Java-based graphical user interface for MRUI, a software package for quantitation of in vivo/medical magnetic resonance spectroscopy signals

A. Naressia, C. Couturierb, I. Castangb,1, R. de Beerb,∗, D. Graveron-Demillyc

aFacultät für Physik und Geowissenschaften, Universität Leipzig, Linnéstrasse 5, Leipzig, Deutschland
bDepartment of Applied Physics, University of Technology Delft, P.O. Box 5046, 2600 GA Delft, Netherlands
cLaboratoire de RMN, CNRS UMR 5012, Université Claude Bernard Lyon I-CPE, Boulevard du 11 Novembre 1918, 69622 Villeurbanne Cédex, France

Received 18 July 2000; accepted 18 January 2001

Abstract

This article describes a Java-based graphical user interface for the magnetic resonance user interface (MRUI) quantitation package. This package allows MR spectroscopists to easily perform time-domain analysis of in vivo/medical MR spectroscopy data. We have found that the Java programming language is very well suited for developing highly interactive graphical software applications such as the MRUI system. We also have established that MR quantitation algorithms, programmed in the past in other languages, can easily be embedded into the Java-based MRUI by using the Java native interface (JNI).

© 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Java; Graphical user interface (GUI); In vivo/medical magnetic resonance spectroscopy (MRS); Quantitation; MRUI package

1. Introduction

1.1. A GUI for the MRUI package

The home page of the magnetic resonance user interface (MRUI) package [1] defines MRUI as “a graphical user interface that allows MR spectroscopists to easily perform time-domain analysis of in vivo MR data”. Starting around 1985 [2] we have shown that quantitating the nuclear magnetic resonance (NMR) signals, coming from “living objects”, by means of model function fitting in the
measurement domain of the experiment can considerably improve the quantitation results. This is particularly true if prior knowledge on the signals can be exploited in the model functions used (see also [3,4]). Also we have established that various forms of processing of the NMR signals, prior to the quantitation, may give another gain in the ultimate results (see for instance [5]).

About a decade ago we felt the need of integrating the various processing and quantitation algorithms, present at that time in the field of MR time-domain data analysis, into one software package. This package ultimately was given the name MRUI. In the first version of the official MRUI manual, written in the context of the EU Human Capital & Mobility/Networks project “Advanced Signal Processing for Medical Magnetic Resonance Imaging and Spectroscopy” (number CHRX-CT94-0432) [6], it can be seen that the graphical user interface (GUI) of the MRUI software runs in the MATLAB [7] environment. In fact, the GUI part is written as a set of MATLAB M-scripts and functions. This requires MATLAB version 4.0 or higher.

The European Union project, currently funding the MRUI development, is part of the Training and Mobility of Researchers/Networks (TMR) programme (project number ERB-FMRX-CT97-0160 [8]). At present there are more than 200 registered MRUI users [1], many of them working in medical environments such as university hospitals. In the mean time we have had various feedbacks that the need to install a commercial software environment such as MATLAB, including a license with a yearly payment, was felt by many user groups as a rather annoying burden. This must be seen in the light that the MRUI package itself is not commercial. Therefore, in the context of the just mentioned TMR project, it was decided to undo the MATLAB software requirement. To that end a new GUI has been designed, written completely in the Java programming language [9–14]. The required Java development software, called the Java development kit (JDK) can be downloaded for free from Sun Microsystems [15].

In this article we describe the essential components of the GUI. All functionalities, present in the MATLAB-based MRUI, have now been incorporated into the new Java-based MRUI. In addition we have added some new functionalities, particularly on performing quantitation of “series” of MRS signals and on presenting quantitation results in a form directly suited for the Internet. At present about 6 person years were needed in order to develop and generate the new MRUI GUI. The persons, involved, were researchers working in the context of the TMR project [8].

1.2. Why Java?

Since Java is designed to be object oriented from the ground up, writing software applications in Java means that one can benefit by the concepts of object technology [16–20]. Apart from this important general aspect there are some specific reasons why Java is our programming language of choice:

- Java source code is compiled into Java bytecode, which is claimed to be platform independent. Hence, Java programs can run on any platform with a Java Virtual Machine (the Java interpreter and run-time system [9]).
- The standard Java library contains the Swing package [13,14], which is a complete collection of GUI elements.
- Java supports multithreading [21], which means that parts of a computer program can be executed in separate threads. Each thread executes its code independently of the other threads in the
program. This offers the opportunity of enhancing the performance of programs, especially in case of interactive graphical applications.

