FICO

Mosel: An Overview

日本語訳　　版

FICO™ Xpress Optimization Suite Whitepaper

Last update 24 June, 2008

Mosel: An Overview

Y. Colombani and S. Heipcke
Xpress Team, FICO, Leam House, Leamington Spa CV32 5YN, UK
http://www.fico.com/xpress
May 2002, last rev. June 2008

概要
Msel An Overview では、基本的なMosel言語について解説しています。
このソフトウエアは、モデリングおよびソリューションをスタンダードなマトリックスを
ベースとしたソルバーにレポートするインターフェースとして使用するためのソフトウエアです。
さらに、Moselを使用し、最も複雑なソリューションアルゴリズムを実装する方法も
解説します。
Moselライブラリを使用することにより、Mosel言語で生成したモデルはC,C++, Java, C#, Visual Basic.
などのプログラミング言語で実装されたアプリケーション・プログラムからアクセスおよび統合することも可能です。
モデリングおよび問題を解くためのmodular architecture of the Mosel 環境は簡単に拡張できるように設計
されています。
特定の問題やソルバータイプによる制限はありません。
このMosel An Overviewではユーザーの皆様がインスタンスとして、他のソルバーにアクセス
する場合、新機能として既存のMosel言語をどのように使用することが可能なのかということも解説します。

目次

1 はじめに
1.1 ソルバー・モジュール
1.2 その他のモジュール
1.3 ユーザーモジュール
1.4 ツール
1.5 I/O ドライバー
1.6 本書の内容
2 Overview
2.1 Mosel 言語を使った簡単な例題
2.2 例題の拡張
3 mosel言語
3.1 タイプおよびデータ構造
3.2 データ/データファイルアクセスの初期化
3.3 言語拡張
3.31 選択する
3.4 ループ
3.5 例題を設定し、実行する
3.6 サブルーチン
4 Mosel ライブラリ
5 モジュール
5.1 利用可能なモジュール
5.2 mmxprs:MIP 変数をヒューリスティックに固定する
5.3 mmquad: QP問題を定義し解く
6 パッケージ
6.1 新しいサブルーチンで定義する
7 ユーザーのモジュールでモデルを生成する
7.1 新しいサブルーチンで定義する
7.2 新しいタイプを生成する
7.2.1モジュールのコンテキスト
7.2.2 生成と削除の種類
7.2.3 文字列の転送方法の種類
8 おわりに

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

1 はじめに

Moselはモデリングおよび問題を解くための環境であり、ライブラリ形式または
スタンドアロン・プログラムとして提供されています。
Moselはモデリングとプログラミング言語、双方の概念の長所を生かして設計されている
言語です。
問題がモデリング言語とアルゴリズム演算を使い記述されるAMPLのような従来の
モデリング環境でスクリプト言語で記述されているのとは対照的に、Moselでは、
モデリングステートメント（意思決定変数を宣言する、または制約式を表現する）
と問題を実際に解く手続き（最適化コマンドをコールする）が分離されていません。
インタレースなモデリングと問題を解くためのスタートメントの相乗効果の結果、
一回で、複雑なソリューション・アルゴリズムをプログラムすることが可能です。

1.1　ソルバー・モジュール

プログラムの各項目ごとに、自身の特定の変数タイプや制約式が加えられると
1種類のソルバーでは、いかなるケースにおいても効率的に作業を進めることができません。
このような点を考慮し、Moselではデフォルトによる各ソルバーの統合は行いませんが、モジュールとして提供される外部のソルバーに、動的なインターフェースを行います。
どのソルバー・モジュールもプロシージャーのセットおよび直接、拡張できるボキャブラリー関数と言語機能を備えています。
この基本設計がMoselと、使用されているソルバー間の効率的なリンクを保証しています。
同様な接続として他のシステムを使うMPL(Maximal,2001)も提供されていますが、この接続形式はモジュールの概念に基づくものです。
Moselはあらゆるソルバータイプにも制限されることなく、どのソルバーも特定のユーザーレベルのによる制限はありません。
例えば、手続きsetcoeff(ctr,var,coeff)を変数vaとしてctrにその制約式のマトリックス係数をセットし定義する場合、このような手続きは他のソルバータイプ向けではありません。
モジュールmmquadはXpress-Optimizerのような適切なソルバーで処理することにより、二次式としてこの言語のsystaxを拡張する場合も同様です。
主なモジュール・アーキテクチャの長所は、新しいソリューション技術にアクセスするために
Moselを修正する必要がないということです。

現在、Moselで利用可能なソルバー・モジュールは下記の通りです。
mmyprs:　線形計画問題、混合整数計画問題および二次計画問題を解くために、Xpress-Optimizer
　　にアクセスする
mmxslp: 逐次線形問題として、非線形制約式で問題を定義し解く
mmsp:　確率問題を形式化し、解く

1.2　その他のモジュール

ここではMoselのモジュール・アーキテクチャを使かうことでソルバー以外のソフトウエア環境でも開発することが可能なことについて解説いたします。
例えば、一つのMoselモジュール(mmodbc)を持っていればユーザーはスタンダードSQLのコマンドを使いODBCのインターフェースに定義したaccess databasesやspreadsheetも利用することが可能です。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

他のライブラリでは特定のデータベースに(ODBCに加え、mmodbcはMicrosoft Exsel向け特定のインターフェースとして提供しております。)
接続するために必要な関数を定義することも可能です。

他のMoselモジュール(mmjobs)では、複合的モデルを処理するための機能も実装しておりメモリー内で同期やデータ交換を行う機能も備えています。

上記に述べたように、広範的なアルゴリズムを持つMoselを用いて
実装することができます。
このようなアルゴリズムの例題につきましては、Xpress Whiteparper Mulitple models and parallel solving with Moselを参照されたい。
モジュールmmiveを活用し、ユーザーの皆様はグラフィカル環境Xpress-IVEでご自身のグラフィックを生成することができます。
この機能をさらに一歩進め、モジュールmmxad（Xpress-アプリケーション開発；XAD）
を使い、Moselモデルの中から完全なグラフィカルアプリケーションを定義することができます。
その他のモジュールでは汎用的でないインターフェースとして他のアプリケーションへ接続するために必要な関数を定義することで利用することができます。

1.3 ユーザー・モジュール

提供されたモジュールの他に、Moselはユーザーの皆様が追加するいかなるタイプのモジュールでも展開することができます。
Moselとユーザーが追加するモジュール間の通信には、特定のプロトコルとライブラリを使います。
このNative Interfaceはパブリックでき、ユーザーの皆様は自身のモジュールを実装することが可能です。
Cプログラム形式で生成するプログラミング作業の場合、モジュールを使ってMosel言語から変換することも可能です。
モジュールを使い、ユーザーの皆様が行える作業は下記の通りです。
・特定のデータ処理を行うアプリケーション
（複合データ形式の定義、データをメモリにインプット/アウトプットする）
・外部プログラムへアクセスする
・モジュールからソリューションアルゴリズムへのアクセスおよびプログラミング言語を使い
　ヒューリスティックな実装を行う
　（既存コードの再利用が可能）
・重要なプログラミング作業にかかる時間（ヒューリスティックな作業で頻繁に呼びだす必要が
　があるアルゴリズムの並び替え）など、効率的な実装を行う。

1.4 ツール

1組のセットでMoselには、モデルのプロフェッサーおよび、プログラミング時に最も重要となるデバッカ、ステップ実行や、ブレイクポイントの調整など、標準的なデバック作業機能を搭載しており、
Moselモデルを分析するためのプロファイラーも含まれています。
デバック機能やプロファイラーのツールはMoselのコマンド行からアクセスでき、グラフィカル開発環境IVEによりサポートされています。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

1.5 I/Oドライバー

Moselで使われているFileという文字は、通常私たちが使っているFileの意味と同様です。
例えば、
・フィジカルファイル（テキストまたはバイナリ）
・メモリーのブロック
・オペレーティング・システムによって与えられるファイル記述子
・コールバック機能
・データベース
MoselはI/Oドライバーセットは特定のデータソース向けインターフェースとして提供されており、アプリケーションを実行するMoselライブラリとMoselモデル間の情報のやりとりをダイレクトな方法で行うことを可能にします。
そしてMosel Native interface(NI)もユーザーの皆様が自身のドライバーに実装することをサポートします。
I/Oドライバーを用いた作業は、Moselの最新機能を使いますので、
この機能についての詳しい解説は、別冊Xpress Whiteparper[Generalized file handling in Mosel]
を参照されたい。

