Modificação de um plug-in ST para Terraview utilizando Terralib 5

Trabalho SER 300

Aluno Diego Monteiro

1 INTRODUÇÃO

1.1 Descrição do problema

Cada dia temos acesso a um maior número de dados espaciais, sejam eles imagens ou medições, sejam em formato vetorial ou matricial. E esses dados são utilizados comumente no dia-a-dia seja para planejar férias em família, como para descobrir o melhor caminho para o trabalho.

Os modelos atuais de SIG (sistema de informações geográficas) trabalham de forma harmoniosa com dados estáticos no eixo do tempo, porém para a visualização de mudanças em um determinado ambiente, ou a movimentação de um objeto apenas começam a surgir modelos para visualizar esses dados no eixo do tempo.

Monitorar uma área especifica e analisar mudanças ao longo do tempo é fundamental para por exemplo prevenir o desmatamento de uma floresta, ou conhecer a expansão de uma cidade.

1.2 Motivação

Tempo é essencial, todo evento que ocorre ocorre tanto em um lugar no tempo quanto no espaço, e representar essa diferença no tempo é um desafio.

Para realizar tais representações são desenvolvidos modelos, o surgimento de novos modelos, estimula a criação de protótipos para valida-los

2 OBJETIVO

2.1 Objetivo geral

Propõe-se modificar um *plugin* para a observação de *Coverage Series* para o programa Terraview, utilizando a biblioteca Terralib 5.

2.2 Objetivos Específicos

Para se atingir o objetivo geral, etapas intermediárias se fazem necessárias:

(a) Criar um layer de CoverageSeries na TerraLib 5/TerraView 5
(b) Visualização dinâmica desse layer -> usando e propondo melhorias no slider já existente
(c) Extração de séries temporais.

3 FUNDAMENTOS E PROCESSO

3.1 Coverage Series

Segundo Ferreira *et al.* podemos dividir obervações espaço temporais em 3 tipos, series temporais, trajetorias e coverages. Coverages, que são tratadas neste trabalho, são dados obtidos quando se determina um espaço fixo, controla-se o tempo e mede-se uma terceira variavel de interesse. Como por exemplo o nivel de clorofila de um lago, ou como mostrado na figura 1 a precipitação pluviometrica em uma determinada região.

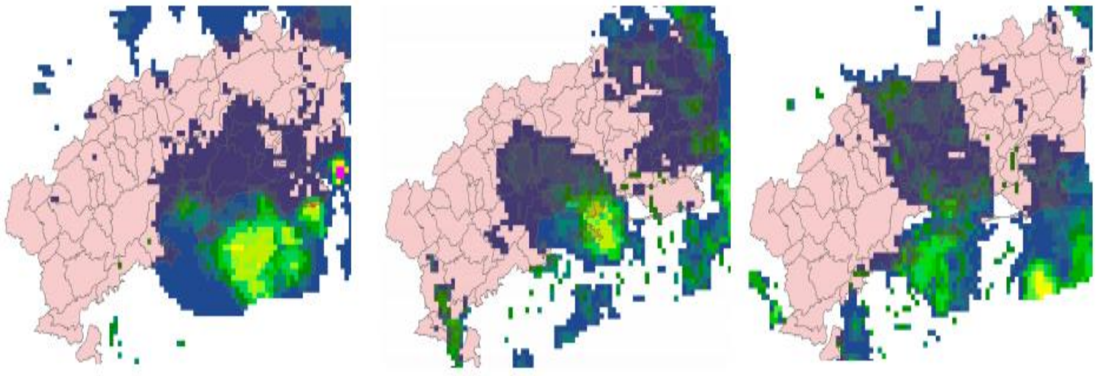


Figura 1 Examplo de coverages: chuva no estado de Rio de Janeiro, Brasil, em 2011

3.2 Terralib 5

Terralib é uma biblioteca de *software* SIG de código aberto, que permite o desenvolvimento e *customização* de aplicações geográficas.

Para esse trabalho está biblioteca foi obtida através da página *wiki*, da DPI (divisão de processamento de imagens) do INPE

Para ser compilada esta biblioteca precisa de um ambiente especifico montado no computador, que a receberá.

3.2.1 Ambiente

Para o desenvolvimento deste trabalho, instalou-se o ambiente recomendado pela documentação obtida junto ao código da biblioteca terralib5, que são os seguintes, para o sistema operacional *Windows*:

- Visual Studio 2010
- CMake 3.2.2
- Qt 5.x

O ambiente necessário foi instalado a primeira vez, porém não houve êxito na compilação da biblioteca, posteriormente observou-se que o sistema já possuía várias versões dos *softwares* Visual Studio e Qt, que provavelmente estivessem em conflito com os requisitos necessários, optou-se por formatar a máquina e instalar novamente os requisitos, desta vez obtendo êxito na compilação.

Outros softwares de terceiros foram utilizados também porém estes vieram em um pacote zip, pré-montado no site da wiki, supõe-se que estes softwares estavam em ordem pois foram utilizados ambas as vezes.

3.3 Terraview

Terraview é um SIG desenvolvido pela DPI do INPE. A principal característica do TerraView é a manipulação de dados vetoriais e matriciais. O TerraView permite a criação de mapas temáticos com variados tipos de legendas.

3.4 Aprendizagem

Para o desenvolvimento foi necessário aprender programação em C++, foi necessário aprender a desenvolver interface gráficas utilizando Qt e foi necessário realizar o estudo do modelo ST proposto por Ferreira *et al.*, e estudar o plug-in já existente anteriormente.

4 Desenvolvimento

Para melhor compreensão do relacionamento entre as classes em C++ utilizadas para o desenvolvimento, tentou-se montar um diagrama de classes simples. E também um diagrama de estados para abstrair como o usuário utilizaria o plug-in implementado. O diagrama de classes pode ser visualizado na figura 2 e o diagrama de estados pode ser encontrado na figura 3.

