ANEXO 1

Código fonte das funções relevantes para a implementação da estratégia olho-de-peixe para os browsers do sistema HyperProp:

```
/*********************************************************
 * PUC-Rio                                                *
 * Departamento de Informatica                            *
 * HyperProp Project                                      *
 * Base/HyperBase Browser                                 *
 *********************************************************
 * Fisheye View Functions Interface                       *
 *********************************************************
 * Author: Debora Christina Muchaluat                     *
 * E-Mail: debora@inf.puc-rio.br                          *
 * October / 1995                                         *
 *********************************************************
 * fisheye.h                                              *
 *********************************************************/

#include <hipcliinterface.h>
#include "list.h"

/*********************************************************/
/***************** CLASS DECLARATIONS ******************/
/*********************************************************/

/************************************************
 *                NODE PERSPECTIVE CLASS        *
 ************************************************/

class node_perspective
{
private:
        EntityId        node;                   // ponteiro para no real no MCA
        int                     ident;          // identificador layout grafos D-ABDUCTOR
        int                     api;            // api do no
        int                     de;             // distancia em elos p/ foco
        int                     dc;             // distancia em contextos p/ foco
        int                     d;              // distancia p/ foco
        int                     doi;            // grau de interesse p/ foco
        int                     visible;        // se esta visivel
        int                     visited;        // se ja' foi visitado p/ calcular distancia em elos
        node_perspective*father;                // ponteiro para o no' pai

public:

        List                    sons;           // lista de nos filhos

        node_perspective(EntityId pnode, node_perspective *pfather, int id);  // constructor
        EntityId GetNode();
        void SetNode(EntityId node_version);
        node_perspective *GetFather();
```

```
        void SetIdent(int value);
        int GetIdent();
        void SetApi(int value);
        int GetApi();
        void SetDe(int value);
        int GetDe();
        void CalcDc(node_perspective *focus);
        int GetDc();
        void CalcD();
        int GetD();
        void CalcDoi();
        int GetDoi();
        void Visible();
        void NotVisible();
        int IsVisible();
        void Visited();
        void NotVisited();
        int IsVisited();
};


/***************** FUNCTIONS INTERFACE ******************/

int IsPerspective(node_perspective *focus,node_perspective *node);
int min(int v1, int v2);
node_perspective *FindNodePerspectiveIdent(int ident);
node_perspective *FindIdent(node_perspective *nodep, int ident);
void CalcDe(node_perspective *nodep,int distini);
void InitFisheye();
void InitFisheyeContext();
void InitFisheyeBase();
int ComputeFisheye(node_perspective *focus,int threshold);
void CalculateDc(node_perspective *focus);
void NotVisited(node_perspective *nodep);
void CalculateDoi(int k);
void New_Threshold(int k);
void Collapse(node_perspective *nodep);
void LandMark(node_perspective *nodep);
void NotLandMark(node_perspective *nodep);
int IsLandMark(node_perspective *nodep);
node_perspective *FindSon(node_perspective *context, EntityId son);
```

```
/*********************************************************
 * PUC-Rio                                                *
 * Departamento de Informatica                            *
 * HyperProp Project                                      *
 * Base/HyperBase Browser                                 *
 *********************************************************
 * Fisheye View Functions                                 *
 *********************************************************
 * Author: Debora Christina Muchaluat                     *
 * E-Mail: debora@inf.puc-rio.br                          *
 * October / 1995                                         *
 *********************************************************
 * fisheye.c                                              *
 *********************************************************/

#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include "fisheye.h"
#include "globals.h"

extern Hipinterface hip;    // interface com o MCA

extern float semiv_value;

int is_fisheye_base;                /* if it is fisheye of a private base */
node_perspective *current_context;          // contexto mais externo
List *calcdoi;                              // lista de nos para computar doi
List *land_marks;                           // lista de land marks

int min_api = 0;                    // minimal api

/****************** D-ABDUCTOR FUNCTIONS ******************/

extern "C" {

void mghip_set_semiv();             // especifica se elem aparece no layout
void mghip_set_elem_width();        // seta grossura da linha

}

/*****************************************************************
 *          NODE_PERSPECTIVE CLASS IMPLEMENTATION                *
 *****************************************************************/

/********** NodePerspective Constructor ***********/

node_perspective::node_perspective(EntityId pnode, node_perspective *pfather, int id)
{
        node = pnode;
        father = pfather;
        visible = 0;
        visited = 0;
        dc = -1;
        de = -1;
```

```
	ident = id;
	if (pfather) {
		api = pfather->GetApi() - 1;
		pfather->sons.InsertLast(this);
	}
	else
		api = 0;

	if (api < min_api)
		min_api = api;
}

/********** GetNode **************/

EntityId node_perspective::GetNode()
{
	return node;
}

/********** SetNode **************/

void node_perspective::SetNode(EntityId node_version)
{
	node = node_version;
}

/********** GetFather **************/

node_perspective * node_perspective::GetFather()
{
	return father;
}

/********** SetIdent **************/