1.6 本書の内容

本書の冒頭では、Moselのシステムに関する基本的なアーキテクチャの概要およびMosel言語を簡単な例題を用いて解説いたします。
特定のMosel言語のついては、いくつかのプログラムを展開することにより、2章でさらに詳しい解説を行います。
以下の章では、Moselのモジュール概念、および現在ご利用いただけるモジュールについてご紹介します。
Moselで、これまでよりさらに向上したソリューション・アルゴリズムをどのように実装するのかについても解説いたします。
ソフトウエアを統合する目的は、Mosel言語で定義されているプログラミング言語のオブジェクト内からどのようにアクセスを行うかについて理解することがとても重要であるためです。
（このトピックは、次章で解説しています。）
最後の二つの章では、ユーザーの皆様が自身のMoselライブラリをどのように実装することが可能なのかを解説します。
パッケージ形式(Mosel言語で書かれたライブラリ)や最新モジュール（Mosel Native Interfaceが使われているCライブラリ）を用いることも可能です。

2 Overview

2.1 Mosel言語　簡単な例題

下記の二つのProduct（XSは小さな製品を作る数、Xlは大きな製品を作る数）
二つのリソース制約製品計画の例題を用いて、Moelで簡単な混合整数計画問題をどのように、
記述し、解くのかを解説します。
一般構造；全てのMoselプログラムは、modelというキーワードによりスタートし、そしてモデル
名が続き、end-modelのキーワードで終了させます。
すべてのオブジェクトは割り当てを介し、明確に定義されていない限り、declaration sectionで宣言を行います。（i=1　は整数をiとして定義し、これに値1を代入する.）
モデル内の異なる場所にdeclaration sectionが複数存在する場合があります。
ここで取り上げている例題では、2つの変数xsとmpvar形式のxlで定義します。
このモデルは2つの変数と、制約式xsおよび整数値のみを取るxlとして2つの線形不等制約式で定義します。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

解法
maximizeプロジデュアとして、線形式5*xs + 20*xl.を最大化するために、Xpress-Optimizerをコールします。
Moselにdefaultソルバーはないため、プログラムの最初の部分でmmxprsを使い、スタートメントでXpress-Optimizerを指定します。

アウトプットの印刷
最後の3行に2つの変数におけるソリューション値と最適解の値がプリントアウトされます。
sol(xs)を記述する代わりに、ドット表記法でxs.solと記述しても同様です。

改行
１行に複数のスタートメントを配置することも可能ですが、セミコロン（例えば、
xsis_interger;lis_integerのように）で区切ることも可能です。
しかし、特別なline end(ライン終わり)継続を表す文字がないため、複数行に跨る場合、
スタートメントの各行で、+,>,=などの演算子や文字などで終わらせる必要があります。
これは、スタートメントが継続していることを明確にするために記述します。

コメント行
例題にあるように、Moselでは１行のコメント行は手続き！ではじめ、複数のコメント行に跨る場合は、！ではじめ、!で終わらせます。

```
model Chess
uses "mmxprs" ! Use Xpress-Optimizer for solving
declarations
xs,xl: mpvar ! Decision variables
end-declarations
3*xs + 2*xl <= 160 ! Constraint: limit on working hours
xs + 3*xl <= 200 ! Constraint: raw material availability
xs is_integer; xl is_integer ! Integrality constraints
maximize(5*xs + 20*xl) ! Objective: maximize the total profit
writeln("Solution: ", getobjval) ! Print objective function value
writeln("small: ", getsol(xs)) ! Print solution values for variables xs
writeln("large: ", getsol(xl)) ! and xl
```

2.2.例題を拡張する

Moselの表現力を実際に体験して頂くため、下記のモデルは先ほどのモデルを拡張したものです。

制約式に名称を付ける
この例題では、制約式に名前が付けられています。
線形式は使いませんが、単にソルバーをコールする時には、線形式を参照します。
この参照を行うには、2つの不等式で可能です。
例えば、制約式のソリューション情報にアクセスする場合に使用します。
（dual,slack,activity）
データ構造
2番目の例題では、データ構造のセットと配列も導入します。
ここでは配列DescrVおよびインデックスセットAll varsはサイズとコンテンツが未知なため
ダイナミックです。
コンテンツはdeclarationセクションで与えられた割り当てにより定義されます。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
model Chess
uses "mmxprs" ! Use Xpress-Optimizer for solving
declarations
xs,xl: mpvar ! Decision variables
end-declarations
3*xs + 2*xl <= 160 ! Constraint: limit on working hours
xs + 3*xl <= 200 ! Constraint: raw material availability
xs is_integer; xl is_integer ! Integrality constraints
maximize(5*xs + 20*xl) ! Objective: maximize the total profit
writeln("Solution: ", getobjval) ! Print objective function value
writeln("small: ", getsol(xs)) ! Print solution values for variables xs
writeln("large: ", getsol(xl)) ! and xl
end-model
```

３ Mosel言語

3.1 タイプおよびデータ構造

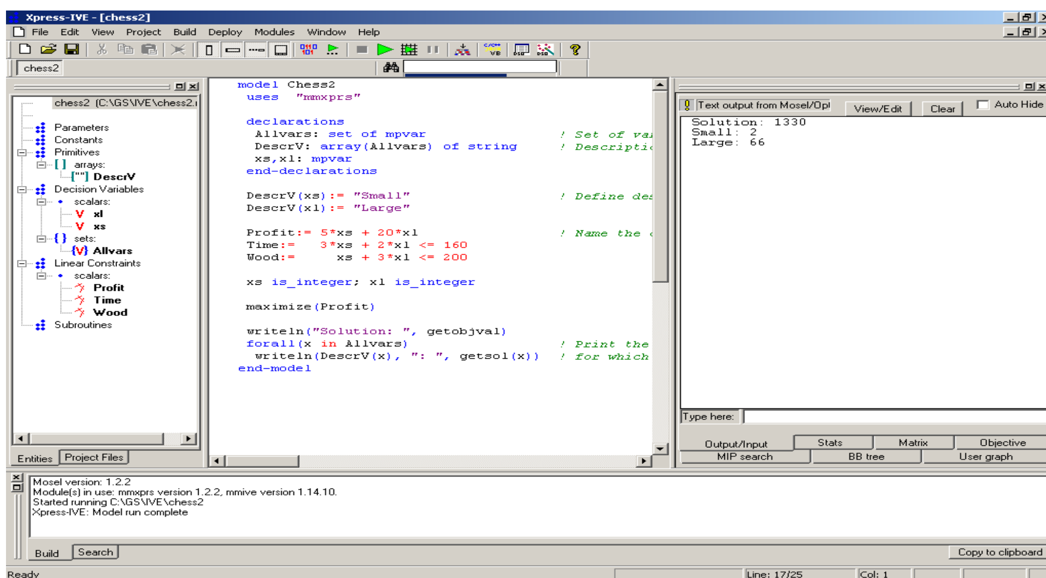
Moselは様々なプログラミング言語の使用を望まれるユーザーの皆様向けにベーシックタイプとして提供しております。例えば、統合、実数（倍精度数浮動小数点）、ブール（実数と負数の記号）、文字列（単一文字やテキスト）などを使ったプログラミングが行えます。

また、MPタイプmpvar（意思決定変数）とlinctr（線形制約式）は特に数理プログラミング向けに提供しており、この形式はMoselのelementaryタイプです。

このelementaryタイプに加え、Moselは構造タイプの設定（特定集合要素）、array(特定タイプのラベルオブジェクト集合)、list(指定された集合要素)およびrecord(他のタイプのオブジェクト集合)を定義します。

前章で解説した通り、Moselにおけるデータ構造(設定および配列)はサイズとコンテンツが未知数の場合、動的です。

下記はconstant set(R1,S1)およびlist s(L1)、静的配列(A1,A2)を定義した結果です。



Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
declarations
R1 = 3..5
S1 = {"red", "green", "blue"}
A1: array(S1) of real
A2: array(R1, -2..1) of mpvar
L1 = [1,2,3,4,5]
end-declarations
```

