NPL Report CISE 43/00

Report to the National Measurement System Policy Unit, Department of Trade and Industry

From the Software Support for Metrology Programme

Design of the Metrology Software Environment (METROS)

By
Robin Barker, NPL &
Geoff Morgan, NAG Ltd

February 2000

Design of the Metrology Software Environment (METROS)

Robin Barker, NPL & Geoff Morgan, NAG Ltd

February 2000

ABSTRACT

The report describes the structure of METROS and how it will be used by the user metrologist. METROS is a web-based library of re-usable software, giving metrologists and software developers access to solutions of problems common to many metrology areas. These solutions are specifications of algorithms, implementations of those algorithms, and guidance on their use.

This report is a deliverable from the metrology software environment design work package of the *Software Reuse* Project of the NMS Software Support for Metrology programme. This report was originally delivered by NAG Ltd to the NMSPU in July 1999.

ISSN 1361–407X

Authorised by Dr Dave Rayner,
Head of the Centre for Information Systems Engineering

National Physical Laboratory,
Queens Road, Teddington, Middlesex, United Kingdom. TW11 OLW

Contents

1 Introduction

The report describes the structure of METROS (the METROlogy Software environment) and how it will be used by the user metrologist. First we describe what problems the METROS design is intended solve, and set out the top level structure of METROS. Then we define the major elements of the structure and the other elements. The last two sections say how the METROS design will be used and draw the conclusion from the initial work on METROS.

The design of METROS will evolve as the system is used. As functions and implementations to METROSare added, lessons will be learned which will modify the way METROS is structured and administered.

2 Requirements

2.1 Aims

The METROS system will allow metrologists to gain access to the software they require for their work. The central idea is that of the *key function*. This is either a function that may be used in many different applications or a function that is seen as being fundamental to an area in metrology. A list of key functions is given in NPL Report CISE 44/00: *"Specification of the* METROS *key functions"*. This list will represent a initial attempt to cover the fundamental computational requirements of metrologists. Most metrologists may not be familiar with these functions but will view their problem from an application perspective. The METROS system will therefore contain applications based on key functions and links between these applications and the key functions. Further, the metrologist will be working in different computing environments and platforms. They will require the appropriate software for their platform. Finally, they will benefit from the support of appropriate case studies and best practice guides which will also be contained in the METROS system.

2.2 METROS community

The METROS system is viewed as collaborative venture with different sections contributing and using the system. For convenience, the METROS community can be seen to involve four groups on a functional basis.

