Contents

1. Introduction

MyM is an environment for the development, execution, and visualization of dynamic mathematical models. The class of mathematical models that can be used are systems of ordinary differential equations, difference equations, and algebraic equations.

This documents introduces the language for the definition of mathematical models within MyM via a series of examples. The language was designed so that the specification of the equations resembles the standard mathematical notation as much as possible. Many examples are provided to specify these equations. The purpose of this document is to provide an introduction tutorial to the MyM language: the features of the language are introduced via a series of examples. For a more complete description see the MyM Language Reference manual.

One of the strengths of the MyM environment is its applicability to many disciplines. All mathematical models that can be described as systems of ordinary differential equations can be handled, irrespective of whether the subject is environmental science, epidemiology, mechanical engineering, or systems control. To explain the language however, we focus on a single application area: filling and emptying containers. This frees the reader from understanding a variety of discipline specific details. We expect that the reader can transpose the examples to his own discipline. We also assume that the reader has a little programming experience in a procedural language, such as Pascal, Basic, or C.

2. Filling a container

The begin with the most simple example. A container is filled from a tap (figure 1). The flow of water from the tap is f_{in}. The volume of water w in the container can then be described by:

```
w(t)  =  w_0  +  f_in t
```

where w_0 is the volume for $t = 0$. If the container has a cross sectional area A, then the height h of the water level is given by:

```
h  =  w/A
```

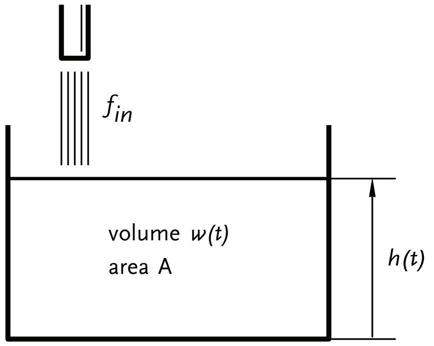

Figure 1 Filling a container

Here the assumption is made that A is large compared to f_{in}.

If we translate these mathematical equations in a MyM model, we get:

```
!
! Filling a container
!
module main;
begin
 real w_0 = 10.0; ! Volume at t = 0
 real f_in = 2.0; ! Flow into container
 real w(t);       ! Volume in container
 w(t) = w_0 + f_in*t; ! Volume equation
 real A = 2.0;    ! Area cross section
                                  container
 real h(t);       ! Water level
 h = w/A;         ! Level equation
end;
```

The MyM model is almost a direct translation of the mathematical model. Therefore understanding the MyM language code above should be easy.

Some remarks: All text on a line after an exclamation mark is ignored, and is considered to be comment. The notation is free format, i.e. spaces, tabs, and new lines have no special meaning. Also, no distinction is made between lower-case and upper-case. The declarations in the following example:

```
real a; real b;
```

are equivalent to

```
ReAl
a
; Real    B   ;
```

although their readibility differs of course.

The MyM language is based on modules. A model consists of a number of connected modules. In every MyM model at least one module must be defined: the main module. In the following sections we will use only this module. In section 7 the use of modules is discussed in more detail.

At a lower level a MyM model can be considered a sequence of statements. Each statement is terminated by a semi-colon. In the example two types of statements are used: declarations and equations. A declaration serves to make known to the system that a variable exists and what its properties are. Here all variables are declared as real variables, internally to be represented as floating point numbers. Further, the volume `w(t)` and the height `h(t)` are declared as functions of the time `t`. Adding `(t)` after the variable name suffices.

Two equations are used in the example. Equations in MyM all have the same format. A dependent variable at the left side is defined to be equal to a mathematical expression at the right side. Here `w(t)` and `h(t)` are the dependent variables. The variables `w_0`, `f_in`, and `A` are input variables, because they do not occur at the left side of an equation. Their value can be set and changed via one of the user interfaces of MyM. An initial default value can be specified with the declaration: just add =, followed by a numerical value.

