

APÊNDICE A

DTD completo da Linguagem TerraML

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Bianca Pedrosa (Particular) -->
<!--DTD generated by XMLSPY v2004 rel. 2 U (http://www.xmlspy.com)-->
<ELEMENT cellprocessor (input, control)>
<!ATTLIST cellprocessor
  author CDATA #REQUIRED
  date CDATA #REQUIRED
  model CDATA #REQUIRED
>
<ELEMENT condition EMPTY>
<!ATTLIST condition
  attribute CDATA #REQUIRED
  op CDATA #REQUIRED
  value CDATA #REQUIRED
>
<ELEMENT control (mode+, transition)>
<!ATTLIST control
  initime CDATA #REQUIRED
  intervals CDATA #REQUIRED
  step CDATA #REQUIRED
  timeUnit (YEAR | MONTH | DAY | HOUR ) #REQUIRED
>
<ELEMENT database EMPTY>
<!ATTLIST database
  host CDATA #REQUIRED
  path CDATA #REQUIRED
  name CDATA #REQUIRED
  user CDATA #IMPLIED
  pass CDATA #IMPLIED
>
<ELEMENT expander EMPTY>
<!ATTLIST expander
  attribute CDATA #REQUIRED
  column CDATA #REQUIRED
  demand CDATA #REQUIRED
>
<ELEMENT fuzzyL EMPTY>
<!ATTLIST fuzzyL
  attribute CDATA #REQUIRED
  column CDATA #REQUIRED
  alpha CDATA #REQUIRED
  beta CDATA #REQUIRED
>
<ELEMENT global EMPTY>
<!ATTLIST global
  name CDATA #REQUIRED
  value CDATA #REQUIRED
>
<ELEMENT input (database, layer, table, neighborhood, temporal, global+)>
<ELEMENT layer EMPTY>
<!ATTLIST layer
  name CDATA #REQUIRED
  layerid CDATA #REQUIRED
>
<ELEMENT localMean EMPTY>
<!ATTLIST localMean
  attribute CDATA #REQUIRED
  column CDATA #REQUIRED
>
<ELEMENT mode (fuzzyL*, localMean?, product, expander?)>
<!ATTLIST mode
  name CDATA #REQUIRED
>
<ELEMENT neighborhood EMPTY>
<!ATTLIST neighborhood
  name CDATA #REQUIRED
  zones CDATA #REQUIRED
>
<ELEMENT pair EMPTY>
<!ATTLIST pair
  attribute CDATA #REQUIRED
  weight CDATA #REQUIRED
>
<ELEMENT product (pair+)>
<!ATTLIST product
  attribute CDATA #REQUIRED

```

```
>
<!ELEMENT table EMPTY>
<!ATTLIST table
  name CDATA #REQUIRED
  columns CDATA #REQUIRED
  lines CDATA #REQUIRED
>
<!ELEMENT temporal EMPTY>
<!ATTLIST temporal
  name CDATA #REQUIRED
  attribute CDATA #REQUIRED
  period CDATA #REQUIRED
  timeunit (YEAR | MONTH | DAY | HOUR ) #REQUIRED
>
<!ELEMENT transition (condition)>
<!ATTLIST transition
  from CDATA #REQUIRED
  to CDATA #REQUIRED
>
```

APÊNDICE B

Código Fonte

Obs: Esta documentação destina-se a fornecer uma visão geral da abordagem de implementação. Os arquivos de cabeçalho (.h) estão completos. Dos arquivos fonte (.cpp) foram selecionadas as funções mais significativas do sistema.