(a) End-users

(b) Contributors

(c) Library and system providers

(d) Co-ordinators

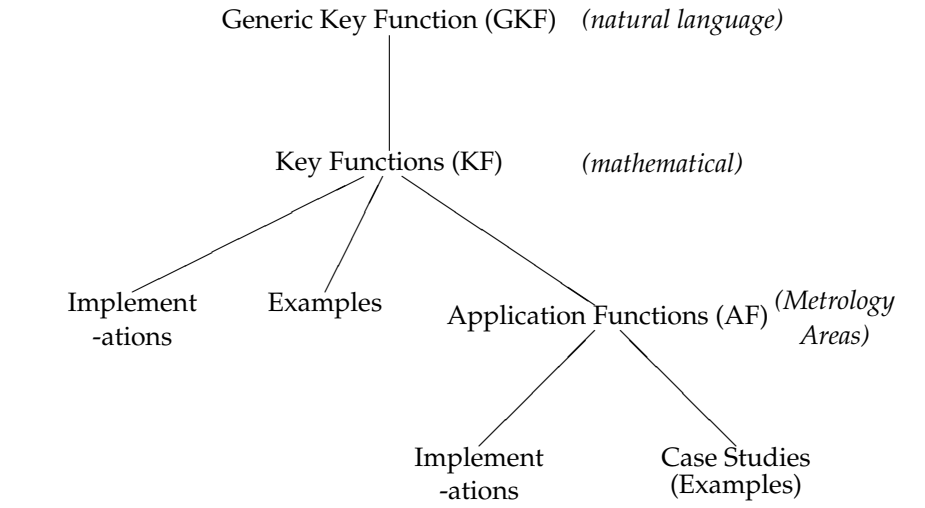

Figure 1: METROS structure

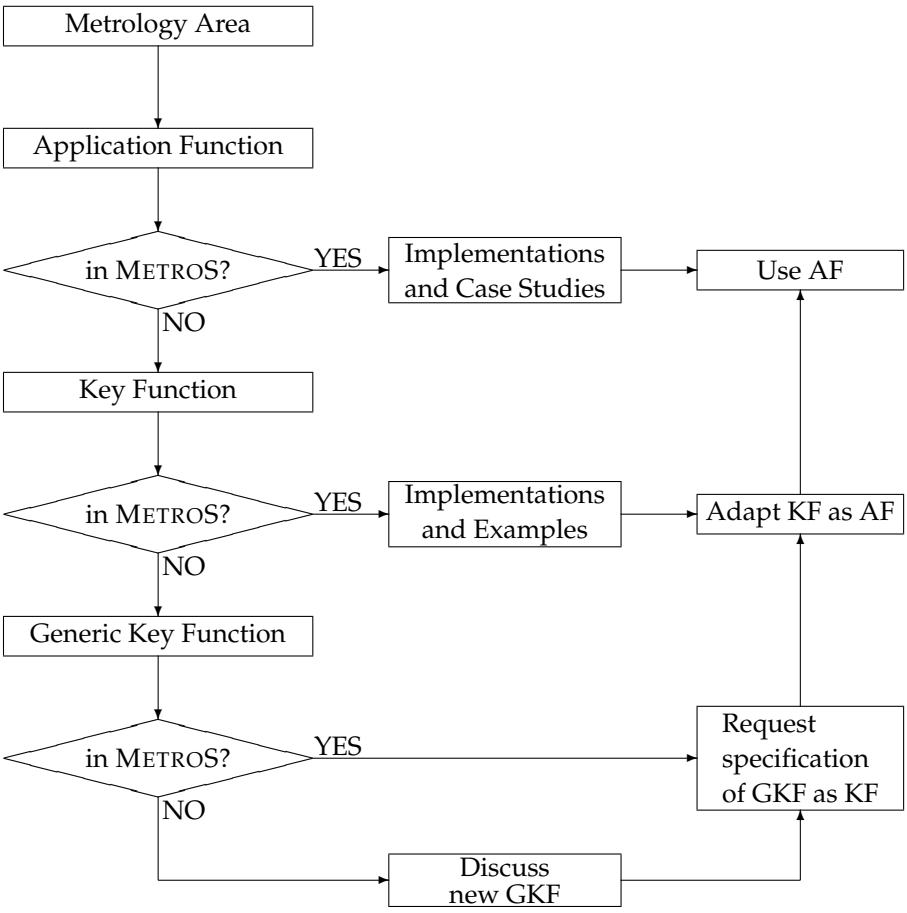

Figure 2: Metrologist view

It is likely, in practice, that individuals and organisations will be represented in more than one group.

The characteristics and requirements of each of the groups will be considered.

(a) End-users

An end-user will be anyone who uses the system to provide them with their required software. They will fall broadly into two classes:

(i) Metrologist End-users who need the software for their own use or to be used by their customers.

(ii) Contractors to end-users, that is computer programmers who have been given the task to create the software required by the metrologist.

The main difference between the two groups will be their position in the spectrum of skills that users are expected to have. The main skills are:

(i) Knowledge of the application area

(ii) Knowledge of the application system

(iii) Knowledge of mathematics/statistics

(iv) Programming skills

Those with sufficient programming and application system skills will be able to adapt functions available in one language/system to another. Those with sufficient mathematical knowledge will naturally relate the problem to the key functions. The system will be useful to such people as a repository of available functions. However, for those that do not have such skills at a sufficient level the system needs to be able to:

(i) Provide an entry point where they have sufficient knowledge, usually the application area.

(ii) Enable them to make the link between their application problem and the required key functions.

(iii) Provide the key function in as convenient a way as possible.

Clearly complete coverage will not be possible, and the project itself will only provide a starting point for further development. However, in so far as is possible given the current state of the METROS System, an end-user should be able to:

(i) Find the most appropriate software (this may not be a perfect match in terms of its adaption to the application area/computing environment).