The statements can be placed in an arbitrary order. The only condition to be satisfied is that a variable is declared before it is used. The variable `t`, the current time, is always automatically declared. Since MyM concerns dynamical models, `t` plays an important role. Later on we will see more examples of its use and how it is controlled.

For the mathematical expressions the same style as in procedural programming languages, like Pascal and Fortran, is used. Operators such as +, −, `*`, and / can be used, where the usual rules of precedence apply. Parentheses can be used to group parts of the expressions.
In the last expression (`h = w/A`) the dependency on time is not explicitly stated. The variables h and w were declared as time dependent, so MyM already knows that these variables depend on time. If desired however, the equation can also be notated as `h(t) = w(t)/A`.

An important final remark. A MyM model often looks deceivingly like a computer program. However, it is ***not*** !

The basic statement in a procedural programming language is the *assignment*. For instance, `x = x+1` means increment the value of x with 1. The basic statement in a MyM model is the *equation*: `a = b` means that a is equal to b, for all values of t. Hence, the MyM equation `x = x+1` makes no sense, because there is no value of `x` that can satisfy this equation.

If the user of MyM has a strong programming background, this will require a mental shift. Once again: MyM is about mathematical models, built up from equations. It is a *functional* language, not a procedural language.

3. Time varying tap

In “Figure 1 Filling a container” on page 4 the tap is open forever: f_{in} is constant. The effect of an inflow $f_{in}(t)$ that varies as a function of time is more realistic and interesting. However, the simple linear relation between w and t does not hold then anymore. It must be replaced by the following more general integral equation:

$$w(t) = w_0 + {}_0\int^t f_{in}\ dt'$$

Obviously, if f_{in} is constant, the solution of this integral equation is the linear equation presented before. We can also describe $w(t)$ via a differential equation and an initial condition:

$dw/dt = f_{in}$, $w(0) = w_0$

Although integral and differential equations are mathematically equivalent, in many cases integral equations better match the process described in a semantic sense. The following MyM model describes the container with the time varying tap.

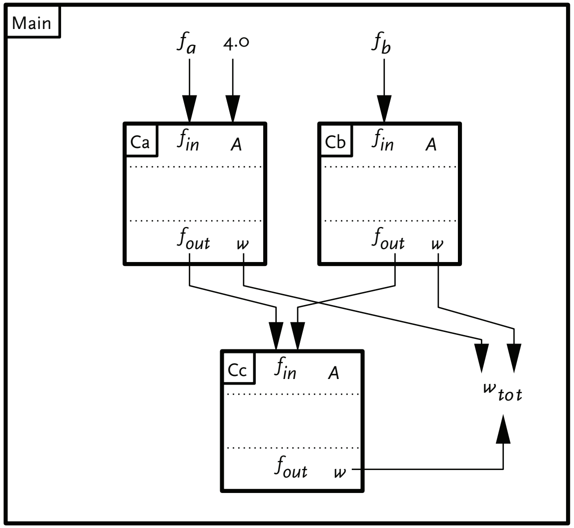

Figure 2 Inflow as a function of t

A number of changes were made to the preceding model. On the first line four real variables are declared in a single declaration. To this end the variables are listed, and separated by commas. The variable `f_in(t)` is now declared as time dependent. Via the graphical user interface its value

```
!
! Time varying tap
!
module main;
begin
 real w_0 = 10, A = 2.0, w(t), h(t);
 real f_in(t) = [0, 0,  ! f_in(0) = 0
                 1, 2,  ! f_in(1) = 2
                 4, 2,  ! f_in(4) = 2
                 5, 0]; ! f_in(5) = 0
 t.min    = 0;    ! start value t
 t.max    = 5;    ! end value t
 t.step   = 0.1; ! step size for inte-
gration
 t.method = RK2; ! Use second order
Runge Kutta
 t.sample = 0.2; ! sample rate
 w(t) = integ(f_in, w_0);
                ! Integral equation volume
 h  = w/A;
end;
```

can be entered by drawing a graph. Default values that vary with `t` can be specified in the declaration as a list of time/value pairs, enclosed by square brackets. These values are interpolated linearly.

