UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

SISTEMA DE MONITORAMENTO E CONTROLE LABORATORIAL

Autor:

Thiago Siguenobu Vargas Arakaki

Orientador:

Prof. Antônio Cláudio Gómez de Sousa, M. Sc.

Examinador:

Prof. Carlos José Ribas D'Ávila, M. Sc.

Examinador:

Prof. Jorge Lopes de Souza Leão, Dr.Ing.

DEL

DEZEMBRO DE 2007

Dedicatória

Aos meus amados pais, Armando e Zulmira, que alicerçaram com princípios e amor a pessoa que sou hoje.

Ao meu amor, Fernanda, que é a pessoa mais incrível que já conheci.

Agradecimentos

A Deus, pois sem esta força maior eu jamais chegaria a lugar algum.

A minha família, que tanto amo e tanto me incentivou em minhas conquistas.

Ao meu professor e orientador, Antônio Cláudio Gómez de Sousa, uma vez que me auxiliou a compreender temas de extrema importância, fundamentais para trilhar a nobre carreira de engenharia.

Ao professor e coordenador do curso de Engenharia Eletrônica e de Computação, Carlos José Ribas D'Ávila, por seu entusiasmo, por sua cordialidade, por estar sempre acessível à todos os alunos do departamento e por ter me cedido, gentilmente, o espaço e os equipamentos do LASPI, utilizados neste projeto.

A Universidade Federal do Rio de Janeiro e todos seus funcionários e professores, que durante os meus cinco anos de graduação foram fundamentais para minha formação, tanto acadêmica, quanto pessoal.

Resumo

Este projeto foi elaborado para a cadeira de projeto final do curso de Engenharia Eletrônica e de Computação da Universidade Federal do Rio de Janeiro.

O projeto em questão consiste na implementação de um sistema de monitoramento e controle de equipamentos laboratoriais. Através do sistema, o usuário poderá controlar e monitorar equipamentos laboratoriais, tais como bombas, medidores de temperatura, osciloscópios e geradores de sinal, por meio de módulos desenvolvidos para comunicação com cada um dos tipos de equipamento. O usuário poderá visualizar graficamente, armazenar e gerar relatórios de dados de processos, gerenciar a integração dos módulos ao sistema, gerenciar a criação e execução de procedimentos para os módulos e visualizar os estados e variáveis de sistemas dos módulos em uma tela sinótica.

Faz parte do escopo deste projeto a implementação de módulos de comunicação para o osciloscópio Tektronix TDS 3052B e o gerador de sinais Agilent 33220A, de modo a disponibilizar a automatização de experimentos de laboratório simples.

Palavras-chave

LECIS

Java

Eclipse RCP

Programação

Controle

Laboratório

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Siglas

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASTM	American Society for Testing and Materials
AWT	Abstract Window Toolkit
BIRT	Business Intelligence and Reporting Tools
CDT	C/C++ Development Tools
CORBA	Common Object Request Broker Architecture
DAO	Data Access Object
DCD	Device Capability Dataset
DCOM	Distributed component object model
DER	Diagrama Entidade Relacionamento
DNS	Domain Name System
DTP	Data Tools Platform
EMF	Eclipse Model Framework
EPL	Eclipse Public License
GEF	Graphical Editing Framework
GPIB	General Purpose Interface Bus
GPL	GNU General Public License
GUI	Graphical user interface
GPL	GNU General Public License
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machines Corporation

IDE	Integrated Development Environment
IP	Internet Protocol
IVI	Interchangeable Virtual Instrument
JAR	Java Archive
JDT	Java Development Tools
JDBC	Java Database Connectivity
JPEG	Joint Photographic Experts Group
JRE	Java Runtime Environment
JVM	Java Virtual Machine
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
LAN	Local Area Network
LECIS	Laboratory Equipment Control Interface Specification
LGPL	Lesser General Public License
MSDN	Microsoft Developer Network
MVC	Model View Controller
OMG	Object Management Group
OSGi	Open Services Gateway initiative
PDE	Plug-in Development Environment
RCP	Rich Client Platform
RPC	Remote Procedure Call
SCD	System Capability Dataset
SGBD	Sistema Gerenciador de Banco de Dados
SLM	Standard Laboratory Module
SOAP	Simple Object Access Protocol

SQL	Structured Query Language
SWT	Standard Widget Toolkit
TCL	Tool Command Language
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
TPTP	Test & Performance Tools Platform
TSC	Task Sequence Controller
UFRJ	Universidade Federal do Rio de Janeiro
USB	Universal Serial Bus
VISA	Virtual Instrument Software Architecture
VME	Versa Module Europa, IEEE 1014-1987
VXI	VME Extensions for Instrumentation
XML	Extensible Markup Language

Capítulo 1

1 Introdução

A difusão da utilização de instrumentos digitais em laboratórios já é uma realidade no Brasil e no mundo, já é possível utilizá-los para qualquer tipo de experiência de laboratório. Já que as medidas realizadas por instrumentos, ainda que analógicos, podem ser digitalizadas por instrumentos digitais especializados.

A partir disso, ficou mais fácil a comunicação dos instrumentos de laboratório com os computadores de mesa (desktops). Sendo assim, hoje já existem diversos tipos de interfaces no mercado para cada tipo de instrumento, como portas serial, paralela, USB, GPIB, Ethernet, entre outras.

Entretanto, cada fabricante de instrumentos de laboratório, normalmente, disponibiliza seus próprios serviços e produtos para a plena utilização da interface com computadores, como drivers, softwares, protocolos de comunicação e assim por diante. Criando assim, um ambiente heterogêneo para a maioria das experiências de laboratório, que utilizam diversos tipos de instrumentos, que na maioria dos casos, também são de diferentes fabricantes.

Portanto, agregando o acima exposto ao fato de que os computadores e seu uso são indispensáveis em todos os laboratórios, observa-se a necessidade da integração do controle e do monitoramento de experiências em um único sistema.

1.1 Visão Geral e Objetivos

O presente documento versa sobre o Sistema de Controle e Monitoramento Laboratorial. O sistema permite ao usuário controlar um conjunto de instrumentos de maneira simples e intuitiva, criar e executar procedimentos laboratoriais complexos, monitorar o estado e as propriedades de um determinado instrumento, e obter gráficos e relatórios referentes aos resultados dos procedimentos executados.

O sistema foi pensado para fazer a interface e controle de instrumentos laboratoriais. Estes normalmente não seguem um padrão específico e industrial de comunicação com computadores e outros dispositivos. Disso deriva que o sistema deve ser flexível o suficiente para se comunicar com protocolos diversos, implementados individualmente por cada dispositivo.

Outro requisito importante do sistema é que ele deve permitir a comunicação com drivers de instrumentos escritos em qualquer linguagem. Para tornar isso possível, realizou-se uma separação entre o sistema principal e os módulos dos instrumentos, com estes se comunicando via protocolo XML-RPC. Com essa abordagem, não só os drivers podem ser escritos em qualquer linguagem (que tenha suporte a XML-RPC) como eles podem estar distribuídos todos num mesmo computador ou em computadores separados, numa configuração arbitrária. O protocolo XML-RPC será explicado mais à frente nesse documento.

Como os módulos de instrumentos se referem a módulos quaisquer, torna-se imperativo, até para viabilizar a implementação da comunicação XML-RPC, que se tenha uma interface comum, estável e bem definida para os módulos dos instrumentos laboratoriais. Essa interface comum, estável e bem definida é provida no sistema pela especificação internacional LECIS, versão OMG. Esse é um padrão aceito internacionalmente, de modo que qualquer desenvolvedor em qualquer país que queira prover um driver para o sistema poderá fazê-lo, ao se basear na especificação LECIS. A LECIS será detalhada mais à frente neste documento.

O sistema foi escrito em Java, embora tenham sido feitos módulos de demonstração para outras linguagens como TCL. A aplicação principal, denominada de TSC

(Task Sequence Controller), em conformidade com a LECIS, foi escrita utilizando-se a plataforma de desenvolvimento de aplicações desktop Eclipse RCP, que faz uso da arquitetura e conceitos da IDE Eclipse para a construção de aplicações não relacionadas com a IDE. O Eclipse e a plataforma RCP serão apresentados mais à frente neste documento.

A abstração dos instrumentos laboratoriais é chamada de SLM (Standard Laboratory Model), em conformidade com a LECIS. Para a implementação específica desses módulos (SLMs) foi construído um conjunto de classes abstratas e interfaces, formando um framework de desenvolvimento, que pode ser facilmente estendido para a implementação de módulos específicos para instrumentos não previstos inicialmente.

Classes abstratas e interfaces também foram desenvolvidas para a implementação de drivers e wrappers de comunicação com os instrumentos, dessa forma também são facilmente estendidas para a construção de componentes de comunicação de diferentes protocolos, que podem ser acoplados ao sistema sem dificuldade.

O Sistema de Controle e Monitoramento Laboratorial é, portanto, um sistema robusto, flexível, podendo ser facilmente modificado e estendido para se adequar a experimentos completamente diferentes dos experimentos inicialmente concebidos para este sistema. Torna-se, portanto, uma ferramenta de valor inestimável para automação de baixo custo de um sistema laboratorial qualquer.

Em conformidade com a LECIS, o Sistema de Controle e Monitoramento Laboratorial pode automatizar quaisquer instrumentos, porém, para isso é necessária a implementação de módulos específicos. Mais especificamente, para este projeto, serão desenvolvidos módulos para os seguintes instrumentos laboratoriais:

- Osciloscópio Tektronix 3052B
- Gerador de sinais Agilent 33220A

A comunicação dos instrumentos será feita a partir de suas respectivas portas Ethernet, sendo conectados arbitrariamente em uma rede, utilizando-se o protocolo de comunicação de cada instrumento.

A arquitetura geral do sistema está ilustrada na figura a seguir:

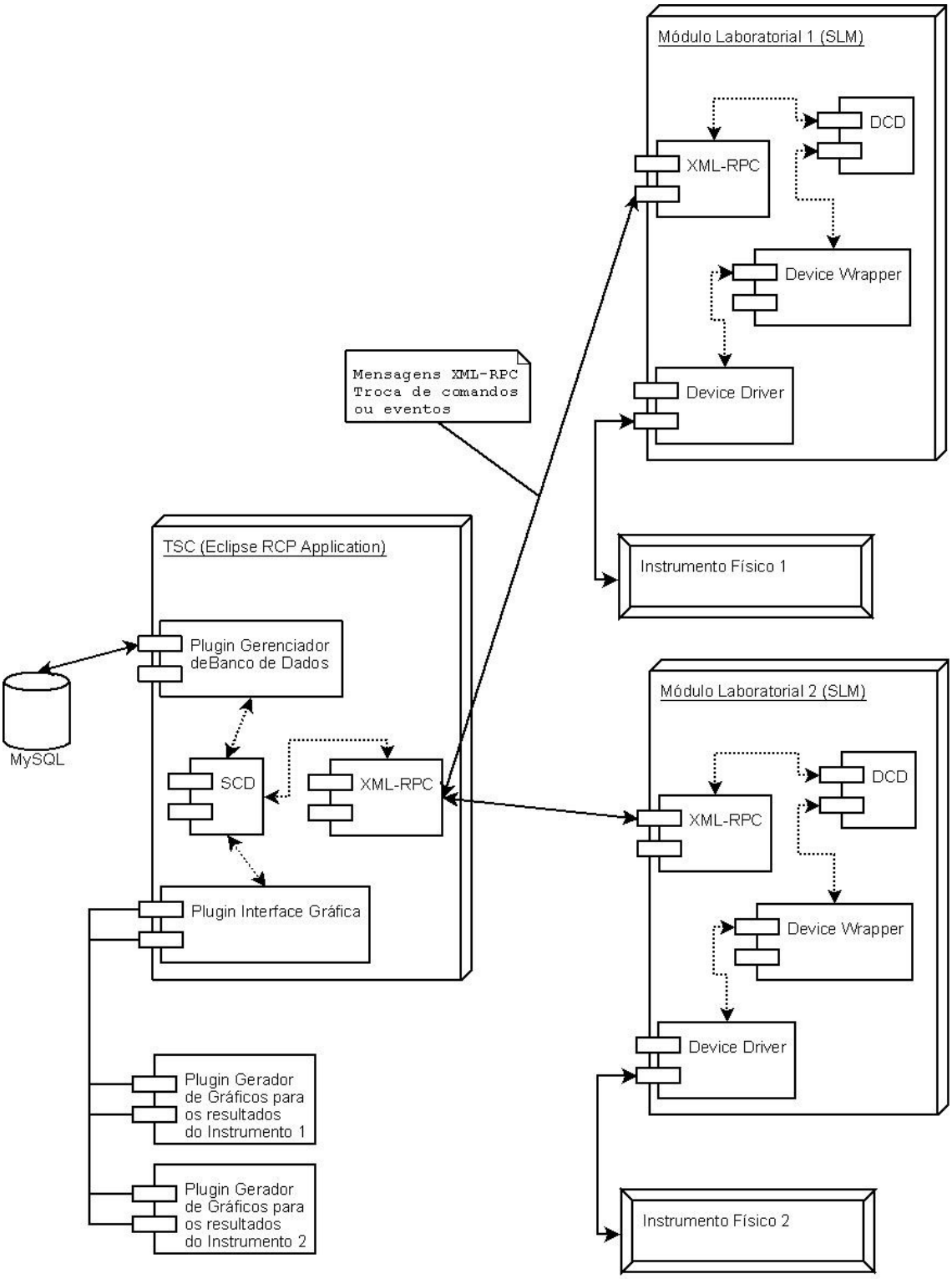


Ilustração 1Arquitetura do Sistema

Os instrumentos físicos 1 e 2 da figura acima podem ser abstraídos como os instrumentos laboratoriais automatizados, Agilent 3220A e Tektronix 3052B.

Os componentes da arquitetura serão descritos mais detalhadamente ao longo deste documento.

1.2 Organização

Este documento está dividido em doze capítulos:

- Do segundo ao sexto capítulo são explicados e detalhados todos os conceitos utilizados para o desenvolvimento deste projeto.
- O sétimo capítulo explica a arquitetura elaborada para o sistema.
- Do oitavo ao décimo capítulo é detalhada a implementação dos componentes da arquitetura do sistema, bem como o desenvolvimento de possíveis extensões ao sistema.
- O décimo primeiro capítulo detalha os instrumentos laboratoriais utilizados na integração com o sistema, bem como o desenvolvimento de seus respectivos módulos e drivers de comunicação.
- No décimo segundo capítulo são expostas considerações finais sobre o Sistema de Monitoramento e Controle Laboratorial e possíveis versões futuras.

Capítulo 2

2 LECIS

2.1 Visão Geral

O intensivo processo de integração de diferentes tipos de equipamentos em um sistema automatizado é um dos maiores problemas de automatização laboratorial atualmente. Normas para hardwares e softwares são necessárias para facilitar a integração entre os equipamentos e um sistema de controle, reduzindo-se assim, os custos e empenhos no desenvolvimento de laboratórios automatizados. Um dos maiores problemas encontrados é que nesse mercado encontram-se muitos protocolos, interfaces e conceitos proprietários, e, além disso, os fabricantes não conseguem chegar a uma norma para interface de controle comum. Se tal norma comum existisse, um usuário poderia utilizar e configurar seu equipamento em um ambiente heterogêneo, que seria mais simples de configurar, mais extensível e muito mais barato de manter.

A LECIS (Laboratory Equipment Control Interface Specification), versão ASTM (American Society for Testing and Materials) Standard E1989-98, é uma tentativa inicial de resolver tais problemas. A LECIS define um modelo de estado discreto e comunicação para equipamentos laboratoriais.

Desde o lançamento da versão ASTM da LECIS, a indústria de automatização laboratorial evoluiu, utilizando-se de ambientes heterogêneos e de paradigmas de controle de equipamentos orientados a objeto. Com a observação das tendências, a LECIS foi atualizada em 2002, com o lançamento da norma OMG CORBA LECIS, tornando mais fácil sua implementação e uso, assim como independente de plataforma e linguagem de programação. Não forçando os fornecedores de equipamentos e usuários a adotar um novo ambiente de TI.

A LECIS especifica também uma representação descritiva das funcionalidades e propriedades de um equipamento através de um arquivo de recursos, chamados DCD (Device Capability Dataset), tornando realidade o conceito de plug-and-play, já que cada módulo ao integrar-se a um controlador, deverá fornecer seu DCD, capacitando assim, o controlador de enviar e receber sinais compreensíveis de cada instrumento. O DCD e sua construção serão detalhados mais adiante.

A partir desse esforço de preencher o vazio de uma norma para o controle distribuído de instrumentos, a proposta da LECIS é definir uma interface genérica de controle de dispositivos, que permitirá deterministicamente o controle remoto de equipamentos laboratoriais independentemente do tipo de hardware. Ou seja, o objetivo é definir a comunicação de comandos e eventos entre os controladores e os instrumentos em um ambiente distribuído, como exemplificado na figura a seguir.

2.2 Conceitos Básicos

TSC (Task Sequence Controller) é o controlador do sistema automatizado, que possui um SCD (System Capability Dataset), que o provê suas funcionalidades. O TSC pode controlar um ou mais SLMs (Standard Laboratory Module), como ilustrado na figura a seguir.

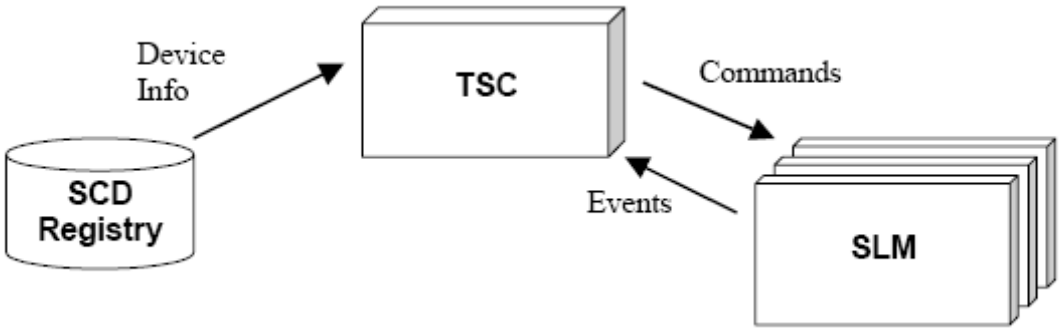


Ilustração 2 Conceito básico de controle

O SLM serve com um encapsulador de cada equipamento laboratorial, ele que é responsável em gerenciar o hardware em si, como ilustrado na figura a seguir. Quando integrado em um ambiente, o SLM fornece o seu DCD (Device Capability Dataset) ao TSC. Com o DCD, o TSC possui informações detalhadas sobre os comandos, tipos de mensagens e eventos do SLM. O SLM processa comandos e coleta dados do hardware do instrumento automatizado, passando-os para o TSC, que fará seu armazenamento e processamento

subseqüente.

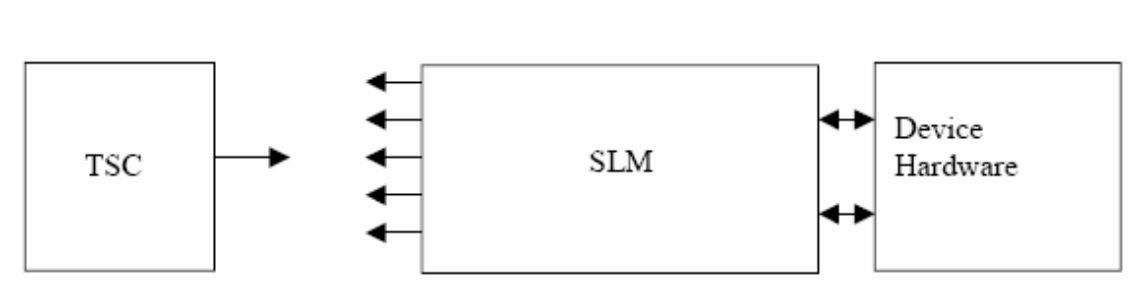


Ilustração 3 Visão abstrata do módulo laboratorial

A iteração entre os SLMs e o hardware não é contemplado na LECIS. Essa iteração é dependente do protocolo de comunicação e interface do fabricante, devendo ser implementada para cada tipo de equipamento.

2.3 Modelo de Controle de Estados

Cada SLM deve ser responsável pelo gerenciamento global de seus estados. Os estados são importantes para que o TSC possa identificar o modo de operação do SLM, enviando-o comandos primários para que o SLM execute rotinas pré-definidas, permitindo que a transição de estados ocorra de acordo com o necessário.

Os estados de controle principal são enumerados como:

- DOWN – O modulo ainda não foi ligado e não pode fazer nada útil.
- POWERED_UP – Representa o estado imediatamente após o módulo ser ligado, caso um módulo não consiga ser ligado ele é transferido novamente para o estado DOWN, caso ocorra um erro que impeça a operação, o módulo é levado ao estado ERROR.
- INITIALIZING – Estado de transição para o estado NORMAL_OP, indicando que o módulo está executando a operação de inicialização.
- NORMAL_OP – Estado de operação normal, único estado em que o módulo pode aceitar comandos específicos do dispositivo.
- ERROR – Estado em que o módulo se encontra em erro.
- CLEARING – Estado de transição para o estado CLEARED, indicando que o módulo está executando o comando primário clear.
- CLEARED – Nesse estado o módulo deve ser novamente inicializado para executar operações específicas do dispositivo.

- SHUTDOWN – Quando um módulo é desligado corretamente ele é levado ao estado SHUTDOWN e em seguida é trazido novamente para o estado DOWN, para sua utilização é necessário religá-lo ou reiniciá-lo.
- ESTOPPED – Em condição de emergência, todas as ações devem ser interrompidas imediatamente, e o módulo é transferido ao estado final ESTOPPED.

A transição entre os estados pode ser melhor observada no diagrama seguir:

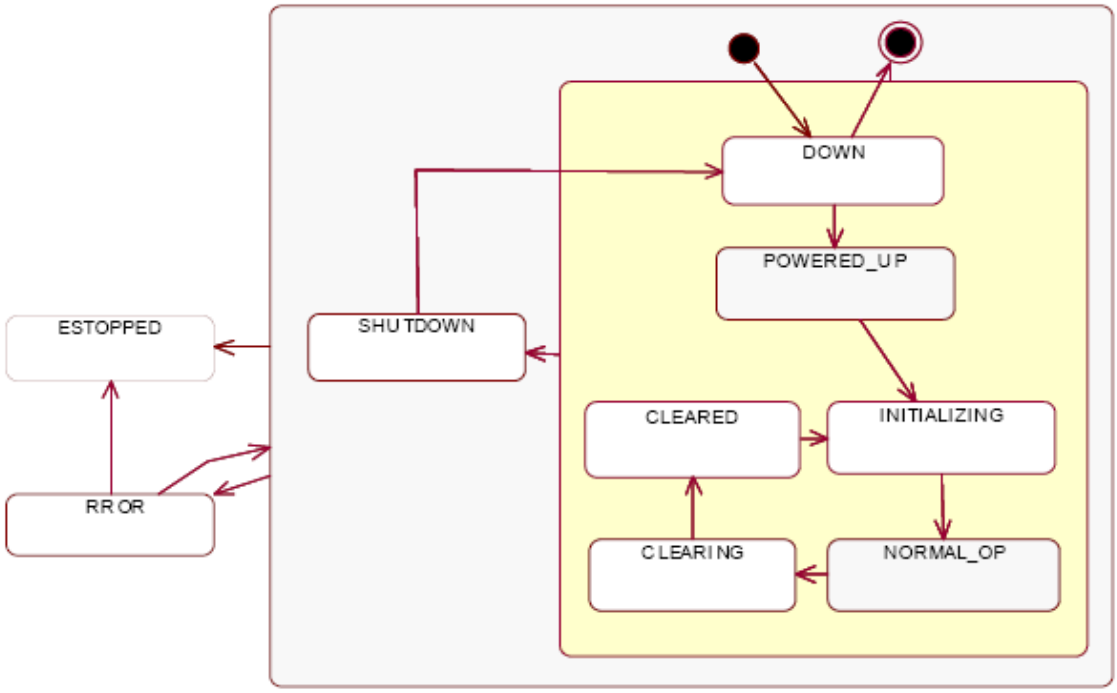


Ilustração 4 Diagrama de Estados

Este documento não contemplará as subunidades e sub-estados da LECIS, já que o sistema não utiliza esse nível de abstração.

2.4 Standard Laboratory Model

2.4.1 Device Capability Dataset

DCD é um arquivo de recursos, onde estão contidas todas as informações necessárias para o controle de um SLM. O DCD contém a identificação do equipamento, dimensões físicas, conjunto de comandos suportados, eventos gerados, representações de status e erros e informações de comunicação.

O DCD é definido em um arquivo no formato XML, a LECIS fornece um arquivo Schema, que determina quais elementos deverão ou poderão estar contidos em um DCD.

Uma representação abstrata do DCD pode ser observada na figura a seguir:

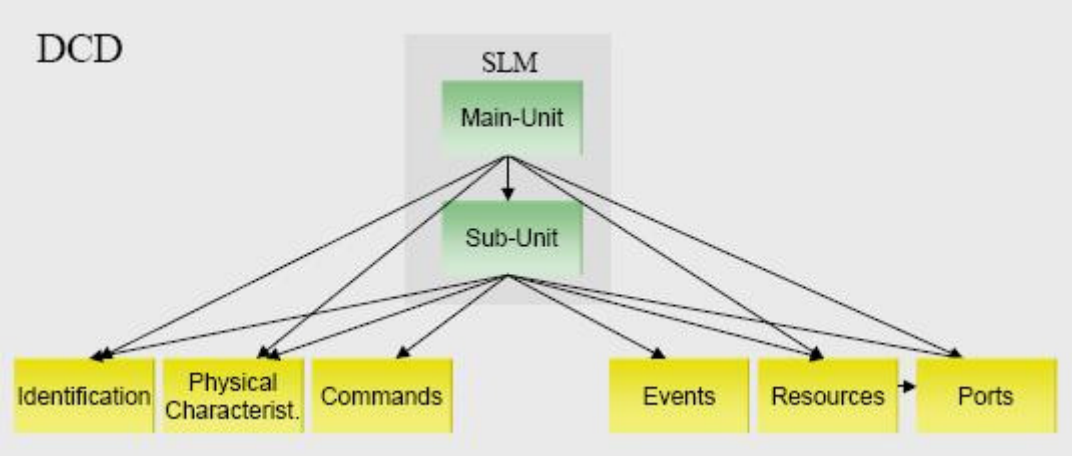


Ilustração 5 Representação Abstrata do DCD

O SLM deve enviar seu DCD ao TSC quando ele é adicionado ao sistema, disponibilizando deste modo, os recursos necessários ao seu controle, assim como atributos estáticos da representação do SLM. O diagrama de seqüência a seguir exemplifica como é deverá ser feita a integração do SLM com o TSC:

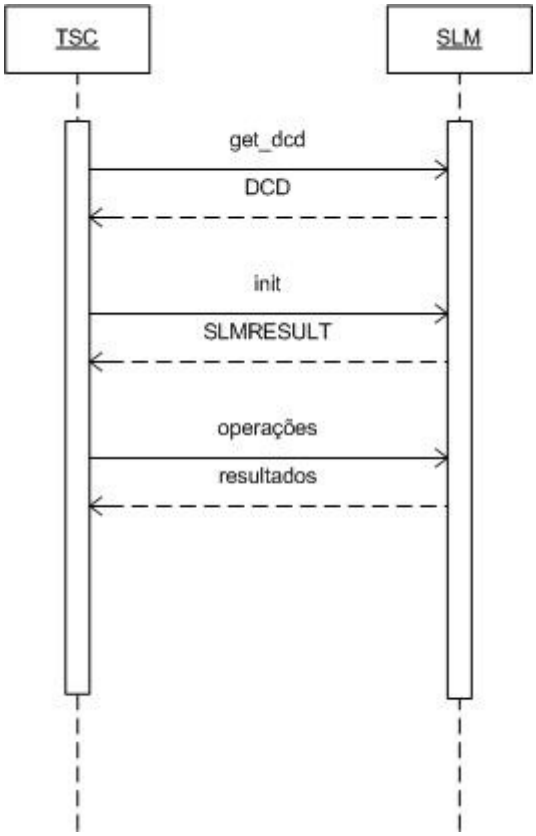


Ilustração 6 Exemplo de seqüência realizada para a integração do SLM com o TSC

2.4.2 SLM ID

Os SLM são mapeados no TSC pelo seu ID, por isso cada SLM baseado na norma LECIS deve ter um ID único no sistema. O SLM ID está definido no DCD e deve corresponder a um único instrumento físico.

2.4.3 Unit ID

A maioria dos comandos definidos na LECIS possui como argumento o Unit ID, esse valor determina qual unidade do SLM está sendo endereçado o comando. O valor 0(Zero) é reservado ao Main-Unit (instrumento como um todo).

2.4.4 Interface ISLMFacade

A maneira criada pela LECIS de abstrair todos módulos laboratoriais foi definindo uma interface padrão de acesso às funcionalidades do SLM, tal interface é a ISLMFacade, que provê os comandos disponíveis às chamadas do TSC.

Cada SLM deverá possuir uma implementação da interface ISLMFacade, que possui comandos primários do controle de estados de um SLM, comandos de configuração e um comando de acesso às funcionalidades específicas de cada instrumento laboratorial.

2.4.4.1 Comandos Primários

Os comandos primários são aqueles definidos na interface ISLMFacade, que influenciam o controle de estados do SLM, tais comandos estão listado a seguir.

2.4.4.1.1 init

O comando init tem como finalidade inicializar o SLM, colocando-o em modo de operação normal (NORMAL_OP), esse comando só pode ser executado quando um SLM está nos estados: DOWN, CLEARED, POWERED_UP, caso o SLM não esteja em um desses estados, o comando deverá retornar o código MAIN_STATE_INCORRECT.

Durante o processo de inicialização, o SLM deverá estar no estado de transição INITIALIZING, e deverá receber os parâmetros específicos de inicialização, definidos em seu arquivo DCD. Um dos parâmetros do comando poderá ser o endereço de retorno (tsc_callback_address) do SLM para o envio de eventos ao TSC, senão, o comando set_TSC_callback deverá ser executado anteriormente.

Caso o comando seja executado com sucesso, o estado final do SLM deverá ser NORMAL_OP.

2.4.4.1.2 clear

O comando clear é executado normalmente quando o SLM está em estado de erro, algum comando foi abortado ou quando algum estado interno precisa ser resetado.

Durante a execução do comando clear, o estado do SLM deve ser CLEARING, e caso o comando seja executado com sucesso, deverá apresentar o estado CLEARED.

No estado CLEARED o SLM deverá ser inicializado para que possa novamente ser operado.

2.4.4.1.3 abort

O comando abort interrompe as operações correntes do SLM. Esse comando só pode ser executado no estado NORMAL_OP, e não afeta o estado principal do SLM. Caso uma operação seja efetivamente abortada, o comando deverá retornar o código EXECUTION_STOPPED.

2.4.4.1.4 estop

O comando estop causa a interropção imediata de todas as operações em execução no SLM, levando o SLM ao estado final ESTOPPED.

2.4.4.1.5 pause

O comando de pause é apenas um pedido ao SLM que pause as operações em execução, o pedido pode ser negado, retornando ao TSC o código PAUSE_REQUEST_DENIED.

O comando não retorna eventos para o TSC, caso seja necessário, o TSC deverá checar se as operações foram efetivamente pausadas.

O estado principal do SLM não é afetado pelo comando de pause.

2.4.4.1.6 resume

O comando resume continua as operações que foram pausadas anteriormente.

Um comando de resume não pode ser negado e não afeta o estado principal do SLM.