(ii) Implement software to solve their actual problem.

The closer the match to the application area/computing environment the easier the implementation of the software will be. It is important that the end-user is pointed to the correct functions even if these are not in the most convenient form for the end-user.

In summary the key requirements of the end-user will be:

(i) Ease of use.

(ii) Providing the software they would like.

(iii) Reliability in acquiring the correct software.

(b) Contributors

Contributors will provide components to be included in the METROS system. These may be key functions, applications, interfaces to specific systems, best practice guides etc.

Contributors will be experts in their areas. They may wish to contribute to METROS for a variety of motivations depending on their circumstances. For those working in academic and government organisations it would part of the dissemination of their work. For those from commercial organisations it could be to gain publicity and promote the use of their software.

Two types of contributions are expected, *validated* and *unsupported*. In the case of validated contributions, the contribution will have to be checked by the co-ordinators of METROS. For unsupported contributions, there would only be minimal checks.

The main requirement of contributors will be the availability of clear specifications for the inclusion of contributions in METROS. They may also like flexibility so that what they have already can be included in METROS with minimal change. This may lead to conflict between the need for clear specification and the needs of the co-ordinators of METROS on the one hand and contributors desire for flexibility on the other. Therefore there will be a requirement for a clear adjudication scheme with the final decision.

(c) Library and system providers

Much of the underlying software required by the key functions is likely to be from Library and system providers. The providers may be commercial or non-commercial (public domain software). In the case of commercial suppliers the users would have to pay a licence fee for the use of the software but would have the possibility of support. The providers of public domain software would not charge a fee but also would not provide any support. It is likely that the use of public domain software will come through contributions to METROS as described above.

For a commercial software house the METROS system provides an opportunity for them to display their software to a specialist audience. Their requirements will be that it is suitably referenced and that there is clear information on licencing and contact with the organisation.

The greater the success of METROS the more likely software houses are to contribute.

(d) Co-ordinators

The Co-ordinators of METROS are providing a service to the metrology community. Their aims will be to increase the awareness of the tools that are available and how they should be used. They will wish for the maximum involvement in METROS by contributors and users. Thus their main requirement is that the needs of these groups are met. In addition, they will require METROS to run without consuming too much resource. Therefore they is a need for a robust structure that is easy to maintain and a clear straightforward procedure for inclusion of contributions in the system.

2.3 Summary

The essential requirements of the four groups within the METROS community can be summarised as:

1. Ease of use for end-users
2. Good coverage and reliability in providing the correct software
3. Clear procedures for inclusion of contributions
4. A robust system that is easy to update
5. Links to software providers

The success in meeting these requirements will be assessed by a number of small case studies within the evaluation of the system.

3 Structure of METROS

3.1 METROS design

The Metrology Software environment (METROS) is primarily a means of accessing information about key functions which form the Metrology Software library. In some cases, it will also be possible to access software which implements the key functions, in various formats. To design an information access system, we shall think in terms of a relational database; at the moment this simply gives us a convenient language for speaking about the METROS design, and is not a decision about how METROSwill be realised. It is intended that METROS will be implemented with a web interface accessible from the internet.

The components will be described in tables, with each major component having its own table. The components will be defined by the other fields of the table. Figure 1 on page 2 shows the basic structure of the components.

3.2 Components

- Metrology Area (MA)

  This will be the metrologists usual entry point into METROS. Metrology areas will follow the existing NMS classification, with further refinements being added as dictated by the needs of metrologist end-users.

- Case Studies (CS)

  Case studies of the use of particular functions from METROS will be an important way of showing metrologist end-users how functions can be used and re-used. Case studies will take the form of technical reports from the metrologist community (with suitable editing from the METROS Co-ordinators, if required).

- Generic Key Functions (GKF)

  The generic key functions are the simplest functional components of METROS. A generic key function is "a problem without a solution": it defines the purpose of a computation without specifying a mathematical technique for finding a solution.

  They consist of a natural language description of a function, with (perhaps) some of the parameters that such a function would use. Generic key functions will be used to group key functions with the same purpose.