Here we specify that from `t = 0` to `1` the tap is opened, it remains open for three units of time, and then it is closed (figure 2).

In the next five statements we specify values of attributes of t. Such statements are called *specifications*. First, we state that we are interested in the time interval from `0` to `5`. The simulation should start at `t = 0` and stop at `t = 5`. For the calculation of the values of the variables a numerical integration procedure is used. The step size of `t` to be used is `0.1`, and the method to be used is the second order Runge Kutta method. Currently, three methods are available: first, second and fourth order Runge Kutta. The keywords `RK1`, `RK2` and `RK4` refer to these methods. First order, also known as Euler's method, is the simplest. For a description of these methods see [recipes]. Finally, with `t.sample` we specify the sample rate at which values of the variables are stored for processing by the user interfaces.

All these specifications serve as defaults: they can be overruled via the user interfaces. If no specifications are used, the following default values are assumed:

```
t.min    = 0;
t.max    = 1;
t.step   = 0.1;
t.method = RK2;
t.sample = 0.1;
```

For integral equations MyM provides the function `integ(a,b)`. Here `a` is the expression to be integrated and `b` is an expression for the initial value. Two restrictions apply to its use. The `integ` function may not be used in combination with other variables or constants, and it may not be nested. Thus, the following

```
w(t) = integ(integ(f_in,0),
w_0) + 10 ;     ! wrong !
```

gives an error message.

4. A hole in the container

Let us assume that the container has a small hole in its wall (figure 3). The vertical distance of the hole to the bottom of the container is d. If $h > d$, water will leak out of the container. The velocity v of the water at the hole is given by the theorem of Torricelli:

$$v = \sqrt{\,2gi\ \max(h-d, 0)\,}$$

where g is the gravitational constant (9.81 m/s²). The outflow f_{out} is given by:

$$f_{out} = CSv$$

where S is the area of the cross section of the hole, and C is the constant of contraction. For a circular hole in a thin wall its value is 0.62 . The integral equation for $w(t)$ has to be changed to incorporate the outflow:

$$w(t) = w_0 + {}_0\int^t f_{in} - f_{out}\ dt'$$

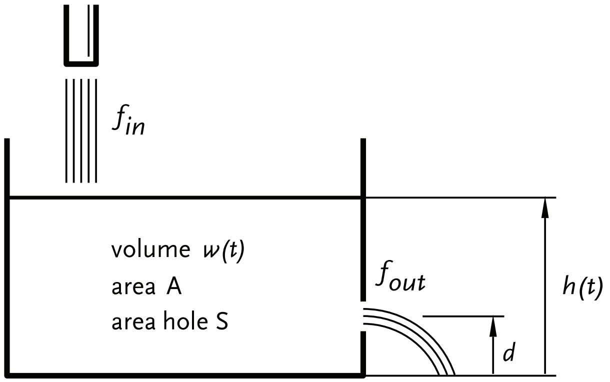

Figure 3 A hole in the container

The new MyM model becomes:

```
!
! A hole in the container
!
module main;
begin
  const g = 9.81; ! Gravitational con-
  stant
  const C = 0.62 ! Constant of contraction
  real w_0 = 10, A = 2.0, w(t), h(t);
  real f_in(t) = 2;
  real d = 0.2;    ! height of hole
  real s = 0.1;    ! area of hole;
  real v(t);       ! velocity at hole
  real f_out(t);   ! outflow through hole
  t.min    = 0;
  t.max    = 5;
  t.step   = 0.1;
  t.method = RK2;
  t.sample = 0.2;
  v(t)     = sqrt(2.0*g*max(h-d, 0));
  f_out(t) = C*s*v;
  w(t)     = integ(f_in - f_out, w_0);
  h        = w/A;
end;
```

For the constants `g` and `C` a new type of statement is used: the definition. Such statements consist of the keyword `const`, a symbolic name of the constant, followed by an equals sign and its numeric value. In later equations the value can be referred to symbolically by its name. Also, constants are hidden by the user interfaces. Their value can not be shown nor changed by the user. If the latter is not desired, e.g. NASA asks us to study containers on the moon, such constants can be changed to standard variables with a suitable default value.