2.4.4.1.7 shutdown

O comando shutdown desliga o módulo, levando-o ao estado DOWN ou ele é religado automaticamente, indo para o estado POWER_UP, esse comportamento é específico de cada implementação.

2.4.4.2 Comandos de Configuração

Os comandos de configuração alteram valores de variáveis básicas de controle dos módulos e podem ser executados independentemente do estado do SLM, não afetando seu controle de estados.

2.4.4.2.1 synchronize_time

A sincronização do tempo do sistema e do instrumento pode ser feita através desse comando, enviando-se um timestamp.

O comando pode não ser disponibilizado pelo SLM, retornado o código TIME_SYNCHRONIZATION_NOT_AVAILABLE.

Caso ocorra um erro durante a sincronização, o código de erro TIME_SYNCHRONIZATION_FAILED deve ser retornado.

2.4.4.2.2 set_tsc_callback

Configura o endereço de retorno de resposta e eventos do SLM ao TSC.

2.4.4.2.3 status

Retorna o status atual do SLM, com o valor do estado atual e último código de retorno.

2.4.4.2.4 get_slm_id

Retorna o valor do Id do SLM.

2.4.4.2.5 get_dcd

Retorna o arquivo DCD do SLM, é primeiro comando a ser executado durante a integração do SLM com o TSC.

2.4.4.2.6 local_remote_req

O comando é utilizado para definir quando o dispositivo pode ser acessado remotamente ou localmente.

2.4.4.2.7 get_subunit_ids

Retorna os ids de todas as subunidades do SLM, capacitando assim, o TSC de verificar quais subunidades estão fisicamente disponíveis.

2.4.4.3 Comandos de Variáveis de Sistema

Os comandos de variáveis de sistema são utilizados para a manipulação das variáveis de sistemas dos SLMs e não podem afetar o estado principal do SLM. As variáveis de ambiente serão melhor detalhas mais adiante.

2.4.4.3.1 set_system_var

Altera o valor de uma variável de sistema. O comando só pode ser executado no estado principal NORMAL_OP.

2.4.4.3.2 get_system_var

Retorna o valor de uma variável de sistema.

2.4.4.4 Comandos Específicos

Os comandos específicos são definidos no arquivo DCD de cada SLM e são executados assincronamente, logo, o TSC deverá requisitar o valor da operação através do comando get_result_data, utilizando-se de um número de identificação da operação.

Durante a execução de um comando específico o SLM poderá enviar eventos ao TSC informando-o com o status da operação. Quando a operação é finalizada o TSC é informado e pode requisitar o valor da operação através do comando get_result_data.

2.4.4.4.1 run_op

O comando run_op possui implementação específica para cada tipo de instrumento e pode executar quantas funcionalidades for necessário, passando-se como parâmetro o nome da operação a ser executada.

O comando só pode ser executado no estado de operação normal (NORMAL_OP).

2.4.4.4.2 get_result_data

Retorna o resultado de uma operação executada pelo comando run_op.

2.4.5 Variáveis de Sistema

Cada SLM poderá também dispor de variáveis de sistema, que são atributos dos SLM que podem ser monitorados e alterados, a princípio, independente de operações em execução. As restrições de acesso podem ser feitas programaticamente durante a implementação de cada SLM em específico.

As variáveis de sistema, chamadas de System Variables ou SysVar na LECIS, estão definidas no DCD de cada SLM. Quando o TSC recebe o DCD do SLM, ele registra quantas e quais variáveis de sistemas estão disponíveis no SLM, podendo requisitar seus atributos.

As variáveis de sistema incluem os seguintes atributos:

- variable_id – identificação da variável de sistema.
- description – breve descrição da variável de sistema.
- category – a categoria específica em que se encaixa a variável.
- valor – o valor atribuído à variável de sistema, pode assumir qualquer tipo de valor.

As variáveis de sistemas não devem substituir as operações específicas (run_op) e não devem alterar o estado principal dos SLMs.

2.5 Task Sequence Controller

2.5.1 Estrutura SLMRESULT

Todos os resultados dos comandos, enviados do TSC a um SLM, são estruturas do tipo SLMRESULT, essa estrutura pode possui as seguintes informações:

- result_code – o código do resultado.
- minor_code – uma extensão do result_code, podendo conter um detalhamento do código especificado.
- main_state – o estado principal do SLM.
- lr_mode – o mode de operação do SLM.
- message – uma mensagem que possa ser lida pelo usuário.

2.5.2 Interface ITSCFacade

A interface ITSCFacade é a responsável por receber os eventos gerados pelos SLM e tratá-los da maneira correta, distribuindo-os pelo sistema.

Existem três razões para que o SLM envie uma mensagem ao TSC:

- O estado do SLM foi alterado.
- Alguma variável de sistema do SLM foi alterada.
- O SLM deseja enviar dados ao TSC.
- Os eventos são identificados pelo TSC a partir dos parâmetros:
- slm_id – Número de identificação do SLM.
- unit-id – Número de identificação da unidade específica do SLM.
- event_id – Número de identificação do evento gerado.
- event_type – Tipo de evento.
- interaction_id – Número de identificação da iteração do evento.
- priority – Prioridade de execução do evento.
- slm_result – Estrutura SLMRESULT, contendo o estado e código de retorno atuais do SLM.
- Argumentos – Parâmetros abertos à implementação.

Os tipos de eventos que um SLM pode enviar para o TSC estão descritos na tabela a seguir:

Tabela 1 EEventType

Tipo de Evento	Descrição	Parâmetros
ALARM	SLM está em condição de erro e envia seu código e mensagem	Código de erro
		Mensagem de erro
MESSAGE	O SLM precisa informar o usuário do TSC, enviando uma mensagem que deverá ser exibida em uma caixa de diálogo	Mensagem ao usuário
DATA_DIRECT	O SLM provê dados de resultados diretamente ao TSC	Seqüência de dados
		Identificação do formato de dados do resultado
		Número de identificação dos dados
		Quantidade de dados

		enviados
DATA_LINK	É utilizado quando o SLM precisa enviar dados de operação muito grandes ou dados que precisem ser gravados em arquivo ou banco de dados	Identificação do formato de dados
		Número de identificação dos dados
		Quantidade de dados enviados
		O tipo de evento de DATA_LINK (próxima tabela)
		O dado a ser transferido
SYS_VAR	Quando alguma variável de sistema do SLM foi alterada	Seqüência de estruturas das variáveis de sistema
CONTROL_STATE_CHANGE	Quando o estado principal do SLM foi alterado	Estrutura SLMRESULT, contendo o estado atual do SLM
DEVICE_STATE_CHANGE	Quando alguma propriedade específica do SLM foi alterada	Valores específicos de implementação e registrados no DCD

Os tipos de eventos de DATA_LINK estão descritos na tabela a seguir:

Tabela 2 EDataLinkType

Tipo de DATA_LINK	Descrição	Parâmetros
FILE	Quando os dados precisam ser gravados em um arquivo	Parâmetros específicos de implementação, que devem conter informações sobre o arquivo
DB	Quando os dados precisam ser gravados em um bando de dados	Parâmetros específicos de implementação, que devem conter informações do banco de dados
OPERATION	Quando o resultado é muito grande para ser enviado pelo comando get_result_data	Parâmetros de identificação e formatação do resultado obtido pelo comando get_result_data

2.6 Típico Fluxo de Controle

O diagrama a seguir exemplifica um típico fluxo de controle, desde a etapa de integração, requisição de DCD, até a execução de uma operação específica do SLM:

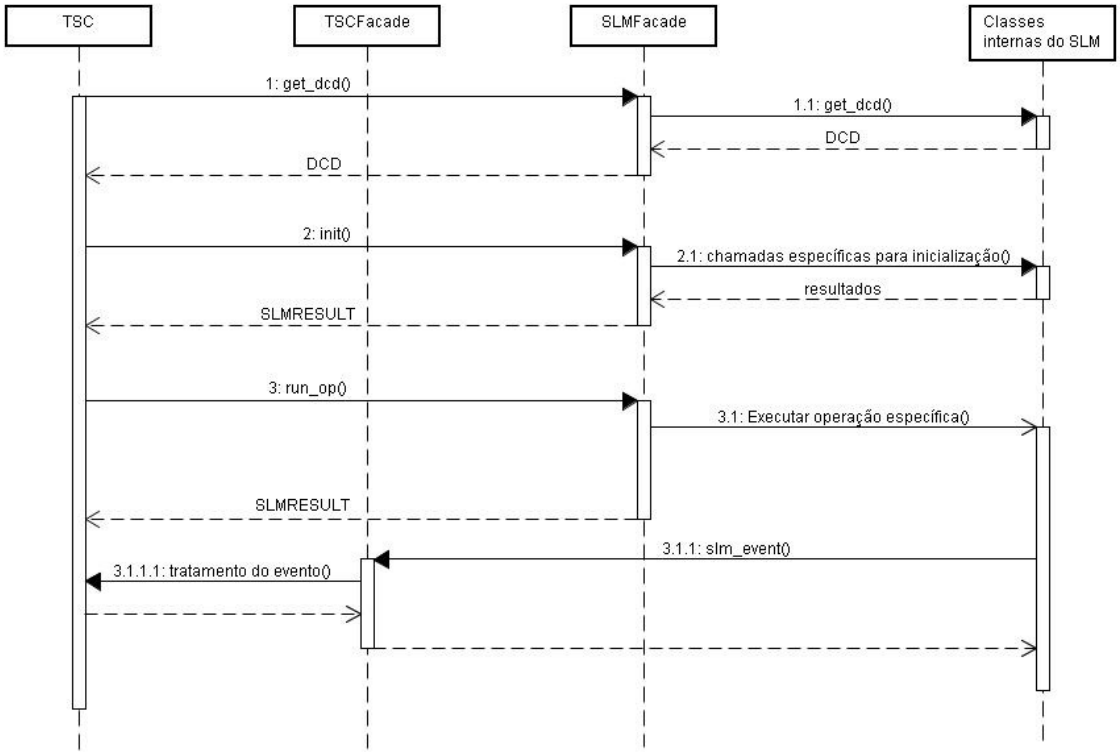


Ilustração 7 Típico Fluxo de Controle

1. O TSC obtém as informações do SLM a partir de seu DCD.
   1.1. O SLM recupera o DCD e suas informações, repassando-as para o Facade.
2. O TSC inicializa o SLM através do comando init.
   2.1. O SLM executa seus procedimentos internos específicos para sua inicialização e transfere seu estado para NORMAL_OP.
3. O TSC envia um comando do tipo run_op para o SLM.
   3.1. O SLM executa a operação especificada no comando run_op.
      3.1.1. Durante sua execução o SLM pode enviar eventos ao TSC através do comando slm_event.
         3.1.1.1.O evento lançado é tratado internamente no TSC.

2.7 Implementação Multithreaded

O TSC deve fazer chamadas síncronas ao SLM, assim como o SLM deve enviar eventos (chamadas), também síncronas, ao TSC. Porém, o TSC deverá ser capaz de controlar, ao mesmo tempo, quantos SLMs forem necessários, sem que sejam bloqueadas as suas funcionalidades, para isso, se faz necessário que a implementação do TSC seja feita utilizando técnicas multithreaded.

Por exemplo, para uma bomba, caso deseja-se acionar sua parada emergencial, essa não poderia esperar um comando de bombear terminar, que no caso seria uma operação específica.

Além disso, o TSC deve ser capaz de receber eventos dos SLM, que podem ser enviados simultaneamente, por isso o servidor do TSC para a classe TSCFacade também deve ter uma implementação multithreaded.

Os servidores dos SLMs também devem possuir implementação multithreaded, para que possam receber comandos de configuração e requisição de dados, mesmo se estiverem executando alguma operação específica.

Capítulo 3

3 Eclipse e a Plataforma RCP

3.1 Eclipse

O Eclipse é um projeto open source, iniciado pela IBM e mais tarde doado à Eclipse Foundation, que visa criar uma plataforma de desenvolvimento composta de diversas frameworks, ferramentas com o objetivo de construir, implantar e manter softwares durante todo o seu ciclo de vida. O projeto Eclipse é totalmente escrito em Java.

O Eclipse é muito conhecido no meio acadêmico e profissional por sua vocação como IDE para a linguagem Java, propósito para o qual foi inicialmente projetado. Neste mérito, ele constitui uma ferramenta extremamente madura, com recursos avançados de debug, deploy, refactoring, controle de versões e outras features que auxiliam em muito o trabalho do desenvolvedor Java.

É possível estender a funcionalidade provida pelo Eclipse através da criação de plugins utilizando o próprio Eclipse. A arquitetura do Eclipse permite inclusive que se construam aplicações com propósitos completamente distintos do propósito inicial de IDE, como é o caso do Sistema de Controle e Monitoramento Laboratorial. O processo de criação de plugins e aplicações a partir do Eclipse serão detalhados mais adiante.

3.2 Arquitetura do Eclipse

Para entender o funcionamento do Eclipse e o processo de criação de plugins e aplicações a partir do Eclipse, faz-se necessário primeiro estudar a sua arquitetura.

A arquitetura do Eclipse é totalmente composta de plugins. O seu funcionamento está baseado na composição das diversas funcionalidades contribuídas pelos seus diferentes plugins, como também pela possibilidade da contribuição de outros.

Tudo no Eclipse é um plugin ou está baseado em um plugin. Isso é extremamente importante no sentido de que existe uma uniformidade no tratamento dos componentes do Eclipse, não existem componentes que não sejam plugins e sejam tratados de maneira diferente.

No Eclipse, existe um conjunto base de plugins, denominado Platform, que fornecem os serviços básicos da plataforma aos outros plugins. Esse é o conjunto mínimo de plugins necessários para a criação de aplicações baseadas no Eclipse. O Platform é composto de dois subconjuntos de plugins: Core e UI. O Core é a parte responsável por serviços não relacionados à interface com o usuário e o UI pelos relacionados à interface com o usuário.

O Core é composto ainda de dois subconjuntos: Runtime e Workspace. O Runtime provê as funcionalidades básicas de runtime, ou seja, de chamada e instanciação de plugins, suas classes e recursos. É tarefa do Runtime também descobrir quais plugins estão disponíveis e verificar se todas as dependências foram resolvidas quando do início da aplicação. O Workspace é o responsável pelas funções relacionadas ao uso de projetos e arquivos.

O UI, por sua vez, é composto dos seguintes subconjuntos: SWT, JFace e Workbench.

O SWT (Standard Widget Toolkit) é um Widget Toolkit criado pela IBM para criação de aplicações Desktop em Java, provendo as abstrações básicas necessárias, como janelas, botões, tabelas, menus, lists, checkboxs e outros componentes gráficos encontrados em abundância nas interfaces gráficas das aplicações dos dias de hoje. O SWT também faz todo o tratamento de eventos relacionados à interação do usuário com a aplicação através de

um sistema de eventos. Dessa forma, o SWT lida diretamente com interrupções geradas pelo hardware e com o sistema operacional hospedeiro e suas system calls. Por esse motivo, o SWT é dependente de plataforma, mas existe uma implementação para cada plataforma popular atualmente.

O SWT foi criado durante o projeto inicial do Eclipse pela IBM, com a decisão de não adotar o Swing, biblioteca padrão do Java para criação de interfaces gráficas. O Swing provê uma camada de abstração para as interfaces gráficas criadas que as tornam totalmente independentes da plataforma em que estão rodando, bastando que exista para ela uma JRE (Java Runtime Environment). Essa independência da plataforma vem ao custo de um desempenho lento e uma interface com componentes muito distintos dos componentes nativos do sistema operacional, de modo que a aplicação ficava com o visual estranho quando colocada próxima às demais aplicações do sistema. Por esse motivo a IBM decidiu não utilizá-lo e criar, do zero, uma nova biblioteca para criação de interfaces gráficas. O SWT foi projetado para criar interfaces gráficas com componentes nativos do sistema operacional hospedeiro e com desempenho mais aceitável.

O JFace é uma biblioteca que utiliza o SWT para criação de abstrações gráficas de nível mais alto, como wizards, viewers, diálogos, toolbars, etc. Assim como SWT, o JFace, é um conjunto de plugins, mas pode ser utilizado fora da plataforma Eclipse, para construção de aplicações Java com interface gráfica.

O Workbench é um conjunto de plugins que se apóia no JFace para prover abstrações de nível mais alto, como views, editors e perspectives, abstrações diretamente relacionadas com o Eclipse e sua interface gráfica.

As figuras abaixo retratam a arquitetura do Eclipse:

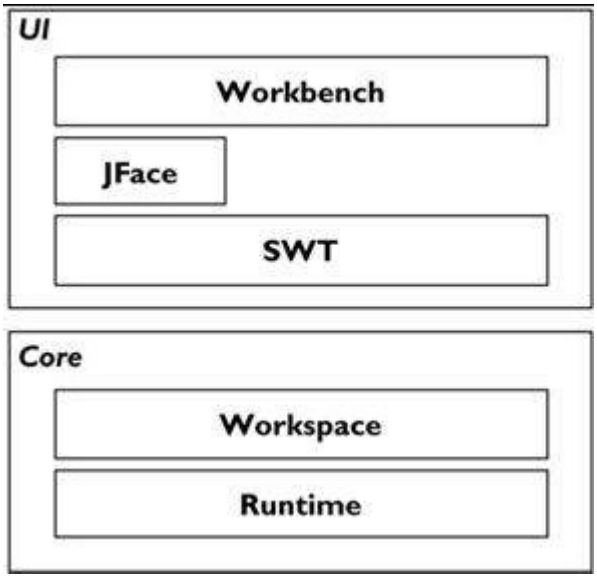


Ilustração 8 Arquitetura do Eclipse

Os demais plugins podem se apoiar em cima dos plugins básicos do Platform ou de outros plugins.

Na figura abaixo, vemos como o conjunto de plugins que constituem as ferramentas para desenvolvimento Java e de plugins se apóiam na estrutura básica provida pelo Platform:

Ilustração 9 Plugins para desenvolvimento Java

3.3 Eclipse RCP

O Eclipse RCP (Rich Client Platform) é um conjunto mínimo de plugins necessários para a criação de aplicações de propósitos gerais e baseadas na arquitetura do Eclipse.

Historicamente, o Eclipse RCP surgiu quando, ainda na versão 2.1, os desenvolvedores do projeto original do Eclipse identificaram que muitas pessoas estavam construindo aplicações Rich Client utilizando o Eclipse como base e o mecanismo de plugins para estender suas funcionalidades. Muitas dessas aplicações não possuíam relação alguma com o desenvolvimento de softwares, sendo aplicações de uso mais gerais. Foi proposta então uma mudança radical na arquitetura da plataforma, de modo a facilitar o trabalho dessas pessoas que queriam desenvolver aplicações baseadas no Eclipse.

Essa mudança deveria extrair um conjunto básico de plugins que proveriam todos os serviços da plataforma. A intenção era que os desenvolvedores de aplicações Rich Client não precisassem carregar desnecessariamente plugins ligados apenas à IDE. Na época do Eclipse 2.1, os plugins estavam todos muito acoplados e não existia esse conjunto básico. Foi então necessário um colossal esforço para separar os interesses do sistema em subsistemas e refinar a estrutura de plugins. Como resultado desse esforço, o Eclipse 3.0 apresentou uma melhora significativa na arquitetura, facilitando inclusive o desenvolvimento de plugins para a IDE pela comunidade.

O conjunto mínimo de plugins foi denominado de Eclipse RCP. O desenvolvimento de uma aplicação RCP tornou-se apenas uma questão de implementar plugins Eclipse que se apóiam sobre os providos pelo Eclipse RCP e provêem as funcionalidades requeridas pela aplicação em questão. Os plugins desenvolvidos para a IDE também podem ser integrados a qualquer aplicação RCP através do mesmo mecanismo básico utilizado para integrar plugins comuns à IDE. Isso abre um conjunto infinito de possibilidades de utilizar na sua aplicação frameworks e ferramentas criadas pela comunidade para suportar o projeto principal do Eclipse.

Pela nova arquitetura, a própria IDE Eclipse se tornou um exemplo de aplicação RCP. Conceitualmente, não existe mais qualquer diferença entre uma aplicação RCP de propósito geral e a IDE Eclipse. Ambas são aplicações RCP, compostas, em última instância, de plugins, que possuem todos uma estrutura conhecida.

Por ser baseada na arquitetura do Eclipse, uma aplicação RCP usufrui naturalmente das seguintes vantagens:

- Estrutura modular baseada em componentes – as aplicações RCP são construídas pela composição de componentes conhecidos como plugins. Os plugins podem ser versionados e compartilhados por várias aplicações. Múltiplas versões de um mesmo plugin podem ser instaladas lado a lado e aplicações podem ser configuradas para usar a exata versão de que precisam. Isso provê uma estrutura modular e facilita a manutenção e evolução das aplicações RCP.
- Infraestrutura pré-existente – toda a infraestrutura necessária para a construção de Rich Clients é provida pela plataforma RCP. O desenvolvedor não precisa se preocupar em implementar mecanismos de UI, ajuda, carregamento automático de componentes quando eles precisarem ser utilizados, atualização pela rede, tratamento de erros, etc, podendo-se ater as particularidades do software que está sendo desenvolvido, simplificando o processo e economizando tempo e dinheiro.
- Interface de usuário nativa – ao contrário de muitas aplicações desktop Java, a plataforma RCP, através do uso do SWT, possibilita a criação de aplicações que possuam um visual nativo do sistema operacional onde o sistema está instalado, melhorando a experiência do usuário final da aplicação.
- Portabilidade – como toda aplicação Java, as aplicações criadas com a plataforma RCP são portáveis. O mesmo código da aplicação pode ser utilizado para construir versões específicas para cada sistema operacional, para cada ambiente onde se quer instalar o sistema. Isso permite o desenvolvimento de aplicações portáveis com muito menos esforço.
- Operação desconectada – as aplicações RCP podem rodar standalone, sem necessidade de uma conexão com rede ou internet.
- Ambiente de desenvolvimento poderoso – o Eclipse provê o PDE (Plugin Development Environment), um conjunto de plugins que provêem uma poderosa e eficiente ferramenta para desenvolvimento de plugins e aplicações RCP.
- Sistema inteligente de instalação e atualizações – a estrutura componentizada de aplicação RCP permite que se instale ou atualize facilmente a aplicação pelo simples deploy e troca de componentes. Os componentes podem ser instalados através da update sites, Java Start, simples cópia de arquivos ou sistemas complexos de gerenciamento de componentes.
- Grande quantidade de componentes prontos – qualquer componente do projeto Eclipse ou de plugins para o Eclipse podem ser utilizados dentro de uma aplicação RCP. Com

isso, tem-se uma variedade de componentes disponíveis, como editores de texto, consoles, frameworks para edição gráfica, frameworks de modelagem, ferramentas para geração de relatórios, manipulação de dados, dentre outras.

- Robustez – caso algum componente esteja causando problemas no sistema, a plataforma RCP garante a possibilidade de desabilitá-lo ou substituí-lo em tempo de execução, sem precisar parar a aplicação. Além disso, o Eclipse possui a filosofia de lazy loading, que significa que um componente só será carregado quando ele realmente for ser utilizado, economizando recursos e evitando erros crônicos causados por um componente que nem estava sendo utilizado.
- Extensibilidade – os plugins de uma aplicação RCP podem expor um ponto de extensão, ou seja, uma funcionalidade que pode ser contribuída por plugins de terceiros, sem ser necessário que se tenha conhecimento da exata implementação dos plugins contribuintes. Com isso se possibilita a criação de uma comunidade em torno da aplicação.
- Suporte a internacionalização – o PDE possui um mecanismo prático para extrair strings e permitir a criação de plugins internacionalizados. Além disso, a aplicação pode detectar automaticamente o idioma do sistema operacional da máquina hospedeira e carregar a configuração adequada.

Por todos esses motivos, decidiu-se pela utilização da plataforma RCP para criação do Sistema de Monitoramento e Controle Laboratorial. Particularmente, foi utilizada a versão 3.2 do Eclipse RCP para construí-lo.

3.4 Principais subprojetos Eclipse

Como foi dito, o projeto Eclipse não cuida apenas do desenvolvimento de uma IDE Java poderosa e flexível. Existem infinitos outros subprojetos que estendem a funcionalidade básica de IDE para criar uma completa ferramenta para se construir, implantar e manter softwares durante todo o seu ciclo de vida.

Dentre os subprojetos Eclipse, cabe destacar os seguintes:

- BIRT (Business Intelligence and Reporting Tools), conjunto de plugins relacionados à área de Business Intelligence e à criação de relatórios.

- CDT (C/C++ Development Tools), conjunto de plugins relacionados ao desenvolvimento de sistemas na linguagem de programação C/C++.
- DTP (Data Tools Platform), conjunto de plugins relacionados a bancos de dados.
- EMF (Eclipse Model Framework), conjunto de plugins para criação de modelos.
- GEF (Graphical Editing Framework), conjunto de plugins relacionados à criação de editores gráficos.
- JDT (Java Development Tools), conjunto de plugins relacionados ao desenvolvimento de sistemas na linguagem de programação Java (IDE Java).
- TPTP (Test & Performance Tools Platform), conjunto de plugins relacionados à criação de ferramentas de teste e performance.

3.5 Conceitos básicos do Eclipse RCP

Existem alguns conceitos básicos sobre o Eclipse, seus plugins e a plataforma RCP que devem ser dominados para o correto desenvolvimento de plugins e aplicações RCP:

- Workbench – abstração ligada à interface gráfica propriamente dita da plataforma. É composta de abstrações como views, editors e perspectives. É importante ressaltar que a workbench não é a janela do Eclipse (ou aplicação RCP). De fato, existe apenas uma workbench para cada Eclipse aberto e ela pode estar ligada a uma ou mais workbench windows, que são essas sim as janelas do Eclipse.
- Workspace – abstração ligada aos projetos do usuário, os arquivos e recursos contidos nesse projeto e a área onde esses projetos são armazenados fisicamente. Cada workbench opera sobre um determinado Eclipse e apenas uma workbench pode operar sobre um dado workspace por vez. Todos os arquivos e projetos criados são armazenados dentro da pasta correspondente ao workspace no sistema de arquivos local. O Eclipse suporta também arquivos e projetos referenciados por links simbólicos, permitindo assim uma abstração completa da localização dos recursos referenciados no workspace.
- View – tipo de janela exibida dentro da workbench window. Uma view tem a finalidade primária de visualização de dados. Uma view pode ser livremente redimensionada e reposicionada dentro da workbench window, empilhada com outras views e até “detachada” (desconectada da janela principal, originando outra janela).

Em termos de API, É representada pela interface org.eclipse.ui.IViewPart e pela sua implementação default, a classe org.eclipse.ui.ViewPart.

- Editor – tipo de janela exibida dentro da workbench window. Um editor tem a finalidade primária de permitir a edição de um determinado recurso através de algum tipo de manipulação por parte do usuário. Existem editors de texto, imagem, diagramas, entre outros. Um editor está necessariamente ligado a um org.eclipse.ui.IEditorInput, interface que representa a entrada (input) de um editor. Um editor pode ser reposicionado dentro de uma área conhecida como Editor Area, mas não pode ser "detachado". Também só pode ser empilhado com outros editors. Em termos de API, é representado pela interface org.eclipse.ui.IEditorPart e pela sua implementação default, a classe org.eclipse.ui.EditorPart. Está associado ao paradigma "abrir-modificar-salvar".
- Perspective – conjunto de janelas relacionadas a um determinado objetivo ou finalidade. Apenas uma perspective é visualizada por vez no Eclipse. Apesar de se poder abrir e fechar livremente editors e views dentro de uma perspective, uma perspective define o conjunto inicial desses elementos que será mostrado na tela.
- Plugin – unidade básica da arquitetura do Eclipse. Um componente é assim chamado na computação quando se "pluga" ao conjunto principal para prover funcionalidades que antes o conjunto principal não possuía. No Eclipse, um plugin é um projeto Java que contribui com funcionalidade para a plataforma Eclipse e para outros plugins, através do mecanismo de extensions e extension points.
- Feature – grupo de plugins que estão funcionalmente relacionados, constituindo um subsistema e devendo ser instalados e atualizados simultaneamente. É através da definição de features que se pode utilizar o mecanismo nativo de install e update do Eclipse em aplicações RCP. Ao se definir uma feature, podem-se impor restrições de instalação ou atualização, como necessidade de se aceitar um termo de compromisso, necessidade de instalar antes outras features, entre outras.
- Fragment – fragmento de plugin, que pode ser usado para estender a funcionalidade provida pelo plugin. Ele funciona basicamente como um patch que é carregado em tempo de execução, alterando uma determinada versão do plugin original e possivelmente provendo novas funcionalidades. Fragments são usados principalmente para criar versões especiais de um determinado plugin para um determinado ambiente

de execução. Também podem ser usados para fornecer novos recursos, como arquivos de imagens, ou para fornecer versões internacionalizadas de um plugin.

- Extension Point – ponto em que um plugin se abre para a contribuição de outros. Um extension point é a definição de uma funcionalidade que o plugin permite que seja implementada por qualquer outro plugin sem necessidade que ele o conheça previamente. A implementação do extension point é a extension.
- Extension – implementação do extension point de um plugin. Geralmente consiste de uma classe que implementa uma interface determinada no extension point. Em tempo de execução, o runtime da plataforma detecta todas as extensions para um determinado extension point e permite que a classe onde esteja definido o extension point instancie os elementos descritos na extension sem precisar conhecê-los previamente, bastando saber que todos eles implementam a interface requerida e utilizando-a para chamar os métodos necessários.
- Action – opção que aparece em menu bars, coolbars e context menus, representa uma ação que pode ser tomada por um usuário. Podem-se restringir os elementos (recursos) sobre os quais uma action atua ou mesmo especificar circunstâncias sobre as quais ela ficará desabilitada. Uma action é representada pelas interfaces org.eclipse.ui.IAction e org.eclipse.ui.IActionDelegate e pode ser adicionada a um menu ou coolbar programaticamente ou através de extensions (esse recurso facilita a tarefa de desacoplar os plugins, visto que o mecanismo de verificação de extensions ocorre dinamicamente e em tempo de execução).
- Coolbar – barra de ferramentas onde podem ser adicionadas actions. A coolbar pode ser livremente reposicionada dentro da workbench window.
- Target – conjunto de plugins a partir dos quais a aplicação RCP será construída. Todos os plugins que forem necessários para a construção da aplicação RCP devem estar no diretório especificado para o target. Podem-se definir configurações mais refinadas para um target criando-se uma target definition.
- Product – definição do produto final a ser construído. Um product, materializado num arquivo .product, contém todas as configurações de aplicação RCP, como quais plugins serão incluídas na aplicação, qual o nome do executável a ser gerado para a aplicação, ícone desse executável, splash screen, about, dentre outras. Podem-se criar várias versões (perfis) para uma mesma aplicação RCP pela simples criação de vários

products. O editor padrão do Product possui um wizard de exportação da aplicação, facilitando o seu deploy, podendo-se ainda exportar-la para diversas plataformas.

3.6 Criação de plugins e extensão de funcionalidade

O processo de criação de plugins no Eclipse é bem simples e será descrito adiante. Para facilitar todo o processo, o projeto Eclipse inovou criando uma ferramenta própria para o desenvolvimento de plugins usando o próprio Eclipse. Essa ferramenta é o PDE (Plugin Development Environment), conjunto de plugins que fornece uma perspective, com algumas views e editors relacionados ao processo. Recomenda-se fortemente usar o PDE para desenvolver plugins.

Um plugin é um projeto Java como outro qualquer. Ele possui dois arquivos que o descrevem: plugin.xml e MANIFEST.MF. Tais arquivos podem ser diretamente editados utilizando um editor visual, chamado "Plug-in Manifest Editor". Esse é o editor padrão para editar esses dois arquivos e facilita sobremaneira a tarefa de configurar tais arquivos.

