MuPAD

An Open Computer Algebra System and its Approach of Software Integration

Ralf Hillebrand & Frank Postel

tonner@mupad.de frankp@mupad.de

Department of Mathematics and Computer Science
MuPAD Research Group
University of Paderborn, Germany

http://www.mupad.de

IMACS/ACA ´99

- MuPAD is a system for mathematical programming and mathematical visualization.
- MuPAD provides 2- and 3-dimensional visualization facilities.
- Graphical user interfaces; the most developed interfaces exist for the Windows platform.
- MuPAD provides a *functional* programming language with *procedural* extensions in Pascal-like syntax.
- Mathematical libraries for doing linear algebra, numerical computations with arbitrary-precession arithmetic, number and group theory, gröbner basis computations, statistics ...

Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

MuPAD is a system for mathematical programming and mathematical visualization. It is intended to be a *general purpose* computer algebra system.

The program consists of a small kernel written in C++ and the mathematical libraries written in the MuPAD programming language. Kernel and library are machine independent.

MuPAD provides 2- and 3-dimensional visualization facilities. Every property of a graphics scene may be set interactively or defined in a MuPAD command.

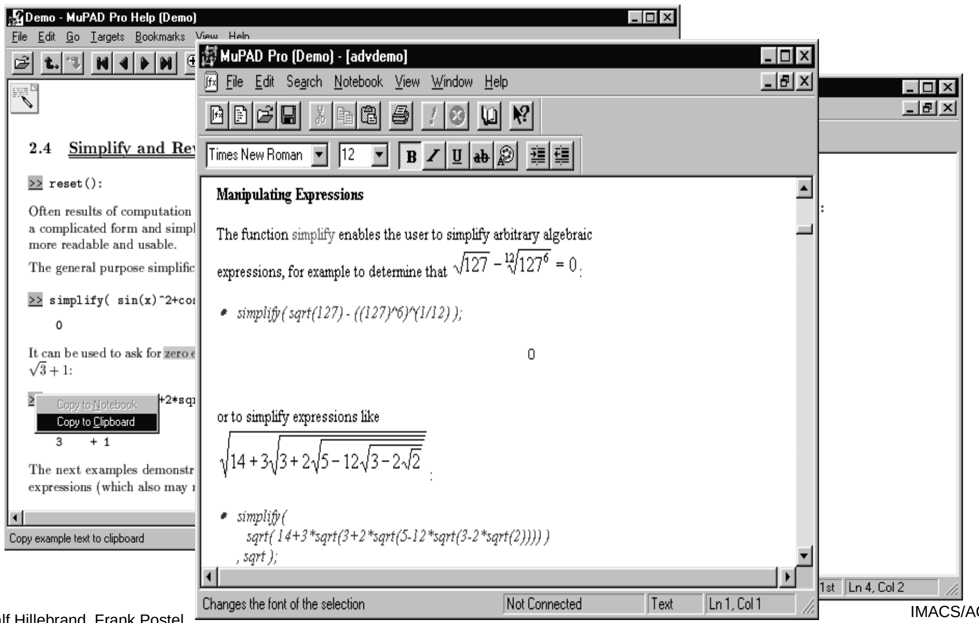
Graphical user interfaces exist for the X-Window-System, Apple Macintosh and Windows 95/NT.

The most developed interfaces for MuPAD exist for the Windows platform. MuPAD provides notebook functionality, structured session documents, editing of text documents, OLE-support, Drag&Drop, hypertext help and embedded graphics.

The language defined by the MuPAD kernel is a *functional* programming language with *procedural* extensions in Pascal-like syntax. Procedures and statements are ordinary data objects and therefore can be manipulated as an object of an orginary data type. Every language construct, including loops, control-statements and even procedures, is available as a system function.

Mathematical libraries for doing linear algebra, numerical computations with arbitrary-precession arithmetic, number and group theory, gröbner basis computations, statistics, and a lot of more.