Beside some declarations, also two equations were added. The equation for `v(t)` shows the use of standard functions. The function `sqrt` gives the square root of its argument. Many standard mathematical functions (trigonometric, exponential, ceiling, etc.) are provided. The function `max` gives the maximum value of its arguments. A variable number of arguments can be used. Also `min` and `avg` can be used, which give the minimum and the average value of their arguments.

In the first equation `h` is already used, while the value of `h` is defined by the fourth equation. MyM takes care of this automatically. The equations are sorted in such a way that each value is calculated before it is used.

The first three equations (v(t), f_out(t) and w(t)) can be replaced by:

```
w(t) = integ(f_in -
       C*s*sqrt(2.0*g*max(h-d, 0)), w_0);
```

and the declarations of `v(t)` and `f_out(t)` can be removed. This equation is more compact, but harder to understand and the removed variables cannot be inspected anymore with respect to their values and relations with other variables.
It is up to the developer of the model to decide which level of detail is modelled.

5. A controlled container

Suppose we want to control the level in the container, i.e. keep it as close as possible to a constant reference level *href* (figure 4). To this end we hire an operator that controls the tap. The rule he applies is:

$$fc(e) = \begin{cases} 0 & \text{if } e < 0 \\ f_{max} e / e_{max} & \text{if } e \geq 0 \text{ and } e < e_{max} \\ f_{max} & \text{otherwise} \end{cases}$$

where e is the deviation from the reference level Unfortunately, the operator has a delay t_Δ in his response. Therefore, the actual inflow $f_{in}(t)$ is given by:

$$f_{in}(t) = f_c(h(t - t_\Delta) - h_{ref})$$

The corresponding MyM model is:

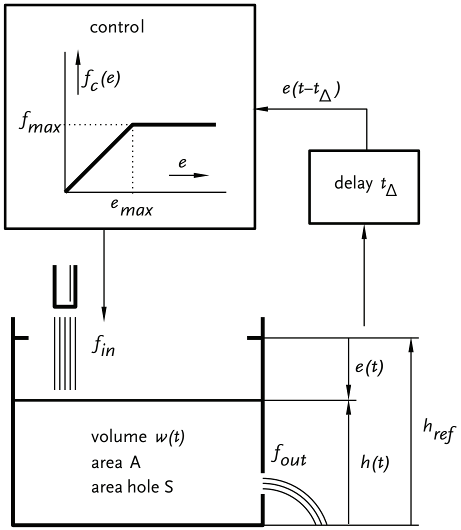

Figure 4 A controlled container

```
!
! A controlled container
!
module main;
  const g = 9.81, C = 0.62;
  real w_0 = 10, A = 2.0, w(t), h(t);
  reald=0.2,s=0.1,v(t),f_in(t),f_out(t);
  real h_ref = 3.0; ! reference height
  real e;
  real f_c(e);      ! inflow as function
  of error e
  real f_max = 2.0; ! maximum inflow
  real e_max = 1.0; ! corresponding error
  real t_del = 1.0; ! delay of control
  t.min    = 0;
  t.max    = 5;
  t.step   = 0.1;
  t.method = RK2;
  t.sample = 0.2;
  f_c(e) = switch( e < 0     ? 0 ,
        e < e_max ? f_max*e/e_max,
      else
        f_max);
  f_in(t)  = switch( t-t_del > t.min ?
        f_c(h_ref - h(t-t_del)),
      else
        0);
  v(t)     = sqrt(2.0*g*max(h-d, 0));
  f_out(t) = C*s*v;
  w(t)     = integ(f_in - f_out, w_0);
  h        = w/A;
end;
```

In this example three new features of the language are introduced. The function `f_c(e)` is defined as a function of `e`, which is a user defined free variable. The right side of the equation contains an expression in `e`. The use of time dependent variables or `t` itself is not allowed here. Such functions must be independent of time. The value of these functions can be entered as data with the declaration, or can be defined with an equation.

