Documentacion Tecnica

1. Requerimientos de software.

2. Descripción general.

3. Módulos

4. Código Fuente.

1. Requerimientos de software.

Sistema Operativo: linux.

Bibliotecas utilizadas:

gtkmm y glademm para la interfaz gráfica.
Libxml2.0 para el manejo del XML.

Herramientas necesarias para compilar: make.

2. Descripción general

Los distintos módulos con los que cuenta la aplicación son:

Modelo: es el módulo central de la aplicación, permite crear archivos en formato postscript y se encarga de la persistencia.

Editor: se encarga de la interfaz gráfica, permite crear y editar plantillas.

Servidor: es el que realiza el servicio SOAP/HTTP. Tiene las clases necesarias para escuchar conexiones, recibir peticiones HTTP-SOAP, procesarlas y devolver una respuesta HTTP-SOAP.

3. Módulos

Módulo Modelo.

Clases

Clase Cálculo: clase que hereda de texto y contiene los atributos necesarios para poder dibujarse (posición inicial, color, tipo y tamaño de letra).

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Esta clase es la encargada de realizar la carga de los elementos del cálculo en una lista.
Recibe un string que contiene el cálculo, parsea la cuenta para que pueda realizarse en notación polaca inversa, que es lo que maneja postscript y luego carga los operandos y operadores de esa forma en una lista.

Por ejemplo si recibe "5 + ((1 + 2) * 4) - 3" carga en la lista "5 1 2 + 4 * 3 - +".

Métodos principales:

agregarElemento: agrega un elemento de cálculo a la lista.

obtenerElementos: recibe el string con el cálculo a realizar, lo parsea y carga la lista de elementos. Devuelve true en el caso de que no hubo errores.

reiniciarConteo: reinicia la lista de elementos de cálculo.

cantidadCorrecta: revisa si la cantidad de operandos y la operadores es válida para realizar el cálculo. Devuelve true en caso correcto.

Accept: método utilizado para dibujar en postscript con el patrón visitor.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos del cálculo.

Utiliza la clase ***ParseadorRPN***, encargada de parsear el cálculo a notación polaca inversa.

Método principal:

shuntingYard: recibe el string del cálculo y lo parsea a notación polaca inversa utilizando el algoritmo que lleva el nombre del método.

Clase Columna: representa una columna que contiene un encabezado (texto fijo), luego una celda cuerpo donde puede ir cualquier tipo de texto (fijo, variable o calculo) y por último contiene una lista de textos adicionales.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Método principal:

agregarTextosAdicionales: agrega un texto a lista de textos adicionales.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los elementos de la columna.

Clase DocTemplate: representa un documento con diferentes versiones de templates. Si se abre en sólo lectura los cambios no se guardan al cerrarse. Permite acceder a las versiones en forma secuencial o por fecha.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Métodos principales:

isOpen: devuelve true si se logro abrir y cargar el documento
saveAs: guarda el DocTemplate si es que se no se abrio en modo soloLectura.

getVersion(Fecha&f): Devuelve un template en el cual la fecha pasada por parámetro pertenezca al período de válidez. Si hay varios devuelve el primero. Si no existe un template que cumpla con el requisito devuelve NULL.

getVersion(unsigned int num = 0): Devuelve el primer Template de la lista. Si no existe ninguna versión devuelve NULL

Método para la persistencia:

toXml: devuelve un xml que contiene todos los elementos del documento.

Clase ElementoDeCalculo: clase abstracta de la cual heredan operador, operandoConstante y operandoVariable.

Clase Fecha: clase para manejar las fechas de las versiones de los documentos.

Método principal:

pertenece: recibe dos fechas fechas (inicial y final) y revisa si está contenida la versión en ese período, y en ese caso devuelve true.

Clase GeneradorPostScript: clase encargada de realizar la creación de archivos postscript. Se pueden crear todos las versiones a imprimir en distintos archivos o en un único archivo.

Método principal:

procesar: recibe un bool que dice si se crea un único archivo o no y se procede a la generación de archivos. En caso de que el xml no tenga información y no se creen los archivos postscript se devuelve false.

Clase Hoja: clase que hereda de objetoGraficoRectangular, contiene los atributos necesarios para imprimirse. Representa la hoja donde se dibujará la factura. Contiene una lista donde se guardan todos los objetos que contiene.