Conjuntamente esses arquivos descrevem uma série de parâmetros do plugin. A seguir falaremos sobre os principais. Cada um deles corresponde a uma guia no Plug-in Manifest Editor.

- dependencies – são os plugins dos quais esse plugin depende. Um plugin só pode fazer uso de classes, interfaces e extension points de outro plugin se ele o tiver como dependência. Em tempo de execução, o runtime do Eclipse verifica se todas as dependências de um plugin estão disponíveis (se os plugins necessários estão fisicamente colocados na pasta plugins dentro do diretório de instalação do Eclipse). Caso eles não estejam disponíveis, o runtime acusará um erro ao se tentar acessar aquele plugin.
- runtime – são as configurações relativas ao classpath (JARs externos que devam ser utilizados) do plugin e quais classes ele disponibiliza (exporta) para outros plugins. Outro plugin só pode fazer uso das classes desse plugin se as classes a serem utilizadas estiverem dentro dos pacotes exportados pelo plugin fonte das classes.

- extensions – são as extensions que esse plugin contribui para outros plugins. Contribuições visuais como views, editors e perspectives também são realizadas através de extensions.
- extension points – são os extension points que esse plugin disponibiliza para outros contribuírem. É possível com esse mecanismo, criar menus, editores e outros conteúdos dinâmicos, resultantes das diversas contribuições de outros plugins e que são determinados apenas em tempo de execução.

Um aspecto interessante dos plugins é que eles só são carregados quando se utiliza uma classe deles. Quando isso acontece, a classe que representa o plugin propriamente dito (uma subclasse de org.eclipse.core.runtime.Plugin ou org.eclipse.ui.plugin.AbstractUIPlugin, dependendo se o plugin faz alguma contribuição para a interface gráfica do Eclipse, ou não) é instanciada, dando início ao ciclo de vida do plugin. Então a classe requerida é instanciada e retornada. Esse mecanismo permite economizar muita memória na medida em que muitos recursos não são utilizados em uma sessão de uso do Eclipse e, portanto, não precisam ser carregados em memória.

Para se criar um plugin, portanto, deve-se configurar o seu plugin.xml e MANIFEST.MF de acordo com as características dele. Depois se devem codificar as classes necessárias ao funcionamento dele, tomando cuidado especial na codificação das classes que implementam as extensions desse plugin.

Uma vez finalizado seu desenvolvimento, um plugin pode ser exportado sobre a forma de um JAR ou de uma pasta contendo os arquivos .class e recursos necessários. A partir da versão 3.0 do Eclipse, a forma de JAR passou a ser a forma recomendada de exportar um plugin. O plugin exportado deve então ser colocado dentro da pasta “plugins” contida na pasta raiz de instalação do Eclipse.

Com o devido conhecimento de como se criar plugins, fica fácil estender a funcionalidade do Sistema de Monitoramento e Controle Laboratorial. O sistema é uma aplicação RCP e, como tal, aceita que se inclua qualquer plugin Eclipse, desde que se respeitem as devidas dependências. Dessa forma, estender o sistema torna-se um mérito de criar um plugin Eclipse com a funcionalidade desejada.

Capítulo 4

4 Graphical Editing Framework

4.1 Introdução

O GEF (Graphical Editing Framework) é um framework (conjunto de classes e interfaces que podem ser estendidas para prover a solução desejada) para criação de editores gráficos. Ele possui uma série de vantagens:

- Possui praticamente todos os recursos necessários para a implementação de um editor gráfico de qualquer espécie.
- É um projeto open source conhecido e bem documentado.
- É um conjunto de plugins para o Eclipse, de modo que faz uso de suas abstrações e permite uma integração fácil e direta com ele.

O GEF, como a maioria dos outros plugins do projeto Eclipse, está licenciado sob a licença EPL (Eclipse Public License), o que garante que seu código é aberto e permite que ele seja utilizado em aplicações comerciais, de código fechado.

Como todos os plugins visuais do Eclipse, ele faz uso do SWT para criação de componentes visuais e para capturar eventos gerados pela interação do usuário com a interface gráfica. O GEF faz uso do Draw2d para exibição de figuras no diagrama, embora não haja obrigatoriedade de se usá-lo.

4.2 Draw2d

O Draw2d é um plugin que se apóia no SWT para fornecer serviços próprios de diagramas, como por exemplo:

- Rendering - renderização de imagens, que podem ser atualizadas e marcadas como inválidas, de modo a serem posteriormente redesenhadas na tela.
- Layout - posicionamento automático dos elementos na tela de acordo com alguma regra pré-definida, independente do tamanho do editor.
- Coordinate systems - sistema de coordenadas absoluto e relativo para posicionamento dos elementos no diagrama.
- Scaling - capacidade de representar os elementos de maneira consistente e proporcional quando se faz o uso de “zoom out” ou “zoom in”.
- Hit testing - verificar se em um dado ponto existe alguma figura.
- Layers - camadas de figuras, para facilitar a representação delas, permitindo operar somente em uma dada camada, ao invés de operar em todas.
- Connections - ligações entre os elementos gráficos, conhecidos como “nodes”.
- Routing - roteamento, ou seja, definição do trajeto automático das connections, de acordo com alguma regra pré-definida.

A vantagem de se usar o plugin Draw2d para prover figuras ao GEF é que ele possui uma integração maior, provendo facilidades ao desenvolvedor que, de outro modo, ele teria que implementar sozinho.

4.3 Model View Controller

O GEF ataca o problema clássico de editar um modelo a partir de um editor gráfico. As ações do usuário na interface gráfica devem repercutir no modelo, alterando sua representação gráfica. A figura abaixo ilustra esse processo:

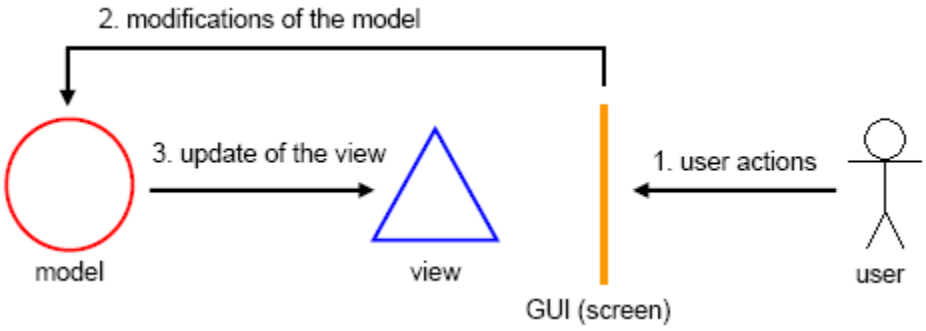


Ilustração 10 Processo do GEF

Para resolver esse problema, os desenvolvedores do GEF resolveram empregar uma arquitetura seguindo o design pattern MVC (Model – View – Controller). É importante ressaltar esse ponto, pois a arquitetura do plugin que se deseja criar para usar o GEF deve ser compatível com a arquitetura dele. A framework cuida dos detalhes de como as figuras são criadas organizadas no editor, para que as ações do usuário repercutam no modelo e assim por diante. No entanto, o desenvolvedor deverá criar todas as classes necessárias para que a framework possa operar corretamente.

4.3.1 Model

O model é o modelo que se deseja editar com o editor gráfico. Pode ser usada qualquer classe como o model. Existem, no entanto, algumas ressalvas que devem ser feitas acerca do model:

- Deve conter todas as informações importantes e que devem ser editadas pelo usuário. Se alguma coisa for ser editada via editor gráfico, ela tem que estar contida no model.
- Não deve conhecer nada sobre a view ou o controller. O model tem que ser totalmente desacoplado dos demais componentes. A framework se encarregará de fazer as ligações necessárias entre esses componentes.
- Deve implementar um mecanismo de notificação quando ocorrem mudanças em suas propriedades. Costuma-se usar o mecanismo de java.beans.PropertyChangeListener e java.beans.PropertyChangeEvent já existentes na J2SE.

4.3.2 View

A view é a representação visual do model no editor gráfico. Normalmente é uma subclasse de org.eclipse.draw2d.Figure. Deve possuir as seguintes características:

- Não conhecer nada sobre o model ou o controller. Assim como o model, esse componente não deve conhecer nada sobre os demais componentes.
- Não deve conter nenhuma informação relevante que já esteja representada no model.

4.3.3 Controller

O controller é o componente principal da framework. Deve ser obrigatoriamente uma subclasse de org.eclipse.gef.editparts.AbstractGraphicalEditPart. O controller é o componente que ouve as mudanças do model e modifica a view de acordo. Portanto, o controller deve conhecer tanto a view, como o model. O método org.eclipse.gef.EditPart.setModel(Object) é o método que configura qual model estará relacionado com o controller. O método org.eclipse.gef.editparts.AbstractGraphicalEditPart.createFigure() é o método que deve ser sobrescrito de modo a criar a view correspondente ao model. O método org.eclipse.gef.editparts.AbstractGraphicalEditPart.getContentPane deve retornar uma org.eclipse.draw2d.IFigure que representa a área da view onde serão adicionadas as views dos models filhos do model associado a esse EditPart. O método org.eclipse.gef.editparts.AbstractEditPart.refreshVisuals() é o método que deve ser chamado para modificar a view sempre que ocorrer alguma mudança no model. Outro método que vale a pena comentar a respeito é o método org.eclipse.gef.editparts.AbstractEditPart.getModelChildren() que deve ser sobrescrito para retornarem, respectivamente, os filhos do model.

É importante ressaltar que a seleção no editor está relacionado com os EditParts. Tudo que deva ser selecionado deve ter seu próprio EditPart.

4.3.4 Connections

No sistema não foram criadas connections, mas a teoria necessária para construir essa feature é praticamente a mesma para construir outras figuras no editor gráfico. As connections, que podem ser reposicionadas ou deletadas, devem também possuir um EditPart associado. Connections são tratadas da exata mesma forma que elementos comuns. Elas devem possuir um model, que pode ser de qualquer classe e deve ser retornado quando chamarmos o método org.eclipse.gef.editparts.AbstractEditPart.getModelSourceConnections() ou

org.eclipse.gef.editparts.AbstractEditPart.getModelTargetConnections() do EditPart do model pai (o método a ser chamado depende se a connection está saindo ou chegando ao model pai dela), uma view, que deve implementar as interfaces org.eclipse.draw2d.Connection e org.eclipse.draw2d.IFigure), e um controller, que deve estender a classe org.eclipse.gef.editparts.AbstractConnectionEditPart.

4.3.5 Factory

Deve-se codificar também uma classe que implemente a interface org.eclipse.gef.EditPartFactory. A finalidade dessa classe é criar uma factory que, dado um model, construa o controller correspondente.

Dessa forma, é possível concluir que para cada elemento diferente que se deseja exibir e manipular no editor gráfico é necessário implementar, pelo menos, três classes: seu model, sua view e seu controller.

4.4 Editor

É o editor propriamente dito que conterá o diagrama onde os elementos poderão ser manipulados e editados. Ele é uma subclasse de org.eclipse.gef.ui.parts.GraphicalEditor. Todo GraphicalEditor possui um org.eclipse.gef.GraphicalViewer, que pode ser obtido com o método org.eclipse.gef.ui.parts.GraphicalEditor.getGraphicalViewer(). Um GraphicalViewer é o responsável por instalar uma visualização do modelo em um SWT Control. Deve-se fornecer ao GraphicalViewer uma EditPartFactory, para criar os EditParts necessários, e um org.eclipse.gef.RootEditPart (um EditPart especial, sobre o qual todos os outros são adicionados) e um model especial chamado contents. O contents do GraphicalViewer de um GraphicalEditor é o model que representa o diagrama como um todo, que deverá conter o model de todos os elementos a serem representados no diagrama.

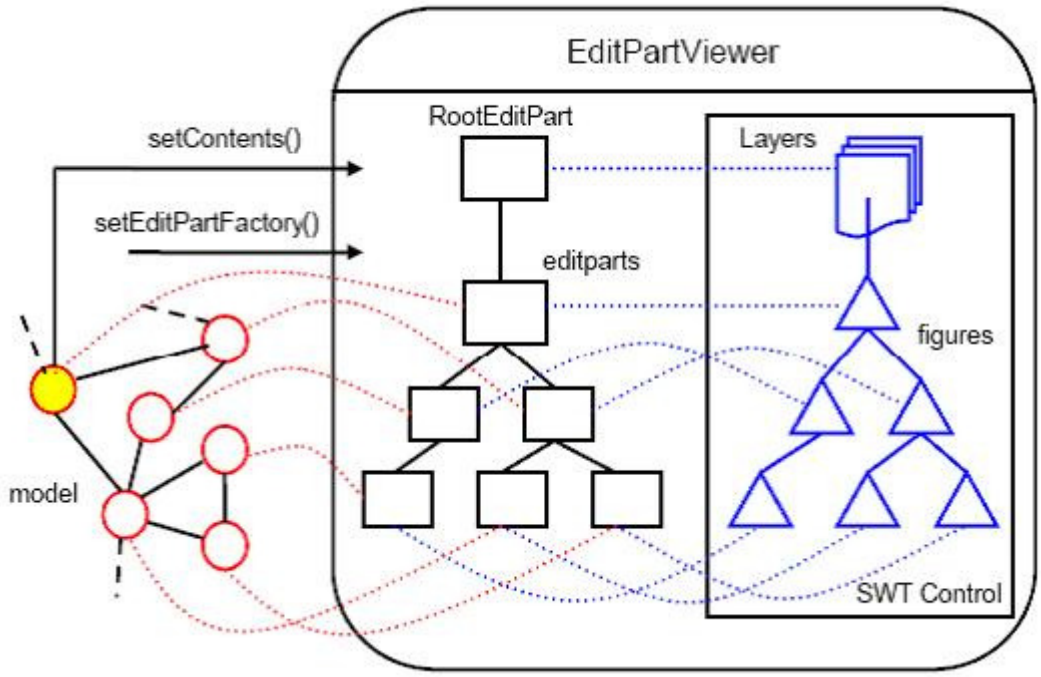


Ilustração 11 Visualização e Edição de Elementos

O processo que a framework usa para construir os elementos no diagrama é simples e recursivo. Primeiramente ela recupera o GraphicalViewer associado ao GraphicalEditor, então ela recupera o contents e procura na EditPartFactory, associada ao GraphicalViewer, um EditPart adequado para aquele model, a framework então instancia essa classe e cria a view adequada, adicionando-a ao layer primário do diagrama. A partir disso, a framework captura os models filhos e procura o EditPart adequado na EditPartFactory, intanciando-o, e criando posteriormente a view adequada e adicionando-a ao contents pane da view pai. Esse processo então se repete até que não haja mais models filhos (ou seja, o método getModelChildren() de todos os últimos models instanciados retornem nulo). O processo como um todo é ilustrado na figura a seguir:

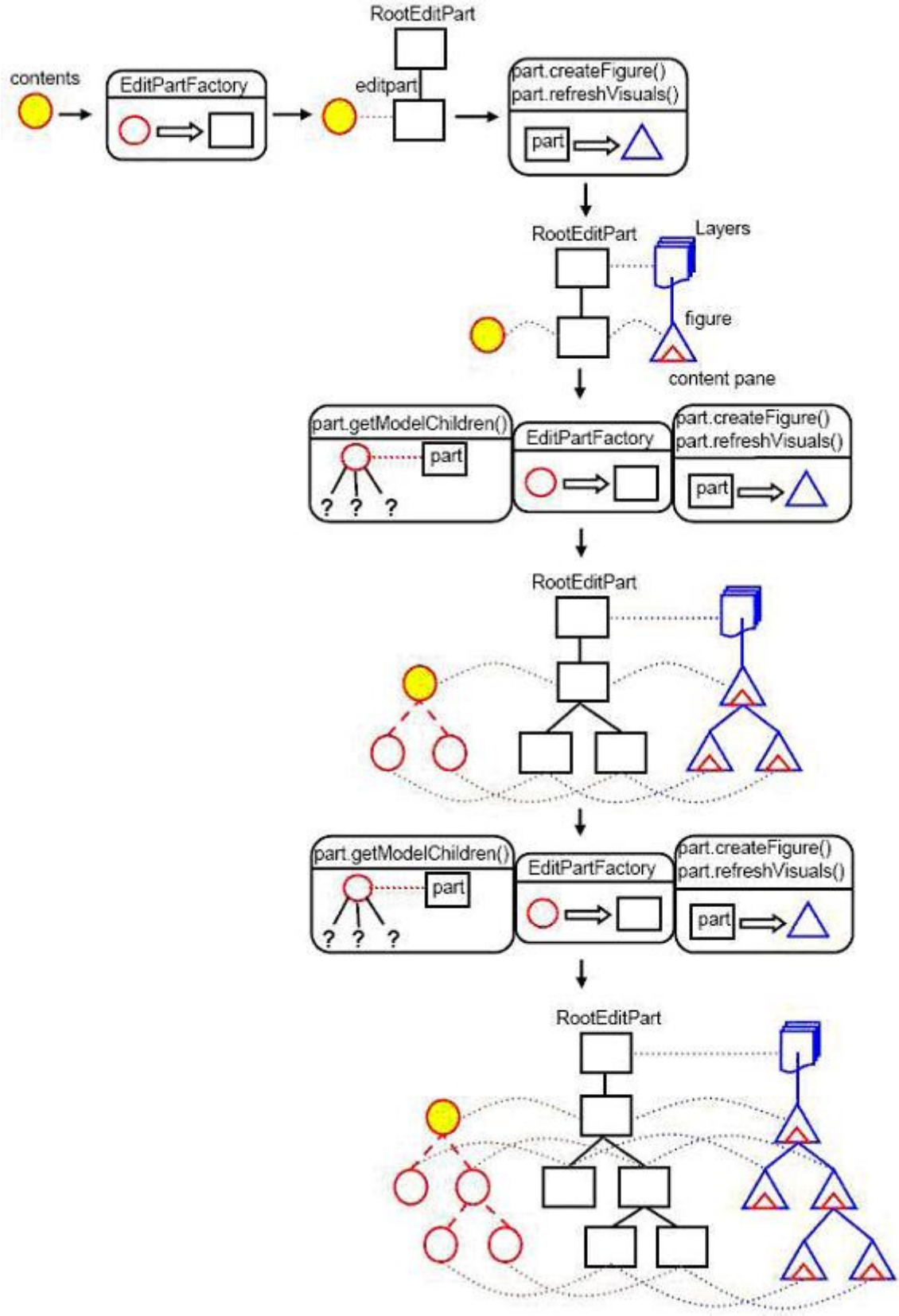


Ilustração 12 Construção de Elementos

A interação com o usuário é modelada através de dois design patterns: requests e commands. Ao clicar, mover ou manipular de alguma forma um elemento do diagrama, uma requisição é gerada para o elemento com o qual se interagiu. O EditPart desse elemento é então consultado para saber se ele sabe lidar com o request gerado. Essa informação está localizada nas edit policies (que são subclasses de org.eclipse.gef.editpolicies.AbstractEditPolicy) instaladas. Caso o EditPart saiba tratar esse request, um command (especificado na edit policy que foi capaz de tratar o request) é gerado para modificar o model adequadamente. Caso o EditPart não saiba tratar esse request, ele é repassado para o EditPart pai e o processo se repete até que se tenha chegado ao RootEditPart ou que o request, tenha sido tratado. A utilização de commands permite que facilmente se implemente undo e redo infinitos.

Capítulo 5

5 XML-RPC

5.1 Conceitos

XML-RPC é um protocolo de chamadas de procedimentos remotos, que usa XML para sua codificação e HTTP(HyperText Transfer Protocol) como mecanismo de transporte.

XML-RPC é um protocolo muito simples, já que define apenas os tipos de dados e comandos mínimos e necessários. A sua especificação pode ser impressa em apenas duas folhas de papel. O que é um grande contraste com os outros tipos de protocolos de chamadas de procedimentos remotos, onde suas especificações são gigantescas e necessitam muitas vezes de suporte de outros softwares para serem utilizados. Apesar disso, o protocolo XML-RPC permite que complexas estruturas de dados possam ser transmitidas, processadas e retornadas.

A escolha pelo padrão XML-RPC foi feita em detrimento de SOAP, CORBA, DCOM e outras tecnologias para comunicação remota, pois XML-RPC é madura, leve e consideravelmente mais simples que as mencionadas, por eliminar funcionalidades desnecessárias para este projeto.

Devido à sua simplicidade, bibliotecas clientes e servidores XML-RPC encontram-se disponíveis para as mais diversas linguagens de programação, permitindo comunicação transparente entre objetos em distintas plataformas e linguagens. O fato de utilizar HTTP simples como transporte de mensagens (chamadas) também garante alta

aplicabilidade por permitir comunicação entre nós computacionais em distintos ambientes e redes, mesmo através de firewalls.

XML-RPC é um subconjunto de SOAP e tem a vantagem significativa de ser estável e ter sua especificação "congelada", enquanto SOAP ainda é continuamente modificada. O fato de ser um subconjunto de SOAP permite também um processo de transição futuro facilitado caso surja a necessidade de comunicação com outros sistemas baseados em SOAP. Ou seja, a tecnologia XML-RPC foi escolhida por ser extensível, portável e simples manutenção.

A figura abaixo exemplifica de forma simples e resumida o protocolo XML-RPC, onde dados são codificados em um XML, o qual é enviado por HTTP para outra aplicação, que então é decodificado e transformado em dados novamente para o seu devido processamento.

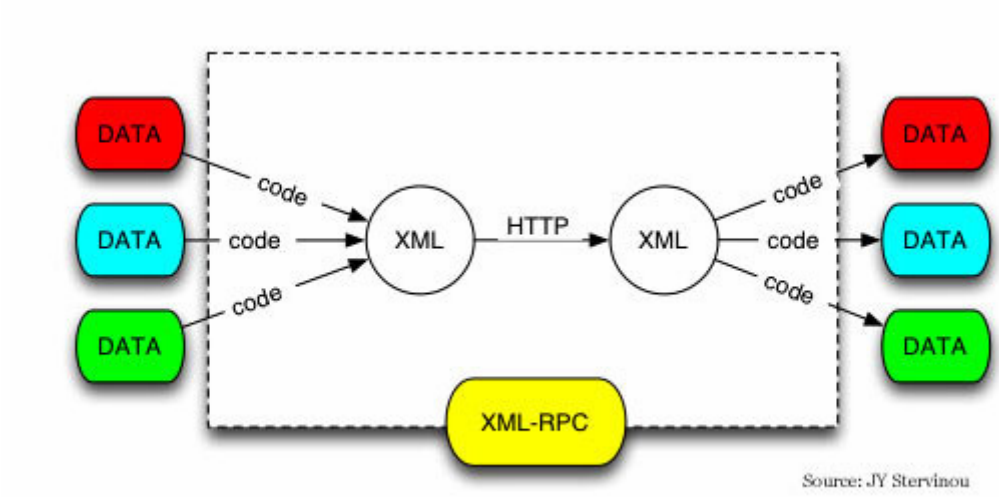


Ilustração 13 Protocolo XML-RPC

5.2 Limitações

Devido a sua grande simplicidade, o protocolo XMP-RPC possui algumas limitações, as quais estão listadas abaixo, porém, nenhuma delas implicou em uma perda de performance perceptível ao usuário e nem na inviabilidade do projeto.

- Chamada de Métodos – A chama de métodos remotos é feita utilizando-se o nome do método, o qual pode conter apenas caracteres minúsculos e maiúsculos de A a Z, caracteres numéricos de 0 a 9 e o caractere sublinhado "_".
- Estruturas de dados nomeadas – Matrizes e estruturas de dados são sempre enviadas anonimamente. Quando múltiplas estruturas de dados são enviadas simultaneamente, o programador precisa necessariamente estabelecer a ordem que estas serão enviadas para que seu processamento ocorra de maneira correta. Isto não é um grande programa, mas em algumas ocasiões nomear as estruturas poderia ser consideravelmente melhor.
- Tipos de dados – De modo a simplificar e possibilitar a comunicação entre diferentes tipos de linguagens, o protocolo XML-RPC define apenas um conjunto mínimo de tipos de dados a serem transferidos. Dessa maneira, em Java, que é uma linguagem orientada a objetos, um objeto inteiro precisa ser quebrado em seus tipos básicos para ser enviado, quando recebido é então novamente agrupado para se construir um novo objeto.
- ASCII - Em uma comunicação utilizando o protocolo XML-RPC, todas as mensagens são transferidas em um XML, utilizando o formato ASCII. Desta forma, existe um pequeno overhead, já que todos os tipos precisam ser transformados de String (ASCII) para o tipo básico correspondente. No caso de se transferir dados binários, é utilizada a codificação base 64. Logo, todos os dados binários são codificados em base 64, transferidos em um XML e depois decodificados, gerando novamente o dado binário. Portanto, se fosse necessário ao sistema, uma comunicação constante de dados binários, a performance poderia ser seriamente prejudicada por essa abordagem.

5.3 Segurança

Como já mencionado, a comunicação XML-RPC pode ocorrer mesmo através de firewalls, mas para isso, faz-se necessária a sua configuração.

Para que os servidores XML-RPC dos SLMs e TSC tenham maior segurança, recomenda-se configurar as diretivas do firewall para proteger o seu acesso. Como o acesso é realizado por meio de remoting em porta TCP, é necessário que as portas em que os servidores são disponibilizados possam ser acessados apenas pelas máquinas em que o sistema se encontra.

Portanto, a configuração do firewall deve permitir a completa comunicação exclusivamente entre os servidores do TSC e dos SLMs e não permitir o acesso (recusando conexão, ignorando pedido ou retornando erro) a todos outros endereços de rede.

Capítulo 6

6 JFreeChart

O JFreeChart é uma biblioteca para criação de gráficos (charts) dos mais variados tipos. Ele foi feito totalmente em Java e está licenciado sob a licença LGPL (Lesser GNU Public License), o que garante que o seu código é aberto e permite também que ele seja utilizado em aplicações comerciais, de código fechado.

O JFreeChart permite a criação de gráficos tipo torta (em 2D ou 3D), de espalhamento, histograma, polar, bolha, Gantt, XY ou de tempo, com um ou mais eixos de ordenadas, dentre outros.

6.1 Ponte AWT-SWT

O JFreeChart usa a biblioteca Swing para criação dos componentes gráficos desejados. Portanto, um gráfico gerado por essa biblioteca terá as aparências já conhecidas de um componente Swing, podendo ser facilmente integrado numa aplicação toda baseada nele.

Usar o JFreeChart com uma aplicação baseada no SWT ao invés do Swing requer um trabalho adicional, mas a integração é possível. Para fazê-lo, basta usar a classe org.eclipse.swt.awt.SWT_AWT, que faz uma ponte entre a biblioteca AWT (compatível com Swing) e o SWT. Essa classe possui um método estático chamado SWT_AWT.new_frame(Composite) que cria um java.awt.Frame filho, integrado ao Composite que foi passado como parâmetro. A partir disso, qualquer componente Swing que precisar ser adicionado, pode ser adicionado nesse Frame. Para esse método funcionar o

Composite passado como parâmetro deve ter sido criado usando-se a flag SWT.Embedded como style.

6.2 Criação e Configuração de Gráficos

A criação de um tipo específico de gráfico é simples: basta criar um objeto da classe java.awt.Frame, contendo um objeto da classe org.jfree.chart.ChartPanel, criado a partir de um objeto da classe org.jfree.chart.JFreeChart, que por sua vez é gerado por uma instância da classe org.jfree.chart.ChartFactory. O JFreeChart é o objeto que contém informações gerais do gráfico como título, subtítulo, rótulo dos eixos e tipo de gráfico (definido a partir do método da ChartFactory invocado para criar o JFreeChart).

Os dados que serão plotados no gráfico são obtidos a partir de uma instância da interface org.jfree.data.general.Dataset. Diversas classes implementam essa interface. Dentre elas, podemos citar a classe XYSeriesCollection, que implementa um conjunto de dados para gráficos XY.

Os gráficos XY (ou de tempo) tradicionais geralmente possuem apenas uma linha (conjunto de pontos). Embora não tenha sido implementado no sistema, é possível criar com o JFreeChart um gráfico com múltiplas linhas, indexadas cada qual por sua própria ordenada. Para isso, basta criar uma SeriesCollection, contendo cada qual uma Series. Essas SeriesCollection podem então ser mapeadas para um eixo de ordenadas, chamando os métodos setDataset e mapDatasetToRangeAxis do org.jfree.chart.plot.XYPlot associado à instância de JFreeChart.

Existe uma diferenciação natural entre o primeiro eixo criado das ordenadas (eixo primário) e os demais eixos das ordenadas. O primeiro eixo criado é obtido diretamente do XYPlot e está sempre posicionado do lado esquerdo do gráfico. Os demais eixos devem ser criados separadamente e mapeados no XYPlot, podendo ser posicionados livremente, tanto na direita quanto na esquerda do gráfico.

Para todos os eixos podem ser definidas opções como cor da linha do gráfico, cor do eixo, se o eixo irá exibir o menor intervalo de valores necessário para se exibir completamente os pontos do gráfico (auto-range), limite máximo, limite mínimo pros valores do eixo.

Para o gráfico como um todo, é possível configurar opções, como cor do background do gráfico, espaço entre as linhas do grid, cor das linhas do grid, entre outras.

No sistema, os gráficos gerados são estáticos, isto é, uma vez criados, eles não mudam mais. É possível, no entanto, acrescentar dinamicamente pontos a um gráfico, criando assim um gráfico de tempo real de uma determinada propriedade. Essa extensão poderosa pode ser feita tornando o "panel" do gráfico em um listener dos eventos de mudança de propriedade e fazendo com que o panel a cada evento propagado modifique o Series adequado. A modificação do Series irá repercutir no gráfico, que será então atualizado, contando com o novo ponto incluído.

Um gráfico gerado pode ser facilmente exportado em formato JPEG ou outros formatos conhecidos, através de opções acessíveis pelo seu menu de contexto.

Capítulo 7

7 Arquitetura

7.1 Visão Geral

O Sistema de Controle e Monitoramento Laboratorial foi projetado para ser capaz de controlar e monitorar experimentos laboratoriais compostos de diferentes tipos de instrumentos, comunicando com estes a partir de uma interface comum definida pela LECIS (Laboratory Equipment Control Interface Specification).

A arquitetura do sistema é distribuída, onde existem dois tipos de componentes básicos, o TSC e o SLM. O SLM é a abstração básica de um instrumento laboratorial qualquer, sendo responsável pelas chamadas ao hardware do equipamento em específico. O sistema pode possuir quantas instancias do tipo SLM forem necessárias, já que cada instancia deve corresponder a um instrumento físico laboratorial. Entretanto, deve possuir apenas uma instancia do TSC, ele é a interface com o usuário e o concentrador do controle e monitoramento dos SLMs.

A utilização da tecnologia Eclipse RCP permite dividir o sistema em subsistemas, onde cada um poderia ser desacoplado, respondendo por uma funcionalidade distinta. As funcionalidades estão separadas por plugins, mas todos necessitam ter como dependência, além dos plugins básicos da arquitetura RCP, dois plugins em comum para o seu interfaceamento. Eles são o “commom”, responsável pelas funções gerais do sistema e o “app”, plugin responsável pelo gerenciamento dos plugins e pontos de extensão.

Uma visão geral dos componentes básicos da arquitetura pode ser analisada na figura a seguir. Nela é possível observar como o usuário, o TSC, os SLMs e os instrumentos laboratoriais se relacionam. São apresentados apenas dois instrumentos, mas o TSC se comunica com quantos SLMs forem necessários para a experiência, não há limitações.