- Java supports the Java native interface (JNI) [22], which enables users to make calls to native code.

Examples in the literature of other projects, using Java for scientific data analysis, can be found in [23–25].

2. Software architecture

2.1. Introduction

In this section we give an overview of the software architecture, used in our Java-based GUI. We will do this by means of so-called unified modeling language (UML) class diagrams. UML is a standard notation for visually describing and interpreting object-oriented software applications [19,20]. The UML class diagrams presented in this work are more or less drawn in the specification perspective, that is to say we are looking at software, but often rather at types than at actual implementation details of classes [19] (e.g. we are not indicating the visibility of attributes or methods).

2.2. Directory/package hierarchy

In Fig. 1, the directory/package hierarchy of the MRUI software system is shown. The notation is such that /PATH/mrui/name1 indicates subdirectories /mrui/name1 containing the source codes of the Java packages [9] mrui.name1 (thus name1 stands for conversion, quantitation and so on). Furthermore, /PATH/mrui/name1/name2 stands for subdirectories /mrui/name1/name2, containing native source codes (mostly in Fortran 77) of related quantitation or preprocessing methods (for instance, name2 may be substituted with Cadzow, AMARES or HLSVD; see for instance, Sections 2.5 and 2.6). Finally, /PATH/mrui/images denotes the subdirectory/mrui/images containing the system icon files.

From the figure it can be seen that mrui is the leading Java package. This package contains, among others, the Mrui class, which is the class running the main method [9] of the Java application. The subpackages mrui.name1 take care of various aspects of the MRUI software [6] such as conversion of NMR spectrometer data files, quantitation, preprocessing or simulation of magnetic resonance spectroscopy (MRS) signals, graphical presentation of results and tracking of preprocessing history of MRUI sessions.

In the next subsections the mrui package and related mrui.name1 subpackages will be discussed.

2.3. The mrui package

The Mrui class of the mrui package is the class being responsible for launching the MRUI application. To that end its main method creates an instance (an object) of the Mrui class. During initialization the Mrui constructor creates a MainWin object and a History object (see Fig. 2).
Another important task of Mrui is that it takes care of providing user-selected working modes, such as one-dimensional, two-dimensional or time-series MRS, one-dimensional Gabor or magnetic resonance spectroscopic imaging (MRSI).

The MainWin object, launched by Mrui's constructor, creates and manages the main window of the MRUI graphical user interface (GUI). Its class extends the JFrame Java Swing GUI component [13,14]. One of the important tasks of MainWin is to open or close child windows for being able to carry out the various activities of MRUI (conversion, quantitation, preprocessing or simulation; see Fig. 1). To be able to embed these windows inside MainWin's JFrame main window, we have adopted the approach of working with Java Swing JInternalFrame's [13,14]. This also offers the opportunity of using Swing-provided Desktop classes for managing desktop icons (JDesktopPane) and look-and-feel specific actions (DesktopManager).

The various MRUI-GUI activities, accessible via the MainWin main window, are generated by clicking/selecting Swing GUI components of the MainMenu or the MainToolBar class (see again Fig. 2). Instances of these classes work closely together with instances of the GUIMain class. That is to say events, generated by the Swing GUI components, are handled by the actionPerformed method [11] implemented in GUIMain. Proper event handling is effectuated by comparing typeOfEvent integers with named constants integers [9]. Closely related to these activities are the various launch methods provided by the Mrui class. The accessMethods methods, mentioned in the class diagrams of MainMenu and MainToolBar, take care of calling the setEnabled method of the various Swing GUI components during handling of the MRUI-GUI events.
An interesting feature of the Java-based MRUI graphical user interface is that one can follow the various preprocessing steps, taken in the course of an MRUI session. To that end instances of the Preprocessing, Data and History classes work closely together. Also, the OutputToolBar attribute of MainWin (see Fig. 2) is directly involved in this visualization of MRUI preprocessing history.

Since the Data class implements the Serializable interface, complete states of objects of that class can easily be stored on disk by calling the writeObject method of an ObjectOutputStream [10]. This mechanism offers the MRUI user the opportunity of undoing preprocessing steps.
2.4. The mrui.conversion package

Conversion of the file structure of commercial NMR spectrometer data files into a structure suited for the MRUI package is of paramount importance [6]. The mrui.conversion package contains a number of classes to deal with this activity. By checking the file extension of the data files the MRUI software is capable in an automated way of choosing the correct conversion algorithm. At present conversion algorithms for various commercial NMR spectrometers are available, including Bruker, GE, Philips and Siemens.