MuPAD - The Open Computer Algebra System



Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

IMACS/ACA
25 June 1999

Three snapshots of the MuPAD Pro Windows Version.

Several testsuites were developed (e.g., see the new book from Michael Wester) which describe mathematical features of the current MuPAD version 1.4.

In our talk we do not want to give you a general introduction to the MuPAD system but to concentrate on some special features of MuPAD.

Special Features

- Domains - User defined data types
- Object-oriented programming facilities
  - Overloading concept
  - Writing of polymorphic algorithms
  - Inheriting functionality from domains and categories.

Example of a gcd implementation in polymorphic style ...

```
category Cat::EuclideanDomain

gcd := proc(x, y) local l, i, g;
begin
    case args(0)
    of 2 do
        x:= dom::unitNormal(x);
        y:= dom::unitNormal(y);

        while not dom::iszero(y) do
            dom::rem(x, y); x:= y; y:= %2
        end_while;

        return( dom::unitNormal(x) )
    of 1 do
        return( dom::unitNormal(x) )
    of 0 do
        return( dom::zero )
    otherwise
        g:= dom::gcd(l[1], l[2]);
        for i from 3 to args(0) do
            g:= dom::gcd(g, l[i])
        end_for;
        return( g )
    end_case
end_proc;
```

A piece of code taken from the current developers version of MuPAD.

Domains

Domains are user-definable data structures and can be created by the MuPAD command `domain`. Almost every MuPAD function (and therefore each MuPAD operator) can be overloaded for a domain. We only can give you an idea of this concept, there is not enough time for an introduction to this topic.

Object-Oriented Programming Facilities

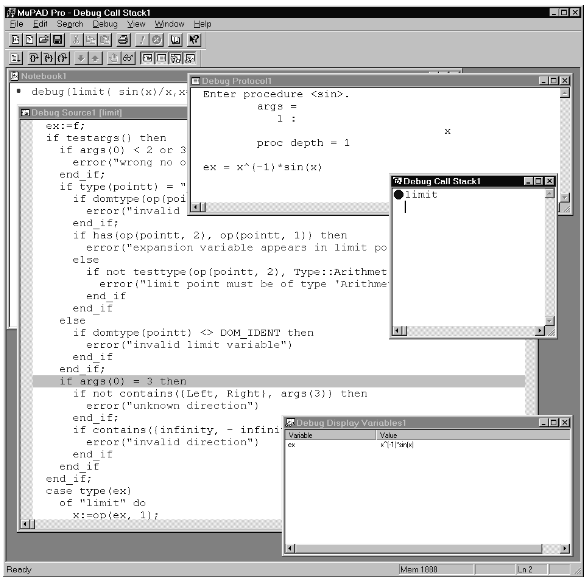
So-called *domains constructors* allow easy object-oriented programming. A domain created by a domain constructor can be derived from another domain. The domain then inherits all the methods from its father domain.
A domain can belong to certain *categories*. The domain inherits the methods of the categories, which are implemented in the domain.
Example: `Dom::SquareMatrix`, the *domain constructor* for n-square matrices over a certain coefficient domain R, is derived from the domain `Dom::Matrix(`R`)`.

As domains define the representation of their objects, categories and their methods are independent of a certain representation and therefore describe an algebraic property (e.g., a ring, an integral domain, ...) in an abstract way.
Example: The coefficient domain for matrices must be a commutative ring in order to define the matrix addition and multiplication in the usual way. The category of commutative rings is `Cat::CommutativeRing`.

MuPAD - The Open Computer Algebra System

Special Features

- ♦ Graphical source-code debugger
- ♦ „Dynamic Modules“



Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

IMACS/ACA
25 June 1999

Programming & Debugging

MuPAD offers the programmer several tool for testing and checking of MuPAD code, e.g. profiling, tracing and debugging. The MuPAD kernel provides a source code debugger which allows to step through procedures, print and observe values of variables, alter values of variables, execute instructions, display the procedure stack. We will demonstrate the debugger for the current developer version of MuPAD Pro for Windows 9x/NT.