動的セット(R2,S2),リスト(L2,L3)および配列(A3,A4)は下記のように定義することにより表現されます。

```
declarations
S2: set of string
A3: array(S2,S2) of linctr
A4: array(R2:range) of boolean
L2: list of real
L3: array(set of integer) of list of string
end-declarations
```

この例題に特別なタイプセットのR1とR2を導入します。これはレンジを示すものです
(整数順序集合)
3…5はR1が全ての整数集合は3から5であることを示しています。
レコードの定義はdeclarationブロックと同様です。
レコードの定義はキーワード"record"でスタートし、その後にリストのフィールド名とタイプが続き、キーワード"end-record"で定義の終わりを明示します。
このレコードの定義はdeclarationブロック内に配置しなければなりません。
下記のコードはネットワーク内のアークを表現するために使用し、3つのフィールド(Source,Sink,Cost)を展開し定義します。

```
declarations
arc = record
Source,Sink: string ! Source and sink of arc
Cost: real ! Cost coefficient
end-record
OneArc: arc
ARCS: array(ARCSET:range) of arc
end-declarations
OneArc.Source:= "A"; OneArc.Sink:= "B"
```

Mosel言語の定義済みタイプと同様に処理される新しいタイプとして定義することも可能です。
レコード"arc" の名前は上記のレコードのようなユーザー型定義の例です。

3.2 データ/データファイルの初期化

Moselモデルにダイレクトに値を割り当てる場合や、外部ファイルから読み込んだ値で初期化する場合のデータ構造について解説いたします。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
declarations
V: real
A1: array(3..5) of real
S: set of string
L: list of integer
A2: array(range,range) of boolean
AL: array(set of string) of list of real
end-declarations
V := 0.5 ! Assign a value to a scalar
A1 :: [1.5, 2.3, 4.5] ! Initialize an array with known index range
A1(3) := 1.8 ! (Re)Assign a single array entry
S := {"A", "BC", "DEF"} ! Assign a set
L := [1, 2, 3, 3, 4, 5, 4] ! Assign a list
A2 :: (1..2 0..2)[true, false, false, ! Initialize a 2-dim. array
true, true, false]
AL :: (["A", "B"])[[3, -6, 1.5], [2, 8.4]] ! Initialize an array of lists
AL("C") := [1, 2.5, 4, -0.5] ! Assign an array entry
```

下記の例題は、Moselに数値割り当てを行う場合の例題です。
データファイルから読み込みを行う場合、通常ダイナミックデータが使われます。
この例題に見られるようにMoselのinitializationsセクションを使い表現し展開します。

```
declarations
A: array(1..6) of real ! Static array definition
S: set of string ! Dynamic set
C: array(S) of real ! Dynamic array
L: array(set of integer) of list of string ! Dynamic array of lists
R: array(range) of arc ! Dynamic array of records
end-declarations
initializations from "initdata.dat"
A C S L R
end-initializations
writeln("I = ", I, "¥nA = ", A, "¥nC = ", C )
writeln("S = ", S, "¥nL = ", L, "¥nR = ", R)
```

データファイルinitdata.datは下記のプログラムにより読み込みを行います。
静的配列Aのインデックスは概知数であり、この値は単純リストとして与えられます。
動的配列に関しても、データファイルにインデックスを含める必要があります。
このデータファイルによりプログラムは下記のようなアウト・プットとなります。

```
I: 10
A: [2 4 6 8]
C: [("red") 3 ("green") 4.5 ("blue") 6 ("yellow") 2.1]
S: ["white" "black"]
L: [(1) ["a" "b"] (3) ["c" "d" "e" "f"] (6) ["i" "i" "i"]]
R: [(1) ["A" "B" 1.2] (2) ["A" "C" 4.5] (3) ["B" "C" 3] ]
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

動的配列Aにおいて、全入力が定義されたので、インデックスを指定する必要はありません。
動的配列n組リストとして一番目のn-1構成要素にプリントされ、これがインデックスおよび
配列エントリ値のリストです。
さらに順応性を高めるために、外部ファイルへの読み込み/入力を行うため手続き、read/
readlnまたはwrite/writelnの使用も可能です。

```
# Data:
A(1) = 5.2
A(2) = 3.4
```

ここでmoselプログラムに下記のようなコーティングを行います。

```
declarations
j: integer
B: array(range) of real
end-declarations
fopen("readdata.dat", F_INPUT)
fskipline("#") ! Skip lines starting with a '#'
repeat
readln('A(' j,') =',B(j)) ! Read the indices and the value on a line
until (getparam("nbread")<4) ! until a line is not properly constructed
fclose(F_INPUT)
```

3.3 言語構造

データタイプの他に、Moselはプログラミング言語に必要とされる基本的はフローコントロールも備えています。（selectionsおよびloops）

3.3.1 セレクション

最も簡単な選択形式はスタートメントif-thenです。
もう一方の選択肢を連続して実行し二つの構造をテストするためにif-then-elseまたはif-then-elif-then-slseを拡張しても構いません。
下記の例題は二つとも実行しない場合の例題です。

```
declarations
A : integer
x: mpvar
end-declarations
if A >= 20 then
x <= 7
elif A <= 10 then
x >= 35
else
x = 0
end-if
```

Aの値が20より大きい、または等しい場合、上限値7は変数xに加えられ、Aの数値が10より小さいまたは等しい場合、下限値35がxに加えられます。
その他のケースでは（すなわちAの数値が10より大きく、20より小さい場合）xは0に固定されます。
それぞれ相互に排他的条件の場合、（ここではAの値）テストを行い、caseスタートメントは下記のように使用してください。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
declarations
A : integer
x: mpvar
end-declarations
case A of
  #NAME?
20..MAX_INT : x <= 7
12, 15 : x = 1
else x = 0
end-if
```

3.4 ループ

ループ群の実行を何回、繰り返す必要があるのかということは、
あるインデックスの全ての値やカウンタ(forall)または条件が満たされているかどうかに
関係します。
Moselにはforallとwhileループの二つのバージョンがあります。

```
declarations
x: array(1..10) of mpvar
end-declarations
forall(i in 1..10) x(i) is_binary
```

inline:単一スタートメント全体をループする場合
forall-do(while-do);スタートメントブロックを囲う場合、
end-doの表現で終わらせる場合

```
declarations
x: array(1..10) of mpvar
end-declarations
forall(i in 1..10) do
x(i) is_integer
x(i) <= 100
end-do
```

3.5 例題 セットで実行する

下記の例題では集合演算子といくつかのループタイプを使い解説を行います。
このプログラムはSieve of Eratosthenesを使い、素数集合は2と指定して上限間を計算します。
(各素数、すなわち発見された全倍数は残存数集合から削除されます。)

集合演算子 サブセットは演算子+ =や- =が使われている集合から追加または削除が可能です。
(この集合はインデックスセットとして一度使われた場合、サイズは縮小しません。)
Moselはセットにスタンダードな演算子もまた定義します。
union:交差と差(演算子+ * -)
Run time parameter この例題ではパラメータセクションを導入しています。
このセクションで定数値が定義されモデルの実行により、リセットすることも可能です。または指定されたデフォルト値も使うことが可能です。
ここではプログラムを実行する人が集合数の上限を選択することが可能です。(その他一般的な
使用法としては、パラメータの記述方法でデータファイル名を指定することも可能です。)

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
model Prime
parameters
LIMIT=100 ! Search for prime numbers in 2..LIMIT
end-parameters
declarations
SNumbers: set of integer ! Set of numbers to be checked
SPrime: set of integer ! Set of prime numbers
end-declarations
SNumbers:={2..LIMIT}
writeln("Prime numbers between 2 and ", LIMIT, ":")
n:=2
repeat
while (not(n in SNumbers)) n+=1
SPrime += {n}
i:=n
while (i<=LIMIT) do
SNumbers-= {i}
i+=n
end-do
until SNumbers={}
writeln(SPrime)
end-model
```

3.6 サブルーチン

Mosel は定義済みサブルーチンセットを備えています。
例えば、write/writelnなどの手続きやcos, exp, ln, などの論理的関数、または特定プログラムの必要に応じて、新しい手続きや関数を定義することも可能です。
moselでユーザー定義されたサブルーチンはそれぞれend-procedure and function/end-function手続きで命令する必要があります。
関数の戻り値は下記の例題に見られるように、値が戻され、割り当てられます。
パラメータをサブルーチンへわたすことも可能です。
このパラメータは下記のサブルーチン名をカッコ内に追加します。

