Design choices for optimization applications

Susanne Heipcke

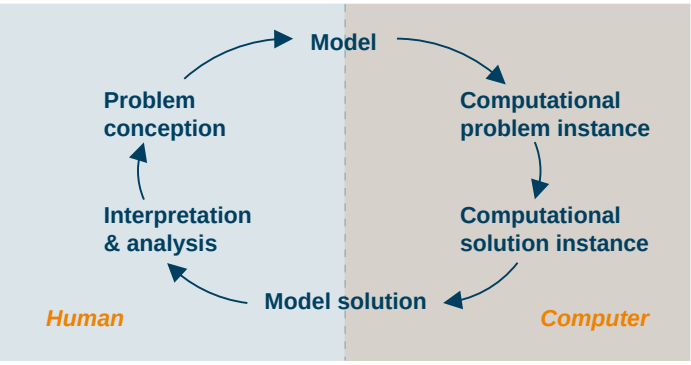
Xpress Team, FICO
http://www.fico.com/xpress

Contents

1 Modeling platforms

Notes

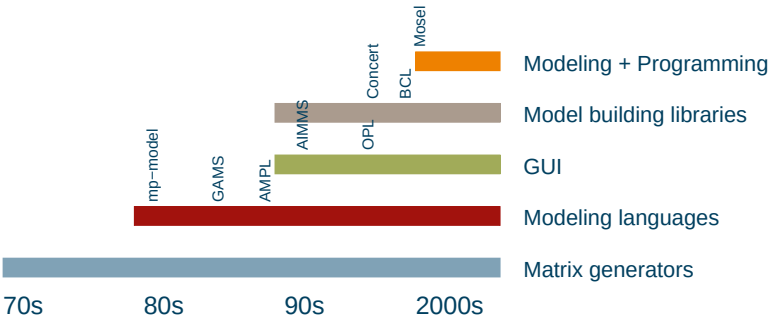
Model development cycle



Why use modeling software?

- Developing a working model is the difficult bit
- Important to have software that helps
  - speed to market
  - verify correctness
  - maintenance & modification
  - algorithmic considerations
  - execution speed

Modeling platforms



	Modeling language	Modeling library	Matrix based
Verify correctness	easy	quite easy	very hard
Maintenance	easy	harder	difficult
Data handling	high level	native/some intrinsic	native language
Building algorithms	language dependent	easy	quite easy
Model execution speed	possibly slower	faster	fastest
Speed to market	fast	slow	slowest

Xpress modeling interfaces

- Mosel
  - formulate model and develop optimization methods using Mosel language / environment
- BCL
  - build up model in your application code using object-oriented model builder library
- Optimizer
  - read in matrix files
  - input entire matrix from program arrays

Mosel

- A modeling and solving environment
  - integration of modeling and solving
  - programming facilities
  - open, modular architecture
- Interfaces to external data sources (*e.g.* ODBC, host application) provided
- Language is concise, user friendly, high level
- Best choice for rapid development and deployment

Xpress-BCL

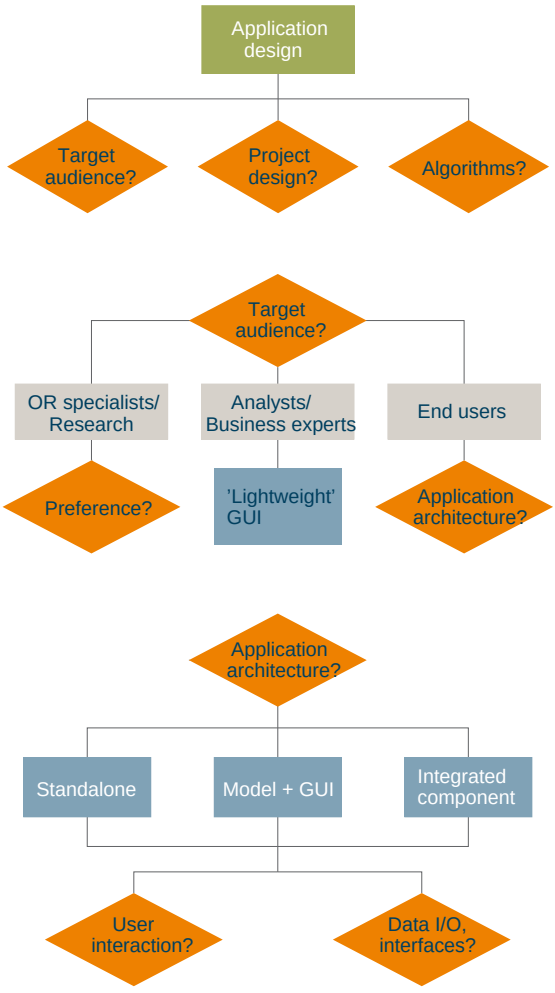
- Model consists of BCL functions within application source code (C, C++, Java, C# or VB)
- Develop with standard C/C++/Java/C#/VB tools
- Provide your own data interfacing
- Lower level, object oriented approach
- Enjoy benefits of structured modeling within your application source code

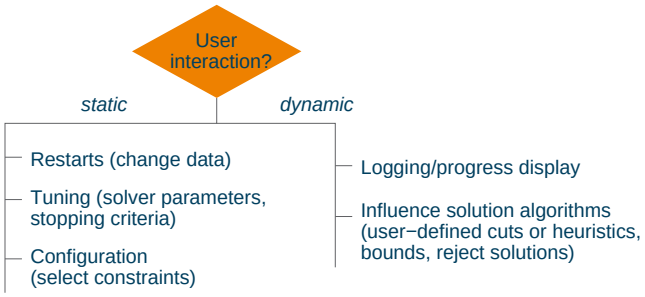
Xpress-Optimizer

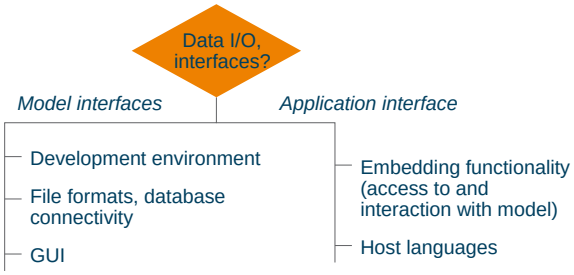
- Model is set of arrays within application source code (C, Java, C#, or VB)
- May also input problems from a matrix file
- Develop with standard C/C#/Java/VB tools
- Provide your own data interfacing
- Very low level, no problem structure
- Most efficient but lose easy model development and maintenance

