008 Pontes, Renata França de.

Um framework para auxílio na criação de diagnosticadores de falhas em sistemas / Renata França de Pontes. ─ Campina Grande, 2008.

147 f. : il

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Referências.

Orientador: Dr. Jacques Philippe Sauvé.

1. Falhas em Sistemas. 2. Diagnóstico de Falhas. 3. Framework. I. Título.

CDU 004.03

"UM *FRAMEWORK* PARA AUXÍLIO NA CRIAÇÃO DE DIAGNOSTICADORES DE FALHAS EM SISTEMAS"

RENATA FRANÇA DE PONTES

DISSERTAÇÃO APROVADA EM 28.03.2008

PROF. JACQUES PHILIPPE SAUVÉ, Ph.D
Orientador

PROF. JORGE CÉSAR ABRANTES DE FIGUEIREDO, D.Sc
Examinador

PROFa. MARIA DE FÁTIMA QUEIROZ VIEIRA, Ph.D
Examinador

CAMPINA GRANDE – PB

Resumo

Este trabalho tem como objetivo facilitar a criação de diagnosticadores de falhas baseados em modelos de sistemas. Existem diversos diagnosticadores que utilizam os mesmos métodos em sua criação. A solução sugerida é a implementação de um *framework* que utilize a estrutura em comum existente nesses tipos de diagnosticadores e que possa reaproveitar os métodos de diagnóstico de falhas utilizados pelos mesmos. O trabalho mostra que este reaproveitamento traz mais facilidade e rapidez na construção desses diagnosticadores.

Abstract

This work aims to facilitate the creation of Fault diagnosticians based on model system. Several diagnosticians use the same methods in their creation. The suggested solution is the implementation of a framework that uses the existing common structure between these diagnosticians, and that can reuse the fault diagnosis methods. This reuse would make the diagnostician construction easier and quicker.

Agradecimentos

Agradeço a Deus, que me trouxe calma nos momentos necessários. Aos meus pais, por me incentivarem, me apoiarem e serem o exemplo de pessoas nas quais eu me espelho. Às minhas irmãs por terem compreendido a minha falta de tempo e de paciência. Ao meu orientador, Jacques, pela presença constante, reuniões, observações, correções, incentivos e ensinamentos. Aos professores Peter, Joseana e Jorge, pelas dicas, ajudas e amizade ao longo do meu mestrado. A Víctor, pela compreensão nos momentos mais agitados e pela atenção e carinho quando precisei. Aos meus amigos que sempre estiveram presentes nas horas das tristezas e alegrias, obrigada por me agüentarem. Obrigada a todos que contribuíram, de maneira direta ou indireta, para a conclusão dessa fase da minha vida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Equações

1. Introdução

Capítulo 1

Este capítulo contextualiza o diagnóstico de falhas em sistemas. Em seguida, o problema que este trabalho propõe-se a resolver é definido e são apresentados o objetivo, a relevância e a metodologia utilizada para a sua solução.

1.1. Contextualização

Sistema, para este trabalho, é definido como qualquer conjunto de elementos que se relacionam e interajam com o meio gerando um resultado, podendo abranger desde dispositivos elétricos, tais como transformadores de energia, até redes de computadores.

Estes sistemas são vulneráveis a falhas. **Falha** é definida como “um desvio não permitido de pelo menos uma característica própria do sistema” [Ise06]. Portanto, diagnosticar falhas em sistemas consiste em detectar resultados indesejáveis e a causa desses resultados. Para se trabalhar com sistemas, é preciso diagnosticar falhas para que seja possível corrigi-las ou contorna-las.

Dependendo da importância dos resultados gerados e das conseqüências trazidas por eles, diagnosticar falhas será importante para o sistema, em termos de tempo e dinheiro, já que, encontrando o que causou esta falha, a correção da mesma poderá ser feita de forma mais rápida.

Para diagnosticar falhas, primeiro é preciso perceber que algo diferente do normal está acontecendo, este é o momento de detecção da falha. Esta detecção é um alerta para que seja identificado o que está causando essa anormalidade.

Dado que é necessária uma detecção e depois uma posterior identificação da falha, o processo de diagnóstico pode ser divido da seguinte forma [Ise06]:

- **Detecção de Falhas** – fase em que se percebe que uma anormalidade no sistema ocorreu e quando ocorreu.

- **Isolamento de Falhas** – fase em que se delimita a falha e tenta-se isolá-la (isto é, determinar que falha ocorreu), para que seja mais fácil identificá-la.

- **Identificação de falhas** – fase em que certos atributos de uma falha são determinados, incluindo seu tipo, sua amplitude e o que (quem) a causou.

Existem diferentes definições para o termo *diagnóstico de falhas*. Para Korbicz et al. [KKKC04] diagnosticar falhas inclui as três fases acima. Para Isermann [Ise06] diagnosticar falhas refere-se somente às fases de isolamento e identificação de falhas, existindo, portanto, dois momentos: o de detecção e o de diagnóstico. Como, neste trabalho, será observado todo o processo para diagnosticar falhas, contendo as três etapas citadas acima, preferiu-se utilizar a definição de [KKKC04], facilitando, assim, referências futuras a esse processo.

Cada uma dessas fases é realizada com a utilização de métodos criados para a sua resolução. Por motivos de simplificação, denominaremos métodos para a detecção, o isolamento e a identificação de falhas apenas de “métodos”. A Figura 1 exemplifica a estrutura de diagnóstico de falhas em sistemas.

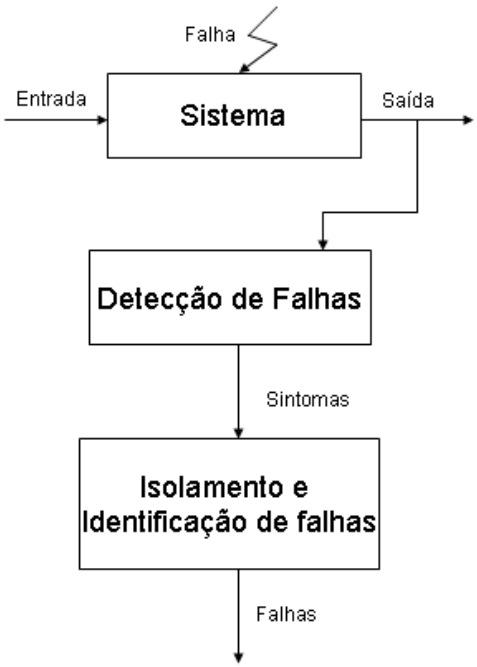

Figura 1: Estrutura para Diagnóstico de Falhas.

Existem diversas formas de se diagnosticar falhas. Algumas delas empregam redundância física para avaliar o sistema. Na redundância física, algumas partes ou todo o sistema é duplicado. Com essa redundância, pode ser feita a comparação entre as saídas desses dois sistemas iguais, podendo assim, ser detectada uma diferença nos valores observados, mostrando que o sistema contém falha. Para que esta detecção seja possível, é necessário garantir que o sistema redundante esteja correto. Caso contrário, só será possível dizer que um dos dois sistemas contém falha, necessitando de um terceiro sistema para a comprovação de qual dos dois valores observados está correto. Esta alternativa pode-se tornar bastante cara com a compra redundante desses sistemas.

Outra forma de tratar esse problema é utilizar redundância analítica, trabalhando com um modelo para a comparação com os valores gerados pelo sistema. As figuras 2 e 3 demonstram a seqüência de passos necessária para o diagnóstico de falhas baseado em modelos, podendo ser: modelo do sinal ou modelo do sistema.

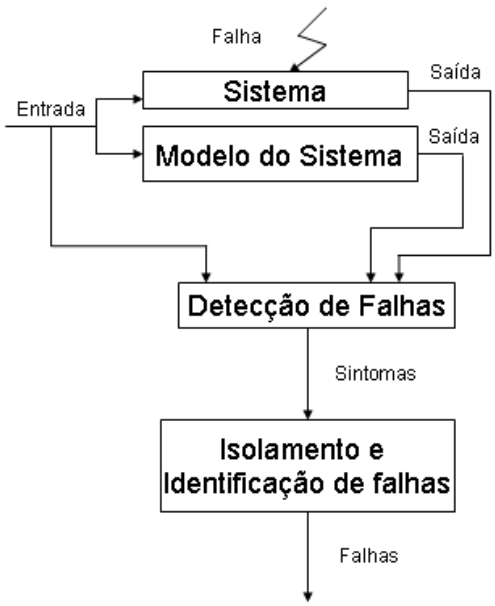

Figura 2: Estrutura para Diagnóstico de Falhas baseado no Modelo do Sistema.

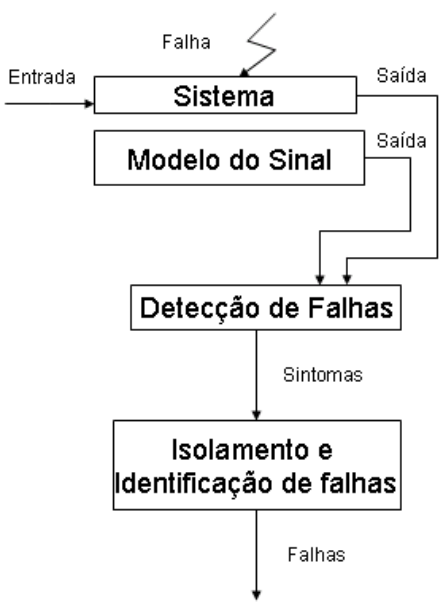

Figura 3: Estrutura para Diagnóstico de Falhas baseado no Modelo do Sinal.

O **modelo do sinal** leva em consideração o comportamento de oscilação ou de tempo cíclico de alguns sistemas, como por exemplo, máquinas de rotação e ruído acústico. Tanto os sinais periódicos como os estocásticos podem ser usados na detecção de falhas, desde que a mudança

nos seus modelos seja causada por falhas nos sistemas. O diagnóstico de falhas baseado no modelo do sinal utiliza somente as saídas deste modelo para avaliação desse sistema.

O **modelo do sistema** consiste em uma representação mais próxima possível do sistema, contendo entradas, saídas, possíveis distúrbios e falhas, como mostrado na Figura 4:

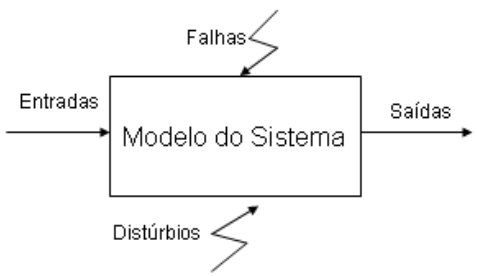

Figura 4: Modelo de um Sistema.

A Figura 4 exemplifica o modelo de um sistema. Essas **entradas** "representam as influências do meio no sistema"[CRB01], são o contato que ele tem com o mundo que o cerca. As **saídas** "representam a influência do sistema no meio" [CRB01]. É o que o sistema retorna para o mundo que o cerca, o resultado do trabalho para o qual ele foi desenvolvido.

Os **distúrbios** "são mudanças desconhecidas no meio, são interpretadas com entradas de ruído" [CRB01]. São, portanto, entradas adicionais de uma mudança de comportamento no meio.

As **falhas**, como já dito anteriormente, são comportamentos anormais do sistema. Elas podem ser representadas como entradas que mudam esses comportamentos. Mais detalhes sobre diagnóstico de falhas em sistemas são apresentados no Apêndice A.

Como demonstrado nas Figuras 2 e 3, após a detecção de falhas, um conjunto de sintomas é gerado. **Sintomas** são resultados da comparação feita

entre os valores medidos do sistema e valores esperados do modelo, de tal forma a verificar que alguma anormalidade está acontecendo. Esse conjunto traz informações importantes para o isolamento e a identificação das falhas. A fase seguinte utiliza as informações fornecidas pelos sintomas.

Como o diagnóstico de falhas baseado no modelo do sinal só observa a saída deste modelo, a entrada do mesmo é ignorada. Com isso, a dependência que existe entre a saída gerada e sua respectiva entrada é desconsiderada. No diagnóstico de falhas baseado no modelo do sistema esta dependência é levada em consideração, podendo assim, observar que a entrada está gerando o comportamento diferente do esperado na saída.

Este trabalho foi direcionado, então, para o diagnóstico baseado no modelo do sistema, que, segundo [KKKC04], é um dos meios mais avançados de diagnóstico de falhas, pelos motivos já descritos.

Diversos métodos para diagnóstico de falhas são apresentados na literatura. Dash e Venkatasubramanian [DV00] citam alguns desses métodos, como, por exemplo, estimativas de parâmetros, observadores de estados, equações de paridade, classificação de Bayes, redes neurais, entre outros. Isermann tem diversos estudos nessa área e em um de seus últimos livros [Ise06], faz uma síntese de vários métodos que já foram estudados. Mais detalhes são fornecidos no Apêndice A.

1.2. Definição do Problema

Na literatura existem diversos trabalhos relacionados ao diagnóstico de falhas utilizando modelo de sistemas, mas os diagnosticadores apresentados nesses trabalhos são implementados para sistemas específicos. A cada novo sistema estudado, um diagnosticador é gerado para ele. Muitos desses utilizam métodos em comum, a partir dos quais se percebe que um reaproveitamento desta implementação poderia ser feito, poupando trabalho e tempo.

Além dos métodos utilizados poderem ser os mesmos, o diagnóstico de falhas baseado em modelos do sistema tem uma estrutura que é utilizada em todos os diagnosticadores. Esta estrutura em comum pode também ser reaproveitada nas diversas soluções.

Na detecção de falhas baseada em modelos de sistema, é feita uma comparação entre o sistema real e o modelo esperado. Esta comparação gera **resíduos**, que segundo Isermann [Ise06] é "um indicador de falha, baseado na diferença entre um valor medido e um esperado". Após a **geração de resíduos,** uma avaliação destes é necessária para que sejam quantificados os sintomas. De acordo com Chiang, Russel e Braatz [CRB01], este esquema "consiste em transformar valores residuais quantitativos em qualitativos identificando os sintomas". Esta é a **avaliação de resíduos**. Na Figura 5, é mostrada a estrutura para diagnóstico de falhas baseada em modelos do sistema.

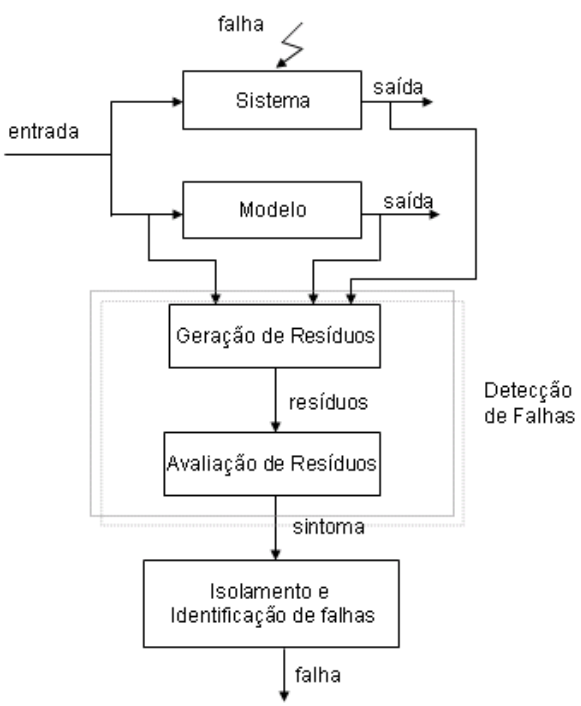

Figura 5: Estrutura de Detecção de Falhas Baseada no Modelo do Sistema.

A contribuição deste trabalho não foi encontrar uma solução de diagnóstico melhor que as existentes, mas sim **facilitar a construção de**

diagnosticadores de falhas baseados em modelos, utilizando o que há em comum nos diversos diagnosticadores, e criar uma forma de reaproveitamento dos métodos de diagnóstico utilizados por diagnosticadores de diferentes tipos de sistemas.

1.3. Objetivos do Trabalho

Este trabalho propõe a criação de um *framework* para auxílio na construção de diagnosticadores de falhas em sistemas baseados em seus modelos. Um *framework* é uma implementação que captura a funcionalidade comum a várias aplicações. É uma implementação incompleta, que precisa acoplar partes de funcionalidades específicas para que possa desempenhar sua função.

A estrutura em comum que existe entre os diagnosticadores de falhas baseados no modelo do sistema pode ser utilizada neste *framework*. Além disso, nele é possível fazer a comunicação entre os métodos de detecção e os métodos de isolamento e identificação das falhas e implementar uma interface de comunicação com uma ferramenta de modelagem, para auxílio na modelagem dos sistemas.

Implementando os métodos de diagnóstico de falhas na forma de componentes, é possível, montando diferentes combinações entre os mesmos, gerar diversos diagnosticadores. Além disso, com o uso de componentes o diagnosticador pode ser montado em tempo de execução.

Um componente, segundo D'Souza e Wills [DW99], é um pacote coerente de implementação de software que: pode ser desenvolvido independentemente e entregue como unidade; tem interfaces explícitas e bem definidas para os serviços que oferece; tem interfaces explícitas e bem definidas para os serviços que requer; e pode ser composto com outros componentes, talvez após a customização de algumas propriedades, mas sem modificar os componentes em si.

Além disso, um componente deve permitir que todo o trabalho (ou quase todo) seja feito pela composição de pedaços existentes. Ou seja, componentes podem ser incluídos em uma aplicação sem a necessidade de código fonte, não sendo preciso haver programação para que o componente possa ser acoplado e utilizado. Esta seria uma construção de aplicações por montagem. Ao explicitar suas interfaces, estas devem ser auto-descritivas, podendo ser descobertas em tempo de execução.

As funcionalidades específicas necessárias para o funcionamento do *framework* foram implementadas na forma de componentes.

1.4.Relevância do Trabalho

Com o auxílio do *framework*, o trabalho de implementação dos métodos só é feito uma vez. Além disso, toda a comunicação feita entre esses componentes já está previamente implementada, havendo um reaproveitamento de código, poupando trabalho e tempo do desenvolvedor.

Por outro lado, uma vez que o método já foi implementado e utilizado em diversos diagnosticadores, sua funcionalidade passa a ser mais confiável. O risco da funcionalidade do método não ter sido bem implementada é reduzido a cada nova utilização bem sucedida desse método.

Do mesmo modo que o *framework* contribui para o reaproveitamento de código e, por conseqüência, de métodos, ele traz uma grande contribuição na maior facilidade de criação de diagnosticadores. Sua montagem e execução podem ser feitas de forma rápida, uma vez que os métodos necessários já foram criados. Isto facilita o trabalho de usuários que necessitam diagnosticar falhas em diferentes sistemas, existindo a possibilidade do surgimento de novos sistemas ao longo do seu trabalho. Em empresas que precisam trabalhar com diagnóstico de falhas em diversos sistemas e nas quais existe uma renovação destes ao longo dos anos, este *framework* traria a facilidade na criação desses novos diagnosticadores necessários para estes novos sistemas.

1.5.Metodologia de Desenvolvimento

A validação de cada diagnosticador implementado foi feita em termos de código gerado e componentes criados e reutilizados. Foram criados diagnosticadores para quatro sistemas diferentes e, para cada novo diagnosticador criado, a quantidade de código gerada foi observada. Foi observada também a necessidade de criação de novos componentes para o funcionamento do diagnosticador.

A metodologia utilizada na realização deste trabalho adotou as seguintes etapas:

Tabela 1: Metodologia Utilizada.