対応するサブルーチンにおいてのみ有効な宣言を行う部分的なパラメータ向けdeclarationセクションを包括している場合、このサブルーチン構造はモデルと非常に類似した構造となります。

Forward declaration:Mosel ではユーザーがキーワードforwardを使いサブルーチンの定義とは独立的にサブルーチンを宣言することができます。

```
model "Simple subroutines"
function timestwo(b:integer):integer
returned := 2*b
end-function
procedure printstart
writeln("The program starts here.")
end-procedure
printstart
a:=3
writeln("a = ", a)
a:=timestwo(a)
writeln("a = ", a)
end-model
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

Overloading: Moselではサブルーチン名を再利用することができ、様々な数値またはパラメータタイプの全種類が備わっています。この機能は一般的にOverloadingと呼ばれています。
ユーザーはMoselで自身の関数や手続きで定義しオーバーロードされたサブルーチンを定義することが可能です。

4 Mosel ライブラリ

Mosel言語で作成されたモデルは、Mosel C インターフェースを通じC言語から操作が可能です。（利用可能なインターフェース Java, C#, Visual Basic）
このインターフェースは2つのインターフェース形式で提供しています。
特に、モデル統合やMoselで生成したソリューション・アルゴリズムを
様々なシステム内で実行したい場合に使用します。
またすでにアルゴリズムの既存パーツがC言語で生成されている場合などMosel言語を他のソフトウエアでインターフェースを行う場合に使用します。
Moselモデルコンパイラー・ライブラリはモデルファイルをBIMファイル(Binary Model file)へコンパイルするために使用します。
その後、このBIMファイルはモデルを実行するためにMosel Runtimeライブラリによってインプットを行います。
Moselライブラリを使うことで、コンパイルやモデル実行だけではなく様々なモデリングオブジェクトの情報へアクセスすることも可能になります。
下記の例題では3.5章のモデルを使いどのようにコンパイルを行うか、またパラメータ制限として異なる値でモデルを実行し素数の結果のプリントを行います。
（この例題では、model prime2を使い実際に実行を行いますが、C言語で生成されているためアウトプットの印刷は行いません）
目標結果（2から500間の素数）を含む集合SprimeのコンテンツをプリントするためまずXPRMfindident関数を使いMosel参照にこのオブジェクトをリトリーブする必要がります。
その後、集合要素を列挙し各値を得ることも可能です。

```
#include <stdio.h>
#include "xprm_mc.h"
#include "xprm_rt.h"
int main()
{
XPRMmodel mod;
XPRMalltypes rvalue, setitem;
XPRMset set;
int result, i, size, first, last;
XPRMinit();
XPRMcompmod(NULL, "prime2.mos", NULL, NULL); /* Compile model Prime2 */
mod=XPRMloadmod("prime2.bim", NULL); /* Load the BIM file */
XPRMrunmod(mod, &result, "LIMIT=500"); /* Run the model */
XPRMfindident(mod, "SPrime", &rvalue); /* Get the object 'SPrime' */
set = rvalue.set;
size = XPRMgetsetsize(set); /* Get the size of the set */
if(size>0)
{
first = XPRMgetfirstsetndx(set); /* Get number of the first index */
last = XPRMgetlastsetndx(set); /* Get number of the last index */
printf("Prime numbers from 2 to 500:¥n");
for(i=first;i<=last;i++) /* Print all set elements */
printf(" %d,", XPRMgetelsetval(set,i,&setitem)->integer);
printf("¥n");
}
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
XPRMfinish();
return 0;
}
```

5 モジュール

Moselの独自機能であるモジュールを利用することで言語を拡張することが可能です。
(Mosel Nativeインターフェースによるコンベーション設定に見られるように、ダイナミック・ライブラリではC言語で生成されます。)
下記に記したものを使い1つのモジュールでMosel言語を拡張することが可能です。
・新しい定数記号(constant symbols)
・新しいサブルーチン(subroutines)
・新しいタイプ(Type)
・新しいI/Oドライバ
・新しいコントロール・パーラメント(control parameters)
このリスト上で、サブルーチンとタイプは最も重要なアイテムです。
サブルーチン:まったく新しい関数、または手続き、オーバーロードを既存するMoselのサブルーチンによってモジュールが定義されます。
モジュールは、例えばアルゴリズムまたはヒューリスティクス解をコールするサブルーチンを備えており、CまたはC++ライブラリ機能形式で簡単に利用することが可能です。
モジュールにより定義された新しいタイプは自身のMoselタイプと全く同様に処理されます。
これらは複合データ構造に利用することが可能です。(配列、集合etc...)初期設定を行うセクションでファイルからの読み込みや、サブルーチン、パラメータとして表現されます。
Moselにおけるこの操作は新しいタイプとして実行するために、オーバーロードすることが可能です。
新しいタイプの定義は、有限ドメイン制約ソルバーなどでサポートすることができます。
MoselはI/Oドライバのセットとともに販売しており、特定のデータソースのインターフェース(ODBCなど)として、またはメモリー内でデータパッシングのやりとりを行う様々な機能により、非常ににダイレクトな方法でアプリケーションを実行するMoselライブラリとMoselモデル間の情報交換が行えます。
ユーザーの皆様方は、例えば、圧縮ファイルまたは暗号化されたファイルの読み込み/生成などを追加ドライバで定義することが可能です。
モジュールによって提供されたconstantおよびcontrolparameterはそれ自身だけではほとんど意味を持たず、constantおよびcontrolparameterは通常これらのタイプまたはサブルーチンとともに併用して使います。
モジュールは、ソリューションアルゴリズムのラピッド・プロトタイピングとして、例えば様々なエリアからのソリューション戦略やソルバーの組み合わせを含む適切な方法として使用することができます。
例えば、Kalisのように、有限ドメイン制約ソルバーへアクセスするために提供されているmmxprsやモジュールを使用します。
制約伝搬アルゴリズムを解くbranching Solvers subproblemの全てのノード探査を定義し、その結果次第でMIP問題を生成カットすることが可能です。
(この例題の実装に関する詳細情報はXpress Whitepaper *Hybrid MIP/CP solving with Xpress-Optimizer and Xpress-Kali*を参照されたい)

5.1 利用可能なモジュール

現在、下記のモジュールをご利用いただけます。
Solvers: mmxprs, mmquad, mmnl, mmxslp, mmsp, kalis
Data handling: mmodbc, mmoci, mmetc
System: mmsystem
Model handling: mmjobs
Graphics: mmive, mmxad

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
uses "mmxprs"
declarations
x: array(1..10) of mpvar
end-declarations
public procedure printsol
writeln("Solution:", getsol(Objective))
forall(i in 1..10) write("x(",i,")=", getsol(x(i)), "¥t")
writeln
end-procedure
setcallback(XPRS_CB_INTSOL, "printsol")
```

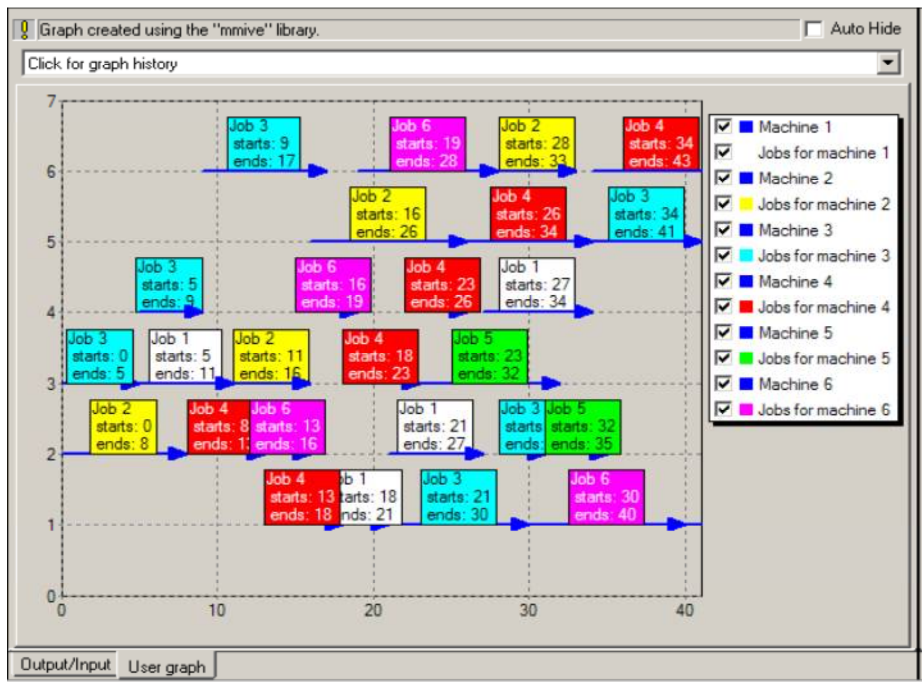
前記したモジュールmmxprsの例題は、すでにXpress-Optimizerで問題を解くために、使用しました。
基本的なソリューション作業やMosel言語からアクセス可能なアルゴリズムのセッティングを生成するほか、このモジュールの特別な機能は、下記の例題に見られるように、mosel内からCライブラリの基本ソルバーのcallback functionを定義することが可能な点です。
下記の例題は現在のソリューションをプリントアウトするため関数を定義します。すなわち整数解をコールする度に発見されることを意味します。
モジュールmmquadのサポートにより二次計画問題の形式化および問題を解くことが可能です。（5.3章の例題を参照されたい。）