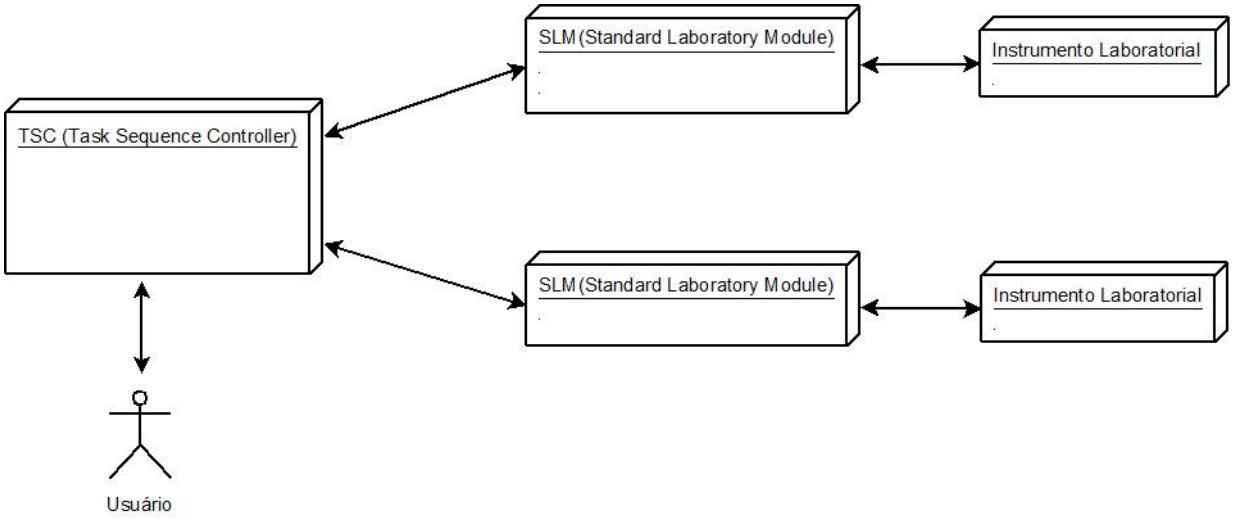


Ilustração 14 Componentes básios da arquitetura

7.2 SLM

O SLM de um instrumento laboratorial pode ser desenvolvido em qualquer linguagem de programação ou arquitetura de projeto, porém, ele deve atender as especificações da LECIS e comunicar-se através de XML-RPC com o TSC.

Os SLM são aplicações (programas de computador), que podem ser executados em linha de comando, com ou sem interface gráfica. São responsáveis basicamente pelas as abstrações necessárias para que o TSC possa controlá-lo, independentemente do tipo de equipamento e tecnologia utilizada.

7.2.1 Arquitetura básica

Apesar de cada SLM poder ter uma implementação diferente, é obrigatória a prática dos componentes da arquitetura básica definida, que está ilustrada na figura a seguir.

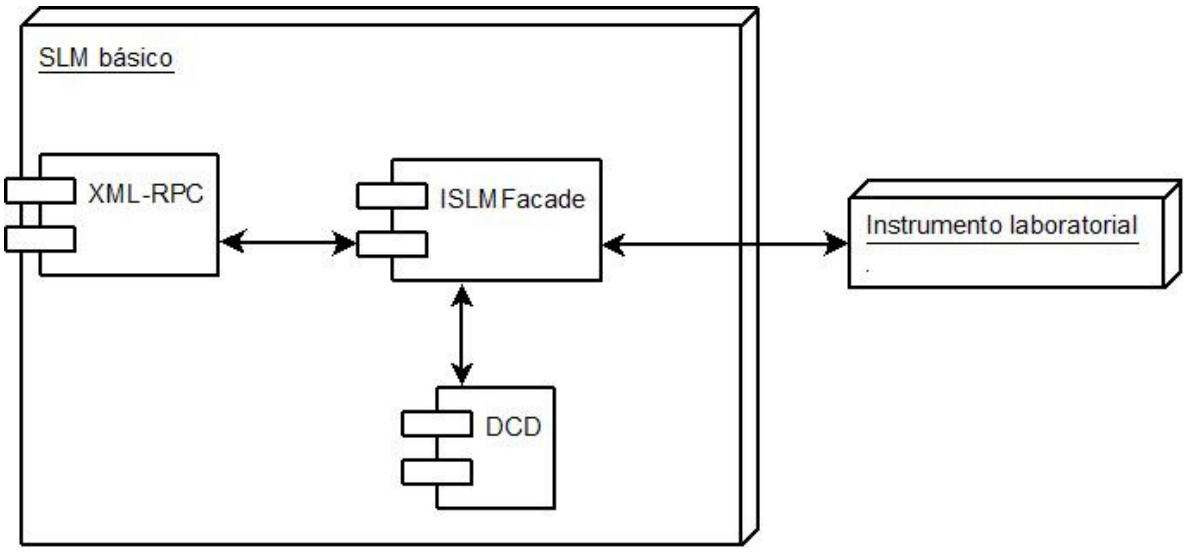


Ilustração 15 Arquitetura básica do SLM

Para que um SLM possa ser integrado ao sistema, deve possuir os componentes da arquitetura básica ilustrada na figura anterior. Estes componentes são:

- XML-RPC – Responsável por realizar chamadas remotas no TSC e de servir chamadas remotas ao ISLMFacade.
- ISLMFacade – Interface entre o TSC e o SLM, deve possuir a implementação dos métodos básicos da LECIS, além de ser também a interface entre o SLM e o instrumento laboratorial. Este componente pode ser modularizado em outros.
- DCD – Arquivo de recursos do SLM, deve possuir informações estáticas sobre o módulo.

Esta é arquitetura básica de um SLM, porém, nada impede que a implementação possua mais componentes para facilitar sua operação, ter mais iteratividade com o usuário ou mesmo, contemplar funcionalidades extras, que o TSC não possui. Como por exemplo, estes componentes extras podem ser interface gráfica, persistência de dados, geração de relatórios, entre outros.

7.2.2 Arquitetura utilizada

Para este projeto, foi produzido um framework de desenvolvimento de SLMs em Java. O framework foi desenvolvido basicamente para melhor gerenciamento e reaproveitamento de código. Para isso, foi elaborada uma arquitetura mais detalhada e mais

modularizada para a implementação de uma SLM. Separando os componentes por funcionalidades distintas. A figura a seguir ilustra tais componentes.

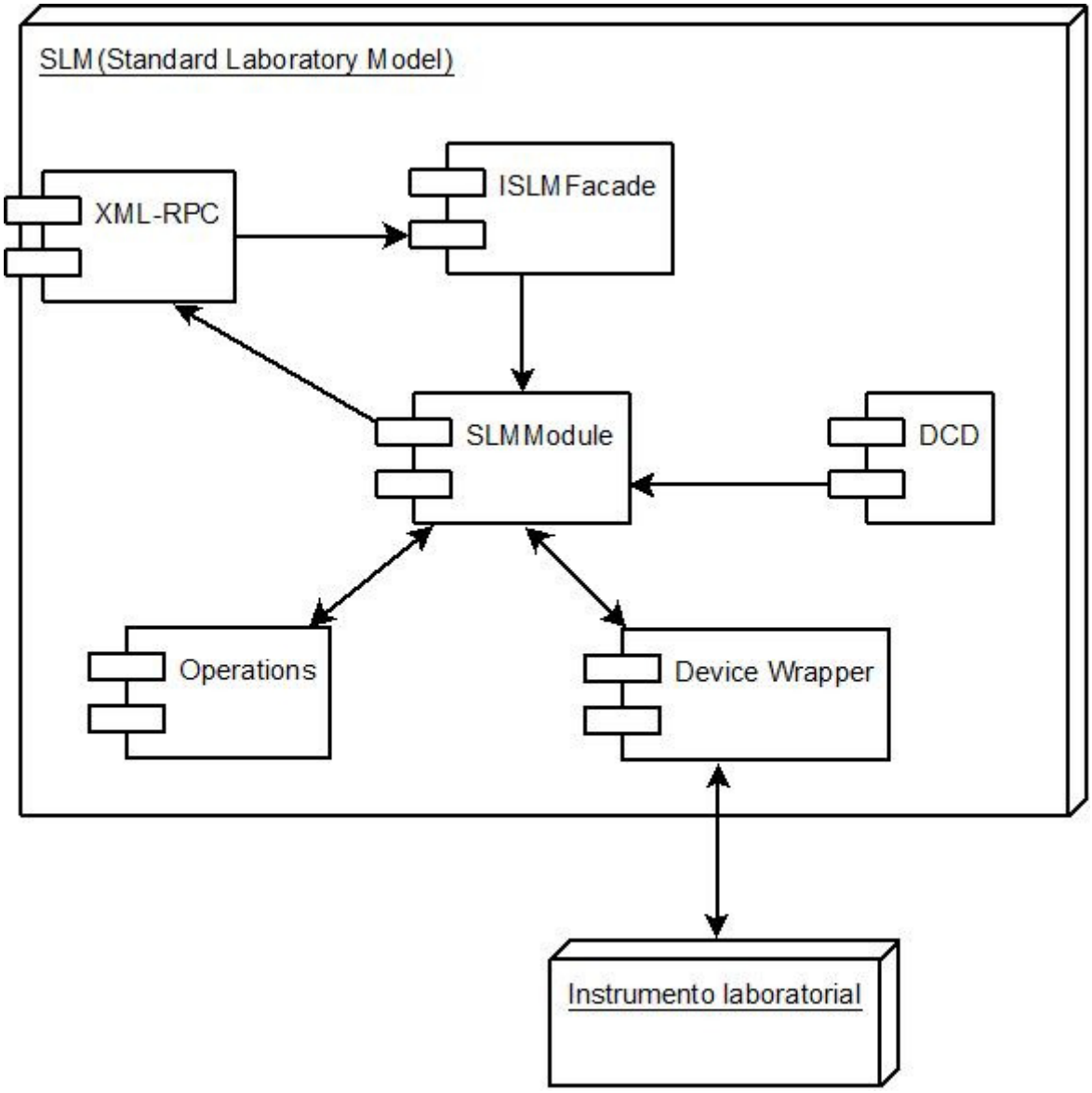


Ilustração 16 Arquitetura utilizada SLM

Os componentes da arquitetura básica ainda estão contemplados na arquitetura utilizada, que possui os seguintes componentes:

- XML-RPC – Responsável por realizar chamadas remotas no TSC e de servir chamadas remotas ao ISLMFacade.

- ISLMFacade – É apenas a interface entre o TSC e o SLM, possui os métodos básicos da LECIS, que realizam chamadas ao SLMModule para sua efetivação e validação.

- DCD – Arquivo de recursos do SLM, deve possuir informações estáticas sobre o módulo.
- SLMModule – Componente responsável por gerenciar as funcionalidades do SLM, possui as abstrações definidas pela LECIS, como a máquina de estados e a persistência e gerenciamento das variáveis de sistema (System Variables).
- Operations – São as operações específicas do módulo laboratorial. As operações são denominadas run_op pela LECIS.
- Device Wrapper – É responsável pelo encapsulamento das chamadas ao Driver ou ao hardware do instrumento, simplificando e disponibilizando as funções do instrumento necessárias à sua automação pelo sistema.

7.3 TSC

O TSC é uma aplicação Java, que tem como maior objeto interagir com os SLMs e o usuário do sistema. A maior parte das funcionalidades desta implementação do TSC não são requisitos da LECIS e não estão em seu escopo. Elas foram desenvolvidas para uma melhor interação entre o usuário e os SLMs, além de proporcionar mecanismos mais elaborados de controle e monitoração remota. Um exemplo claro disto é a tela sinótica do TSC, nela é possível monitorar os estados e variáveis de sistemas de todos os SLMs integrados.

A figura a seguir ilustra a arquitetura do TSC e seus componentes básicos:

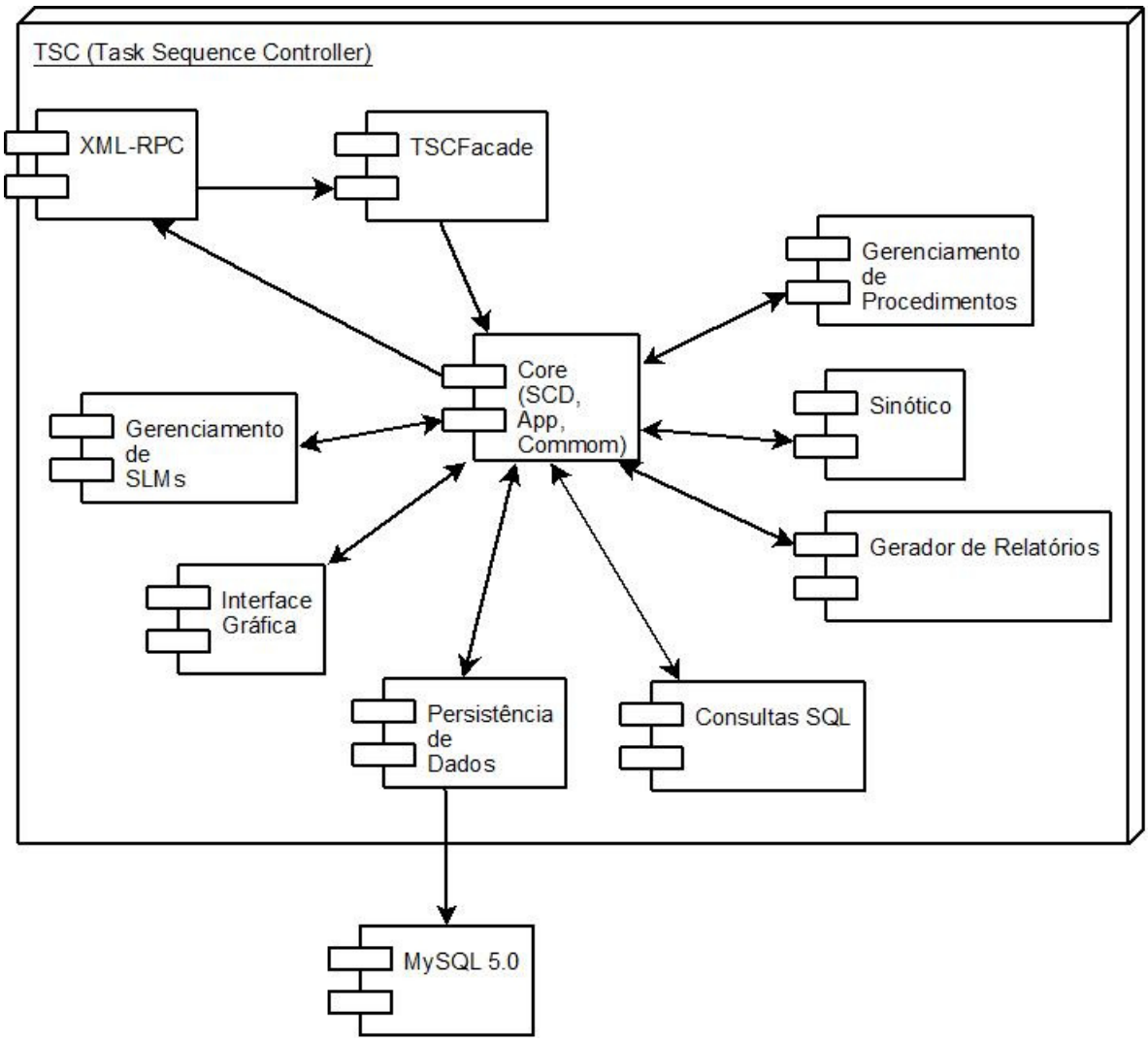


Ilustração 17 Arquitetura do TSC

- XML-RPC – Assim como no SLM, é responsável por realizar chamadas remotas, porém para todos os SLMs e de servir chamadas remotas ao TSCFacade.

- TSCFacade – É apenas a interface entre o TSC e o SLM, possui os métodos básicos da LECIS, que realizam chamadas ao core para sua efetivação e validação.
- Core – É o núcleo do TSC, composto pelas definições do SCD, definidas pela LECIS, App, que realiza o gerenciamento dos plugins e pontos de extensão conectados ao TSC, e o Commom, que é um plugin global a todos os componentes do TSC, com classes singleton, interfaces e classes abstratas para a integração de todos os componentes.
- Gerenciamento de SLMs – É o componente responsável pela integração de novos SLMs ao TSC e também por armazenar as informações básicas dos módulos já integrados, como o DCD e o endereço XML-RPC.

- Interface gráfica – É a interface entre o usuário e o TSC, todos os componentes da arquitetura podem contribuir a este através do Core.
- Persistência de dados – É responsável pela interface do sistema com o banco de dados, realizando a persistência dos dados do sistema, como dados históricos de processos, entre outros.
- Consultas SQL – É um plugin que permite ao usuário consultar diretamente os dados do banco de dados através de rotinas SQL.
- Gerador de relatórios – Gera ao usuário relatórios de dados históricos de TAGs dos SLMs através de consultas à base de dados.
- Gerenciamento de procedimentos – Este componente é responsável por criar, executar e monitorar procedimentos. A partir do DCD de cada SLM é possível obter seus comandos disponíveis, e deste modo, criar procedimentos que englobam tais comandos. Os procedimentos podem ser complexos, com fluxos de controle através de valores fixos ou de valores das variáveis de sistema.
- Sinótico – Provê ao usuário telas sinóticas, onde o usuário pode incluir quais SLMs disponíveis deseja monitorar. Através do sinótico o usuário pode executar procedimentos já criados e alterar os valores de variáveis de sistema.

A implementação dos componentes acima descritos será detalhada no capítulo especifico sobre o TSC, já que aqui está sendo explanada apenas a arquitetura desenvolvida para o sistema.

Capítulo 8

8 SLM (Standard Laboratory Model)

Como já dito no capítulo sobre a LECIS, o SLM (Standard Laboratory Model) é a abstração básica para os equipamentos laboratoriais, provendo comandos e métodos para o controle e monitoramento de cada instrumento. Quando integrado ao TSC, o SLM fornece o seu DCD (Device Capability Dataset), arquivo do tipo XML que contém todas as informações necessárias para a sua utilização, como quais comandos estão disponíveis, tipos de eventos que serão enviados e quais variáveis de sistema o módulo possui. Portanto, cada SLM é responsável por processar comandos e coletar dados do instrumento automatizado, passando-os para o TSC, que fará seu armazenamento e processamento subseqüente.

8.1 Implementação

Neste projeto, foi realizada uma implementação básica do SLM, escrito em Java, gerando um framework que pode ser facilmente utilizado e estendido para a implementação de módulos de instrumentos laboratoriais gerais. Esse framework possui a implementação geral para quaisquer tipos de módulos laboratoriais baseados na LECIS, como o controle de estados, gerenciamento de variáveis de sistema, métodos de interface bem definidos, manipulação do arquivo DCD, entre outros.

Para a correta integração com o TSC implementado para este projeto, toda implementação de um SLM deverá possuir obrigatoriamente uma classe SLMFacade, que deverá ser a implementação da interface ISLMFacade, essa classe deverá também, estar disponível para receber e enviar chamadas do tipo XML-RPC. A interface pode ser vista no Javadoc.

Cada implementação de um SLM deverá também ser responsável por gerenciar as chamadas ao hardware do instrumento, ou seja, cada SLM deverá possuir um encapsulador específico do driver do instrumento laboratorial. A implementação do encapsulador, ou "Device Wrapper", deverá ser específica e única para cada tipo de instrumento laboratorial, dispondo-se em um projeto à parte do SLM, gerando assim, uma biblioteca que deverá ser referenciada pelos projetos que a utilizam.

8.2 Conformidade com a LECIS

A implementação do framework de desenvolvimento de SLMs em Java, assim como todos os SLMs implementados para este projeto, seguiram a versão da LECIS CORBA OMG, porém, a adaptando para a comunicação XML-RPC. Isso acarretou na mudança do número de parâmetros dos métodos previamente definidos pela LECIS, já que para a comunicação XML-RPC, o número de parâmetros e o nome dos métodos precisam ser bem definidos e conhecidos pela parte que enviará os comandos. Os parâmetros que precisaram ser estendidos foram encapsulados em outro XML, podendo assim, ter um número necessário de parâmetros para cada chamada, sem a necessidade de criar outros métodos.

8.2.1 Implementação Multithreaded

Em conformidade com a LECIS, a implementação dos SLMs foi feita utilizando-se de técnicas Multithreaded. Ou seja, cada SLM pode processar, em paralelo, chamadas simultâneas do TSC. Porém, para isso, cada SLM possui um mecanismo de validação das requisições, para que chamadas que necessitam ser executadas seqüencialmente não sejam executadas em paralelo. Normalmente, chamadas que precisam de acesso ao hardware do instrumento precisam ser executadas em seqüência e chamadas de requerimento de status e dados podem ser executadas em paralelo.

8.2.2 Máquina de Estados

Os SLMs desenvolvidos para este projeto possuem a mesma implementação da máquina de estados da LECIS, que foi desenvolvida no framework acima descrito. A máquina de estados faz a correta transição de estados nos métodos primários de controle do SLM.

A máquina de estados está implementada mais especificamente na classe abstrata AbstractSLMFacade, essa classe implementa a interface ISLMFacade, ou seja, já possui todos

os métodos a serem implementados por qualquer SLM. A classe faz a validação do estado corrente na chamada dos métodos implementados da interface, depois disso faz a chamada à implementação específica de cada SLM, fazendo a correta transição de estados, quando necessário. A implementação específica de cada SLM deverá ser feita na classe concreta que estende a classe abstrata SLMModule, essa classe abstrata possui métodos abstratos a serem implementados para a execução específica de cada SLM durante as chamadas dos métodos da interface ISLMFacade. Exemplificando, o TSC faz uma chamada ao SLM através dos métodos da classe SLMFacade, que então valida o estado corrente e faz as suas chamadas específicas, controlando seu estado sempre que necessário.

A abordagem de uma implementação geral para a máquina de estados em um framework foi realiza para o maior aproveitamento de código, já que todos os SLMs que seguirem a LECIS deverão ter o mesmo controle. Caso necessário, também é possível implementar outras máquinas de estados para os SLMs, sem que seja necessário modificar o TSC, já que ele faz apenas o gerenciamento do envio de comandos descritos no DCD de cada SLM, abstraindo assim, o controle de estados, ou seja, cada SLM deve ser responsável pelo seu próprio controle de estados. Nada impede também de se estender a máquina de estados já implementada, sobrescrevendo algum método, criando assim, um comportamento específico e diferente para o SLM desejado.

8.2.3 Envio de eventos ao TSC

Segundo a LECIS, cada SLM deve poder enviar, assincronamente, eventos ao TSC, informando-o de transições de estados, dados de processos, status, atualizações de valores das variáveis de sistema, entre outros.

Para o envio de eventos, cada SLM deverá possuir uma implementação de chamada ao método slm_event da classe TSCFacade por comunicação XML-RPC. Para módulos escritos em Java, essa implementação já está realizada no framework disponibilizado, mais especificamente na classe TSCClientUtil. Cada evento possui parâmetros de validação e parâmetros específicos, que são passados como um array nessa implementação, mas na chamada ao slm_event do TSCFacade, deverão ser encapsulados e enviados em um XML.

8.2.4 Validação do remetente de chamadas

Originalmente na LECIS não há alguma preocupação com a segurança da rede e validações extras. Para aumentar a confiabilidade e a segurança no mecanismo de troca de mensagens, foi incorporado o endereço do remetente como parâmetro de validação em todas as mensagens trocadas entre os SLMs e o TSC. Dessa forma, os SLMs devem sempre validar as mensagens recebidas com os endereços inicialmente enviados na inicialização de suas comunicações. O endereço a ser verificado é o endereço de retorno ao TSC (TSC Callback).

8.2.5 Tratamento de erros

Todas as exceções tratadas em um SLM são do tipo SLMException, as exceções podem ter os seguintes níveis:

- UNDEFINED – quando a exceção não possui origem definida.
- INFO – quando o erro ocorrido não é grave e a operação pode continuar.
- FINE – quando o erro ocorrido não é grave, não impactando no estado do módulo, mas a operação em questão deve ser abortada e comunicada ao usuário ou registrada no sistema.
- SEVERE – quando o erro ocorrido é grave e o módulo deve ser levado ao estado de ERROR.

8.2.6 Construção do DCD

A construção do DCD de um SLM é um dos passos mais importantes no seu desenvolvimento. O DCD pode ser escrito em um arquivo texto do tipo XML, que pode ser editado como texto plano ou por editores específicos de XML. No caso, foi utilizado um editor específico de arquivos XML, já que os mesmos já possuem ferramentas automáticas de validação e indentação, que facilitam no desenvolvimento.

O arquivo Schema XML original da LECIS, que descreve os elementos do arquivo XML do DCD não foi modificado para a implementação dos SLMs. Porém, não foi utilizado todo o seu nível de detalhamento, alguns de seus elementos são apenas apresentados aos usuários, não interferindo na camada de controle.

Os elementos básicos que o DCD de um SLM deve possuir para o sua correta integração com o TSC são:

- SLM_ID – É a identificação do SLM, deve ser único para cada um. Mesmo se em uma experiência conter mais de um SLM do mesmo tipo, cada SLM deverá possuir um arquivo DCD com um SLM_ID único.
- ADMINISTRATIVE – Contém todas as informações administrativas do SLM, como nome, protocolo de comunicação, número de série, número do modelo, nome do fabricante, versão do firmware, versão do DCD, versão do software do SLM, entre outras. No TSC essas informações são apenas apresentadas ao usuário, não podendo ser editadas e sem efeito sobre o controle do SLM pelo TSC.
- FUNCTIONALITY – Possui informações sobre a funcionalidade do SLM, assim como o ADMINISTRATIVE, esse elemento também não possui efeito sobre as funcionalidades do SLM, serve apenas como informação ao usuário.
- PHYSICAL_CHARACTERISTICS – Possui informações sobre as características físicas do instrumento, como altura, largura, comprimento e peso. Essas características são apenas apresentadas ao usuário, também não possuem efeitos sobre o TSC ou o SLM.
- PRIMARY_COMMANDS – Este elemento é um dos mais importantes do DCD, nele são listados e descritos todos os comandos que o SLM pode executar, mesmo aqueles obrigatórios pela LECIS. Este elemento será mais detalhado nos tópicos adiante.
- SYSTEM_VARIABLES – Este elemento possui as informações sobre as variáveis de sistema do SLM. É a partir dele que o TSC consegue obter informações de como manipulá-las, obtendo seu nome, descrição e tipo. Este elemento também será mais detalhado nos tópicos adiante.

Os demais elementos, como MACROS, não foram utilizados na implementação do SLM e do TSC, já que possuem abstrações de mais baixo nível ou utilizam sub-unidades, que não foram contempladas neste projeto.

8.2.6.1 Comandos primários

No arquivo DCD de um SLM, cada comando primário deverá ser representado por um elemento do tipo PRIMARY_COMMANDS. A partir desses elementos o TSC conhecerá os comandos disponíveis para o SLM, sabendo a quantidade e os tipos de parâmetros de cada comando. Os comandos são listados em categorias ao usuário para a criação de procedimentos.

Para uma melhor especificação, cada elemento PRIMARY_COMMANDS possui sub-elementos para descrever os comandos. Os elementos contidos no elemento PRIMARY_COMMANDS são os seguintes:

- COMMAND_ID – É a identificação do comando. Cada comando deve possuir uma identificação única em um mesmo SLM.
- NAME – É o nome do comando a ser utilizado internamente na chamada remota.
- ALIAS – É o nome do comando a ser apresentado ao usuário.
- DURATION – É o tempo médio aproximado de duração de execução do comando. É também um elemento opcional.
- CATEGORY – É a categoria do comando.
- TYPE – Descreve o tipo de comando, podendo ser do tipo atômico, por exemplo.
- DESCRIPTION – É a descrição do comando, tem como objetivo detalhar o comando ao usuário.
- FORMAL_ARGUMENTS – São os argumentos do comando, este elemento pode ter múltiplas ocorrências e deverá listar todos os parâmetros para o comando. Caso o valor do argumento for "null", ele será enviado como vazio, senão será enviado o valor atribuído a ele. Cada elemento do tipo FORMAL_ARGUMENTS possui os seguintes elementos:
    - NAME – Nome do argumento. Se este elemento tiver o valor "ignored", ele não será apresentado ao usuário, ou seja, não poderá ser editado e será sempre enviado com o valor do elemento DEFAULT_VALUE. Esta abordagem foi utilizada para que os comandos pudessem ser pré-configurados no arquivo DCD, limitando as opções do usuário.
    - ARGUMENT_TYPE – Tipo do valor do argumento.
    - DEFAULT_VALUE – Valor padrão, que será apresentado ao usuário como escolha padrão do valor do argumento. O usuário poderá editar esse valor caso o nome do argumento não seja "ignored".
    - TRANSFER_TYPE – Indica o tipo de transferência do argumento.
    - DESCRIPTION – É uma breve descrição do argumento que será apresentado ao usuário em uma tooltip.

- PROPERTIES – São propriedades específicas desta implementação do TSC, que foram adaptações necessárias à LECIS CORBA OMG para a comunicação XML-RPC. E deverão possuir os seguintes tipos:
  - GROUP_ID – Deve possuir obrigatoriamente valores numéricos, indicando a ordem de agrupamento dos argumentos.
  - SERIALIZE – Podem ser atribuídos os valores XML ou NO. No caso do valor XML, todos os argumentos de mesmo GROUP_ID serão encapsulados em apenas um, sendo codificados em um XML. No caso do valor NO, eles serão enviados separadamente.

8.2.6.2 Variáveis de Sistemas

Cada variável de sistema de um SLM deve estar descrita em seu arquivo DCD, sendo representado por um elemento do tipo SYSTEM_VARIABLES. A partir desses elementos o TSC conhecerá todas as variáveis de sistema do SLM, podendo requisitar seus nomes, descrições e valores. A partir destas informações o TSC disponibiliza para o usuário as variáveis de sistema de cada SLM em uma tela sinótica, além de também poderem ser utilizadas na formulação de procedimentos complexos.

No DCD, as variáveis de sistema devem possuir os seguintes elementos:

- VARIABLE_ID – É a identificação da variável de sistema, e deve ser única em cada SLM.
- DESCRIPTION – É a descrição da variável de sistema, tendo como única finalidade informar ao usuário sobre a mesma.
- DATA_TYPE – É o tipo de variável de sistema, porém, como adaptação à comunicação XML-RPC, o único tipo aceitável é a STRING. Os tipos das variáveis podem se diferenciar internamente na implementação específica de cada SLM.

8.3 Divisão de projetos e pacotes

Para uma melhor organização, um projeto Java de um SLM deve ser dividido em pacotes e deve conter apenas sua implementação específica. Neste projeto, a implementação comum a todos os SLMs está no projeto Java “common”.

No pacote principal de um SLM devem estar as implementações das seguintes classes:

- AbstractSLMFacade – Classe abstrata que estende a interface ISLMFacade, que possui todos os métodos de acesso do TSC ao SLM. Essa classe que contem a implementação geral da máquina de estados para todos os SLM. Na implementação desta classe abstrata deverá ser indicada a implementação escolhida do módulo.
- SLMModule – Classe abstrata que faz a representação do instrumento físico. Na implementação desta classe abstrata deverá ser escrito o comportamento específico do SLM em cada transição de estados e requerimento de informações.
- AbstractSLMServer – Classe abstrata responsável por receber chamadas XML-RPC do TSC. Na implementação desta classe deverá ser estabelecida a porta em que serão escutadas as chamadas XML-RPC, assim como, qual classe servirá como façade.