void node_perspective::SetIdent(int value)
{
	ident = value;
}

/********** GetIdent **************/

int node_perspective::GetIdent()
{
	return ident;
}

/********** SetApi **************/

void node_perspective::SetApi(int value)
{
	api = value;
}

/********** GetApi **************/
```

```
int node_perspective::GetApi()
{
	return api;
}

/********** SetDe **************/

void node_perspective::SetDe(int value)
{
	de = value;
}

/********** GetDe **************/

int node_perspective::GetDe()
{
	return de;
}

/********** IsPerspective **************/
/* verifica se "node" esta' na mesma perspectiva de "focus" */
int IsPerspective(node_perspective *focus,node_perspective *node)
{
	node_perspective *aux = focus;
	/* verify ascendent line - father */
	while (aux)
		if (aux == node)
			return 1;
		else
			aux = (*aux).GetFather();

	/* verify descendent line - sons */
	if (FindIdent(focus,(*node).GetIdent()))
		return 1;

	return 0;		/* didn't find */
}

/********** CalcDc **************/

/* Calcula distancia em contextos para no em foco */

void node_perspective::CalcDc(node_perspective *focus)
{
	dc = 0;
	node_perspective *aux = focus;
	/* verify ascendent line - father */
	while (aux)
		if (aux == this)
			return;
		else if (FindIdent(aux,(*this).GetIdent())) {
			dc = ((*aux).GetApi() - (*focus).GetApi()) + ((*aux).GetApi() - (*this).GetApi());
			return;
		}
		else {
			aux = (*aux).GetFather();
```

```
                        dc++;
                }
        return;
}

/********** GetDc **************/

int node_perspective::GetDc()
{
        return dc;
}

/********** CalcD **************/

/* calcula distancia */

void node_perspective::CalcD()
{
        if (de != -1)
                d = min(de,dc);
        else
                d = dc;
}

/********** GetD **************/

int node_perspective::GetD()
{
        return d;
}

/********** CalcDoi **************/

/* Calcula funcao de grau de interesse */

void node_perspective::CalcDoi()
{
        doi = api - d;
}

/********** GetDoi **************/

int node_perspective::GetDoi()
{
        return doi;
}

/********** Visible **************/

void node_perspective::Visible()
{
        visible = 1;
}

/********** Not Visible **************/
```

```
void node_perspective::NotVisible()
{
	visible = 0;
}

/********** IsVisible **********/

int node_perspective::IsVisible()
{
	return visible;
}

/********** Visited **********/

void node_perspective::Visited()
{
	visited = 1;
}

/********** NotVisited **********/

void node_perspective::NotVisited()
{
	visited = 0;
}

/********** IsVisited **********/

int node_perspective::IsVisited()
{
	return visited;
}

/********** Minimum **********/

int min(int v1, int v2)
{
	if (v1 <= v2)
		return v1;
	else
		return v2;
}

/****************** Find Node Perspective Ident ******************/

/* encontra o no' na estrutura de perspectivas */

node_perspective *FindNodePerspectiveIdent(int ident)
{
	return FindIdent(current_context,ident);
}

/****************** Find Ident **********************************/

node_perspective *FindIdent(node_perspective *nodep, int ident)
{
```

```
	node_perspective *np,*npfound;

	if ((*nodep).GetIdent() == ident)
		return nodep;

	nodep->sons.RestartList();
	while ((Type *) np = nodep->sons.Current()) {
		npfound = FindIdent(np,ident);
		if (npfound)
			return npfound;
		nodep->sons.GotoNext();
	}
	return NULL;
}

/********** CalcDe **************/

/* Calcula distancia em elos do no p/ todos os outros */

void CalcDe(node_perspective *nodep,int distini)
{
	int distance, answer, i, j, dist_initial;
	char *name;
	EntityId link[NUMLINKS];
	EntityId tp;
	EntityId next_node, node;
	EntityId context;
	node_perspective *np, *father;

	name = new char[ENTITYNAMEMAXSIZE];

	/* para todos os contextos da perspectiva */
	node = (*nodep).GetNode();
	father = (*nodep).GetFather();
	while (father) {

		if ((!father->GetFather()) && (is_fisheye_base)) return;
			/* se estiver na base, nao tem elos */

		context = (*father).GetNode();	/* contexto pai */

		i = 0;
		hip.GetFirstLink(context,&link[i]);
		while (link[i] != 0) {
			i++;
			hip.GetNextLink(context, &link[i]);
		}

		/* para todos os elos do contexto */
		i--;
		while (i >= 0) {

/*			fprintf(stderr,"++++++++++++++++++++++++++++++++\n");
			fprintf(stderr,"%s %d\n","DISTINI:",distini); */
			hip.GetName(node,name);
/*			fprintf(stderr,"%s %s\n","NO:",name); */
```

```
/* se o no' e' origem do elo, encontra o destino */
hip.NodeIsSourceLink(context,link[i],node,&answer);
if (answer) {
        hip.GetFirstSourceTPLink(context,link[i],&tp);
        hip.GetNumElemSourceTP(context,link[i],tp,&dist_initial);
        dist_initial--;      /* menos o proprio no' */

        hip.GetFirstTargetTPLink(context,link[i],&tp);
        hip.GetFirstNodeTargetTP(context,link[i],tp,&next_node);
        j = 1;
        np = father;
        while (next_node != 0) {
                distance = distini + dist_initial + j;

                np = FindSon(np, next_node);
                if (!np) fprintf(stderr,"Elo inconsistente\n");

                if (((*np).GetDe() == -1) || ((*np).GetDe() > distance)) {
                        (*np).SetDe(distance);
                        (*np).NotVisited();
                }

                hip.GetNextNodeTargetTP(context,link[i],tp,&next_node);
                j++;
        }
        hip.GetLastNodeTargetTP(context,link[i],tp,&next_node);
}
else {
        /* se o no' e' destino do elo, encontra a origem */
        hip.NodeIsTargetLink(context,link[i],node,&answer);
        if (answer) {
                hip.GetFirstTargetTPLink(context,link[i],&tp);
                hip.GetNumElemTargetTP(context,link[i],tp,&dist_initial);
                dist_initial--;      /* menos o proprio no' */

                hip.GetFirstSourceTPLink(context,link[i],&tp);
                hip.GetFirstNodeSourceTP(context,link[i],tp,&next_node);
                j = 1;
                np = father;
                while (next_node != 0) {
                        distance = distini + dist_initial + j;

                        np = FindSon(np, next_node);
                        if (!np) fprintf(stderr,"Elo inconsistente\n");

                        if (((*np).GetDe() == -1) || ((*np).GetDe() > distance)) {
                                (*np).SetDe(distance);
