PODERICO LUIGI

OTI

AN OPTIMIZATION TEMPLATE INTERFACE

LUPODERI@TISCALI.IT
WWW.PODERICO.IT

OTI

AN OPTIMIZATION TEMPLATE INTERFACE

INTRODUCTION

This document describe a proposal for a c++ template based abstraction interface for mathematical optimization software.

The major problem for an optimization software writer is to switch from a solver to another. This because every solver has got its own interface. The most famous project facing this problem is OSI, that uses abstract classes and virtual functions to build and to implement the general interface and the several implementations.

My proposal is to use template and the basic concepts of template meta programming. This allows solver abstraction with no run-time costs. I've reused many concepts and ideas from STL and boost.

Using the most common subdivision, oti library provides *environment* and *problem* classes. Each class is a template on an integer representing the solver to implement. Currently three solvers are provided as example: cplex, osl and mosek. Just the mosek implementation is used and tested in real applications.

From the user point of view, the user:

1. selects which solver to use;
2. using the type generator asks to OTI to generate the right types for an environment and a problem;
3. uses them with the common interface.

See the commented example for further details.

A classes list is provided where to find a short reference.

BASIC CONCEPTS

ENVIRONMENT AND PROBLEM

An environment is the memory space where solver parameters are stored. A solver uses an environment to solve a problem.

In OTI, a generic environment is represented by OptEnvironment class.

A problem represents several concepts:

1. the problem to resolve;
2. the operations to build and modify it;
3. the operations to solve it.

In my opinion at this class too many responsibilities are assigned. But this is what the common commercial solver do and so what we have done in oti. In particular we have used the CPLEX implementation in the definitions of the operations that build, modify and solve a problem.

In OTI, a generic problem is represented by OptProblem class.

With the term *solver* we intend a solver ables to solve the following problems:

1. Linear programmin
2. Integer linear programming
3. Quadratic programming

The solver supported by OTI are: mosek, osl and mosek. They are represented by the enumeration eSolverKind.

TYPE GENERATOR

A type generator is one of the concepts of template meta programming. It allow the programmer to generate at compile time the types to use for a specific task.

Let we take an example. Suppose that we want to use mosek to solve a problem. So what we need is an environment class and a problem task that use mosek as solver.

With oti this is translated as following:

```
// Generate the type
typedef oti::OptGenerator<oti::eMsk>::tProblem TOptProblem;
typedef oti::OptGenerator<oti::eMsk>::tEnvironment TOptEnvironment;

//
```

```
TOptProblem myOptEnv;
TOptEnvironment myOptProblem(myOptEnv, "Problem name");

// Setup the environment
myOptEnv.setParam(EnvImplementation<oti::eMsk>::eRtolpinf, 1e-8);

// Use the problem
myOptProblem.chgprobtype(OTIMPSolver::OptProblem::eLp);
```

COMMENTED EXAMPLE

Let we take a look at an oti usage example. Supposing to use mosek as solver.

The main oti paradigm is to have an environment and a problem. These are obtained in two steps: 1) generating the right types and 2) instantiating the related objects.

The enumeration oti::eMsk indicates the solver to instantiate.

```
typedef oti::OptGenerator<oti::eMsk>::tProblem TOptProblem;
typedef oti::OptGenerator<oti::eMsk>::tEnvironment TOptEnvironment;
//
TOptProblem myOptEnv;
TOptEnvironment myOptProblem(myOptEnv, "Problem name");
```

Now we change the environment setting the tolerance that stops the algorithm.

```
myOptEnv.setParam(EnvImplementation<oti::eMsk>::eRtolpinf, 1e-8);
```

Now the problem is constructed. We say that the problem is a linear programming ones; that has got a given number of variables with an objective function and range constraints; that the problem is to minimise the objective function; that the coefficients matrix is built from a sparse representation

```
myOptProblem.chgprobtype(OTIMPSolver::OptProblem::eLp);
myOptProblem.newcols( myNumCols , &myObj[0] , &myLb[0] , &myUb[0] , NULL) ;
myOptProblem.chgobjsen( OtiStatic::MINIMIZE );
myOptProblem.addrows(0, myRowCounts, myNonZero, &myRhs, &mySense,
myMatrixBegin, myMatrixIndex, myMatrixVals);
```

Now we write the problem on the disk to check that everything is ok.

```
myOptProblem.write( "OTIMP.lp" );
```

The problem is solved using a primal simplex.

```
myOptProblem.primopt()
```

Now we are able to get the objective value and the primal variables.

```
myOptProblem.objval();
myOptProblem.getX(myPrimals);
```

CLASS LIST

ENVIMPLEMENTATION

This class represent a placeholder for the pointer of the implementation specific environment. When we define the template it is unknown so it is empty.

```
template <int Solver>
class EnvImplementation
{
};
```

OPTENVIRONMENT

This template represent the generic interface for any environment implementations.

```
template <int Solver>
class OptEnvironment
{
private:
    OptEnvironment (const OptEnvironment &rhs);
    OptEnvironment& operator= (const OptEnvironment& rhs);

    EnvImplementation<Solver>* fEnvImplementation;

public:
    OptEnvironment();
    ~OptEnvironment();

    EnvImplementation<Solver>* GetEnvironment();

    void getParam (int paramnum, int& rvalue) const;
    void getParam (int paramnum, double& rvalue) const;

    void setParam (int paramnum, int rvalue);
    void setParam (int paramnum, double rvalue);
};
```