Cada projeto de um SLM também deverá possuir um pacote para a implementação de comandos específicos (run_op). Neste pacote deverão estar contidas as implementações das seguintes classes ou interfaces:

- Operation – É uma classe abstrata que já possui todos os mecanismos necessários para a execução de um comando específico, cada operação do tipo run_op disponível no SLM deverá ter uma classe correspondente que implementa esta classe abstrata.
- IOperationFactory – Esta interface possui o método getOperation, a implementação deste método deverá escolher e fabricar a operação específica (Operation) e correta a ser executada.

Cada SLM também poderá ter um projeto ou um pacote específico para a comunicação com o hardware do instrumento, no caso de um mesmo driver ser utilizado para outros instrumentos laboratoriais ou mesmo outros SLMs, recomenda-se a criação de um projeto a parte, onde todos os projeto que o utilizarem deverão tê-lo como referência.

Cada SLM poderá ter também plugins correspondentes para o TSC, no caso de se desejar criar maneiras de se plotar gráficos de resultados específicos do SLM. Como por exemplo, o resultado de uma amostra de um osciloscópio ou de um espectrofotômetro. Para isso deverá ser criado um novo projeto no Eclipse para o plugin, que deverá ter a

implementação do ponto de extensão correto. Esse passo a passo está descrito mais detalhadamente no capítulo do TSC.

Outro plugin do SLM que pode ser desenvolvido para o TSC é o plugin responsável por gerar gráficos históricosz de valores de variáveis de sistema ou TAGs. O TSC consegue, sem a necessidade de um plugin específico, plotar gráficos de resultados históricos gerais, mas no caso da necessidade de um comportamento específico, pode-se criar gráficos diferentes. Para isso também se deve criar um novo projeto no Eclipse para o plugin, implementando o ponto de extensão correspondente. Esse passo a passo também está detalhado no capítulo do TSC.

8.4 Desenvolvendo um novo SLM em Java

Como já dito anteriormente, Para se desenvolver um novo SLM em Java, que poderá ser integrado e controlado pelo TSC, pode-se utilizar o framework já implementado para os SLM deste projeto, ou simplesmente seguir as definições da LECIS e os tópicos escritos acima.

Na escolha de se implementar um novo SLM em Java, utilizando-se das classes abstratas e interfaces Java do framework, deve-se seguir o passo a passo que encontra-se em apêndice à esta documentação.

8.5 Desenvolvimento de módulos em outras Linguagens

Apesar de os SLMs desenvolvidos para esse projeto terem sido escritos em Java, é possível integrar ao sistema SLMs escritos em qualquer linguagem que tenha suporte à comunicação XML-RPC. Entretanto, para isso, os SLMs devem seguir os requisitos descritos nos tópicos acima, assim como também, as definições da LECIS.

8.5.1 TCL

Para uma prova de conceito e testes, foi desenvolvido um SLM escrito em TCL, que faz apenas operações matemáticas, sem associação a um instrumento físico.

Para a troca de mensagens XML-RPC foram utilizadas as bibliotecas TclHttpd 3.5.1 e TclSoap 1.6.8. A biblioteca TclHttpd serve para se criar um servidor XML-RPC, que servirá métodos exportados pela biblioteca TclSoap. A biblioteca TclSoap foi utilizada também para executar chamadas XML-RPC, servindo como um cliente.

Os arquivos desenvolvidos para o módulo escrito em TCL estão em anexo à esta documentação.

Capítulo 9

9 TSC (Task Sequence Controller)

Neste capítulo serão detalhados a implementação do TSC e seus componentes, que formam subsistemas, alguns deles não definidos pela LECIS. Já que a LECIS especifica apenas as formas de controle e interface com os módulos laboratoriais. Desta forma, não padroniza a interface gráfica com o usuário, persistência de dados, entre outros.

Como definido pela LECIS, o TSC é o controlador do sistema automatizado, o qual pode controlar um ou mais SLMs. Através do TSC é possível controlar qualquer tipo de instrumento laboratorial, já que ele se comunica com os SLMs a partir de uma interface padrão e executa apenas as funções definidas nos arquivos de recursos (DCD) disponibilizados por cada SLM.

9.1 Implementação

O TSC é um aplicativo Eclipse RCP, logo, escrito na linguagem de programação Java. Suas funcionalidades estão divididas em plugins acoplados apenas por um plugin comum, chamado de “common”. Toda aplicação Eclipse RCP deve possuir um plugin central que realiza o gerenciamento dos demais plugins. Neste projeto o plugin central foi chamado de “app”.

Por ser uma aplicação Eclipse RCP, a interface gráfica do TSC foi desenvolvida em cima do plugin SWT/JFace, que fornece as abstrações necessárias para a construção de interfaces gráficas com usuário de alto nível de desempenho e nativas ao sistema operacional hospedeiro.

Os plugins utilizados para o desenvolvimento do TSC, assim como os pontos de extensão disponibilizados para a criação de plugins adicionais, serão detalhados e definidos nos capítulos de cada subsistema.

9.2 Conformidade com a LECIS

A LECIS é muito sucinta em sua especificação no que diz respeito à implementação do TSC, definindo apenas a interface com os SLMs. Ou seja, para que o TSC esteja conforme com a LECIS basta que sejam enviados comandos e recebidos eventos conforme a sua definição.

9.2.1 Implementação Multithreaded

Em conformidade com a LECIS, a implementação do TSC foi feita utilizando-se de técnicas Multithreaded. Desta forma o TSC pode receber e tratar inúmeros eventos dos SLM simultaneamente. Além de conseguir também enviar comandos a um SLM, monitorando sua execução, sem causar travamentos ao usuário e às outras funcionalidades.

9.2.2 Recebimento de eventos

Para o servidor XML-RPC foram utilizadas as classes do pacote org.apache.xmlrpc, que é uma implementação da especificação internacional XML-RPC. Este servidor foi utilizado para o recebimento de eventos dos SLMs, já que possui como opção a sua utilização multithreaded, podendo-se configurar se serão utilizadas desde uma até infinitas threads.

A classe servidor de chamadas XML-RPC é a classe TSCFacade, sendo a única e exclusiva responsável por receber e tratar os eventos lançados pelos módulos. Durante toda a execução do TSC esta classe fica servindo as chamadas XML-RPC em uma determinada porta pré-estabelecida.

Os eventos são tratados cada um de uma forma, podendo-se propagar outros eventos internamente ao TSC. Os eventos que repercutem alterações à interface gráfica devem ser encapsuladas por um método do plugin SWT (Display.syncExec ou Display.asyncExec) para que estas chamadas tenham um método de acesso bem definido, sendo síncrono ou assíncrono.

A classe TSCFacade, em conformidade com a LECIS, possui o slm_event como único método disponível à chamadas dos SLM. Entretanto, como o protocolo de comunicação XML-RPC utiliza apenas tipos básicos, todos os parâmetros deste método foram definidos como do tipo String, para serem posteriormente validados e tratados. Os parâmetros do método slm_event a serem utilizados pelos módulos laboratoriais são os seguintes:

- slm_id – É o número de identificação do módulo que está enviando o evento. O evento é automaticamente descartado e não é tratado caso o TSC não possua o SLM de slm_id correspondente em sua lista de módulos integrados.
- unit_id – Em conformidade com a LECIS o método slm_event possui este parâmetro, porém, neste projeto não foram consideradas as subunidades dos módulos.
- event_id – É o número de identificação do evento enviado, é utilizado para o registro do evento enviado.
- event_type – É o tipo de evento enviado, a partir deste parâmetro que o TSC fará o tratamento correto do evento. A este parâmetro deve-se atribuir um dos valores da estrutura EEVENTCATEGORY da LECIS, caso contrário o TSC registrará um erro e não fará o tratamento do evento.
- priority – É o número correspondente da prioridade do evento, quando maior este número, menor será a sua prioridade. Sendo 1 a prioridade máxima.
- slm_result – Corresponde à estrutura SLMRESULT da LECIS, informando ao TSC o estado atual do SLM, além de atualizar o TSC com códigos de retorno e mensagens de erro. É a partir deste parâmetro que o TSC atualiza o estado atual de cada módulo no sinótico correspondente. A estrutura SLMRESULT deve ser codificada e decodificada para o formato String pelas classes do pacote javax.xml.blind. Este processo será detalhado mais adiante.
- args – São os argumentos do tipo de evento enviado. A quantidade de parâmetros varia de acordo com o tipo de evento enviado. Porém, ele é sempre encapsulado em um XML no formato definido pelos métodos da classe XmlStringUtil, a qual será detalhada mais adiante.

Para registro histórico e tratamento de eventuais erros no sistema, todos os eventos recebidos são armazenados no banco de dados, salvando-se todos os parâmetros recebidos pelo método slm_event.

Como descrito anteriormente, os tipos de eventos que o TSC pode tratar estão definidos pela LECIS na estrutura EEVENTCATEGORY. Na tabela a seguir, estão listados os tipos de eventos, seus parâmetros e o tratamento implementado neste projeto. Se o evento enviado não possuir a quantidade certa de parâmetros para o tipo de evento correspondente ou se eles não forem enviados na ordem correta, uma mensagem de erro é gerada pelo TSC.

Tabela 3 Tratamento de eventos pelo TSC

Tipo de Evento	Ordem dos parâmetros	Descrição dos parâmetros	Tratamento
ALARM	0	Mensagem de alarme	A mensagem especificada é apresentada ao usuário em uma caixa de diálogo de erro
MESSAGE	0	Mensagem de informação	A mensagem especificada é apresentada ao usuário em uma caixa de diálogo de informação
DATA_DIRECT	0	Número de identificação da operação	O resultado da operação é apresentado ao usuário na view de resultados
	1	Título do procedimento da operação	
	2	Resultado da operação	
DATA_LINK / OPERATION	0	Não é utilizado, pode ser atribuído qualquer valor	A operação especificada é marcada como concluída na tabela de operações do TSC
	1	O seu valor deve ser sempre OPERATION	
	2	É o número de	

		identificação da operação	
DATA_LINK / DATA BASE	0	É o valor da TAG a ser atualizada no banco de dados	A TAG de número de identificação especificado é atualizada no banco de dados com o seu novo valor
	1	O seu valor deve ser sempre DB	
	2	É o número de identificação da TAG que será atualizada no banco de dados	
SYS_VAR	0	Identificação da variável de sistema	Atualiza os valores das variáveis de sistema dos módulos integrados ao TSC. Atualizando também os sinóticos em que estes módulos estão inseridos
	1	O novo valor da variável de sistema	
CONTROL STATE CHANGE		Não possui parâmetros	Atualiza o estado do módulo a partir da estrutura SLMRESULT, atualizando também os sinóticos em que o SLM correspondente está inserido

Os eventos DATA_LINK / FILE e DEVICE_STATE_CHANGE não foram implementados por não apresentarem necessidade ao sistema desenvolvido.

Como dito anteriormente, o argumento args do método slm_event é na verdade vários argumentos condensados em um XML, que foi uma solução encontrada para a utilização do protocolo XML-RPC para o recebimento de eventos dos SLMs. A codificação e

decodificação do XML é feita pela classe XmlStringUtil do plugin common. O XML a ser formado possui o elemento raiz “parameters” e um ou mais elementos do tipo “param”, que possui como atributos um “id”, que pode ser a ordem do parâmetro, e um “value”, que corresponde ao valor do parâmetro.

O seguinte exemplo demonstra o parâmetro args enviado para um evento de MESSAGE:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<parameters>
        < param id=”0” value=”Mensagem para o usuário” />
</parameters>
```

A codificação e decodificação do SLMRESULT também são feitas em XML, porém, esse processo é realizado pelas classes do pacote javax.xml.bind. As classes realizam a codificação a partir das propriedades da estrutura. Um exemplo pode ser observado a seguir:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SLM_RESULT message=”Não é possível inicializar, estado incorreto"
main_state="NORMAL_OP" result_code="MAIN_STATE_INCORRECT"/>
```

9.2.3 Envio de comandos

O TSC pode enviar comandos aos SLMs a partir da interface comum definida pela LECIS como SLMFacade. A classe do TSC responsável pelo envio de comandos é a SLMClient, localiza no plugin common. Esta classe utiliza a biblioteca Apache XML-RPC, que é uma implementação da especificação internacional XML-RPC. O comando é executado pelo SLM, que retorna um SLMRESULT. Caso necessário, o SLM poderá enviar eventos ao TSC durante a execução dos comandos, atualizando-o com informações adicionais.

9.3 Subsistemas

O TSC foi dividido em subsistemas, onde cada um é responsável por funcionalidades distintas. Os subsistemas podem interagir entre si através do plugin common ou do plugin app.

9.3.1 Configurações

As configurações do TSC foram desenvolvidas no plugin app, utilizando o ponto de extensão org.eclipse.ui.preferencePages. O ponto de extensão provê um novo item na janela de preferências do sistema, onde que cada item pode possuir outros subitens. Para cada elemento do ponto de extensão são atribuídas páginas de configuração que devem seguir uma interface comum. Os elementos deste ponto de extensão devem possuir as seguintes propriedades:

- Id – É a identificação do elemento do ponto de extensão.
- Name – É o nome de apresentação do elemento do ponto de extensão e será exibido na janela de preferências.
- Class – É o nome completo da classe que implementa a interface comum para as páginas de configuração, esta interface é a org.eclipse.ui.IWorkbenchPreferencePage.
- Category – É a categoria onde este elemento deverá se encaixar. Esta propriedade só deve ser utilizada quando o elemento em questão é um subitem de outro elemento.

A janela de preferências para acessar os elementos de configuração está sobre o menu "Ajuda". Este caminho foi especificado na classe ApplicationActionBarAdvisor, através da fábrica de ações org.eclipse.ui.actions.ActionFactory.

As configurações que utilizam o ponto de extensão acima referido são as configurações de banco de dados, as configurações do TSC (controlador) e as configurações do subsistema de consultas SQL. Estas configurações estão detalhadas nos tópicos a seguir.

9.3.1.1 Configurações do TSC

Para que o TSC possa receber eventos dos SLMs adicionados, é necessário configurar o número identificador do TSC e a porta em que serão escutados os eventos lançados pelos SLMs. Para isso, devem-se utilizar os parâmetros de configuração do controlador através das preferências, localizado no menu "Ajuda" da janela principal. A seguinte figura exemplifica essa janela:

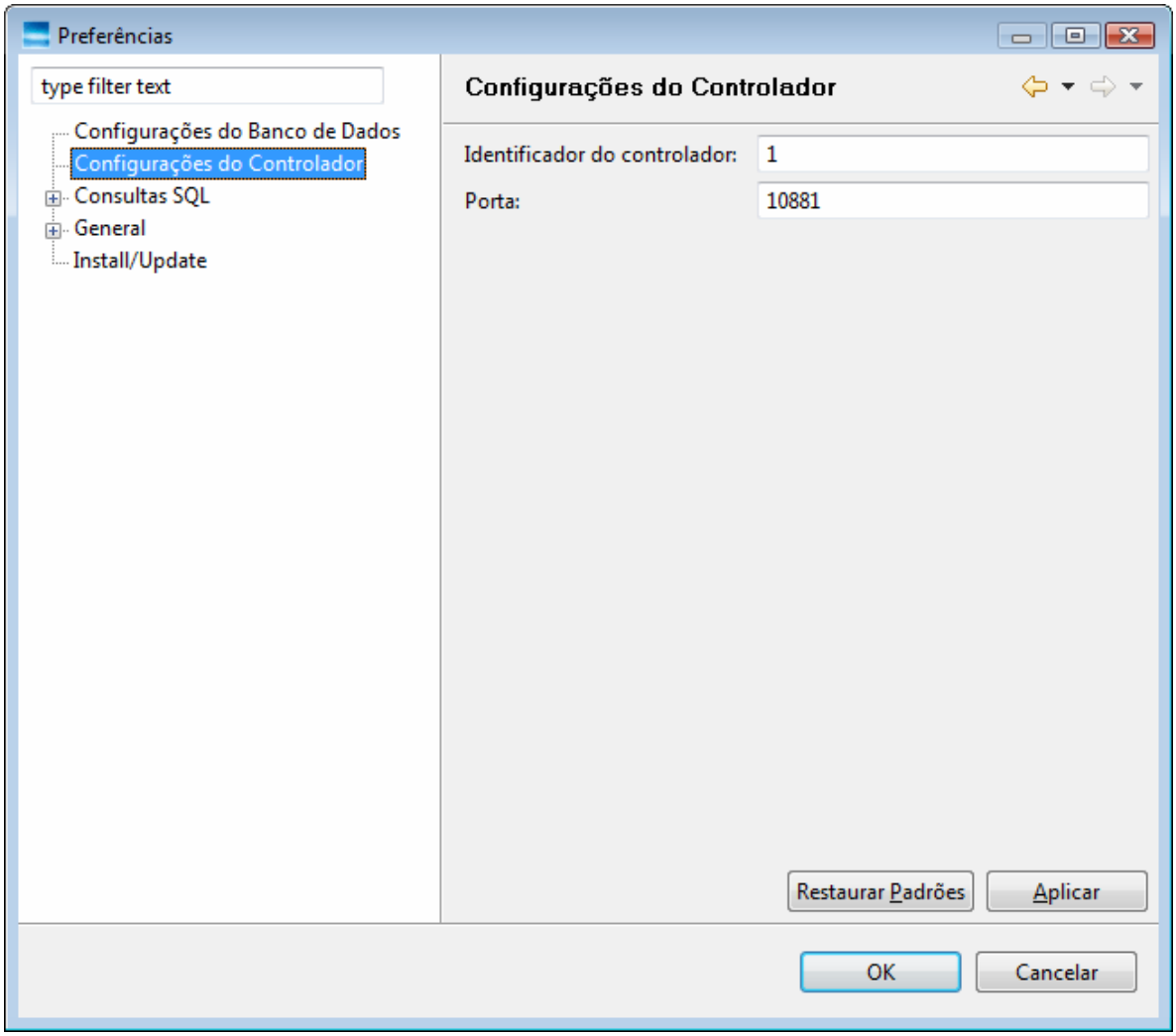


Ilustração 18 Configurção TSC

O endereço de callback do TSC deverá ser configurado nos módulos integrados ao sistema, através do comando set_TSC_callback, definido pela LECIS. O endereço é composto pelo endereço IP e a porta do servidor XML-RPC do TSC.

No caso de alterações das configurações, o TSC reiniciará o seu servidor XML-RPC para atender as novas configurações.

O elemento do ponto de extensão org.eclipse.ui.preferencePages utilizado para a construção desta janela é o tscConfigPage, com o nome de “Configurações do Controlador” e com a classe TSCPreferencePage.

9.3.1.2 Configurações do banco de dados

Para a configuração dos parâmetros do banco de dados foi produzido um elemento do ponto de extensão org.eclipse.ui.preferencePages, cuja identificação é dbConfigPage, nome é "Configurações do Banco de Dados" e a classe feita para esta página é a DatabasePreferencePage.

Para a conexão ao banco de dados do sistema é necessário configurar os seus parâmetros na janela de configuração. Os parâmetros são os seguintes:

- Host – Endereço IP ou DNS correspondente ao endereço da máquina servidor de banco de dados.
- Porta – O número da porta em que o serviço de bando de dados está escutando.
- Nome do usuário – Usuário do banco de dados com privilégios de acesso à base de dados do sistema.
- Password – É a senha do usuário correspondente.
- Database – É o nome do schema da base de dados do sistema.

A seguinte figura ilustra a janela de configuração dos parâmetros do banco de dados:

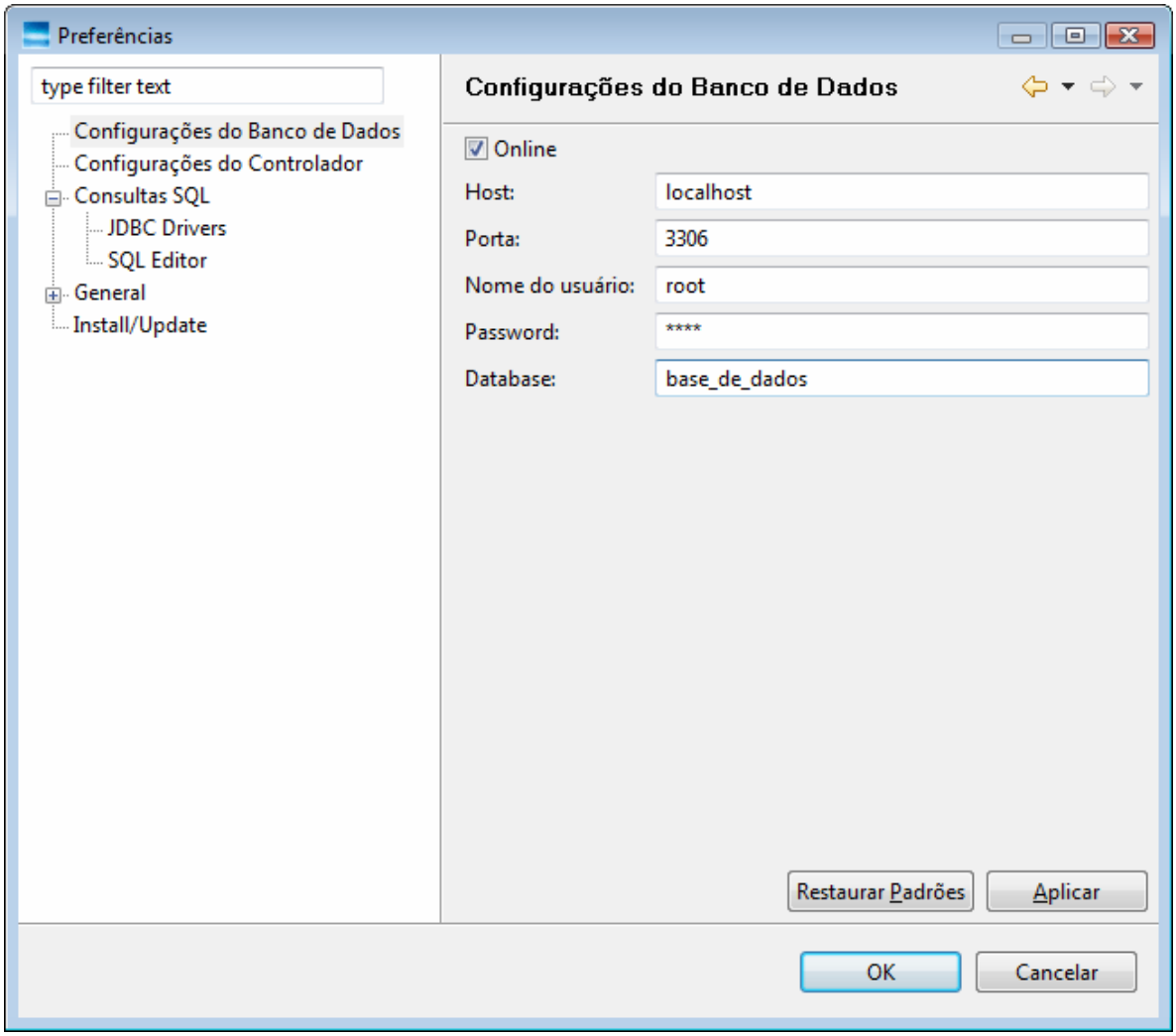


Ilustração 19 Configurações do banco de dados

Depois de configurados os parâmetros de conexão à base de dados, o sistema recupera todas as informações dos SLMs integrados, assim como os procedimentos associados à esses SLMs. Este passo de configuração é essencial para a utilização plena de todos as funcionalidades do sistema.

9.3.1.3 Configurações de consultas SQL

O gerenciamento das configurações do subsistema de consultas SQL, assim como todos os seus pontos de extensão utilizados e elementos produzidos são realizados pelo plugin SQLExplorer, porém, estas funcionalidades foram traduzidas para o português através de um fragment do plugin.

As configurações podem ser acessas através da janela de preferências, selecionando o item “Consultas SQL”, como demonstrado na figura a seguir:

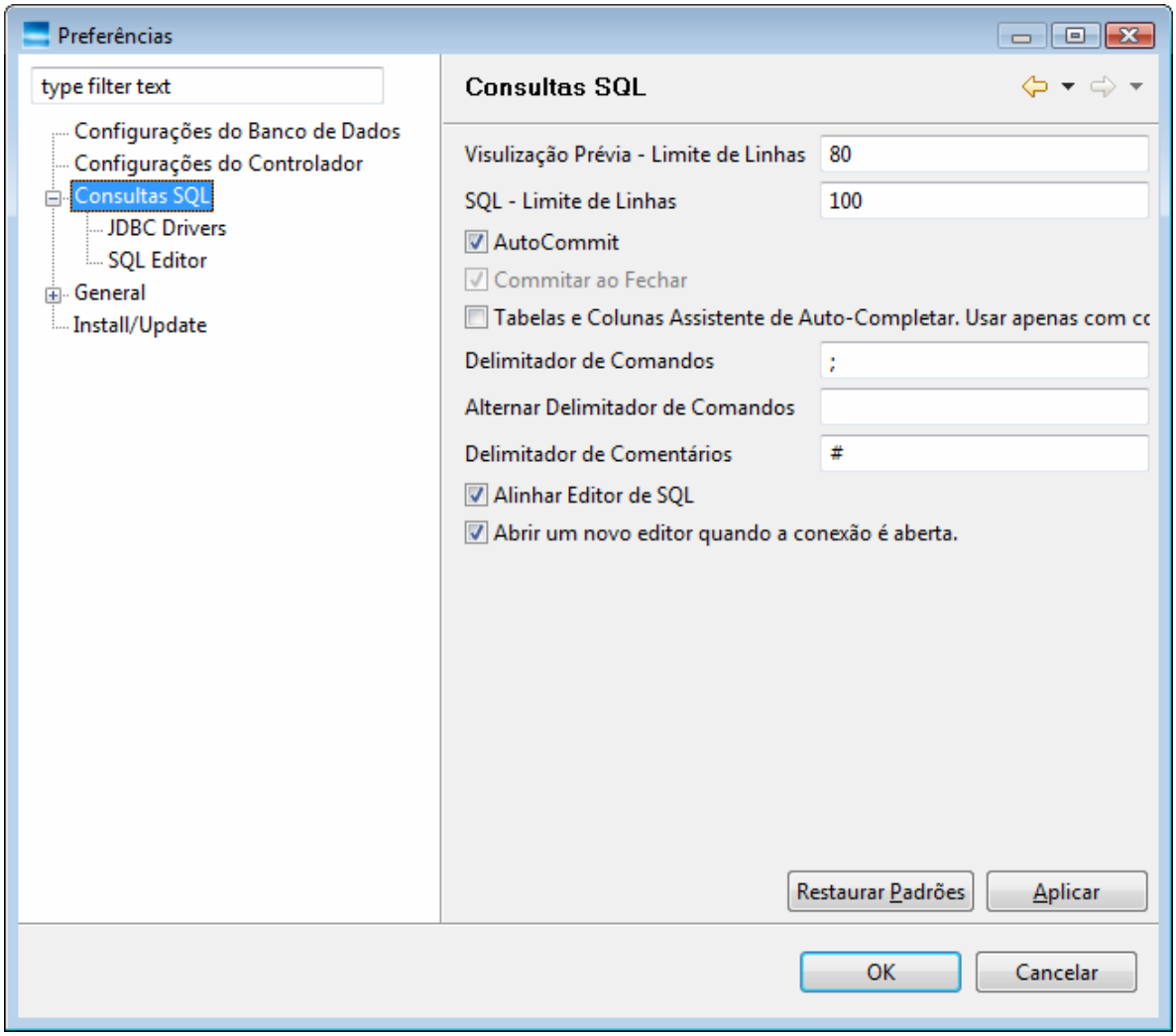


Ilustração 20 Configuração de consulta SQL

9.3.2 Gerenciamento de SLMs

O subsistema responsável pelo gerenciamento da integração dos SLMs ao TSC é o Gerenciamento de SLMs, através dele o usuário pode visualizar graficamente quais SLMs foram adicionados ao TSC e quais estão disponíveis para a criação de procedimentos e experiências laboratoriais, além de poder adicionar novos módulos ao sistema através de seus endereços.

Para este subsistema foi utilizado o ponto de extensão org.eclipse.ui.views para a criação de uma view, que está contida na perspectiva Sinótico. A view foi nomeada como "Módulos", representando o subsistema de gerenciamento de SLMs.

A view de módulos possui uma Treeviewer, onde o usuário pode visualizar os módulos integrados ao sistema e interagir com eles através de um menu de contexto. A view ainda possui botões em uma barra, que representam funcionalidades gerais.

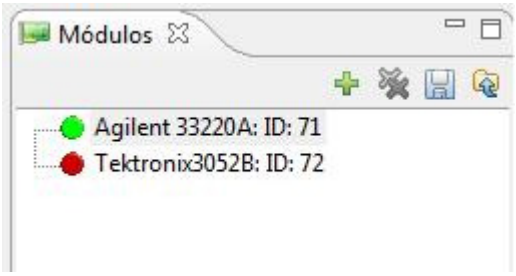


Ilustração 21 View de módulos

A view de módulos está ilustrada na figura acima, como exemplo, pode-se observar os módulos Agilet 33220A e o Tektronix 3052B, que estão integrados ao sistema. A cor verde indica que o módulo está disponível para execução, enquanto que a cor vermelha indica que o módulo precisa ser inicializado e seu endereço atualizado.

9.3.2.1 Adicionar módulos

Para adicionar novos módulos ao sistema é necessário indicar o endereço de um SLM ativo, ou seja, o SLM deve estar em execução, já que nesta etapa o TSC realiza uma requisição ao SLM, pedindo seu DCD. Através do arquivo de recursos o TSC adiciona o SLM correspondente na view de módulos, sendo descrito pelo seu nome e seu número identificador.

9.3.2.2 Atualizar módulos

É possível atualizar as informações de um SLM através da funcionalidade de atualização. Novamente é necessário indicar o novo endereço do módulo, um novo arquivo de recursos será carregado, atualizando as funcionalidades e propriedades do SLM correspondente.

9.3.2.3 Remover módulos

Os módulos já adicionados ao sistema podem ser removidos através desta funcionalidade. Os módulos que estiverem executando um procedimento também podem ser removidos, porém, uma mensagem de confirmação é exibida ao usuário.

9.3.2.4 Salvar módulos

Os módulos devidamente integrados ao sistema podem ser salvos no banco de dados através desta funcionalidade. São persistidos apenas o nome, o número identificador e o endereço dos módulos salvos. A partir destes dados o TSC efetua o carregamento de cada módulo, validando as suas informações com o DCD requerido.

9.3.2.5 Carregar módulos

Os SLMs persistidos no banco de dados podem ser carregados na view de módulos. Para isso, o TSC recupera o endereço de cada módulo no banco de dados e efetua um requerimento do DCD, validando suas informações. Se o SLM não está disponível ou o TSC não consegue efetivamente realizar a comunicação com o SLM, o módulo é carregado no sistema com o status de desativado, para a sua ativação é necessário efetuar uma atualização do módulo correspondente.

Este procedimento é sempre executado automaticamente durante a inicialização do sistema ou quando as configurações do banco de dados são alteradas.

9.3.2.6 Criar procedimentos

Através da view de módulos também é possível criar procedimentos de experiências laboratoriais para os módulos integrados. Esta funcionalidade faz parte do subsistema de gerenciamento de procedimentos.

9.3.3 Gerenciamento de Procedimentos

O controle dos módulos laboratoriais é feito através da execução de procedimentos. Estes fazem parte do subsistema de Gerenciamento de Procedimentos, que é responsável pela sua execução, criação, manutenção e persistência. Este subsistema responde também pela visualização e armazenamento de resultados das operações que são executadas com sucesso. Neste subsistema, todas as atividades do usuário são gravadas no banco de dados, a fim de se ter um registro histórico dos acontecimentos durante uma experiência.