                                (*np).NotVisited();
                        }


hip.GetNextNodeSourceTP(context,link[i],tp,&next_node);
                        j++;
                }
                hip.GetLastNodeSourceTP(context,link[i],tp,&next_node);
```

```
                                }
                        }

                        if (answer) {       /* if node is source or target of link */

                                hip.GetName(next_node,name);
/*                              fprintf(stderr,"%s %s %d %s %d\n",name,
                                "Calculou distancia em elos: ",distance,"DE:",(*np).GetDe()); */

                                if (!(*np).IsVisited()) {
                                        (*np).Visited();
                                        if (np != nodep) CalcDe(np,distance);
                                }
                        }

                        i--;
                }

                father = father->GetFather();
        }
}

/******************** Initialize Fisheye ************************/

void InitFisheye()
{
        // just the first nesting level is visible
        // get current context sons and insert in calcdoi list,
        // making them visible

        node_perspective *aux;

        (*current_context).sons.RestartList();

        while ((Type *) aux = current_context->sons.Current()) {
                calcdoi->InsertLast(aux);
                (*aux).Visible();
                (*current_context).sons.GotoNext();
        }
}

/******************** Initialize Fisheye Context ************************/

/* inicializa fisheye de um context, usado no hyperbase browser */

void InitFisheyeContext()
{
        is_fisheye_base = 0;
        calcdoi = new List;
        land_marks = new List;
        InitFisheye();
}

/******************** Initialize Fisheye Private Base ************************/

/* inicializa fisheye de uma base privada, usado no base browser */
```

```
void InitFisheyeBase()
{
	is_fisheye_base = 1;

	if (calcdoi) {
		calcdoi->RestartList();
		while (calcdoi->RemoveCurrent());
	}
	else
		calcdoi = new List;

	if (land_marks) {
		land_marks->RestartList();
		while (land_marks->RemoveCurrent());
	}
	else
		land_marks = new List;

	InitFisheye();
}

/******************** Calculate Dc ************************/

void CalculateDc(node_perspective *focus)
{
	// get calcdoi elements and compute Distance in context to the focus

	node_perspective *aux;

	calcdoi->RestartList();
	while ((Type *) aux = calcdoi->Current()) {
		(*aux).CalcDc(focus);
		calcdoi->GotoNext();
	}
}

/******************** Calculate Doi ************************/

void CalculateDoi(int k)
{
	char		*name;
	EntityId node;
	name			= new char[ENTITYNAMEMAXSIZE];
	node_perspective *aux;
	int		min_api = 0;

	// get calcdoi elements and compute DOI to the focus

	calcdoi->RestartList();
	while ((Type *) aux = calcdoi->Current()) {
		node = (*aux).GetNode();
		hip.GetName(node,name);
		(*aux).CalcD();		/* compute distance */
		(*aux).CalcDoi();		/* compute DOI */
		if ((*aux).GetDoi() >= k) {
```

```
                        (*aux).Visible();
                        mghip_set_semiv(aux->GetIdent(),semiv_value);
                        /* fprintf(stderr,"%s %s %s %d %s %d %s %d %s %d\n","Visible:",name,"API:",

        (*aux).GetApi(),"DC",(*aux).GetDc(),"DE",(*aux).GetDe(),"DOI",(*aux).GetDoi()); */
                        /* calcula minima api dos nos visiveis, para usar no Land Mark */
                        if (aux->GetApi() < min_api)
                                min_api = aux->GetApi();
                }
                else {
                        (*aux).NotVisible();
                        mghip_set_semiv(aux->GetIdent(),0);
                        /* fprintf(stderr,"%s %s %d %d %d %d\n","Not Visible:",
                        name,(*aux).GetApi(),(*aux).GetDc(),(*aux).GetDe(),(*aux).GetDoi()); */
                }
                calcdoi->GotoNext();
        }

        /* verify Land Marks */
        land_marks->RestartList();
        while ((Type *) aux = land_marks->Current()) {
                node = (*aux).GetNode();
                hip.GetName(node,name);
                if (aux->GetApi() >= min_api) {
                        /* (*aux).Visible(); */
                        mghip_set_semiv(aux->GetIdent(),semiv_value);
                        /* fprintf(stderr,"%s %s %s %d\n","Land Mark Visible:",name,
                        "API:",(*aux).GetApi()); */
                }
                else {
                        /* (*aux).NotVisible(); */
                        mghip_set_semiv(aux->GetIdent(),0);
                        /* fprintf(stderr,"%s %s %s %d\n","Land Mark Not Visible:",name,
                        "API:",(*aux).GetApi()); */
                }
                land_marks->GotoNext();
        }
}

/****************** Not Visited ***********************************/

/* marca todos os nos como naos visitados,
   para calcular distancia em elos */

void NotVisited(node_perspective *nodep)
{
        node_perspective *np;

        (*nodep).NotVisited();
        (*nodep).SetDe(-1);
        nodep->sons.RestartList();
        while ((Type *) np = nodep->sons.Current()) {
                NotVisited(np);
                nodep->sons.GotoNext();
        }
        return;
```

```
}

/******************** Compute Fisheye ************************/

/* calcula visao olho-de-peixe, parametros:
   no' em foco / threshold (valor de K) */

int ComputeFisheye(node_perspective *focus, int threshold)
{
        int maxthreshold,k;

        // all the components of the focus node must have their DOI calculated
        node_perspective *aux;
        (*focus).sons.RestartList();
        while ((Type *) aux = focus->sons.Current()) {

                if (!calcdoi->Contain(aux))
                /* if elem is not in the list */
                        calcdoi->InsertLast(aux);

                (*aux).NotVisible();
                (*focus).sons.GotoNext();
        }

        // make visited = false for all the nodes, DE = -1
        NotVisited(current_context);
        CalculateDc(focus);
        (*focus).SetDe(0);
        CalcDe(focus,0);

        /* define minimum detail */
        /* if it is a context node kmax = api_focus - 2,
           else kmax = api_focus - 1 */