- Key Functions (KF)

  These are the central functional components of METROS. They consist of a mathematical specification of calculation which is important for metrologists; without specifying how that calculation should be implemented. Key functions are specifications of generic key functions and will have at least those parameters defined in the generic key functions.

- Application Functions (AF)

  These are the functional components of METROS which a user is most likely to use directly, consisting of a key function (or combination of key functions) adapted to a particular metrology area. The application functions will have a similar structure to a key function specification.

- Implementations of Key Functions (IKF)

  The key function implementation components of METROS will contain information on available implementations of key functions. These include both source code implementations, and executable code implementation for a particular platform (computing environment). The key function implementation components will point to the key function which it implements; and will contain documentation for the function, including additional parameters (beyond those specified for the key function), information on accuracy and valid parameter ranges, and the meaning of error codes.

- Implementations of Application Functions (IAF)

  The application function implementation components of METROS will contain information on available implementations of application functions. They will be the useful functions for metrologists in a particular metrology area, whether or not that functions has been identified as an existing functions within METROS. They will be similar in structure to key function implementation components.

- Key Function Headers (KFH)

  The key function headers will provide a common interface for implementations of a key function in a particular language. Key function implementations of the same key function which conform to the same key function header will be able to used interchangeably.

  These will need to be simple to provide a solution to support multi-language programming from METROS.

The top level relationship between the components is shown in the following table. There are entries in the table where the component of the row refers directly to the component of the column. The entries show how many (0, 1 or "many") "column" components one "row" component may refers to. Empty rows and columns have been omitted.

Refers to	MA	GKF	KF	AF	KFH	IKF	IAF	CS
MA								
GKF								
KF		1						
AF	1		n					
KFH		1						
IKF			1		?			
IAF				n	?	*		
CS	n		n	*		*	*	*

Key: "1" precisely one, "n" many (at least 1), "*" any (may be zero), "?" 0 or 1.

3.3 Sub-components of implementations

Implementations (IKF and IAF) will be specified in terms of sub-components, which we describe here.

- Library/Core Functions (LC)

  Implementations will use routines from existing software libraries. Such libraries will have to be available to the end-user in order to use the implementation of the key/application function.

- Platforms (PL)

  Implementations may be built (compiled) for a particular platform (i.e. hardware architecture and operating system). It is important that an implementation of the key/application function identifies the platform or platforms on which it will run.

- Computing environment (CE)

  Implementations may exist as functions in a particular computing environment (i.e. a software package for software development), for instance Excel, MATLAB or LabView; or they may exist to be called from the environment. The implementation of the key/application function will identify its computing environment.

- Languages (L)

  Implementations will be written to be called from a particular programming language. They may also be provided as source code in that language, in which case the user may be able to create an implementation for his own platform/computing environment by compiling the source code.

There are different sorts of key/application function implementation with different sub-components.

1. Source code implementations need not refer to a PL/CE but a binary executable implementation will specify one (or more) PL/CE.
2. An implementation of an application function may be defined directly in terms of library/core functions (LC) or it may be defined in terms of one or more key function implementations (IKF) and not use library/core functions.
3. An executable may be produced by compiling a platform-independent source-code implementation or may be constructed directly in the programming environment.

When describing an implementation for inclusion in METROS we need to consider whether functions are general or specific to a metrology area; we also need to consider whether functions, implemented in a particular language, are nevertheless platform-independent or specific to a particular computing environment.

3.4 Examples of METROS components

For the key function of *circle fitting* we give examples of the components which are relevant to this key function.

MA Length: dimensional metrology. Electrical: network analysis.

CS Examples in the SS*f*M modelling project best practice guide.

GKF "a function to fit a circle to data".

KF circle fitting by least squares.

AF roundness functions, network analysers.

IKF Implementations of the key function in Fortran77 and Matlab.

IAF Implementations of the application functions in CMM software and Network Analyser software.

Example of sub-components of implementations are

LC Libraries: NAG, BLAS, MINPACK.

Functions: non-linear least squares, linear regression, minimax, robust methods.