Contiene la clase ***XmlNodo*** para manejar la persistencia

Método principal:

agregarObjeto: permite agregar cualquier tipo de objeto grafico a la lista.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos de la hoja.

Clase Linea: clase que hereda de objetoGrafico, contiene los atributos necesarios para imprimir una línea horizontal, vertical u oblicua.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Métodos principales: getteres y setters de sus atributos.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos de la línea.

Clase ObjetoGrafico: clase de la que heredan todos los objetos soportados en el editor. Contiene los atributos mínimos para poder dibujarse un objeto.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Métodos principales: getters y setters de sus atributos.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos de la del objetoGrafico.

Clase ObjetoGraficoRectangular: clase que hereda de objetoGrafico. Posee los atributos para dibujarse.

Contiene la clase XmlNodo para manejar la persistencia.

Métodos principales: getters y setters de sus atributos.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos del objetoGrafico.

Clase ObjetoLista: clase que hereda de texto y permite dibujar una lista que puede contener distintos tipos de textos (texto fijo, variable o calculo). La lista se compone de columnas.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Método principal:

agregarColumna: agrega una columna a la lista.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos de la lista.

Clase Operador: clase que hereda de ElementoDeCalculo y se encarga de crear el operador para realizar el calculo.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Método principal:

getContenido: devuelve un string con el valor del operador, en este caso puede ser add, sub, mul y div.

Método para la persistencia:

toXml: devuelve un xml que contiene el operando.

Clase OperandoConstante: clase que hereda de ElementoDeCalculo y contiene un operando que es número.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Método principal:

getValor: devuelve el número que es el operando.

Método para la persistencia:

toXml: devuelve un xml que contiene el operando.

Clase OperandoVariable: clase que hereda de ElementoDeCalculo y contiene un operando que surge del archivo xml de datos y se busca en el VisitorPostscript de acuerdo a la ruta xpath.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Método principal:

getContenido: devuelve un string con el valor del xpath.

Método para la persistencia:

toXml: devuelve un xml que contiene el xpath del operando.

Clase ParseadorDeArgumentos: clase encargada de revisar y obtener los argumentos desde la línea de comandos.

Método principal:

parsear: se encarga de controlar si los argumentos son correctos y en caso de que sea en modo línea obtiene el puerto.

Clase PostScript: clase que contiene todos los métodos para dibujar en lenguaje postscript.

Métodos principales:

moveto: para setear la posicipon en donde dibujar.

setrgbcolor: para setear el color.

Setlinewidth: para setear el borde.

truncarTexto: para que cuando un texto ocupa más espacio que el permitido se muestre hasta el último renglón dentro del recuadro.

crearRecuadro: para bordear el texto.

imprimirStack: para mostrar el resultado de las operaciones hechas en el stack.

Clase Template: Clase que representa un único template. Posee una fecha de validez y un objetoGrafico que es la hoja principal.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos del template

Clase Texto: clase que hereda de objetoGraficoRectangular y contiene los atributos mínimos para poder dibujar un texto.

Contiene la clase ***XmlNodo*** para manejar la persistencia.

Métodos principales: getters y setters.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos del texto.

Clase TextoFijo: clase que hereda de texto y contiene un texto.

Métodos principales: getters y setters.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos del texto fijo.

Clase TextoVariable: clase que hereda de texto y contiene una ruta xpath que contiene un texto.

Métodos principales: getters y setters.

Método para la persistencia:

toXml: devuelve un xml que contiene todos los atributos del texto fijo.

Clase Visitor: clase abstracta que contiene los métodos accept que serán redefinidos en el visitorPostscript.

Clase VisitorPostScript: clase encargada de recorrer todos los objetos gráficos y generar lenguaje postScript.

Clases: ***Xml***, ***XmlNodeSet*** y ***XmlNodo*** encapsulan las funciones de la librería libxml2.0.

Módulo Servidor:

Clases

Clase Server: se encarga de recibir conexiones a travéz de un puerto y crear un objeto del tipo AtencionCliente para atender esa conexión.

Clase AtencionCliente: Recibe una petición HTTP y valida que la petición sea correcta de acuerdo al protocolo HTTP 1.0. Si la petición es una petición HTTP-SOAP válida, instancia un objeto de tipo Web-Service que se encargará de procesar la petición.Atención cliente, devuelve una respuesta HTTP-SOAP y cierra la conexión.