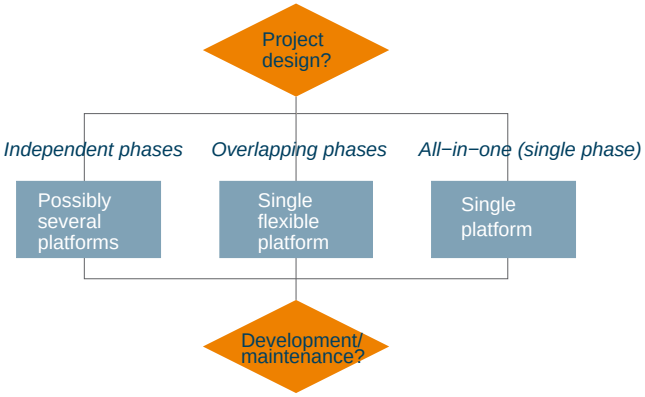
2 Application design

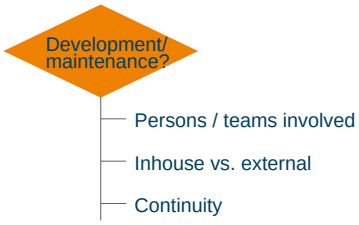
Notes

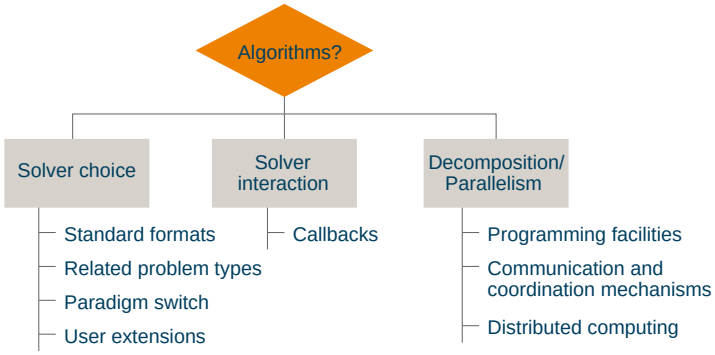












Notes

- A high-level modeling language combined with standard functionality of programming languages
  - implementation of models and solution algorithms in a *single environment*
- Open, modular architecture
  - *extensions to the language* without any need for modifications to the core system
- Compiled language
  - *platform-independent* compiled models for distribution to protect intellectual property

...and also

- *Mosel modules*
  - solvers: mmxprs, mmquad, mmxslp, mmnl, kalis
  - data handling: mmetc, mmodbc, mmoci
  - model handling, utilities: mmjobs, mmsystem
  - graphics: mmive, mmxad
- *IVE*: visual development environment (Windows)
- *Library interfaces* for embedding models into applications (C, Java, C#, VB)
- *Tools*: debugger, profiler, model conversion, preprocessor

Example: Portfolio optimization Problem description

- An investor wishes to invest a certain amount of money into a selection of shares.
- Constraints:
  1. Invest at most 30% of the capital into any share.
  2. Invest at least half of the capital in North-American shares.
  3. Invest at most a third in high-risk shares.
- Objective: obtain the highest expected return on investment

Example: Portfolio optimization Mathematical model

$$\text{maximize} \sum_{s \in SHARES} RET_s \cdot frac_s$$

$$\sum_{s \in RISK} frac_s \leq 1 / 3$$

$$\sum_{s \in NA} frac_s \geq 0.5$$

$$\sum_{s \in SHARES} frac_s = 1$$

$$\forall s \in SHARES : 0 \leq frac_s \leq 0.3$$

Example: Portfolio optimization Mosel model

```
model "Portfolio optimization with LP"
 uses "mmxprs"                      ! Use Xpress-Optimizer

 declarations
  SHARES = 1..10                    ! Set of shares
  RISK = {2,3,4,9,10}               ! Set of high-risk values among shares
  NA = {1,2,3,4}                    ! Set of shares issued in N.-America
  RET: array(SHARES) of real        ! Estimated return in investment

  frac: array(SHARES) of mpvar      ! Fraction of capital used per share
 end-declarations

 RET:: [5,17,26,12,8,9,7,6,31,21]

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 sum(s in RISK) frac(s) <= 1/3

! Minimum amount of North-American values
 sum(s in NA) frac(s) >= 0.5

! Spend all the capital
 sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= 0.3

! Solve the problem
 maximize(Return)

! Solution printing
 writeln("Total return: ", getobjval)
 forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")

end-model
```

Example: Portfolio optimization Logical Conditions

1. Binary variables

```
declarations
 frac: array(SHARES) of mpvar      ! Fraction of capital used per share
 buy: array(SHARES) of mpvar       ! 1 if asset is in portfolio, 0 otherwise
end-declarations

! Limit the total number of assets
 sum(s in SHARES) buy(s) <= MAXNUM

 forall(s in SHARES) do
  buy(s) is_binary                  ! Turn variables into binaries
  frac(s) <= buy(s)                 ! Linking the variables
 end-do
```