OTIEXCEPTION

If some OptProblem operation incurs in an error an OtiException is raised. The function name and an error code are returned.

```
class OtiException
{
public:
    OtiException( int status , const char* funcname = "unknown" )
        : fStatus( status ) , fFuncname( funcname ) {}

        const char* funcname () const;

        int errcode () const;
};
```

OTISTATIC

This class collect constants used when interacting with a OptProblem object.

```
class OtiStatic
{
public:
    static char* kLowerBound /* = "L" */ ;
    static char* kUpperBound /* = "U" */ ;
    static char* kBound /* = "B" */ ;


    enum eVarType
    {
        eContinuos = 0,
        eBinary,
        eInteger
    };

    enum eConstrType
    {
        eEqualTo = 0,
        eLessThen,
        eGreaterThen,
        eRanged
    };

    enum ObjSense { MAXIMIZE = -1, MINIMIZE = 1 };
};
```

OPTIMPLEMENTATION

This class represent a placeholder for the pointer of the implementation specific problem. When we define the template it is unknown so it is empty.

```
template <int Solver>
struct OptImplementation
{
};
```

OPTPROBLEM

This class define a generic problem interface. A cplex like interface it is adopted.

```
template <int Solver>
class OptProblem
{
   OptProblem (OptEnvironment<Solver>& renv,
        const char* probname = "problem");
    ~OptProblem ();

    void write (const char *filename,
        const char *filetype = NULL) const;

    enum eProblemType
    {
        eUnset = -1,
        eLp = 0, // Linear problem
        eMip, // Mixed integer problem
        eQp, // Quadratic problem
        eMIQP // Mixed quadratic integer problem
    };
```

```
void chgprobtype(int type);

eProblemType getprobtype() const;

void chgbds (int cnt, int indices[], char lu[], double bd[]);
void chgrhs (int cnt, int indices[], double values[]);
void chgrngval
    (int cnt, int indices[], double values[]);
void chgcoef(int i, int j, double val);
void chgqpcoef
    (int i, int j, double newvalue);
void newrows(int rcnt, double rhs[],
    OtiStatic::eConstrType sense[], double rngval[] = 0,
    char *rowname[] = 0);
void newcols(int ccnt, double obj[],
    const double lb[], const double ub[],
    OtiStatic::eVarType ctype[] = 0, const char* colname[] = 0);
void addrows(int ccnt, int rcnt, int nzcnt,
    double rhs[], OtiStatic::eConstrType sense[], int rmatbeg[],
    const int rmatind[], const double rmatval[],
    char *colname[] = 0, char *rowname[] = 0);
void delrows(int begin, int end);
void delcols(int begin, int end);
void chgobjsen
    (OtiStatic::ObjSense objsen);
void chgobj
    (int cnt, int indices[], double values[]);


void primopt ();
void dualopt ();
void baropt ();
void qpopt();
void mipopt();

int stat () const;

enum eOTIReturnCode
{
    eUnsolved = -1,
    eOptimal = 0,
    eFeseable,
    eUnFeasible,
    eUnBounded,
    eError
};

/*
stat() returns the original return code of the solver, each
solver having its own set of different return values, while
otistat() translates the return codes into a (possibly smaller
and less accurate) set of standard return values.
*/

eOTIReturnCode otistat () const;

double objval () const;


void getX (double x[], int begin = 0, int end = -1) const;
void getPi (double Pi[], int begin = 0, int end = -1) const;
void getPiLB (double Pi[], int begin = 0, int end = -1) const;
void getPiUB (double Pi[], int begin = 0, int end = -1) const;
```

```
    void getslack (double slack[], int begin = 0, int end = -1) const;
    void getdj (double& dj, int begin, int end) const;
    void getcoef(int row, int col, double& coef) const;
    void getrhs (double rhs[], int begin, int end) const;
    void getrows (int& nzcnt_p, int* rmatbeg, int* rmatind,
        double* rmatval, int rmatspace, int& surplus_p,
        int begin, int end) const;
    void getax(double ax[], int begin, int end) const;
    void getobj(double obj[], int begin, int end) const;

    int numcols () const;
    int numrows () const;
    static double getinfinite ();

    // If any, read and modify the environment params
    void getParam (int paramnum, int& rvalue) const;
    void getParam (int paramnum, double& rvalue) const;

    void setParam (int paramnum, int rvalue);
    void setParam (int paramnum, double rvalue);
};
```

OPTGENERATOR

Utility class. Useful as type generator.

```
template <int Solver>
class OptGenerator
{
public:
    typedef OptEnvironment<Solver> tEnvironment;
    typedef OptProblem<Solver> tProblem;
};
```

CONCLUSION

The OTI library is currently used in commercial and academic software. This demonstrate two things:

1. it is possible to apply the template meta-programming also in operational research fields;
2. it is possible to increase the performances when a generic solver is used with a unified interface.

The use of OTI require a good c++ template knowledge, especially when compiler errors incures.

It is possible to download the OTI library at www.poderico.it/oti. Any comment is appreciated at lupoderi@tiscali.it