Clase WebService: Es la clase encargada de "ejecutar" la petición SOAP.Primero verifica que la petición tenga el formato
correcto de acuerdo al wsdl (ver apéndice B), luego, de acuerdo al valor de los parámetros genera el archivo postscript si es necesario y construye la respuesta SOAP.

Clase AtenciónCliente:

Métodos de alto y bajo nivel para el envío y recepción de mensajes a travéz de un socket. Permite enviar y recibir peticiones y respuestas HTTP procesando los bytes recibidos, obteniendo el campo content-lenght y de acuerdo a eso leyendo el cuerpo del mensaje.

Módulo Editor - GUI (editor de plantillas)

El Editor Gráfico esta implementado mediante el patrón MVC (Modelo-Vista-Controlador). Por cada ObjetoGrafico existe un ObjetoGui(modelo), una Figura(Vista) y ésta tiene un Comportamiento(Controlador). El patrón es implementado mediante Observador-Observer.

Figura

La clase Figura es una interfaz que tiene toda la lógica de dibujo y representación de un ObjetoGui en la pantalla. Sus dos métodos más importantes son:

draw: Dibuja la Figura en un área de dibujo.
getComportamiento(): Devuelve un comportamiento en el cual se delegan las acciones que realiza el usuario.

imagen: FiguraComportamientoGeneral

ObjetoGui

La clase ObjetoGui es una adaptación del modelo que permite acceder a los atributos de cada objeto gráfico en forma de lista. Muy útil para representarlos en la pantalla y permitir su edición.En general siempre tiene una referencia a su par en el modelo.Su principal método es:

getAtributos(std::list<FormatoAtributo>& listaDeAtributos): Devuelve una lista de atributos del objeto. Cada FormatoAtributo tiene un control para que pueda ser modificado.

imagen: GuiModeloAtributos

Comportamiento

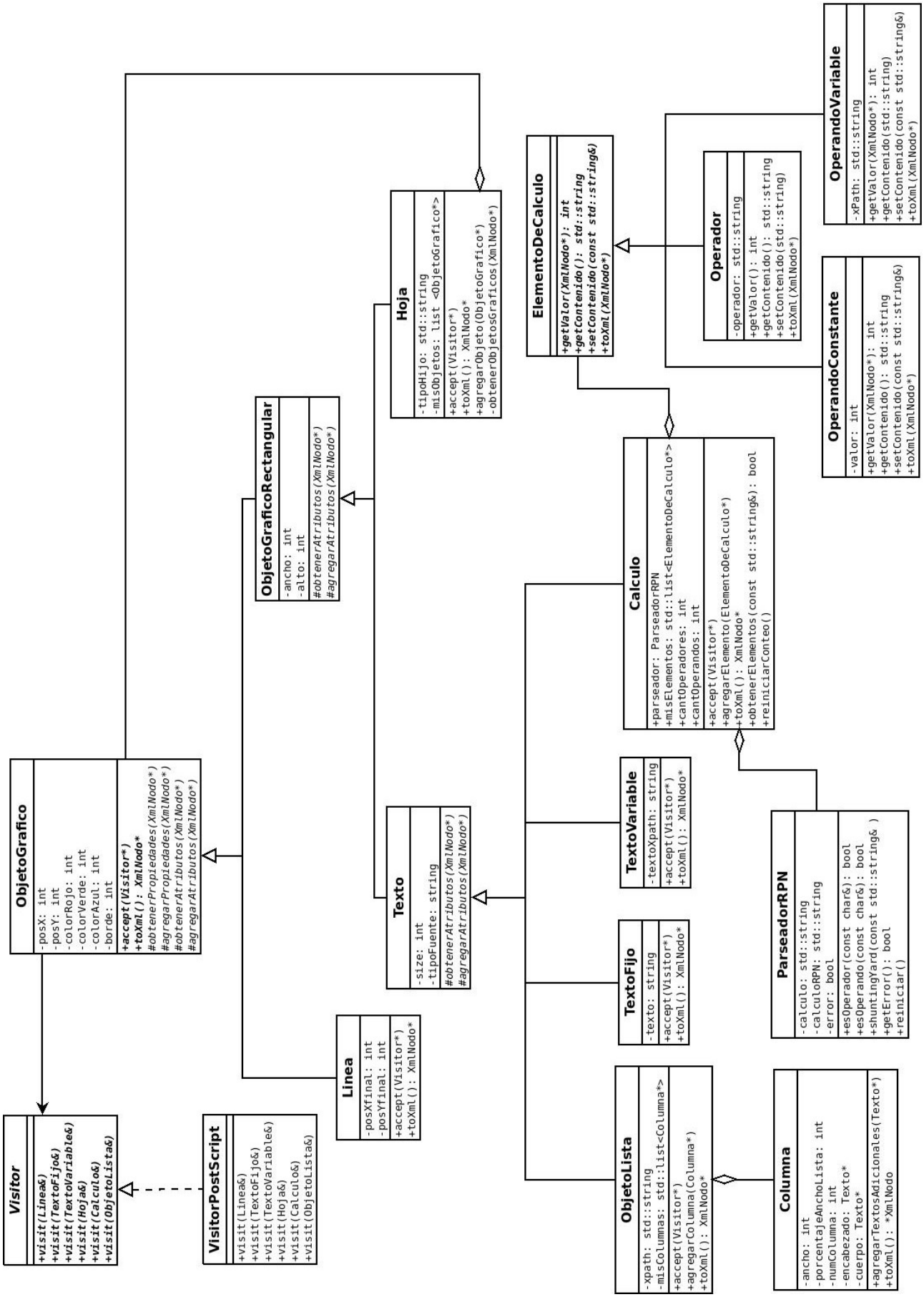
El Comportamiento es una interfaz que responde a los eventos de button_press, button_release y motion_notify.
El área de dibujo, delega estos eventos en el objeto lo que le permite variar su comportamiento en forma dinámica.