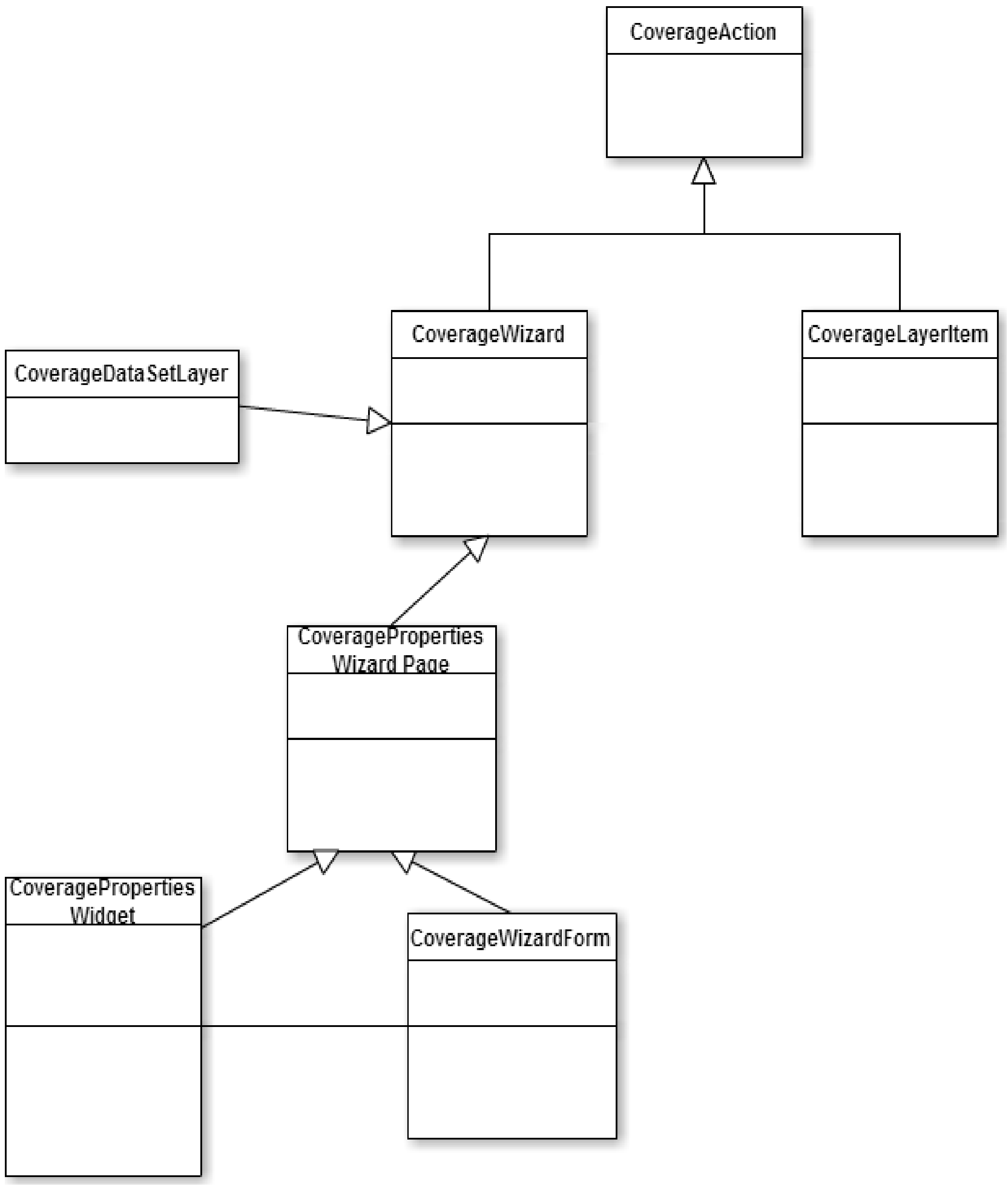


Figura 2 Diagrama de classes

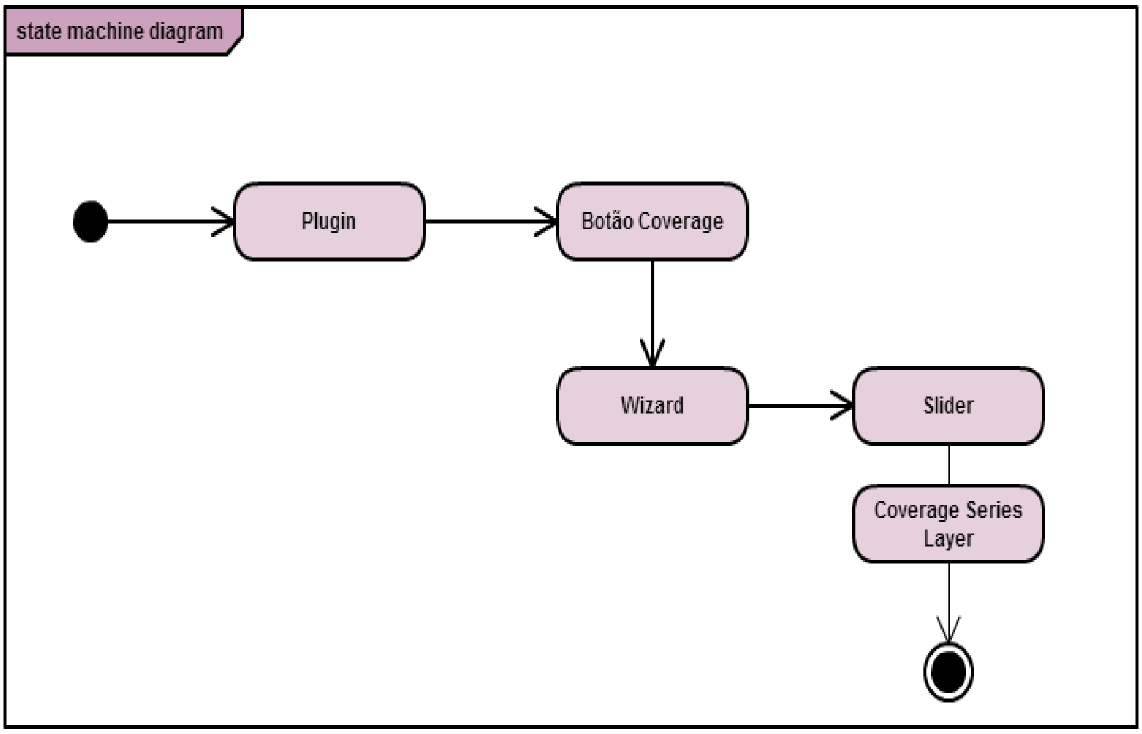
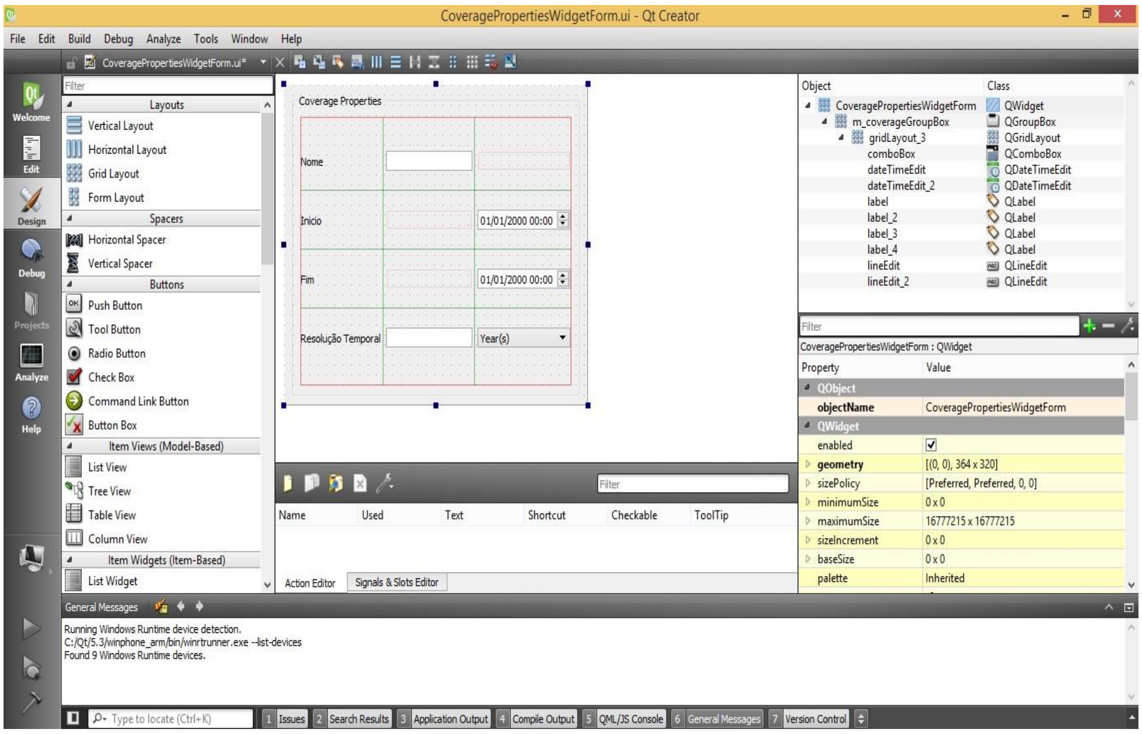


Figura 3 Diagrama de estados

No desenvolvimento inicialmente, foi criado o *form* no programa Qt como é visto na figura 4



Object	Class
CoveragePropertiesWidgetForm	QWidget
m_coverageGroupBox	QGroupBox
gridLayout_3	QGridLayout
comboBox	QComboBox
dateTimeEdit	QDateTimeEdit
dateTimeEdit_2	QDateTimeEdit
label	QLabel
label_2	QLabel
label_3	QLabel
label_4	QLabel
lineEdit	QLineEdit
lineEdit_2	QLineEdit