```
#ifndef TeCellA_H
#define TeCellA_H

//Includes
//-----TerraLib
#include <TeCellAlgorithms.h>
#include <TeAdoDB.h>
#include <TeLayer.h>
#include <TeGeometry.h>
#include <TeSelectedObject.h>
#include <TeImportExport.h>
#include <TeTime.h>
#include <TeInitRasterDecoders.h>
#include <TeDecoderDatabase.h>
#include <TeRasterRemap.h>
#include <AuxiliarAlgorithms.h>
//-----Parser
#include <dom/DOM_Node.hpp>
#include <dom/DOM_Element.hpp>
//-----TerraML
#include "TeProxMtx.h"
#include "controlmode.h"
#include "bicell.h"

class TeCellAutomata
{
public:
    TeCellAutomata(char *x);
    void builder (DOM_Node& aNode);
    void buildsection (DOM_Node& aNode);
    int init( string, string , string , string , string, int, string, char *, int);
    void inputdata (DOM_Node& aNode);
    void inputcontrol (DOM_Node& aNode);
    void inputtime (DOM_Node& aNode);

    void printcells();
    void printDynCells();
    void print_neighbors();
    void printcontrols();

    bool createneighbors(char *archivo, int zonas);
    bool createDynAttribute();

    bool execute();
    bool process(ControlMode, int t);
    void calculate (Flow *f, int t);
    void fuzzyL(FuzzyL *f, int t);
    void localMean(LocalMean *f,int t);
    void sum (Product *f,int t) ;
    void expander(Expander *f, int t);
    double sort(string attribute, double demand, int t);
    void transit(string attribute, string column, double threshold, int t);
    ControlMode *getMode(string s);

private:
    BiCellSet cells_;
    TeProxMatrix neighborhood_;
    TeTimeSequence *timeseq_;
    ModeSet modes_;
    TransitionSet transitions_;

};

#endif
```

```
#ifndef BICELL_H
#define BICELL_H

#include <vector>
#include <map>
#include <TeGeometry.h>
#include <TeAttribute.h>

typedef vector<double> doubleRow;
typedef vector<doubleRow> doubleTable;
class BiCell : public TeCell
{
private:
    TeTableRow      attribute_; // Modeling Attributes - TerraML
    doubleTable     dynAttribute_; // Dynamic Attributes - TerraML

public:
    BiCell(string o) {objectId_=o;}

    BiCell (string o, int c, int l)
    {   objectId_=o;
        this->column(c);
        this->line(l);
    }

    //! Adds modeling attribbutes to the cell
    void setAttributes(TeTableRow s) { attribute_ = s;}

    //! Gets modeling attributes (the entire row)
    TeTableRow getAttributes () {return attribute_;}

    //! Adds modeling attribbutes to the cell
    void attribute(string s) { attribute_.push_back(s);}

    //! Return the i-th attribute of the cell
    string attribute(int i) {return attribute_[i];}

    //! Adds new line of dynamic attributes to the cell
    void dynAttribute(doubleRow s) { dynAttribute_.push_back(s);}

    //! Return the dynamic attributes of the cell
    doubleRow dynAttribute(int i) {return dynAttribute_[i];}

    //da
    void dynAtt(int l, int c, double value)
    {   dynAttribute_[l][c]=value; }

    double dynAtt(int l, int c)
    {return dynAttribute_[l][c];}
};
```

```

//! A class for handling sets of BiCells
class BiCellSet: public TeGeomComposite<BiCell>
{
    double resX_; //!< the X resolution of a set of cells
    double resY_; //!< the Y resolution of a set of cells

    TeAttributeList    attributes_;
    TeAttributeList    dynAttributes_;

public:

    //! operador de copia from TeCell
    operator = ( TeCellSet cells)
    {
        cout << "copiando TeCell em BiCell..."<< endl;
        TeCellSet::iterator cells_it=cells.begin();
        while (cells_it != cells.end())
        {
            BiCell bcell((*cells_it).objectId(),(*cells_it).column(),(*cells_it).line());
            this->add(bcell);
            cells_it++;
        }
    }

    //! Returns the X resolution of a cell set
    double resX ()
    { return resX_; }

    //! Returns the Y resolution of a cell set
    double resY ()
    { return resY_; }

    //! Sets the X resolution of a cell set
    void resX (double reX)
    { resX_ = reX; }

    //! Sets the Y resolution of a cell set
    void resY (double reY)
    { resY_ = reY; }