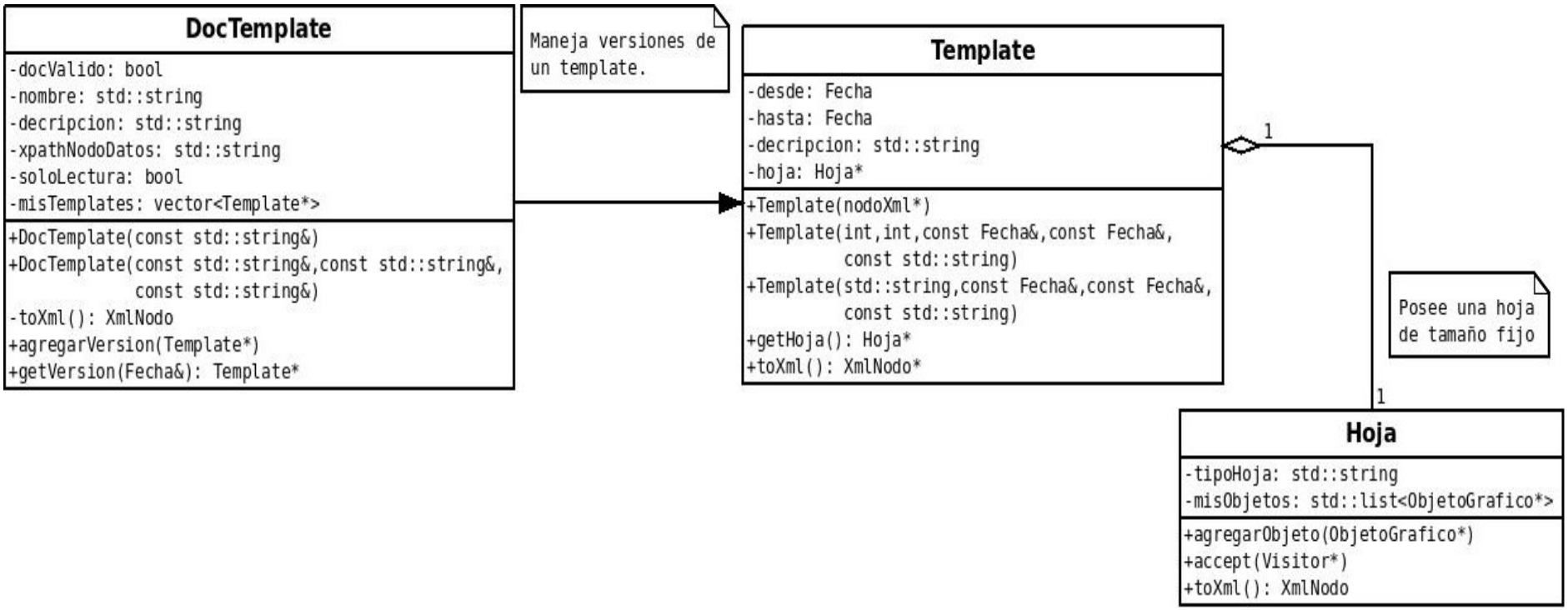
MyArea

El Area funciona como Area de dibujo, contenedor de Figuras y ObjetosGui.Se encarga de delegar sus eventos a la clase Comportamiento de acuerdo al caso.
Es Capaz de recibir un objeto Hoja del modelo, construir los ObjetosGui y Figuras de esa hoja para representar una plantilla.
El Área, tiene un ComportamientoSeleccion.Este Comportamiento es particular, ya que detecta que objeto se encuentra seleccionado en el editor y delega el comportamiento en ese objeto.