        EntityId node = (*focus).GetNode();
        int type;
        hip.GetType(node,&type);
        if (type >= USER_CONTEXT)      /* nos de composicao */
                k = (*focus).GetApi() - 2;
        else                                    /* nos terminais */
                k = (*focus).GetApi() - 1;

        maxthreshold = k;
        // if threshold = 0, take the maximum vale of k
        if ((threshold < k) && (threshold != 0))
                k = threshold;
        CalculateDoi(k);

        return maxthreshold;
}

/******************** New_Threshold ************************/

/* muda o valor de K */

void New_Threshold(int k)
{
```

```
        char            *name;
        EntityId node;
        node_perspective *aux;
        int             min_api = 0;

        name = new char[ENTITYNAMEMAXSIZE];

        // get calcdoi elements and compute DOI to the focus

        calcdoi->RestartList();

        while ((Type *) aux  = calcdoi->Current()) {
                node = (*aux).GetNode();
                hip.GetName(node,name);
                if ((*aux).GetDoi() >= k) {
                        (*aux).Visible();
                        mghip_set_semiv(aux->GetIdent(),semiv_value);
                        /* fprintf(stderr,"%s %s %d %d %d %d\n","Visible:",
                        name,(*aux).GetApi(),(*aux).GetDc(),(*aux).GetDe(),(*aux).GetDoi()); */
                        if (aux->GetApi() < min_api)
                                min_api = aux->GetApi();
                }
                else {
                        (*aux).NotVisible();
                        mghip_set_semiv(aux->GetIdent(),0);
                        /* fprintf(stderr,"%s %s %d %d %d %d\n","Not Visible:",
                        name,(*aux).GetApi(),(*aux).GetDc(),(*aux).GetDe(),(*aux).GetDoi()); */
                }
                calcdoi->GotoNext();
        }

        /* verify Land Marks */
        land_marks->RestartList();
        while ((Type *) aux = land_marks->Current()) {
                node = (*aux).GetNode();
                hip.GetName(node,name);
                if (aux->GetApi() >= min_api) {
                        /* (*aux).Visible(); */
                        mghip_set_semiv(aux->GetIdent(),semiv_value);
                        /* fprintf(stderr,"%s %s %s %d\n","Land Mark Visible:",name,
                        "API:",(*aux).GetApi()); */
                }
                else {
                        /* (*aux).NotVisible(); */
                        mghip_set_semiv(aux->GetIdent(),0);
                        /* fprintf(stderr,"%s %s %s %d\n","Land Mark Not Visible:",name,
                        "API:",(*aux).GetApi()); */
                }
                land_marks->GotoNext();
        }

}

/******************** Collapse **********************/

/* tira os componentes de um no de contexto do calculo
```

```
  da funcao DOI */

void Collapse(node_perspective *nodep)
{
        /* take its components out of the calcdoi list */
        /* update semiv to 0 (ABD) */

        node_perspective *aux;
        char *name = new char[ENTITYNAMEMAXSIZE];

        nodep->sons.RestartList();
        while ((Type *) aux = nodep->sons.Current()) {
                hip.GetName(aux->GetNode(),name);
                mghip_set_semiv(aux->GetIdent(),0);
                calcdoi->Remove(aux);
                Collapse(aux);
                nodep->sons.GotoNext();
        }
}

/******************** Land Mark ***********************/

/* marca um no' e sua perspectiva como land mark */

void LandMark(node_perspective *nodep)
{
        /* coloca todos os nos ascendentes na lista de land marks */

        int width = 6;      /* largura da linha do desenho land mark */
        node_perspective *aux;
        char *name = new char[ENTITYNAMEMAXSIZE];
        aux = nodep;
        while (aux.GetFather()) {
                if (!land_marks->Contain(aux)) {
                        land_marks->InsertLast(aux);
                        EntityId no = aux->GetNode();
                        hip.GetName(no,name);
                        /* printf("Land Mark; %s\n",name); */
                        mghip_set_elem_width(aux->GetIdent(),width);
                }
                else
                        return;   /* se o no for land_mark, todos os ancestrais sao */
                aux = aux.GetFather();
        }
}

/******************** Land Mark ***********************/

/* tira um no' e seus componentes da lista de land marks */

void NotLandMark(node_perspective *nodep)
{
        /* falta tirar diferenciacao dos land marks */

        /* tira toda a perspectiva descendente da lista de land marks */
```

```
        int width = 3;      /* largura da linha do desenho default */
        node_perspective *aux;
        char *name = new char[ENTITYNAMEMAXSIZE];
        EntityId no = nodep->GetNode();
        hip.GetName(no,name);
        /* printf("Nao e' mais Land Mark; %s\n",name); */

        land_marks->Remove(nodep);
        if (!calcdoi->Contain(nodep)) {
                (*nodep).NotVisible();
                mghip_set_semiv(nodep->GetIdent(),0);
        }
        else if (!nodep->IsVisible())
                mghip_set_semiv(nodep->GetIdent(),0);

        mghip_set_elem_width(nodep->GetIdent(),width);   /* tira marca de land mark */

        nodep->sons.RestartList();
        while ((Type *) aux = nodep->sons.Current()) {
                /* se o no nao for land mark, nenhum de seus descendentes e' */
                if (land_marks->Contain(aux))
                        NotLandMark(aux);
                nodep->sons.GotoNext();
        }
}

/******************** Is Land Mark ************************/

/* verifica se um no' e' land mark */

int IsLandMark(node_perspective *nodep)
{
        if (land_marks->Contain(nodep))
                return 1;
        else
                return 0;
}

/******************** Find son ************************/

node_perspective * FindSon(node_perspective *context, EntityId son)
{
        node_perspective *aux;

        context->sons.RestartList();
        while ((Type *) aux = context->sons.Current()) {
                if (aux->GetNode() == son)
                        return aux;
                context->sons.GotoNext();
        }

        /* EntityId parent = context->GetNode(); */