    //! Sets the attribute list of a cell set
    void attributes(TeAttributeList t) {attributes_=t;}

    void dynattributes(TeAttributeList t) {dynAttributes_=t;}

    string attributes(int n) {return attributes_[n].rep_.name_;}

    string dynAttributes(int n) {return dynAttributes_[n].rep_.name_;}

    int attribute(string s) //! retorna o nro da coluna do atributo s
    {
        int j = 0;
        TeAttribute attr;
        TeAttributeList::iterator att_it = attributes_.begin();
        while ( att_it != attributes_.end())
        {
            if ((*att_it).rep_.name_ == s)
            {
                return j;
            }
            j++;
            att_it++;
        }
        return -1;
    }
}

```

```
int dynAttribute(string s) //! retorna o nro da coluna do atributo s
{
    int j = 0;
    TeAttribute attr;
    TeAttributeList::iterator att_it = dynAttributes_.begin();
    while ( att_it != dynAttributes_.end())
    {
        if ((*att_it).rep_.name_ == s)
        {
            return j;
        }
        j++;
        att_it++;
    }
    return -1;
}

//! Sets the dynamic attribute list of a cell set
void dynAttributes(TeAttributeList t) {dynAttributes_=t;}

//! Returns the attribute list of a cell set
TeAttributeList attributes() {return attributes_;}

//! Returns the dynamic attribute list of a cell set
TeAttributeList dynAttributes() { return dynAttributes_;}

};

#endif
```

```

#ifndef TEPROXMTX_H
#define TEPROXMTX_H
#include <map>

struct ProxMatrixInfo {
    float weight_;
    int slice_;
    float d1_,d2_,d3_; //d1=centroidDistance, d2=NetDistance, d3=NetMinPath

    ProxMatrixInfo (): weight_(0.00), slice_(0), d1_(0.00), d2_(0.00), d3_(0.00){}

    ProxMatrixInfo (float w,int s,float d1,float d2,float d3):
        weight_(w), slice_(s), d1_(d1), d2_(d2), d3_(d3){}

    ProxMatrixInfo (const ProxMatrixInfo& c): weight_(c.weight_), slice_(c.slice_), d1_(c.d1_),
        d2_(c.d2_), d3_(c.d3_){}

    ProxMatrixInfo& operator = (const ProxMatrixInfo& x)
    { weight_ = x.weight_;
      slice_ = x.slice_;
      d1_ = x.d1_;
      d2_ = x.d2_;
      d3_ = x.d3_;
      return *this;
    }
    void getProxMatrixInfo()
    { cout << " weight = " << weight_ << " slice = " << slice_ << " d1 = " << d1_ << endl;
    }
    bool empty()
    { if (slice_>=0) return true;
      return false;
    }
    float weight() { return weight_;}
};

typedef map<TeCell*, ProxMatrixInfo> stringRow;
typedef map<string, stringRow> sparceMatrix;

class TeProxMatrix: public TeSingleton<TeProxMatrix>
{
public:
    void insertNeighbour (
        const string& o1, TeCell* o2,
        int slice, float weight, float d1, float d2, float d3)
    {
        ProxMatrixInfo pm ( weight, slice, d1, d2, d3);
        matrix_ [o1] [o2] = pm;
    }

    stringRow getNeighborhood ( string o1 )
    {
        return matrix_ [o1];
    }

    ProxMatrixInfo getANeighbor ( string o1, TeCell* o2 )
    {
        return matrix_ [o1] [o2];
    }
protected:
    sparceMatrix matrix_;
};

#endif

```

```

#ifndef CTRLMODE_H
#define CTRLMODE_H

#include <string>

using namespace std;

class Flow
{
    string attr_;
    string math_;
public:
    Flow(string a, string m): attr_(a), math_(m){}
    string attr() { return attr_;}
    string math() { return math_;}
};

typedef vector <Flow*> FlowSet;

class Condition
{
    string attr_, value_, oper_;
public:
    Condition(string a, string v, string o): attr_(a), value_(v), oper_(o){}
    string attr() { return attr_;}
    string oper() { return oper_;}
    string value() { return value_;}
};

class ControlMode
{
    string name_;
    FlowSet flows_;