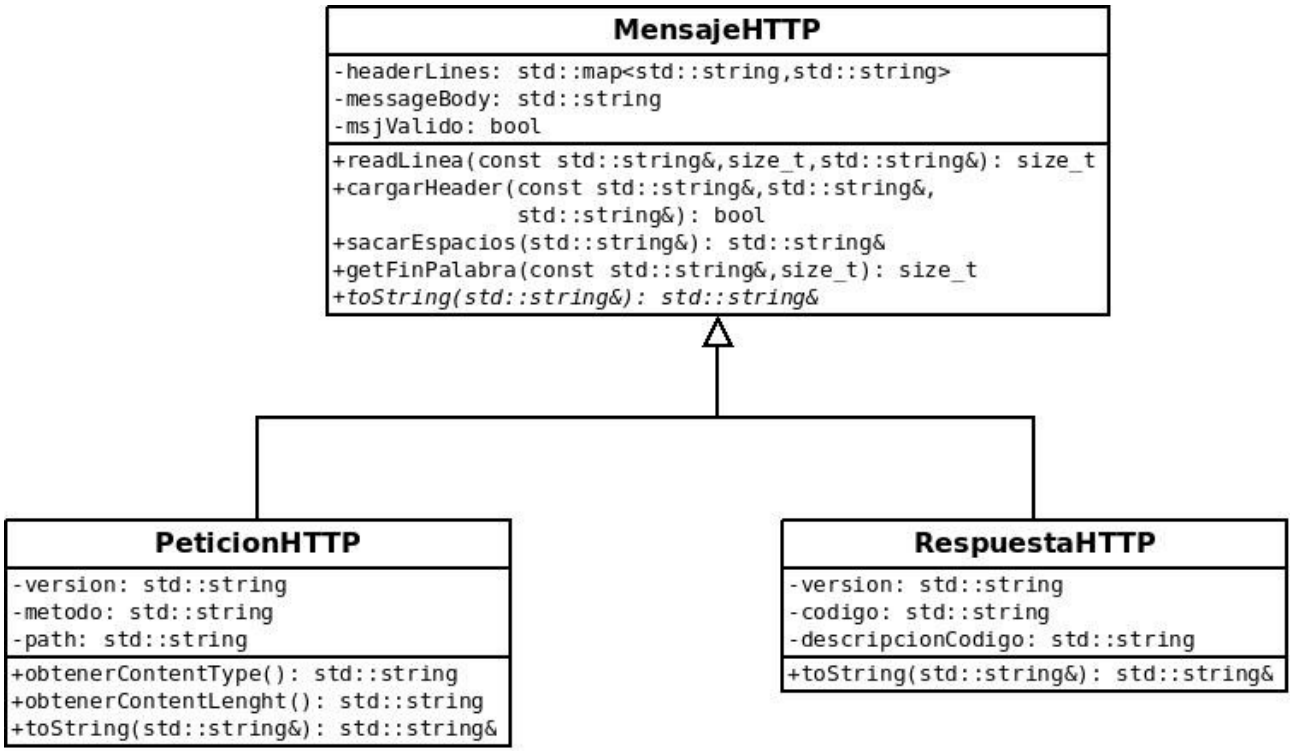
imagen:FiguraComportamientoSeleccion

Diagramas de clase
Modelo

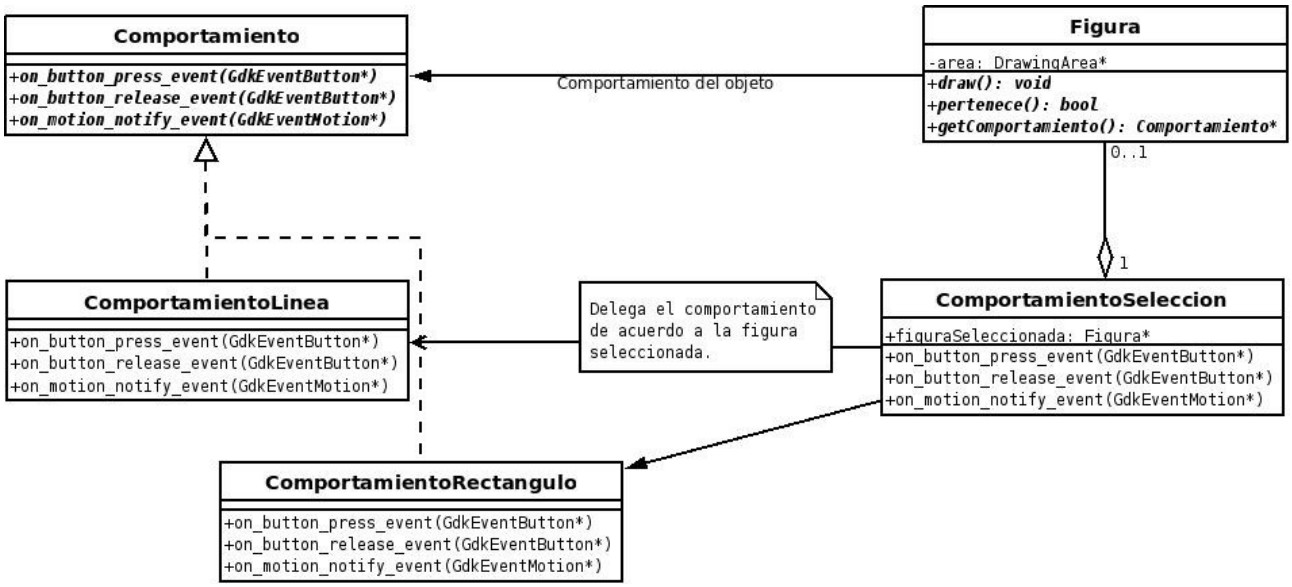



DocTemplate
-docValido: bool
-nombre: std::string
-decripcion: std::string
-xpathNodoDatos: std::string
-soloLectura: bool
-misTemplates: vector<Template*>
+DocTemplate(const std::string&)
+DocTemplate(const std::string&,const std::string&,
const std::string&)
-toXml(): XmlNodo
+agregarVersion(Template*)
+getVersion(Fecha&): Template*
Maneja versiones de un template.
Template
-desde: Fecha
-hasta: Fecha
-decripcion: std::string
-hoja: Hoja*
+Template(nodoXml*)
+Template(int,int,const Fecha&,const Fecha&,
const std::string)
+Template(std::string,const Fecha&,const Fecha&,
const std::string)
+getHoja(): Hoja*
+toXml(): XmlNodo*
1
1
Posee una hoja de tamaño fijo
Hoja
-tipoHoja: std::string
-misObjetos: std::list<ObjetoGrafico*>
+agregarObjeto(ObjetoGrafico*)
+accept(Visitor*)
+toXml(): XmlNodo

Server


MensajeHTTP
-headerLines: std::map<std::string,std::string>
-messageBody: std::string
-msjValido: bool
+readLinea(const std::string&,size_t,std::string&): size_t
+cargarHeader(const std::string&,std::string&,
std::string&): bool
+sacarEspacios(std::string&): std::string&
+getFinPalabra(const std::string&,size_t): size_t
+toString(std::string&): std::string&
PeticionHTTP
-version: std::string
-metodo: std::string
-path: std::string
+obtenerContentType(): std::string
+obtenerContentLenght(): std::string
+toString(std::string&): std::string&
RespuestaHTTP
-version: std::string
-codigo: std::string
-descripcionCodigo: std::string
+toString(std::string&): std::string&

Editor


Comportamiento
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)
Comportamiento del objeto
Figura
-area: DrawingArea*
+draw(): void
+pertenece(): bool
+getComportamiento(): Comportamiento*
0..1
1
ComportamientoLinea
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)
Delega el comportamiento de acuerdo a la figura seleccionada.
ComportamientoSeleccion
+figuraSeleccionada: Figura*
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)
ComportamientoRectangulo
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)