最新の追加としては、非線形制約式を扱うためのモジュールmmnlです。
このモジュールはソルバーではありません。mmvprsと組み合わせることで二次的に制約された二次計画問題や線形的に定式化されたコンベックス最適化問題を定式化し解くことができるようになります。
一般的な非線形問題は逐次線形プログラミング（SLP）を行うモジュールmmxslpを用いて定式化し解くことが可能です。
このモジュールmmspはMoselでの確率プログラミング(SP)をサポートするためのモジュールです。
モジュールKalisはArtelyにより制約プログラミングを行うソルバーkalisにアクセスするためのモジュールです。
モジュールで、データファイルに追加したいインターフェースを定義することが可能です。
・mmetcはMPモデルのデータ入力/アウトプットを列挙する手続きディスクデータを定義します。[Dash,1999]
・mmociはOracleデータベースに特定のインターフェースを定義します。
・mmodbcはMoselの初期設定ブロック、またはスタンダードSQLを通じ、多目的な柔軟性を用いることによりODBCインターフェースなどのいかなるデータソースへのアクセスも可能です。
またこのモジュールはExcel使用のインターフェース・ソフトウエアも定義します。
下記のプログラムはODBCの接続を通じ、Ms-Excelスプレットシートから２次元配列とそのサイズを抽出し読み込んだものです。
：注　異なるデータソースへの切り替えは、データベースのように単にファイル名の変更を行うだけです。

```
model sizes
uses "mmodbc"
declarations
Nprod, Nrm: integer
end-declarations
initializations from 'mmodbc.odbc:ssxmpl.xls'
Nprod Nrm
end-initializations
declarations
PneedsR: array(1..Nprod,1..Nrm) of real
end-declarations
initializations from 'mmodbc.odbc:ssxmpl.xls'
PneedsR as 'USAGE'
end-initializations
end-model
```

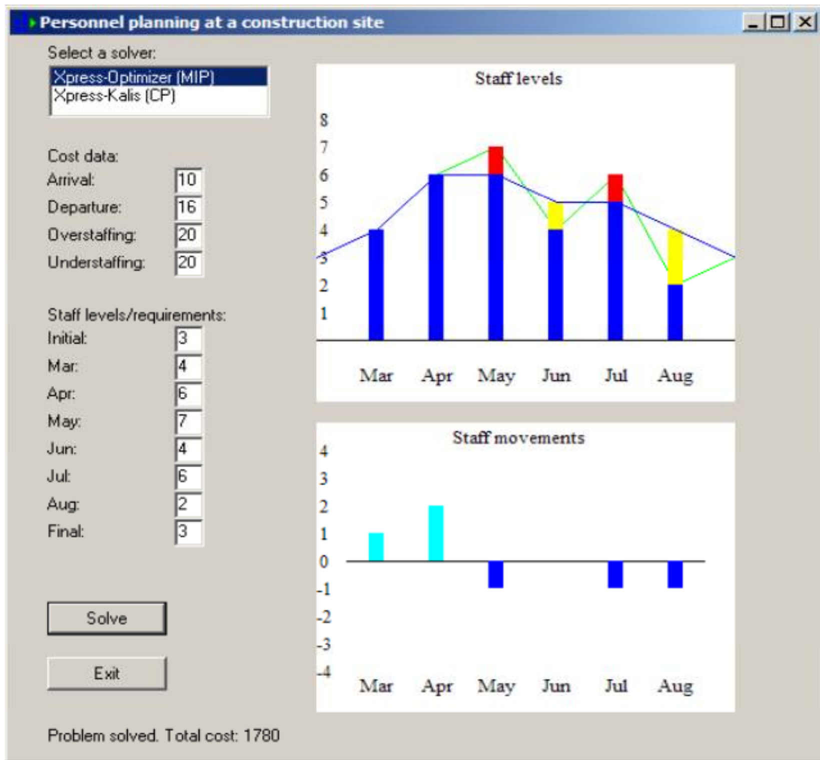
Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

モジュールmmsystemはgettimeやファイル処理機能のような機能を備えています。
このモジュールではmosel言語からオペレーティング・システム・コマンドを使うことが可能です。
後者の利益、費用について記載しています。
モジュールmmiveはユーザーの皆様のグラフを生成するために(図2のようなグラフ)グラフィカルインターフェースXpress-IVEによりMoselモデルの作業が行えます。



Xpress Application Desigher(モジュールmmxad)は、図3の人員計画アプリケーションなどMoselで完全なグラフィカルアプリケーションの生成が行えます。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved



5.2 mmxprs;MIPに変数をヒューリスティックスに固定する

この章ではXpress-Optimizerを使い、混合整数計画問題を解くためのヒューリスティック解の例題を使います。
全てのサブルーチン、タイプ、定数およびコントロールパラメータはこのMosel言語のソルバーモジュールがボルト体で表示されます。
問題の実装システムをサポートすることを目的とした問題の形式化やソリューションアルゴリズムは複数のサブルーチンに分解するだけではなくメイン・モデルファイルにより別のファイルに含められます。

```
model "Fixing binary variables"
uses "mmxprs"
include "fixbv_pb.mos"
include "fixbv_solve.mos"
solution:=solve
writeln("The objective value is: ", solution)
end-model
```

下記の例題はファイルfixbv_pb.mosに含まれているモデル定義の展開です。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
declarations
RF=1..2 ! Range of factories (f)
RT=1..4 ! Range of time periods (t)
(...)
open: array(RF,RT) of mpvar
end-declarations
(...)
forall(f in RF,t in RT) open(f,t) is_binary
```

このモデルには下記のステップで実装を行った場合、変数をヒューリスティクスに固定するためにバイナリ変数が読み込まれます。
・LP問題を解く
・約0か1の値でバイナリ変数を固定
・得られたMIP問題を再保存し、バウンドとして修正した問題のソリューション値を使用し最初のMIP問題を解く
・ヒューリスティクス解を実装した関数を解き、ファイルfixbv_solv.mosに定義する

```
function solve:real
declarations
TOL=5.0E-4
osol: array(RF,RT) of real
bas: basis
end-declarations
setparam("zerotol", TOL) ! Set Mosel comparison tolerance
setparam("XPRS_CUTSTRATEGY", 0)
setparam("XPRS_HEURSTRATEGY", 0)
setparam("XPRS_PRESOLVE", 0)
maximize(XPRS_TOP, MaxProfit) ! Solve the LP problem
savebasis(bas) ! Save the current basis
forall(f in RF, t in RT) do ! "Round" binaries
osol(f,t):= getsol(open(f,t))
if osol(f,t) = 0 then
setub(open(f,t), 0)
elif osol(f,t) = 1 then
setlb(open(f,t), 1)
end-if
end-do
maximize(MaxProfit) ! Solve the modified MIP
ifgsol:=false
if getprobstat=XPRS_OPT then ! If an integer feas. solution was found
ifgsol:=true
solval:=getobjval ! Get the value of the best solution
end-if
forall(f in RF, t in RT) ! Restore the original problem
if ((osol(f,t) = 0) or (osol(f,t) = 1)) then
setlb(open(f,t), 0); setub(open(f,t), 1)
end-if
loadbasis(bas) ! Reload the basis
if ifgsol then ! Set the "cutoff" to the
setparam("XPRS_MIPABSCUTOFF", solval) ! best known solution
end-if
maximize(MaxProfit) ! Solve the original MIP
returned:=if(getprobstat=XPRS_OPT,getobjval,solval)
end-function
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