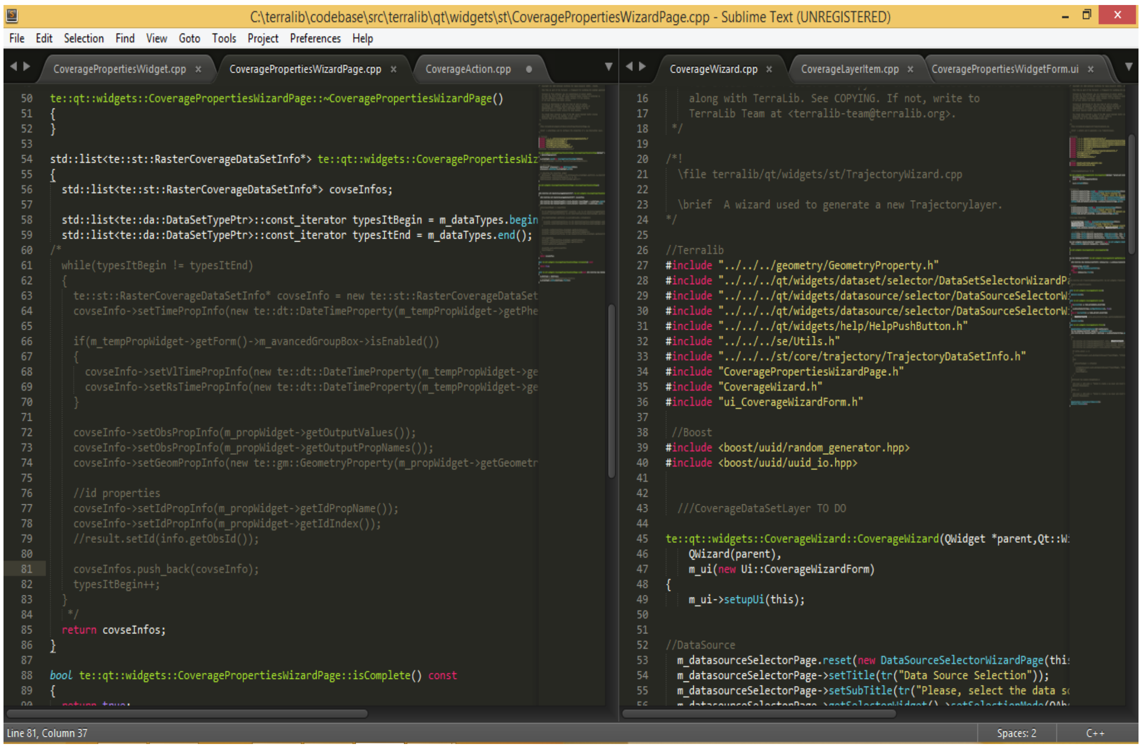
CoveragePropertiesWidgetForm : QWidget

Property	Value
QObject	
objectName	CoveragePropertiesWidgetForm
QWidget	
enabled	✓
geometry	[(0, 0), 364 x 320]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited

```
Running Windows Runtime device detection.
C:/Qt/5.3/winphone_arm/bin/winrtrunner.exe --list-devices
Found 9 Windows Runtime devices.
```

Figura 4 Criação do Form para Coverage Series

Utilizou-se o programa sublimeText para as alterações no form e edição das classes que se baseavam em classes pré-existentes um exemplo de seu funcionamento pode ser visto na figura 5, a lista de classes editadas segue abaixo e em anexo as classes em si podem ser encontradas.


C:\terralib\codebase\src\terralib\qt\widgets\st\CoveragePropertiesWizardPage.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
CoveragePropertiesWidget.cpp
CoveragePropertiesWizardPage.cpp
CoverageAction.cpp
CoverageWizard.cpp
CoverageLayerItem.cpp
CoveragePropertiesWidgetForm.ui
Line 81, Column 37
Spaces: 2
C++

```cpp
te::qt::widgets::CoveragePropertiesWizardPage::~CoveragePropertiesWizardPage()
{
}

std::list<te::st::RasterCoverageDataSetInfo*> te::qt::widgets::CoveragePropertiesWiz
{
  std::list<te::st::RasterCoverageDataSetInfo*> covseInfos;

  std::list<te::da::DataSetTypePtr>::const_iterator typesItBegin = m_dataTypes.begin
  std::list<te::da::DataSetTypePtr>::const_iterator typesItEnd = m_dataTypes.end();
/*
  while(typesItBegin != typesItEnd)
  {
    te::st::RasterCoverageDataSetInfo* covseInfo = new te::st::RasterCoverageDataSet
    covseInfo->setTimePropInfo(new te::dt::DateTimeProperty(m_tempPropWidget->getPhe

    if(m_tempPropWidget->getForm()->m_avancedGroupBox->isEnabled())
    {
      covseInfo->setVlTimePropInfo(new te::dt::DateTimeProperty(m_tempPropWidget->ge
      covseInfo->setRsTimePropInfo(new te::dt::DateTimeProperty(m_tempPropWidget->ge
    }

    covseInfo->setObsPropInfo(m_propWidget->getOutputValues());
    covseInfo->setObsPropInfo(m_propWidget->getOutputPropNames());
    covseInfo->setGeomPropInfo(new te::gm::GeometryProperty(m_propWidget->getGeometr

    //id properties
    covseInfo->setIdPropInfo(m_propWidget->getIdPropName());
    covseInfo->setIdPropInfo(m_propWidget->getIdIndex());
    //result.setId(info.getObsId());

    covseInfos.push_back(covseInfo);
    typesItBegin++;
  }
  */
  return covseInfos;
}

bool te::qt::widgets::CoveragePropertiesWizardPage::isComplete() const
{
```

```cpp
    along with TerraLib. See COPYING. If not, write to
    TerraLib Team at <terralib-team@terralib.org>.
 */

/*!
  \file terralib/qt/widgets/st/TrajectoryWizard.cpp

  \brief  A wizard used to generate a new Trajectorylayer.
*/

//Terralib
#include "../../../geometry/GeometryProperty.h"
#include "../../../qt/widgets/dataset/selector/DataSetSelectorWizardPa
#include "../../../qt/widgets/datasource/selector/DataSourceSelectorW
#include "../../../qt/widgets/datasource/selector/DataSourceSelectorW
#include "../../../qt/widgets/help/HelpPushButton.h"
#include "../../../se/Utils.h"
#include "../../../st/core/trajectory/TrajectoryDataSetInfo.h"
#include "CoveragePropertiesWizardPage.h"
#include "CoverageWizard.h"
#include "ui_CoverageWizardForm.h"

 //Boost
#include <boost/uuid/random_generator.hpp>
#include <boost/uuid/uuid_io.hpp>


  ///CoverageDataSetLayer TO DO

te::qt::widgets::CoverageWizard::CoverageWizard(QWidget *parent,Qt::W
    QWizard(parent),
    m_ui(new Ui::CoverageWizardForm)
{
    m_ui->setupUi(this);


//DataSource
  m_datasourceSelectorPage.reset(new DataSourceSelectorWizardPage(thi
  m_datasourceSelectorPage->setTitle(tr("Data Source Selection"));
  m_datasourceSelectorPage->setSubTitle(tr("Please, select the data s
```

Figura 5 Código sendo editado

Lista de classes editadas:

- Plugin.cpp;
- CoveragePropertiesWidget.cpp;
- CoveragePropertiesWizardPage.cpp;
- CoverageAction.cpp;
- CoverageWizard.cpp;
- CoverageLayerItem.cpp;
- Plugin.h;
- CoveragePropertiesWidget.h;
- CoveragePropertiesWizardPage.h;
- CoverageAction.h;
- CoverageWizard.h;
- CoverageLayerItem.h;
- CoverageWizardForm.ui;
- CoveragePropertiesWidgetForm.ui.

5 Conclusão

O objetivo de gerar o layer junto ao slider não foi cumprido, porém a criação do botão e do wizard para coverage series foi bem sucedido como pode ser visto nas imagens seguintes.

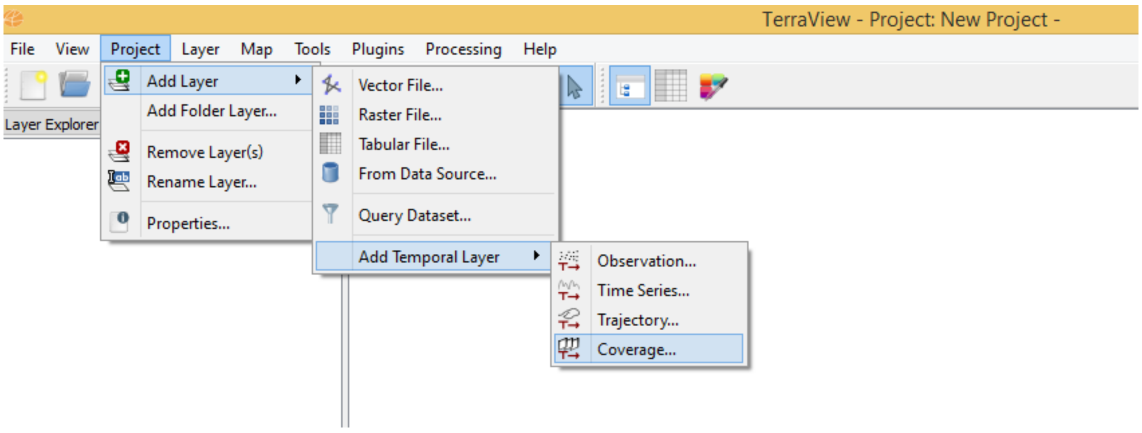
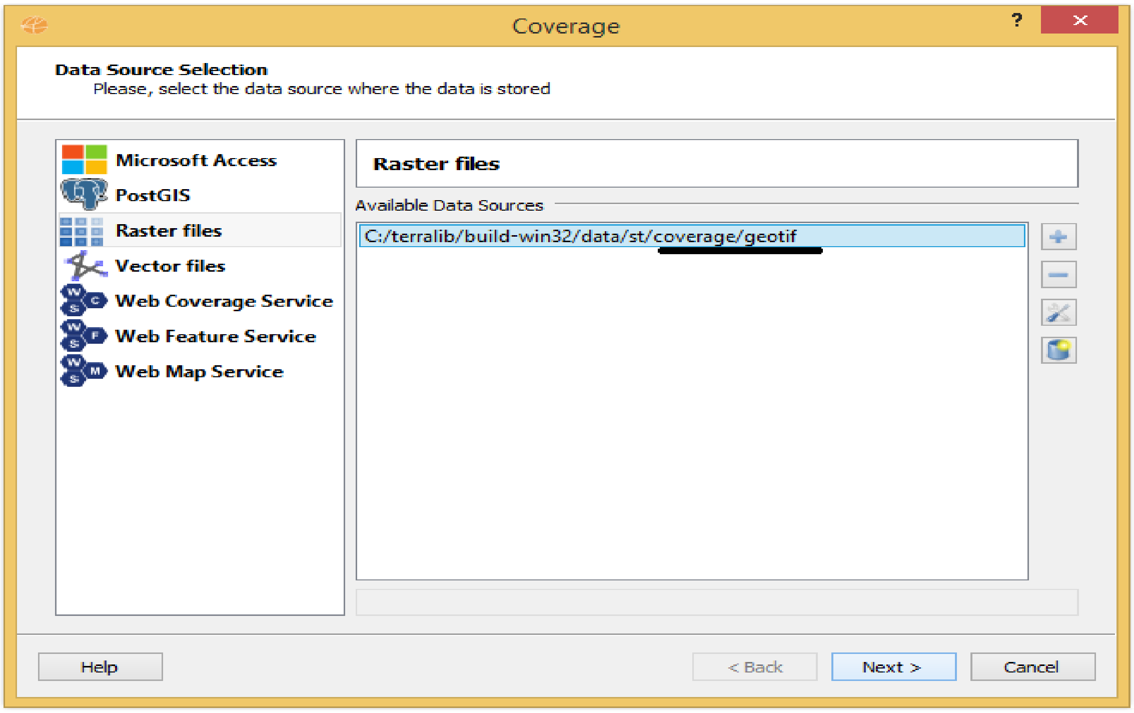