Atividade	Descrição
1	Realização de pesquisa bibliográfica sobre diagnóstico de falhas.
2	Elaboração de um relatório técnico, contendo o Estado da Arte do assunto pesquisado.
3	Elaboração do projeto inicial de *framework.*
4	Implementação do *framework* inicial.
5	Implementação dos métodos e demais componentes necessários para a construção do diagnosticador para o sistema dos três tanques.
6	Criação do primeiro diagnosticador, utilizando os componentes criados.
7	Implementação dos métodos e demais componentes necessários para a construção do diagnosticador para o sistema algébrico.
8	Criação do segundo diagnosticador, utilizando os componentes criados.
9	Implementação dos métodos e demais componentes necessários para a construção do diagnosticador para o sistema da rede da UFCG.
10	Criação do terceiro diagnosticador, utilizando os componentes criados.

11	Simulação de falhas para avaliação da qualidade do terceiro diagnosticador, já que, neste tipo de sistema, o diagnóstico não é feito desta forma.
12	Implementação dos métodos e demais componentes necessários para a construção do diagnosticador para o sistema de redes com limite de banda.
13	Criação do quarto diagnosticador, utilizando os componentes criados.
14	Simulação de falhas para avaliação da qualidade do quarto diagnosticador, pelo mesmo motivo da avaliação do diagnosticador anterior.
15	Validação dos resultados obtidos com o *framework*.

No caso dos sistemas de redes de computadores, a forma de diagnóstico geralmente é feita com monitoração e gerência desta rede, utilizando SNMP e outros mecanismos de coleta de dados da mesma. Com isso, além de ser gerado o diagnosticador de falhas para estes sistemas e serem coletados os resultados do diagnóstico, foi importante mostrar que, de fato, esta nova forma de diagnóstico gerava os resultados esperados.

1.6. Estrutura da Dissertação

A dissertação está organizada em cinco capítulos. O Capítulo 1 contém informações necessárias para a contextualização do assunto a ser desenvolvido, o problema observado, o objetivo do trabalho, a relevância e a metodologia utilizada.

No Capítulo 2 são definidos os requisitos funcionais e não-funcionais da solução proposta. No Capítulo 3 é descrita a solução sugerida nesta dissertação. O Capítulo 4 traz a validação da solução, mostrando os resultados obtidos ao final do trabalho. Finalmente, o Capítulo 5 expõe as conclusões sobre o que foi produzido, relatando dificuldades e contribuições, além de propor sugestões para trabalhos futuros.

2. Requisitos Funcionais e Não-Funcionais

Capítulo 2

Neste capítulo serão descritos os requisitos funcionais e não-funcionais da solução proposta para o problema a ser tratado. Requisitos funcionais expressam uma função a ser implementada em um sistema, já os não-funcionais expressam condições de comportamento e restrições que devem prevalecer [Cys98].

Antes da descrição desses requisitos serão definidos os usuários e sua relação com esta solução. Esta relação será mostrada através de diagramas de contexto e de casos de uso.

2.1. Usuários da Solução

De modo geral, um usuário potencial do projeto é todo e qualquer profissional responsável por identificar e corrigir falhas que surgem em sistemas. Existem três tipos de usuários distintos: o modelador, o criador de diagnosticadores e o usuário dos diagnosticadores.

2.1.1. Modelador

Como o projeto trabalha com técnicas baseadas no modelo do sistema, esse modelo precisa ser criado. O papel do modelador é criar o modelo do sistema que se quer diagnosticar para que possa ser utilizado pelo diagnosticador.

2.1.2. Criador de Diagnosticadores

Em um segundo momento é preciso ser criado o diagnosticador, escolhendo e configurando os métodos de diagnóstico de falhas que devem ser usados para que o sistema de diagnóstico obtenha bons resultados. O papel do

criador de diagnosticadores é fazer essa escolha, montando um diagnosticador eficiente.

2.1.3.Usuário dos Diagnosticadores

Com o modelo e o diagnosticador prontos, é possível obter o diagnóstico de um sistema. O usuário final do projeto poderá escolher o diagnosticador apropriado e gerar os resultados sobre possíveis falhas no sistema avaliado.

Existe a possibilidade de ser necessário criar um quarto usuário que somente observe o resultado final do diagnóstico, não se preocupando com a escolha do modelo e do diagnosticador a ser utilizado. Este utilizaria a escolha feita pelo usuário descrito anteriormente.

2.2.Diagramas de Contexto

Os diagramas de contexto a seguir mostram a relação entre cada um dos usuários e o sistema.

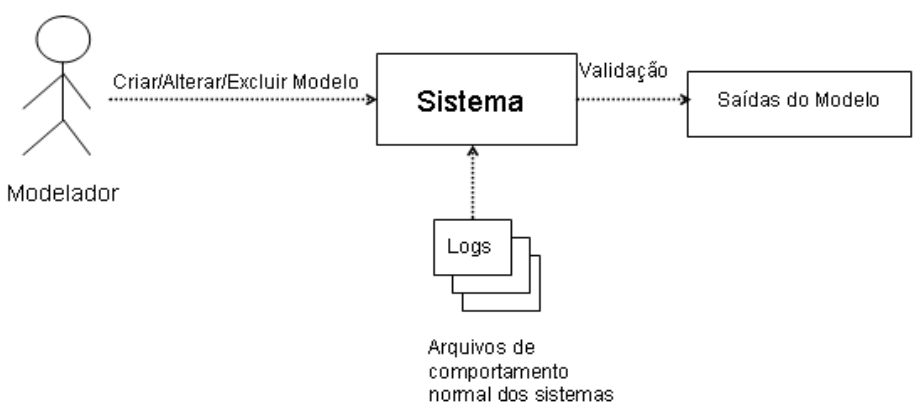

Figura 6: Diagrama de Contexto do Sistema para o Modelador.

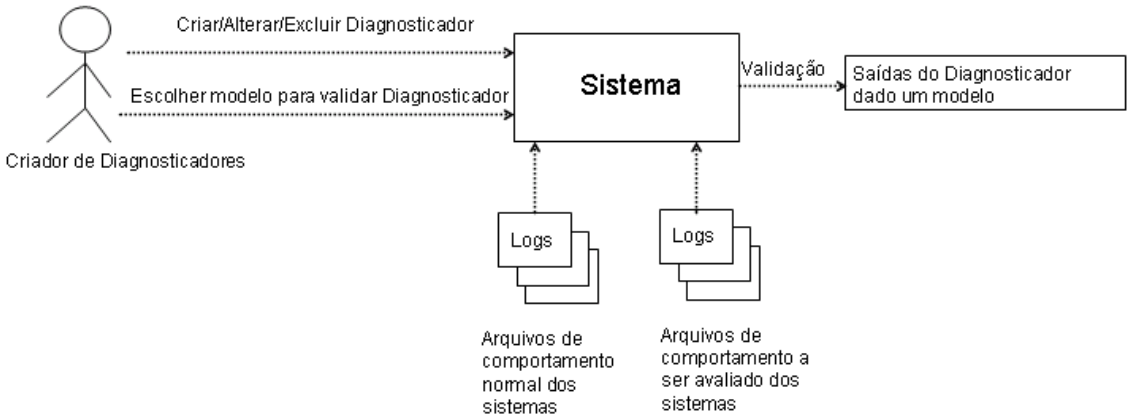

Figura 7: Diagrama de Contexto do Sistema para o Criador de Diagnosticadores.

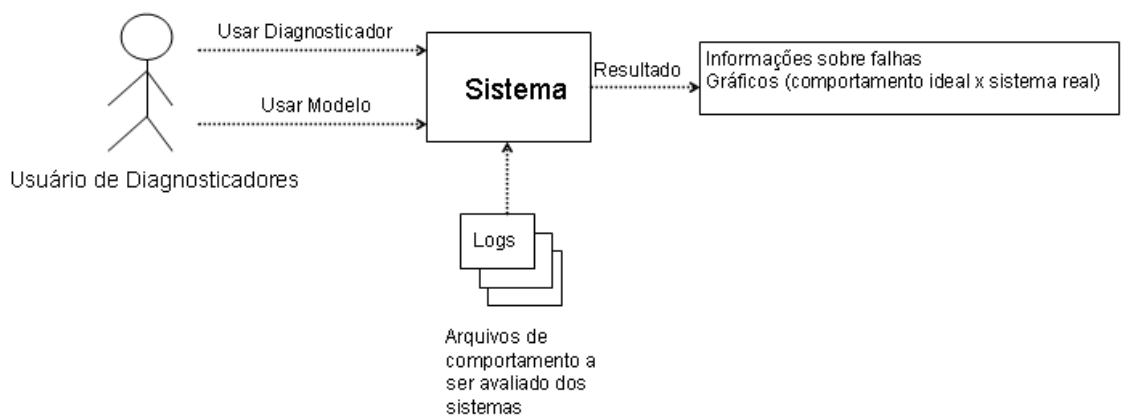

Figura 8: Diagrama de Contexto do Sistema para o Usuário dos Diagnosticadores.

A Figura 6 mostra a relação entre o modelador e o sistema. Este usuário cria, altera e exclui o modelo do sistema, podendo também validá-lo.

A Figura 7 mostra a relação entre o criador de diagnosticadores e o sistema. Esse usuário cria o diagnosticador escolhendo seus métodos de detecção, isolamento e identificação de falhas e passando informações para que esses métodos funcionem. Os diagnosticadores podem ser alterados e excluídos. Este usuário pode conectar um modelo ao diagnosticador e observar seu comportamento para poder validá-lo.

A Figura 8 mostra a relação entre o usuário dos diagnosticadores e o sistema. Esse usuário usa os diagnosticadores já criados para diagnosticar falhas. O diagnosticador irá gerar resultados contendo informações sobre possíveis falhas, podendo gerar gráficos desses resultados.

2.3.Casos de Uso do Produto

Os casos de uso do sistema são descritos nesta seção. Um Caso de Uso especifica uma interação de um usuário com o sistema.

Os seguintes itens são usados para descrever um Caso de Uso:

- **Nome**: Breve descrição do Caso de Uso.
- **Descrição**: Descrição mais detalhada da interação com o sistema.
- **Usuário**: O tipo de usuário que executará o Caso de Uso.
- **Referências**: Identificadores de requisitos funcionais e não funcionais relacionados a este Caso de Uso.

Caso de Uso 1	
Nome	Criação de Modelos
Usuário	Modelador
Descrição	O usuário poderá criar um modelo do sistema a ser avaliado.
Referências	EscolhaModelagem, EspecificaçãoModeloComED, EstimaçãoDeParâmetros, EntradaViaArquivo, ComposiçãoModelo, EspecificaValoresIniciais, AtributosModelo, SalvaModelo

Caso de Uso 1.1	
Nome	Criação de Modelos Especificados Manualmente
Usuário	Modelador
Descrição	O usuário poderá criar um modelo do sistema onde os parâmetros serão fornecidos manualmente.
Referências	EspecificaçãoModeloComED, EspecificaValoresIniciais, AtributosModelo, SalvaModelo

Caso de Uso 1.2	
Nome	Criação de Modelos com Estimativa de Parâmetros
Usuário	Modelador
Descrição	O usuário poderá criar um modelo do sistema onde alguns parâmetros terão que ser estimados.
Referências	EstimaçãoDeParâmetros, AtributosModelo, SalvaModelo

Caso de Uso 1.3	
Nome	Composição de Modelos
Usuário	Modelador
Descrição	O usuário poderá criar novos modelos combinando modelos já existentes.
Referências	ComposiçãoModelo, AtributosModelo, SalvaModelo

Caso de Uso 2	
Nome	Manutenção de Modelos
Usuário	Modelador
Descrição	O usuário poderá acessar um modelo já existente, fazer alterações e excluí-lo.
Referências	ListaModelos, AlteraModelo, ExcluiModelo

Caso de Uso 3	
Nome	Validação de Modelos
Usuário	Modelador
Descrição	O usuário poderá conectar uma fonte de entrada no modelo e comparar suas saídas com saídas esperadas para que possa ser feita a validação do que foi modelado.
Referências	ValidaModelo, ConectaFonte

Caso de Uso 4	
Nome	Criação do Diagnosticador
Usuário	Criador de Diagnosticadores
Descrição	O usuário passará informações necessárias para que os métodos escolhidos possam ser carregados. Esses métodos tratarão da detecção, do isolamento e da identificação de possíveis falhas. A organização dos métodos (nessa seqüência de passos), necessários para se diagnosticar falhas, será feita nesse momento.
Referências	EscolhaMétodos, AtributosDiagnosticador, ConectaMétodos, SalvaDiagnosticador

Caso de Uso 5	
Nome	Manutenção de Diagnosticadores
Usuário	Criador de Diagnosticadores
Descrição	O usuário poderá acessar um diagnosticador já existente, fazer alterações e excluí-lo.
Referências	ListaDiagnosticadores, AlteraDiagnosticador, ExcluiDiagnosticador

Caso de Uso 6	
Nome	Validação de Diagnosticadores
Usuário	Criador de Diagnosticadores, Usuário de Diagnosticadores
Descrição	O usuário poderá testar o diagnosticador criado observando as saídas geradas para um determinado modelo conectado a ele.
Referências	ConectaModeloDiagnosticador, ValidaDiagnosticador, InterfaceGráficaDiagnóstico, TestaDiagnosticador

Caso de Uso 7	
Nome	Geração de Resultados
Usuário	Usuário de Diagnosticadores
Descrição	O usuário emitirá diagnóstico do sistema avaliado.
Referências	ListaPossíveisDiagnosticadores, ListaPossíveisModelos, ListaPossíveisSistemas, GeraçãoDiagnóstico, ExibiçãoFalhas, GeraçãoGráficos

2.4.Requisitos Funcionais

Os requisitos funcionais são aqueles que descrevem o comportamento de um sistema, ou seja, é aquilo que descreve o que tem que ser feito pelo sistema.

Os requisitos funcionais do projeto podem ser divididos em áreas, de acordo com o fluxo básico de informação mostrado na Figura 9. Estes requisitos são identificados com uma abreviação (p.ex. [EscolhaModelagem]) que serve para identificar cada requisito inequivocamente.

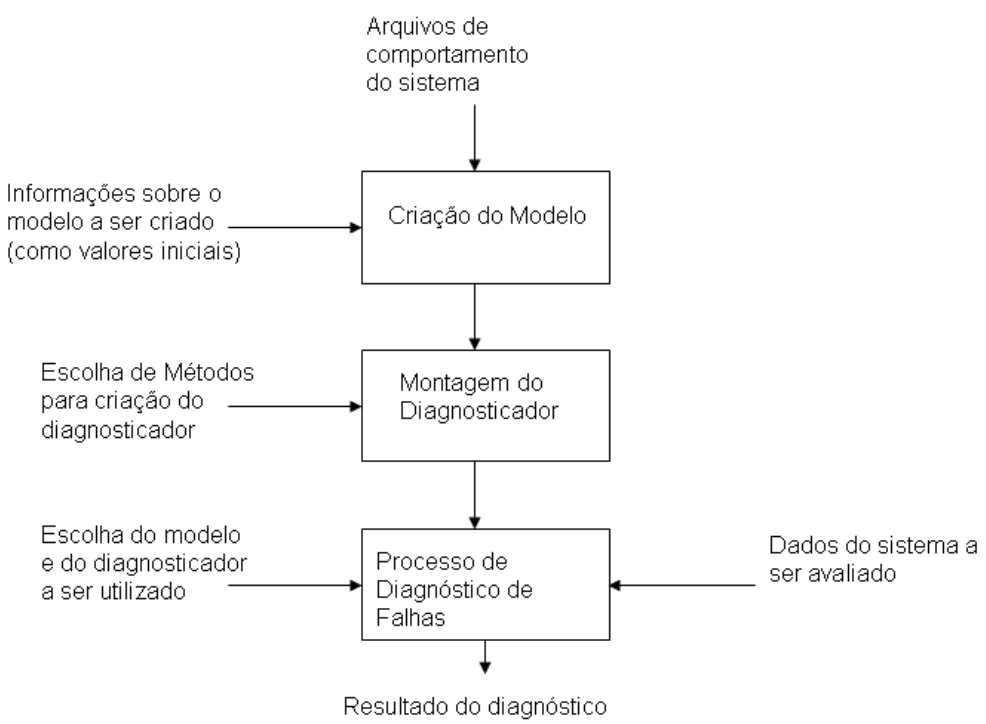

Figura 9: Fluxo básico de informação e áreas de interesse da solução.

Pode-se observar, na Figura 9, que a solução é dividida em três fases: uma primeira, referente à criação do modelo, na qual é preciso receber informações sobre o modelo para que se possa criá-lo; uma segunda de montagem do diagnosticador, na qual são escolhidos os métodos a serem utilizados no diagnóstico; e uma terceira que corresponde ao processo de diagnóstico em si, no qual são escolhidos o modelo e o diagnosticador que irão gerar melhores resultados para um determinado sistema a ser avaliado.

Essas três fases são distintas e algumas são dependentes entre si. O modelo pode ser criado e salvo para uso posterior pelas outras fases. Da mesma forma, o diagnosticador também pode ser criado e salvo para uso posterior. Mas, caso se queira observar os resultados gerados por este diagnosticador, será preciso que o modelo já esteja pronto para que possa ser conectado a ele.

Já o momento de diagnóstico necessita de que as duas fases anteriores já tenham sido feitas. Essa separação permite que usuários diferentes construam os modelos e os diagnosticadores e que estes fiquem disponíveis para uso por qualquer outro usuário.

2.4.1.Requisitos de Criação do Modelo

Os requisitos deste grupo estão relacionados à fase de criação do modelo para que ele possa ser utilizado pelo diagnosticador. Estes requisitos são listados a seguir:

RF 1. [EscolhaModelagem] O sistema deve permitir a escolha de um tipo de modelagem a ser feita, podendo ser: especificação de um modelo através de equações diferenciais, especificação de um modelo utilizando estimativa de parâmetros e a composição de modelos existentes formando um novo modelo.

RF 2. [EspecificaçãoModeloComED] O sistema deve permitir a criação do modelo a partir da especificação do seu comportamento de entrada e saída através de equações diferenciais.

RF 3. [EstimaçãoDeParâmetros] O sistema deve permitir a criação do modelo utilizando técnicas de estimativa de parâmetros como o Least Squares.

RF 4. [EntradaViaArquivo] O sistema deve receber sinais de entrada vindos de arquivos.

RF 5. [ComposiçãoModelo] O sistema deve prover a montagem de um modelo novo a partir da composição de modelos existentes.

RF 6. [EspecificaValoresIniciais] O sistema deve permitir a especificação de valores iniciais de um modelo com equações diferenciais.

RF 7. [AtributosModelo] O modelo deve ter os seguintes atributos: título, criador, data de criação, data da última alteração, descrição do modelo.

RF 8. [SalvaModelo] O sistema deve permitir que o usuário salve o modelo criado.

RF 9. [ListaModelos] O sistema deve permitir que os modelos existentes sejam listados.

RF 10. [AlteraModelo] O sistema deve permitir que o usuário altere o modelo.

RF 11. [ExcluiModelo] O sistema deve permitir que um modelo possa ser excluído.

RF 12. [ValidaModelo] O sistema deve permitir que o usuário valide o modelo criado.

RF 13. [ConectaFonte] O sistema deve permitir que o usuário conecte uma fonte de entrada ao modelo.

2.4.2.Requisitos de Montagem do Diagnosticador

Os requisitos deste grupo estão relacionados à fase de criação do diagnosticador para que ele possa ser utilizado no diagnóstico da falha. Estes requisitos são listados a seguir:

RF 14. [EscolhaMétodos] O sistema deve possibilitar ao usuário escolher os métodos que serão utilizados na construção do diagnosticador.

RF 15. [AtributosDiagnosticador] O diagnosticador deve ter os seguintes atributos: título, criador, data de criação, data da última alteração, descrição do diagnosticador, lista dos métodos utilizados.

RF 16. [ConectaMétodos] O sistema deve montar o diagnosticador conectando os métodos escolhidos pelo usuário.

RF 17. [SalvaDiagnosticador] O sistema deve permitir que o usuário salve o diagnosticador criado.

RF 18. [ListaDiagnosticadores] O sistema deve permitir que os diagnosticadores existentes sejam listados.