public:
    //ControlMode();
    ControlMode():name_(""){ }
    ControlMode(string s) {name_=s;}
    string name() {return name_;}
    FlowSet flows() {return flows_;}
    void add(Flow *aFlow) {flows_.push_back(aFlow); }
    void print()
    { cout << name_ << endl;
      int n=flows_.size();
      for (int i=0;i<n;i++)
          cout << (*flows_[i]).attr() <<endl;
    }
};

typedef vector<ControlMode> ModeSet;

class Transition
{
    ControlMode *frommode_,*tomode_;
    vector<Condition > jumps_;

public:
    Transition (ControlMode* f, ControlMode* t): frommode_(f),tomode_(t){}
    void add (Condition aCondition) {jumps_.push_back(aCondition);}
    void print()
    { cout << (*frommode_).name() << ' ' << (*tomode_).name() << endl;
      for (int i=0;i<jumps_.size();i++)
          cout << jumps_[i].attr() << ' ' << jumps_[i].oper()<< ' '
          <<jumps_[i].value()<<endl;
    }
};

typedef vector <Transition> TransitionSet;

```

```
class FuzzyL : public Flow
{
    string column_;
    double alpha_;
    double beta_;
public:
    FuzzyL (string a, string m, string c, double aa, double b):Flow(a,m), column_(c), alpha_(aa),
    beta_(b){}
    string column() {return column_; }
    double alpha() {return alpha_; }
    double beta() { return beta_; }
};

class LocalMean : public Flow
{
    string column_;
public:
    LocalMean (string a, string m, string c):Flow(a,m), column_(c){}
    string column() {return column_; }
};

class Expander : public Flow
{
    string column_;
    double demand_;
public:
    Expander (string a, string m, string c, double d):Flow(a,m), column_(c), demand_(d){}
    string column() {return column_; }
    double demand() { return demand_; }
};

#endif
```

```

TeCellAutomata :: TeCellAutomata (char *xmlFile)
{
    DOM_Node doc_ = xmlParser(xmlFile);
    builder(doc_);
}

void TeCellAutomata::builder(DOM_Node& aNode)
{
    if (is_section( aNode.getNodeName().transcode() ))
    {
        buildsection(aNode);
        aNode = aNode.getNextSibling();
    }
    else
    {
        // Test for the presence of children
        DOM_Node child = aNode.getFirstChild();
        while( child != 0)
        {
            builder(child);
            child = child.getNextSibling();
        }
    }
}

void TeCellAutomata::buildsection (DOM_Node& aNode)
{
    // Test for the presence of children
    if (!strcmp(aNode.getNodeName().transcode(),"input"))
        inputdata(aNode);
    if (!strcmp(aNode.getNodeName().transcode(),"control"))
        inputcontrol(aNode);
    if (!strcmp(aNode.getNodeName().transcode(),"simulation"))
        inputtime(aNode);
    else
    {
        DOM_Node child = aNode.getFirstChild();
        while( child != 0)
        {
            child = child.getNextSibling();
        }
    }
}

int TeCellAutomata::init( string host, string dbname, string user, string pass, string layername, int
layerid, string tablename, char *neighborsname,int zonas)
{
    cout << "lendo banco " << host << ' ' << dbname << "..." <<endl ;
    // Opens a connection to a database accessible though ADO
    TeDatabase* ado_db = new TeAdo();
    if (!ado_db->connect(host,user,pass,dbname,0))
    {
        cout << "Error: " << ado_db->errorMessage() << endl;
        exit(0);
    }
    TeCellSet cells;
    if (!ado_db->loadCellSet (layerid, layername, "", cells))
    {
        cout << "\tLayer de entrada inexistente: " << layername << endl;
        ado_db->close();
        exit(0);
    }
    cells_ = cells;
    TeTable cellstable;
    if (!ado_db->loadTable (tablename, cellstable))
    {
        cout << "\tTable de entrada inexistente: " << tablename << endl;
        ado_db->close();
        exit(0);
    }
    assignAttr(cells_,cellstable); //adiciona atributos nas celulas
    createneighbors(neighborsname,zonas); //gera matriz de proximidade
    return 1;
}

```