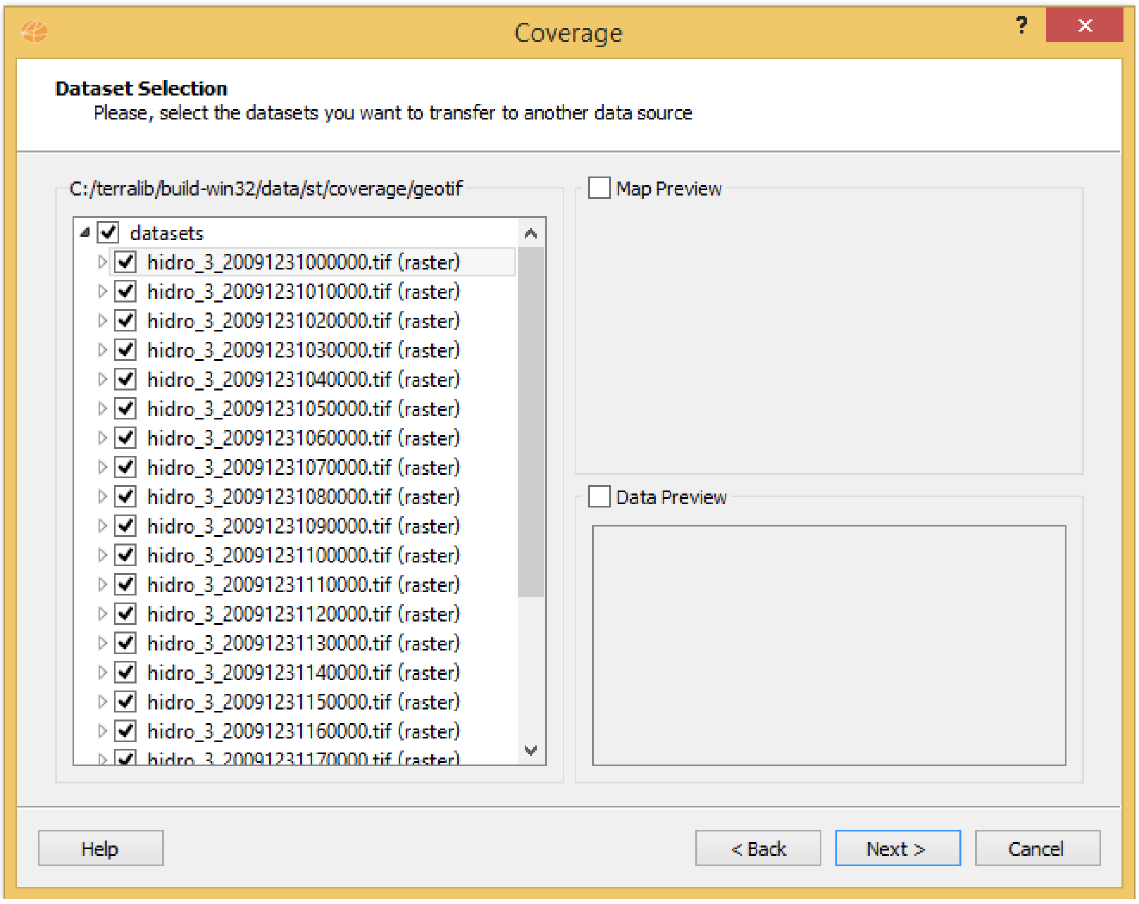
Figura 6 Botão para adicionar Coverage Series



Data Source Selection
Please, select the data source where the data is stored

C:/terralib/build-win32/data/st/coverage/geotif

Figura 7Seleção do Data Source



Coverage

Dataset Selection

Please, select the datasets you want to transfer to another data source

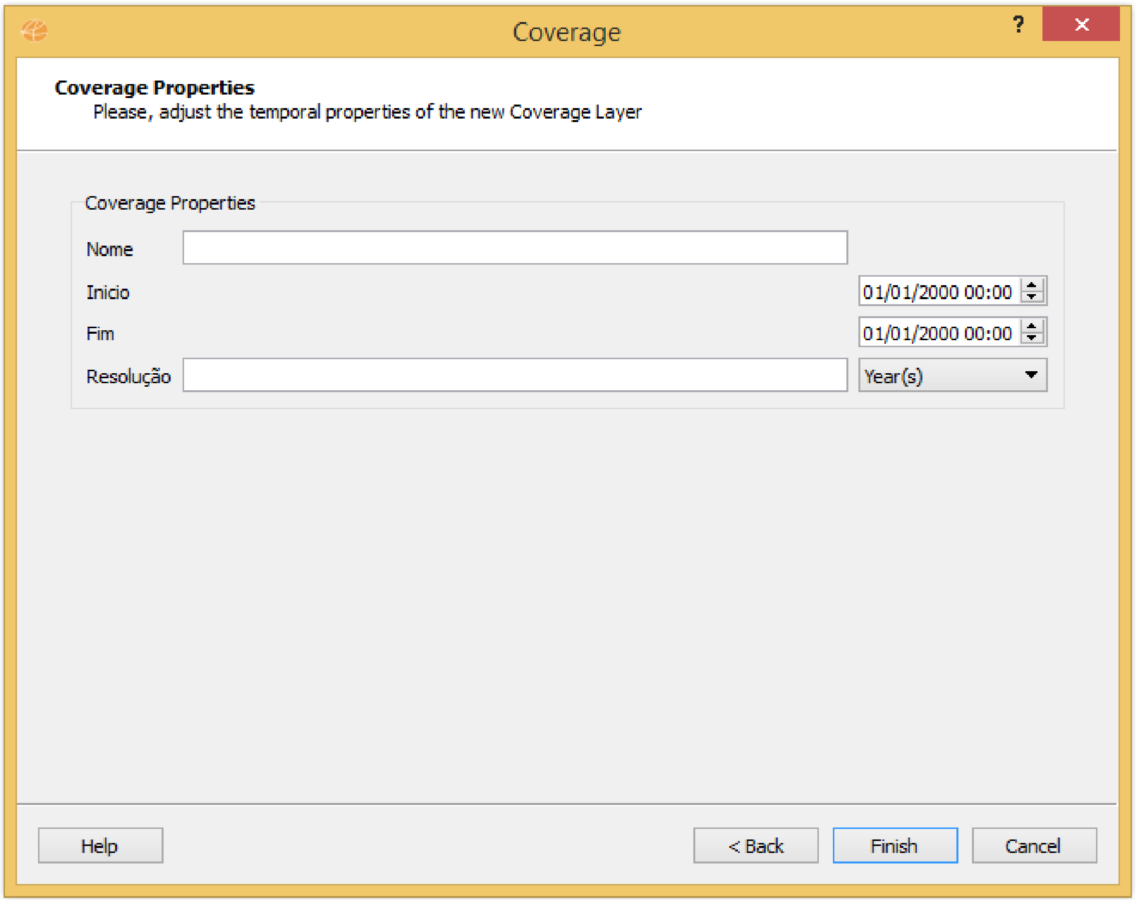
C:/terralib/build-win32/data/st/coverage/geotif

