UE MOBJ [4L103]

Jean-Paul Chaput
Jean-Paul.Chaput@lip6.fr

SESI

2018-2019

IV.3

```
// Term.h
#ifndef  SCHEMATIC_TERM_H
#define  SCHEMATIC_TERM_H
class Term {
    // ...
};
#endif
```

```
// Instance.h
#ifndef  SCHEMATIC_INSTANCE_H
#define  SCHEMATIC_INSTANCE_H
#include "Term.h"

class Instance {
    // ...
};
#endif
```

```
// Cell.h
#ifndef  SCHEMATIC_CELL_H
#define  SCHEMATIC_CELL_H
#include "Term.h"
#include "Instance.h"
class Cell {
    // ...
};
#endif
```

IV.3

```
// Instance.h
#ifndef  SCHEMATIC_INSTANCE_H
#define  SCHEMATIC_INSTANCE_H
#include "Cell.h"
class Instance {
  // ...
  Cell* getCell() const;
};

#endif
```

```
// Cell.h
#ifndef  SCHEMATIC_CELL_H
#define  SCHEMATIC_CELL_H
#include "Instance.h"
class Cell {
  // ...
  Instance* getInstance
    (const std::string&) const;
};
#endif
```

```
// Cell.cpp
#include "Instance.h"
#include "Cell.h"

Instance* Cell::getInstance(const std::string&) const
{ }
#endif
```

IV.5

```
class Cell {
  private:
    static  std::vector<Cell*>        cells_;
            std::string               name_;
            std::vector<Term*>        terms_;
            std::vector<Instance*>    instances_;
            std::vector<Net*>         nets_;
            unsigned int              maxNetIds_;
};
```

IV.5

```
class Cell {
                   Cell             ( const std::string& );
                  ~Cell             ();
  const std::string&                getName       () const;
  const std::vector<Instance*>&     getInstances  () const;
  const std::vector<Term*>&         getTerms      () const;
  const std::vector<Net*>&          getNets       () const;
        Instance* getInstance       ( const std::string& ) const;
        Term*     getTerm           ( const std::string& ) const;
        Net*      getNet            ( const std::string& ) const;
};
```

IV.5

```
class Cell {
  public:
    void            add      ( Instance* );
    void            add      ( Term* );
    void            add      ( Net* );
    void            remove   ( Instance* );
    void            remove   ( Term* );
    void            remove   ( Net* );
    bool            connect  ( const std::string& name, Net* );
    unsigned int    newNetId ();
};
```

IV.5

```
// Cell.h
class Cell {
  public:
    static std::vector<Cell*>& getAllCells ();
    static Cell*               find ( const std::string& );
  //...
};

// Cell.cpp
vector<Cell*>  Cell::cells_;
```

IV.5

```
Cell* Cell::find ( const string& name ) {
  for( size_t i=0 ; i < cells_.size() ; ++i ) {
    if (cells_[i]->getName() == name) return cells_[i];
  }
  return NULL;
}

Cell::Cell ( const string& name ) : name_      (name)
                                  , terms_     ()
                                  , instances_()
                                  , nets_      ()
                                  , maxNetIds_(0) {
  if (find(name)) {
    cerr << "[ERROR]␣Attempt␣to␣create␣duplicate␣of␣Cell␣<"
         << name << ">.\n" << "␣␣␣␣␣␣␣␣Aborting..." << endl;
    exit( 1 );
  }
  cells_.push_back( this );
}
```

IV.5

```
Cell::~Cell ()
{
  for ( vector<Cell*>::iterator icell=cells_.begin()
      ; icell != cells_.end() ; ++icell ) {
    if (*icell == this) {
      cells_.erase( icell );
      break;
    }
  }
}
```

IV.6

```
class Instance {
  private:
    Cell*                     owner_;
    Cell*                     masterCell_;
    std::string               name_;
    std::vector<Term*>        terms_;
    Point                     position_;
};
```

IV.7

```
class Instance {
    Instance   ( Cell* owner, Cell* model, const std::string& );
   ~Instance   ();
    const std::string&
               getName        () const;
    Cell*      getMasterCell  () const;
    Cell*      getCell        () const;
    const std::vector<Term*>&
               getTerms       () const;
    Term*      getTerm        ( const std::string& ) const;
    Point      getPosition    () const;
    bool       connect        ( const std::string& name, Net* );
    void       add            ( Term* );
    void       remove         ( Term* );
    void       setPosition    ( const Point& );
    void       setPosition    ( int x, int y );
};
```

IV.8

```
class Node {
  public:
    static const size_t  noid; // numeric_limits<size_t>::max();
  public:
                    Node           ( Term*, size_t id=noid );
                   ~Node           ();
    inline  Point   getPosition () const;
    inline  void    setPosition ( const Point& );
    inline  void    setPosition ( int x, int y );
    inline  size_t  getId          () const;
            Net*    getNet         () const;
    inline  Term*   getTerm        () const;
    inline  void    setId          ( size_t );
            void    toXml          ( std::ostream& ) const;
  protected:
    size_t  id_;
    Term*   term_;
    Point   position_;
};
```

IV.8

```
class Term {
  public:
    enum Type      { Internal=1, External=2 };
    enum Direction { In=1, Out=2, Inout=3, Tristate=4, Transcv=5
                   , Unknown=6 };
  private:
    void*          owner_;
    std::string    name_;
    Direction      direction_;
    Type           type_;
    Net*           net_;
    Node           node_;
};
```

IV.8

```
class Term {
  public:
    Term      ( Cell* , const std::string& name, Direction );
    Term      ( Instance*, const Term* modelTerm );
   ~Term      ();
    bool                isInternal   () const;
    bool                isExternal   () const;
    const std::string& getName      () const;
    Node*               getNode      ();
    Net*                getNet       () const;
    Cell*               getCell      () const;
    Cell*               getOwnerCell () const;
    Instance*           getInstance  () const;
    Direction           getDirection () const;
    Point               getPosition  () const;
    Type                getType      () const;
    void                setNet       ( Net* );
    void                setNet       ( const std::string& );
    void                setPosition  ( const Point& );
    void                setPosition  ( int x, int y );
    void                setDirection ( Direction );
};
```

IV.8

```
Cell* Term::getCell () const
{ return (type_ == External) ? static_cast<Cell*>(owner_)
                              : NULL; }

Instance* Term::getInstance () const
{ return (type_ == Internal) ? static_cast<Instance*>(owner_)
                              : NULL; }
```

IV.9

```
class Cell;
class Node;

class Net {
  private:
    Cell*                    owner_;
    std::string              name_;
    unsigned int             id_;
    Term::Type               type_;
    std::vector<Node*>       nodes_;
};
```

IV.9

```
class Net {
  public:
                              Net           ( Cell*
                                            , const std::string&
                                            , Term::Type );
                             ~Net           ();
        Cell*                 getCell       () const;
        const std::string&    getName       () const;
        unsigned int          getId         () const;
        Term::Type            getType       () const;
        Node*                 getNode       ( size_t id ) const;
        const std::vector<Node*>&
                              getNodes      () const;
        size_t                getFreeNodeId () const;
        void                  add           ( Node* );
        bool                  remove        ( Node* );
};
```

IV.10

```
class Net {
  public:
      Net ( Cell*, const std::string& name, Term::Type dir );
      Net ( Instance*, const std::string& name, Term::Type dir );
     ~Net ();
  private:
      Net ( const Net& );
  //...
};
```