5.3 mmquad: QPを定義し解く

先に述べたように、モジュールmmquadは二次計画問題を定式化し、解くことが可能です。
このモジュールは新しいタイプqexpを定義します。これはXpress-QPソルバーによるインプットとして提供されます。この場合、Mosel言語の拡張はモジュールmmquadにより提供され、他のモジュールでも扱うことが可能です。(*inter-module communication*)
下記の例題では組み合わせポートフォリオを決定したいと思います。
資産の上限および、総額の制限を条件として選択します。
検討している資産間の関連性は二次費用関数により導かれます。

```
model Portfolio
uses "mmxprs","mmquad"
parameters
DATAFILE = "portf.dat" ! Name of the data file
LIMIT = 20 ! Maximum number to be chosen
end-parameters
declarations
NVAL = 30 ! Total number of assets
RV = 1..NVAL
LCOST: array(RV) of real ! Coeff. of linear part of the obj.
QCOST: array(RV,RV) of real ! Coeff. of quadratic part of the obj.
UBND: array(RV) of real ! Upper bound values
n: integer ! Counter for chosen assets
x: array(RV) of mpvar ! Amount taken into the portfolio
y: array(RV) of mpvar ! 1 if asset i is chosen, else 0
Cost: qexp ! Objective function
end-declarations
initializations from DATAFILE
UBND LCOST QCOST
end-initializations
Cost:= sum(i in RV) ( LCOST(i)*x(i) + ! Define the (quadratic) cost function
QCOST(i,i)*x(i)^2 +
sum(j in i+1..NVAL) QCOST(i,j)*x(i)*x(j) )
sum(i in RV) x(i) = 100 ! Amounts chosen must add up to 100%
sum(i in RV) y(i) <= LIMIT ! Limit on total number of values
forall(i in RV) do
x(i) <= UBND(i)*y(i) ! Upper limits
y(i) is_binary ! Variables are binary
end-do
minimize(Cost) ! Minimize the total cost
writeln("Solution: ", getobjval) ! Solution printing
writeln("(quadratic part: ",
getsol(sum(i in RV) ( QCOST(i,i)*x(i)^2 +
sum(j in i+1..NVAL) QCOST(i,j)*x(i)*x(j))), ")" )
forall(i in RV)
if(getsol(y(i)) > 0.000001) then
writeln(i, ": ", getsol(x(i)))
n+=1
end-if
writeln("¥n", n, " assets have been selected")
end-model
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

6 パッケージ

Mosel言語はユーザーの皆様のwrittenライブラリを通じ、拡張することが可能です。
ライブラリは2つの形式を使うことができます。
・パッケージ　Mosel言語で書かれたライブラリは新しい制約式、サブルーチンおよびMoselタイプを定義します。
・モジュール　Cプログラミング言語で書かれたダイナミックライブラリ(Dynamic Shared Object, DSO)
この章の後半で、ユーザーパッケージに関する例題を解説します。
ユーザーモジュールの2つの例題は、次の章で解説いたします。
パッケージ構造は、モデルと似ており、パッケージによりキーワードmodelを置換します。
パッケージはモデル内にusesステートメントとして含まれており、モジュールを使用する場合と同様です。
moselコードとは異なり、モデル内に含まれるインクルード・ステートメント、パッケージは別々にコンパイルされ、
これらのコンテンツは、ユーザーは見えません。
代表的なパッケージの使用法は
・私的なtool boxの開発
・コンテンツを開示せずに提供することを望むMosel言語で生成されたモデル部分（再形式化など）やアルゴリズムを含む場合。
・モジュールのアドオンをMosel言語でさらに読みやすく生成する場合。

6.1 新しいサブルーチンを書く

問題を解いたのち、その解をリトリーブし、保存を行いたい場合、Moselは一つずつソリューション値にアクセスするための関数getsolを備えていますが、一度に一つの意思決定変数配列のソリューションへアクセスし解くためのサブルーチンではありません。
ですが、このような作業を実行するサブルーチンはモジュール形式で簡単に実装できます。
前章の例題では下記のようなソリューションをプリントアウトすることができます。

すでに解説したように、モジュールsolarrayの環境を使った場合、下記のMoselコードにsolarrayのパッケージを導入します。

```
model Portfolio
uses "solarray", "mmxprs"
... ! Data initialization
declarations
x: array(RV) of mpvar ! Amount taken into the portfolio
sol: array(RV) of real ! Solution values
end-declarations
... ! Formulate and solve the problem
solarray(x,sol) ! Retrieve the solution for all variables
writeln(sol) ! Print the solution
end-model
```

このモジュールは配列にタイプおよびインデックス集合数を処理する、単一C関数のみを定義するのに対し、
配列インデックス集合の全コンフィギュレーションを使う場合にはここで一度オーバーロードされたサブルーチンsolarrayを明示的に定義する必要があります。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
package solarraypkg
public procedure solarray(x:array(R:set of integer) of mpvar,
s:array(set of integer) of real)
forall(i in R) s(i):=getsol(x(i))
end-procedure
public procedure solarray(x:array(R1:set of integer,
R2:set of integer) of mpvar,
s:array(set of integer,
set of integer) of real)
forall(i in R1, j in R2) s(i,j):=getsol(x(i,j))
end-procedure
public procedure solarray(x:array(R1:set of integer,
R2:set of integer,
R3:set of integer) of mpvar,
s:array(set of integer,
set of integer,
set of integer) of real)
forall(i in R1, j in R2, k in R3) s(i,j,k):=getsol(x(i,j,k))
end-procedure
end-package
```

このパッケージコードはsolarray.mosとしてセーブされBIMファイルにスタンダードMoselモデルと同様な方法でコンパイルされます。
（ファイル名をパッケージ名で表示している場所がsolarray.bimです。）
作業ディレクトリに含まれていない場合、環境変数MOSEL-DSOをその場所に設定する必要があります。

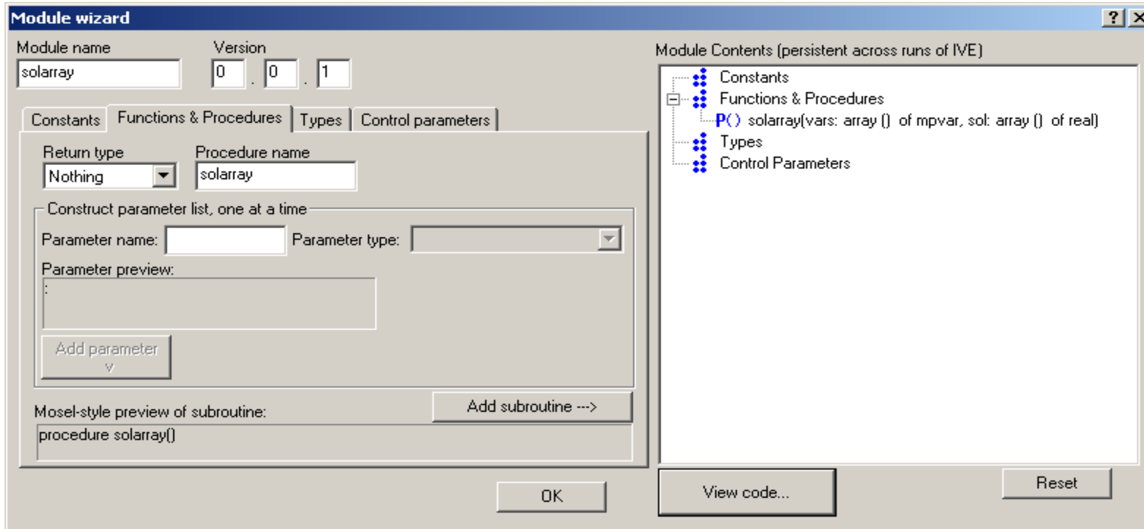
7 ユーザーモジュールで生成する

オペレーターシステムの観点から、モジュールはCプログラミング言語で生成されているダイナミックライブラリです。(Dynamic Shared Object,DSO)
Moselネイティブインターフェースはコンベンションセットであり、DSOはMoselのモジュールとして提供されることにご注意ください。
Cライブラリ形式で実装することができる機能に関しましては、Mosel言語内からの利用が可能です。
この章では演算子を使いこの作業を行うために新しいサブルーチン（1）、および新しいタイプの定義（2）をどのように行うかを解説いたします。