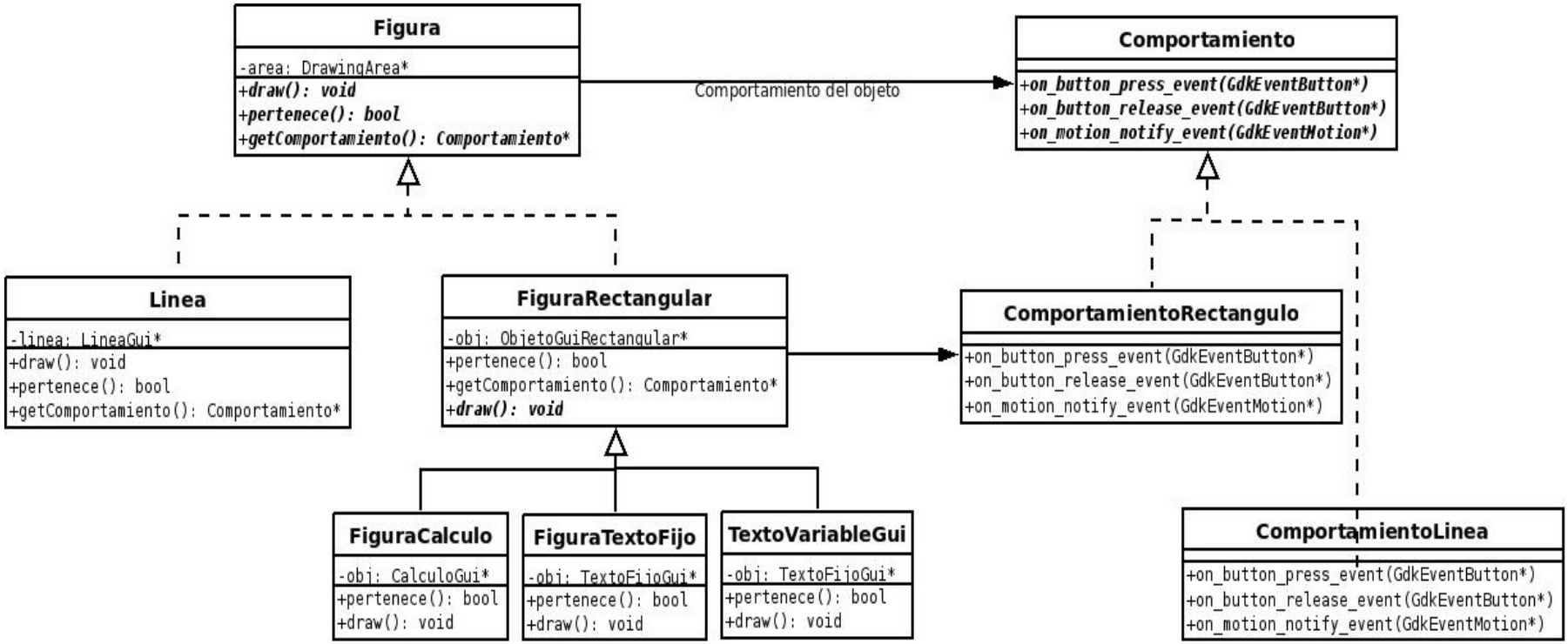


Figura
-area: DrawingArea*
+draw(): void
+pertenece(): bool
+getComportamiento(): Comportamiento*
Comportamiento del objeto
Comportamiento
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)
Linea
-linea: LineaGui*
+draw(): void
+pertenece(): bool
+getComportamiento(): Comportamiento*
FiguraRectangular
-obj: ObjetoGuiRectangular*
+pertenece(): bool
+getComportamiento(): Comportamiento*
+draw(): void
ComportamientoRectangulo
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)
FiguraCalculo
-obj: CalculoGui*
+pertenece(): bool
+draw(): void
FiguraTextoFijo
-obj: TextoFijoGui*
+pertenece(): bool
+draw(): void
TextoVariableGui
-obj: TextoFijoGui*
+pertenece(): bool
+draw(): void
ComportamientoLinea
+on_button_press_event(GdkEventButton*)
+on_button_release_event(GdkEventButton*)
+on_motion_notify_event(GdkEventMotion*)

Apéndice A

Formato de los mensajes de Comunicación

El servicio SOAP funcionará sobre el protocolo HTTP versión 1.0.Recibirá las peticiones HTTP-SOAP mediante el método POST.Las posibles respuestas de error HTTP implementadas en nuestro servidor son las siguientes:

"400 Bad Request" : Formato inválido en la petición HTTP.
"501 Not Implemented" : Método distinto a POST en la petición.
"505 HTTP Version Not Supported" : Vesión diferente a HTTP 1.0
"411 Length Required": La petición no especifica el tamaño del contenido del mensaje.