```

bool TeCellAutomata::execute()
{
    createDynAttribute();
    for (int t=0; t<(*timeseq_).num_steps(); t++)
    {
        cout << "T " << t << endl;
        process((*modes_.begin()),t); //generalizar - assumi 1o modo local
        result(cells_,t);
    }

    return true;
}

bool TeCellAutomata::process (ControlMode Local, int t)
{
    FlowSet flows= Local.flows();
    int n=flows.size();
    for (int i=0;i<n;i++) //calcula todos os flows deste ControlMode
    {
        cout << "flow " << i << endl;
        calculate(flows[i],t);
    }
    return true;
}

void TeCellAutomata::fuzzyL(FuzzyL *f,int t)
{
    cout << "calculating a fuzzyLow...\n";
    int attcol=cells_.attribute((*f).column());
    double alpha=(*f).alpha();
    double beta=(*f).beta();
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        double x=atof((*cells_it).attribute(attcol).c_str());
        double value = (1 / (1 + ( alpha * pow( x - beta,2)))) ;
        (*cells_it).dynAtt(t, cells_.dynAttribute((*f).attr()), value);
        cells_it++;
    }
}

void TeCellAutomata::localMean(LocalMean *f,int t)
{
    cout << "calculating a local mean...\n";
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        string attribute = (*f).column();
        double result=0.00;
        string o1 = (*cells_it).objectId();
        for (BiCellSet::iterator it=cells_.begin(); it!= cells_.end(); it++)
        {
            ProxMatrixInfo pmi =TeProxMatrix::instance().getANeighbor(o1,it);
            if (pmi.empty())
            {
                if (t==0)
                {
                    if (atof((*it).attribute(cells_.attribute(attribute)).c_str()) > 0)
                        result+=pmi.weight();
                }
                else
                {
                    if ((*it).dynAtt(t-1,cells_.dynAttribute(attribute)) > 0)
                        result+=pmi.weight();
                }
            }
        }
        (*cells_it).dynAtt(t, cells_.dynAttribute((*f).attr()), result);
        cells_it++;
    }
}

```

```
void TeCellAutomata::expander(Expander *f, int t)
{
    int globalPotential = 0;
    string attribute=(*f).attr();
    double demand=(*f).demand();
    string column=(*f).column();
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        if ((*cells_it).dynAtt(t, cells_.dynAttribute(column))>0)
            globalPotential++;
        cells_it++;
    }
    cout << "total de celulas com potencial para mudanca: " << globalPotential << endl;
    double threshold = 0.00;
    if (globalPotential>demand)
        threshold = sort(column, demand, t);
    transit(attribute, column, threshold, t);
}
```

```
double TeCellAutomata::sort(string attribute, double demand, int t)
{
    cout << "expanding..." << endl;
    //gera uma lista de potenciais para poder ordenar
    list<double> myList;
    BiCellSet::iterator cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        double value = (*cells_it).dynAtt(t, cells_.dynAttribute(attribute));
        myList.push_back(value);
        cells_it++;
    }
    // ordena potenciais e obtem limiar
    myList.sort();
    double threshold;
    list<double> :: iterator it=myList.end();
    for(int i=0;i<demand;i++)
    {
        threshold=(*it);
        it--;
    }
    return threshold;
}
```

```
void TeCellAutomata:: transit(string attribute, string column, double threshold, int t)
{
    double value;
    BiCellSet::iterator cells_it=cells_.begin();
    cells_it=cells_.begin();
    while (cells_it != cells_.end())
    {
        if ((*cells_it).dynAtt(t, cells_.dynAttribute(column))>=threshold)
            value=1;
        else
            value=atof( (*cells_it).attribute(cells_.attribute(attribute)).c_str() );

        (*cells_it).dynAtt(t, cells_.dynAttribute(attribute), value);

        cells_it++;
    }
}
```

```
int main( int argc, char ** argv )
{
    TeCellAutomata ca(argv[1]);
    ca.execute();
    return 1;
}
```