7.1 新しいサブルーチンを定義する

前章で解説したパッケージsolarrayのsolarray関数をモジュールsolarrayより同様に定義することが可能であり、この機能を使うことでMoselモデルへの修正が不要です。
solarrayモジュールは配列に様々なタイプやインデックス数を用いて、割り当てる単一C関数を定義します。
その一方、Mosel言語では、配列インデックス集合の設定を利用する場合、サブルーチンsolarrayの一度オーバーロードしたものを明示的に定義する必要があります。
この新しい機能として提供しているモジュールの実装を行うためのコードは下記の通りです。
多目的なインターフェース構造、初期化機能およびsolarrayを実装するプラトタイプ・ライブラリ関数ar_getsolもまたXpress-IVEのモジュール生成機能により自動的に生成することが可能です。
ユーザーが行わなければならない作業は、関数ar_getsolの本文を埋める作業です。
ここではエラー処理を省略し、理解しやすいようにしています。この関数により実行する必須オペレーションのみを記載しています。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved



①　Moselスタックから二つの配列参照を得る（意思決定変数の１配列と実数の1配列）
②変数配列に定義された入力を列挙する（スペースもしくはデンスの場合はMAXIM次元まで可能）
③各入力に対するソリューション値を得て、実数配列内へソリューション値をコピーする

```
#include <stdlib.h>
#include "xprm_ni.h"
#define MAXDIM 20
static int ar_getsol(XPRMcontext ctx,void *libctx);
/* List of subroutines */
static XPRMdsofct tabfct[]=
{
{"solarray", 1000, XPRM_TYP_NOT, 2, "A.vA.r", ar_getsol}
};
/* Interface structure */
static XPRMdsointer dsointer=
{
0, NULL,
sizeof(tabfct)/sizeof(XPRMdsofct), tabfct,
0, NULL,
0, NULL
};
/* Structure for getting function list from Mosel */
static XPRMnifct mm;
/* Module initialization function */
DSO_INIT solarray_init(XPRMnifct nifct, int *interver, int *libver,
XPRMdsointer **interf)
{
mm=nifct; /* Get the list of Mosel functions */
*interver=XPRM_NIVERS; /* Mosel NI version */
*libver=XPRM_MKVER(0,0,1); /* Module version: must be <= Mosel NI version */
*interf=&dsointer; /* Pass info about module contents to Mosel */
return 0;
}
static int ar_getsol(XPRMcontext ctx,void *libctx)
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
{
XPRMarray varr, solarr;
XPRMmpvar var;
int indices[MAXDIM];
/* Get variable and solution arrays from stack in the order that they are
used as parameters for 'getsol' */
varr=XPRM_POP_REF(ctx);
solarr=XPRM_POP_REF(ctx);
/* Error handling:
 #NAME?
 #NAME?
*/
/* Get the solution values for all variables and copy them into the solution
array */
if(!mm->getfirstarrtruentry(varr,indices))
do
{
mm->getarrval(varr,indices,&var);
mm->setarrvalreal(ctx,solarr,indices,mm->getvsol(ctx,var));
} while(!mm->getnextarrtruentry(varr,indices));
return XPRM_RT_OK;
}
```

このコードはダイナミックライブラリに拡張子.dsoを与えてコンパイルするために、必要不可欠なコードです。
その後、その場所をMoselに認識させます。（この作業は環境変数MOSEL_DSOで設定します。）
そしてこれを、分散型Moselモジュールとして利用することが可能です。
例題に見られるように、実装言語拡張の形式をモジュールもしくは、パッケージとするのかを選択することが可能です。

パッケージは、スタンダードMoselモジュールのような動作をし、モジュールでは、低レベル言語で作成されます。
一般的に高レベルの開発は努力を要しますが、高速実行が可能です。
主要な機能（この例題に見られるような任意配列インデックスや下記のような複素数に演算子を定義する。）はモジュールでのみ提供されます。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

7.2 新しいタイプを生成する

この章では、複素数を表現するために、新しいcomplexタイプの実装法を紹介します。
Moselでは下記のようなモデル生成が可能です。

標準的な初期化や生成関数タイプ、そしてモジュールでこの新しいタイプのコンストラクター、標準的な演算子、同等演算子比較、および印刷定義を行います。
これまでのタイプは例題にある通り、Moselデータ構造（集合、配列）を用いて自動的に定義されます。
標準的なMosel算術演算子定義を用いることにより、全積または合計のような集合演算子を演繹することが可能です。
またオペランド（演算対象）および否定定義の情報交換も行えます。
同等比較定義を不等式から導いても同様に行うことができ、また標準論理演算子（ここでは定義されていない）集合演算子定義も生成されます。
印刷機能の定義は初期化とともにファイルへアウト・プットし利用することも可能です。
テキストの複数ページにこのモジュールに関する完全なコードが指定されました。
ここではこのモジュールの特徴キーワードのハイライトのみにします。

```
model Complex numbers
uses "complex"
declarations
c:complex ! Define a single complex number
t:array(1..10) of complex ! Define an array of complex numbers
end-declarations
forall(j in 1..10)
t(j):=complex(j,10-j) ! Initialize with 2 integers or reals
t(5):=complex("5+5i") ! Initialize with a string
c:=prod(i in 1..5) t(i) ! Aggregate PROD operator
if c<>0 then ! Comparison with an integer or real
writeln("Product: ", c) ! Printing a complex number
end-if
writeln("Sum: ", sum(i in 1..10) t(i)) ! Aggregate SUM operator
! Arithmetic operators
c:=t(1)*t(3)/t(4) + if(t(2)=0, t(10), t(8)) + t(5) - t(9)
initializations to "complex_out.dat" ! Output to a file
c t
end-initializations
end-model
```

・モジュールコンテキスト
・タイプ生成、削除
・文字列へタイプ変更のやりとり
・算術演算子のオーバーロード
前述した例題と同様、対応関数の本体を記入する必要があるため、IVEのモジュールコード生成関数は要求インターフェース構成を使い、生成されたことを前提としています。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

7.2.1 モジュール コンテクスト

全ての割り当てられたスペースで、モデル実行時にオブジェクトが分離し、削除される可能性があるため、モデルの実行中にモジュールは生成された全てのオブジェクトの追跡を行います。
この機能はモジュールcontextで実行されます。
この例題の場合、コンテクストは複素数の連鎖リストにすぎません。

```
typedef struct
{
s_complex *firstcomplex;
} s_cxctx;
```

どちらも複素数は下記の構造により表現されたことを前提としています。

```
typedef struct Complex
{
double re, im;
int refcnt;
struct Complex *next;
} s_complex;
```

モジュールコンテクストは、現行のコントロールパネラー値やモデル実行中にモジュール機能へ様々なコール間での保存する必要がある情報を保存するために使用することが可能です。
Service functionのリセットはモジュールを使い、最初の部分でMoselプログラムの実行削除をコールします。
最初のコールで、リセット機能、モデルコンテクストの初期設定が生成され、次のコールでこのテクスト（このモデルのモジュールにより使われた任意のリソース）が削除されます。

7.22 タイプの生成と削除

オブジェクトタイプの例として、関数を生成したり、削除するには外部タイプで表現したC構造を利用したり（生成/初期化またはデリート/リセット）モデルコンテクスト内に保存されている情報に応じてアップデートを行います。
この例題ではモジュールで生成したオブジェトに基本メモリ管理（複素数）の実装を行います。
一回ごとに一つの数が生成され、対応するスペースを割り当て、これを削除する場合は割り当てを解除します。
現実的な観点から説明すると、1つのモジュールでメモリの大部分やスペースの再利用の割り当てを行う場合、このモジュールで、より簡単に割り当てを行うことができます。
下記のように、複素数をcreation functionに定義します。
数字がすでに存在する場合、そのreference counterを増やすか、またはこの数字に新しいC構造の割り当て、および初期化を行います。