Dynamic Modules

Besides the conventional concept of libraries written in the language of the CA system, MuPAD offers another library concept, called *dynamic modules*.

Software Integration in MuPAD

Exchange of data and commands between two or more software components.

Dynamic linking provides highest efficiency and the highest flexibility.

Distributed systems can be built up by integrating any existing IPC protocol.

Re-using special purpose software and adding new features efficiently

Dynamic Modules

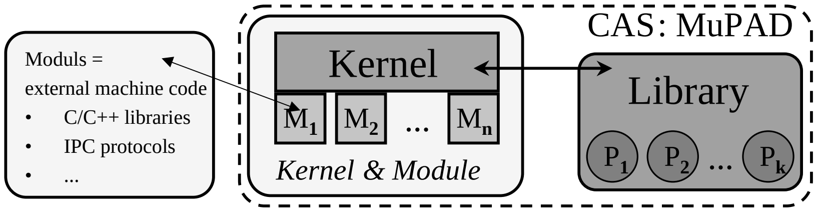
Another strength of MuPAD is a very flexible and efficient concept of **software integration** called **dynamic modules.**

For technical applications, but also in research, software integration in interactive general purpose computer algebra systems gets for and more relevancy.

As a base technology we use the concept of dynamic linking.

This also enables us to integrate other concepts of software integration based on network services and interprocess communication protocols on demand.

Concept of Dynamic Modules



- arbitrary dynamically linked kernel extensions
- consist of user code compiled from C/C++, Fortran, ...
- used in the same way as library packages (domains)
- can be unloaded at any time, are re-loaded on demand
- can interface any kind of (mathematical) software

The typical situation: A CAS kernel with **built-in functions** and a **library** that is usually **interpreted** by the kernel but may also consist of **pre-compiled** library functions.

MuPAD offers so-called module functions available with dynamic modules which can be **linked** into the MuPAD kernel **on demand**.

Module functions can be added and used as easy as library functions. They are fully integrated into the OOP language of MuPAD.

With this, MuPAD is a non-monolithic CAS.

Linking Dynamic Modules

- modules are linked at run-time, they are tightly coupled
- MuPAD shares internal routines and data structures
- high efficiency, suitable for very fine-grained interaction
- no problems with program control integration
- load-on-demand, unloading, resource management
- source code (or a dynamic link library) is needed

➜ C, C++, Fortran, Pascal,...

Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

+ control integration: no special treatment, just do not unlink dynamic libraries that are in use (other: head-in-the-sand strategy)
+ data exchange: exchanging references (e.g. C/C++ pointers), can be done in constant time, independent from size and amount of data
+ n^2-problem: for this we can integrate special tools like MP ...

Let me give a first trivial example on using, writing and compiling a dynamic module.

Using Dynamic Modules

```
$ mupad
     *----*    MuPAD 1.4.1  ---  Multi Processing Algebra Data Tool
    /|   /|
   *----* |    Copyright (c) 1992-98 by B. Fuchssteiner, Automath
   | *--|-*    University of Paderborn.  All rights reserved.
   |/   |/
   *----*      Licensed to: University of Paderborn
>> module(demo);                         # load the dynamic module #
                             demo
>> info(demo);                           # display information     #
 Module: 'demo' created on 05.Nov.98 by mmg R-1.4.1
 A fast sin function for MuPAD
 Interface:
 demo::sin, demo::doc                    # doc was created by mmg  #
>> demo::sin(3.14);                      # call a module function  #
                        0.001592652916
```

Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

MuPAD session: After loading the module, it can be used exactly like a MuPAD library package, e.g. ´**info**´ displays user information and

syntax of calling functions is as known from library packages

module functions can be exported like library function

The module computes the sin using machine floating point numbers.

Note, that MuPAD uses arbitrary precision arithmetic otherwise.