RF 19. [AlteraDiagnosticador] O sistema deve permitir que o usuário altere o diagnosticador escolhido.

RF 20. [ExcluiDiagnosticador] O sistema deve permitir que um diagnosticador possa ser excluído.

RF 21. [ConectaModeloDiagnosticador] O sistema deve conectar um modelo criado a um diagnosticador montado.

RF 22. [ValidaDiagnosticador] O sistema deve permitir que o usuário valide o diagnosticador criado.

2.4.3.Requisitos de Processo de Diagnóstico

Os requisitos deste grupo estão relacionados à fase de escolha e uso do diagnosticador apropriado para o diagnóstico de falhas de um sistema a ser avaliado, gerando os resultados desse diagnóstico. Estes requisitos são listados a seguir:

RF 23. [InterfaceGráficaDiagnóstico] O sistema deve oferecer uma interface gráfica para a escolha do diagnosticador a ser utilizado e para a exibição dos resultados do diagnóstico de falhas do sistema em si, ou seja, para todo o processo de diagnóstico.

RF 24. [ListaPossíveisDiagnosticadores] O sistema deve permitir que os possíveis diagnosticadores sejam listados.

RF 25. [ListaPossíveisModelos] O sistema deve permitir que os possíveis modelos dos sistemas sejam listados.

RF 26. [ListaPossíveisSistemas] O sistema deve permitir que os possíveis sistemas a serem avaliados sejam listados.

RF 27. [TestaDiagnosticador] O sistema deve permitir que o usuário conecte o sistema a ser avaliado e seu modelo ao diagnosticador escolhido e observe a saída desse diagnosticador comparando essas saídas com saídas esperadas, testando, portanto, o diagnosticador.

RF 28. [GeraçãoDiagnóstico] O sistema deve permitir ao usuário conectar o modelo e dados de um sistema a ser avaliado a um diagnosticador e coletar os resultados do diagnóstico feito.

RF 29. [ExibiçãoFalhas] O sistema deve exibir informações sobre o resultado do diagnóstico feito, mostrando as possíveis falhas.

RF 30. [GeraçãoGráficos] O sistema deve permitir a geração de gráficos com a saída real e a saída esperada do sistema, mostrando as regiões de falha, para uma melhor visualização dos resultados.

2.5.Requisitos Não-Funcionais

Esta seção descreve os requisitos não-funcionais que devem ser satisfeitos pela solução. Como os requisitos não-funcionais indicam *como* o trabalho da solução deve ser realizado, estes requisitos podem ser relacionados a tópicos como: custo da solução, manutenibilidade, portabilidade, confiabilidade, precisão, integração, documentação, usabilidade e requisitos de interface.

2.5.1.Requisitos de Custo

RNF 1. [Hardware] O sistema deve ser executável em plataforma de hardware de baixo custo, tipo microcomputador, com processador de desempenho médio (tipo Pentium 4, 3 GHz com processador único), com memória de 512 MB.

RNF 2. [SoftwareBásico] O sistema deve ser executável em plataforma de software tipo Linux ou Windows, com software de apoio (Ferramenta de Modelagem: Modelica).

2.5.2.Requisitos de Manutenibilidade

RNF 3. [AdiçãoMétodos] O sistema deve permitir a criação de novos métodos de diagnóstico de falhas e a fácil integração destes ao sistema.

RNF 4. [TestesAutomáticos] No sentido de manter a qualidade do software ao longo de sua evolução, toda a lógica de negócio deve ser testável automaticamente através de baterias de testes de funcionalidade (testes de aceitação). É desejável, mas não obrigatório, que a lógica de apresentação e a lógica de controle (interface entre a entrada/saída e a lógica de negócio) sejam testadas automaticamente.

2.5.3.Requisitos de Portabilidade

RNF 5. [Portabilidade] O sistema deve ser portável para plataformas operacionais Windows e Linux.

2.5.4.Requisitos de Interface e de Usabilidade

RNF 6. [Usabilidade] O sistema deverá ser amigável podendo ser facilmente utilizado após um treinamento básico.

RNF 7. [Erros] O sistema deve reportar qualquer erro com mensagens explicativas. Não se devem expor mensagens internas (do tipo “Null Pointer Exception”) para o usuário.

2.5.5.Requisitos de Confiabilidade

RNF 8. [Integridade] O sistema deve ter um mínimo de controle na verificação da integridade dos seus dados internos de configuração.

2.5.6.Requisitos de Precisão

RNF 9. [Precisão] Dados numéricos devem manter a precisão natural associada ao tipo de informação sendo tratada.

2.5.7.Requisitos de Documentação

RNF 10. [Documentação] O sistema deve dispor de documentação de desenvolvimento (JavaDoc).

2.5.8.Requisitos de Integração

RNF 11. [FerramentaModelagem] Por força dos requisitos funcionais, o sistema deverá se integrar com ferramentas de modelagem.

Este trabalho faz parte de um projeto maior, o Smart Analysis, projeto em parceria com a CHESF para diagnóstico de falhas em equipamentos elétricos. Alguns requisitos são necessários para este projeto, mas ficaram fora do escopo desse trabalho, já que o intuito desta pesquisa é mostrar a eficácia do *Framework* sugerido na solução do problema de diagnóstico de falhas, e não um *software* completo para uso da CHESF. No entanto, esses requisitos estão incluídos no projeto maior de produção deste *software*. Tais requisitos são: propor uma interface gráfica para criação de modelos e diagnosticadores; gerar relatórios HTML ou PDF contendo dados sobre o sistema avaliado e o resultado desta avaliação; prover informações de auxílio (*help*) on-line para uso do sistema e dispor de mídia de instalação (CDROM) que permita a instalação completa do sistema.

3. Solução Proposta para a Criação de Diagnosticadores

Capítulo 3

Neste capítulo, será mostrada a solução proposta por esse trabalho para resolver o problema descrito no capítulo 1 e atendendo aos requisitos do capítulo 2. Como dito anteriormente, existe um esforço na construção de diagnosticadores de falhas. Cada novo trabalho de diagnóstico, relacionado a um novo sistema, exige a implementação de um diagnosticador específico.

A maioria dos trabalhos nessa área utiliza a técnica de diagnóstico de falhas baseado em modelos do sistema. Com essa técnica, métodos para detecção, isolamento e identificação de falhas são implementados, o resultado de um método é passado para outro método e, ao final do processo, o diagnóstico é gerado. Esses métodos muitas vezes se repetem em diversas soluções, não havendo reaproveitamento destes em implementações futuras.

Esta solução propõe a criação de um *framework* para a construção desses diagnosticadores, podendo, assim, reaproveitar os métodos já implementados em vários diagnósticos. Os métodos para a detecção de falhas baseado em modelos têm uma estrutura em comum, que foi utilizada nesse *framework*. Este também faz a comunicação entre os métodos de detecção e os métodos de isolamento e identificação das falhas. Componentes foram projetados para a geração de métodos de diagnóstico.

Com essa estrutura existe a necessidade de geração do modelo do sistema. Essa geração será feita utilizando a ferramenta de modelagem Modelica [Mod], embora outras ferramentas como Simulink [Sim] e Anylogic [Any] possam ser utilizadas. Para tanto, uma interface de comunicação entre esta ferramenta de modelagem e o *framework* foi necessária.

3.1. Arquitetura da Solução

Avaliando o problema a ser solucionado, notou-se que algumas decisões, referentes às características do problema e às possíveis soluções a serem tomadas, precisavam ser feitas para a definição da arquitetura.

3.1.1. Criar um Framework

A idéia de criar um *framework* surgiu assim que o problema se caracterizou, já que a idéia do *framework* é justamente utilizar o que há em comum em diversas implementações, como já foi dito acima. Dessa forma, foi possível criar uma única vez o que existe em comum nos diversos diagnosticadores e deixar essa estrutura pronta para ser reutilizada.

3.1.2. Utilizar Componentização

Com a criação de um *framework* é necessário criar componentes que implementem as funcionalidades restantes, que são específicas de cada diagnosticador, necessárias ao funcionamento do *framework*. A idéia de componentes também surgiu para que essas funcionalidades pudessem ser acopladas a solução em tempo de execução. Com isso, os métodos implementados na forma de componentes podem ser adicionados à solução posteriormente, sem a necessidade de algum tipo de codificação para o acoplamento. Esta característica faz com que outros desenvolvedores possam utilizar o *framework*, acrescentando novos métodos sempre que necessário.

3.1.3. Definir Interfaces Padrões para Possíveis Saídas

Ao utilizar componentes, era importante definir um padrão que estes deveriam seguir para que pudessem ser acoplados corretamente ao *framework* sem a necessidade de codificação extra. Para isso foram criadas interfaces as quais os componentes deveriam obedecer. Como a comunicação entre os métodos ocorre com a saída de um componente sendo utilizada como entrada

de outro, essas interfaces foram definidas de acordo com os tipos de saídas possíveis a serem geradas pelos componentes.

Existem três tipos diferentes de saídas, referentes a três fases diferentes do diagnóstico. Em um primeiro momento trabalha-se com saídas numéricas contínuas, resultantes das coletas de dados do sistema a ser avaliado ao longo do tempo e da fase de geração de resíduos onde essas saídas são comparadas com as saídas, também contínuas do modelo.

Em um segundo momento, a avaliação dos resíduos é feita, e uma saída booleana é gerada, identificando se há ou não um problema na saída observada. O terceiro tipo de saída é uma coleção com as possíveis falhas diagnosticadas. Esta saída é gerada pelo Isolador e Identificador de Falhas, que finaliza o processo de diagnóstico.

Dessa forma, independente de que método está sendo utilizado, se as interfaces de saídas forem obedecidas, haverá comunicação entre os diversos componentes criados.

3.1.4.Controlar Passagem de Tempo

O diagnóstico é gerado ao longo do tempo, as saídas do sistema a ser avaliado são observadas continuamente. Desta forma, um problema que inicialmente não existia pode passar a ser diagnosticado após algum tempo de observação do sistema.

Desta forma, todo o processo de diagnóstico tem de ser feito continuamente, para cada ciclo do relógio. Isto acarreta na necessidade de um relógio que controle a passagem de tempo durante esse processo de forma única e síncrona, para que nenhum resultado seja perdido, ou mal gerado, por problema de sincronia entre os métodos.

Existe uma dependência entre os métodos para a geração de cada saída. O método de isolamento e identificação de falhas precisa das saídas do

avaliador de resíduos, que por sua vez precisa das saídas do gerador de resíduos.

Uma forma de se implementar essa passagem de dados ao longo do tempo é fazer com que cada método gere seus resultados e passe adiante suas saídas, fazendo com que o método seguinte possa utilizá-las para gerar seus resultados.

Para isso, as informações deveriam começar a serem geradas do sistema para a geração de resíduos e dele para os demais métodos na seqüência do diagnóstico. O que o corre é que, para a geração de resíduos, é necessária a comparação dos dados do sistema e do modelo, ou seja, seria preciso que o sistema e o modelo enviassem, ao mesmo tempo, seus valores para que o gerador de resíduos pudesse gerar seus resultados. Caso, por algum problema, uma dessas saídas não fosse gerada, ou fosse gerada com um atraso, seria complicado tratar este tipo de comportamento no gerador de resíduos.

Outra forma de controlar essa passagem de dados, evitando o problema de sincronia entre o envio dos dados do modelo e do sistema, é fazer com que o método que necessita gerar um resultado peça as informações que ele necessita. O isolador e identificador de falhas, para identificar as falhas, pediria o resultado ao avaliador de resíduos, que, por sua vez, para gerar os sintomas, pedia os dados ao gerador de resíduos. O gerador de resíduos pediria os dados ao sistema e ao modelo e, ao recebê-los, geraria seu resultado e responderia ao pedido do avaliador de resíduos.

Esta segunda forma foi a utilizada na implementação da passagem do tempo do *framework* criado, por gerar menos problemas em relação à sincronia no envio dos resultados.

3.1.5.Utilizar Ferramenta de Modelagem

Ao trabalhar com diagnóstico de falhas baseado no modelo do sistema, é necessário criar um modelo que represente o sistema a ser avaliado. Como já existem ferramentas para a criação desses modelos, fez parte da solução, criar uma comunicação com uma ferramenta de modelagem.

Existem diversas ferramentas de modelagem, e a ferramenta Modelica foi escolhida por ser grátis e de ter certa facilidade de comunicação com Java, além do mecanismo de geração das saídas ser compatível com a passagem de tempo do *framework*. Com isso, era possível pedir os dados à medida que o tempo passava.

Com outra ferramenta, Anylogic [Any], não era possível obter os dados de saída à medida que o tempo passava, a própria ferramenta gerava o resultado com sua própria passagem de tempo, sendo necessário sincronizar o relógio da ferramenta com o relógio do *framework*.

Utilizando como base a estrutura do diagnóstico de falhas mostrada na figura 5, os requisitos apresentados no capítulo 2 e as decisões mostradas acima, a arquitetura da solução é definida.

A visão geral da arquitetura é apresentada através de diagramas arquiteturais e de comentários associados a estes diagramas. Cada diagrama apresenta elementos arquiteturais (módulos do sistema, subsistemas inteiros) e as interconexões entre os elementos. A apresentação é “top-down” com o diagrama mais geral apresentado primeiro e refinamentos de subsistemas apresentados em seguida. Esta arquitetura é mostrada nos diagramas arquiteturais abaixo.

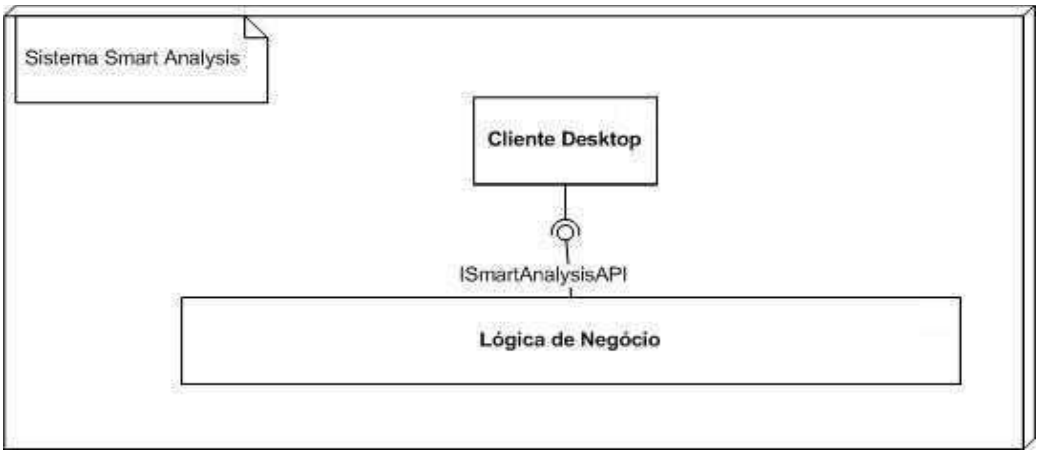

Figura 10: Arquitetura Geral da Solução.

A arquitetura geral da solução está divida em camadas:

- Lógica de negócio: consiste da parte da solução que implementa os serviços principais do sistema. A lógica de negócio é acessada através de uma interface principal, ISmartAnalysisAPI.

- Cliente Desktop: esta camada é a interface de comunicação com o usuário.

Na visão geral do diagrama da Figura 10, é mostrada a ligação entre a lógica de negócio e a interface gráfica com o usuário final. A separação em duas camadas faz com que a lógica possa ser utilizada por tipos diferentes de apresentação para o cliente, tendo uma interface que provê o acesso à lógica. Essa interface atende aos requisitos para a escolha e configuração do modelo e do diagnosticador a ser utilizado, além da exibição dos resultados. Já os requisitos relacionados ao funcionamento do diagnosticador, comunicação com a ferramenta de modelagem, bem como a leitura de arquivos com dados a serem avaliados estão sendo atendidos na lógica de negócio.

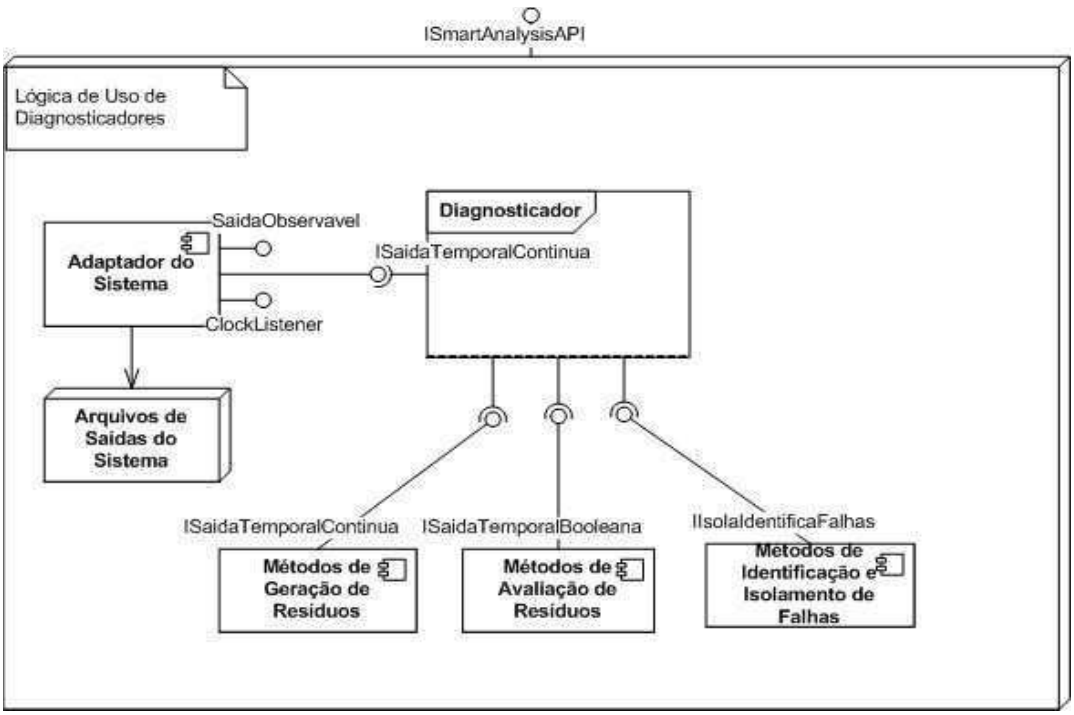

Figura 11: Lógica de Negócio.

Dentro da lógica foi utilizada a estrutura da Figura 11. Os métodos foram implementados de forma componentizada e obedecem a interfaces para que possam ser acoplados ao *framework*. Os métodos de geração de resíduos obedecem à interface ISaidaTemporalContinua, os métodos de avaliação de resíduos obedecem à interface ISaidaTemporalBooleana e os métodos de identificação e isolamento de falhas obedecem à interface IIsolaIdentificaFalhas. Estas interfaces serão descritas mais adiante. Um adaptador foi criado para ler os arquivos com os dados do sistema a ser avaliado. Este adaptador obedece à interface ISaidaTemporalContinua, além de poder observar o tempo (ClockListener) e poder disponibilizar sua saída para que ela seja observada por outros componentes (SaidaObservavel).

A estrutura do *framework* criado para viabilizar esta solução é mostrada na Figura 12.