このdeletion functionは複素数を使いスペースを空けるか、モジュールコンテクストで保存してリストからそれを削除します。（この数字の参照を行わない限り、そのままです。deletion functionの場合は、reference counterが減少するだけです。）

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
static void *cx_create(XPRMcontext ctx, void *libctx, void *todup, int typnum)
{
s_cxctx *cxctx;
s_complex *complex;
if(todup!=NULL)
{
((s_complex *)todup)->refcnt++;
return todup;
}
else
{
cxctx=libctx;
complex=(s_complex *)malloc(sizeof(s_complex));
complex->next=cxctx->firstcomplex;
cxctx->firstcomplex=complex;
complex->re=complex->im=0; /* Initialize the complex number */
complex->refcnt=1;
return complex;
}
}
```

7.2.3 文字列にタイプ変換のやりとりを行う

新しいタイプの複素数が持つ初期化ブロックを利用することで、2つの関数を文字列内にその数字を変更するために定義し、文字列からその数字の初期化を行います。
関数の表現もまたこの表現とこのタイプを印刷するために手続きwritelnを使用します。
関数の読み込みもまた、タイプ・インスタンス生成関数が字列に与えられた時、適用されます。
文字列形式は言うまでもなく、このタイプによって決定します。
この例題では分かりやすい形式である"re+im*i*"です。
下記の関数は複素数をプリントします。

```
static int cx_tostr(XPRMcontext ctx, void *libctx, void *toprt, char *str,
int len, int typnum)
{
s_complex *c;
if(toprt==NULL)
{
strcpy(str, "0+0i");
return 4;
}
else
{
c=toprt;
return sprintf(str, "%g%+gi", c->re, c->im);
}
}
```

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

次の関数は複素数への読み込みを行います。

```
static int cx_fromstr(XPRMcontext ctx, void *libctx, void *toinit, const char *str,
int typnum)
{
double re,im;
s_complex *c;
if(sscanf(str,"%lf%lf",&re,&im)!=2)
return XPRM_RT_ERROR;
else
{
c=toinit;
c->re=re;
c->im=im;
return XPRM_RT_OK;
}
}
```

7.2.4 算術演算子のオーバーロード

変更のみを行うタイプ、すなわちMoselで自動的に実行され、整数値から実数値へ変更するタイプであり、外部タイプを伴うタイプの変更は行えません。
したがって、2つの数字間を2つの複素数としてまた、複素数と実数値間のすべての演算子を定義する必要があります。
しかし可換的演算子に関しては、(加算、乗算、比較)、統合している2つのタイプを1つのバージョンで定義する必要する必要があります。
その他の意味はMoselで演繹されます。
乗算の例題を使い、乗算の2つの複素数を定義します。$(a + bi) \cdot (c + di) = ac - bd + (ad + bd)i$

```
static int cx_mul(XPRMcontext ctx, void *libctx)
{
s_complex *c1,*c2;
double re,im;
c1=XPRM_POP_REF(ctx);
c2=XPRM_POP_REF(ctx);
if(c1!=NULL)
{
if(c2!=NULL)
{
re=c1->re*c2->re-c1->im*c2->im;
im=c1->re*c2->im+c1->im*c2->re;
c1->re=re;
c1->im=im;
}
else
c1->re=c2->im=0;
}
cx_delete(ctx,libctx,c2,0);
XPRM_PUSH_REF(ctx,c1);
return XPRM_RT_OK;
}
```

そして、乗算の1つの複素数もまた実数値$(a + bi) \cdot r = ar + bri$で定義します。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

```
static int cx_mul_r(XPRMcontext ctx, void *libctx)
{
s_complex *c1;
double r;
c1=XPRM_POP_REF(ctx);
r=XPRM_POP_REAL(ctx);
if(c1!=NULL)
{
c1->re*=r;
c1->im*=r;
}
XPRM_PUSH_REF(ctx,c1);
return XPRM_RT_OK;
}
```

この演算子は可換的であり、Moselはこの場合、演繹するため複素数で乗算の実数を定義する必要はありません。
加算の2つの複素数と加算の1つの複素および実数値は乗算の方法と大変、類似した方法で実装します。
一度、加算の2つのタイプを得れば、Moselで*subtraction* (real – complex and complex – complex)を演繹するために否定(–complex)を実装するだけです。
divisionに関しては、この演算子は可換的ではないため、全てのケースを実装する必要があります。
complex/complex, complex/real and real/complex.
さらに、それぞれ加算および乗算として単位元を定義する必要があります。

```
static int cx_zero(XPRMcontext ctx, void *libctx)
{
XPRM_PUSH_REF(ctx,cx_create(ctx,libctx,NULL,0));
return XPRM_RT_OK;
}
static int cx_one(XPRMcontext ctx, void *libctx)
{
s_complex *complex;
complex=cx_create(ctx,libctx,NULL,0);
complex->re=1;
XPRM_PUSH_REF(ctx,complex);
return XPRM_RT_OK;
}
```
Writing

一度、加算と0要素が定義されれば、Moselは集合演算子SUMを演繹します。
乗算と1－要素を使い、新しいタイプとして集合演算子PRODを得ます。
このモジュールで実装されたその他の演算子は割り当ておよび比較演算子のコンストラクターです。

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved

8 おわりに

Moselは最適化問題をモデリングし解くための多目的環境を提供します。
この小冊子で解説されている例題は、Moselを様々な問題で最大限生かす方法を紹介しています。
主にオペレーションリサーチの分野の開発者の方々は自身のモデルを迅速かつ、これまでよりメンテナンスを施しやすいフォームへの実装を望んでらっしゃると思います。
実際、Mosel言語では数理モデルの定式化が代数形式に近い形で表現することが可能です。（例題集Guéret et al., 2002を参照されたい）
Moselはまた、数行のコードのバージョン・ソースからでもデータの読み込み、生成を可能にすためのデータ処理に対し十分なサポートを提供しています。
Mosel言語を使い、ソリューションアルゴリズムを生成し、直接ヒューリスティックスにモデルの可能性を拡大する時、大変、役に立ちます。

上級ユーザーの方や、研究者の方々が、特定のデータソース、新しいソルバーまたは外部のソリューションアルゴリズムをもつ、自身のアプリケーションに必要とするいかなる新しい機能も自分自身で追加が行えるこの可能性を非常に高く評価しています。
この小冊子ではMoselの素晴らしい特性を使用しています。
モジュールアーキテクチャ：softwareadやアプリケーション：拡張したMosel言語でモジュール形式に特定機能の追加を容易にする。
この方法は使用条件としてモジュールに新しいデータや変数タイプの公開を許可が可能な場合に限ります。（例えば、線形、混合整数プログラミング。ツール以外でソルバーをサポートする場合。）マトリックスベース、ソルバーまたはさまざまな費用版問題方法が使われているソフトウエアの場合。（有限領域制約ソルバーなど）
多目的関連製品群のMosel環境、FICO Xpress-Software suiteは、
一連のアプリケーション開発をカバーし、プロトタイプ版から（試作品をXpress-IVEを使うことで簡単な確認が行える）完全な最適化ソリューションの実装を行い、（開発、解析および改良をMoselデバッカ、またはXpress-Tunerなどのツールで行う）
それを企業のインフォメーション・システムに組み込んだり。外部のデータソース（データベース、オンラインデータ）へ接続を行ったり、（ライブラリ・インターフェース）または完全なるグラフィカル・アプリケーション・インターフェース（XAD）の開発を開発過程におけるいかなる状況でもソフトウエアのプラットホームを変更することなく行えます。
総合的な例題集はXpress例題集のデータベース、WEBサイトを参照されたい。
http://example.xpress.fico.com/example.pl

Bibliography

[Dash, 1999] Dash Associates. XPRESS-MP Reference Manual, 1999.
[Fourer et al., 1993] R. Fourer, D. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. The Scientific Press, San Francisco, CA, 1993.
[Guéret et al., 2002] C. Guéret, S. Heipcke, C. Prins, M. Sevaux (2002). *Applications of Optimization with Xpress-MP*. Dash Optimization, Blisworth, UK.
[Maximal, 2001] Maximal Software. MPL for Windows Reference Manual, 2001.
[Van Hentenryck, 1998] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, 1998.
Conclusion c 2009 Fair Isaac Corporation. All rights reserved. page 25

Japanese Copyright 2012-MSI Co.Ltd. All Right reserved