        /* printf("ERROR: Son not found !!!! %d em %d\n",son,parent); */

        return NULL;
```

```
}
```

ANEXO 2

Implementação das classes do Modelo de Contextos Aninhados do sistema HyperProp:

```
// ------------------------------------------------------------------------//
// PUC/RJ - Pontificia Universidade Catolica do Rio de Janeiro
// ------------------------------------------------------------------------//
// Declaracao das classes do Modelo de Contextos Aninhados
// ------------------------------------------------------------------------//

//-------------------------------------------------------------------------//
//*************************************************************************//
//************************* DECLARACAO DAS CLASSES ************************//
//*************************************************************************//
//-------------------------------------------------------------------------//

#ifndef __HIPNEW_H__
#define __HIPNEW_H__

#include "hipglobals.h"
#include "list.h"

// ------------------------------------------------- //
//   DEFINICAO DA CLASSE ENTIDADE   //
// ------------------------------------------------- //
class Entity
{
        private:
                EntityId id;                    // identificador
        public:
                Entity();
                ~Entity() {;}
                EntityId Get_id() {return id;}
                void Set_id(EntityId value) {id = value;}

};

// ------------------------------------------------//
// DEFINICAO DA CLASSE ANCORA      //
// ------------------------------------------------ //

class Anchor : public Entity
{
private:
    int region;      /* a regiao identifica o no inteiro por enquanto (1) */
public:
    Anchor() : Entity() { region = 1; }  // 1  = lambda
};

// ------------------------------------------ //
//   DEFINICAO DA CLASSE NO      //
// ------------------------------------------ //
```

```
class Context;  // declaracao "forward"

class Node : public Entity
{
        friend class Hyperbase; // a hiperbase pode usar os membros privados da classe Node

private:
        char        name[ENTITYNAMEMAXSIZE];

public:
        List anchors;                       // a lista de ancora deve ser visivel pela interface
        Node(char node_name[ENTITYNAMEMAXSIZE]);          // construtor

    // METODOS REFERENTE AO ATRIBUTO NOME

        char * Get_name();                  // retorna o nome para o no
        void Set_name(char *node_name);
        virtual int NodeType() {return NODE_NULL;}              // node
        Anchor *GetAnchor(EntityId anchor_id);
        Node *CreateVersion(char *version_name);                // cria versao do no'

};

// ------------------------------------------------------------ //
//   DEFINICAO DA CLASSE NO TERMINAL     //
// ------------------------------------------------------------ //

class TerminalNode : public Node
{
public:
        TerminalNode(char *node_name) : Node(node_name) {;} // construtor usado na criacao
        ~TerminalNode() {;}

        virtual int NodeType() {return NODE_NULL;}              // terminal node

};

// ------------------------------------------------------------ //
//   DEFINICAO DA CLASSE NO AUDIO        //
// ------------------------------------------------------------ //

class AudioNode : public TerminalNode
{
public:
        AudioNode(char *node_name) : TerminalNode(node_name) {;} // construtor usado na criacao
        ~AudioNode() {;}

        virtual int NodeType() {return AUDIO;}          // audio node

};

// ------------------------------------------------------------ //
//   DEFINICAO DA CLASSE NO VIDEO        //
// ------------------------------------------------------------ //
```

```
class VideoNode : public TerminalNode
{
public:
        VideoNode(char *node_name) : TerminalNode(node_name) {;} // construtor usado na criacao
        ~VideoNode() {;}

        virtual int NodeType() {return VIDEO;}          // video node

};

// ------------------------------------------------------ //
//   DEFINICAO DA CLASSE NO IMAGE       //
// ------------------------------------------------------ //

class ImageNode : public TerminalNode
{
public:
     ImageNode(char *node_name) : TerminalNode(node_name) {;} // construtor usado na criacao
        ~ImageNode() {;}

        virtual int NodeType() {return IMAGE;}          // image node

};

// ------------------------------------------------------ //
//   DEFINICAO DA CLASSE NO TEXTO       //
// ------------------------------------------------------ //

class TextNode : public TerminalNode
{
public:
     TextNode(char *node_name) : TerminalNode(node_name) {;} // construtor usado na criacao
        ~TextNode() {;}

        virtual int NodeType() {return TEXT;}           // text node

};

// ------------------------------------------------------------//
//   DEFINICAO DA CLASSE TERMINAL POINT  //
// ------------------------------------------------------------//

class TerminalPoint : public Entity
{
        private:
                Anchor *anchor;         // ancora
        public:
                List nodes;             // lista de nos, o ultimo no e' o que
                                        // tem a ancora

                TerminalPoint() : Entity() {anchor = new Anchor;}
                ~TerminalPoint();
                Anchor *Get_anchor()            {return anchor;}
                void Set_anchor(Anchor *value)  {anchor = value;}
};
```

```
// ---------------------------------------------------------//
//   DEFINICAO DA CLASSE SENTENCE       //
// ---------------------------------------------------------//

class Sentence : public Entity
{
        private:
                int action;                     // acao
        public:
                List source_terminal_points;    // pontos terminais origem
                List target_terminal_points;    // pontos terminais destino

                Sentence();
                ~Sentence();

                int Get_action()                {return action;}
                void Set_action(int value)      {action = value;}
};

// -----------------------------------------//
//   DEFINICAO DA CLASSE ELO //
// -----------------------------------------//

class Link : public Entity
{
 friend Context; // a classe contexto pode acessar os membros privados de elo.