2.5. The mrui.quantitation package

Quantitating (in vivo/medical) MRS signals is one of the core activities when running the MRUI program. The quantitation process in fact means fitting some form of model function to complex-valued data points, sampled in the time domain [6]. The MRS signals can have either a one-dimensional (1D) or two-dimensional (2D) structure, or can be time series of several 1D signals. Furthermore, the MRS signals can be acquired either as free induction decays (FIDs) or as nuclear spin echoes [6].

In order to improve the quality of the quantitations it usually is of vital importance to employ prior knowledge on the MRS signals as much as possible. This prior knowledge can be knowledge on spectral parameters of the signals but also knowledge on instrumental effects when acquiring the signals.

In the next subsubsections we describe how to deal with spectral prior knowledge on MRS signals and how to handle quantitation of MRS time series. We will do this on the hand of the AMARES method [26,27], an MRS quantitation approach added recently to the MRUI package [6].

2.5.1. Handling prior knowledge on MRS signals

Extracting parameters from MRS signals by fitting model functions to the data usually requires a lot of involvement from the users. This stems from the fact that nonlinear (least-squares) fitting methods demand starting values for the nonlinear parameters involved (see for instance [26]). In addition, other highly graphical interactions, to be performed by the users, concern the provision of prior knowledge on the MRS signals concerned. This prior knowledge, usually coming from the field of biomedicine/chemistry, can normally be expressed as linear relations between similar spectral parameters.

Supposing that the type of MRS signal, concerned, is being quantitated for the first time, a first step to do is to obtain starting values for the spectral parameters via peak-picking on the Fourier transform of the signal [6]. In the UML class diagram of mrui.quantitation.Amares.GUI.AmaresSV_PKWin (see Fig. 3) this functionality is indicated by the peakPicking method of the AmaresSV_PKWin class.

When comparing Fig. 3 with Fig. 2 it can be seen that the interface approach is more or less the same for both packages. In fact, the interface approach could be made the same for all MRUI packages. This possibly can be considered as a benefit for working with powerful libraries and built-in tools as provided by the Java programming language.

Performing peak-picking in order to obtain starting values for in vivo MRS quantitation actually means collecting interactively two numbers for each NMR peak involved. The first number concerns the position of the NMR peak in the frequency domain, or in other words its frequency. The second
number concerns the position of the NMR peak at half-height. From this position the decay constant in the related time domain can be calculated, assuming one knows the decay function [6].

The starting values, established for each NMR peak to be quantified, are presented in the AmaresSV_PKWin window via a JList Java Swing GUI component [13,14] called SVlist (see Fig. 3). The JList component is needed to provide the users with items that can be selected (see below).

In order to be able to add items to SVlist a StartingValues class was developed. This class belongs to the mrui.quantitation.Amares.priorknowledge Java package (see Fig. 4). The task
of StartingValues is to add to, get from or change starting values stored into a Java Vector attribute called SVvector.

After obtaining starting values, the next step in the process of preparing an AMARES-based MRS quantitation is to provide spectral prior knowledge on the NMR peaks involved. To that end a number of classes were developed, also being part of the mrui.quantitation.Amares.priorknowledge Java package (see again Fig. 4). The central prior knowledge class is called PriorKnowledge. Its task is to fill, retrieve and change prior knowledge information, to be stored in a number of Java Vector's. This prior knowledge information is distributed over the parameters forming an NMR peak, that is to say the frequency, time-domain damping (decay) constant, amplitude, phase and the kind of damping function.

Directly related to PriorKnowledge is the Equation class. This class is to be used to capture prior knowledge-based constraints on spectral parameters such as ratios and shifts and upper and lower bounds [26].

Like the starting values, the prior knowledge information is to be presented via a JList GUI component. To that end the PKlist attribute was added to AmaresSV_PKWin (see Fig. 3). Once the SVlist and PKlist are completed, the next step to do is to link items from both lists.
the MRUI-GUI software a single combination of such an SVlist and PKlist item is called a Metabolite (see Fig. 4).

A final step to be done before starting the MRS quantitation is to create a so-called Model. This is accomplished by inserting various Metabolite’s into a JList attribute called modellist (see again Fig. 3). Once the modellist is completed the actual AMARES quantitation code [26] is to be called. This is performed via the nativeQuantitation method of the Amares class. Since the AMARES code is written in Fortran, it means that calling the code from the MRUI Java environment should be carried out via the Java native interface (JNI) [22]. At present a direct call to Fortran via JNI is not supported. To circumvent this problem we work with an interface function written in ANSI C.