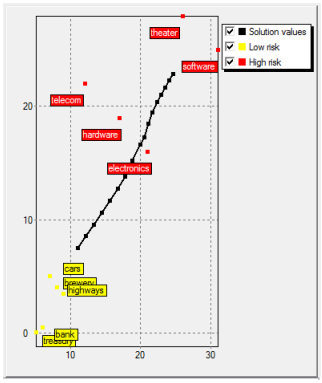
2. Semi-continuous variables

```
 declarations
  frac: array(SHARES) of mpvar      ! Fraction of capital used per share
 end-declarations

 ! Upper and lower bounds on the investment per share
 forall(s in SHARES) do
  frac(s) <= MAXVAL
  frac(s) is_semcont MINVAL
 end-do
```

Example: Portfolio optimization Extended problem

- We wish to
  - run the model with different limits on the portion of high-risk shares,
  - represent the results as a graph, plotting the resulting total return against the deviation as a measure of risk.
- Algorithm: for every parameter value
  - re-define the constraint limiting the percentage of high-risk values,
  - solve the resulting problem,
  - if the problem is feasible: store the solution values.

```
! Solve the problem for different limits on high-risk shares
 ct:=0
 forall(r in 0..20) do
  ! Limit the percentage of high-risk values
   Risk:= sum(s in RISK) frac(s) <= r/20

   maximize(Return)                  ! Solve the problem

   if (getprobstat = XPRS_OPT) then  ! Save the optimal solution value
    ct+=1
    SOLRET(ct):= getobjval
    SOLDEV(ct):= getsol(sum(s in SHARES) DEV(s)*frac(s))
   else
    writeln("No solution for high-risk values <= ", 100*r/20, "%")
   end-if
 end-do

! Drawing a graph to represent results (`plot1') and data (`plot2' & `plot3')
 declarations
  plot1, plot2, plot3: integer
 end-declarations

 plot1 := IVEaddplot("Solution values", IVE_BLACK)
 plot2 := IVEaddplot("Low risk", IVE_YELLOW)
 plot3 := IVEaddplot("High risk", IVE_RED)

 forall(r in 1..ct) IVEdrawpoint(plot1, SOLRET(r), SOLDEV(r));

 forall(r in 2..ct)
  IVEdrawline(plot1, SOLRET(r-1), SOLDEV(r-1), SOLRET(r), SOLDEV(r))

 forall (s in SHARES - RISK) do
  IVEdrawpoint(plot2, RET(s), DEV(s))
  IVEdrawlabel(plot2, RET(s)+3.4, 1.3*(DEV(s)-1), s)
 end-do

 forall (s in RISK) do
  IVEdrawpoint(plot3, RET(s), DEV(s))
  IVEdrawlabel(plot3, RET(s)-2.5, DEV(s)-2, s)
 end-do
```



Data handling

- Physical files:
  - text files (Mosel format, *new: binary format*, diskdata; free format, *new: XML*,
  - spreadsheets, databases (ODBC or specific drivers)
- In memory:
  - memory block/address
  - streams; pipes; callbacks (*new: IO callback*)

```
! Data input from spreadsheet
initializations from "mmodbc.excel:" + DATAFILE
  [RET,RISK,NA] as DBDATA
end-initializations

...

! Solution output to spreadsheet
declarations
 Solfrac: array(SHARES) of real      ! Solution values
end-declarations

forall(s in SHARES) Solfrac(s):= getsol(frac(s))*100

initializations to "mmodbc.excel:" + DATAFILE
  Solfrac as "grow;"+DBSOL
end-initializations
```

Data ranges used by "folioexcel.mos":

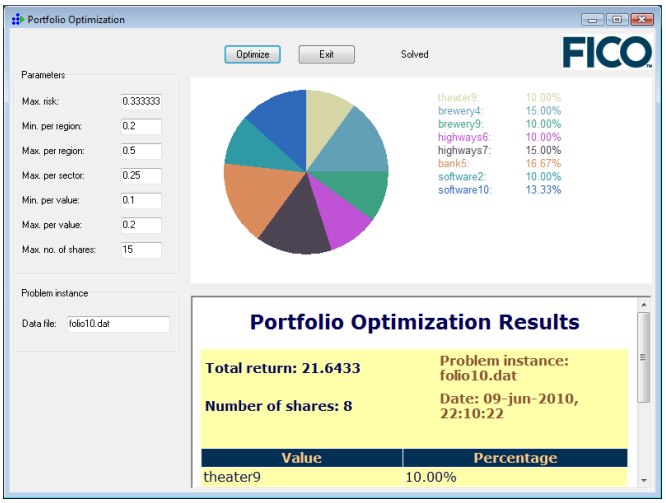
Range "foliodata":

SHARE	RET	RISK	NA
treasury	5		1
hardware	17	1	1
theater	26	1	1
telecom	12	1	1
brewery	8		
highways	9		
cars	7		
bank	6		
software	31	1	
electronics	21	1	

Range "folioresult":

SHARE	SOL

XAD application



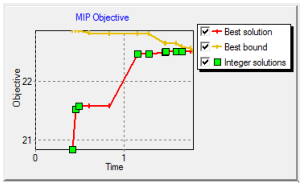
Advanced solving tasks

- Infeasibility handling
  - definition of slack variables
  - IIS (irreducible infeasible sets)
  - infeasibility repair meachanism
- Solution enumeration
  - obtain the N best solutions

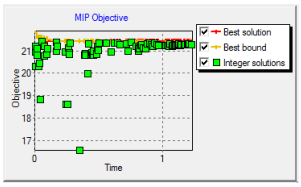
Solution enumeration

```
! Set the max. number of solutions to store (default: 10)
 setparam("XPRS_enummaxsol", 25)

! Solve the problem, enabling the solution enumerator
 maximize(XPRS_ENUM, Return)

! Print out all solutions saved by the enumerator
 forall(i in 1..getparam("XPRS_enumsols")) do
  selectsol(i)                    ! Select a solution from the pool
  writeln("Solution ", i)
  print_sol
 end-do

! Solution printing
 procedure print_sol
  writeln("Total return: ", getobjval)
  forall(s in SHARES | getsol(frac(s))>0)
   writeln(s, ": ", getsol(frac(s))*100, "% (", getsol(buy(s)), ")")
 end-procedure
```