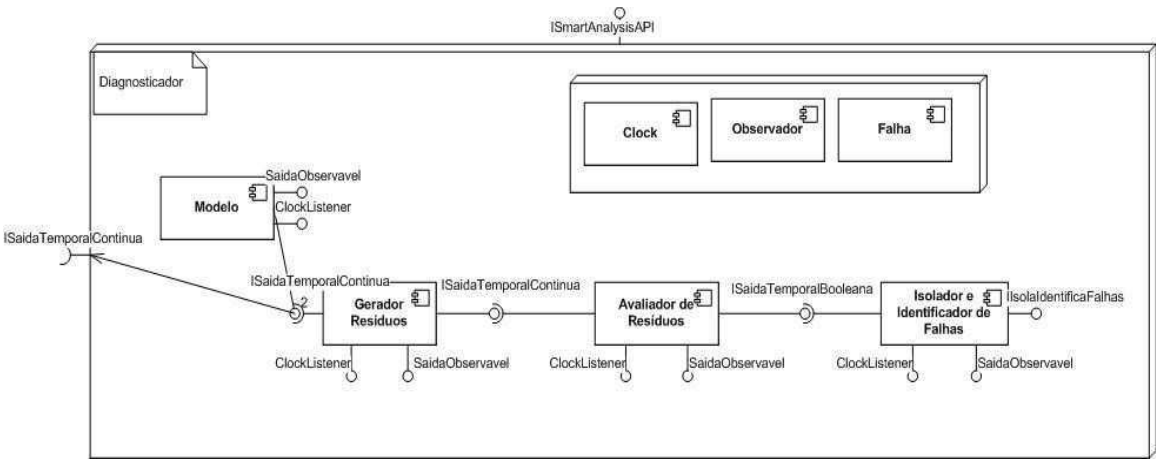

Figura 12: Estrutura do *framework*.

O *framework* recebe as saídas do sistema, vindas do adaptador que recebe esses valores do arquivo que armazena esses dados. Os métodos são escolhidos e acoplados ao *framework,* que faz a comunicação entre eles. Todos os componentes que geram ou acessam o modelo de uma ferramenta, ou que implementam os métodos de diagnóstico de falhas, obedecem à interface *ClockListener*, podendo acompanhar a passagem do tempo, e a *SaidaObservavel*, podendo assim, sua saída ser observada por um outro componente, no caso, um observador. Este componente chamado observador é encarregado de coletar os dados dos componentes que serão listados na interface gráfica.

O diagnóstico é gerado ao longo do tempo. Os dados do sistema e do modelo são coletados, comparados e avaliados a cada instante do tempo. A figura a seguir mostra o funcionamento do *framework*:

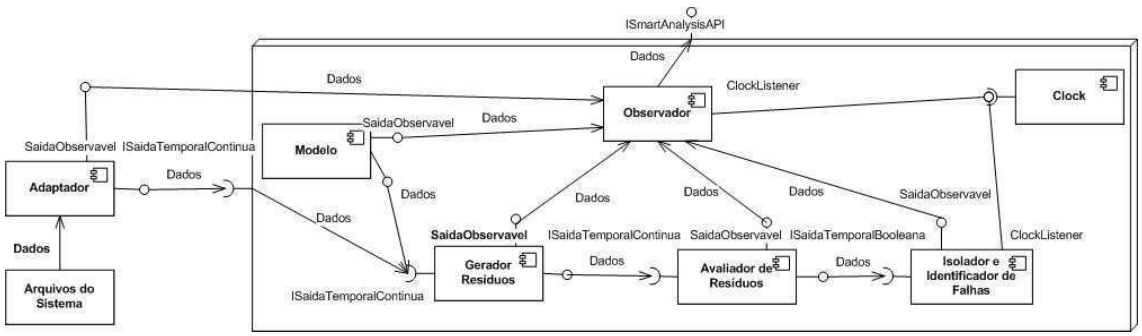

Figura 13: Funcionamento do *Framework*.

Este funcionamento segue os seguintes passos: o relógio avisa ao observador a passagem do tempo; a cada passagem do tempo, para que o resultado possa ser gerado, dizendo se há falhas ou não, o observador precisa do resultado do isolador e identificador de falhas; este, por sua vez, precisa do resultado do avaliador de resíduos (os sintomas) para gerar seus resultados; ao ser pedido o resultado do avaliador de resíduos, este precisa dos resíduos gerados pelo gerador de resíduos para poder calcular esses sintomas; o gerador de resíduos, por sua vez, precisa dos valores do sistema e do modelo para que possam ser gerados os resíduos.

Dessa forma, o *framework* funciona “puxando” a informação do fim para o início do diagnosticador à medida que ela se torna necessária. O componente Observador utiliza um padrão de desenvolvimento chamado *Observer* [GHJV95] para guardar informações de cada dado gerado, disponibilizando-os para que os mesmos possam ser mostrados em gráficos ou telas de resultados.

A descrição dos componentes citados acima é feita a seguir:

- Adaptador: lê os arquivos que contêm os dados do sistema a serem avaliados. Serve para ler qualquer arquivo que siga uma formatação padrão, sendo assim, gera uma saída contínua no tempo utilizando um arquivo de dados.

- Modelo: Pode ser um modelo definido dentro do próprio sistema ou um modelo que tenha sido gerado por uma ferramenta de modelagem e que

uma adaptação foi criada para que seus dados fossem acessados. Gera uma saída contínua no tempo para os valores do modelo do sistema.

- Gerador de Resíduos: recebe as saídas do sistema e do modelo e gera os resíduos implementando algum método de geração de resíduos.

- Avaliador de Resíduos: recebe as saídas do gerador de resíduo e gera os sintomas implementando algum método de avaliação de resíduos.

- Isolador e Identificador de Falhas: recebe os sintomas do avaliador de resíduos e gera uma coleção de falhas detectadas. Caso não seja detectada nenhuma falha, a coleção estará vazia.

- Clock: gera uma saída síncrona, avisando aos componentes que o observam a passagem correta e única do tempo.

- Observador: coleta as saídas dos componentes cujos dados serão listados na interface gráfica. Essa coleta é feita a cada aviso do relógio (clock).

No diagrama de seqüência a seguir é possível observar a dependência dessas entidades ao longo do tempo:

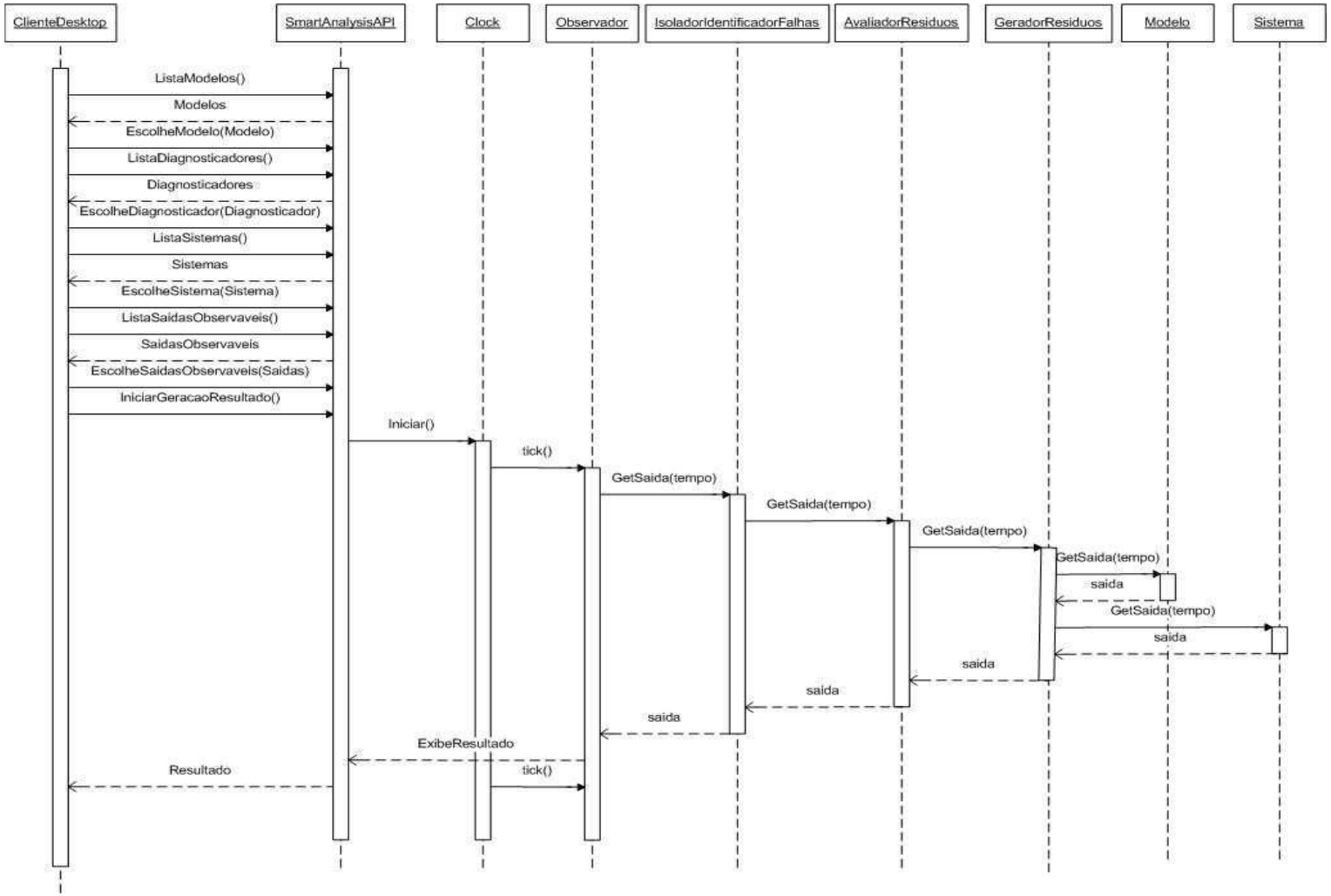

Figura 14: Diagrama de seqüência do Funcionamento do *Framework*

É possível observar que inicialmente são escolhidos o modelo, o sistema e o diagnosticador. Após esta escolha, o processo de diagnóstico é iniciado ativando o relógio para que ele controle a passagem do tempo. A cada aviso de passagem de tempo, o observador, encarregado de coletar os resultados, requisita a saída do isolador e identificador de falhas. Este, por sua vez, requisita as saídas do avaliador de resíduos, que requisita a saída do gerador de resíduos. O gerador de resíduos coleta as saídas do modelo e do sistema, gera seus resultados e retorna para o avaliador.

Cada método, ao receber as entradas necessárias, gera suas saídas, chegando o resultado para aquele momento no tempo ao observador. Todo esse processo é refeito a cada novo aviso de passagem do tempo enviado pelo relógio até que um aviso de parar o processo seja enviado.

As interfaces importantes do projeto, citadas anteriormente, são descritas a seguir:

Interface	**ISaidaTemporalContinua**
Descrição	Permite comunicação com componentes cujas saídas são séries temporais contínuas.
Serviços	• Operação getSaida: através do momento no tempo e da posição da saída desejada, obtêm-se o valor da saída do componente. Um componente pode ter mais de uma saída, a posição refere-se justamente a esse valor. • Operação getQntSaida: retorna a quantidade de saídas existente no componente. • Operação getNome: retorna o nome do componente.
Implementado por	• Gerador Resíduos, Modelo, Adaptador do Sistema

Interface	ClockListener
Descrição	Permite ao componente observar um relógio.
Serviços	• Operação clock: recebe um evento de clock.
Implementado por	• Adaptador do Sistema, Modelo, Gerador de Resíduos, Avaliador de Resíduos, Isolador e Identificador de Falhas.

Interface	SaidaObservavel
Descrição	Permite ao componente ser observado por um objeto.
Serviços	• Operação addOutputListener: adiciona um observador da saída do componente. • Operação removeOutputListener: remove um observador da saída do componente.
Implementado por	• Adaptador do Sistema, Modelo, Gerador de Resíduos, Avaliador de Resíduos, Isolador e Identificador de Falhas.

Interface	ISaidaTemporalBooleana
Descrição	Permite comunicação com componentes cujas saídas são booleanas e dependentes do tempo.
Serviços	• Operação getSaida: através do momento no tempo e da posição da saída desejada, obtêm-se o valor booleano da saída do componente. Um componente pode ter mais de uma saída, a posição refere-se justamente a esse valor. • Operação getQntSaida: retorna a quantidade de saídas existente no componente. • Operação getNome: retorna o nome do componente.
Implementado por	• Avaliador de Resíduos

Interface	IIsolaIdentificaFalhas
Descrição	Permite comunicação com componentes que retornam as falhas encontradas no diagnóstico.
Serviços	• Operação getFalhas: retorna uma coleção de falhas para um valor de tempo específico. • Operação getNome: retorna o nome do componente.
Implementado por	• Isolador e Identificador de Falhas

Interface	ISmartAnalysisAPI
Descrição	Fornece acesso às funcionalidades do sistema através de uma interface.
Serviços	• Operação carregaDiagnosticador: através de um arquivo XML contendo a definição dos componentes e da conexão entre eles, montar o diagnosticador. • Operação simulaDiagnosticador: roda o diagnosticador montado. • Operação getFfalhas: retorna uma coleção de falhas para um determinado valor de tempo. • Operação paraSimulacao: para a simulação do diagnosticador. • Operação alteraVelocidade: Altera a velocidade da simulação.
Implementado por	• Diagnosticador

3.2. Interface da Solução

Uma interface gráfica foi criada para uma melhor visualização dos resultados gerados pela solução. Nesta interface é possível escolher o sistema que será observado, o modelo do sistema e o diagnosticador a ser utilizado. Após essas escolhas, é possível escolher quais valores desenhar nos gráficos.

Esses valores podem ser, desde as saídas do próprio sistema e de seu modelo, até as saídas de qualquer um dos métodos do diagnosticador.

No momento da exibição do diagnóstico, é possível aumentar a velocidade de geração dos resultados para que o gráfico possa ser gerado mais rapidamente. A Figura 15 mostra a interface no momento de escolha do sistema, do modelo e do diagnosticador.

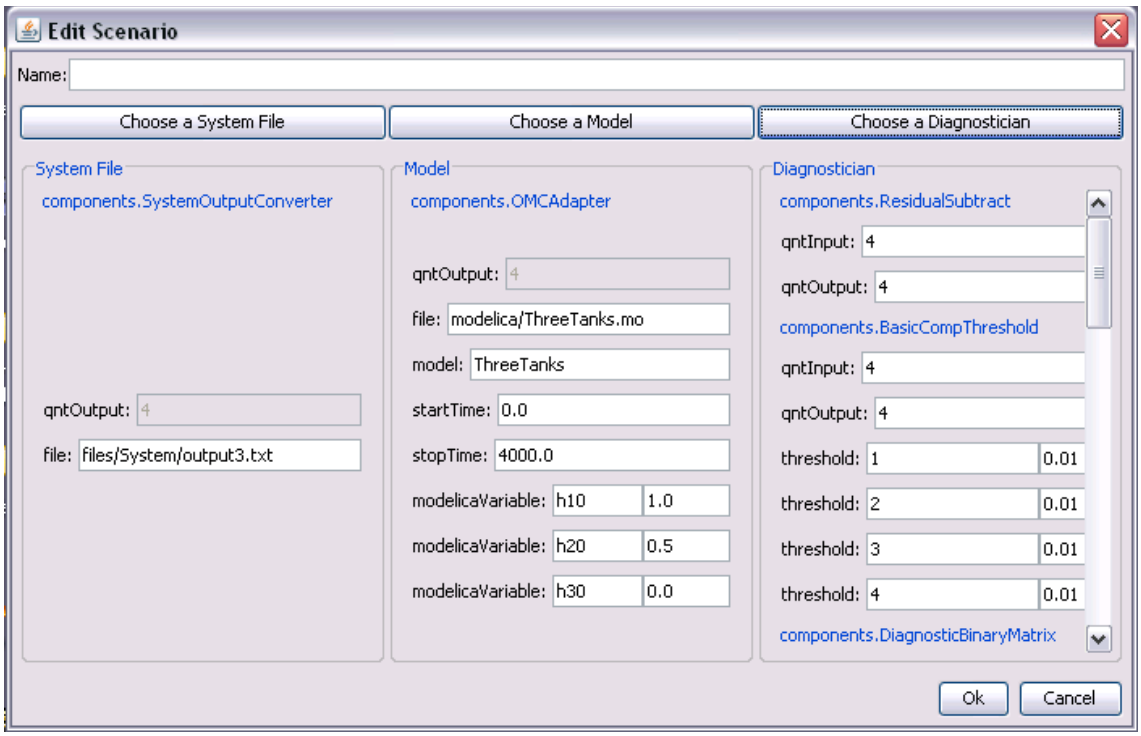

Figura 15: Interface gráfica de escolha do diagnosticador.

Neste primeiro momento, é possível escolher o sistema que será avaliado, seu modelo e o diagnosticador a ser utilizado. Na escolha do sistema indica-se o arquivo que contém as saídas do sistema. Tanto no modelo, quando no diagnosticador é possível alterar valores iniciais de variáveis dos componentes.

Esta combinação de um sistema, um modelo e um diagnosticador é chamada de cenário. Este cenário pode ser salvo para uso posterior. Todos os sistemas, modelos e diagnosticadores estão salvos em XML [XML]. Estes arquivos contêm referências aos componentes utilizados, dados necessários

para inicializar esses componentes e informações de como eles se conectam entre si.

O cenário também é salvo em XML e armazena referências aos arquivos XML do sistema, do modelo e do diagnosticador escolhidos, além de guardar informações dos gráficos que serão gerados para a exibição dos resultados. Exemplos da estrutura de cada XML podem ser encontrados no Apêndice B. A Figura 16 mostra a interface gráfica de escolha dos dados a serem exibidos nos gráficos.

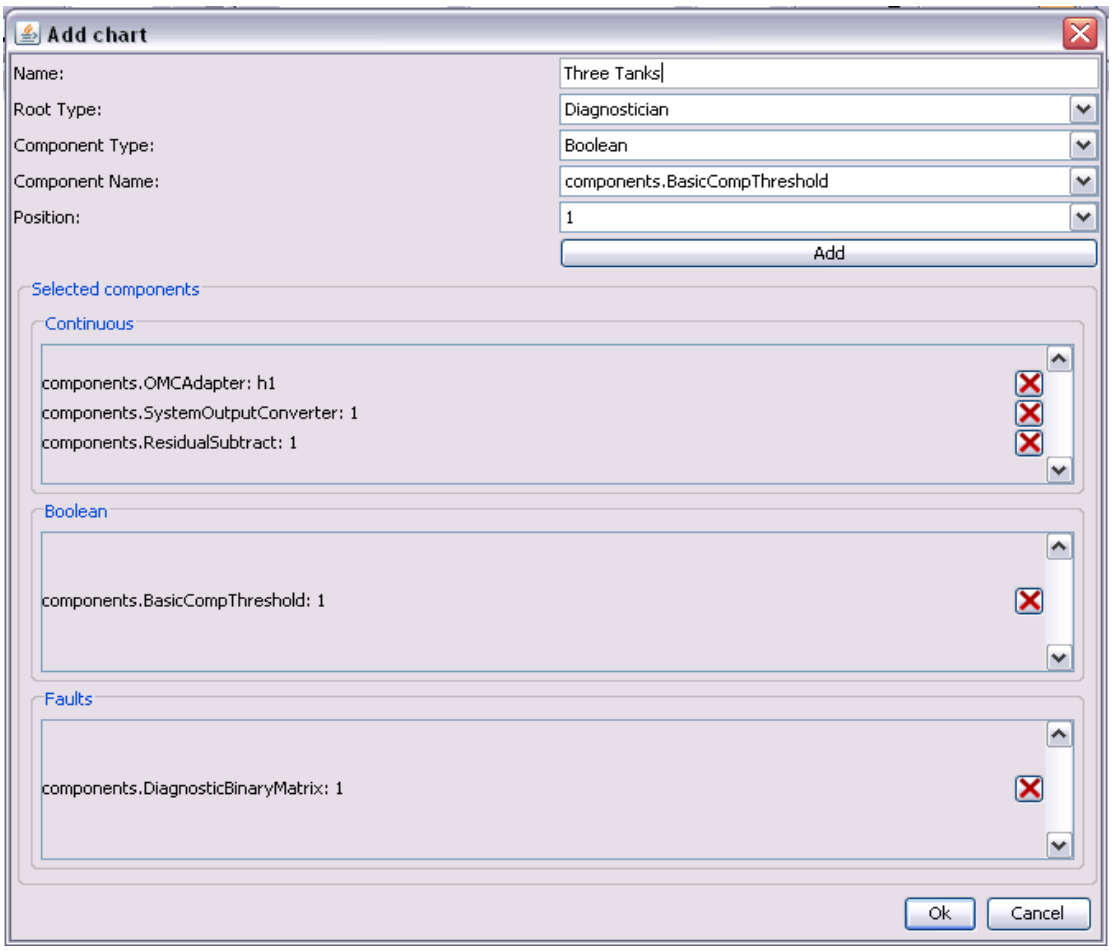

Figura 16: Interface gráfica de escolha de gráficos.