Writing Dynamic Modules

```
MFUNC( sin, MCnop )                 // Define a module function 'sin'
{                                   // with No OPtion (default case)
   double  cvar;                    // Declare your C/C++ variables
   MTcell  mvar;                    // Declare any MuPAD variable

   MFnargsCheck( 1 );               // Exactly one argument is needed
   MFargCheck( 1, MCnumber );       // The argument must be a number

   mvar= MFarg( 1 );                // Get first function argument
   cvar= MFdouble( mvar );          // Convert into C/C++ double
   cvar= sin( cvar );               // Compute machine float Sin
   mvar= MFdouble( cvar );          // Convert into MuPAD fp number

   MFreturn( mvar );                // Return the result to MuPAD
} MFEND
```

Writing of Modules is facilitated by *MAPI*,
the MuPAD Application Programming Interface

This is the **complete** user-defined **source** code of the dynamic modules demonstrated on the previous slide.

The **blue** keywords belong to **MAPI**, the MuPAD Application Programming Interface. Everything else is plain C/C++ code.

MFUNC introduces the definition of a module function. We check for one argument which must be a number, convert it into a C/C++ double, compute the sin and reconvert it into a MuPAD number. Then we return it to MuPAD.

As easy as writing module functions, is to compile the executable module...

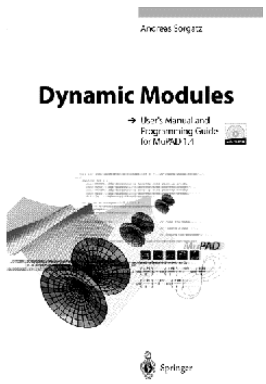
Application Programming Interface

MAPI is a lean C/C++ kernel interface for ...

defining module functions, supporting error and parameter handling and data conversion

constructing and manipulating expression trees, lists, sets, domains..., calling MuPAD built-in and library functions

writing hybrid algorithms, with direct use of the internal arbitrary precision arithmetic,... and more,...



Dynamic Modules -- User's Manual and Programming Guide for MuPAD 1.4, Andreas Sorgatz, October 1998, Springer Verlag Heidelberg · New York, includes a CD-ROM.

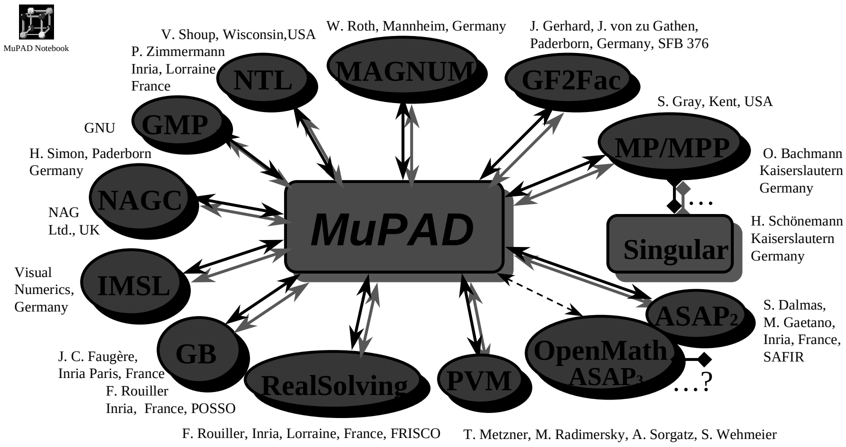
The MuPAD Application Programming Interface is published at Springer Verlag including examples and module applications. MAPI supports...

MuPAD - The Open Computer Algebra System

Applications of Dynamic Modules

http://www.mupad.de/PAPER/MODULES/

Connectivity to a broad field of mathematical applications



Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

IMACS/ACA
25 June 1999

Let me demonstrate some minor examples first...

With dynamic modules, the variety of different packages and algorithms which can be utilised simultaneously from within MuPAD increases drastically. In this sense dynamic modules apply the principle of software integration to the broad field of mathematical applications.

The following packages and systems have been linked to MuPAD using the concept of dynamic modules. Some of the projects are still prototypes.