- ✓ datasets
  - ✓ hidro_3_20091231000000.tif (raster)
  - ✓ hidro_3_20091231010000.tif (raster)
  - ✓ hidro_3_20091231020000.tif (raster)
  - ✓ hidro_3_20091231030000.tif (raster)
  - ✓ hidro_3_20091231040000.tif (raster)
  - ✓ hidro_3_20091231050000.tif (raster)
  - ✓ hidro_3_20091231060000.tif (raster)
  - ✓ hidro_3_20091231070000.tif (raster)
  - ✓ hidro_3_20091231080000.tif (raster)
  - ✓ hidro_3_20091231090000.tif (raster)
  - ✓ hidro_3_20091231100000.tif (raster)
  - ✓ hidro_3_20091231110000.tif (raster)
  - ✓ hidro_3_20091231120000.tif (raster)
  - ✓ hidro_3_20091231130000.tif (raster)
  - ✓ hidro_3_20091231140000.tif (raster)
  - ✓ hidro_3_20091231150000.tif (raster)
  - ✓ hidro_3_20091231160000.tif (raster)
  - ✓ hidro_3_20091231170000.tif (raster)

☐ Map Preview

☐ Data Preview

Figura 8 Seleção dos arquivos


Coverage

Coverage Properties

Please, adjust the temporal properties of the new Coverage Layer

Coverage Properties

Nome		
Inicio		01/01/2000 00:00
Fim		01/01/2000 00:00
Resolução		Year(s)

Figura 9 Configurações do raster temporal

No futuro ainda se faz necessário unir o slider com o Layer de Coverage Series, e pretende-se no futuro permitir que um banco de dados seja utilizado como fonte dos dados.

Bibliografia

DPI INPE. (12 de 06 de 2015). *TerraLib and TerraView 5.0 Wiki Page*. Fonte: TerraLib and TerraView 5.0 Wiki Page: http://www.dpi.inpe.br/terralib5/wiki/doku.php?id=start

Ferreira, K. R. (2015). *Projeto TerraLib.*

Ferreira, K. R., Câmara, G., & Monteiro, A. M. (2013). An Algebra for Spatiotemporal Data: From Observations to Events. *Transactions in GIS*.

Ferreira, K. R., Câmara, G., & Monteiro, A. M. (2013). *AN ALGEBRA FOR SPATIOTEMPORAL DATA: FROM OBSERVATIONS TO EVENTS.*

Ferreira, K. R., Oliveira, A. G., Monteiro, A. M., & Almeida, D. B. (s.d.). Temporal GIS and Spatiotemporal Data Sources.

Queiroz, G. R., Ferreira, K. R., Vinhas, L., Camara, G., Souza, R. W., Souza, R. C., . . . Sanchez, A. (2015). WTSS: um serviço web para extração de séries temporais de imagens de sensoriamento remoto. *XVII Simpósio Brasileiro de Sensoriamento Remoto - SBSR.* João Pessoa: INPE.

Terraview - Wikipedia. (15 de 06 de 2015). Fonte: Wikipedia: http://pt.wikipedia.org/wiki/TerraView

Anexos

A - CoveragePropertiesWizardPage.cpp

```
#include "../../../st/core/coverage/RasterCoverageDataSetInfo.h"
#include "TemporalPropertiesWidget.h"
#include "CoveragePropertiesWidget.h"
#include "CoveragePropertiesWizardPage.h"
#include "ui_CoveragePropertiesWidgetForm.h"


te::qt::widgets::CoveragePropertiesWizardPage::CoveragePropertiesWizardPage(QWidget
* parent)
  : QWizardPage(parent)
{
  m_propWidget.reset(new CoveragePropertiesWidget(this));

  // Adjusting...
  QGridLayout* propLayout = new QGridLayout(this);
  propLayout->addWidget(m_propWidget.get());


}

te::qt::widgets::CoveragePropertiesWizardPage::~CoveragePropertiesWizardPage()
{
}

std::list<te::st::RasterCoverageDataSetInfo*>
te::qt::widgets::CoveragePropertiesWizardPage::getInfo(const te::da::DataSourceInfoPtr
dsInfo)
{
  std::list<te::st::RasterCoverageDataSetInfo*> covseInfos;

  std::list<te::da::DataSetTypePtr>::const_iterator typesItBegin = m_dataTypes.begin();
  std::list<te::da::DataSetTypePtr>::const_iterator typesItEnd = m_dataTypes.end();

  return covseInfos;
}

bool te::qt::widgets::CoveragePropertiesWizardPage::isComplete() const
{
  return true;
}
```

```
void te::qt::widgets::CoveragePropertiesWizardPage::set(const
std::list<te::da::DataSetTypePtr> dataTypes)
{
  m_dataTypes = dataTypes;

  m_propWidget->setUp(dataTypes.front());
}
```

B – CoveragePropertiesWizardPage.h

```
#ifndef __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEPROPERTIESWIZARDPAGE_H
#define __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEPROPERTIESWIZARDPAGE_H

// TerraLib
#include "../Config.h"
#include "terralib/dataaccess/datasource/DataSourceInfo.h"
#include "terralib/dataaccess/dataset/DataSetType.h"

// Qt
#include <QWizardPage>

// STL
#include <memory>

namespace te
{
  namespace st { class RasterCoverageDataSetInfo; }

  namespace qt
  {
    namespace widgets
    {
    //Forward declarations
   // class TemporalPropertiesWidget;
    class CoveragePropertiesWidget;

    /*!
        \class CoveragePropertiesWizardPage

        \brief A WizardPage used to configure the general properties of a new spatio-
temporal layer.
      */
      class TEQTWIDGETSEXPORT CoveragePropertiesWizardPage : public QWizardPage
      {
      Q_OBJECT

        public:
          CoveragePropertiesWizardPage(QWidget* parent = 0);
          ~CoveragePropertiesWizardPage();

          std::list<te::st::RasterCoverageDataSetInfo*> getInfo(const te::da::DataSourceInfoPtr
dsInfo);

          bool isComplete() const;

          void set(const std::list<te::da::DataSetTypePtr> dataTypes);

        private:

          std::list<te::da::DataSetTypePtr>          m_dataTypes;      //!< The list of datasettypes
used to configure the trajectory(ies)
          std::auto_ptr<CoveragePropertiesWidget>  m_propWidget;     //!< The widget used
to configure the unique CoverageLayer's properties
       // std::auto_ptr<TemporalPropertiesWidget>     m_tempPropWidget;  //!< The widget
used to configure the general TrajectoryLayer's properties
      };
    } // end namespace widgets
  }   // end namespace qt
}     // end namespace te

#endif  // __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEPROPERTIESWIZARDPAGE_H
```