Fazem parte deste subsistema as views de Procedimentos, Resultados e Tabela de Resultadas. Em que cada uma contém as funcionalidades que compõem o subsistema de

Gerenciamento de Procedimentos. As views citadas estão todas contidas na perspectiva Procedimentos.

9.3.3.1 Estrutura de controle

Os procedimentos são compostos de ações de comandos dos SLMs, de ações de desvio do fluxo e de ações de espera. Esta estrutura está representada no diagrama de classes conceitual a seguir:

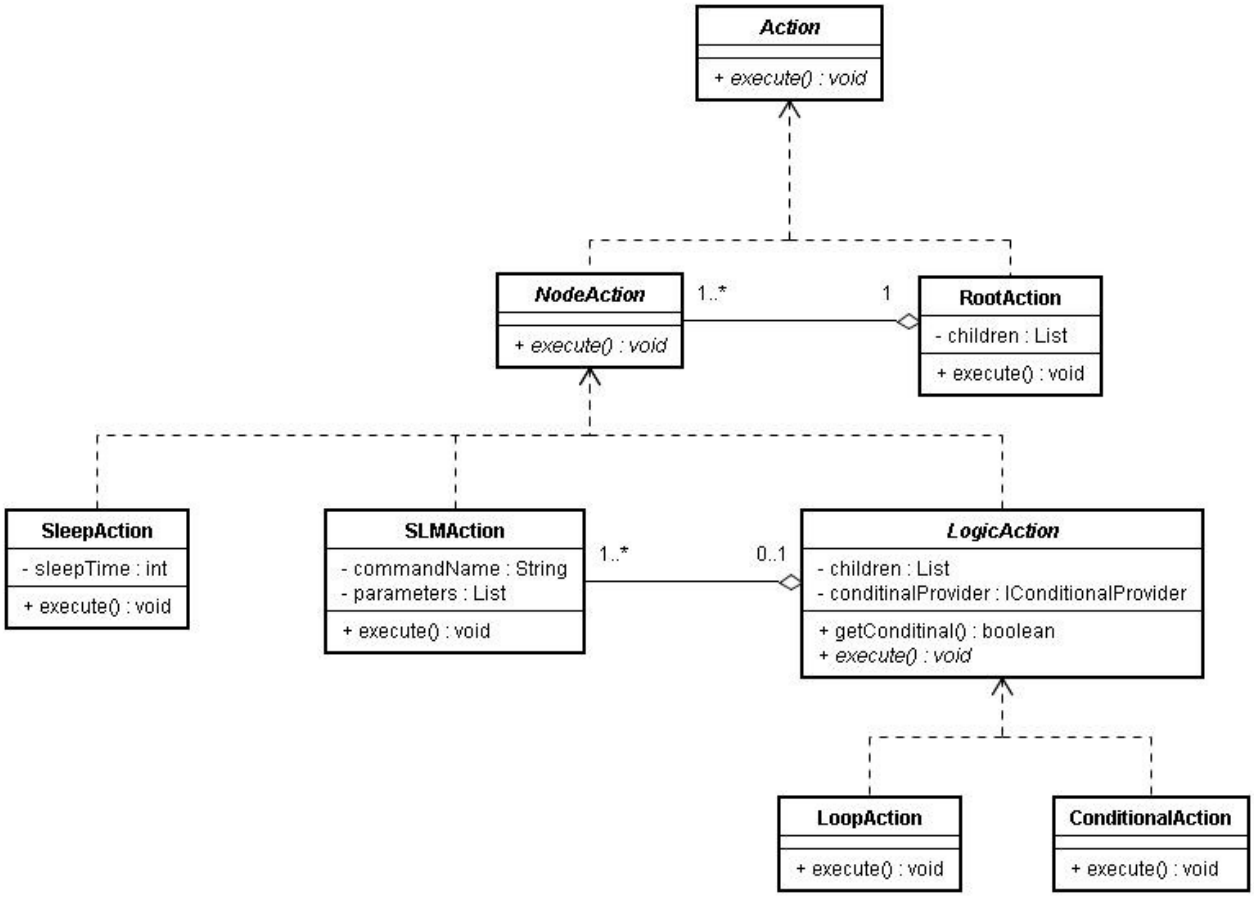


Ilustração 22 Diagrama de classes de ação

- Action – É a classe abstrata que representa uma ação a ser executada, possui o método execute, que deve ser implementado pelas classes que a estendem.
- RootAction – É o container das ações de nó, que estão inseridas no atributo de lista children. A sua operação execute executará todas as ações de nó seqüencialmente.

- NodeAction – É a classe abstrata que representa as ações de nó, que devem estar, necessariamente, contidas em um container de ações.
- SLMAction – É a ação a ser executada em um SLM. A ação possui o nome e os parâmetros do comando, que são dados pelo usuário a partir da tela de criação de procedimentos. A sua implementação da operação execute envia o comando e seus parâmetros ao SLM, que deverá realizar a execução dada e retornar o seu código.
- SleepAction – Esta ação faz com que o procedimento fique aguardando o tempo especificado pelo usuário. A interface ficará destravada, sem impedimentos ao usuário, já que os procedimentos são executados em uma thread separada.
- LogicAction – É uma classe abstrata que representa as ações que dependem de uma lógica. É também um container de SLMActions.
- ConditionalAction – Representa a condição "se", ou seja, se a sua condição for verdadeira, as suas SLMActions filhas são executadas seqüencialmente.
- LoopAction – Representa a condição "enquanto", ou seja, o seu bloco de SLMActions filhas é executado enquanto a condição dada for verdadeira.

As ações do tipo LogicAction devem verificar a se a sua condição dada é verdadeira, estas condições podem ser as seguintes:

- Variável de sistema e valor fixo – Nesta condição, o valor da variável de sistema de SLM é comparado com um valor fixo.
- Variável de sistema e outra variável de sistema – Nesta condição os valores de duas variáveis de sistema são comparadas. É possível comparar tanto variáveis de SLMs diferentes, quando variáveis de um mesmo SLM.

A interface que abstrai os tipos de condição é a IConditionProvider e as duas classes que representam as condições existentes e implementam esta interface são: SysvarToFixedValueComparison e SysvarToSysvarComparison. Desta forma, se futuramente for necessário acrescentar novos tipos de comparação, bastará criar outras classes que implementam a interface IConditionProvider.

Portanto, todo procedimento possui apenas uma RootAction, que é um container de NodeActions. As NodeActions podem ser de três tipos, SLMActions, que são as ações dos comandos dos SLMs em si, SleepActions, que são ações de aguardo ou espera e as LogicActions, que são ações que executam um container de SLMActions dependendo de uma dada condição. Todo procedimento de qualquer experiência seguirá esta estrutura.

A partir do imediatamente acima exposto e das classes e seu diagrama, é notório que a estrutura dos procedimentos atende as experiências de laboratório simples e gerais. Já que o fluxo das execuções dos comandos dos módulos pode ser alterado dependendo de uma condição específica de suas variáveis de sistema.

9.3.3.2 Criação de procedimentos

Os procedimentos podem ser criados pelo usuário a partir dos módulos ativos no sistema na view de módulos ou em um sinótico aberto. Depois de escolher os módulos que comporão o procedimento, o usuário pode definir as ações do procedimento a partir de um passo a passo.

O procedimento deverá ter um título, que o identificará, a partir dele o usuário poderá efetuar chamadas ao procedimento. Com isso, o sistema disponibiliza todas as ações que o usuário poderá escolher para o procedimento. As ações específicas de cada SLM são determinadas através de seu arquivo DCD correspondente, disponibilizando assim, todas os comandos e seus parâmetros. Dado um comando escolhido, o usuário define o valor dos parâmetros, adicionando, deste modo, a ação ao procedimento.

A partir do momento que o procedimento é criado, ele é adicionado na lista de procedimentos do sistema. Esta lista está representada em uma classe singleton, chamada de ProcedureList. A classe contém todos os procedimentos em memória e as operações básicas de persistência dos procedimentos, como salvar, carregar e remover. É a partir desta classe que a lista de procedimentos disponíveis é apresentada na view de procedimentos.

O passo a passo de criação de procedimentos foi implementado utilizando-se a abstração de Wizards do JFace. Porém, suas classes foram estendidas para a configuração adequada ao problema, por exemplo, foram adicionados novos botões na barra de botões padrão. Estas classes provêem um diálogo simples e didático com o usuário, onde ele deverá

efetuar a criação dos procedimentos de cada experiência. As classes referentes estão no pacote e subpacotes do common.wizard.

É possível também editar um procedimento criado utilizando-se do mesmo wizard de criação. Porém para esta funcionalidade, o wizard é aberto e preenchido com o procedimento dado, a partir dele, o usuário poderá acrescentar novas ações, editar as antigas ou mesmo remover as ações que desejar.

As classes relacionadas à criação e manutenção dos procedimentos estão no pacote common.procedures e common.procedures.events , e estão listadas a seguir:

- Procedure – É a classe que representa a entidade de procedimentos, possuindo os atributos necessários para a sua representação.
- ProcedureCreator – É a classe de controle para a criação de procedimentos.
- ProcedureList – É a classe singleton de controle que armazena em memória todos os procedimentos relacionados aos módulos incluídos no sistema.
- ProcedureListModificationEvent– É a classe que representa o evento de modificação da lista de procedures em memória.
- ProcedureListModificationListener – É a interface que define os métodos de escuta dos eventos do tipo ProcedureListModificationEvent.

9.3.3.3 Remover procedimentos

O processo de remoção de procedimentos é simples, na view de procedimentos estão listados todos os procedimentos em memória, os quais podem ser removidos através de um botão. O procedimento é removido da memória, ou seja, da lista de procedimentos da classe singleton ProcedureList. Para esta modificação ser persistida no banco de dados é necessário salvar a lista.

9.3.3.4 Procedimentos em arquivo

Os procedimentos podem ser salvos em um arquivo texto no formato XML, basta ser definido o caminho a ser utilizada para que o arquivo seja salvo. Este caminho pode ser local ou da rede. O procedimento é salvo utilizando-se o meio de codificação da classe

XMLEncoder. Porém, este processo pode ser alterado futuramente para que o procedimento possa ser salvo em arquivos de formatos diversos.

Os arquivos de procedimentos gerados pelo sistema também podem ser recuperados para a memória, utilizando-se do mesmo meio de codificação, porém da classe XMLDecoder.

9.3.3.5 Procedimentos no banco de dados

Os procedimentos podem ser salvos no banco de dados do sistema através do subsistema de persistência de dados. Porém, por motivos de segurança, é persistido apenas o caminho do arquivo do procedimento correspondente. Desta forma, o procedimento antes de ser persistido no banco de dados é necessário que seja salvo em arquivo. Este processo é feito automaticamente pelo sistema, salvando todos os procedimentos em um pasta padrão. Portanto, para que o usuário possa carregar os procedimentos salvos no banco de dados é necessário que ele os possua nesta pasta padrão.

9.3.3.6 Execução de procedimentos

A execução de procedimentos pode ser feita através da view de procedimentos ou através de um sinótico aberto. Quando um procedimento é executado uma nova thread é aberta para a sua execução. É chamado então, o método execute da instância da classe RootAction correspondente ao procedimento.

Para todo procedimento executado é criado um log de execução, que pode ser visualizado na view de procedimentos. Neste log são registrados todos os comandos enviados aos seus respectivos SLMs, assim como o resultado destes, que são apresentados na forma da estrutura SLMRESULT. Este log é feito a partir de eventos gerados durante a execução do procedimento, portanto, cada instância da view de procedimentos é um listener desses tipos de eventos. Podem-se facilmente estender as funcionalidades desses registros criando novos listeners para estes eventos.

As classes de execução e de envio e leitura de eventos estão dentro dos pacotes e subpacotes common.procedures, as quais são:

- ProcedureExecutor – É a classe de controle para a execução de procedimentos e envio de eventos relacionados.
- NodeActionExecutingEvent – É a classe que representa o evento de execução de ações do tipo NodeAction.
- NodeActionExecutingListener– É a interface que define os métodos de escuta dos eventos do tipo NodeActionExecutingEvent.
- ProcedureStartedEvent – É a classe que representa o evento de início de execução de procedimentos.
- ProcedureStartedListener – É a interface que define os métodos de escuta dos eventos do tipo ProcedureStartedListener.

9.3.3.7 Tabela de Resultados

Os procedimentos podem possuir ações específicas dos instrumentos laboratoriais, ou seja, comandos do tipo run_op. Estes comandos possuem resultados que devem ser armazenados e monitorados pelo TSC. Para essa funcionalidade foi criada a view "Tabela de Resultados", representada pela classe ResultsTableView. A view possui uma tabela em que ficam registrados todos os comandos executados do tipo run_op.

A classe que representa a entidade de resultados de uma operação específica é a ResultEnry. Ela contém todos os atributos para esta representação. Com isso, são criados objetos desta classe pela classe de controle ProcedureExecutor durante a execução do tipo de operação run_op e, por fim, introduzida na tabela de resultados.

A partir da tabela de resultados o usuário poderá efetuar as operações de persistência dos resultados em memória ou no banco de dados, para que eles possam ser analisados posteriormente.

A atualização do status e da data de fim das operações específicas é feita através do envio de eventos dos SLMs ao TSC. Com isso, o usuário poderá requisitar o valor do resultado correspondente e o TSC realizará esta requisição através da execução do comando get_result_data, passando como parâmetro o número de identificação da operação requisitada.

9.3.3.8 Visualização de Resultados

Através da tabela de resultados, o usuário do sistema poderá exibir um resultado da experiência laboratorial. Os resultados são exibidos em uma view separada, denominada "Resultados". Porém, para isso, o TSC invoca o plugin do resultado correspondente para que seja exibida corretamente a representação gráfica do resultado. Ou seja, cada SLM poderá contribuir com plugins ao TSC para a representação gráfica de seus resultados.

No caso do TSC não encontrar plugins do SLM para a visualização de seus resultados, será utilizado, automaticamente, o plugin de visualização genérico, que apenas imprime o texto do resultado na view "Resultados".

Este artifício foi criado para a visualização de resultados gráficos laboratoriais quaisquer. Já que os instrumentos laboratoriais geram resultados distintos e muitas vezes únicos, tornando impossível a criação de uma abstração única. Um exemplo disto é o monitoramento de sinais de um osciloscópio, que é totalmente diferente do monitoramento de um sensor de temperatura.

9.3.3.9 Plugins de visualização de resultados

Para a visualização de resultados gráficos de experimentos laboratoriais quaisquer foi criado o ponto de extensão "slmShowResults", que fornece ao desenvolvedor um composite SWT, ou seja, um espaço onde poderão ser incluídos componentes gráficos compatíveis com o plugin SWT/JFace.

O ponto de extensão da aplicação Eclipse RCP do TSC criado para a visualização de resultados de operações deverá possuir elementos que têm as seguintes propriedades:

- id – Identificação do plugin, deve ser único.
- name – Corresponde ao nome do SLM correspondente ao resultado gerado.
- showResultsClass – É o nome completo da classe que estende a classe abstrata SLMShowResults.

Como toda aplicação Eclipse RCP, o TSC quando executado, são carregados todos os plugins de sua pasta "plugins". O TSC identifica estes plugins e os coloca em seu

container. Apesar do TSC carregar as informações básicas, ou seja, os dados do arquivo plugin.xml, de todos seus plugins em memória, os plugins só são efetivamente carregados ou descartados conforme a sua necessidade, não sobrecarregando o sistema operacional desnecessariamente.

Para a apresentação da representação gráfica do resultado correto, o TSC percorrerá todos os elementos do ponto de extensão "slmShowResults" carregados, procurando pelo elemento que possui a propriedade name igual ao nome do SLM do resultado correspondente.

Como listado acima, os pontos de extensão "slmShowResults" deverão ter também o elemento showResultsClass, que corresponde ao nome completo da classe que estende a classe abstrata SLMShowResults, esta classe abstrata possui principalmente o método createControls, que recebe um composite, em que poderão ser incluídos elementos Java compatíveis com o plugin SWT/JFace e deverá retorná-lo preenchido com a representação gráfica o resultado correspondente.

9.3.3.10 Criação de plugins de visualização de resultados

Para a criação dos plugins de visualização de resultados de operações especificas dos módulos laboratoriais deste projeto foi seguido o passo a passo desenvolvido em apêndice à esta documentação. Porém, o mesmo passo a passo poderá ser utilizado para a criação de plugins de visualização de resultados de novos módulos laboratoriais não previstos inicialmente.

9.3.4 Persistência de dados

A persistência de dados do Sistema de Monitoramento e Controle Laboratorial é feita em um banco de dados relacional. O sistema gerenciador de banco de dados adotado foi o MySQL, versão 5.0.

O subsistema de persistência de dados será mais detalhado no capítulo de persistência de dados, onde estão descritos as classes e o modelo de dados criados, entre outros.

9.3.5 Consultas SQL

O subsistema de consultas SQL deve ser acessado através da perspectiva intitulada "Consultas SQL". Através dela o usuário é capaz de consultar dados diretamente do banco de dados do sistema, efetuar a manutenção do banco de dados, assim como inserir ou atualizar os dados do sistema, como por exemplo, a inclusão de novas TAGs de dados laboratoriais.

O subsistema de consultas SQL foi feito basicamente utilizando-se o plugin SQLExplorer, o qual é um projeto de software livre para a manipulação de Sistemas de Gerenciamento de Banco de Dados gerais. Este plugin provê as funcionalidades de acesso à base de dados através da execução de scripts SQL.

A tradução do plugin SQLExplorer para o português foi feita através de um novo fragment, que apenas atualiza o arquivo de mensagens, ou seja, os textos apresentados ao usuário. Este fragment só é ativado quando o computador em que está rodando o aplicativo está sobre um sistema operacional configurado para o Brasil, ou seja, quando seu locale é igual a "pt-BR", diferentemente dos Estados Unidos, que possuem locale "en-US". Entretanto, é possível ativar o fragment mesmo rodando o aplicativo fora do locale pt-BR, basta inserir na linha de comando do aplicativo –nl pt_BR para executar o programa sobre o locale "pt-BR", ou –nl en_US para executar o programa sobre o locale "en-US".

Este subsistema possui as seguintes views e funcionalidades específicas:

- Conexões – Lista o pool de conexões disponíveis, podendo-se conectar aos bancos de dados disponíveis. A base de dados do sistema já possui um perfil de conexão configurado automaticamente.
- Estrutura do Banco de Dados – Apresenta a estrutura do banco de dados conectados, mostrando as tabelas e suas propriedades.
- Detalhes do Banco de Dados – São detalhes da estrutura do banco de dados selecionada.
- SQL Editor – É o editor de scripts SQL, através dele o usuário poderá editar, salvar e executar os scripts SQL que desejar.
- Resultados SQL – São os resultados das execuções dos scripts SQL.

- Histórico SQL – Todas os scripts SQL executados são armazenados, criando um histórico. O usuário pode pesquisar e executar novamente os scripts SQL executados anteriormente.

9.3.6 Relatórios

O subsistema Relatórios possibilita ao usuário a criação de relatórios de dados históricos das experiências monitoradas e controladas. Os dados históricos possuem as seguintes propriedades:

- Data – Corresponde à data em que o dado foi gerado e registrado pelo módulo correspondente.
- Tag – Corresponde à TAG do dado, as Tags são, normalmente, as variáveis de sistema.
- SLM – Corresponde ao módulo que registrou o dado.
- Valor – Corresponde ao valor da Tag do dado histórico.

Uma TAG possui as seguintes propriedades:

- Id – É a identificação da TAG dentro do sistema. O sistema não pode possuir TAGs de mesmo id.
- Name – É o nome da TAG apresentado ao usuário.
- Type – É o tipo da TAG, o sistema pode ter várias TAGs do mesmo tipo.
- MeasureUnit – Corresponde à unidade de medida da TAG.

O relatório pode ser gerado em dados tabulares ou em gráficos, sendo que o usuário pode definir limites de mínimo e máximo para a data dos dados históricos. O relatório gerado pode ser salvo em arquivo ou impresso.

Os dados históricos são gerados pelos SLM, através da aquisição de dados dos instrumentos laboratoriais, porém, a persistência dos dados é feita pelo TSC. Logo, o SLM deve enviar um evento ao TSC para este registro, o evento utilizado é do tipo DATA_LINK / DATA BASE e são enviados todas os parâmetros necessários para o registro do dado histórico.

Para confecção do relatório de dados tabulares foi utilizado o plugin Eclipse BIRT. Este plugin possui um report designer, ou seja, um aplicativo que ajuda a desenhar um relatório, através dele foi gerado o relatório de dados históricos, passando a conexão com a base de dados do sistema e o script SQL correspondente para a consulta dos dados correspondentes aos dados históricos.

Para este subsistema foi desenvolvido o plugin report, que fornece a perspectiva "Relatórios" ao Sistema de Controle e Monitoramento Laboratorial. A sua principal view é a view "Parâmetros", que definem quais serão os valores dos parâmetros para a criação do relatório. Os parâmetros são os seguintes:

- TAG – Corresponde à Tag cadastrada no banco de dados.
- Data inicial – É a data inicial do relatório, apenas serão coletados os dados que tiverem registro posterior a esta data.
- Data final – É a data de limite superior, só serão coletados os dados que tiverem registro anterior a esta data.

Definidos os parâmetros, o usuário poderá gerar o relatório. O relatório tabular é exibido em uma view separada, denominada "Relatório". Este processo é comum e igual a todos os tipos de TAG do sistema, e independe do módulo que a gerou. Já o seu gráfico é exibido na view "Gráfico", porém, este processo depende do tipo da TAG escolhida. Foi desenvolvido um ponto de extensão para que os gráficos sejam exibidos, ou seja, o TSC invoca o plugin correspondente para a exibição do gráfico. Assim, cada tipo de TAG poderá ter contribuições com plugins ao TSC para a representação gráfica de seus dados históricos.

No caso do TSC não encontrar plugins para a visualização específica dos dados históricos de um tipo de TAG, será utilizado, automaticamente, o plugin de visualização genérico, que tenta converter os valores em dos dados históricos em números e depois traçar um gráfico XY, ou seja, valor x tempo.

Porém, nem sempre os dados históricos são valores numéricos, eles podem ter outros tipos de valores, o que impossibilita a geração automática de gráficos para qualquer tipo de TAG. Nesse caso, é necessária a construção de um plugin especifico. Por exemplo, a forma de onda dos sinais de um gerador de sinais, pode assumir valores históricos como

senoidal, triangular, pulso, entre outros. Ou uma válvula, que pode ter suas posições como A, B, C, no caso de três posições.

9.3.6.1 Plugins para gráficos de relatórios

Para a visualização de gráficos de dados históricos de experiências quaisquer foi criado o ponto de extensão "graphViewControl", que fornece ao desenvolvedor um composite SWT, ou seja, um espaço onde poderão ser incluídos componentes gráficos compatíveis com o plugin SWT/JFace.

O ponto de extensão da aplicação Eclipse RCP foi criado dentro do plugin "report", ou seja, para a criação de elementos deste ponto de extensão é necessário a sua dependência. Este ponto de extensão, utilizado para a visualização de dados históricos de experiências quaisquer, possui os seus elementos deverão ter as seguintes propriedades:

- id – Identificação do plugin, deve ser único.
- name – Corresponde ao tipo da TAG.
- GraphControlClass – É o nome completo da classe que implementa a interface IGraphViewPartControl.

Para a apresentação do gráfico correto, o plugin de relatórios possui um rotina que percorrerá todos os elementos do ponto de extensão "graphViewControl" carregados, procurando pelo elemento que tem a sua propriedade name igual ao tipo da TAG que deverá ser gerado o relatório.

Como listado acima, os elementos do ponto de extensão "graphViewControl" deverão ter também a propriedade GraphControlClass, que corresponde ao nome completo da classe que implementa a interface IGraphViewPartControl, esta interface possui apenas um método, que recebe um composite, em que poderão ser incluídos elementos Java compatíveis com o plugin SWT/JFace e deverá retorná-lo preenchido com a representação gráfica dos dados históricos correspondentes.

9.3.6.2 Criação de plugins para gráficos de relatórios

Para este projeto foi desenvolvido um plugin genérico de visualização de dados históricos de módulos laboratoriais, o qual apenas plota no tempo, o valor da TAG cadastrada

para o modulo. Para o desenvolvimento deste plugin foi utilizado o passo a passo descrito em apêndice à esta documentação.

Apesar disso, o mesmo passo a passo poderá ser utilizado para a criação de plugins de visualização de resultados de novos módulos laboratoriais não previstos inicialmente.

9.3.7 Sinótico

O subsistema Sinótico é o subsistema responsável pela criação de representações visuais dos instrumentos relacionados a um dado experimento, de onde se podem acompanhar visualmente os estados e propriedades de um dado instrumento. O subsistema foi todo criado utilizando-se a framework GEF (Graphical Editing Framework), versão 3.2.

Para cumprir tal objetivo, o subsistema Sinótico fornece uma perspectiva e um editor, onde o usuário pode colocar representações visuais dos módulos de cada instrumento.

9.3.7.1 Palheta

Na view do Sinótico existe uma palheta com todos os módulos cadastrados no sistema correspondente, ou seja, inseridos na lista em memória da classe singleton SLMList. A palheta pode ser reposicionada no editor, retraída quando não utilizada e reexibida quando o mouse é posicionado sobre ela. Ela está integrada com o subsistema Gerenciamento de SLMs, de modo que, caso um módulo seja adicionado ou removido do sistema, a palheta responderá de acordo, acrescentando ou removendo a entrada daquele módulo. Essa comunicação foi feita no código utilizando-se eventos.

O usuário pode selecionar uma entrada da palheta e posicionar o elemento selecionado na parte livre da tela do sinótico. Uma instância, ou elemento, daquela entrada será criado. Para um mesmo módulo, podem-se criar quantos elementos o usuário quiser na tela do sinótico. Todos esses elementos estarão associados a um mesmo módulo e, portanto, suas propriedades serão todas atualizadas simultaneamente quando a propriedade do módulo associado for modificada. O elemento criado exibirá, além da imagem associada para o módulo, um label com um nome (denominado alias) específico para aquele elemento e um label com o estado atual do módulo (tal label é alterado de acordo quando o estado do módulo muda).

9.3.7.2 Representação Gráfica dos Módulos

É possível escolher uma imagem específica para representar um módulo. Esse recurso é disponibilizado como uma action “Importar imagem para módulo...” no menu principal do programa. Alterar a imagem de um módulo modifica a imagem de todas as representações em todos os sinóticos. O mecanismo de importar imagem cria uma estrutura para a imagem na pasta destinada ao plugin do sinótico. A estrutura criada é da forma <id_do_modulo>/slm.jpg, onde <ID_do_modulo> é o ID (numérico, em princípio) do SLM para o qual se deseja importar a imagem, sendo que o ID de cada módulo deve ser único. É possível também escolher a imagem padrão do sistema para um módulo.

9.3.7.3 Propriedades

No Sinótico, através de um menu de contexto para um elemento selecionado, o usuário tem acesso às opções, como “Mostrar propriedades”. Ao se selecionar essa opção, o sistema exibe uma view de Properties com diversas propriedades do módulo associado ao elemento selecionado e mais algumas específicas do elemento selecionado, como, por exemplo, o alias do elemento. Algumas dessas propriedades são editáveis e alterar tais propriedades repercute na representação do elemento selecionado e, caso se aplique ao módulo, altera todos os elementos associados e a própria representação do módulo no sistema, possivelmente enviando comandos para o driver do instrumento respectivo. Por exemplo, alterar a propriedade “Forma de onda” de um módulo de um gerador de sinais para “SIN”, poderá fazer com que a saída do gerador seja um sinal senoidal. Porém, a validação dos valores digitados nas propriedades é de responsabilidade do módulo alvo e não do TSC. Ou seja, caso algum erro ocorra, o SLM deverá enviar um evento ao TSC.

9.3.7.4 Variáveis de Sistema

Também no menu de contexto do Sinótico encontra-se um submenu “Monitorar variáveis de sistema” que, quando expandido, exibe todas as possíveis variáveis de sistema para o módulo correspondente. Ao se selecionar alguma variável de sistema (que passa a ficar com uma marca de check), um label com o nome e valor daquela propriedade é acrescentado ao elemento gráfico representativo do sinótico correspondente. Esse label é atualizado automaticamente sempre que o valor da propriedade é modificado. Usando o mesmo submenu, clicando na mesma variável, ela perde a marca de check e o label é removido do elemento.

Existem também duas opções dentro do submenu, “Todas” e “Nenhuma” que, respectivamente, incluem todas as variáveis para monitoramento (marcando-as como checked e adicionando os labels respectivos aos elementos) ou retiram todas as variáveis do monitoramento (marcando-as como não checked e retirando os labels respectivos dos elementos). Toda a comunicação, entre o subsistema Gerenciamento de SLMs e o subsistema Sinótico para transmissão de informações como o estado e propriedades de um módulo é feita no código fonte utilizando-se de eventos.

9.3.7.5 Criação e Execução de Procedimentos

O subsistema Sinótico também está integrado ao subsistema Gerenciamento de Procedimentos, possibilitando a criação e execução de procedimentos através do sinótico. Para criar um procedimento, deve-se selecionar um ou mais elementos e abrir o menu de contexto, selecionando a opção de “Criar procedimento”. A partir daí o processo seguirá o mesmo da criação pelo subsistema de Gerenciamento de Procedimentos (de fato, o wizard invocado é o mesmo que o Gerenciamento de Procedimentos invoca).

Para executar um procedimento, deve-se selecionar um elemento e abrir o menu de contexto, selecionando o submenu de “Executar procedimentos”, de onde será listado o subconjunto de procedimentos relacionados ao módulo do elemento selecionado. Apenas os procedimentos que contenham aquele módulo serão exibidos. Se mais de um elemento for selecionado, as listas de ambos os módulos serão somadas e exibidas (apenas uma entrada por procedimento é exibida, independente de ambos os módulos estarem relacionados àquele procedimento). Qualquer alteração feita na lista de procedimentos será automaticamente propagada para a lista de procedimentos exibida para cada elemento. A comunicação entre os subsistemas é feita no código utilizando-se eventos.

9.3.7.6 Undo e Redo

Também é possível posicionar livremente os elementos no módulo e removê-los se desejado. O subsistema Sinótico possui as funcionalidades de “undo” e “redo” infinito (implementado usando-se o design pattern Command), permitindo desfazer alterações de posição e de inclusão remoção de elementos.

9.3.7.7 **Ações**

Em cada sinótico encontra-se um coolbar no topo da tela com actions que permitem ao usuário salvar o sinótico, salvar como, dar zoom in e zoom out, habilitar um grid e imprimir o sinótico atual (o qual é impresso em escala normal e sem grid).