Numerical Computing (NAG/IMSL)

```
$ mupad
     *----*     MuPAD 1.4.1  ---  Multi Processing Algebra Data Tool
    /|   /|
   *----* |     Copyright (c) 1992-98 by B. Fuchssteiner, Automath
   | *--|-*     University of Paderborn.  All rights reserved.
   |/   |/
   *----*       Licensed to: University of Paderborn
>> module(nag);
                    nag
>> nag::roots( poly(2*x^3 +5*x^2 +3) );
```

[-2.705001171 +0.00000000000000005551115123 I,

0.1025005859 -0.7375784977 I, 0.1025005859 +0.7375784977 I]

Here: to find approximations for all complex roots of a given polynomial.

Also: to build efficient solutions for so-called real-world-applications

This examples was demonstrated before.

Combined symbolic and numerical computations for real-world-applications.

This can be done with only a few lines of C/C++ source code...

ISSAC´97 Challenge - Problem 6

Compute the following Gröbner Basis in LexOrder:

```
>> p1:= poly( 8*w^2 +5*w*x -4*w*y +2*w*z +3*w +5*x^2 +2*x*y
   -7*x*z -7*x +7*y^2 -8*y*z -7*y +7*z^2 -8*z +8, [w,x,y,z]):
>> p2:= poly( 3*w^2 -5*w*x -3*w*y -6*w*z +9*w +4*x^2 +2*x*y
   -2*x*z +7*x +9*y^2 +6*y*z +5*y +7*z^2 +7*z +5, [w,x,y,z]):
>> p3:= poly( -2*w^2 +9*w*x +9*w*y -7*w*z -4*w +8*x^2 +9*x*y
   -3*x*z +8*x +6*y^2 -7*y*z +4*y-6*z^2 +8*z +2, [w,x,y,z]):
>> p4:= poly( 7*w^2 +5*w*x +3*w*y -5*w*z -5*w +2*x^2 +9*x*y
   -7*x*z +4*x -4*y^2 -5*y*z +6*y -4*z^2 -9*z +2, [w,x,y,z]):
```

This approach solves the problem in less then 10s on a Laptop (Pentium 70):

```
>> loadlib( "sing" ):
>> gb:= sing::stdfglm( Ideal(p1,p2,p3,p4) ):
```

Behind the curtain:
module(mp); **mp**::launch(**Singular**); mp::send(question); mp::receive(answer);

For the ISSAC´97 Challenge they gave us the following problem...

At that time, this problem was very hard to solve for general purpose CAS .

But this approach, ...

This was much faster than any other participant of the challenge could do it.

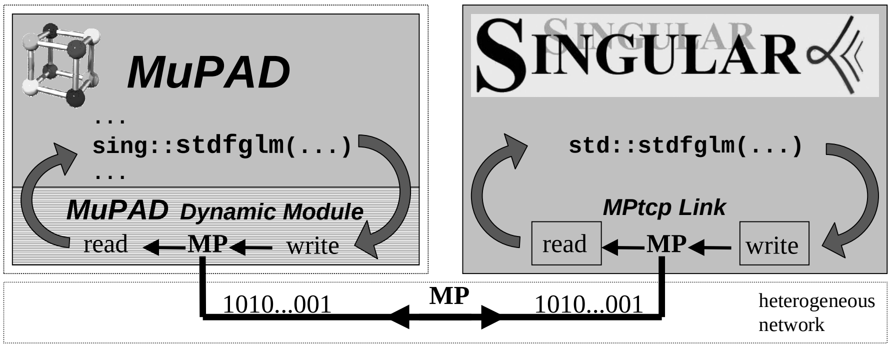
The output is not printed because it needs about than 7-10 pages.

From the users point of view, we just loaded and used a library package...

Behind the curtain we used a dynamic module **mp** which encapsulates the **MP** protocol, which is similar to **OpenMath**. With this, we started the **Singular** system as a back end computing server.