 private:
        Sentence *sentence;                     // sentenca

 public:
        List source_terminal_points;            // pontos terminais origem
        List target_terminal_points;            // pontos terminais destino

        Link();
        ~Link();
        void Set_sentence(Sentence *sent) {sentence = sent;}
        Sentence * Get_sentence() {return sentence;}
        TerminalPoint *GetSourceTP(EntityId tp_id);
        TerminalPoint *GetTargetTP(EntityId tp_id);
};

// ---------------------------------------------------------- //
//   DEFINICAO DA CLASSE COMPOSICAO  //
// ---------------------------------------------------------- //

class Composite : public Node
{
private:
public:
     List nodes;                    // lista de nos
     List links;                    // lista de elos

     Composite(char *node_name) : Node(node_name) {;}
        ~Composite() {;}
        virtual int NodeType() {return NODE_NULL;}              // composite node
        Node * GetNode(char *);         // Retorna o ponteiro para o no'cujo
```

```
                                        // nome foi passado como parametro
        Node * GetNode(EntityId);       // Retorna o ponteiro para o no'cujo
                                        // id foi passado como parametro
        Link * GetLink(EntityId);
};

// ------------------------------------------------------ //
//   DEFINICAO DA CLASSE PERSPECTIVA  //
// ------------------------------------------------------ //
class Perspective : public Entity
{
private:
        Node *node;                     // no' que possui a perspectiva

public:
        List ancestors;                 // lista de contextos ancestrais do no'

        Perspective(Node *n);
        ~Perspective();

        Node * Get_node()       {return node;}


};

// ------------------------------------------------ //
//   DEFINICAO DA CLASSE TRILHA  //
// ------------------------------------------------ //

class Trail : public Composite
{
    friend class PrivateBase; // a base privada pode usar os membros privados da trilha

private:
        Context *associated_context;    /* no' `a que a trilha esta associada */

public:
        List views;     /* lista de visoes: perspectivas dos nos da trilha */

        Trail(char *trail_name, Context *context_node);
        ~Trail() {;}
        virtual int NodeType() {return TRAIL;}          // trail node

        Context * Get_associated_context() {return associated_context;}
        void Set_associated_context(Context *context) {associated_context = context;}
        Perspective * Current();                // no corrente
        Perspective * First();                  // navega p/ o primeiro
        Perspective * GotoNext();               // navega para o proximo
        Perspective * GotoPrevious();           // navega para o anterior
};

// ------------------------------------------------- //
//   DEFINICAO DA CLASSE CONTEXTO  //
// ------------------------------------------------- //

class Context : public Composite
```

```
{
public:

      Context(char *);                  // construtor usado na criacao
            ~Context();
            virtual int NodeType() {return NODE_NULL;}                  // context node

};

// ------------------------------------------------------ //
//   DEFINICAO DA CLASSE HYPERBASE  //
// ------------------------------------------------------ //

class Hyperbase : public Context
{
public:
            Hyperbase(char *base_name) : Context(base_name) {;}
            ~Hyperbase()      {;}
            virtual int NodeType() {return HYPERBASE;}                  // hyperbase node
};

// ------------------------------------------------------------ //
//   DEFINICAO DA CLASSE PRIVATE_BASE  //
// ------------------------------------------------------------ //

class PrivateBase : public Context
{
private:
            Trail *system_trail;                  // system private base trail

public:
            PrivateBase(char *base_name);
            ~PrivateBase();
            virtual int NodeType() {return PRIVATE_BASE;}   // privte base node
            Trail * Get_system_trail() { return system_trail; }
            void CheckOut(Node *node);                              // checkout
            void Shift();                                           // shift
            Node * CheckIn(Node *node, char *version_name); // checkin
            Node * Open(Node *node, char *version_name);     // open
};

// ------------------------------------------------------------------------ //
//   DEFINICAO DA CLASSE CONTEXTO DO USUARIO  //
// ------------------------------------------------------------------------ //

class UserContext : public Context
{
private:
public:

            UserContext(char *node_name) : Context(node_name) {;}
            ~UserContext() {;}
            virtual int NodeType() {return USER_CONTEXT;}            // context node

};
```

```
#endif
```

```
// ------------------------------------------------------------------------//
// PUC/RJ - Pontificia Universidade Catolica do Rio de Janeiro
// ------------------------------------------------------------------------//
// Implementacao dos Metodos das Classes de Contextos Aninhados
// declarados em hipengine.h
// ------------------------------------------------------------------------//

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "hipengine.h"

//-------------------------------------------------------------------------//
//*************************************************************************************//
//*** IMPLEMENTACAO DOS METODOS DAS CLASSES DECLARADOS EM hipengine.h ********//
//*************************************************************************************//
//-------------------------------------------------------------------------//

//********************** GLOBAL OBJECTS ************************//

Hyperbase hb("hyperbase");          /* node repository -> */
        /* contem todos os nos terminais e de contexto de usuario */
Hyperbase public_hb("public");      /* public hyperbase */
Hyperbase trails_repository("hyperprop_trails");/* trails repository */