PL PC(MSDOS and Windows 95), Unix, CMM

CE Matlab, MS Excel

L Fortran77 and Matlab

3.5 Use of classification schemes

METROS contains one classification scheme from the outset: the metrology areas. There are a number of other commonly used classification schemes for science, computation and mathematics. Components can be labelled against one or more of these other schemes and other views of the components can be obtained by grouping components with a common classification.

Classification schemes which could be used are:

- AMS (American Mathematical Society): mathematical;
- GAMS (from NIST): computational;
- NAG library classification: computational;
- PACS: physical.

3.6 Key function suites

A number of key functions may provide the solution to the problem defined in a generic key function. A key function suite consists of a generic key function and a number of key functions which provide a solution: within the key function suite, the key functions are indexed by the (mathematical) method used to provide the solution.

4 Specification of METROS Components

The following specifies the components of METROS as given in section 3.2. Except where indicated all items are mandatory, but a null entry may be used if appropriate. For example, in the 'library/core functions used' of the Implementation of Application Function specification there may be no underlying library functions.

4.1 Metrology Area

This component has one field, which is the NMS Metrology Area: Acoustics, Electrical, Foundational, Ionising Radiation, Length, Mass, Photonics, Optical Radiation, Thermal, Time and Frequency, VAM.

If required the metrological area could be linked to other subject classification schemes, e.g., American Institute of Physics, American Mathematical Society.

4.2 Case studies

4.2.1 Fields

- metrology area
- who
- where
- best practice guides used or contributed to
- platform
- language(s)
- key function(s)
- application functions, implementations, executables.
- references/bibliography

4.2.2 Example

Circle fitting case study, of the kind to be delivered by other SSfM projects.

4.3 Generic Key Function

4.3.1 Fields

- name
- purpose/aim — problem being solved
- inputs — names, types and meaning (Optional)
- outputs — names, types and meaning (Optional)

Note: the inputs and outputs are optional since different parameterisations for the same function may change the required inputs and outputs.

4.3.2 Example

Circle fitting

Name circle fitting

Purpose to fit a circle to a set of points

Inputs (x_i, y_i): data points

Outputs $(x, y), r$: parameters specifying a circle

4.4 Key functions

4.4.1 Fields

- name
- purpose/aim — problem being solved
- method — name of mathematical technique to be used to implement the computational aim
- corresponding generic key function
- inputs — names, types and meaning
- output — names, types and meaning
- computational aim: relationship between i/o and constraints on i/o; and documentation.
- validation criteria

We can see the computational aim as pre-conditions and post-conditions: the implementations of the key functions will define errors and warnings when various conditions fail.

4.4.2 Example

Circle fitting using non-linear least-squares

Name circle fit: least-squares

Purpose to fit a circle to data

Method non-linear least-squares

Generic circle fit

Inputs

Name	Type	Description
$(x_i, y_i)_{i=1}^m$	Double: array (m) of pairs	data points
$(w_i)_{i=1}^m$	Double: array (m)	(OPTIONAL) weights

Outputs

Name	Type	Description
$(x, y), r$	Double: pair, scalar	parameters specifying a circle
$(d_i)_{i=1}^n$	Double: array (m)	distances of points to the circle
Error	*	Indicates presence/nature of error

Computational Aim $\{(x, y), r\}$ minimises $\sum_{i=1}^m w_i^2 d_i^2$,

where d_i = distance of (x_i, y_i) from the circle defined by $\{(x, y), r\}$

Validation Criteria Quantified behaviour or reference data sets.

4.5 Application function

4.5.1 Fields

- name
- metrological area
- purpose/aim
- method
- inputs
- outputs
- computational aim: relationship between input/output (i/o) and constraints on i/o.
- list of KF which may be used to implement function
- validation criteria

4.5.2 Example

Reflection co-efficient calculation for sliding load: using circle fitting

Name Sliding load zero reflection co-efficient

Metrological area Electrical: network analysis

Purpose To calculate the matched load with zero reflection coefficient, from the reflection coefficients when a sliding load is attached to a reflectometer and an electromagnetic wave is passed along the load.