MuPAD - The Open Computer Algebra System

Using MP and Singular



The Multi Protocol (**MP**) supports efficient communication of mathematical data among scientific computing systems with shared knowledge of syntax and semantics of the data exchanged. In this sense it compares to **OpenMath**.

Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

IMACS/ACA
25 June 1999

This slides shows how the MuPAD Singular connection is realised.

Singular is a special purpose system for comutative algebra having very **fast Groebner basis** algorithms. It supports network communication using the **MP** protocol as an built-infeature.

Because MuPAD did not provide MP links, we implemented a dynamic **module** which **uses the MP library** to add this feature to MuPAD. This was done without touching the MuPAD kernel itself in any sense.

When the command **sing::stdfglm** is started in MuPAD, this is automatically transformed into a (binary encoded) MP syntax and semantics tree by the dynamic module **mp**. This is send over the network to Singular. Singular reads the command, computes the solution and sends it, encoded as an MP tree, to MuPAD. The dynamic module **mp** reads the result, converts it into a MuPAD representation and returns it to the user. All this is transparent to the user.

The Multi Protocol...

While this a rather **academic application**, useful for research, the next example demonstrates a **technical appication**.

Connecting MuPAD and Singular via MP, which was made in tight cooperation with **Olaf Bachmann** and Hans **Schönemann** from the University of Kaiserslautern. **MapleTech article**.

MuPAD - The Open Computer Algebra System

Using GB / RealSolving

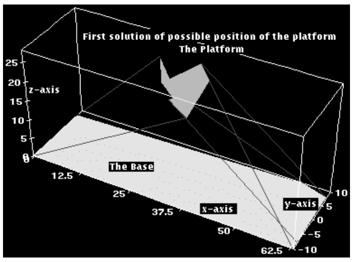
http://www.loria.fr/~rouillie/binaries.html

GB = Solving Gröbner basis (in this sense it compares to **Singular**)
RealSolving = Computing information about real roots of polynomials

Robot: a base (A1,..,A6), a platform (B1,..,B6), arm lengths (l1,..,l6)
Problem: find all possible positions for the platform so that AiBi=li.

Computing all solutions using

- **MuPAD: formula transformations**
- **GB: computing Gröbner basis**
- **RealSolving: isolating real roots**
- **MuPAD: collecting intermediate results and displaying the solution.**



Ralf Hillebrand, Frank Postel
MuPAD Research Group, University of Paderborn

IMACS/ACA
25 June 1999

Fabrice Rouillier from INRIA Lorraine has connected GB and RealSolving to MuPAD. The corresponding dynamic modules are available on his home page (see above). He uses MuPAD/GB/RealSolving for problems like...

Robot, Problem...

I do not want to describe the mathematical solution but just want to give an idea how this is realised very efficiently...

As my last example of an application of dynamic modules, I want to introduce the so-called **Macro Parallelism** in MuPAD.

Dynamic Modules provide...

- Flexibility and Efficiency
  - kernel extensions: C/C++ including Fortran, ...
  - suitable for fine-grained interaction (down to single FPOs)
- Small Modular Kernel
  - extensible and freely configurable
  - dynamic module development by special interest groups
- Universal Mathematical Shell
  - interacting with special purpose tools
- Framework to...
  - dynamically link external components directly
  - dynamically link arbitrary protocols and docking components,
    e.g.: **MP**, OpenMath, CORBA,..., **PVM**,..., pipes, sockets, DDE, ...

To sum it up. Dynamic Modules provide...

Efficient: nearly no communication overhead

Special interest groups: GB/RealSolving.

Universal Shell: MuPAD can be used as a flexible interface for symbolic and numerical computations integrating numeric and symbolic libraries as well as special purpose systems.

This is a first step towards building **problem solving environments** for **real-world-applications** in MuPAD.

* However, the concept of dynamic modules is no replacement for protocols like MP, OpenMath, PVM and others, but extends the flexibility and increases the efficiency of interactive general purpose computer algebra systems drastically.