C – CoverageWizard.cpp

```
#include "../../../geometry/GeometryProperty.h"
#include "../../../qt/widgets/dataset/selector/DataSetSelectorWizardPage.h"
#include "../../../qt/widgets/datasource/selector/DataSourceSelectorWidget.h"
#include "../../../qt/widgets/datasource/selector/DataSourceSelectorWizardPage.h"
#include "../../../qt/widgets/help/HelpPushButton.h"
#include "../../../se/Utils.h"
#include "../../../st/core/trajectory/TrajectoryDataSetInfo.h"
#include "CoveragePropertiesWizardPage.h"
#include "CoverageWizard.h"
#include "ui_CoverageWizardForm.h"

 //Boost
#include <boost/uuid/random_generator.hpp>
#include <boost/uuid/uuid_io.hpp>


  ///CoverageDataSetLayer TO DO

te::qt::widgets::CoverageWizard::CoverageWizard(QWidget *parent,Qt::WindowFlags f) :
   QWizard(parent),
   m_ui(new Ui::CoverageWizardForm)
```

```
{
   m_ui->setupUi(this);


//DataSource
  m_datasourceSelectorPage.reset(new DataSourceSelectorWizardPage(this));
  m_datasourceSelectorPage->setTitle(tr("Data Source Selection"));
  m_datasourceSelectorPage->setSubTitle(tr("Please, select the data source where the
data is stored"));
  m_datasourceSelectorPage->getSelectorWidget()-
>setSelectionMode(QAbstractItemView::SingleSelection);
  m_datasourceSelectorPage->getSelectorWidget()-
>showDataSourceWithRasterSupport(true);
  setPage(PAGE_DATASOURCE_SELECTION, m_datasourceSelectorPage.get());

//DataSet
  m_datasetSelectorPage.reset(new DataSetSelectorWizardPage(this));
  m_datasetSelectorPage->setTitle(tr("Dataset Selection"));
  m_datasetSelectorPage->setSubTitle(tr("Please, select the datasets you want to transfer
to another data source"));
  setPage(PAGE_DATASET_SELECTION, m_datasetSelectorPage.get());

//Coverage Properties

  m_PropWidgetPage.reset(new CoveragePropertiesWizardPage(this));
  m_PropWidgetPage->setTitle(tr("Coverage Properties"));
  m_PropWidgetPage->setSubTitle(tr("Please, adjust the temporal properties of the new
Coverage Layer"));
  setPage(PAGE_COVERAGE_PROPERTIES_SELECTION, m_PropWidgetPage.get());

   // connect signals and slots
  connect(this->button(QWizard::NextButton), SIGNAL(pressed()), this, SLOT(next()));
  connect(this->button(QWizard::BackButton), SIGNAL(pressed()), this, SLOT(back()));
  connect(this->button(QWizard::FinishButton), SIGNAL(pressed()), this, SLOT(finish()));

te::qt::widgets::HelpPushButton* helpButton = new te::qt::widgets::HelpPushButton(this);
  this->setButton(QWizard::HelpButton, helpButton);

}
te::qt::widgets::CoverageWizard::~CoverageWizard()
{
}

te::da::DataSourceInfoPtr te::qt::widgets::CoverageWizard::getDataSource() const
{
  std::list<te::da::DataSourceInfoPtr> datasources = m_datasourceSelectorPage-
>getSelectorWidget()->getSelecteds();
```

```
  if(datasources.empty())
    return te::da::DataSourceInfoPtr();
  else
    return datasources.front();
}

/*
std::list<te::st::TrajectoryDataSetLayerPtr>
te::qt::widgets::TrajectoryWizard::getTrajectoryLayers()
{
  return m_trajectoryLayers;
} */

void te::qt::widgets::CoverageWizard::back()
{
  QWizard::back();
}

void te::qt::widgets::CoverageWizard::next()
{
  if(currentId() == PAGE_DATASOURCE_SELECTION)
  {
    m_datasetSelectorPage->set(getDataSource(), true);
  }
  else if (currentId() == PAGE_DATASET_SELECTION)
  {
   // m_PropWidgetPage->set(m_datasetSelectorPage->getCheckedDataSets());
  }
  QWizard::next();
}
void te::qt::widgets::CoverageWizard::finish()
{
  QApplication::setOverrideCursor(Qt::WaitCursor);
  te::da::DataSourceInfoPtr dataInfo = getDataSource();
  std::list<te::da::DataSetTypePtr> dataTypes = m_datasetSelectorPage-
>getCheckedDataSets();

  QApplication::restoreOverrideCursor();
  QWizard::finished(0);
}
```

D – CoverageWizard.h

```
/*!
  \file terralib/qt/widgets/st/CoverageWizard.h

  \brief  A wizard used to generate a new Coveragelayer.
*/
```

```
#ifndef __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEWIZARD_H
#define __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEWIZARD_H

//Terralib
#include "../../../dataaccess/datasource/DataSourceInfo.h"
#include "../../../dataaccess.h"
#include "../../../st/maptools/TrajectoryDataSetLayer.h"
#include "../Config.h"

//Qt
#include <QWizard>

//Forward declaration
namespace Ui { class CoverageWizardForm; }

namespace te
{
  namespace st { class TrajectoryDataSetLayer; }

  namespace qt
  {
    namespace widgets
    {

    //Forward declarations
    class DataSourceSelectorWizardPage;
    class DataSetSelectorWizardPage;
    class CoveragePropertiesWizardPage;

    /*!
       \class TrajectoryDialog

       \brief A Dialog used to generate a new TrajectoryLayer
      */

class TEQTWIDGETSEXPORT CoverageWizard : public QWizard
{
    Q_OBJECT

public:
    CoverageWizard(QWidget *parent = 0,Qt::WindowFlags f=0);

    ~CoverageWizard();

    te::da::DataSourceInfoPtr getDataSource() const;

    //std::list<te::st::TrajectoryDataSetLayerPtr> getTrajectoryLayers();
```

```
 protected slots:

         void back();

         void next();

         void finish();


private:

       enum
         {
           PAGE_DATASOURCE_SELECTION,
           PAGE_DATASET_SELECTION,
           PAGE_COVERAGE_PROPERTIES_SELECTION
         };

         std::auto_ptr<Ui::CoverageWizardForm>       m_ui;                   //!< The wizard's form
       std::auto_ptr<DataSourceSelectorWizardPage>    m_datasourceSelectorPage;  //!<
The wizard page used to select the datasource
       std::auto_ptr<DataSetSelectorWizardPage>      m_datasetSelectorPage;     //!< The
wizard page used to select the dataset
       std::auto_ptr<CoveragePropertiesWizardPage>  m_PropWidgetPage;         //!< The
widget used to configure the properties of the new TrajectoryLayer
       //std::list<te::st::TrajectoryDataSetLayerPtr>   m_trajectoryLayers;       //!< The new
Trajectory Layer(s);



};
}}}

#endif // __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEWIZARD_H
```

E – CoverageLayerItem.cpp

```
#include "../../../common/Translator.h"
#include "../../../se/Style.h"
#include "../../../qt/widgets/Exception.h"
#include "../../../qt/widgets/layer/explorer/ChartItem.h"
#include "../../../qt/widgets/layer/explorer/GroupingItem.h"
#include "../../../qt/widgets/layer/explorer/LegendItem.h"
#include "CoverageLayerItem.h"

// Qt
#include <QMenu>
#include <QWidget>
```

```
te::qt::plugins::st::CoverageLayerItem::CoverageLayerItem(const
te::map::AbstractLayerPtr& l, QObject* parent)
  : te::qt::widgets::AbstractTreeItem(parent)
{
  m_layer = boost::dynamic_pointer_cast<te::st::TrajectoryDataSetLayer>(l);
}

te::qt::plugins::st::CoverageLayerItem::~CoverageLayerItem()
{
}

int te::qt::plugins::st::CoverageLayerItem::columnCount() const
{
  return 1;
}

QVariant te::qt::plugins::st::CoverageLayerItem::data(int /*column*/, int role) const
{
  if(role == Qt::DecorationRole)
    return QVariant(QIcon::fromTheme("coverage-layer"));

  if(role == Qt::DisplayRole)
    return QVariant(QString::fromStdString(m_layer->getTitle()));

  if(role == Qt::CheckStateRole)
    return QVariant(m_layer->getVisibility() == te::map::VISIBLE ? Qt::Checked :
Qt::Unchecked);

  return QVariant();
}

QMenu* te::qt::plugins::st::CoverageLayerItem::getMenu(QWidget* /*parent*/) const
{
  return 0;
}

bool te::qt::plugins::st::CoverageLayerItem::canFetchMore() const
{
  return (((m_layer->getStyle() != 0) && (!m_layer->getStyle()->getRules().empty())) ||
m_layer->getGrouping() != 0 || m_layer->getChart() != 0);
}

Qt::ItemFlags te::qt::plugins::st::CoverageLayerItem::flags() const
{
  return Qt::ItemIsUserCheckable | Qt::ItemIsDragEnabled | Qt::ItemIsDropEnabled;
}
```

```
void te::qt::plugins::st::CoverageLayerItem::fetchMore()
{
  if(m_layer->getStyle() && children().empty())
  {
    const std::vector<te::se::Rule*>& rules = m_layer->getStyle()->getRules();

    for(std::size_t i = 0; i != rules.size(); ++i)
      new te::qt::widgets::LegendItem(rules[i], this);
  }

  if(m_layer->getGrouping() && !hasGroupingItem())
    new te::qt::widgets::GroupingItem(m_layer->getGrouping(), this);

  if(m_layer->getChart() && !hasChartItem())
    new te::qt::widgets::ChartItem(m_layer->getChart(), this);
}

bool te::qt::plugins::st::CoverageLayerItem::hasChildren() const
{
  return ((m_layer->getStyle() != 0) && (!m_layer->getStyle()->getRules().empty())) ||
m_layer->getGrouping() != 0 || m_layer->getChart() != 0;
}

bool te::qt::plugins::st::CoverageLayerItem::setData(int column, const QVariant& value, int
role)
{
  if(role == Qt::CheckStateRole)
  {
    Qt::CheckState checkState = static_cast<Qt::CheckState>(value.toInt());

    if(checkState == Qt::Checked)
      m_layer->setVisibility(te::map::VISIBLE);
    else if(checkState == Qt::Unchecked)
      m_layer->setVisibility(te::map::NOT_VISIBLE);

    m_layer->updateVisibilityOfAncestors();

    return true;
  }

  return false;
}

te::map::AbstractLayerPtr te::qt::plugins::st::CoverageLayerItem::getLayer() const
{
  return m_layer;
}
```

```
const std::string te::qt::plugins::st::CoverageLayerItem::getItemType() const
{
  return "Coverage_LAYER_ITEM";
}

bool te::qt::plugins::st::CoverageLayerItem::hasGroupingItem() const
{
  te::qt::widgets::GroupingItem* groupingItem =
findChild<te::qt::widgets::GroupingItem*>();

  return groupingItem != 0;
}

bool te::qt::plugins::st::CoverageLayerItem::hasChartItem() const
{
  te::qt::widgets::ChartItem* chartItem = findChild<te::qt::widgets::ChartItem*>();

  return chartItem != 0;
}
```