Es decir que nuestro servidor HTTP, recibe solo mensajes tipo POST, version HTTP1.0 y que tenga el campo Content-Length especificando el tamaño del contenido.

Todos los errores relacionados al formato de la petición SOAP retornan un mensaje con error genérico "500 Internal Server Error".Incluyendo el caso en que el campo fechaValidez tenga un formato inválido.

Formatos de petición y respuesta

<crlf> es el fin de linea del protocolo http, correspondiente a los caracteres \r y \n.

Formato de petición HTTP-SOAP

POST http://localhost:5000/ HTTP/1.0

Accept-Encoding: gzip,deflate

Content-Type: text/xml;charset=UTF-8

User-Agent: Jakarta Commons-HttpClient/3.1

Host: localhost:5000

Content-Length: 269

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <nombreTemplate>?</nombreTemplate>
    <fechaValidez>?</fechaValidez>
    <archivoDatos>?</archivoDatos>
  </soapenv:Body>
</soapenv:Envelope>
```

El contenido del mensaje tiene la petición SOAP de nuestro WEb-Service.Se deben especificar tres parámetros:

nombreTemplate: Nombre de la plantilla para generar archivos postscript.En nuestra implementación el servidor
busca el archivo con ese nombre dentro de su directorio.

fechaValidez: Fecha de validez de los datos.El servidor soporta solo el formato "dd/mm/aaaa".

archivoDatos: Archivo Xml de datos.El wsdl lo define como un string.En la petición debe ser colocado con el
formato <![CDATA["archivo Xml"]]>.

Formato de respuesta HTTP-SOAP

HTTP/1.0 200 OK

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <codeReturn>0</codeReturn>
    <nombreArchivoResultado>Factura.ps</nombreArchivoResultado>
  </soapenv:Body>
</soapenv:Envelope>
```

Una respuesta HTTP-SOAP tiene un campo codeReturn que specifica el resultado del procesamiento de la la petición. Los posibles valores son:

SOAP_OK 0 : Se generó satisfactoriamente el archivo postscript y se devuelve en el parámetro "nombreArchivoResultado" el nombre del archivo generado. El archivo contiene una hoja por cada elemento del archivo de datos.

SOAP_SIN_VERSION 1: Existe una plantilla pero no contiene ninguna version correspondiente a la fecha de validez pasada por parámetro.

SOAP_SIN_TAG_DATOS 2: Existen una plantilla y una versión válidas ,pero el archivo de Datos no tiene ningún nodo con el nombre especificado en la plantilla.

SOAP_SIN_TEMPLATE 3: No existe un archivo de plantilla con ese nombre en el sistema.

Archivo WSDL del web-service

```
?xml version="1.0" encoding="UTF-8"?>
<definitions name="toPostScriptService"
  targetNamespace="http://www.tpFinal.com/wsdl/toPostScriptService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.tpFinal.com/wsdl/toPostScriptService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="toPostScriptRequest">
    <part name="nombreTemplate" type="xsd:string"/>
    <part name="fechaValidez" type="xsd:date"/>
    <part name="archivoDatos" type="xsd:string"/>
  </message>
  <message name="toPostScriptResponse">
    <part name="nombreArchivoResultado" type="xsd:string"/>
    <part name="cantidadDeArchivos" type="xsd:int"/>
    <part name="codeReturn" type="xsd:int"/>
  </message>
  <portType name="toPostScript_PortType">
    <operation name="toPostScript">
      <input message="tns:toPostScriptRequest"/>
      <output message="tns:toPostScriptResponse"/>
    </operation>
  </portType>

  <binding name="toPostScript_Binding" type="tns:toPostScript_PortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="toPostScript">
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="literal"/>
      </input>
      <output>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="toPostScript_Service">
    <documentation>WSDL File for toPostScript</documentation>
    <port binding="tns:toPostScript_Binding" name="toPostScript_Port">
      <soap:address
        location="http://localhost:8080/toPostScript"/>
    </port>
  </service>
</definitions>
```

4. Código Fuente.