EntityId counter = 0;               /* contador de identificadores */
List isolated_terminalpoints;       /* lista de pontos terminais isolados */
List isolated_sentences;    /* lista de sentencas isoladas */
List isolated_perspectives;         /* lista de perspectivas isoladas */
        /* estas listas de entidades isoladas possuem entidades que ainda
        nao foram inseridas em outras entidades definidas no MCA,
        por exemplo, elos e trilhas */
List private_bases;                 /* lista de todas as bases privadas */
List trails;                        /* lista de todos as trilhas */

// ---------------------------------------------------------- //
// IMPLEMENTACAO DA CLASSE ENTIDADE //
// ---------------------------------------------------------- //

Entity::Entity()
{
        id = ++counter;
}

// ------------------------------------------------- //
// IMPLEMENTACAO DA CLASSE NO    //
// ------------------------------------------------- //

Node::Node(char node_name[ENTITYNAMEMAXSIZE]) : Entity()
{
         strcpy(name,node_name);
}

char *Node::Get_name()
{
   return name;
}
```

```
void Node::Set_name(char *node_name) {
   strcpy(name,node_name);
}

Anchor * Node::GetAnchor(EntityId anchor_id)
{
        return (Anchor *)anchors.GetElem(anchor_id);

}

// ---------------------------------------- //
// CRIA VERSAO DE UM NO'        //
// ---------------------------------------- //

Node * Node::CreateVersion(char *version_name)
{
        switch (NodeType()) {
                case TEXT:
                        TextNode *tnode = new TextNode(version_name);
                        hb.nodes.InsertLast((Node *)tnode);
                        return tnode;
                case AUDIO:
                        AudioNode *anode = new AudioNode(version_name);
                        hb.nodes.InsertLast((Node *)anode);
                        return anode;
                case VIDEO:
                        VideoNode *vnode = new VideoNode(version_name);
                        hb.nodes.InsertLast((Node *)vnode);
                        return vnode;
                case IMAGE:
                        ImageNode *inode = new ImageNode(version_name);
                        hb.nodes.InsertLast((Node *)inode);
                        return inode;
                case USER_CONTEXT:
                        UserContext *context_version = new UserContext(version_name);
                        UserContext *context = (UserContext *) this;
                        Node *component;
                        context->nodes.RestartList();
                        while ((Type *)component = context->nodes.Current()) {
                                context_version->nodes.InsertLast(component);
                                context->nodes.GotoNext();
                        }
                        Link *link;
                        context->links.RestartList();
                        while ((Type *)link = context->links.Current()) {
                                context_version->links.InsertLast(link);
                                context->links.GotoNext();
                        }
                        hb.nodes.InsertLast((Node *)context_version);
                        return context_version;
                default: return NULL;
        }
        /* atualizar contexto de versoes */
}
```

```
// ------------------------------------------------------------------//
//   IMPLEMENTACAO DA CLASSE SENTENCE       //
// ------------------------------------------------------------------//

Sentence::Sentence() : Entity()
{
        action = 1;                             // 1 = acao play
}

Sentence::~Sentence()
{
         source_terminal_points.RestartList();
         while( source_terminal_points.RemoveCurrent() );

         target_terminal_points.RestartList();
         while( target_terminal_points.RemoveCurrent() );

}

// ------------------------------------------------- //
// IMPLEMENTACAO DA CLASSE ELO   //
// ------------------------------------------------- //

Link::Link() : Entity()
{
}

TerminalPoint * Link::GetSourceTP(EntityId tp_id)
{
        TerminalPoint *aux;
        source_terminal_points.RestartList();
        while((Type *) aux = source_terminal_points.Current())
        {
                if (aux->Get_id() == tp_id)
                        return aux;
                source_terminal_points.GotoNext();
        }
        return NULL;

}

TerminalPoint * Link::GetTargetTP(EntityId id1)
{
        TerminalPoint *aux;
        target_terminal_points.RestartList();
        while((Type *) aux = target_terminal_points.Current())
        {
                if (aux->Get_id() == id1)
                        return aux;
                target_terminal_points.GotoNext();
        }
        return NULL;

}

// ------------------------------------------------------------ //
```

```
// IMPLEMENTACAO DA CLASSE COMPOSICAO //
// ---------------------------------------------------------- //

Node * Composite::GetNode(char *name)
{
	Node *aux;
	nodes.RestartList();
	while((Type *) aux = nodes.Current())
	{
		if (!strcmp(aux->Get_name(), name))
			return aux;
		nodes.GotoNext();
	}
	return NULL;

}

Node * Composite::GetNode(EntityId id1)
{
	Node *aux;
	nodes.RestartList();
	while((Type *) aux = nodes.Current())
	{
		if (aux->Get_id() == id1)
			return aux;
		nodes.GotoNext();
	}
	return NULL;

}

Link * Composite::GetLink(EntityId id1)
{
	Link *aux;
	links.RestartList();
	while((Type *) aux = links.Current())
	{
		if (aux->Get_id() == id1)
			return aux;
		links.GotoNext();
	}
	return NULL;

}

// ---------------------------------------------------------- //
// IMPLEMENTACAO DA CLASSE CONTEXTO //
// ---------------------------------------------------------- //

Context::Context(char *node_name) : Composite(node_name)
{
	return;
}

Context::~Context()
{
```

```
        nodes.RestartList();
        while( nodes.RemoveCurrent() );

        links.RestartList();
        while( links.RemoveCurrent() );

}

// ------------------------------------------------------------------------//
//   IMPLEMENTACAO DA CLASSE PERSPECTIVA       //