Após a escolha do sistema, do modelo e do diagnosticador a serem utilizados, uma segunda fase é a escolha do que será observado nos gráficos. Pode ser escolhido observar as saídas de qualquer componente, desde o sistema ou o modelo até componentes de geração de resíduos, avaliação de

resíduos e identificação de falhas. Esse último mostra a falha diagnosticada pelo processo de diagnóstico.

Como já foi dito anteriormente, existem três tipos diferentes de saídas: as saídas contínuas no tempo, as saídas booleanas e as falhas. As saídas contínuas no tempo são as geradas pelo sistema, modelo e gerador de resíduos que, ao longo do tempo, geram saídas de valores numéricos. As saídas booleanas são geradas pelo avaliador de resíduos, que recebe os resíduos e gera os sintomas. Os sintomas são valores booleanos, indicando se aquela saída avaliada está fora do padrão ou não. As saídas de falhas são geradas pelo identificador de falhas que retorna uma coleção com as possíveis falhas ocorridas em um determinado momento.

Cada saída pode ser adicionada ou retirada dos gráficos, sendo armazenadas no cenário, caso seja necessário. Após essa escolha, a coleta dos resultados com a observação dos gráficos já pode ser feita. A Figura 17 mostra a interface com a exibição das saídas nos gráficos.

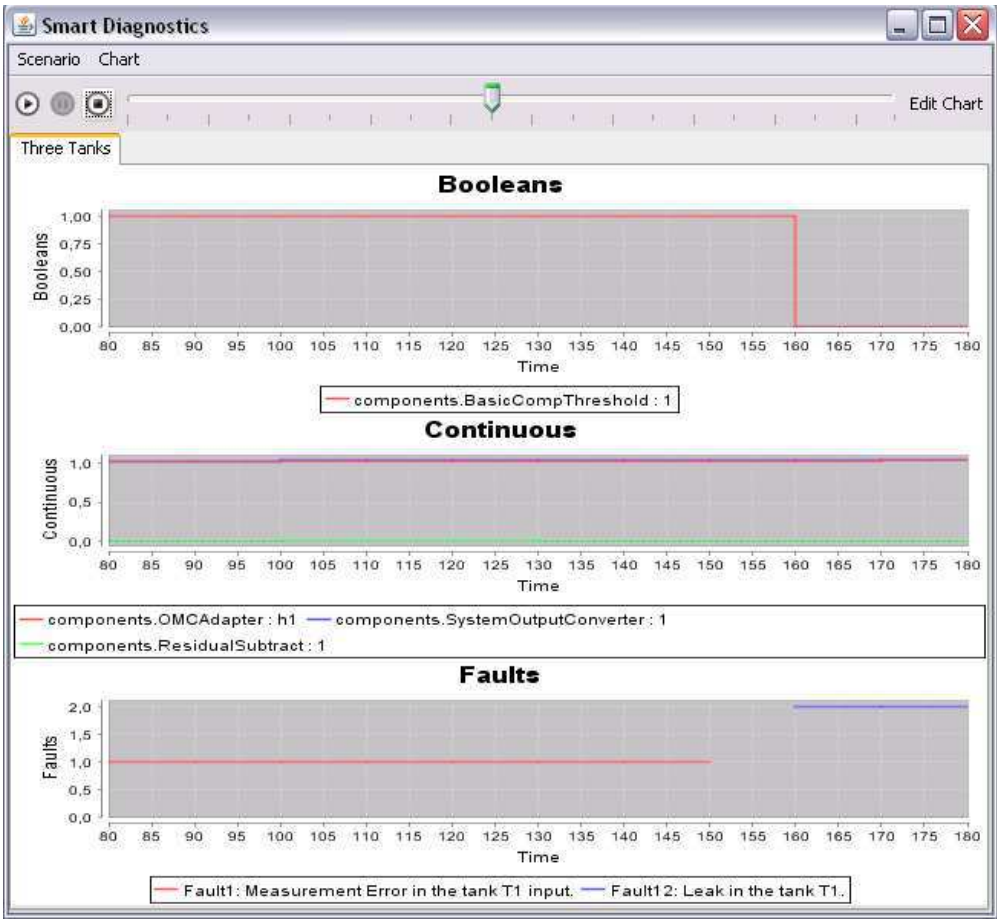

Figura 17: Interface gráfica de exibição dos gráficos.

Esta é a interface que mostra o resultado do diagnóstico de falhas do sistema escolhido. Nela é possível observar todas as saídas dos componentes desejados. A velocidade da geração dos gráficos pode ser controlada e a geração do diagnóstico pode ser pausada e reiniciada. Os gráficos podem ser editados para a adição ou retirada de uma saída.

3.3. Metodologia de Desenvolvimento

O software foi testado utilizando testes de unidade e aceitação. Os testes de unidade foram feitos utilizando JUNIT [Jun], um *framework* de testes em Java. Os testes de aceitação, por sua vez, foram feitos utilizando o Easyaccept [Eas], uma ferramenta para automatização de testes de aceitação para projetos em Java. O projeto foi conduzido através de *test-driven development*, fazendo com que os testes de aceitação fossem criados antes da

implementação de cada funcionalidade. Desta forma, somente o que for definido pelo cliente nos testes de aceitação foi implementado, seguindo rigorosamente o que se desejava como resultado de cada funcionalidade.

A equipe de desenvolvimento foi composta por quatro alunos de graduação, sendo gerenciados por um aluno de mestrado envolvido no projeto. O processo seguido para desenvolvimento do projeto foi o XP [XP], processo de desenvolvimento que prima pela comunicação, simplicidade e *feedback* do que é produzido. Para acompanhamento do projeto, foi utilizada uma ferramenta chamada Xplanner [Xpl], ferramenta para planejamento e acompanhamento de equipes que utilizam o processo de desenvolvimento XP.

Além do Xplanner, outra forma de acompanhar o desenvolvimento do projeto foi gerando *bigcharts*, gráficos com métricas a serem observadas ao longo das semanas. Estes gráficos crescem à medida que o projeto é desenvolvido. As métricas coletadas foram: linhas do código como um todo, linhas de testes de aceitação, linhas de testes de unidades, linhas de arquivos XML, linhas da interface gráfica e linhas do componentes. Um exemplo de bigchart gerado é mostrado na Figura18:

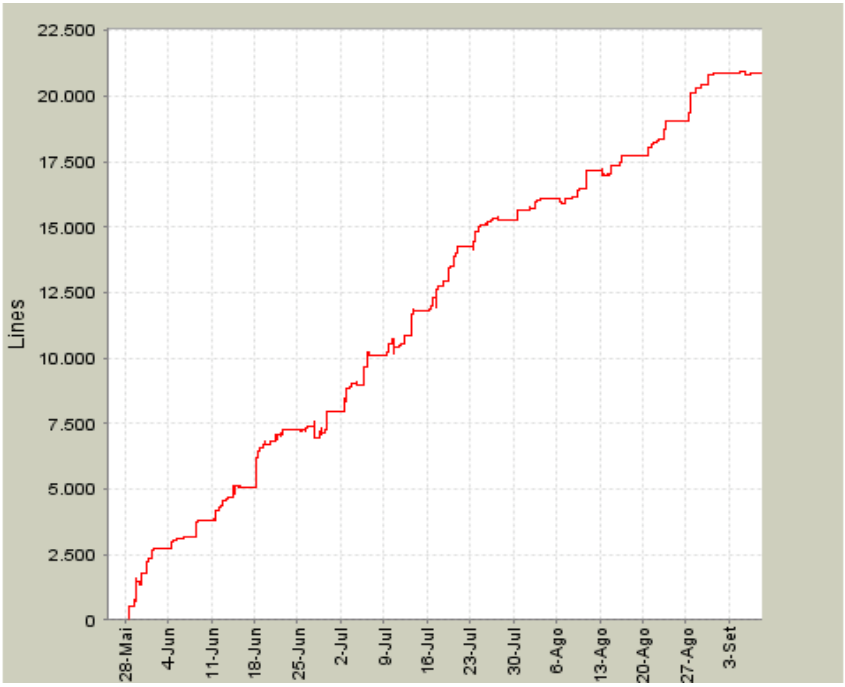

Figura 18: BigChart das linhas de código do projeto.

Com isso foi possível observar o que era produzido a cada semana e a cada modelo novo criado. Observava-se também se os testes cresciam à medida que o código era produzido, provando que o código estava sendo testado.

Após quatro meses de desenvolvimento, o projeto ainda não tinha sido assinado e com isso a equipe se desfez. Os componentes necessários para os dois últimos sistemas foram desenvolvidos apenas pela aluna de mestrado utilizando o framework, as interfaces e os demais componentes já criados.

Foram criadas 186 classes com 20.000 linhas de código, sendo 30 dessas classes de testes de unidade contendo 4.000 linhas de código. Existem 41 arquivos de testes de aceitação com 1950 linhas que utilizam 333 arquivos XML, com 13900 linhas, para representação de sistemas, modelos, diagnosticadores e cenários de testes.

4. Validação da Solução

Capítulo 4

Após a implementação do *framework*, a validação da solução foi feita. Esta validação consistiu na criação de quatro diagnosticadores para sistemas diferentes. Foi realizada uma comparação em termos de código gerado e abrangência de métodos de cada um desses diagnosticadores.

A idéia era que somente novos métodos necessários na construção dos diagnosticadores fossem implementados, havendo com isso um reaproveitamento de código com a utilização do *framework*. Além dos métodos, as implementações necessárias foram aquelas relacionadas com novos modelos, já que diversos métodos de modelagem foram utilizados.

A validação em termos de tempo de desenvolvimento foi descartada já que é uma forma de medição relativa. O tempo gasto para a implementação de novos métodos para a criação de um novo diagnosticador iria depender de quem estivesse fazendo essa implementação, sua familiaridade com a linguagem utilizada e com o *framework*.

4.1. Primeiro Modelo: Sistema dos Três Tanques

O primeiro modelo escolhido foi o sistema dos três tanques, definido com equações diferenciais. Desta forma, um primeiro modelo matemático do tipo dinâmico pode ser criado.

Este é um modelo utilizado em diversas referências como um exemplo de modelo matemático de um sistema. A definição utilizada é a encontrada em [KKC04], mostrada a seguir. No Apêndice C este sistema e os demais são definidos mais detalhadamente.

4.1.1.Definição do Sistema

O sistema consiste de três tanques como mostra a figura abaixo.

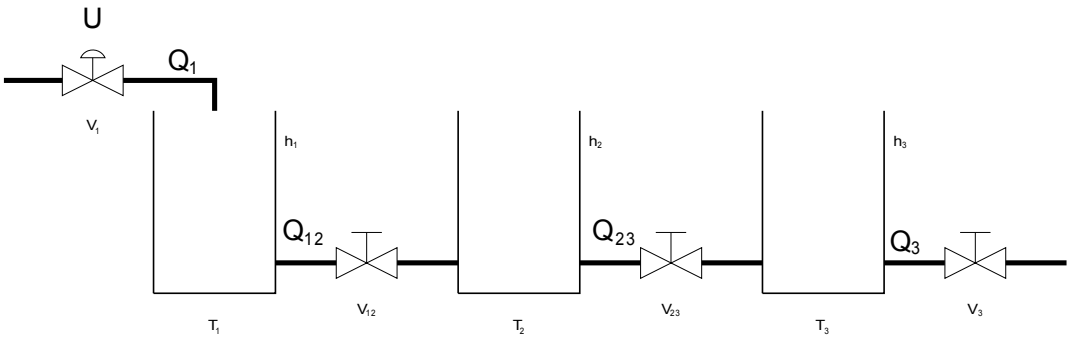

Figura 19: Sistema dos Três Tanques.

Um líquido entra no primeiro tanque à esquerda, denominado T_1 e é controlado pela válvula V_1. Um sinal de controle denominado *U* é aplicado à válvula, controlando o seu percentual de abertura. A vazão de líquido que entra no primeiro tanque é denominada Q_1. Este líquido é armazenado no tanque T_1 e passa para o tanque T_2 através de uma válvula que está sempre aberta, denominada V_{12}. O mesmo esquema segue para os demais tanques até a saída no último tanque T_3. Os sinais medidos neste sistema são: a vazão que entra no tanque T_1 e a altura da coluna de líquido em cada tanque.

4.1.2.Implementação do Modelo

Foi utilizada a ferramenta de modelagem Modelica [Mod] para a implementação do modelo deste sistema. Com isso, foi necessário implementar um componente que fizesse a comunicação entre a ferramenta de modelagem Modelica e o *framework* de diagnóstico de falhas.

Este componente recebeu o nome de OMCAdapter e obedeceu a interface de comunicação com o *framework,* que viabiliza a recepção de dados gerados nesse componente pelo gerador de resíduos.

4.1.3.Implementação do Diagnosticador

O diagnosticador criado para o primeiro modelo utiliza três métodos de diagnóstico de falhas. Como mostrado do capítulo 1, o diagnóstico de falhas baseado no modelo do sistema divide a detecção de falhas em duas fases: geração de resíduos e avaliação de resíduos. O primeiro método foi o Subtrator, utilizado para a geração de resíduos, subtraindo os valores de saída do sistema real e do modelo do sistema. Desta forma, qualquer diferença entre esses valores gera os resíduos deste método.

O segundo método criado foi o de avaliação de resíduos por limite de checagem simples, contendo valores limites para possíveis diferenças entre os valores reais e ideais. Como o sistema e seu modelo podem conter diferenças, por ser difícil modelar com total fidelidade alguns sistemas mais complexos, existem diferenças aceitáveis entre os valores reais e ideais. Desta maneira, o limite determinado neste avaliador de resíduos define quão diferentes podem ser esses resultados e a partir de que momento essa diferença já caracteriza um sintoma do sistema, e não uma diferença de modelagem ou de ruído.

O terceiro método criado foi o da matriz binária de diagnóstico (MBD), método de isolamento e identificação de falhas que relaciona a combinação de sintomas detectados em cada uma das saídas com as possíveis falhas do sistema. Esta matriz possui várias linhas, uma para cada sintoma s_j, e várias colunas, uma para cada falha f_k que pode ser diagnosticada. Se a falha f_k produz o aparecimento do sintoma s_j, então a matriz tem valor M_{jk}= 1, ou 0 caso contrário. O método segue o seguinte comportamento: se o conjunto de entradas s = {s1(t), s2(t),... sn(t)} casar com a k-ésima coluna da MBD, então a saída f_k é 1.

Com esses três métodos, foi possível criar o diagnosticador de falhas para o primeiro sistema e observar seus resultados. A Figura 20 mostra os componentes utilizados para o diagnóstico desse sistema:

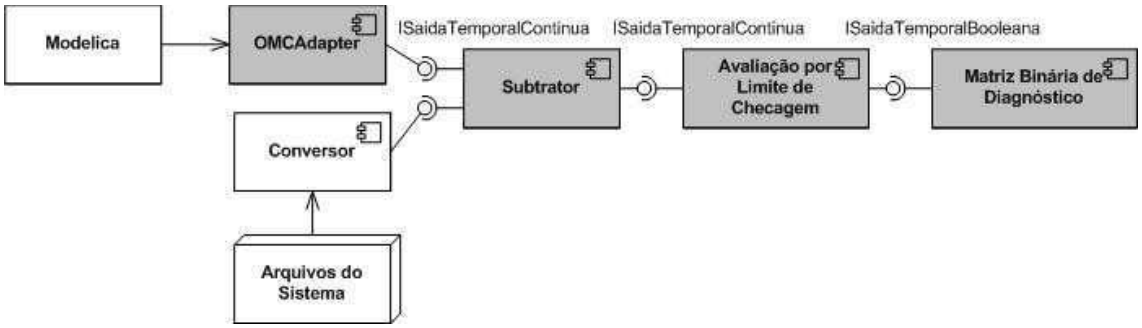

Figura 20: Componentes utilizados para primeiro diagnosticador.

Houve a necessidade de implementar os componentes destacados com a cor cinza para a criação do primeiro diagnosticador.

4.1.4.Validação do Resultado

O modelo desse sistema e os métodos utilizados para seu diagnosticador foram retirados do livro [KKKC04]. Os valores do sistema foram listados em um arquivo e lidos durante a geração do diagnóstico. Estes valores continham variações em relação aos valores ideais; desta forma, duas falhas foram detectadas: erro na medição da vazão de entrada no tanque *T1* e vazamento no tanque *T1*.

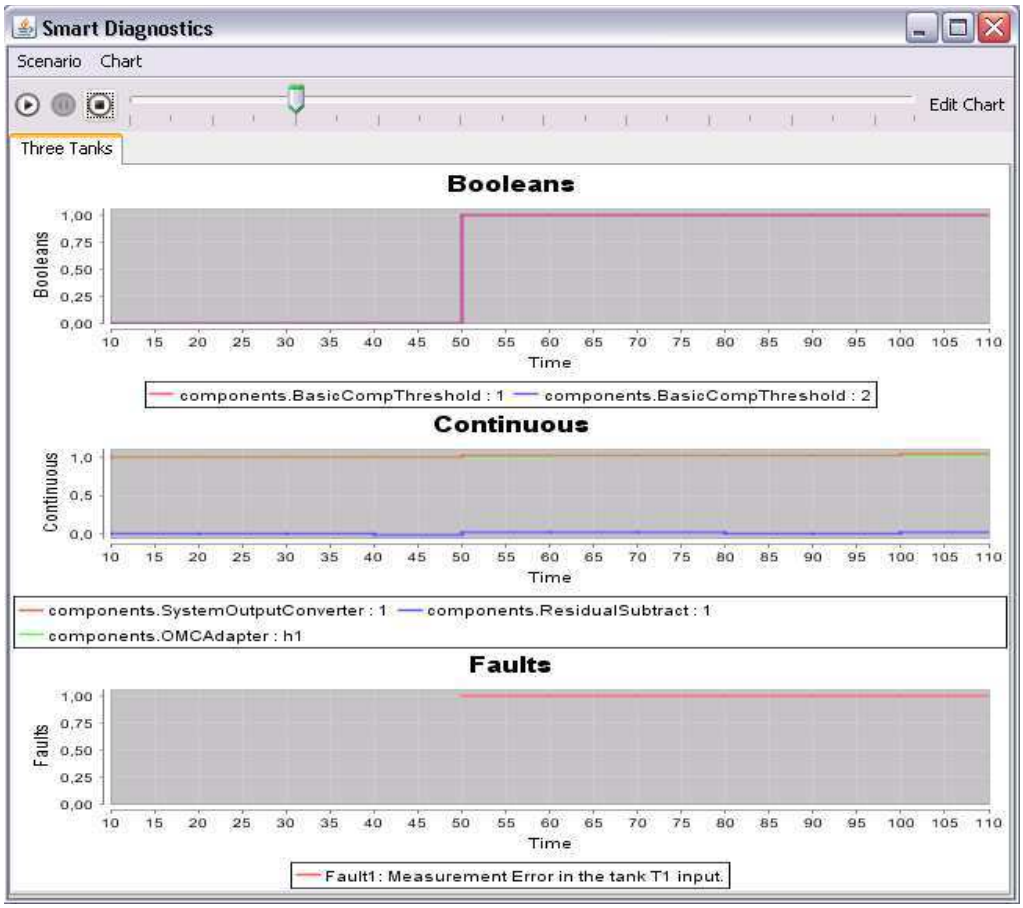

Figura 21: Erro na medição da vazão de entrada do tanque T1.