The `switch` function serves to handle conditional functions. The argument list after the `switch` contains a series of Boolean expressions, followed by a question mark and a value. The Boolean expressions are evaluated in order until one of them is true. The value after this expression is then used. If none of them matches, the value after the keyword `else` is used.

A general mechanism is provided to handle delays. For all time dependent variables instead of `t` an expression can be used as argument. The calculated values of such variables are stored, and via interpolation the requested values are determined.
Only values from `t.min` to `t` are known, so a `switch` function has to be used if values outside this range are requested.

6. A cascade of containers

Suppose that we have a container with a hole and a time varying inflow. What will happen if we catch the outflow of this container in another container?
And this output in another one?
Let's generalize. We have a cascade of N containers (figure 5), where the inflow of container i is the outflow of container $i-1$ for $i = 2, ..., N$. We ignore the delay caused by the distance between the containers, and we assume that the geometry of all containers is the same.
The initial volume in the first container is w_o, the others are empty. Further, we are interested in the total volume w_{tot} in all containers together:

$$w_{tot} = \sum_{i=1}^{N} w_i$$

The following MyM model does the job:

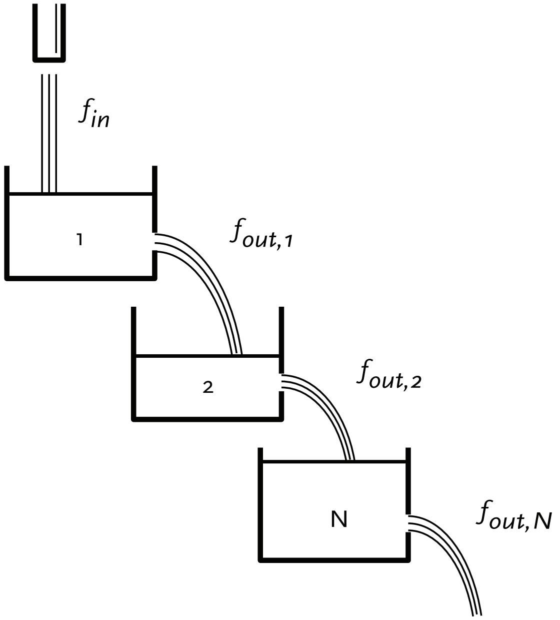

Figure 5 A cascade of containers

```
!
! A cascade of containers
!
module main;
begin
  const g   = 9.81, C = 0.62;
  real  w_0 = 10,   A = 2.0,
      d   = 0.2,  s = 0.1;
  const N = 3;    ! number of containers;
  real f_in(t) = [0, 0, 1, 2, 4, 2, 5,
  0];
                       ! For each container:
  real w[N](t);         ! volume,
  real h[N](t);         ! water level,
  real f_out[N](t);     ! outflow, and
  real v[N](t);         ! velocity at hole.
  real w_tot(t);        ! total volume
  t.min    = 0;
  t.max    = 5;
  t.step   = 0.1;
  t.method = RK2;
  t.sample = 0.2;
  v(t)     = sqrt(2.0*g*max(h-d, 0));
  f_out(t) = C*s*v;
  h        = w/A;
  w[1](t)  = integ(f_in - f_out[1], w_0);
  w[i](t)  = integ(f_out[i-1] -
                f_out[i], 0), i = 2 to N;
  w_tot    = lsum(i = 1 to N, w[i]);
end;
```

Indexed variables are represented in MyM as arrays. The example shows a number of features that are provided to simplify their manipulation. The number of containers is N, defined as a constant. The parameters for each container (`w`, `h`, `f_out`, and `v`) are declared as arrays. Each array has `N` elements, indexed from `1` to `N`. Here only one-dimensional arrays were used, but MyM permits up to twenty dimensions for arrays.

Probably most surprising are the first three equations. These are the same as in previous versions of the model. The language provides a mechanism called implicit loops. If the dependent variable is an array, then the subscripts at the left side, as well as the subscripts of arrays with the same dimensions as the dependent variable, can be dropped. The equation is automatically assumed to hold for all elements of the array. Note that to use different areas for the containers only a different declaration of `A` is required.