9.3.7.8 Salvar e Abrir Sinóticos

Os sinóticos salvos em arquivo podem ser abertos depois, a fim de se continuar a operação com aquele sinótico. O conteúdo de um sinótico quando salvo é codificado sobre a forma de um XML, com formato semelhante ao apresentado abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<slmDiagram>
        <slmElement alias="Java_Finance:ID:25">
                <slm id="25" name="Java_Finance"/>
                <location x="123" y="51"/>
                <size height="97" width="91"/>
        </slmElement>
        <slmElement alias="Agilent 33220A:ID:71">
                <slm id="71" name=" Agilent 33220A "/>
                <location x="490" y="220"/>
                <size height="100" width="100"/>
        </slmElement>
</slmDiagram>
```

Existe apenas um elemento “slmDiagram”, o qual é o elemento raiz, que pode possuir um ou mais elementos do tipo “slmElement”, a fim de representar os módulos adicionados ao sinótico. Dentro dos elementos “slmElement”, encontram-se elementos que especificam qual o módulo associado àquele elemento (“slm”), qual a posição do elemento no sinótico (“location”) e qual o tamanho do elemento (“size”). Os arquivos que contém os XMLs gerados (arquivos .synoptic) são colocados dentro da pasta destinada ao plugin do sinótico.

9.3.7.9 Importação e Exportação de Sinóticos

Os sinóticos criados com esse subsistema e as imagens importadas podem ser exportados para arquivos do tipo zip e depois importados em uma outra instalação do TSC. Esta funcionalidade foi implementada através de extensions do plugin para os extension points org.eclipse.ui.exportWizards e org.eclipse.ui.importWizards. Essas extensions contribuem com wizards específicos para a funcionalidade genérica de import/export, de

modo a permitir que se possa importar e exportar tais elementos. A opção de importar e a de exportar apresentam-se no menu “Arquivo” do sistema.

Ao se fechar o sistema, o subsistema Sinótico cria um arquivo chamado openedEditors.xml, que codifica, em XML, quais eram os editores de sinótico abertos na última sessão de uso. O arquivo XML tem o seguinte formato:

```
<?xml version="1.0" encoding="UTF-8"?>
<openedEditors>
        <editor path="caminho completo do arquivo 1" title="sinotico1.synoptic"/>
        <editor path="caminho completo do arquivo 2" title="sinotico2.synoptic"/>
</openedEditors>
```

Existe apenas um elemento“openedEditors”, que é o elemento raiz do XML, para cada editor que estava aberto quando se fechou o aplicativo da última vez, um elemento “editor” é criado, com atributos que especificam sua localização no sistema de arquivos atual e seu título. Esse mecanismo de codificação da condição do sistema na última sessão permite que se continue rapidamente o trabalho iniciado em outra sessão. Porém, é necessário observar que os Sinóticos não salvos não são recuperados por este processo.

9.4 Estendendo e modificando os subsistemas

A maneira mais simples de se estender as funcionalidades dos subsistemas é incluir actions em menus de contexto ou coolbars, ou ainda modificar a arquitetura atual, respeitando-se a arquitetura dos plugins utilizados.

Como o TSC é uma aplicação Eclipse RCP, composta de plugins, podem-se também modificar seus comportamentos criando-se fragments e implementando um novo comportamento para a funcionalidade desejada.

Capítulo 10

10 Persistência de dados

10.1 Introdução

O Sistema de Monitoramento e Controle Laboratorial faz sua persistência de dados em um banco de dados relacional, o SGBD (Sistema Gerenciador de Banco de Dados) escolhido foi o MySQL, versão 5.0.

Como já visto, o sistema pode estar de maneira totalmente distribuída, com o software do TSC em uma máquina conectada na rede e o SGBD sendo executado em uma máquina dedicada em outro ponto da rede, um diagrama ilustrativo pode ser observado a seguir:

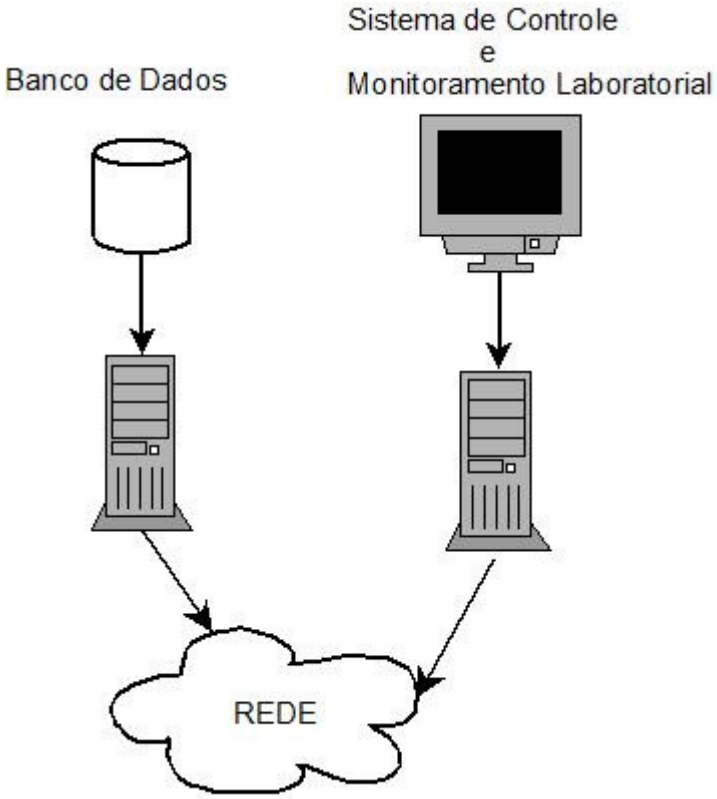


Ilustração 23 Configuração Distribuída

Outra configuração possível e mais simples é concentrar os sistemas em apenas uma máquina, tal configuração está ilustrada no diagrama a seguir:

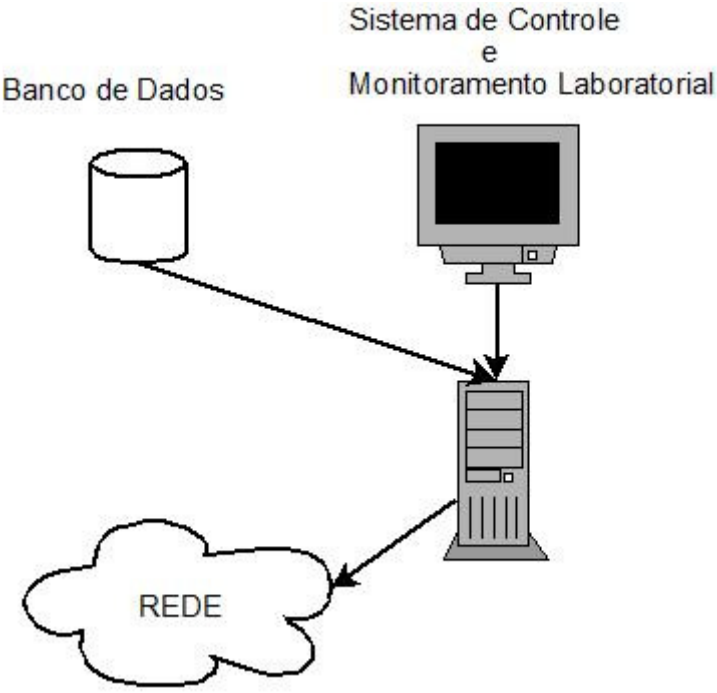


Ilustração 24 Configuração Concentrada

10.2 MySQL

MySQL é um SGBD que utiliza linguagem SQL como interface. É, atualmente, um dos SGBD mais populares, estão entre seus usuários: NASA, Friendster, Banco Bradesco, Dataprev HP, Nokia, Sony, Lufthansa, exército dos Estados Unidos, Cisco System, entre outros.

O MySQL é Open Source e distribuído sobre a licença GPL, permitindo que seus usuários alterem seu código fonte e o utilize sem a necessidade de comprar licenças.

A versão do MySQL adotada no sistema é a 5.0, que está disponível para diversos sistemas operacionais, como Windows, Linux, FreeBSD, Solaris, Mac OS X, entre outros.

10.3 Modelo de Dados

10.3.1 Diagrama Entidade Relacionamento

A modelagem de dados do sistema foi feita de modo a atender as necessidades de persistência dos dados, gerando assim, um diagrama entidade relacionamento, que simplifica o entendimento, em alto nível de abstração, do modelo de dados.

As entidades e seus relacionamentos podem ser visualizados no diagrama a seguir:

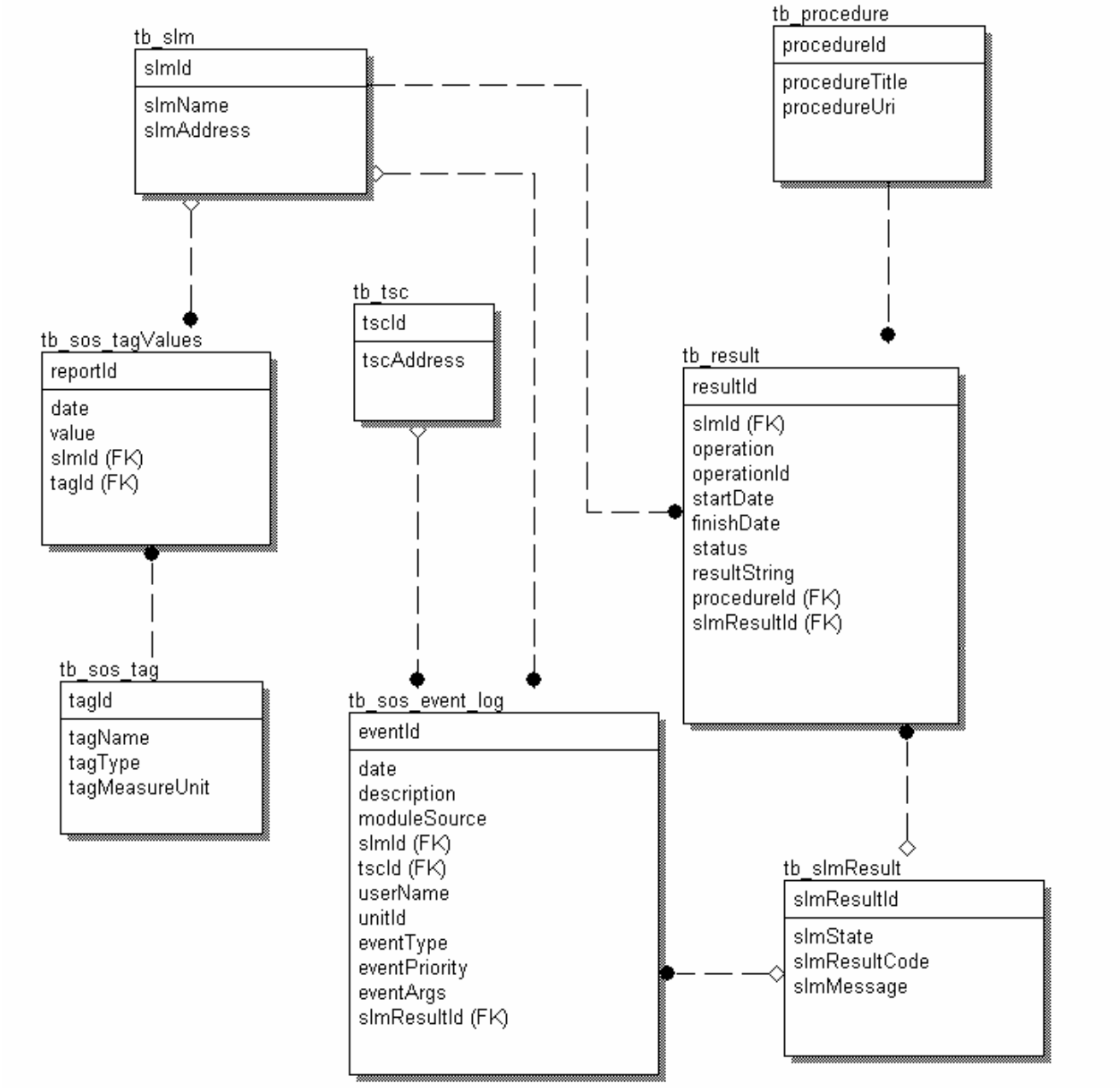


Ilustração 25 Diagrama Entidade Relacionamento

10.3.2 Entidades

A tabela a seguir lista as entidades do modelo de dados e suas descrições:

Tabela 4 Entidades do modelo de dados

Entidades	Descrição
tb_slm	Módulos, acoplados ao sistema, e suas configurações

tb_tsc	Configurações dos controladores e visualizadores
tb_procedure	Procedimentos de ações dos módulos e suas configurações
tb_result	Resultados gerados pelos módulos
tb_sos_event_log	Log de operações do sistema
tb_slm_result	Resultados parciais dos módulos
tb_sos_tag	Tags, cujos valores serão gravados pelo sistema
tb_sos_tag_values	Histórico dos valores de tags gerados pelos módulos

10.4 Criação da Base de Dados

A criação inicial da base de dados no MySQL foi feita executando-se um script SQL, contendo todas as rotinas necessárias para a criação das tabelas, configurações dos campos e tipos de acesso, para que fiquem compatíveis com a implementação de leitura e escrita do sistema.

O sistema não possui telas de configurações e gerenciamento de tags, que são utilizadas pelos módulos, por isso, se faz necessária a sua inserção manual, o que deverá ser feito pelo desenvolver do SLM. As tags deverão ser incluídas e configuradas na entidade tb_sos_tag.

O sistema também não possui telas de gerenciamento e configuração de controladores e visualizadores, por isso, se faz necessárias a inserção e manutenção destes, manualmente, pelo administrador do sistema.

10.5 Administração do SGBD

Os procedimentos de administração do sistema de gerenciamento de banco de dados (SGBD) utilizado para a persistência de dados deste projeto, Sistema de Monitoramento e Controle Laboratorial, estão em apêndice à esta documentação.

10.6 Pattern DAO

No projeto foi utilizada a design pattern DAO, que é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados.

A pattern DAO provê uma interface abstrata para o acesso à base de dados ou qualquer outro mecanismo de persistência de dados, disponibilizando operações específicas sem a necessidade de expor detalhes de execução. Esta isolação separa os interesses de que tipo de acesso aos dados é necessário à aplicação. Ou seja, podemos, facilmente, alterar o tipo de persistência de dados do sistema, optando por diferentes bancos de dados ou até por arquivos texto, por exemplo.

Os principais componentes utilizados na implementação do DAO são:

- Fábrica de classes DAO – É a classe que define qual tipo de implementação do DAO deverá ser utilizada pela camada lógica do sistema.
- Interface DAO – É quem define as operações básicas a serem implementadas pelas classes concretas. Geralmente são disponibilizados métodos de salvar, carregar, alterar e remover.
- Classe concreta DAO – É a classe que implementa a interface DAO, disponibilizados os meios reais de acesso aos dados, ou seja, são em seus métodos que estão escritos os meios reais de acesso físico aos dados.

Nesse contexto, as classes DAO do sistema estão contidas no projeto database e possui apenas sete interfaces DAO, que são:

- ProcedureDAO – Métodos de acesso aos procedimentos persistidos pelo sistema.
- ResultEntryDAO – Métodos de acesso aos resultados de processos persistidos pelo sistema.
- SLMDAO – Métodos de persistência de dados dos SLMs integrados ao sistema.
- SLMResultDAO – Métodos de persistência dos objetos SLMRESULT retornados pelos SLMs.
- EventLogDAO – Métodos de persistência dos logs de operações do sistema.
- TagDAO – Métodos de acesso às TAGs cadastradas no sistema.

- TagValuesDAO – Métos de persistência dos valores históricos das TAGs cadastradas no sistema.

Todas as interfaces DAO listadas anteriormente possuem uma classe concreta correspondente de acesso ao banco de dados MySQL. Caso seja necessário, no futuro, é possível mudar o tipo de acesso, implementado apenas as interfaces DAO para o acesso à outra base de dados ou outro tipo de persistência e trocando o retorno da fábrica de classes DAO.

A fábrica de classes DAO correspondente no sistema é a RtrdbFactory.

10.7 Segurança dos dados

Como já mencionado, o banco de dados pode estar arbitrariamente em diferentes nós da rede, e sua comunicação ocorrer mesmo através de firewalls, mas para isso, faz-se necessária a sua configuração.

Para que a comunicação do banco de dados tenha maior segurança é necessario configurar as diretivas do firewall para proteger o seu acesso. Como o acesso é realizado por meio de remoting em porta TCP, é necessário que as portas em que o servidor é disponibilizado possam ser acessadas apenas pelas máquinas em que o sistema se encontra.

Portanto, a configuração do firewall deve permitir a completa comunicação exclusivamente entre os servidores do TSC, dos SLMs e do banco de dados não permitir o acesso (recusando conexão, ignorando pedido ou retornando erro) a todos outros endereços de rede.

Outra importante medida de segurança dos dados é a realização periódica de um backup completo do banco, prevenindo assim, perdas de dados históricos de processos, de configurações e de operações realizadas pelos usuários.

Capítulo 11

11 Instrumentos Laboratoriais

Para o Sistema de Monitoramento e Controle Laboratorial, desenvolvido neste projeto, foram implementados módulos de comunicação para os instrumentos laboratoriais Agilent 33220A (gerador de sinais) e Tektronix 3052B (osciloscópio digital).

Para ambos os instrumentos foram desenvolvidos módulos escritos em Java, utilizando-se o framework comum, descrito no capítulo sobre SLM. Para a comunicação específica com o hardware dos instrumentos foram utilizados os drivers dos fabricantes, que foram encapsulados em uma outra camada. A camada que encapsula as chamadas dos drivers foi feita em C#, recebendo as chamadas por XML-RPC, utilizando a biblioteca open-source XML-RPC.net.

A camada de encapsulamento do driver é extremamente necessária para que os SLMs dos instrumentos possam abstrair as chamadas ao hardware específico do instrumento. Portanto, o encapsulador, ou wrapper, possibilita a troca do driver por um outro, alterando apenas a implementação do wrapper, sem a necessidade de modificações no SLM ou no TSC. Este mecanismo é fundamental, visto que as especificações dos protocolos de comunicação e os sistemas operacionais estão em constante desenvolvimento, o que acarretam nas alterações dos drivers existentes e na criação de novos.

Foram disponibilizados os comandos básicos de ambos os instrumentos, podendo-se monitorar sinais de saída de uma experiência, a partir do osciloscópio Tektronix 3052B. E também atuar sobre o sinal de entrada de uma experiência, a partir do gerador de sinais 33220A, modificando sua forma de onda, amplitude, freqüência ou offset.

Desta forma, a partir do projeto do Sistema de Monitoramento e Controle Laboratorial e os módulos desenvolvidos, é possível automatizar experimentos laboratoriais simples, monitorando e controlando seus sinais de entrada e saída.

A partir da flexibilidade e extensibilidade do sistema é possível também integrar novos módulos laboratoriais, permitindo a automatização de outros tipos de experiências.

11.1 Gerador de Sinais Agilent 33220A

O gerador de sinais 33220A, ilustrado na figura a seguir, é fabricado pela empresa Agilent Technologies. Através dele pode-se gerar formas de ondas pré-definidas ou ondas arbitrárias.

Ilustração 26 Agilent 33220A

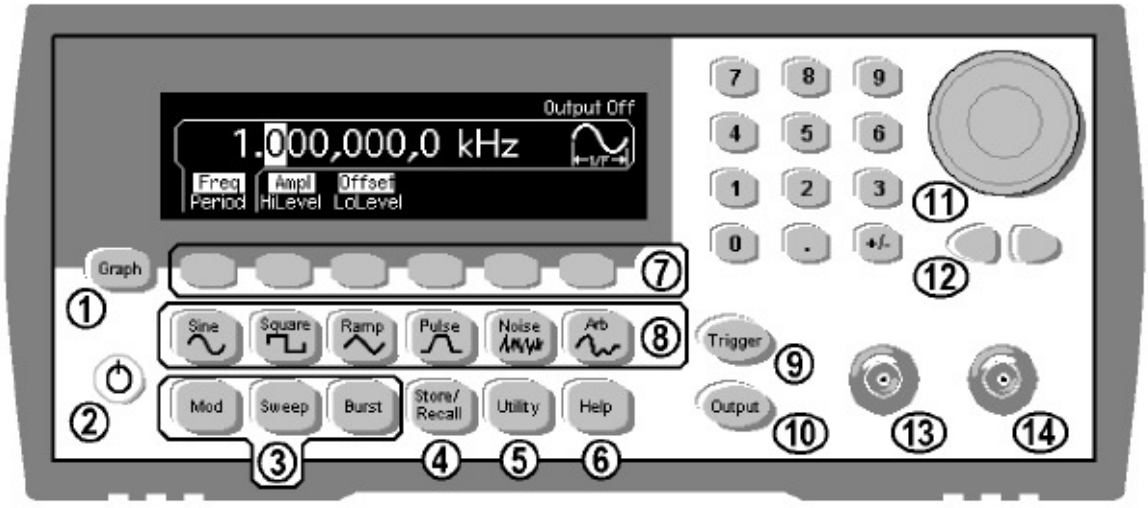
Neste projeto, as funcionalidades de carregamento de ondas arbitrárias não foram disponibilizadas para a automação do instrumento, visto que em experimentos laboratoriais simples, as formas de onda pré-definidas, listadas a seguir, são comumente utilizadas, sem a necessidade de se criar formas diferentes.

- Senóide (Sine);
- Quadrada (Square);
- Rampa (Ramp);
- Pulso (Pulse);
- Ruído (Noise);

- DC.

Entretanto, as funcionalidades de criação de ondas arbitrárias podem ser facilmente estendidas e adicionadas futuramente a este projeto.

As formas de onda e seus parâmetros e as configurações do Agilent 33220A podem ser alteradas manualmente a partir do painel frontal, que pode ser visto na figura a seguir:



1 Graph Mode/Local Key
2 On/Off Switch
3 Modulation/Sweep/Burst Keys
4 State Storage Menu Key
5 Utility Menu Key
6 Help Menu Key
7 Menu Operation Softkeys
8 Waveform Selection Keys
9 Manual Trigger Key (*used for Sweep and Burst only*)
10 Output Enable/Disable Key
11 Knob
12 Cursor Keys
13 Sync Connector
14 Output Connector

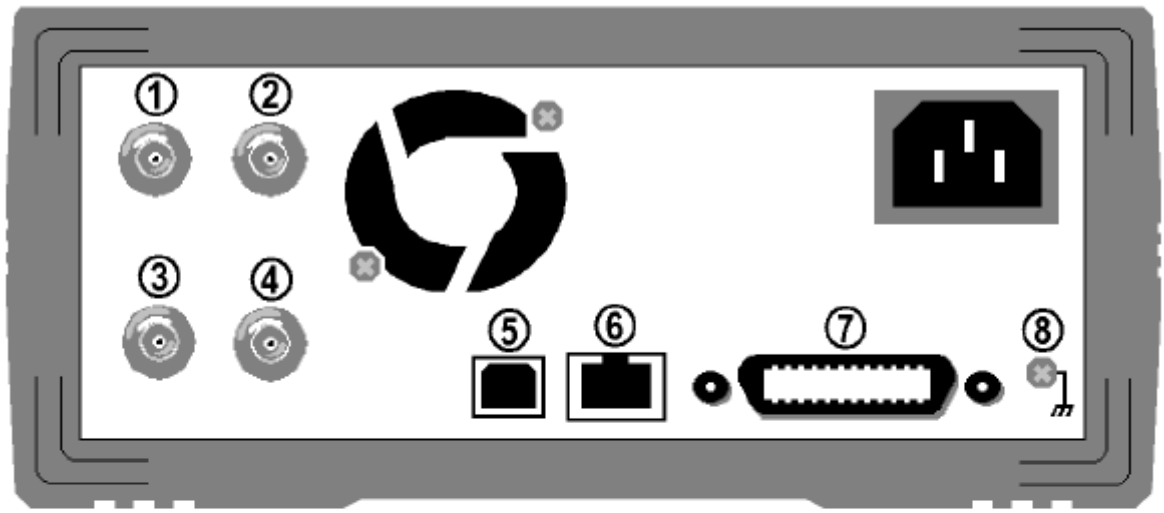
Ilustração 27 Painel Frontal Agilent 33220A

O gerador de sinais Agilent 33220A possui três tipos de interfaces para o seu controle remoto a partir de um computador desktop, que são:

- GPIB (IEEE-488);
- USB

- LAN

Os cabos dos respectivos tipos de interface podem ser conectados na parte traseira do gerador de sinais, que pode ser vista na figura a seguir. As interfaces GPIB e LAN devem ser configuradas antes de uma primeira conexão, escolhendo-se o endereço GPIB de conexão ou configurando os parâmetros de configuração da rede local para a conexão LAN.



1 External 10 MHz Reference Input Terminal (Option 001 only).
2 Internal 10 MHz Reference Output Terminal (Option 001 only).
3 External Modulation Input Terminal
4 Input: External Trig/FSK/Burst Gate
Output: Trigger Output
5 USB Interface Connector
6 LAN Interface Connector
7 GPIB Interface Connector
8 Chassis Ground

Ilustração 28 Painel Traseitro Agilent 33220A

A Agilent Technologies disponibiliza um conjunto de ferramentas computacionais e softwares para seus instrumentos, que tem como objetivo conectar e comunicar estes com computadores desktop. Uma dessas ferramentas é o Agilent IO Libraries Suíte, que tem a finalidade de facilitar ao usuário a criação e configuração de um ambiente de desenvolvimento de softwares para os instrumentos do fabricante, fornecendo bibliotecas de conexão e comunicação para todos os tipos de equipamentos da Agilent. Outro software que acompanha os instrumentos da Agilent Technologies é o Agilent Connection Expert, que

proporciona os testes de conexão e de envio de comandos ao hardware dos equipamentos, através de uma interface gráfica simples e amigável, podendo-se testar atividades antes de implementá-las em uma solução própria.

O gerador de sinais Agilent 33220A possui também um servidor web embarcado, o qual tem uma página web para o controle e configurações do instrumento. A página pode ser acessada através do endereço IP de rede do equipamento. As vantagens de se utilizar a página é que pode-se monitorar o instrumento de qualquer nó da rede, sem a necessidade de instalação de qualquer tipo de software, basta apenas possuir um navegador web. A página de configuração pode ser vista na figura a seguir, além dela, estão disponíveis páginas de controle, status e dúvidas.

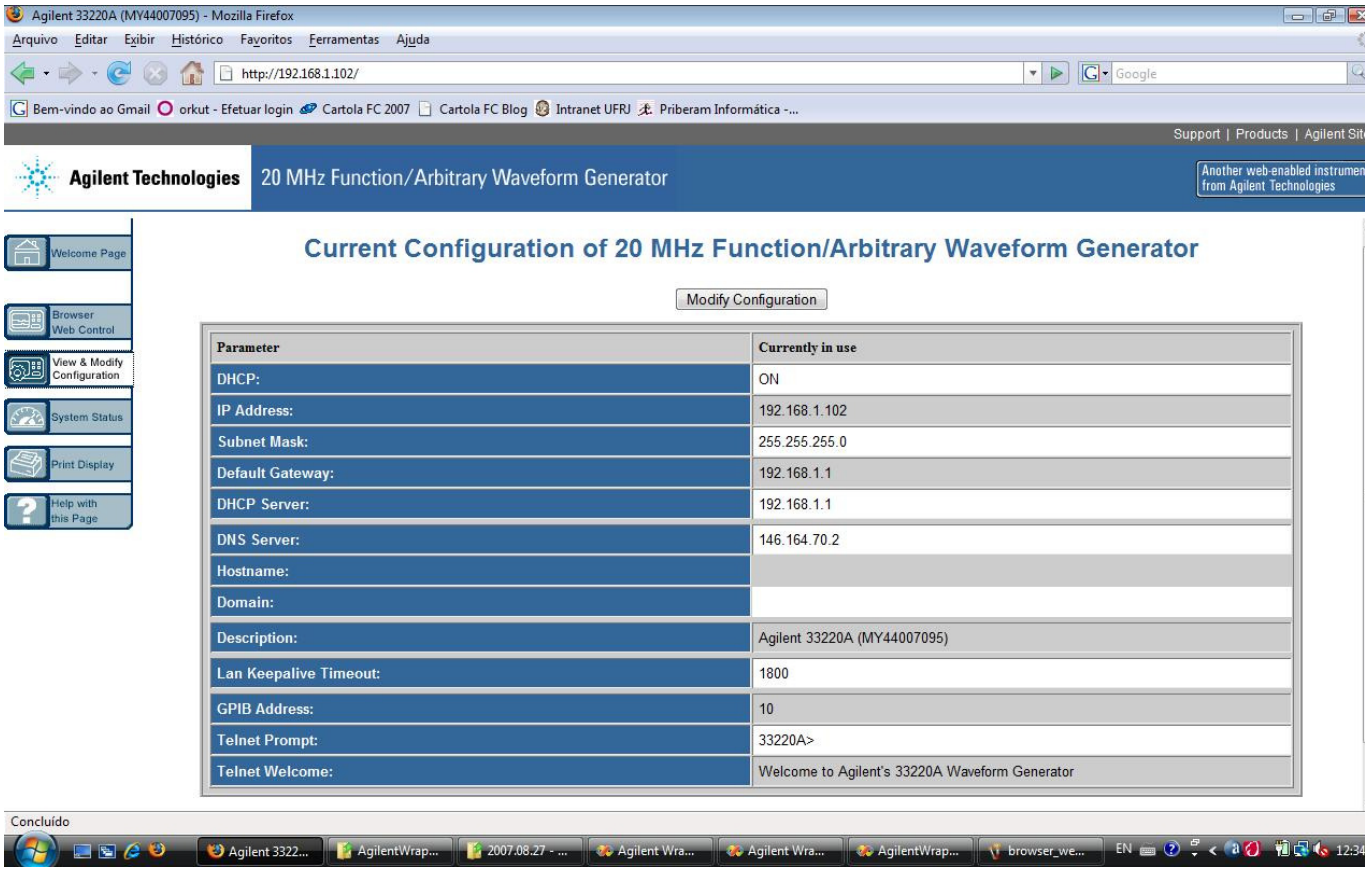


Ilustração 29 Página de configuração Agilent 33220A

Tais ferramentas apresentadas foram amplamente utilizadas para o projeto e a implementação do módulo de comunicação do Agilent 33220A com o Sistema de Monitoramento e Controle Laboratorial.

11.1.1 Protocolo de Comunicação

É possível comunicar-se com o hardware do gerador de sinais Agilent 33220A através de três tipos de protocolos para a comunicação: VXIplug&play, VISA-COM e VISA. Os três tipos são, atualmente, especificações da IVI Foundation. Essa fundação foi criada para promover especificações de programação de instrumentos de teste, que tem como finalidade facilitar a troca de equipamentos e a comunicação entre eles. A VXIplug&play Systems Alliance se fundiu à IVI Foundation em 2003, aumentando assim, o número de especificações controladas por elas.

O protocolo escolhido para ser utilizado neste projeto foi o VISA-COM, já que, a partir dele é possível conectar-se ao instrumento através da interface de conexão LAN. Essa escolha também se deve ao fato de que o osciloscópio Tektronix 3052B também possui a interface LAN e utiliza o protocolo de comunicação TekVisa, que é uma outra implementação do protocolo VISA-COM, com apenas algumas restrições.

A interface LAN também facilita a integração do instrumento em ambientes de rede distribuídos arbitrariamente, já que os laboratórios de pesquisa já possuem, comumente, redes internas, onde seus computadores e acessórios ficam dispostos à utilização. Portanto, cabos de redes e pontos de acesso estão, normalmente, disponíveis para o emprego dos instrumentos. Ao contrário das interfaces USB e GPIB, que necessitam de placas e cabos especiais para a sua utilização e não podem ser distribuídos, já que precisam ficar fisicamente próximo ao seu computador de controle.

11.1.2 Driver

A partir do protocolo de comunicação escolhido, seria possível implementar um driver para ele. Porém, o fabricante do gerador de sinais, a Agilent Technologies, já disponibiliza uma suíte com todos os drivers de todos os seus instrumentos. A suíte se chama Agilent IO Libraries Suíte, e foi realizado download de sua versão 14.2 no site oficial da empresa. Além disso, também foram feitos downloads de exemplos de utilização das bibliotecas de comunicação.

Portanto, a partir da suíte, foram realizados alguns testes de utilização das bibliotecas do fabricante para a comunicação com o gerador de sinais e posteriormente o seu encapsulamento, para serem utilizadas, apenas as funções necessárias a este projeto.

11.1.3 Wrapper

O wrapper, ou encapsulador, do driver teve de ser desenvolvido para disponibilizar apenas as funções necessárias ao projeto para o SLM do gerador de sinais.

O driver, por restrições do fabricante, foi disponibilizado apenas para ser utilizado em um ambiente de desenvolvimento do Visual Studio da Microsoft nas linguagens C++, C# e Visual Basic.

Portanto, o encapsulador foi escrito em C#, pela sua facilidade de aprendizagem e velocidade de implementação de novos aplicativos e interfaces gráficas. Utilizando-se o Visual Studio 2005 Pro, licenciado pelo programa MSDN Academic Alliance, firmado entre a UFRJ e a Microsoft.

Entretanto, o SLM do gerador de sinais foi escrito em Java, e para a integração com o wrapper, foi feita a comunicação entre eles por XML-RPC, que possibilita a integração entre aplicativos escritos em diferentes linguagens de programação. Para isso, foi utilizada a biblioteca XML-RPC.net para o envio e recebimento de chamadas XML-RPC em C#.

O wrapper executa funções através do driver, enviando-o um texto correspondente ao comando e seus parâmetros. Os comandos disponíveis deste encapsulador, assim como os comandos associados ao driver, estão listados na tabela a seguir:

Tabela 5 Comandos do Wrapper Agilent 33220A

Comando do wrapper	Comando do driver	descrição	Parâmetros	Possíveis parâmetros
WriteFunction	FUNC	Altera a forma de onda	Forma de onda a ser utilizada	SIN – para senóide
				SQU – para onda quadrada
				RAMP – para rampa
				PULS – para pulso
				NOIS – para ruído

				DC – para onda contínua
WriteFrequency	FREQ	Altera a freqüência do sinal	O valor da freqüência	Valor em ponto flutuante da freqüência
			Unidade de freqüência	Hz, KHz, MHz e assim por diante
WriteVoltage	VOLT	Altera o valor pico a pico da amplitude do sinal	Valor da amplitude	Valor em ponto flutuante da amplitude
WriteOffset	VOLT:OFFS	Altera o valor de offset do sinal	Valor de offset do sinal	Valor em ponto flutuante do offset, o padrão é 0V
ReadFunction	FUNC?	Lê a forma de onda do sinal de saída	Não tem parâmetros	Não tem parâmetros
ReadFrequency	FREQ?	Lê o valor da freqüência do sinal de saída	Não tem parâmetros	Não tem parâmetros
ReadVoltage	VOLT?	Lê o valor da amplitude do sinal de saída	Não tem parâmetros	Não tem parâmetros
ReadOffset	VOLT:OFFS?	Lê o valor de offset do sinal de saída	Não tem parâmetros	Não tem parâmetros

Durante o desenvolvimento do wrapper, foi construída uma classe no Visual Studio que realiza os testes unitários dos comandos, verificando sempre seus retornos e tratamentos de erros.

11.1.4 Variáveis de Sistemas

O SLM desenvolvido para o gerador de sinais possui quatro variáveis de sistema, que mapeiam as propriedades do sinal de saída do instrumento em que o usuário pode modificar. As variáveis de sistema são:

- Amplitude – corresponde à amplitude do sinal de saída, assumindo valores de ponto flutuante;
- Freqüência – corresponde à freqüência do sinal de saída, assumindo valores de ponto flutuante;
- Offset – corresponde ao valor de Offset do sinal de saída, assume valores de ponto flutuante;
- Função – corresponde à forma de onda do sinal de saída, assume um dos valores possível de forma de onda, SIN (senoidal), SQU (quadrada), RAMP (rampa), PULS (pulso), NOIS (ruído) ou DC (contínua).

Para cada variável de sistema foi criada uma TAG correspondente no banco de dados para o armazenamento histórico dos valores de cada uma. Ou seja, sempre que há alterações nos valores das variáveis de sistema, o SLM envia um evento ao TSC para que o novo valor seja gravado no banco de dados. Os valores históricos podem ser utilizados para gerar um relatório com todos os valores assumidos pela variável em um intervalo de tempo.

11.1.5 Operações Específicas

Para o SLM do gerador de sinais Agilent 33220A foram disponibilizadas apenas duas operações específicas (run_op). Tais operações realizam funções específicas do instrumento, executando chamadas ao driver do equipamento para a comunicação com seu hardware.

A primeira das operações, chamada de UpdateVariablesOperation, executa a atualização das variáveis de sistema do SLM. Não são necessários parâmetros para a sua chamada. O SLM efetua uma requisição dos valores de freqüência, amplitude e offset, além do tipo de forma de onda, atualizando os seus valores internamente e para o usuário.

A outra operação realiza alterações no sinal de saída do gerador, o nome dado a ela é ApplyOperation, que recebe como parâmetros os valores de freqüência, amplitude e

offset e também o tipo de forma de onda do sinal. Devem ser deixá-los em branco os campos que se desejar manter inalterado.

11.2 Osciloscópio digital Tektronix TDS 3052B

O osciloscópio digital TDS 3052B, ilustrado na figura a seguir, é fabricado pela empresa Tektronix, Inc. e faz parte da série de osciloscópios digitais 3000B. Suas principais funcionalidades são a captura de sinais, medição de sinais (amplitude, freqüência, entre outros) e a comparação de sinais (canal 1 e canal 2).

Ilustração 30 Tektronix TDS 3052B

Neste projeto, foram disponibilizadas apenas as principais funcionalidades de captura e visualização de um sinal e as medições de amplitude pico a pico e de freqüência. Já as funcionalidades como comparação de fases entre sinais e o carregamento de ondas arbitrárias (para fins de medição e comparação entre sinais) não foram disponibilizadas para a automação deste instrumento.

Entretanto, caso estas funcionalidades sejam necessárias futuramente, poderão ser acopladas facilmente ao sistema. Já que nenhum componente do TSC precisará ser alterado.

Os sinais de entrada e suas medidas podem ser visualizados na tela principal do osciloscópio. Podendo-se ajustar o sinal automaticamente através de seu botão autoset.

O osciloscópio digital Tektronix TDS 3052B possui dois tipos de interfaces para o seu controle remoto a partir de um computador desktop, que são:

- Serial DB-25, RS-232
- LAN

Os cabos dos respectivos tipos de interface podem ser conectados na parte traseira do equipamento. As interfaces devem ser configuradas antes de uma primeira conexão, escolhendo-se o endereço serial de conexão ou configurando os parâmetros de da rede local para a conexão LAN, como, por exemplo, mascara de rede, servidor de DNS, entre outros, para que as conexões ocorram com sucesso.

A Tektronix Inc., assim como a Agilent Technologies, disponibiliza um conjunto de ferramentas computacionais e softwares para seus instrumentos, que tem como objetivo conectar e comunicar estes com computadores desktop. A principal ferramenta gratuita da Tektronix é o OpenChoice Desktop, que permite ao usuário capturar as imagens das telas de um osciloscópio da Tektronix, além de poder controlá-lo. Porém, é uma solução fechada, não podendo ser integrada a instrumentos de outros fabricantes. Além disso, só pode ser executado em uma máquina com sistema operacional Windows.

Para a comunicação remota com seus instrumentos a partir de soluções desenvolvidas por terceiros, a Tektronix disponibiliza a biblioteca de programação TekVisa, que é uma implementação da especificação VISA-COM da IVI Foundation.

O osciloscópio digital Tektronix TDS 3052B também possui um servidor web embarcado, no qual pode-se acessar uma página web para o seu controle remoto, que facilita testes e aplicações de utilidade simples.

Tais ferramentas apresentadas foram amplamente utilizadas para o projeto e a implementação do módulo de comunicação do Tektronix TDS 3052B com o Sistema de Monitoramento e Controle Laboratorial.

11.2.1 Protocolo de Comunicação

O protocolo de comunicação utilizado pelo osciloscópio digital Tektronix TDS 3052B a partir da interface LAN é o TekVisa, que, como dito anteriormente, é uma implementação da especificação VISA-COM. Já o utilizado pela interface serial é o protocolo proprietário da Tektronix, no qual são enviados bytes no formato ASCII a partir do protocolo de transporte RS-232.

A interface de conexão remota escolhida foi a LAN, utilizando o protocolo de comunicação TekVisa, que como dito anteriormente, facilita em muito a integração do instrumento em ambientes de rede distribuídos arbitrariamente, principalmente em um ambiente heterogêneo, utilizando-se instrumentos de diferentes fabricantes, diferentes protocolos de comunicação e diferentes sistemas operacionais. Visto que, atualmente, todos os laboratórios já possuem uma rede privada, onde toda a infra-estrutura já está disponível para a utilização do sistema, sem a necessidade de esforços adicionais que outros tipos de interface teriam, como a criação de redes multidrop RS-232 ou RS-485, onde os conflitos entre diferentes de sinais e protocolos inviabilizam a sua utilização de diferentes tipos de equipamentos, ou seja, para isso seria necessário à criação de múltiplas redes, cada uma para cada tipo de equipamento.

11.2.2 Driver

Como o driver fornecido pela Tektronix para a comunicação com seus instrumentos é uma implementação da especificação VISA-COM, seria possível criar outra versão de seu driver, seguindo a especificação criada pela IVI Foundation. Entretanto, para facilitar ainda mais a integração dos instrumentos e a configuração dos ambientes de produção e desenvolvimento, será utilizada neste projeto a mesma implementação utilizada pela Agilent Technologies.

A implementação VISA da Agilent, possui algumas funções de redundância, controle interno e de tratamento de erros que o TekVisa não possui. Como por exemplo, a checagem de erros acumulados no equipamento, os instrumentos da Tektronix não fazem essa armazenagem. Já os da Agilent, gravam todos os erros não tratados pelo seu controlador, podendo ser requisitados ou apagados.

Portanto, a partir da mesma suíte de bibliotecas da Agilent, foram realizados testes de comunicação e controle remoto do osciloscópio digital Tektronix TDS 3052B. A partir do sucesso dos testes, foi desenvolvido um encapsulador, que faz chamadas ao driver para serem utilizadas apenas as funções necessárias a este projeto.

11.2.3 Wrapper

O wrapper, ou encapsulador, do driver teve de ser desenvolvido para disponibilizar apenas as funções necessárias ao projeto para o SLM do osciloscópio digital Tektronix TDS 3052B.

O driver utilizado para o desenvolvimento do Wrapper, como dito anteriormente, foi o mesmo do gerador de sinais Agilent 33220. Logo, o encapsulador para o equipamento da Tektronix também foi escrito em C#, em um projeto do Visual Studio 2005 PRO, separado do projeto do encapsulador desenvolvido para o gerador de sinais.

O wrapper do Tektronix TDS 3052B também executa suas funções através do driver, enviando-o um texto correspondente ao comando e seus parâmetros. Os comandos disponíveis deste encapsulador, assim como os comandos associados ao driver, estão listados na tabela a seguir:

Tabela 6 Comandos do Wrapper Tektronix TDS 3052B

Comando do wrapper	descrição	Comandos utilizados do driver	Descrição
ReadAmplitude	Lê a amplitude pico a pico do sinal de entrada	MEASUREMENT:IMMED:SOURCE1	Define o canal do sinal de entrada a ser utilizado
		MEASUREMENT:IMMED:TYPE	Define o tipo de medida a ser utilizado, no caso, amplitude pico a pico
		MEASUREMENT:IMMED:VALUE?	Comando utilizado para

			retornar o valor da medida
ReadFrequency	Lê a freqüência do sinal de entrada	MEASUREMENT:IMMED:SOURCE1	Define o canal do sinal de entrada a ser utilizado
		MEASUREMENT:IMMED:TYPE	Define o tipo de medida a ser utilizado, no caso, freqüência
		MEASUREMENT:IMMED:VALUE?	Comando utilizado para retornar o valor da medida
Autoset	Executa a mesma função do botão autoset do osciloscópio	AUTOSET EXECUTE	Executa a função de autoset do osciloscópio
ReadWaveForm	Lê a forma de onda do sinal de entrada do osciloscópio	DATA:SOURCE	Define o canal de entrada do osciloscópio a ser utilizado
		DATA:START	Define a posição do primeiro ponto a ser amostrado do

			sinal
		DATA:STOP	Define a posição do último ponto a ser amostrado do sinal
		DATA:WIDTH	Define a quantidade de bytes do tamanho de cada ponto amostrado do sinal
		WFMPRE:XUNIT	Define a unidade a ser utilizada no eixo X para a amostragem
		WFMPRE:YUNIT	Define a unidade a ser utilizada no eixo Y para a amostragem
		WFMPRE:XINCR	Determina o intervalo de amostragem do sinal
		CURVE?	Realiza a amostragem do sinal, retornando os valores de

			cada ponto amostrado

Durante o desenvolvimento do wrapper do Tektronix TDS 3052B também foi construída uma classe no Visual Studio que realiza os testes unitários dos comandos, verificando sempre seus retornos e tratamentos de erros.

11.2.4 Variáveis de Sistemas

O SLM desenvolvido para o osciloscópio digital Tektronix TDS 3052B possui apenas quatro variáveis de sistema, que mapeiam a amplitude e a freqüência dos sinais de entrada dos canais 1 e 2. O nome das variáveis de sistema são:

- Canal 1: Amplitude – Amplitude pico a pico do sinal de entrada do canal 1.
- Canal 1: Freqüência – Freqüência do sinal de entrada do canal 1.
- Canal 2: Amplitude – Amplitude pico a pico do sinal de entrada do canal 2.
- Canal 2: Freqüência - Freqüência do sinal de entrada do canal 2.

Para cada uma das variáveis de sistema foram criadas TAGs correspondentes no banco de dados para o armazenamento histórico dos valores de cada uma. Ou seja, sempre que há alterações nos valores das variáveis de sistema, o SLM envia um evento ao TSC para que o novo valor seja gravado no banco de dados. Os valores históricos podem ser utilizados para gerar um relatório com todos os valores assumidos pela variável em um intervalo de tempo.

11.2.5 Operações Específicas

Foram disponibilizadas três operações específicas (run_op) para o SLM do osciloscópio TDS Tektronix 3052B. As três operações realizam chamadas ao wrapper do driver do instrumento, visando atuar no experimento laboratorial.

A primeira das operações, chamada de UpdateVariablesOperation, executa a atualização das variáveis de sistema do SLM. Não são necessários parâmetros para a sua chamada. O SLM efetua uma requisição dos valores de freqüência e amplitude dos sinais de entrada dos canais 1 e 2.

A segunda operação, chamada AutosetOperation, executa a mesma função do botão autoset do painel do osciloscópio, ajustando automaticamente os sinais de entrada. A função também não recebe nenhum parâmetro.

A última operação, e a mais importante, captura as formas de onda do osciloscópio. A operação recebe como parâmetro o canal de entrada do sinal que se deseja capturar. O instrumento responde uma série de números, separados por vírgulas com os valores de cada ponto amostrado. O período de amostragem é pré-definido em uma constante do sistema. A série retornada é então salva pelo sistema, podendo-se plotar o gráfico correspondente na janela de resultados. O gráfico é gerado pelo plugin gráfico criado a partir dos pontos de extensão do Sistema de Monitoramento e Controle Laboratorial.

11.3 Estendendo funcionalidades

Para ambos os instrumentos não foram disponibilizadas todas as suas funções remotas, como, por exemplo, para o gerador de sinais Agilent 33220ª não foram disponibilizadas as funções de utilização de formas de onda arbitrarias e para o osciloscópio Tektronix 3052B foram não dispobilizadas as funções de configuração remota dos cursores horizontal e vertical.

Porém, caso seja necessário, é possível agregar aos SLMs do gerador de sinais e osciloscópio, as funções restantes. Primeiramente é necessário mapear as funções desejadas no manual do fabricante, que fornece as maneiras de acesso à todas as funções de cada instrumento. Feito isso, é necessário implementar as funções mapeadas no Wrapper do instrumento correspondente, dessa maneira, o SLM poderá ter acesso às funções desejadas. Então, por fim, as funções deverão ser escritas no padrão da LECIS no arquivo DCD do módulo correspondente.

Portanto, as funcionalidades não implementadas dos instrumentos, acima descritos, podem ser facilmente agregadas ao sistema devido à estrutura já desenvolvida.

Capítulo 12

12 Conclusão

A partir do sistema desenvolvido neste projeto é possível automatizar experimentos laboratoriais simples, que utilizam geradores de sinais e osciloscópios. Porém, com a abordagem da LECIS, pode-se agregar, facilmente, novos instrumentos ao sistema, devido à especificação de uma interface genérica de comunicação entre o controlador (TSC) e os módulos laboratoriais (SLM).

A automatização laboratorial proporcionada pelo Sistema de Controle e Monitoramento Laboratorial possibilita também a execução remota de experiências laboratoriais com fins acadêmicos, tornando possível, assim, o ensino à distância de práticas laboratoriais.

Outra prática importante de automatização laboratorial é a automatização de testes de circuitos, entre outros tipos de equipamentos. O sistema poderia ser utilizado para testes que utilizam diversos tipos de instrumentos e de diferentes fabricantes ou que precisam alterar constantemente os tipos de instrumentos utilizados, proporcionando desta maneira, a utilização constante de uma mesma plataforma de testes.

A escolha por ferramentas e softwares open-source reconhecidos garante que qualquer problema encontrado futuramente possa ser resolvido pela comunidade responsável ou pelo próprio desenvolvedor do sistema. Com a permissão de acessar e modificar qualquer parte dos códigos fonte, o desenvolvedor poderá consertar bugs e até modificar o sistema de forma que o mesmo atenda melhor o seu objetivo. Uma outra vantagem das ferramentas open-source é o desenvolvimento sem gastos em compras de licença.

A única ferramenta não open-source utilizada foi o Visual Studio 2005 Pro, da Microsoft Corporation, apesar disso, o seu único emprego foi o desenvolvimento dos encapsuladores dos drivers dos instrumentos Agilent 33220A e o Tektronix TDS 3052B. Porém, esta escolha foi efetuada, única e exclusivamente, por restrições dos fabricantes, que disponibilizaram suas ferramentas apenas para esta plataforma de desenvolvimento.

O controlador do Sistema de Monitoramento e Controle Laboratorial é multiplataforma, podendo ser executado em Linux, Windows e em outros sistemas operacionais que possuam uma máquina virtual Java da Sun Microsystems, desta forma pode-se economizar com os custos de licenças de sistemas operacionais, o que torna o sistema ainda mais flexível.

Já os módulos dos instrumentos laboratoriais podem ser desenvolvidos em qualquer linguagem de programação e estar em qualquer nó da rede local, rodando em um sistema operacional arbitrário. Logo, pode-se utilizar quaisquer ferramentas e plataformas de desenvolvimento, em que o desenvolvedor tenha mais aptidão ou que não necessite da compra de novas licenças. Facilitando também a integração dos diferentes tipos de instrumentos laboratoriais e fabricantes, que em muitas vezes disponibilizam a comunicação remota de seus instrumentos para plataformas específicas.

Portanto, a partir do acima exposto, conclui-se que o sistema desenvolvido é uma ferramenta de baixo custo, fácil manutenção, flexível e extensível, podendo ser utilizada para diferentes tipos de experiências laboratoriais e diferentes propósitos.

Apêndice A

A Desenvolvimento de um SLM em Java

Para o desenvolvimento de um SLM em Java, deve-se possuir uma máquina virtual Java da Sun Microsystems, Inc. instalada no sistema operacional. Para isso, é necessário o download e instalação da JRE 1.5, a qual é oferecida no site oficial da Sun, http://www.sun.com.

Além disso, o desenvolvedor de um novo SLM escrito em Java para o Sistema de Controle e Monitoramento Laboratorial deverá possui a versão 3.2 da plataforma de desenvolvimento Eclipse, assim como os plugins e projetos desenvolvidos neste projeto.

A partir disto, o ambiente de desenvolvimento estará completo e para o desenvolvimento do novo SLM deverá ser seguido o passo a passo abaixo, obedecendo à ordem estabelecida.

1. Criar um novo projeto Java no Eclipse, que deverá ter como dependência os projetos: common (para utilização das classes e interfaces a serem estendidas), e xmlrpc (para comunicação XML-RPC).
2. Criar o arquivo DCD do SLM a ser desenvolvido. O arquivo deverá declarar principalmente quais métodos serão disponibilizados e quais variáveis de sistemas serão declaradas.
3. Criar a camada de acesso ao instrumento físico podendo-se criar o driver propriamente dito ou apenas uma camada lógica de encapsulamento de um driver já existente. Esta camada é denominada “Device Wrapper” e deverá ser integrada com as outras camadas do SLM. Aconselha-se a criar um novo projeto no Eclipse para o Device Wrapper.

4. Criar uma nova classe para o modulo, estendendo a classe abstrata SLMModule. Essa classe fará a implementação específica dos métodos definidos pela LECIS. A classe abstrata declara os métodos da LECIS a serem implementados por esta classe criada. Esta classe deverá fazer chamadas ao Device Wrapper, para a implementação de comportamentos específicos do instrumento laboratorial. Não é necessário implementar a validação e o controle de estados, eles serão feitos automaticamente por outra classe.
5. Criar uma nova classe OperationFactory, que implementa a interface IOperationFactory, esta classe é a responsável pela fabricação dos objetos das classes que implementam as operações específicas do SLM. Ou seja, ao se executar um comando run_op definido pela LECIS, a classe OperationFactory será responsabilizada por executar a operação específica correta.
6. Criar as operações específicas do SLM. Para cada operação deverá ser criada uma nova classe, a qual estenderá a classe abstrata Operation. Estendendo-se essa classe, deverá ser necessário implementar os métodos básicos de execução de operações específicas de instrumentos de laboratório. Para cada classe criada deverá ser necessário também atualizar a classe OperationFactory com a operação correspondente.
7. Criar uma nova classe SLMFacade, que estende a AbstractSLMFacade. Esta classe deverá apontar para a classe que implementa a classe abstrata SLMModule. A classe AbstractSLMFacade faz toda lógica geral definida pela LECIS, como a máquina de estados, por exemplo.
8. Inserir no banco de dados as TAGs que o módulo desenvolvido utilizará.
9. Criar o arquivo de build (deployment) do SLM, o build deverá utilizar a tecnologia Ant, que cria os arquivos Jar necessários contendo os arquivos .class das classes do SLM. Em apêndice pode ser observado como executar os módulos escritos em Java gerados pelo Ant.
10. Se necessário, deverá ser criado um plugin para o TSC para a visualização dos dados históricos das TAGs criadas para o SLM. Este item está mais detalhado no capítulo do TSC.
11. Se necessário, deverá ser criado um plugin para o TSC para a visualização gráfica dos resultados gerados pelas operações específicas criadas para o SLM. Este item está mais detalhado no capítulo do TSC.

Apêndice B

B Criação de plugins de visualização de resultados

O seguinte passo a passo poderá ser utilizado como base para o desenvolvimento de plugins de visualização de resultados específicos para cada tipo de instrumento laboratorial, ou seja, cada SLM deverá contribuir com um plugin para que seus resultados sejam exibidos corretamente pelo TSC.

1. Criar um novo projeto no Eclipse, utilizando o wizard de criação de um novo projeto de plugin para o Eclipse. O plugin deverá ter como dependência o plugin principal do TSC (app), assim como os plugins a serem utilizados para a criação de gráficos.
2. Adicionar uma nova Extension, escolhendo-se o Extension Point ShowResult. A propriedade name do elemento da Extension deverá ter o nome do SLM do resultado correspondente.
3. Criar um nova classe que estende a classe abstrata SLMShowResults. A classe deverá implementar o método createControls, preenchendo o Composite dado com a representação gráfica do resultado obtido pelo SLM. Os campos do resultado, assim como seu valor, podem ser requeridos através do atributo protected resultEntry, sendo possível requerer o status, nome da operação executada, data de execução, nome do procedimento, entre outros.
4. Criado o plugin, ele deverá ser colocado como dependência do produto do TSC, dessa maneira ele será carregado quando for necessária a visualização de um resultado específico do SLM implementado.

Apêndice C

C Criação de plugins para gráficos de relatórios

O seguinte passo a passo poderá ser utilizado como base para o desenvolvimento de plugins de visualização de dados históricos de módulos laboratoriais. O plugin criado será gerenciado pelo subsistema de relatórios do TSC.

1. Criar um novo projeto no Eclipse, utilizando o wizard de criação de um novo projeto de plugin para o Eclipse. O plugin deverá ter como dependência o plugin principal de relatórios (report) e o plugin de gerenciamento de banco de dados (database), assim como os plugins a serem utilizados para a criação de gráficos.
2. Adicionar uma nova Extension, escolhendo-se o Extension Point graphViewControl. A propriedade name do novo elemento da Extension deverá ser o tipo da TAG.
3. Criar um nova classe que implementa a interface IGraphViewPartControl. A classe deverá implementar o método createGraphViewPartControl, preenchendo o Composite dado com a representação gráfica dos dados históricos da TAG. A identificação da TAG, a data de início e a data fim são passados como parâmetros do método, de modo a poder se obter os dados necessários do banco de dados.
4. Criado o plugin, ele deverá ser colocado como dependência do produto do TSC, dessa maneira ele será carregado quando for necessária a visualização de gráficos históricos de TAG's do tipo implementado.

Apêndice D

D Administração do Banco de Dados do Sistema

Os seguintes procedimentos de administração do sistema de gerenciamento de banco de dados (SGBD) MySQL 5.0 poderão ser utilizados para a manutenção do Sistema de Monitoramento e Controle Laboratorial.

Procedimento de Instalação e Configuração do Banco de Dados

1. Instalação do MySQL

A instalação do MySQL pode ser feita em um dos sistemas operacionais disponíveis para a versão 5.0, basta realizar o download do instalador correspondente no site oficial do MySQL: http://dev.mysql.com/downloads/mysql/5.0.html.

Em um ambiente Linux, que possui o comando apt-get, pode-se realizar o download e instalação do MySQL através da linha de comando: apt-get install mysql-server-5.0.

2. Configuração de Senha

Para uma maior segurança do sistema, recomenda-se configurar a senha do administrador do sistema. Para isso execute o seguinte comando no shell do MySQL:

```
SET PASSWORD FOR user@host=PASSWORD('password');
```

Exemplo de uma aplicação do comando:

```
SET PASSWORD FOR root@localhost=PASSWORD('senha123');
```

3. Configuração de Acesso ao Banco

Para que outras máquinas possam acessar o banco de dados, deve-se executar o seguinte comando:

```
grant all privileges on *.* to 'user'@'%' identified by 'password';
```

Exemplo de uma aplicação do comando:

```
grant all privileges on *.* to 'root'@'%' identified by 'senha123';
```

4. Configuração do Tipo de Acesso Externo ao Banco

Para que o acesso ao banco possa ser feito via socket externo, em um banco instalado em uma máquina Linux, é necessário configurar o seguinte arquivo /etc/mysql/my.cnf, para isso é necessário modificar o arquivo, trocando apenas as linhas apresentadas abaixo:

- skip-external-locking
- #skip-networking

Procedimentos de Importação e Exportação da Base de Dados

O MySQL possui diversas ferramentas de administração do banco, umas delas é o mysqldump, que possibilita a criação de backups das bases de dados de um sistema. A criação de backups a partir do mysqldump é completamente configurável, podendo-se realizar o backup de todo sistema ou de apenas tabelas ou campos específicos.

Para se exportar as bases de dados, é possível executar o seguinte comando em um terminal do sistema operacional:

mysqldump -u "usuário" --password="senha" -c -t -e --databases "banco 1" "banco 2" > "nome do arquivo"

Exemplo de uma aplicação do comando:

mysqldump -u root --password=senha123 -c -t -e --databases base_de_dados > backup.sql

Para se importar as bases de dados de um arquivo de backup, deve-se executar o seguinte comando em um terminal do sistema operacional:

```
mysql -u "usuário" -p --database="nome do banco" < "nome do arquivo"
```

Exemplo de uma aplicação do comando:

```
mysql -u root -p < backup.sql
```

É recomendado que se execute o procedimento de backup periodicamente, para que dados históricos não sejam perdidos.

Bibliografia

AGILENT. Agilent Technologies, http://www.home.agilent.com/, Acesso em 14 de Outubro de 2007.

ASTM LECIS. E 1989-98 (2004), American Society for Testing and Materials.

BIRT. Overview, http://www.eclipse.org/birt/phoenix/intro/, Acesso em 26 de Outubro de 2007.

CLAYDBERG, Erich, RUBEL, Dan. Eclipse: Building Commercial – Quality Plug-ins. 2. Ed. Addison Wesley Professional, 2006.

DAUM, Berthold. Professional Eclipse 3 for Java™ Developers. Wrox, 2004.

ECLIPSE RCP. Página oficial, http://www.eclipse.org/rcp, Acesso em 5 de Outubro de 2007.

GEF, Descrição sobre GEF, http://eclipsewiki.editme.com/GefDescription, Acesso em 5 de Outubro de 2007.

GAMMA, Erich, BECK, Kent. Contributing to Eclipse: Principles, Patterns, and Plug-Ins. Addison Wesley, 2003

HOLZNER, Steve. Eclipse Cookbook. O'Reilly, 2004.

HTTP. Informações sobre HTTP, http://pt.wikipedia.org/wiki/Http, Acesso em 13 de Outubro de 2007.

IVI. Interchangeable Virtual Instrument Foundation, http://www.ivifoundation.org/, Acesso em 14 de Outubro de 2007.

JFREECHART. Biblioteca JFreeChart, http://www.jfree.org/jfreechart/, Acesso em 14 de Outubro de 2007.

LECIS. Página oficial, http://www.lecis.org/about.htm, Acesso em 5 de Outubro de 2007.

LEMIEUX, Jean-Michel; MCAFFER, Jeff. Eclipse Rich Client Platform. U.S.A.. Addison-Wesley. 2006.

MCAFFER, Jeff, LEMIEUX, Jean-Michel. Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications. Addison Wesley, 2005.

MSDN. Microsoft Developer Network, http://msdn2.microsoft.com/pt-br/default.aspx, Acesso em 14 de Outubro de 2007.

MYSQL. Manual MySQL, http://dev.mysql.com/doc/refman/5.0/en/, Acesso em 15 de Novembro de 2007.

OMG LECIS. E 2002, Object Management Group, Inc.

SQLEXPLORER. Projeto SQLExplorer, http://eclipsesql.sourceforge.net/, Acesso em 15 de Novembro de 2007.

SULLIVAN, Sean Sullivan. Advanced DAO programming. http://www.ibm.com/developerworks/library/j-dao/, Acesso em 15 de Novembro de 2007.

SUN. Sun Microsystems, Inc., http://www.sun.com, Acesso em 15 de Novembro de 2007.

TCL. Biblioteca Tcl Httpd, http://tclhttpd.sourceforge.net/, Acesso em 9 de Outubro de 2007.

TCL. Biblioteca Tcl Soap, http://tclsoap.sourceforge.net/, Acesso em 9 de Outubro de 2007.

TCL. Tcl Developer Xchange, http://www.tcl.tk/, Acesso em 9 de Outubro de 2007.

TEKTRONIX. Tektronix Inc., http://www.tek.com/, Acesso em 14 de Outubro de 2007.

VXIPLUG&PLAY. VXIplug&play Systems Alliance, http://www.vxipnp.org/, Acesso em 14 de Outubro de 2007.

WARNER, Rob. The Definitive Guide to SWT and JFace. U.S.A.. Apress. 2004.

XML-RPC. Biblioteca XML-RPC para plataforma .NET, http://www.xml-rpc.net/, Acesso em 13 de Outubro de 2007.

XML-RPC. Especificação, http://www.xmlrpc.com/spec, Acesso em 13 de Outubro de 2007.

XML-RPC. Informações sobre XML-RPC, http://en.wikipedia.org/wiki/XML-RPC, Acesso em 13 de Outubro de 2007.