// ------------------------------------------------------------------------//

Perspective::Perspective(Node * n) : Entity()
{
        node = n;
}

Perspective::~Perspective()
{
        ancestors.RestartList();
        while( ancestors.RemoveCurrent() );
}

// ------------------------------------------------------- //
// IMPLEMENTACAO DA CLASSE TRILHA   //
// ------------------------------------------------------- //

Trail::Trail(char *trail_name, Context *context_node) : Composite(trail_name)
{
        associated_context = context_node;
}

Perspective * Trail::Current()      /* perspectiva do no corrente da trilha */
{
        return (Perspective *) views.Current();
};

Perspective * Trail::First()        /* perspectiva do primeiro no da trilha */
{
        views.RestartList();
        return (Perspective *) views.First();
};

Perspective * Trail::GotoNext()             /* vai para o proximo no da trilha */
{
        views.GotoNext();
        return (Perspective *) views.Current();
};

Perspective * Trail::GotoPrevious()         /* vai para o no anterior da trilha */
{
        views.GotoPrevious();
        return (Perspective *) views.Current();
};

// ------------------------------------------------------------------- //
```

```
// IMPLEMENTACAO DA CLASSE BASE PRIVADA   //
// ---------------------------------------------------------------- //

PrivateBase::PrivateBase(char *base_name) : Context(base_name)
{
        /* create system private base trail */
        char *trail_name = new char [ENTITYNAMEMAXSIZE];
        strcpy(trail_name, base_name);
        strcat(trail_name, "_trail");
        system_trail = new Trail(trail_name,(Context *) this);
        trails.InsertLast(system_trail);
}

PrivateBase::~PrivateBase()
{
        trails.Remove(system_trail);
}

// ---------------------------------------------------------------- //
// CHECKOUT EM UM NO' DA BASE PRIVADA     //
// ---------------------------------------------------------------- //

void PrivateBase::CheckOut(Node *node)
{
        int type = node->NodeType();
        if (type <= USER_CONTEXT) {    /* se for terminal ou user_context */
                if (type == USER_CONTEXT) {
                /* se for user_context, da' checkout nos componentes */
                        Node *aux;
                        Composite *composite = (Composite *)node;
                        composite->nodes.RestartList();
                        while ((Type *)aux = composite->nodes.Current()) {
                                CheckOut(aux);
                                composite->nodes.GotoNext();
                        }
                }
                if (!public_hb.nodes.Contain(node)) {
                        public_hb.nodes.InsertLast(node);
                        printf("dei checkout em : %s\n",node->Get_name());
                }
                nodes.Remove(node);
        }
}

// ------------------------------------------------ //
// SHIFT NA BASE PRIVADA                  //
// ------------------------------------------------ //

void PrivateBase::Shift()
{
        Node *node;
        int type;
        nodes.RestartList();
        while ((Type *)node = nodes.Current()) {
                type = node->NodeType();
                switch (type) {
```

```
                        case PRIVATE_BASE:
                                PrivateBase *pb = (PrivateBase *) node;
                                pb->Shift();
                                nodes.GotoNext();
                                break;
                        case ANNOTATION:
                        case TRAIL:
                                nodes.Remove(node);/* o remove reinicia a lista de nos (RestartList) */
                                Trail *trail = (Trail *)node;
                                if (!trails_repository.nodes.Contain(trail)) {
                                        trails_repository.nodes.InsertLast(trail);
                                        /* se a trilha estiver associada a base, associa-a a hiperbase */
                                        if (trail->Get_associated_context() == this)
                                                trail->Set_associated_context((Context *)&public_hb);
                                }
                                break;
                        default:
                                CheckOut(node);/* o remove reinicia a lista de nos (RestartList) */
                                break;
                }
        }
}

// ------------------------------------------------------------ //
// CHECKIN EM UM NO' NA BASE PRIVADA      //
// ------------------------------------------------------------ //

Node * PrivateBase::CheckIn(Node *node, char *version_name)
{
        Node *aux = node->CreateVersion(version_name);
        nodes.InsertLast(aux);
        return aux;
}

// ------------------------------------------------------------ //
// OPEN EM UM NO' NA BASE PRIVADA         //
// ------------------------------------------------------------ //

Node * PrivateBase::Open(Node *node, char *version_name)
{
        Node *version = CheckIn(node,version_name);
        int type = node->NodeType();

        if (type == USER_CONTEXT) {
                /* cria versoes para todos os componentes do no */
                Node *aux1, *aux2;
                char *name = new char[ENTITYNAMEMAXSIZE];
                Composite *composite = (Composite *)node;
                Composite *version_composite = (Composite *)version;

                composite->nodes.RestartList();
                while ((Type *)aux1 = composite->nodes.Current()) {
                        strcpy(name,aux1->Get_name());
                        /* nome da versao = nome da ultima versao do no + "V" */
                        strcat(name,"V");
                        while (hb.nodes.GetNamedElem(name))
```

```
                    strcat(name,"V");
                aux2 = Open(aux1,name);
                version_composite->nodes.Remove(aux1);
                version_composite->nodes.InsertLast(aux2);
                composite->nodes.GotoNext();
            }
        }
        return version;
}
```