This mechanism cannot be used for `w[i](t)`, because the first container must be treated differently than the other containers. The equation for the volume of the first container is:

```
w[1](t) = integ(f_in - f_out[1], w_0);
```

Here explicit references are made to the first elements of the arrays `w` and `f_out`. For the remaining containers the inflow of `a` container *i* is the outflow of container *i*−`1`, with *i* = `2`, ..., *N*. For this equation we use an explicit loop, appended to the equation:

```
w[i](t) = integ(f_out[i-1] -
                f_out[i], 0), i = 2 to N;
```

The equation is defined in terms of `i`, where the loop variable `i` varies from two to *N*. The loop variables `i`, `j`, and `k` are predefined integers, and do not have to be declared.

Finally, for the calculation of `w_tot` yet another feature is used. The function `lsum` is an example of a loop function. It takes a loop and an expression as arguments. The expression is evaluated for all values of the loop variable, and, in case of `lsum`, they are added. Other available loop functions are `lprod`, `lavg`, `lmin`, and `lmax`, which respectively return the product, the average, minimum, and maximum value of the expressions.

7. Containers as modules

Suppose we have really made it in the container business. Each day requests for new simulations of different configurations enter our office. How can we handle this efficiently?
The MyM language offers a tool for this: modules. We model a general container, which we reuse in each simulation.
Consider the example in figure 6. The two containers C_a and C_b have inflows f_a and f_b. Their outflow goes into a container C_c. The area of this last container is 4, while the area of a default container is 2. Again, we are interested in the total volume w_{tot} of all containers together.

The following MyM model on the next page handles this. One general container is defined as a module, and we make three instances of it:

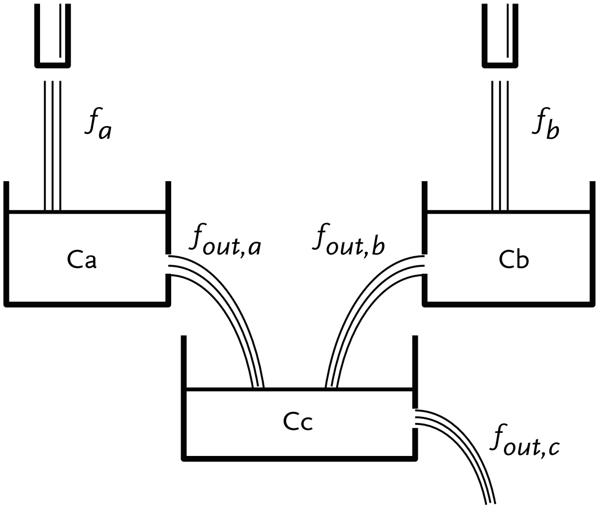

Figure 6 Containers as modules

```
!
! Containers as modules
!
const g = 9.81, C = 0.62;
module CONTAINER;
begin
  import real f_in(t);
  import real A = 2.0, s = 0.1, d = 0.2;
  export real w(t), f_out(t);
  real h(t), v(t);
  v(t)      = sqrt(2.0*g*max(h-d, 0));
  f_out(t)  = C*s*v;
  h         = w/A;
  w(t)      = integ(f_out - f_in, 0);
end;
module MAIN;
begin
  real fa(t) = [0, 0, 1, 2, 4, 2, 5, 0];
  real fb(t) = [0, 0, 1, 2, 4, 2, 5, 0];
  real w_tot(t);
  CONTAINER Ca, Cb, Cc;
  t.min      = 0;
  t.max      = 5;
  t.step     = 0.1;
  t.method   = RK2;
  t.sample   = 0.2;
  Cc.A = 4.0;
  Ca.f_in(t) = fa(t);
  Cb.f_in(t) = fb(t);
  Cc.f_in(t) = Ca.f_out(t) + Cb.f_out(t);
  w_tot = Ca.w + Cb.w + Cc.w;
end;
```

First, some constants are set. Next a general container is defined as a module. In this module some variables are declared and their relations are described with equations. Figure 3 shows the meaning of these variables.