Figura 22: Vazamento no tanque T1.

As figuras mostram a saída da altura do tanque *T1* no sistema real e no modelo, a saída do gerador de resíduos para essas saídas anteriores, as saídas do avaliador de resíduos para a saída da altura do tanque *T1* e para a saída da vazão de entrada do tanque *T1*.

Na medida em que o sintoma 1 e o sintoma 2 são detectados, a falha 1 é diagnosticada; quando o sintoma 1 deixa de ser detectado, outra falha passa a ser diagnosticada, a falha 12, obedecendo aos valores da matriz binária de diagnóstico do método de isolamento e identificação mostrada na tabela abaixo:

Tabela 2: Matriz Binária de Diagnóstico do Sistema dos Três Tanques.

	f_1	F_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
Q_1	1	0	0	0	1	1	1	1	0	0	0	0	0	0
h_1	1	1	1	0	0	0	0	0	1	0	0	1	0	0
h_2	0	1	1	1	0	0	0	0	1	1	0	0	1	0
h_3	0	0	1	1	0	0	0	0	0	1	1	0	0	1

A Falha 1 diz respeito a um erro na medição da vazão de entrada no tanque T1, com isso, existe uma diferença na saída, tanto da vazão quanto na altura do tanque *T1*. Quando a vazão normaliza, somente a altura do tanque *T1* tem uma diferença em relação ao seu modelo. Gerando, portanto, a Falha 12 que representa um vazamento no tanque *T1*.

4.2.Segundo Modelo: Sistema Algébrico

O segundo modelo escolhido foi um modelo algébrico abstrato, pois representa qualquer sistema que possa ser definido como uma função polinomial. Neste momento não foi escolhido um sistema e, a partir dele, definido uma função que o representasse. Foi escolhida uma função polinomial, que representa um sistema abstrato. Desta forma, quando um sistema real a ser avaliado puder ser representado por uma função polinomial, este tipo de modelo poderá ser utilizado.

4.2.1.Definição do Sistema

Foi implementado um componente representando um modelo discreto, estático, não-linear, genérico, com uma entrada e uma saída (SISO - single input, single output), do tipo polinomial:

$$y(t) = b_0 + b_1 u(t) + b_2 u^2(t) + b_3 u^3(t) + \ldots + b_n u^n(t)$$

Equação 1: Modelo abstrato polinomial.

Onde *u(t)* é a entrada e *y(t)* é a saída. Os parâmetros do modelo (b_o, b_1, b_2 ... b_n) são constantes definidas pelo usuário na criação do modelo. A entrada pode ser conectada a uma fonte apropriada (componente que lê dados de um arquivo). Desta forma, uma lista de valores ao longo do tempo, definida pelo usuário, pode ser passada como entrada para a função polinomial.

4.2.2.Implementação do Modelo

Foi necessário implementar o componente que recebe os dados dos parâmetros e gera a função polinomial desejada e que, ao longo do tempo, recebe dados de entrada e gera os valores correspondentes a esta função na sua saída.

Este componente recebe o nome de DENLSISO (discreto, estático, não-linear, SISO) e obedece à interface de comunicação com o *framework* que viabiliza a recepção de dados deste componente pelo gerador de resíduos.

Para que fosse possível adicionar falhas, outros componentes foram criados. Um somador foi criado para receber os dados da entrada do modelo, receber dados de um injetor de falhas e adicionar aos dados de entrada valores desse injetor, criando entradas com valores diferentes do ideal, gerando, assim, uma diferença na saída.

Um outro componente criado foi o DENLSISO com falhas, que gera a função polinomial desejada, mas recebe entradas de falhas que são somadas aos parâmetros definidos inicialmente ($b_1, b_2 \ldots b_n$), fazendo com que a saída seja diferente da ideal.

4.2.3.Implementação do Diagnosticador

O diagnosticador criado para o segundo modelo aproveitou os métodos de geração e avaliação de resíduos do primeiro diagnosticador. O único método criado foi o de isolamento e identificação de falhas, já que é apenas uma saída a ser avaliada, o único diagnóstico possível é ter uma falha ou não.

Por esse motivo, foi criado um componente mais simples que o isolador e identificador de falhas do modelo anterior, que recebe o sintoma do avaliador de resíduos e diz, caso tenha sintoma, que foi detectada uma falha na saída. A Figura 23 mostra os componentes utilizados para o diagnóstico desse sistema:

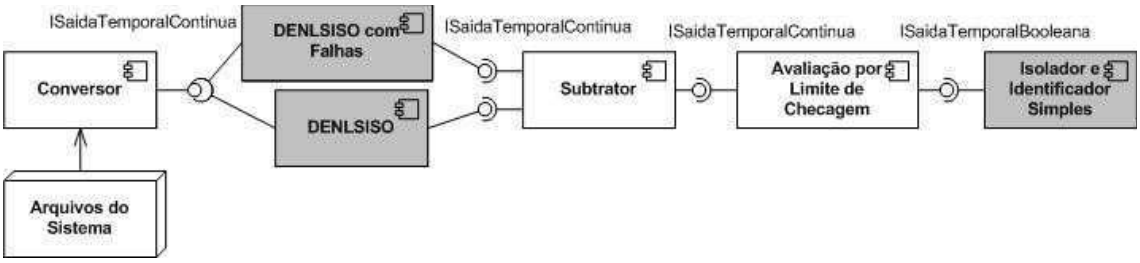

Figura 23: Componentes utilizados para segundo diagnosticador.

Houve a necessidade de implementar os componentes destacados com a cor cinza para a criação do segundo diagnosticador.

4.2.4. Validação do Resultado

Três funções polinomiais foram escolhidas para a validação deste modelo e o sistema de cada uma foi representado pelo modelo com falhas inseridas.

Para a inserção de falhas, três componentes de injeção de falhas foram criados: um componente (rampa) que gera uma saída com aumento gradativo, obedecendo a uma função linear; um componente (degrau) que, em um determinado momento, passa a gerar uma saída diferente de zero; e um componente (intermitente) que, à medida que o diagnóstico se inicia, pode ser ativado ou desativado, gerando ou não saídas diferentes de zero.

A primeira função utilizada como modelo foi:

$$y(t) = 3 + 5u(t)$$

Equação 2: Modelo Algébrico 1.

O componente para injeção de falhas utilizado foi o que tem um aumento gradativo na sua saída. Sua saída foi adicionada ao primeiro parâmetro da função do sistema, no caso, o valor 3 (três) acima. A função linear utilizada para o aumento gradativo no componente de injeção de falhas foi:

$$y(t) = 2 + u(t)$$

Equação 3: Função Linear do Componente Injetor de Falhas .

Os componentes desse sistema são mostrados na Figura 24:

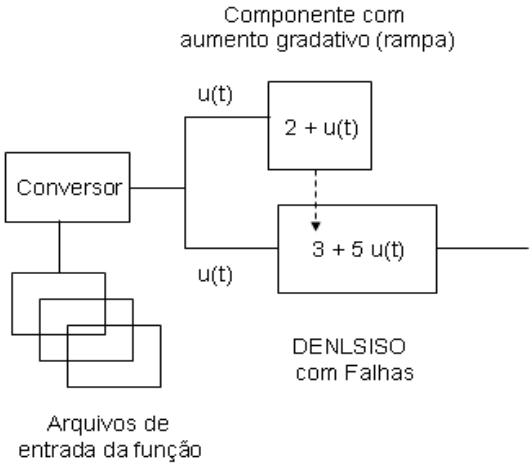

Figura 24: Componentes do modelo algébrico 1.

Em que *u(t)* vem de um arquivo de entrada que lista valores de zero a cem, fazendo com que seja possível observar o comportamento da função ao longo do eixo x. A Figura 25 mostra este diagnóstico sendo gerado.

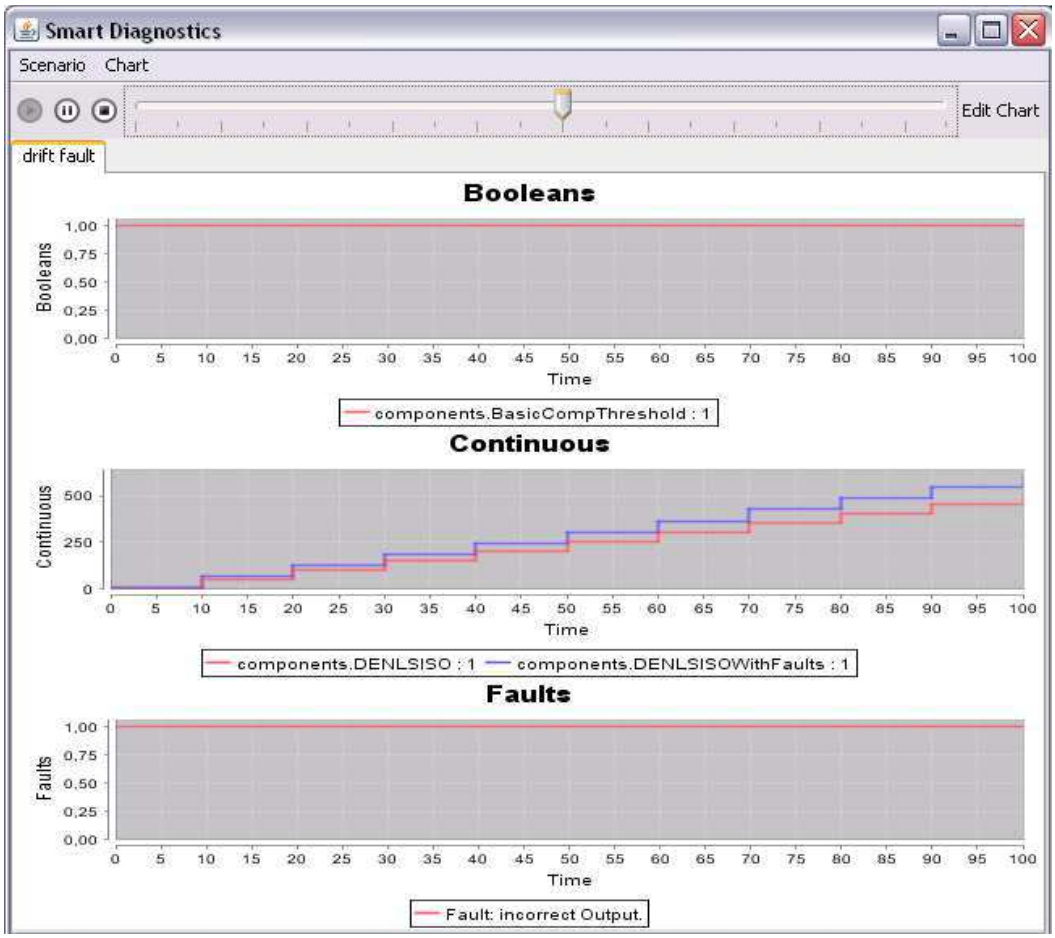

Figura 25: Falha no Modelo Algébrico 1.

No gráfico gerado é possível observar a diferença entre o modelo (DENLSISO) e o sistema com falhas (DENLSISOWithFaults). Sabendo que o comportamento do sistema equivale ao do modelo, definido na Equação 2, adicionado de uma falha no primeiro parâmetro, falha essa definida pela Equação 3, a função que define o comportamento do sistema passou a ser:

$$y(t) = (3 + u(t) + 2) + 5u(t)$$

Equação 4: Equação 3 acrescentada a Equação 2 adicionando a falha.

$$y(t) = 5 + 6u(t)$$

Equação 5: Sistema Algébrico 1.

Tendo a Equação 2 representando o modelo e a Equação 5 representando o sistema, o gráfico da Figura 25 mostra justamente a diferença de resultados entre estas equações. No instante em que o valor de entrada foi *u(t)* = 2, por exemplo, a saída do modelo foi igual a 13 e a saída do sistema foi igual a 17, gerando assim uma falha.

A segunda função utilizada como modelo foi:

$$y(t) = 2 + 3u(t) + u(t)^2$$

Equação 6: Modelo Algébrico 2.

O componente para injeção de falhas utilizado foi o que pode ser ativado ou desativado durante a geração do diagnóstico. Sua saída foi adicionada ao segundo parâmetro da função do sistema, no caso, o valor 3 (três) acima. Os componentes desse sistema são mostrados na Figura 26:

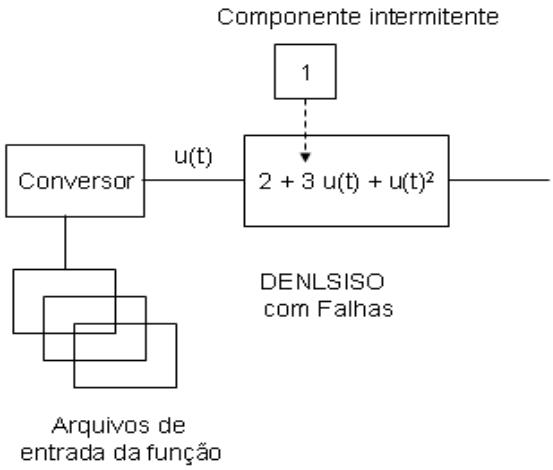

Figura 26: Componentes do modelo algébrico 2.

Para gerar falhas neste sistema foi necessária a inclusão, na interface gráfica, de um mecanismo que permitisse o acesso a esse componente de injeção de falhas durante o diagnóstico. A Figura 27 mostra este diagnóstico sendo gerado.

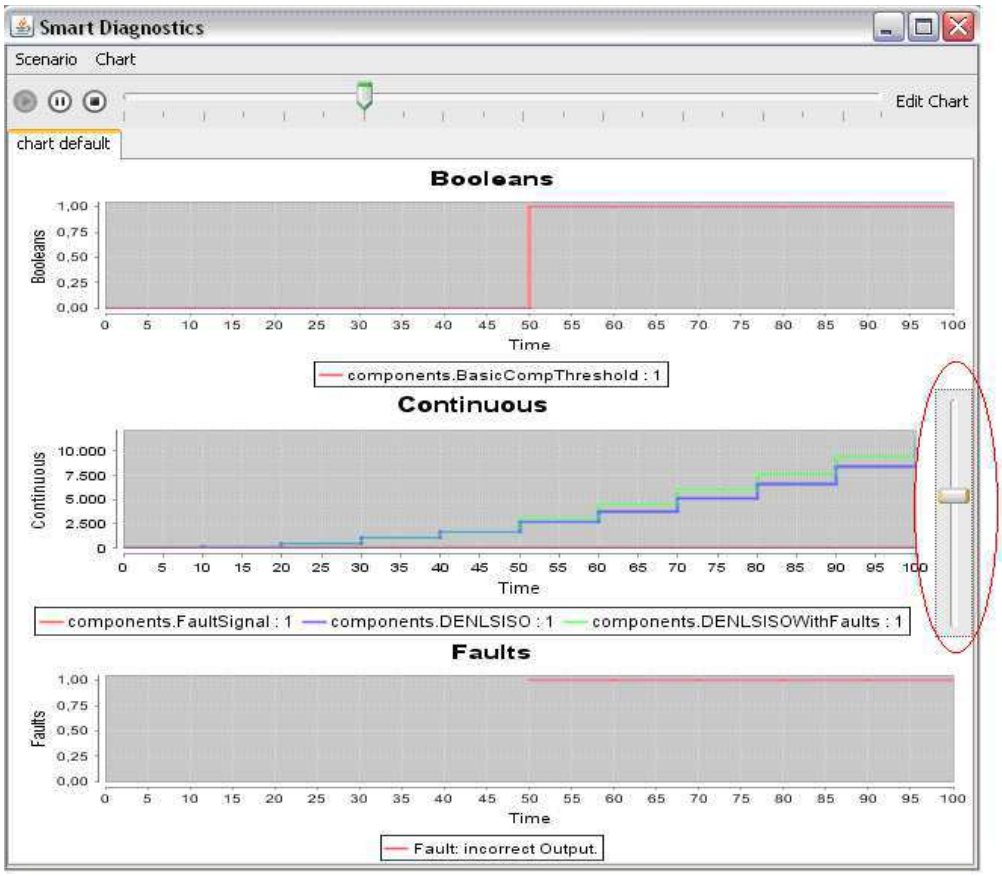

Figura 27: Falha no Modelo Algébrico 2.

É possível observar a barra adicionada na interface, destacada na Figura 27, para a inserção da falha durante o diagnóstico. Observando o gráfico gerado com as saídas do modelo e do sistema, percebe-se que a falha só foi adicionada aos cinqüenta segundos do diagnóstico.

No instante em que o valor de entrada foi u(t) = 2, tanto o modelo como o sistema geraram a mesma saída. No instante em que o valor de entrada passou a ser u(t) = 50, momento em que o componente de falha foi ativado, a saída do modelo foi igual a 2652 e a saída do sistema foi igual a 2702, gerando assim uma falha.

A terceira função utilizada como modelo foi:

$$y(t) = u(t) + u(t)^2 + u(t)^3$$

Equação 7: Modelo Algébrico 3.

O componente para injeção de falhas utilizado foi o que, em um determinado momento, gera uma saída diferente de zero. Os componentes desse sistema são mostrados na Figura 28:

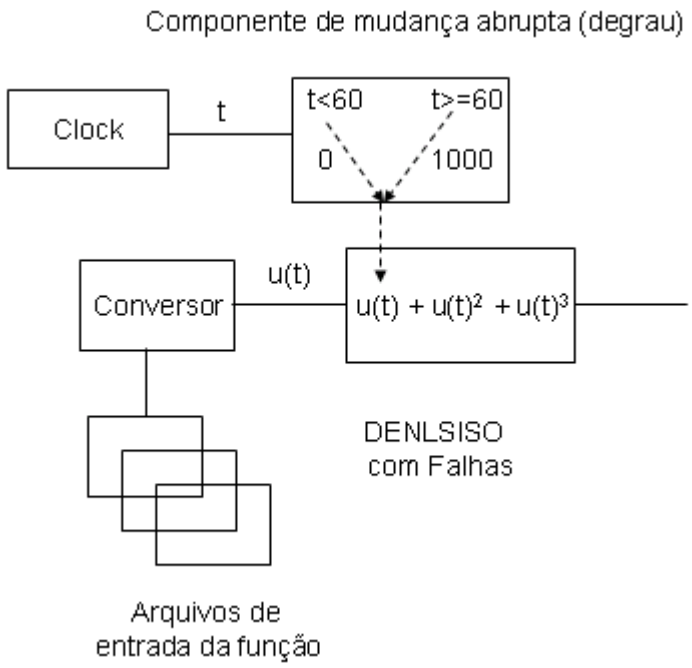

Figura 28: Componentes do modelo algébrico 3.

No caso em questão, quando o processo de diagnóstico completa 60 (sessenta) segundos, este componente passa a injetar um valor igual a 1000 (mil) no primeiro parâmetro da função que representa o modelo. A Figura 29 mostra este diagnóstico sendo gerado.