Standard MIP search:



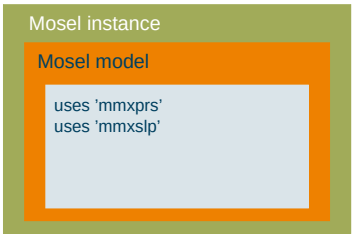
Solution enumerator:



Schemes of decomposition and concurrent solving

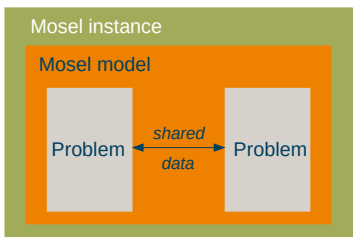
The "multis":

– multi-solver



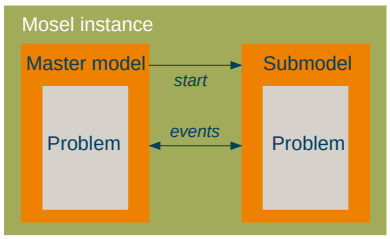
The "multis":

– multi-solver
– multi-problem



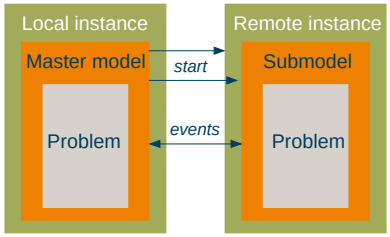
The "multis":

– multi-solver
– multi-problem
– multi-model



The "multis":

– multi-solver
– multi-problem
– multi-model
– multi-node



- Simple parallel runs
  - different data instances
  - different algorithm configurations
- Decomposition
  - Benders
  - Dantzig-Wolffe
- Column generation
  - loop over top node
  - branch-and-price
- Cut generation
  - (cut-and-branch, branch-and-cut)
  - adding constraints

4 Mosel: Selected new features

Notes

4.1 Distributed model execution

- *mmjobs*: facilities for model management, synchronization of concurrent models based on event queues, shared memory IO driver.
- *New:* extending capacities for handling multiple models to distributed computing using several *Mosel instances* (running locally or on remote nodes connected through a network)

- **Mosel instance management:** connecting and disconnecting Mosel instances, access to remote files, handling of host aliases (new type: `Mosel`)
- **Remote connection IO drivers:** two drivers (`xsrv` and `rcmd`) for creating remote Mosel instances.
- **Remote file acces IO drivers:** access to physical files or streams on remote Mosel instances (`rmt`), usable wherever Mosel expects a (generalized) filename, in particular in `initializations` blocks.

- Remote machine must run a server
  - Default (as specified by value of control `conntmpl`): Mosel server *xprmsrv* (started as separate program, available for all platforms supported by Xpress), connect with driver `xsrv`

    ```
    connect(mosInst, "ABCD123")
    ! Same as:   connect(mosInst, "xsrv:ABCD123")
    ```

  - Alternative: other servers, connect with driver `rcmd`, *e.g.* with *rhs*, (NB: Mosel command line option `-r` is required for remote runs):

    ```
    connect(mosInst, "rcmd:rsh ABCD123 mosel -r")
    ```

- The Mosel server can be configured.
  - Use this command to display the available options:

    ```
    xprmsrv -h
    ```

    Configuration options include verbosity settings, choice of the TCP port, and the definition of a log file.
  - Alternatively, use a *configuration file* for more flexible configuration and to define multiple *environments*

    ```
    xprmsrv myconfig.conf
    ```

Configuration file

- Contents of `myconfig.conf`:

```
# Global setting of a log file
LOGFILE=/tmp/logfile.txt

# Add a password to the default environment 'xpress'
[xpress]
PASS=hardone

# Define new environment using a different Xpress version
[xptest]
XPRESSDIR=/opt/xpressmp/testing
XPRESS=/opt/xpressmp/lic
MOSEL_CWD=$XPRESSDIR/workdir
```

- Usage:

```
r1:= connect(inst1, "xsrv:localhost/xpress/hardone")
r2:= connect(inst2, "xrsv:mypcname/xptest")
```

Local instances

- Remote machine may be identical with the current node (new instance started on the same machine in a separate process)

```
connect(mosInst, "")
! Same as:   connect(mosInst, "rcmd:mosel -r")

connect(mosInst, "localhost")
! Same as:   connect(mosInst, "xsrv:localhost")
```

Executing a submodel

```
model "Run model rtparams"
 uses "mmjobs"

 declarations
  modPar: Model
 end-declarations
                          ! Compile the model file
 if compile("rtparams.mos")<>0 then exit(1); end-if
                          ! Load the bim file
 load(modPar, "rtparams.bim")
                          ! Start model execution + parameter settings
 run(modPar, "PARAM1=" + 3.4 + ",PARAM3='a string'" + ",PARAM4=" + true)
 wait                     ! Wait for model termination
 dropnextevent            ! Ignore termination event message

end-model
```