Method circle fitting

Inputs

Name	Type	Description
$(z_i)_{i=1}^m$	Complex: array (m)	reflection coefficients

Outputs

Name	Type	Description
z	Complex	zero reflection coefficient
r	Double	(OPTIONAL) radius of circle
Error	*	Indicates presence/nature of error

Computational aim $\{z, r\}$ minimises $\sum_{i=1}^m d_i^2$,

where

$$\begin{aligned} d_i &= \text{distance of } z_i \text{ from the circle } \mathcal{C} \text{ in the complex plane} \\ \mathcal{C} &= \{w : ||w - z|| = r\} \end{aligned}$$

Validation Criteria reference data sets

Key functions circle fit: non-linear least squares

4.6 Key function header

4.6.1 Fields

- name
- key function
- language
- signature
  - inputs: concrete types for KF inputs and any additional inputs
  - outputs: concrete types for KF outputs and any additional outputs
  - meaning/relationship/constraints on additional parameters
- Pointer to the actual header file or code snippet (Optional)

4.6.2 Example

Circle fitting in MATLAB

Name circle fit header

Key function circle fit: least-squares

Language MATLAB

Signature

Inputs x, y: vector (coordinates, (x_i, y_i))

Outputs

x0: vector (circle centre, (x, y))

r: scalar (circle radius, r)

d: vector (radial distances, (d_i))

ss: scalar (sum of squares of distances, $= \sum d_i^2$)

conv: scalar (convergence, i.e. *Error*)

conv	Meaning
1	method has converged
0	method has not converged
-1	method has diverged

4.7 Implementation of key function

4.7.1 Fields

- name
- key function
- language
- key function header *OR* signature

- strategy/algorithm/high-level design
- test data
- test results
- sample program
- validation status
- library/core functions used
- target platform or computing environment

The documentation of the additional parameters will include parameters used for exception and error handling.

4.7.2 Example

Circle fitting in MATLAB using Gauss-Newton

Name `circle2`

Key function circle fit: least-squares

Language MATLAB

Header circle fit header

Strategy Gauss-Newton

Test data An example of one set of test data is shown over.

```
cir1 =
    1.267772000000000e+002    2.194740000000000e+001
    1.217707000000000e+002    5.365630000000000e+001
    1.072036000000000e+002    8.226700000000000e+001
    8.450040000000000e+001    1.049702000000000e+002
    5.588970000000000e+001    1.195373000000000e+002
    2.418080000000000e+001    1.245438000000000e+002
   -7.518400000000000e+000    1.195074000000000e+002
   -3.610890000000000e+001    1.049290000000000e+002
   -5.880090000000000e+001    8.223710000000000e+001
   -7.337920000000000e+001    5.364650000000000e+001
   -7.841560000000000e+001    2.194740000000000e+001
   -7.340910000000000e+001   -9.761500000000000e+000
   -5.884200000000000e+001   -3.837220000000000e+001
   -3.613880000000000e+001   -6.107540000000000e+001
   -7.528100000000000e+000   -7.564250000000000e+001
    2.418080000000000e+001   -8.064900000000000e+001
    5.587990000000000e+001   -7.561260000000000e+001
    8.447050000000000e+001   -6.103430000000000e+001
    1.071624000000000e+002   -3.834230000000000e+001
    1.217408000000000e+002   -9.751799999999999e+000
x0n =
    2.418080019810833e+001
    2.194740019810833e+001
r0n =
    1.025963876201454e+002
d =
    1.218174621442358e-005
    1.571142275328441e-002
    2.543952790989579e-002
    2.543952790989579e-002
    1.571142275327020e-002
    1.218174622863444e-005
   -1.572250948800047e-002
   -2.546651840678749e-002
   -2.538552905267011e-002
   -1.575315678682898e-002
    1.257796296272318e-005
    1.571192201555505e-002
    2.544008134634623e-002
    2.544008134634623e-002
    1.571192201555505e-002
    1.257796296272318e-005
   -1.575315678681477e-002
   -2.538552905267011e-002
   -2.546651840677328e-002
   -1.572250948798626e-002
ss =
    7.152817645078665e-003
```

Test results Passed

Sample program *would need to be provided*

Validation status tested by NPL

Library/core functions `als2dcircle`, `ls2dcircle`

Platform MATLAB

4.8 Implementation of application function

4.8.1 Fields

An IAF may be an existing example or it may be derived/synthesised from the definition of an application function as a wrapper round a key function and a key function implementations.