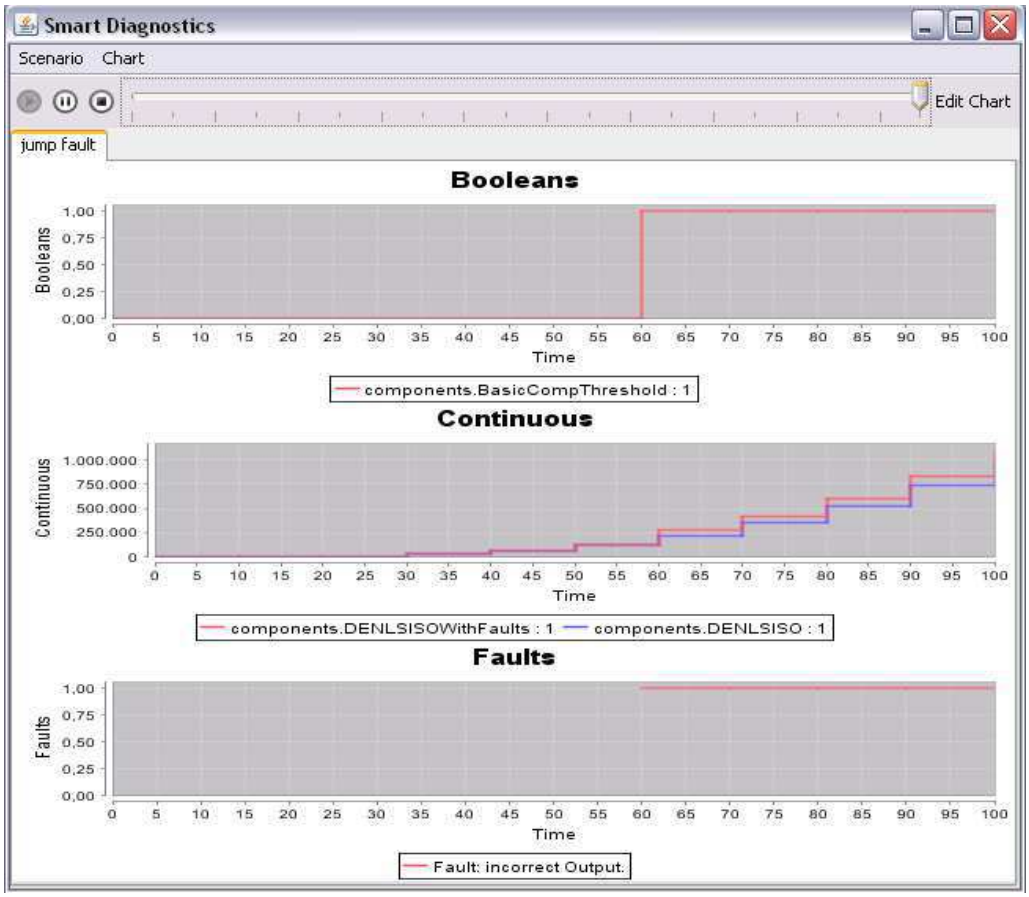

Figura 29: Falha no Modelo Algébrico 3.

Observa-se que, somente a partir dos 60 segundos, a falha é inserida, fazendo com que o gráfico do sistema se diferencie do gráfico do modelo.

No instante em que o valor de entrada foi u(t) = 2, tanto o modelo quanto o sistema geraram a mesma saída. No instante em que o valor de entrada passou a ser u(t) = 60, momento em que o componente de falha começa a gerar a falha, a saída do modelo foi igual a 219660 e a saída do sistema foi igual a 220660, gerando assim uma falha.

4.3.Terceiro Modelo: Estrutura da Rede da UFCG

O terceiro modelo escolhido foi um sistema na área de redes de computadores. Como se pode ter acesso aos dados da rede da Universidade Federal de Campina Grande (UFCG), a idéia foi mostrar que é factível trabalhar com qualquer sistema, desde que seja possível criar seu modelo.

Como este é um sistema no qual não foram definidas as equações que o representassem, foi necessário se trabalhar com identificação de modelos.

4.3.1.Definição do Sistema

O Sistema escolhido foi parte da rede da UFCG. Esta subrede contém três roteadores que definem seu núcleo e os prédios da instituição são conectados a um desses equipamentos. Estes roteadores estão interconectados em anel, para que seja possível prover redundância. A Figura 30 ilustra a rede da UFCG:

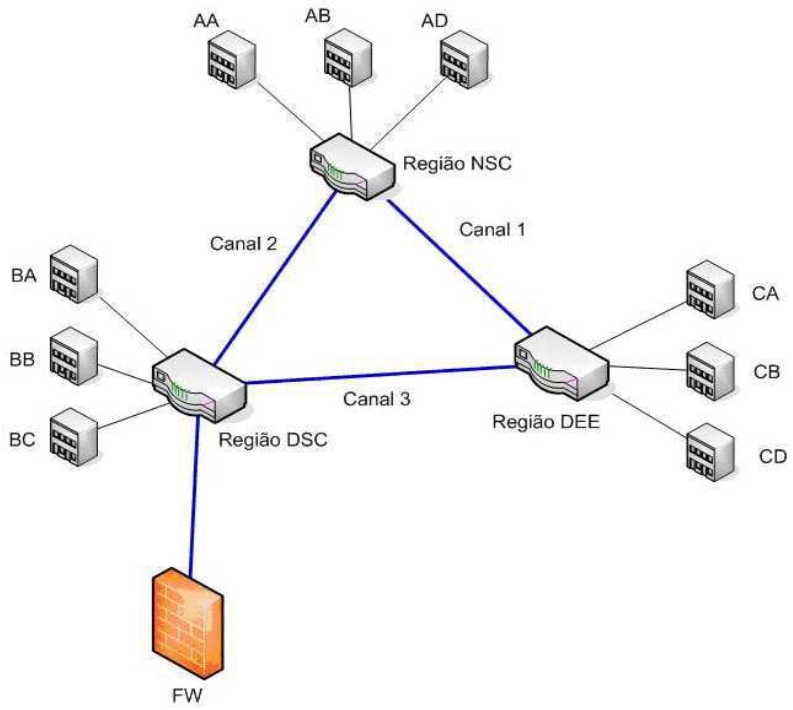

Figura 30: Rede da UFCG.

Os canais 1 e 2 são os utilizados para transmissão dos dados. Para acessar a rede externa à instituição, é preciso passar pelo roteador da região DSC, e, dele, ir para o *Firewall*. Caso o canal 1 ou o canal 2 fique indisponível, seja por rompimento da fibra, seja por problema na porta do roteador onde essa fibra é conectada, o canal 3 passa a funcionar provendo um rota alternativa na rede.

Já que esses equipamentos estão interconectados em anel e não está sendo feito balanceamento de carga da rede (utilização de mais de um canal para transmissão de dados de uma mesma origem para o mesmo destino, fazendo com que um canal não seja sobrecarregado), um dos canais não trafega dados dos clientes, somente dados de comunicação entre esses três equipamentos, para que, caso um dos outros canais seja desativado, uma rota alternativa seja definida.

O sistema observa os dados passados nos três canais, avaliando, assim, se o comportamento da rede está normal ou se o canal alternativo teve de ser acionado e, em caso de falha, em que canal ela correu.

4.3.2.Implementação do Modelo

Poderiam existir equações que definissem o comportamento do sistema, mas, no momento, esta informação não está disponível. Existem somente dados de entrada e saída que mostram seu comportamento. Desta forma, foi preciso criar um componente de identificação de modelos estáticos que, recebendo os dados de entrada e saída normais do sistema, pudesse automaticamente gerar o modelo desse sistema.

Para a identificação deste modelo, foi utilizado o método Least Squares. Este método consiste de uma otimização matemática que procura encontrar o melhor ajustamento para um conjunto de dados. É um método muito utilizado para estimar modelos de funções que representam saídas de sistemas. Para a implementação deste componente, foi utilizada uma biblioteca em Java para este fim, chamada Drej [Dre].

Este componente foi denominado de LeastSquaresMIMO, pois trabalha com múltiplas entradas e múltiplas saídas. Como entradas do sistema foram utilizados os canais das sub-redes de cada uma das três regiões da rede da universidade. A partir da quantidade de dados que cada uma dessas redes gera, os canais principais devem ter uma quantidade de ocupação. A ocupação desses três canais são as saídas do modelo.

4.3.3.Implementação do Diagnosticador

O diagnosticador criado para o terceiro modelo aproveitou os métodos utilizados pelo primeiro modelo: o método de geração de resíduos Subtrator, o método de avaliação de resíduos por limite de checagem simples, e o método de isolamento e identificação de falhas da matriz binária de diagnóstico.

Com isso, nenhuma implementação foi necessária para a criação do diagnosticador, houve reaproveitamento de componentes como esperado no trabalho. A Figura 31 mostra os componentes utilizados para o diagnóstico desse sistema:

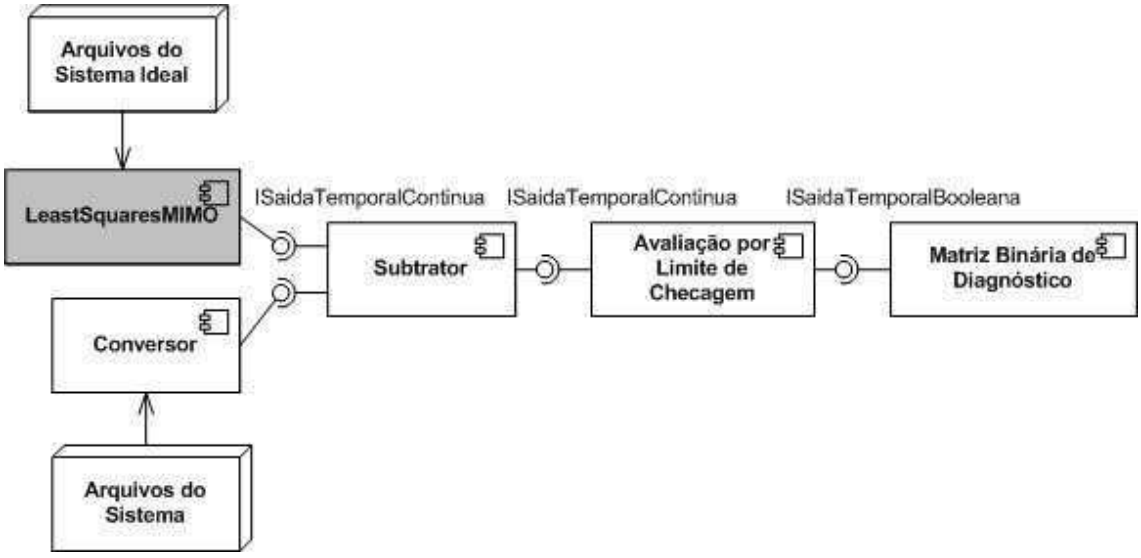

Figura 31: Componentes utilizados para terceiro diagnosticador.

Houve a necessidade de implementar somente o componente destacado com a cor cinza para a criação do terceiro diagnosticador.

4.3.4.Validação do Resultado

Dados dos canais dos blocos e do núcleo da rede da UFCG foram coletados. Com esses dados foi possível gerar o modelo do sistema, utilizando o componente de identificação de modelos. Este modelo identificado obedece a seguinte função:

$$y(t) = \sum ci\sqrt{|ui - x|^2}$$

Equação 8: Função base para o método de identificação de modelos.

Onde *ui* são as entradas utilizadas na identificação do modelo, *i* é um valor de zero até a quantidade de entradas passadas para essa identificação, *x* é a entrada real naquele instante de tempo, para o qual se quer saber a saída, e *ci* são os parâmetros calculados para que a função represente o modelo.

Após esta criação, falhas foram geradas no sistema, seus dados foram coletados e colocados em arquivos para que pudessem ser lidos pelo diagnosticador.

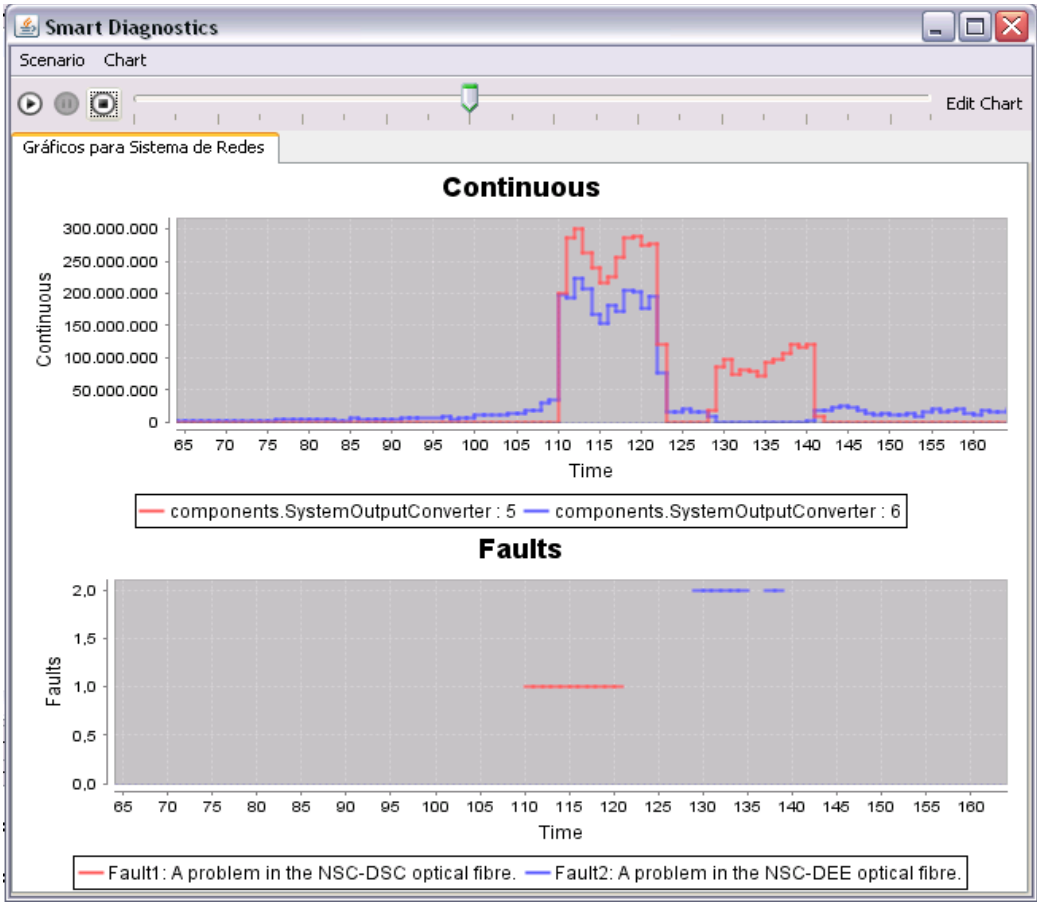

Figura 32: Falha no Modelo da Rede da UFCG.

A Figura 32 mostra o canal entre a região DSC e DEE (de número 5) e o canal entre a região NSC e DEE (de número 6). Quando ocorre uma falha entre

as regiões do DSC e do NSC, os dados do NSC passam a trafegar pela região DEE e o canal entre o DEE e o DSC passa a ser utilizado.

No segundo momento, ocorre uma falha entre a região do NSC e do DEE, com isso, os dados param de trafegar entre essas duas regiões e a região do DEE passa a utilizar o canal com o DSC. É importante ressaltar que, em nenhum momento, a rede da UFCG parou de funcionar devido a essa redundância.

Para avaliar a qualidade do diagnóstico, foram geradas cem falhas para cada uma das falhas possíveis, com 93% de acerto no diagnóstico. Estas falhas foram geradas com a desativação do canal entre a região DSC e NSC para a primeira falha e a desativação do canal entre a região NSC e DEE para a segunda falha.

4.4. Quarto modelo: Rede do Bloco REENGE com Limite de Banda

O quarto modelo escolhido também foi um sistema na área de redes de computadores. Ainda utilizando os dados da rede da Universidade Federal de Campina Grande (UFCG), a idéia foi trabalhar com limite de banda em um canal. Desta forma, pôde-se trabalhar também com modelos que obedecem a condições para determinar seu comportamento.

Como este é um sistema no qual não foi possível obter equações que o representassem, também foi necessário se trabalhar com identificação de modelos.

4.4.1. Definição do Sistema

O Sistema escolhido foi o canal de comunicação entre o bloco CB (REENGE) da UFCG e seu roteador de região. Este bloco foi monitorado e, no modelo de comportamento do canal, foi colocado um limite máximo que este

canal poderia alcançar. Quando os dados monitorados do sistema passavam desse limite, a falha era detectada.

4.4.2.Implementação do Modelo

Poderiam existir equações que definissem o comportamento desse sistema, mas, no momento, esta informação não está disponível. Existem somente dados de entrada e saída que mostram seu comportamento, assim como no modelo anterior. O componente de identificação de modelos LeastSquaresMIMO, criado para o modelo anterior, foi utilizado para a geração deste modelo, e, assim como para o modelo anterior, foram calculados os parâmetros necessários para este modelo.

Um novo componente, denominado IFExpression, foi gerado para a criação de uma condição em um modelo existente. Com ele é possível passar uma expressão que represente uma condição e valores para o caso de a condição ser verdadeira ou não. O mesmo pode receber valores de entrada e neles avaliar alguma condição. A Figura 33 ilustra esse componente:

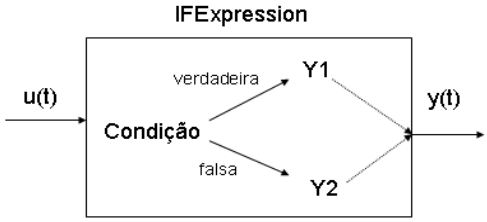

Figura 33: Componente IFExpression.

Com este componente foi definido um limite máximo para a saída do modelo e, caso esse limite fosse ultrapassado, a saída deste componente permaneceria igual à do limite máximo definido.

4.4.3.Implementação do Diagnosticador

O diagnosticador criado para o quarto modelo aproveitou os métodos utilizados pelo segundo modelo: o método de geração de resíduos Subtrator, o

método de avaliação de resíduos por limite de checagem simples, e o método de isolamento e identificação de falhas mais simples, que recebe o sintoma do avaliador de resíduos e diz, caso tenha sintoma, que uma falha na saída foi detectada. A Figura 34 mostra os componentes utilizados para o diagnóstico desse sistema:

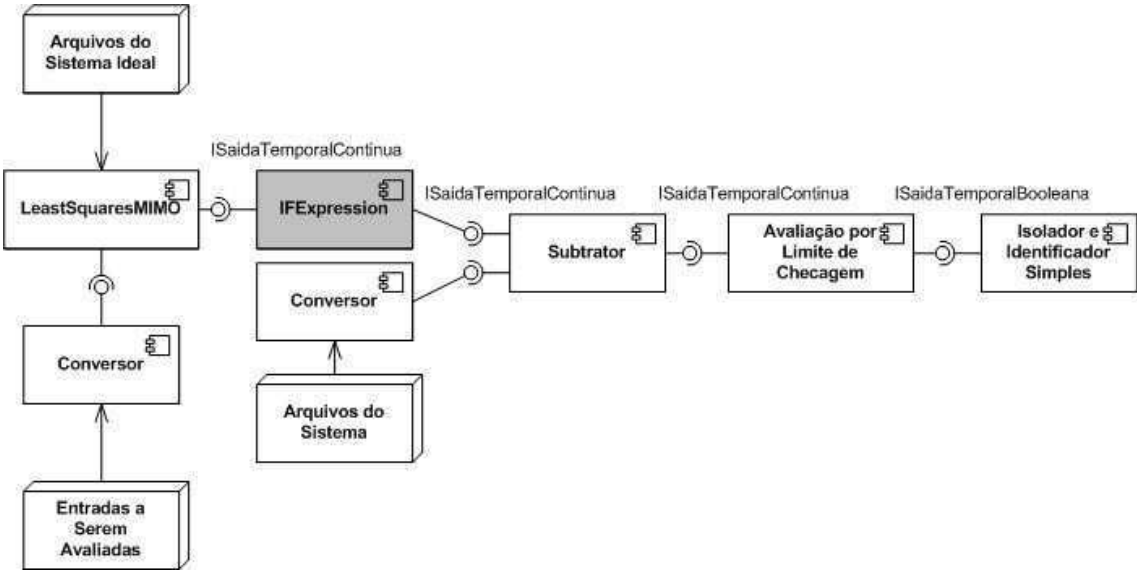

Figura 34: Componentes utilizados para quarto diagnosticador.