Executing a submodel remotely

```
model "Run model rtparams remotely"
 uses "mmjobs"

 declarations
  modPar: Model
  mosInst: Mosel
 end-declarations
                          ! Compile the model file
 if compile("rtparams.mos")<>0 then exit(1); end-if

 NODENAME:= ""            ! "" for current node, or name, or IP address
                          ! Open connection to a remote node
 if connect(mosInst, NODENAME)<>0 then exit(2); end-if
                          ! Load the bim file
 load(mosInst, modPar, "rmt:rtparams.bim")
                          ! Start model execution + parameter settings
 run(modPar, "PARAM1=" + 3.4 + ",PARAM3='a string'" + ",PARAM4=" + true)
 wait                     ! Wait for model termination
 dropnextevent            ! Ignore termination event message
end-model
```

```
model "Compile and run model rtparams remotely"
 uses "mmjobs"

 declarations
  modPar: Model
  mosInst: Mosel
 end-declarations

 NODENAME:= ""              ! "" for current node, or name, or IP address
                            ! Open connection to a remote node
 if connect(mosInst, NODENAME)<>0 then exit(2); end-if
                            ! Compile the model file remotely
 if compile(mosInst, "", "rmt:rtparams.mos", "rtparams.bim")<>0 then
  exit(1); end-if           ! Load the bim file
 load(mosInst, modPar, "rtparams.bim")
                            ! Start model execution + parameter settings
 run(modPar, "PARAM1=" + 3.4 + ",PARAM3='a string'" + ",PARAM4=" + true)
 wait                       ! Wait for model termination
 dropnextevent              ! Ignore termination event message
end-model
```

New and overloaded subroutines

- Instance connection/disconnection

```
r:= connect(myInst, "")
disconnect(myInst)
```

- Remote compilation & loading

```
r:= compile(myInst, "", "filename.mos", "filename.bim")
load(myInst, myModel, "filename.bim")
```

- Redirecting Mosel streams

```
setdefstream(myInst, F_OUTPUT, "rmt:instoutput.txt")
```

Some utilities

- System information

```
compName:= getsysinfo(SYS_NODE);    allinfo:=getsysinfo(myInst)
currNode:= getparam("NODENUMBER");  parent:= getparam("PARENTNUMBER")
modelID:= getparam("JOBID");        instID:= getid(myInst)
```

- Instance status information

```
if getstatus(myInst)<>0 then
  writeln("Instance is not connected")
end-if
```

- Aliases

```
sethostalias("localhost2","localhost")
r:= connect(myInst, "localhost2")
sysName:= gethostalias("localhost2");     getaliases(allAliases)
clearaliases
```

Distributed model execution

- **Documentation:** 'Mosel Language Reference manual', Chapter 7 *mmjobs*
- **Examples:** see newest version of the whitepaper 'Multiple models and parallel solving with Mosel', Section 2.8 *Working with remote Mosel instances*
- Another introductory example in 'Guide for evaluators 2', Section 6 *Working in a distributed architecture*

4.2 IO callbacks

- In-memory communication so far: fixed data structure sizes
- *New:* alternative communication mechanism working with flows enables *dynamic sizing of data structures* on the application level
  - particularly useful for solution output where effective data sizes are not known a priori
  - available in C, Java, .NET

- Pass the address of the function (C) or class (Java) implementing the callback to Mosel via model parameters
- `initializations to`: use the Mosel post-processing library functions to retrieve data from Mosel into the application
- `initializations from`: new set of functions to send data to Mosel, using the same format as the default text file format

IO callbacks (C)

```
mydata: [ ("ind1" 3) [5 1.2] ("ind2" 7) [4 6.5] ]

XPRMcb_sendctrl(cb, XPRM_CBC_OPENLST, 0);     ! [
XPRMcb_sendctrl(cb, XPRM_CBC_OPENNDX, 0);     !   (
XPRMcb_sendstring(cb, "ind1", 0);             !     "ind1"
XPRMcb_sendint(cb, 3, 0);                     !     3
XPRMcb_sendctrl(cb, XPRM_CBC_CLOSENDX, 0);    !   )
XPRMcb_sendctrl(cb, XPRM_CBC_OPENLST, 0);     !   [
XPRMcb_sendint(cb, 5, 0);                     !     5
XPRMcb_sendreal(cb, 1.2, 0);                  !     1.2
XPRMcb_sendctrl(cb, XPRM_CBC_CLOSELST, 0);    !   ]
XPRMcb_sendctrl(cb, XPRM_CBC_OPENNDX, 0);     !   (
XPRMcb_sendstring(cb, "ind2", 0);             !     "ind2"
XPRMcb_sendint(cb, 7, 0);                     !      7
XPRMcb_sendctrl(cb, XPRM_CBC_CLOSENDX, 0);    !   )
XPRMcb_sendctrl(cb, XPRM_CBC_OPENLST, 0);     !   [
XPRMcb_sendint(cb, 4, 0);                     !     4
XPRMcb_sendreal(cb, 6.5, 0);                  !     6.5
XPRMcb_sendctrl(cb, XPRM_CBC_CLOSELST, 0);    !   ]
XPRMcb_sendctrl(cb, XPRM_CBC_CLOSELST, 0);    ! ]
```

IO callbacks (Java)

```
mydata: [ ("ind1" 3) [5 1.2] ("ind2" 7) [4 6.5] ]

ictx.sendControl(ictx.CONTROL_OPENLST);     ! [
ictx.sendControl(ictx.CONTROL_OPENNDX);     !   (
ictx.send("ind1");                          !     "ind1"
ictx.send(3);                               !     3
ictx.sendControl(ictx.CONTROL_CLOSENDX);    !   )
ictx.sendControl(ictx.CONTROL_OPENLST);     !   [
ictx.send(5);                               !     5
ictx.send(1.2);                             !     1.2
ictx.sendControl(ictx.CONTROL_CLOSELST);    !   ]
ictx.sendControl(ictx.CONTROL_OPENNDX);     !   (
ictx.send("ind2");                          !     "ind2"
ictx.send(7);                               !      7
ictx.sendControl(ictx.CONTROL_CLOSENDX);    !   )
ictx.sendControl(ictx.CONTROL_OPENLST);     !   [
ictx.send(4);                               !     4
ictx.send(6.5);                             !     6.5
ictx.sendControl(ictx.CONTROL_CLOSELST);    !   ]
ictx.sendControl(ictx.CONTROL_CLOSELST);    ! ]
```