The main purpose of modules is hiding complexity. Viewed from the outside the container has a simple function, while the actual definition and implementation is hidden. A module can thus be used as a black box. Let's see how this black box communicates with its environment.

We are not interested in the internal, or local variables `h(t)` and `v(t)`. Therefor they are declared in the standard way, and hence their value can only be used inside the module. However, we want to control the inflow `f_in(t)` from outside the module. To this end the keyword `import` has been put before the declaration of `f_in(t)`. Thus, its value can be imported from the outside world. Also, we want to have the possibility to describe different geometries of the container. We therefore declare `A`, `s`, and `d` also as import variables, and provide suitable defaults.

The result of the process in the container is the outflow `f_out(t)` and the volume `w(t)`. The keyword export has been put in front of their declaration, and then we can use these export variables outside the module.

Summarizing, within a module three types of variables can be declared: *local variables*, which can only be used inside the module; *import variables*, which value can be imported from outside the module; and *export variables*, which value can be exported outside the module. A fourth type are *global variables*. Variables declared outside all modules can be used in all modules.

Next we apply this general purpose container. The module MAIN describes the total model. Here first two inflows are declared and initialized, and the total volume is declared. The crucial statement is now:

```
CONTAINER Ca, Cb, Cc;
```

With this statement we create three instances `Ca`, `Cb`, and `Cc` of the standard container. This statement is similar to the declaration of standard variables: with

```
real x, y, z;
```

we create three named instances of the standard type real variable. With the following statements we connect the variables of the three instances:

```
Cc.A = 4.0;
Ca.f_in(t) = fa(t);
Cb.f_in(t) = fb(t);
Cc.f_in(t) = Ca.f_out(t) +
                 Cb.f_out(t);
w_tot = Ca.w + Cb.w + Cc.w;
```

The values of variables inside the instance are referred to by the sequence *instance_name*, period, *variable_name*. Import variables may only be used in the left side of an equation, export variables may only be used in the right side. With the first statement we set the area of container `Cc` to 4. The areas of other containers keep the default value, and also for the other geometric variables we rely on the defaults. Next we specify that the inflows `fa` and fb go into container `Ca` and `Cb` respectively, and that container `Cc` receives the outflow of `Ca` and `Cb`. Finally we specify that the total volume `w_tot` is equal to the sum of the volumes of the three containers. Figure 7 shows the result in terms of modules, instances, and dataflows.

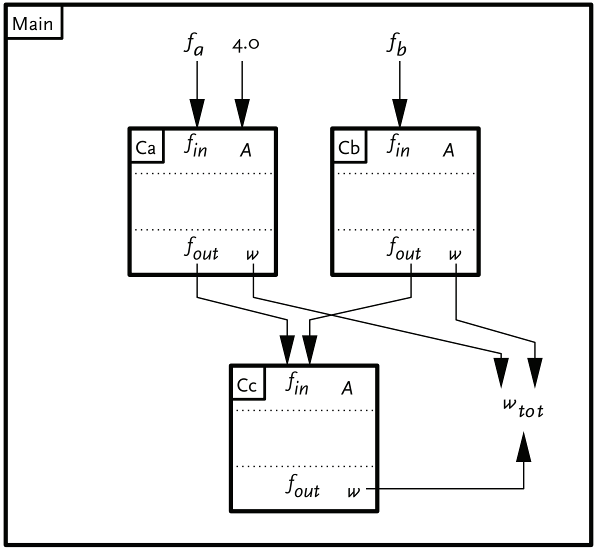

Figure 7 Data flow between modules

8. References

[1] *MyM Language Reference Manual*, 2008, Tizio.

[2] Press, W.H., Flannery, B.P, Teukolsky, S.A., Vetterling, W.T.
Numerical Recipes in C : The Art of Scientic Computing, Cambridge University Press, Cambridge, 1988.