Com isso, nenhuma implementação foi necessária para a criação deste diagnosticador, houve o reaproveitamento dos componentes existentes.

4.4.4. Validação do resultado

Os dados do canal do bloco REENGE foram coletados e com eles foi gerado o modelo do sistema, utilizando o componente de identificação de modelos. Um limite foi definido para o controle de banda do canal e o componente IFExpression foi utilizado, recebendo os dados do modelo e comparando com o limite definido.

Falhas foram geradas no canal, ultrapassando os limites definidos e estes dados foram coletados e colocados em arquivos para que pudessem representar o sistema.

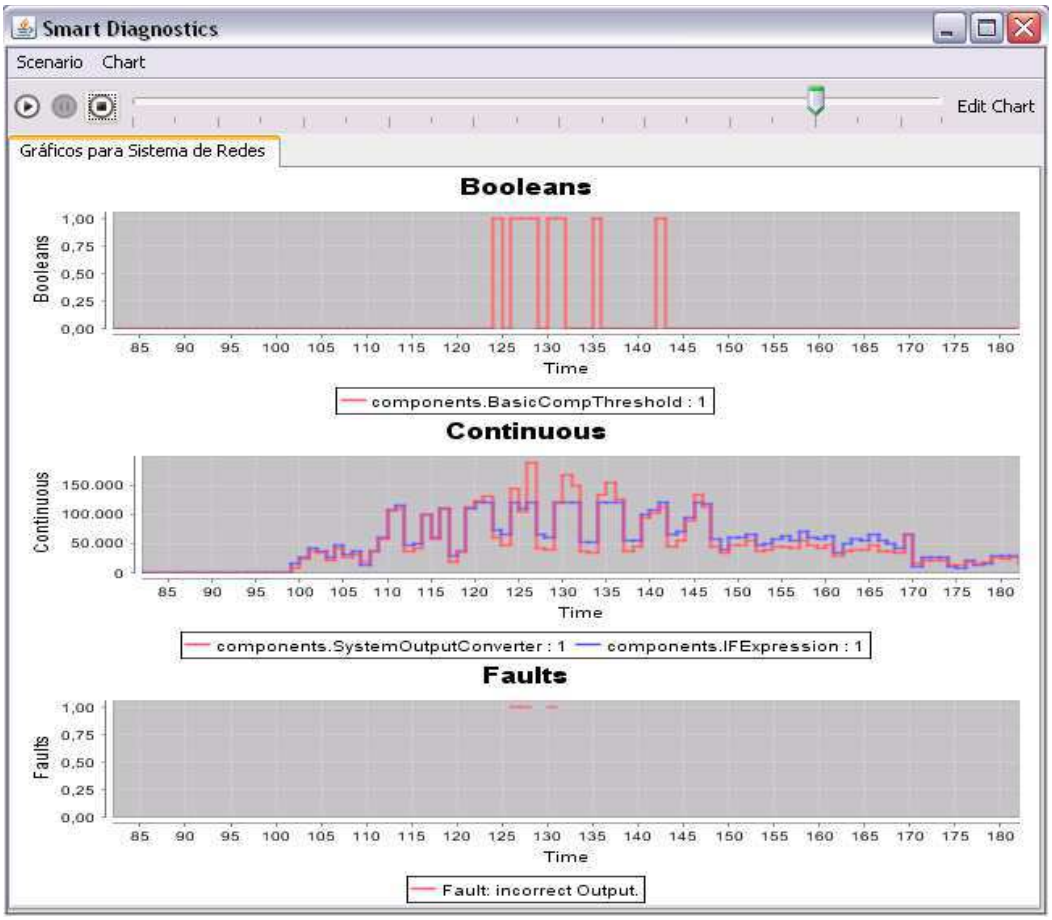

Figura 35: Falha no Modelo do bloco REENGE com Limite de Banda.

A Figura 35 mostra o canal do bloco REENGE e seu modelo com limite de banda. Toda vez que o limite é ultrapassado, o modelo mantém o valor no limite máximo e a diferença entre as saídas gera a falha.

Para avaliar a qualidade do diagnóstico, foram geradas cem falhas, assim como no modelo passado, com 96% de acerto no diagnóstico. Essas falhas foram geradas injetando periodicamente uma grande quantidade de dados no canal monitorado, fazendo com que o limite fosse ultrapassado e, ao ser comparado com o modelo, houvesse uma diferença nas saídas.

4.5. Validação da Aplicação do Framework

Estes quatro sistemas foram utilizados para validar o *framework*. Como visto na descrição de cada um, formas de modelagem e métodos de

diagnóstico de falhas foram aproveitados. Com isso, houve o reaproveitamento de código esperado na solução.

O *framework* em si e o software criado para utilizá-lo, sem os componentes utilizados nos diagnosticadores, tem 15000 linhas de código. Não estão sendo contabilizados os testes de aceitação e unidade.

Para o primeiro sistema, dos três tanques, foi preciso criar todos os componentes necessários, tanto para a modelagem quanto para o próprio diagnóstico do sistema. Para ele foram produzidas 633 linhas de código. Os seguintes componentes foram criados: o OMCAdapter, para a comunicação entre a ferramenta de modelagem e o *framework*; o gerador de resíduos Subtrator; o avaliador de resíduos por limite de checagem simples; e o isolador e identificador de falhas, utilizando a matriz binária de diagnóstico. Todos os componentes gerados são genéricos, ou seja, não contêm dados específicos deste sistema, podendo ser reaproveitados em sistemas futuros.

Para o segundo sistema, o algébrico abstrato, foi preciso criar um componente para gerar a função polinomial que iria representar o modelo; um componente que fosse possível gerar o mesmo modelo e adicionar falhas; componentes injetores de falhas e um isolador e identificador de falhas mais simples do que o do diagnosticador anterior. Para esse sistema foram produzidas 720 linhas de código.

Os seguintes componentes foram criados: DENLSISO, para modelagem de uma função polinomial; DENLSISO com falhas, para a modelagem de uma função polinomial com adição de falhas; três componentes de falhas, para inserção de falhas gradativas, abruptas, em tempo de execução ou em momentos pré-determinados; um somador, para adicionar falhas à entrada do modelo e um isolador e identificador de falhas.

Foram aproveitados os componentes relativos aos métodos de geração e avaliação de resíduos do primeiro sistema, que representam 263 linhas de

código. Assim como no sistema anterior, todos os componentes gerados são genéricos, podendo ser reaproveitados em sistemas futuros.

Para o terceiro sistema, o da rede da UFCG, foi necessário criar somente um componente para identificação de modelos, o LeastSquaresMIMO, já que não existia uma equação que representasse este sistema. Com isso, foram produzidas 137 linhas de código. Este componente também é genérico, mas, por ser a implementação de um método de identificação de modelos, pode não funcionar de maneira satisfatória para todos os sistemas, sendo necessária a escolha de outro método de identificação de modelos.

Os métodos de diagnóstico de falhas foram reaproveitados do primeiro sistema, que representam 388 linhas de código.

Para o quarto sistema, o da rede com limite de banda, um componente foi criado para simular o limite de banda no modelo, o IFExpression, com 245 linhas de código. Os demais componentes foram aproveitados dos sistemas anteriores, geração e avaliação de resíduos do primeiro sistema e identificação e isolamento de falhas do segundo sistema.

Com isso, foi demonstrado que, com o *framework*, foi possível criar quatro diagnosticadores para sistemas diferentes, com modelagens diferentes, e reaproveitar componentes em mais de um diagnosticador. Isso foi possível pelo fato dos componentes criados serem genéricos podendo ser reaproveitados nos diversos diagnosticadores. A Tabela 3 a seguir mostra, de forma resumida, este reaproveitamento:

Tabela 3: Tabela de Resumo dos Resultados.

	Quantidade de componentes utilizados	Quantidade de componentes criados	Quantidade de componentes reaproveitados
Sistema dos Três Tanques	4	4	0
Sistema Algébrico	8	6	2
Sistema da Rede da UFCG	4	1	3
Sistema da Rede do bloco REENGE com Limite de Banda	5	1	4

Nesta lista não estão sendo considerados os componentes criados para o funcionamento do *framework* como o clock, ou o observador. Só estão sendo contabilizados os componentes para criação do modelo e do diagnosticador, já que são eles que trazem novos métodos de diagnóstico.

5. Conclusão e Trabalhos Futuros

Capítulo 5

Este capítulo apresenta as considerações finais sobre o trabalho desenvolvido e sugestões para trabalhos futuros.

O objetivo do trabalho foi a criação de um *framework* para auxílio na construção de diagnosticadores de falhas, baseados no modelo do sistema. Para isso, foi necessário criar um software que pudesse utilizar este *framework* e ter uma interface acessível ao usuário.

Antes da definição da arquitetura do *framework* e da escolha dos métodos, um estudo sobre o estado da arte foi feito e, como resultado, um documento foi gerado. Este documento encontra-se no Apêndice A. Com esse estudo foi possível definir a estrutura do *framework*. Além disso, foi criado um roteiro, com o intuito de relacionar melhor os métodos apropriados para cada tipo de dados disponíveis. Este roteiro pode ser visto no Apêndice D.

Uma interface de comunicação com a ferramenta de modelagem Modelica foi criada, para que o *framework* pudesse interagir com ela, possibilitando o uso dessa ferramenta para criação de modelos dos sistemas a serem avaliados.

Antes de trabalhar com o Modelica, outra ferramenta de modelagem foi utilizada, o Anylogic [Any]. Foi criada uma interface de comunicação entre esta ferramenta e o *framework*, mas esta é uma ferramenta paga e que gera os resultados da saída do modelo utilizando um relógio próprio, sendo mais trabalhoso coletar dados no tempo desejado pelo *framework*.

Os sistemas foram escolhidos ao longo do desenvolvimento do trabalho, pois existia a esperança de que fosse possível trabalhar com algum sistema da CHESF. Dado que o projeto somente tem previsão de início para março de 2008, essa escolha não foi possível.

Com isso, os dois últimos sistemas foram na área de redes, pois foi possível ter acesso a dados da rede da Universidade Federal de Campina Grande. Uma vantagem na escolha desses sistemas foi poder diagnosticar falhas em modelos completamente diferentes entre si.

À medida que os sistemas foram sendo escolhidos, métodos para seus diagnósticos foram implementados. Além desses métodos, outros componentes foram gerados para criação de diferentes tipos de modelos: com equações diferenciais utilizando a ferramenta de modelagem; com uma função polinomial que o representasse; e com identificação de modelos que não têm uma função conhecida pelo usuário do sistema.

Componentes para injeção de falhas também foram criados, com o intuito de inserir falhas em um sistema, para que estas pudessem ser coletadas no diagnóstico, avaliando, assim, o diagnosticador.

Após a criação dos diagnosticadores, a validação do *framework* foi feita em termos de código e de componentes reaproveitados em mais de um diagnosticador. A Tabela 4 a seguir mostra esses resultados:

Tabela 4: Resultados Finais.

	Quantidade de componentes criados	Linhas de código criadas	Quantidade de componentes reaproveitados	Linhas de código reaproveitadas
Sistema dos Três Tanques	4	633	0	15000
Sistema Algébrico	6	720	2	15263
Sistema da Rede da UFCG	1	137	3	15388
Sistema da Rede do bloco REENGE com Limite de Banda	1	245	4	15481
Na coluna linhas de código reaproveitadas estão sendo considerados, além dos componentes criados para o diagnosticador específico, os componentes criados para o funcionamento do *framework* e a estrutura do *framework* em si, pois toda essa estrutura é reaproveitada a cada novo diagnosticador.				

5.1.Conclusão

Pode-se dizer que o *framework* atingiu o objetivo inicial definido neste trabalho, já que propiciou o reaproveitamento de toda a estrutura de diagnóstico de falhas baseado no modelo do sistema, assim como os métodos implementados, além de facilitar a criação de diagnosticadores.

Caso cada um desses diagnosticadores tivesse que ser implementado sem o uso do *framework*, cada método teria que ser implementado novamente e toda a comunicação entre eles teria que ser feita. Além disso, outros componentes necessários para o funcionamento do diagnosticador, como o relógio e o leitor de arquivos com dados dos sistemas a serem avaliados, precisariam ser implementados também.

Outro fato importante é que um diagnosticador novo pode ser criado com componentes existentes sem a necessidade de programar. Com esta solução nenhum conhecimento de programação é necessário, a montagem dos diagnosticadores pode ser feita utilizando uma ferramenta de auxílio na criação dos mesmos.

Observa-se, porém, que, quanto maior a dificuldade de se gerar o modelo do sistema, maior o trabalho para se utilizar esse tipo de diagnóstico de falhas, sendo interessante procurar outras formas de diagnóstico. Além disso, a fidelidade do modelo ao sistema é outro problema. Dependendo da diferença entre o modelo e o sistema que representa, o diagnóstico passa a ser cada vez mais impreciso, podendo chegar a ser incorreto.

Outra dificuldade encontrada foi a escolha de sistemas para a criação de diagnosticadores. Como a idéia não era encontrar um diagnosticador ideal para um sistema, ou criar um modelo mais fiel a este, encontrar esses dados já definidos para que pudessem ser utilizados no *framework* foi um problema a ser enfrentado. Este foi um dos motivos pelos quais foi decidido escolher um sistema da área de redes, devido à proximidade com o grupo de gerência e

suporte da rede da universidade, que facilitou acesso aos dados dessa rede, assim como a possibilidade de inserção de falhas e do seu monitoramento.

Faz-se necessária a criação de mais métodos de diagnóstico que possam ser utilizados por outros diagnosticadores, como por exemplo: métodos de isolamento e identificação de falhas utilizando redes neurais, lógica Fuzzy ou árvores de decisão. Para isso é preciso que outros sistemas que necessitem destes métodos sejam escolhidos para a implementação de seu diagnóstico.

Mesmo para novos sistemas que necessitem de novos métodos de diagnóstico, o *framework* traz toda a base da estrutura de diagnóstico de falhas, facilitando a implementação e execução deste diagnosticador, além do armazenamento de seus dados para uso posterior. E, uma vez implementados, esses métodos poderão ser utilizados para diagnóstico de outros sistemas futuramente.

Além do reaproveitamento de código e da facilidade de criação de diagnosticadores, uma importante contribuição dada pela solução foi a possibilidade de um usuário qualquer, sem nenhuma noção de programação, poder criar diagnosticadores. Uma pessoa que sente a necessidade de criar um diagnosticador pode utilizar esta solução para montar diagnosticadores sem a necessidade de ter algum conhecimento da linguagem *java* ou de programação em geral.

Esta contribuição abre espaço para um conjunto maior de pessoas que podem se beneficiar da solução proposta nesse trabalho. Pessoas que querem aproveitar o *framework* para a criação de novos diagnosticadores, implementando somente componentes novos. Pessoas que querem reaproveitar métodos já criados para outros diagnosticadores. E pessoas que em nenhum momento estão pensando em implementação, código, e que necessitam criar diagnosticadores de forma simples sem ter que se preocupar como a criação e comunicação estão sendo feitas.

5.2.Trabalhos Futuros

Como esta é uma solução que está inserida em um projeto maior, existem sugestões para trabalhos futuros que dêem continuidade a este:

- Implementação de métodos de diagnóstico de falhas, utilizando redes neurais, muito utilizados para diversos tipos de sistemas.

- Implementação de métodos de diagnóstico de falhas, utilizando lógica Fuzzy.

- Criação de novas formas de identificação de modelos de sistemas.

- Comunicação com outras ferramentas de modelagem (Simulink [Sim], por exemplo).

- Mudanças na interface gráfica para criação e alteração de diagnosticadores graficamente e não de forma textual, como é feita no momento.

- Estudo de outras formas de diagnóstico de falhas, sem utilizar modelo de sistemas, como a que utiliza modelo de sinais, por exemplo.

Estes possíveis trabalhos de continuação tornariam o esquema de diagnóstico de falhas, sugerido na solução desta dissertação, mais abrangente e com uma maior facilidade de uso.

Bibliografia

[Ise06] Isermann, R. *Fault-Diagnosis Systems*. Springer, 2006.

[KKKC04] Korbicz, J., Kościelny, J.M., Kowalczuk, Z. e Cholewa, W. *Fault Diagnosis*. Springer, 2004.

[CRB01] Chiang, L.H., Russel, E.L. e Braatz, R.D. *Fault Detection an Diagnosis in Industrial Systems.* Springer, 2001.

[DV00] Dash,S. e Venkatasubramanian, V. *Challenges in the Industrial Applications of Fault Diagnostic Systems.* Laboratory for Intelligent Process Systems, School of Chemical Engineering, Purdue University, 2000.

[DW99] D'Souza, D e Wills, A. Objects, *Components, and Frameworks with UML - The Catalysis Approach*. Addison-Wesley, 1999.

[Cys98] Cysneiros L.M., Leite J.C.S.P., *Utilizando requisitos não funcionais para a análise de modelos orientados a dados*, Workshop de Engenharia de Requisitos - XII SBES, 1998.

[Mod] - Modelica and the Modelica Association, disponível em: http://www.modelica.org/, acesso em fevereiro de 2008.

[Sim] – Simulink, disponível em: http://www.mathworks.com/products/simulink/, acesso em fevereiro de 2008.

[Any] – Anylogic - Simulation Software and Services, disponível em: http://www.xjtek.com/, acesso em janeiro de 2008.

[GHJV95] Gamma, Helm, Johnson e Vlissides, *Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[XML] - Extensible Markup Language, disponível em: http://www.xml.com/, acesso em janeiro de 2008.

[Jun] – JUNIT, disponível em: http://www.junit.org/, acesso em janeiro de 2008.

[Eas] – Easyaccept, disponível em: http://easyaccept.sourceforge.net/, acesso em janeiro de 2008.

[XP] – ExtremeProgramming, disponível em: http://www.extremeprogramming.org/, acesso em fevereiro de 2008.

[Xpl] Xplanner, disponível em: http://www.xplanner.org/, acesso em fevereiro de 2008.

[Dre] – Drej - Java Regression Library, disponível em: http://www.gregdennis.com/drej/, acesso em janeiro de 2008.

[Ise2004] Isermann, R. *Model-Based Fault Detection and Diagnosis - Status and Applications*, IFAC, 2004.

[SSLG05] Sun, H.Q., Sun, L.H., Liang, Y.C. e Guo, Y.J. *The Module Fault Diagnosis of Power Transformer Based on GA-BP Algorithm.* Hebei University of Science and Technology, China, 2005.

[ZDLG96] Zhang, Y., Ding, X., Liu, Y. e Griffin, P. J. *An Artificial Neural Network Approach to Transformer Fault Diagnosis.* IEEE Transactions on Power Delivery, Vol. 11, No. 4, 1996.

[PK04] Patel, K. e Khubchandani, R.K. *ANN Based Power Transformer Fault Diagnosis.* IE (I) Journal, vol 85, 2004.

[HS03] Han, Y e Song, Y.H *Condition Monitoring Techniques for Electrical Equipment -- A Literature Survey*, IEEE Transactions on Power Delivery, vol. 18, No. 1, 2003.

[Grimmelius *et al.*, 99] Grimmelius, H.T., Meiler, P.P., Maas, H.L.M.M., Bonnier, B., Grevink J.S. e Kuilenburg, R.F. van *Three State-of-the-Art Methods for Condition Monitoring*, EEE Transactions on Industrial Electronics, vol. 46, No. 2, 1999.

[Ise97] Isermann, R. *Supervision, Fault-Detection and Fault-Diagnosis Methods - An Introduction*, Laboratory of Control Engineering and Process Automation, Darmstadt University of Technology, Germany, 1997.

[YLT02] Yu, C.C., Lin, T.M. e Tang, Y.C. *The Application of a Simple Neural Network for Fault Diagnosis System*, Proceeding of the Fourth IASTED International Conference Signal and Image Processing Hawaii, 2002.