IO callbacks

- **Documentation:** 'Mosel Library Reference manual', Section 1.5.2.2 *cb driver – Handling of initializations blocks*
- **Examples:** see newest version of the 'Mosel User Guide', Sections 13.4.3 *Dynamic data* (C), 14.1.6.3 *Dynamic data* (Java)

4.3 XML interface

- The module *smew* provides an XML interface for the Mosel language.
- *smew* relies on two external libraries without which the module will not work:
  - *scew* ('simple C expat wrapper') — handling of the XML tree
  - *expat* — the parser

Structure of an XML document

```
<?xml ... ?>                                    Preamble

<root>
  <parent>
    <element attrname="attrvalue">
       contents
       <child>
         <leaf>leafcontents</leaf>
       </child>
       <child>2nd child contents</child>
    </element>
    <emptyelement attrname="attrvalue" />
  </parent>
</root>
```

smew functionality

- *New types:*
  - `xmldoc` represents an XML document
  - `xmleltref` is a reference to a node/element in the document.
    Several `xmleltref` may reference the same element and the module does not check consistency: if an element is removed, it is up to the user to make sure none of its references will be used afterwards

- *Subroutines:*
  - **File access:** `load, save`
  - **Document structure:** `getroot, setroot, isvalid, getpreamble, setpreamble, getchildren, getparent, add, remove`
  - **Handling elements:** `getname, setname, getcontent, get[int|real|bool|str]content, setcontent, getattr, get[int|real|bool|str]attr, setattr, delattr, getallattr`

Example: Portfolio optimization XML data format

```
declarations
 SHARES: set of string          ! Set of shares
 RISK: set of string            ! Set of high-risk values among shares
 NA: set of string              ! Set of shares issued in N.-America
 RET: array(SHARES) of real     ! Estimated return in investment

 AllData: xmldoc                ! XML document
 ShareList: list of xmleltref   ! List of XML elements
end-declarations

! Reading data from an XML file
load(AllData, "folio.xml")
getchildren(getroot(AllData), ShareList, "share")

RISK:= union(l in ShareList | getattr(l,"risk")="high")
 {getstrattr(l,"name")}
NA:= union(l in ShareList | getattr(l,"region")="NA")
 {getstrattr(l,"name")}
forall(l in ShareList) RET(getstrattr(l,"name")):= getintattr(l, "ret")
```

- Data file `folio.xml`:

```
<portfolio>
 <share name="treasury" ret="5" dev="0.1" country="Canada"
                        region="NA" risk="low" />
 <share name="hardware" ret="17" dev="19" country="USA"
                        region="NA" risk="high" />
 ...
 <share name="electronics" ret="21" dev="16" country="Japan"
                        region="Asia" risk="high" />
</portfolio>
```

```
declarations
 SHARES: set of string         ! Set of shares
 frac: array(SHARES) of mpvar  ! Fraction of capital used per share

 AllData: xmldoc               ! XML document
 Share,Root,Sol: xmleltref     ! XML elements
end-declarations

! Create solution representation in XML format
Root:= setroot(AllData, "result")
Sol:= add(Root, "solution")
forall(s in SHARES) do
 Share:= add(Sol, "share")
 setattr(Share, "name", s)
 Share.content:= frac(s).sol
end-do

save(AllData, "result.xml")    ! Save solution to XML format file
save(AllData, "")              ! Display XML format solution on screen
```