F – CoverageLayerItem.h

```
#ifndef __TE_QT_PLUGINS_ST_INTERNAL_COVERAGELAYERITEM_H
#define __TE_QT_PLUGINS_ST_INTERNAL_COVERAGELAYERITEM_H

// TerraLib
#include "../../../st/maptools/TrajectoryDataSetLayer.h"
#include "../../../qt/widgets/layer/explorer/AbstractTreeItem.h"

namespace te
{
  namespace qt
  {
    namespace plugins
    {
      namespace st
       {
        class CoverageLayerItem : public te::qt::widgets::AbstractTreeItem
        {
          Q_OBJECT

          public:

            CoverageLayerItem(const te::map::AbstractLayerPtr& l, QObject* parent = 0);

            ~CoverageLayerItem();

            int columnCount() const;
            QVariant data(int column, int role) const;

            QMenu* getMenu(QWidget* parent = 0) const;

            bool canFetchMore() const;

            Qt::ItemFlags flags() const;

            void fetchMore();

            bool hasChildren() const;

            bool setData(int column, const QVariant& value, int role = Qt::EditRole);

            te::map::AbstractLayerPtr getLayer() const;

            const std::string getItemType() const;

          private:

            bool hasGroupingItem() const;

            bool hasChartItem() const;

          private:

            te::st::TrajectoryDataSetLayerPtr m_layer;
        };
      } // end namespace st
    }   // end namespace plugins
  }    // end namespace qt
}     // end namespace te

#endif  // __TE_QT_PLUGINS_ST_INTERNAL_COVERAGELAYERITEM_H
```

G – CoverageAction.cpp

```
//Terralib
#include "../../../qt/af/events/LayerEvents.h"
#include "../../../qt/widgets/dataset/selector/DataSetSelectorWizardPage.h"
#include "../../../qt/widgets/datasource/selector/DataSourceSelectorWizardPage.h"
#include "../../../qt/widgets/layer/explorer/AbstractTreeItemFactory.h"
#include "../../../qt/widgets/st/CoverageWizard.h"
#include "../../../st/loader/STDataLoader.h"
#include "../../af/ApplicationController.h"
#include "CoverageAction.h"
#include "CoverageLayerItem.h"
```

```
// Qt
#include <QMessageBox>
#include <QWizard>
#include <QWizardPage>

//STL
#include <memory>

// Boost
#include <boost/functional/factory.hpp>
#include <boost/bind.hpp>

te::qt::plugins::st::CoverageAction::CoverageAction(QMenu* menu)
: te::qt::plugins::st::AbstractAction(menu)
{
  createAction(tr("Coverage...").toStdString(), "Coverage-layer");
  te::qt::widgets::AbstractTreeItemFactory::reg("CoverageDATASETLAYER",
boost::bind(boost::factory<CoverageLayerItem*>(),_1, _2));
}

 te::qt::plugins::st::CoverageAction::~CoverageAction()
{
}

void te::qt::plugins::st::CoverageAction::onActionActivated(bool checked)
{



    QWidget* parent = te::qt::af::ApplicationController::getInstance().getMainWindow();

  std::auto_ptr<te::qt::widgets::CoverageWizard> covseWiz;
  covseWiz.reset( new te::qt::widgets::CoverageWizard(parent));

  int res = covseWiz->exec();/*

}
```

H – CoverageAction.h

```
#ifndef __TE_QT_PLUGINS_ST_INTERNAL_COVERAGEACTION_H
#define __TE_QT_PLUGINS_ST_INTERNAL_COVERAGEACTION_H

// TerraLib
#include "Config.h"
#include "AbstractAction.h"

namespace te
{
  namespace qt
  {
    namespace plugins
    {
      namespace st
      {

        /*!
          \class CoverageAction

          \brief This class register the time series action into the St plugin.

        */
        class CoverageAction : public te::qt::plugins::st::AbstractAction
        {
          Q_OBJECT

          public:

            CoverageAction(QMenu* menu);

            virtual ~CoverageAction();

          protected slots:

            virtual void onActionActivated(bool checked);
        };

      } // end namespace st
    }  // end namespace plugins
  }    // end namespace qt
}      // end namespace te

#endif //__TE_QT_PLUGINS_ST_INTERNAL_COVERAGEACTION_H
```

I – CoveragePropertiesWidget.cpp

```
//Terralib
#include "../../../dataaccess.h"
#include "../../../datatype/Property.h"
#include "CoveragePropertiesWidget.h"
#include "ui_CoveragePropertiesWidgetForm.h"

//QT
#include <QWidget>

te::qt::widgets::CoveragePropertiesWidget::CoveragePropertiesWidget(QWidget* parent,
Qt::WindowFlags f)
  : QWidget(parent, f),
   m_ui(new Ui::CoveragePropertiesWidgetForm)
{
  m_ui->setupUi(this);
 }

te::qt::widgets::CoveragePropertiesWidget::~CoveragePropertiesWidget()
{
}

Ui::CoveragePropertiesWidgetForm*
te::qt::widgets::CoveragePropertiesWidget::getForm()
{
  return m_ui.get();
}


void te::qt::widgets::CoveragePropertiesWidget::setUp (const te::da::DataSetTypePtr
dataType)
{
  QString item;
  m_dataType = dataType;

  const std::vector<te::dt::Property*>& properties = dataType->getProperties();

  for (std::size_t i = 0; i < properties.size(); i++)
  {

  }


}
```

J – CoveragePropertiesWidget.h

```
#ifndef __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEPROPERTIESWIDGET_H
#define __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEPROPERTIESWIDGET_H

//TerraLib
#include "../../../dataaccess/dataset/DataSetType.h"
#include "../Config.h"

// Qt
#include <QWidget>

//STL
#include <memory>

namespace Ui { class CoveragePropertiesWidgetForm; }
namespace te
{
 namespace qt
 {
  namespace widgets
  {

   class DoubleListWidget;

  /*!
    \class CoveragePropertiesWidget

    \brief A widget used to adjust a Coverage layer's properties
   */
   class TEQTWIDGETSEXPORT CoveragePropertiesWidget : public QWidget
   {

    Q_OBJECT

    public:

     /*!
      \brief Constructor

      \param dataSetType The datasetType that will be used to generate a
CoverageLayer.
      \param parent this widget's parent
      \param f Window flags used to configure this widget
     */
     CoveragePropertiesWidget(QWidget* parent = 0,  Qt::WindowFlags f = 0);

     /*!
      \brief Destructor
     */
     ~CoveragePropertiesWidget();

     /*!
      \brief Returns a pointer to the widget's form

      \return A CoveragePropertiesWidgetForm type pointer to the widget's form.
      \note The caller will not take ownership of the returned pointer.
     */
     Ui::CoveragePropertiesWidgetForm* getForm();

     /*!
      \brief Returns a vector containing the indexes of the observed properties

      \return A vector containing the indexes of the observed properties.
```

```
*/
std::vector<int> getOutputValues();

/*!
  \brief Returns a vector containing the names of the observed properties

  \return A vector containing the names of the observed properties.
*/
std::vector<std::string> getOutputPropNames();

/*!
  \brief Returns the name of the property that holds the geometry

  \return The name of the property that holds the geometry
.
*/
std::string getGeometryPropName();

/*!
  \brief Returns the index of the temporal property geometry

  \return The index of the temporal property geometry
  \note Will return an invalid index if the dataSeType hasn't been given.
*/
int getGeometryId();

/*!
  \brief Returns the name of the property that holds the Coverage ID

  \return The name of the property that holds the Coverage ID
.
*/
std::string getIdPropName();

/*!
  \brief Returns the index of the Coverage ID

  \return The index of the Coverage ID
  \note Will return an invalid index if the dataSeType hasn't been given.
*/
int getIdIndex();

/*!
  \brief Adjusts the widget's components based on the given datasettype

  \param dataType The datasetType that will be used to configure the widget.
*/
void setUp(const te::da::DataSetTypePtr dataType);
```

```
      private:

        //std::auto_ptr<DoubleListWidget>                  m_obsWidget;  //!< The widget used
to select the observed properties.
        std::auto_ptr<Ui::CoveragePropertiesWidgetForm>  m_ui;        //!< The widget's
form.
        te::da::DataSetTypePtr                          m_dataType;   //!< The datasetType that will
be used to generate the spatio-temporal layer.
      };
    } // end namespace widgets
  }   // end namespace qt
}     // end namespace te

#endif  // __TERRALIB_QT_WIDGETS_INTERNAL_COVERAGEPROPERTIESWIDGET_H
```