- name
- application function
- language
- key function header *OR* signature
- strategy/algorithm/high-level design
- test data
- test results
- sample program
- validation status
- target platform or computing environment
- library/core functions used
- IKFs used

4.8.2 Example

Implementation of reflection coefficients calculations in Fortran.

Name REFCOEFF

Application function Sliding load zero reflection co-efficient

Language Fortran (DEC F90)

Signature

Inputs

`INTEGER N`	number of coefficients, `N` ≤ 100
`COMPLEX Z(1:100)`	reflection coefficients, z_i

Outputs

`COMPLEX Z0`	circle centre, z
`INTEGER ICONV`	convergence, i.e. *Error*

`ICONV` has same meaning as `conv` above.

Strategy using non-linear least-squares circle fitting

Test data (as test data for circle fit)

Test result Passed

Validation status validated by NAG

Platform VMS

Library/core functions NAG Fortran 77 Library

IKFs used circle fit in Fortran

5 The way forward

5.1 Co-ordination of METROS

Over time METROS will grow and the structure of METROS will evolve: we need rules for how components are added to METROS.

There are different rules depending on the component.

1. New generic key functions could be proposed and discussed by anyone.
2. Key functions can only be added and changed by the parties who are co-ordinating METROS.
3. For implementations, these can be added freely (subject to some commercial considerations) but their status will be reflected in some "status" field, for example, untested, tested and validated.
4. Examples and case studies will be encouraged by the co-ordinators. They will be added after having been subject to some technical review (perhaps, by the co-ordinators themselves). Other METROS users can contribute comments on the usefulness of examples and case studies: these comments will be attached to the examples.

5.2 Generic key function headers

The key function header is obviously related to the input and output parameters of the key function, therefore to the parameters of the generic key function. If the key function implementations are written in a suitably stylised way, the key function header should be automatically derivable from the description of the generic key function parameters.

This would allow for the development of a "generic key function header" which could be used to generate key function headers for any language. The generic key function header could also be used to produce the code which

implemented a key function in one language as a "wrapper" round a key function implementation in another language. It would then be possible to automatically generate multi-language key function libraries.

These ideas may be taken up in the report on multi-language programming but may prove infeasible. The aim in this present work package is to ensure that the design of METROS allows for these possibilities.

5.3 Flexibility — METROS can cope

It is expected that METROS will not consist only of functions which fit neatly in the structure: some executable application function implementations may be built directly from library/core functions with no real use of (say) platform-independent key function implementations. However the structure can cope with many different sorts of components. The way forward is to make the different components visible, so that other METROS users can develop (and generalise) a particular component to other platforms and application areas.

6 Summary and Conclusions

The contents of this report need to be considered alongside NPL Report CISE 44/00 *"Specification of the* METROS *key functions"*, that describes the key functions that are targeted to be included in the METROS system. This report has presented the design of the environment that will provide access to these key functions and their implementations along with the related applications, case studies and best practice guides. These components will be brought together into a single system and provided with a Web based interface to allow for easy access. The key fields in a search of the system will include: metrology area, key function and application.

The design of the system should ensure that, when complete, the important computing functionality that metrologists require is readily accessible. The design also permits an incremental approach towards that state in which the user can be directed towards suitable, available components.

The adequacy and efficiency of the design will be assessed during the next Work Package in which a prototype system will be built and tested. Feedback from that exercise will be used to improve the specification described in this report. This will be provided as an appendix to *"Report on implementation and acceptance testing of key functions"* and will be reflected in the Software Support for Metrology Best Practice Guide No. 5 *"Software Re-use Guide to* METROS*"*, currently a "Draft for comment".

A Appendix – Acronyms

The following acronyms are defined in Section 3 and used within the Report.

AF	Application Functions
CE	Computing environment
CS	Case Studies
GKF	Generic Key Functions
IAF	Implementations of Application Functions
IKF	Implementations of Key Functions
KF	Key Functions
KFH	Key Function Headers
L	Languages
LC	Library/Core Functions
MA	Metrology Area
PL	Platforms