- Generated output file `result.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<result>
  <solution>
    <share name="treasury">0.3</share>
    <share name="hardware">0</share>
    ...
    <share name="electronics">0</share>
  </solution>
</result>
```

smew distribution

- Available for download from the Mosel open source webpage
- Archive contains
  - module source file: `smew.c`
  - module library file: `smew.dso` (copy into subdirectory `dso`)
  - library files: `*expat.*` and `*scew.*` (copy into subdirectory `bin` [Windows] or `lib` [Unix])
  - documentation: `smew.txt`
  - examples: `folioxml.mos`, `folioxmlqp.mos`, `booksearch.mos`, `xmltest.mos`

5 Application examples

Notes

5.1 Alternative interfaces: Portfolio rebalancing

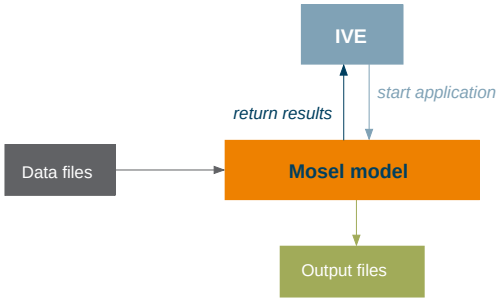
Portfolio rebalancing: Problem description

- Modify the composition of an investment portfolio as to achieve or approach a specified investment profile.

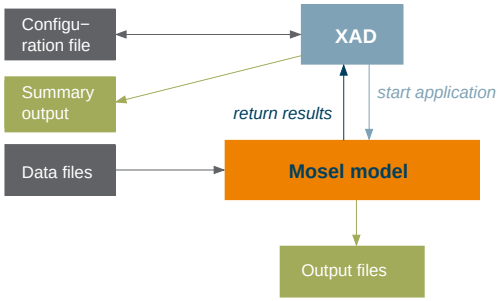
Application architecture

- Single, configurable model file
- Different interfaces for model execution
  - stand-alone mode (command line or through Xpress-IVE) for *development*
  - graphical interface (written with XAD) for single model runs and *simulation*
  - Java application for running *batches of model instances*

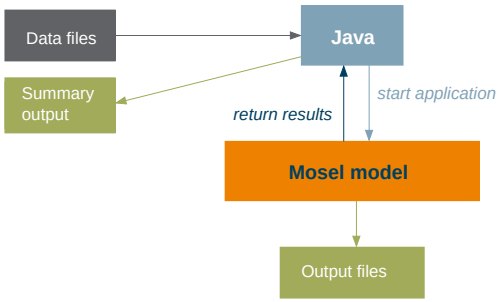
Optimization application in Mosel Standalone



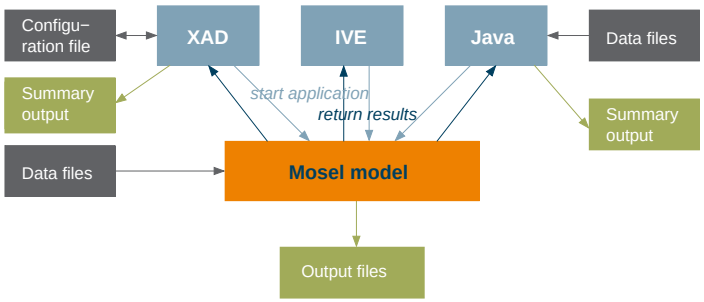
Optimization application in Mosel XAD GUI



Optimization application in Mosel Embedded into host application



Optimization application in Mosel Alternative interfaces



Input

- Stand-alone and XAD: data input from text files directly into Mosel
  - uses a filter module to accomodate different number formats
- Java: data read and stored by host application; communication with model instances through memory

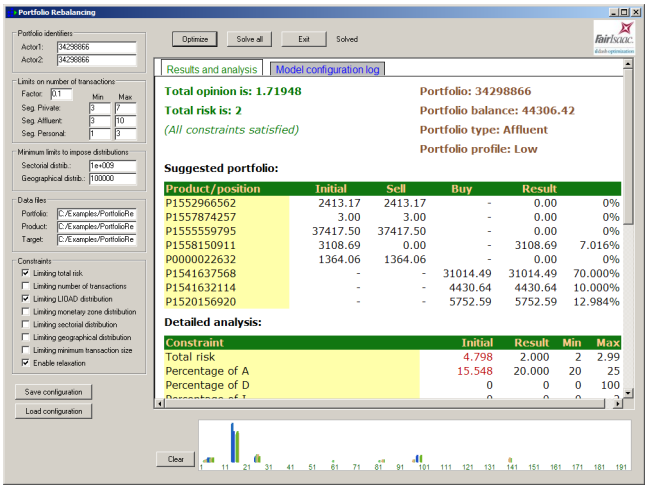
Output

- Textual output log on screen or to file
- Optionally detailed HTML output
- Java: summary statistics of multiple runs
- XAD:
  - summary statistics in the case of multiple runs
  - optional output to Excel

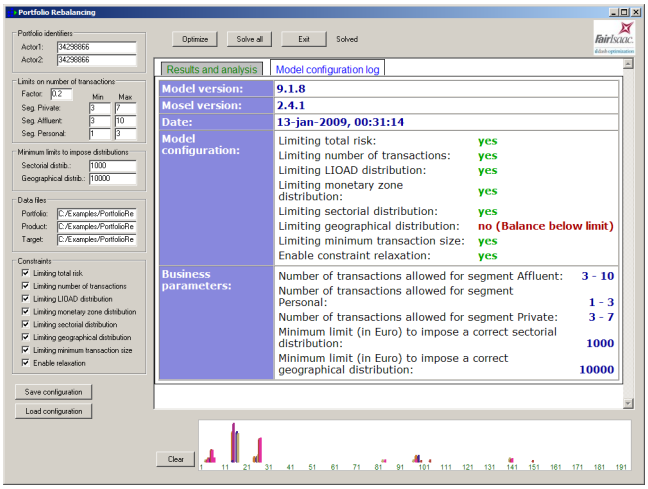
XAD interface

- Graphical user interface (Windows)
- Configuration of model runs
  - data files
  - parameter settings
  - selection of constraints
- Choice of solving mode:
  - repeated runs for a single model (simulation)
  - solve all instances from customer file (evaluation of parameter settings)
- Graphical comparison of results

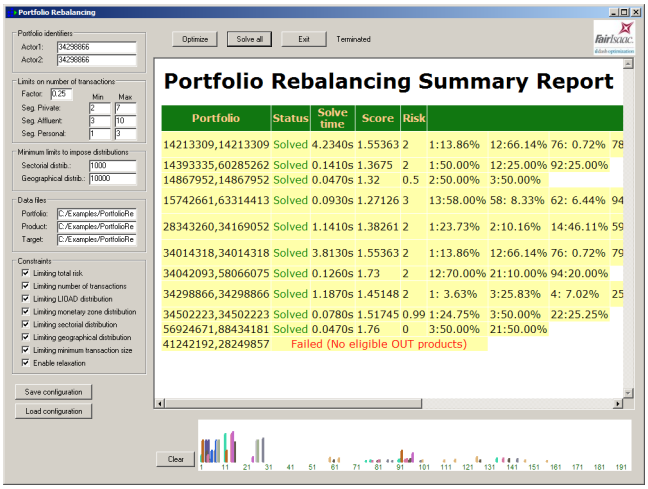
XAD interface: Detailed results



XAD interface: Parameter and version log



XAD interface: Multiple run summary



Some highlights

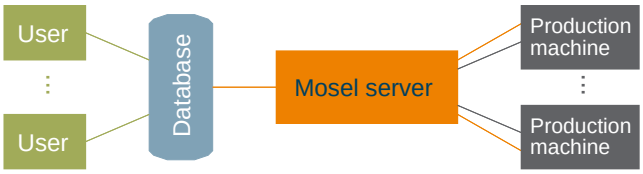
- *Model:*
  - easy maintenance through single model
  - deployment as BIM file: no changes to model by end-user
  - language extensions according to specific needs
- *Interfaces:*
  - several run modes adapted to different types of usages
  - efficient data exchange with host application through memory
  - parallel model runs (Java) or repeated sequential runs (XAD)

5.2 Distributed Mosel: Client-server