K – Plugin.cpp

```
#include "../../../common/Config.h"
#include "../../../common/Translator.h"
#include "../../../common/Logger.h"
#include "../../af/ApplicationController.h"
#include "Plugin.h"

#ifdef TE_QT_PLUGIN_ST_HAVE_SLIDER
  #include "TimeSliderWidgetAction.h"
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_OBSERVATION
    #include "ObservationAction.h"
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_TIMESERIES
    #include "TimeSeriesAction.h"
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_TRAJECTORY
    #include "TrajectoryAction.h"
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_COVERAGE
    #include "CoverageAction.h"
#endif

// QT
#include <QMenu>
#include <QMenuBar>

te::qt::plugins::st::Plugin::Plugin(const te::plugin::PluginInfo& pluginInfo)
```

```
  : te::plugin::Plugin(pluginInfo), m_stMenu(0)
{
}

te::qt::plugins::st::Plugin::~Plugin()
{
}

void te::qt::plugins::st::Plugin::startup()
{
  if(m_initialized)
    return;

// it initializes the Translator support for the TerraLib st Qt Plugin
  //TE_ADD_TEXT_DOMAIN(TE_QT_PLUGIN_ST_TEXT_DOMAIN,
TE_QT_PLUGIN_ST_TEXT_DOMAIN_DIR, "UTF-8");

  TE_LOG_TRACE(TE_TR("TerraLib Qt ST Plugin startup!"));

// add plugin menu
  m_stMenu = te::qt::af::ApplicationController::getInstance().getMenu("Project.Add
Layer.Add Temporal Layer");

  m_stMenu->setTitle(TE_TR("Add Temporal Layer"));

// register actions
  registerActions();

  m_initialized = true;
}

void te::qt::plugins::st::Plugin::shutdown()
{
  if(!m_initialized)
    return;

// unregister actions
  unRegisterActions();

// remove menu
  delete m_stMenu;

  TE_LOG_TRACE(TE_TR("TerraLib Qt ST Plugin shutdown!"));

  m_initialized = false;
}

void te::qt::plugins::st::Plugin::registerActions()
```

```
{
#ifdef TE_QT_PLUGIN_ST_HAVE_SLIDER
   m_sliderAction = new
te::qt::plugins::st::TimeSliderWidgetAction(te::qt::af::ApplicationController::getInstance().f
indMenu("View"));
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_OBSERVATION
   m_observactionAction = new te::qt::plugins::st::ObservationAction(m_stMenu);
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_TIMESERIES
   m_timeSeriesAction = new te::qt::plugins::st::TimeSeriesAction(m_stMenu);
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_TRAJECTORY
   m_trajectoryAction = new te::qt::plugins::st::TrajectoryAction(m_stMenu);
#endif

     #ifdef TE_QT_PLUGIN_ST_HAVE_COVERAGE
   m_coverageAction = new te::qt::plugins::st::CoverageAction(m_stMenu);
#endif
}

void  te::qt::plugins::st::Plugin::unRegisterActions()
{
#ifdef TE_QT_PLUGIN_ST_HAVE_SLIDER
   delete m_sliderAction;
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_OBSERVATION
   delete m_observactionAction;
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_TIMESERIES
   delete m_timeSeriesAction;
#endif

#ifdef TE_QT_PLUGIN_ST_HAVE_TRAJECTORY
   delete m_trajectoryAction;
#endif

     #ifdef TE_QT_PLUGIN_ST_HAVE_COVERAGE
   delete m_coverageAction;
#endif
}

PLUGIN_CALL_BACK_IMPL(te::qt::plugins::st::Plugin)
```

L – Plugin.h

```
#ifndef __TE_QT_PLUGINS_ST_INTERNAL_PLUGIN_H
#define __TE_QT_PLUGINS_ST_INTERNAL_PLUGIN_H

// TerraLib
#include "../../../plugin/Plugin.h"
#include "Config.h"

// Qt
#include <QMenu>

namespace te
{
  namespace qt
  {
    namespace plugins
    {
      namespace st
      {
        class TimeSliderWidgetAction;
                  class ObservationAction;
        class TimeSeriesAction;
        class TrajectoryAction;
                  class CoverageAction;

        class Plugin : public te::plugin::Plugin
        {
          public:

            Plugin(const te::plugin::PluginInfo& pluginInfo);

            ~Plugin();

            void startup();

            void shutdown();

          protected:

            /*!
              \brief Function used to register all raster processing actions.

            */
            void registerActions();

            /*!
              \brief Function used to unregister all raster processing actions.
            */
            void unRegisterActions();

          protected:

            QMenu*                              m_stMenu;            //!< ST Main Menu registered.
            te::qt::plugins::st::TimeSliderWidgetAction*  m_sliderAction;       //!< Slider Process
Action
            te::qt::plugins::st::ObservationAction*      m_observactionAction;  //!< Observation
Layer Action
            te::qt::plugins::st::TimeSeriesAction*       m_timeSeriesAction;    //!< TimeSeries
Layer Action
            te::qt::plugins::st::TrajectoryAction*       m_trajectoryAction;    //!< Trajectory Layer
Action
                              te::qt::plugins::st::CoverageAction*       m_coverageAction;    //!<
Trajectory Layer Action
        };

      } // end namespace st
    }  // end namespace plugins
  }    // end namespace qt
}      // end namespace te

PLUGIN_CALL_BACK_DECLARATION(TEQTPLUGINSTEXPORT);

#endif //__TE_QT_PLUGINS_ST_INTERNAL_PLUGIN_H
```

M – CoverageWizardForm.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>CoverageWizardForm</class>
 <widget class="QWizard" name="CoverageWizardForm" >
  <property name="geometry" >
   <rect>
    <x>0</x>
    <y>0</y>
    <width>640</width>
    <height>480</height>
   </rect>
  </property>
  <property name="windowTitle" >
   <string>Coverage</string>
   </property>
  <property name="wizardStyle">
   <enum>QWizard::ModernStyle</enum>
  </property>
  <property name="options">
   <set>QWizard::HaveHelpButton|QWizard::IndependentPages</set>
  </property>
 </widget>
 <resources/>
 <connections/>
</ui>
```

N – CoveragePropertiesWidgetForm.ui

```
 <?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>CoveragePropertiesWidgetForm</class>
 <widget class="QWidget" name="CoveragePropertiesWidgetForm">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>312</width>
    <height>331</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Trajectory</string>
  </property>
  <layout class="QGridLayout" name="gridLayout">
   <item row="0" column="0">
    <widget class="QGroupBox" name="m_coverageGroupBox">
     <property name="title">
      <string>Coverage Properties</string>
     </property>
     <layout class="QGridLayout" name="gridLayout_4">
      <item row="0" column="0">
       <layout class="QGridLayout" name="gridLayout_3">
        <item row="3" column="0">
         <widget class="QLabel" name="label_3">
          <property name="text">
           <string>Resolução</string>
          </property>
         </widget>
        </item>
        <item row="2" column="0">
         <widget class="QLabel" name="label">
          <property name="text">
           <string>Fim</string>
          </property>
         </widget>
        </item>
```

```
<item row="3" column="2">
 <widget class="QComboBox" name="comboBox">
  <item>
   <property name="text">
    <string>Year(s)</string>
   </property>
  </item>
  <item>
   <property name="text">
    <string>Month(s)</string>
   </property>
  </item>
  <item>
   <property name="text">
    <string>Week(s)</string>
   </property>
  </item>
  <item>
   <property name="text">
    <string>Day(s)</string>
   </property>
  </item>
  <item>
   <property name="text">
    <string>Hour(s)</string>
   </property>
  </item>
  <item>
   <property name="text">
    <string>Minute(s)</string>
   </property>
  </item>
 </widget>
</item>
<item row="1" column="0">
 <widget class="QLabel" name="label_2">
  <property name="text">
   <string>Inicio</string>
  </property>
 </widget>
</item>
<item row="2" column="2">
 <widget class="QDateTimeEdit" name="dateTimeEdit_2"/>
</item>
<item row="1" column="2">
 <widget class="QDateTimeEdit" name="dateTimeEdit"/>
</item>
<item row="3" column="1">
```

```
        <widget class="QLineEdit" name="lineEdit"/>
       </item>
       <item row="0" column="0">
        <widget class="QLabel" name="label_4">
         <property name="text">
          <string>Nome</string>
         </property>
        </widget>
       </item>
       <item row="0" column="1">
        <widget class="QLineEdit" name="lineEdit_2"/>
       </item>
      </layout>
     </item>
    </layout>
   </widget>
  </item>
 </layout>
</widget>
<resources/>
<connections/>
</ui>
```