Distributed Mosel: Problem description

- Multi-user optimization application processing a large number of optimization model instances
- Idea: replace the preselected, static assignment of optimization runs by a Mosel server that controls the job queues

Distributed Mosel: Client-server architecture



Distributed Mosel: Highlights

- Use Mosel lists for representation of dynamic queueing system
- Mosel master ('server') model communicates with database and handles remote submodels

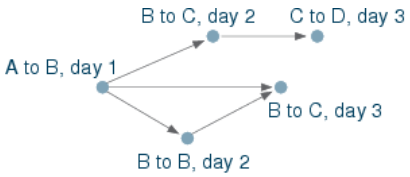
5.3 Visualization: Aircraft routing

Aircraft routing: Problem description

- For given sets of flights and aircraft, determine which aircraft services a flight.
- Aircraft are not identical
  - they cannot all service every flight
  - a specific maintenance site must be used per plane
  - some scheduled long maintenance breaks
- Starting condition: each aircraft has a starting position and a specific amount of accumulated flight minutes

Aircraft routing: Representation

- Temporal (activity on node) network:
  - a flight corresponds to a node
  - 'cost' of node: flight minutes ($\neq$ elapsed time)
  - successor nodes: flights starting from a destination within a given time window after arrival of predecessor
  - maintenance: represented by a node
  - aircraft: commodity traveling through the network



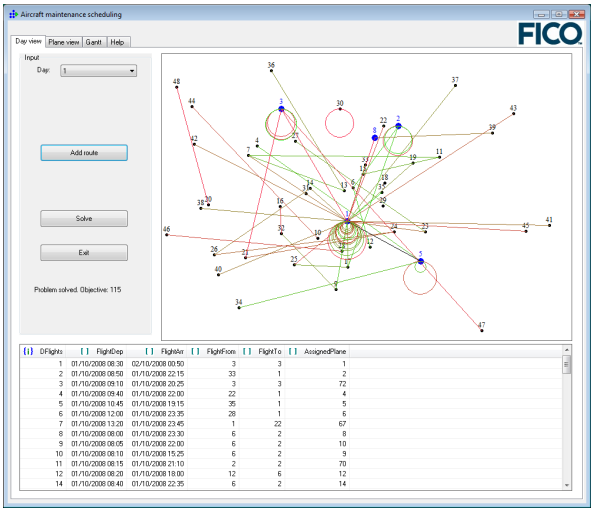
Aircraft routing: Decomposition

- Different views are possible:
  - per time unit (*e.g.*, day)
  - per commodity (aircraft)
- Idea: generate set of *feasible routes per aircraft* by solving optimization subproblems maximizing the flight minutes up to each maintenance stop
  - iteratively force usage of 'less preferred' flights
  - may keep suboptimal solutions

Aircraft routing: Application architecture

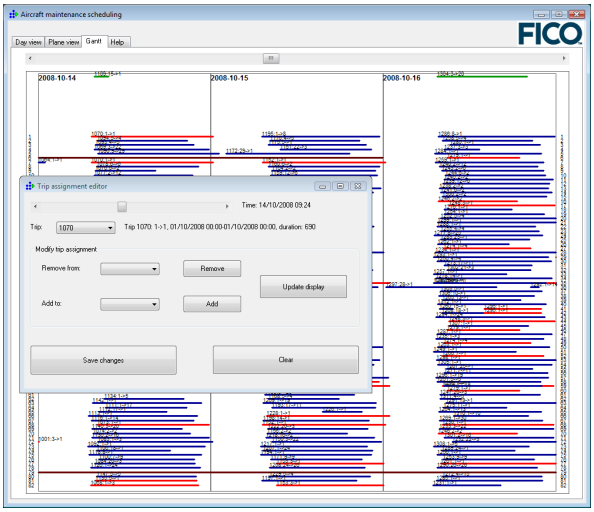
- Master problem: route selection
- Subproblems: route generation (one instance per plane)
  - parallel, possibly remote, execution of submodels
- User interface (optional): XAD GUI

Aircraft routing: Application GUI



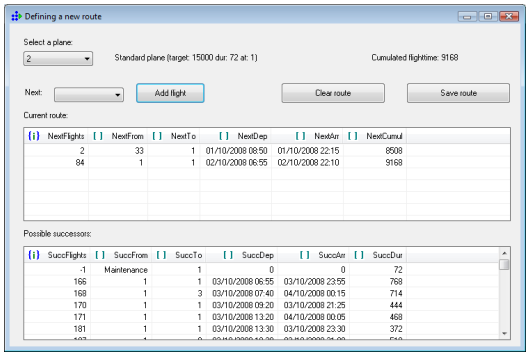
Aircraft routing: Visualization

- Visualization of input data helps with understanding and analysis of the problem
- Representation of intermediate results during development (IVE) or as progress report to users (XAD)



Aircraft routing: User interaction

- Manual construction of routes
- Editing generated plans



Summary

Notes

- Have seen:
  - design choices for optimization applications
- Xpress-Mosel:
  - recent developments make possible implementation of complex algorithms and a high degree of user interaction
  - unique features for handling large-scale problems:
    support of decomposition, concurrent solving, distributed computing, and also 64bit coefficient indexing

Where to get more information

- Xpress website:
  - `http://www.fico.com/xpress`
- Xpress resources (documentation, whitepapers)
  - `http://optimization.fico.com`
- Searchable on-line examples database:
  - `http://examples.xpress.fico.com`
- Trial download:
  - `http://decisions.fico.com/downloadTrial.html`