A derived task of the Amares class is that it should convert the MRUI internal data structure into the structure suited for the AMARES native code. The amaresCalls method takes care of that.

Fig. 4 shows that all classes, being involved in realizing starting values and prior knowledge, implement the Java Serializable interface. This means that instances of these classes can be stored on disk into a MRUI-based prior knowledge database.

2.5.2. Handling quantitation of MRS time series

For various biomedical applications it is of interest to follow certain spectral parameters of MRS signals as a function of time. A well-known example of such a time-series investigation is the study of $^{31}$P MRS of calf muscle as a function of exercise protocols (see for instance [28]).

Very recently, Vanhamme et al. [27] have shown that one obtains statistically better results when quantifying simultaneously all signals of time-series MRS data sets. The AMARES times-series method, described in their paper (see also [26]), is embedded in the new version of MRUI.

An important aspect of quantifying in an automated way all signals of a time-series data set is, that the signals should be arranged into groups having similar spectral prior knowledge. To that end, as part of the mrui.quantitation.Amares.GUI Java package, we have developed the class depicted in Fig. 5.

An AmaresSeriesWin child window, extending Java Swing’s JInternalFrame, is opened after selecting the time-series MRS working mode. The task of AmaresSeriesWin is to provide a (scrollable) list of all MRS signals, involved in the times-series measurement. The AmaresSeriesWin windows also displays a list that should contain groups of MRS signals needing the same quantitation constraints as a consequence of spectral prior knowledge.

At first this second list is empty. Then groups of MRS signals are to be formed by selecting signals from the list of signals and passing them to the list of groups (unpassing is also possible).

2.6. The mrui.preprocessing package

Within the MRUI package processing is defined as performing operations on MRS signals that change the values of the data points [6]. Examples are reduction of noise by applying the Cadzow enhancement procedure [29] or removing unwanted spectral features by applying SVD-based filtering [30]. In the Java-based MRUI-GUI software, described in this paper, the nomenclature preprocessing is used, however. This is done because in MRUI sessions often processing steps are done before carrying out the actual MRS quantitation. This does not take away that processing on its own may be of great importance.
The classes, developed for launching the various preprocessing methods, are forming part of the `mrui.preprocessing` Java package. In Fig. 6, we have depicted the classes related to the Cadzow enhancement procedure. The central class in this preprocessing example is the `CadzowPrepro` class.
This class contains the nativePreprocessing method, being the JNI-based interface [22] to the original Fortran code [29].

Since CadzowPrepro also implements the Java Runnable interface, it means that the run method of the Java Thread class [10] should be implemented. This offers the opportunity of manipulating CPU-intensive Cadzow preprocessing via the mechanism of adjusting thread priorities [21].

A last method of CadzowPrepro, to be mentioned, is the histo method. This method plays a role in visualizing the history of preprocessing steps on the MRUI OutputToolBar (see again Fig. 2).

2.7. The mrui.simulation package

A way of testing new MRS quantitation approaches is to apply them to MRS signals with known spectral parameters. This usually means simulating signals, either with or without including noise realizations. In the current Java-based GUI the simulation functionalities of MRUI [6] can be called via the classes of the mrui.simulation Java package. They are presented in the UML class diagram of Fig. 7.
3. Results

3.1. Hardware and operating systems

The computers, we have used the most for developing and testing our MRUI-GUI software, are multimedia PCs. For instance, we have worked with a Gateway2000 GP7-500 XL Pentium III computer with 256 MB memory, an 8 GB hard disk and a 16 MB STB NVidia Riva TNT AGP graphics card. In addition we have used Windows NT 4.0 as the operating system of choice. Also, for testing platform independency of the GUI part, we have run MRUI under Windows 95 and the Linux operating system.

3.2. Software

The following software tools/packages were used to facilitate the development of the MRUI-GUI:

- The Java development kit (JDK). This is the standard Java package provided by Sun Microsystems. A problem is that the Java software is still not stable. The latest version of Java, we are now working with, is JDK version 1.3 Beta [15].
- Together/J. This is a visual UML modeling tool for the Java programming language. The Java-based applications are designed by drawing the various diagrams of the UML approach [19,20]. In addition, the Together/J program generates automatically the source codes of the corresponding Java classes. At present we are working with Together/J version 3.0 [31].
- GCC. This is the GNU compiler collection. We have used its Fortran (g77) compiler to build the Fortran 77 based libraries and its C++ compiler to compile the C-based interface code between JNI and Fortran. At present we are working with version 2.91.57 [32].

3.3. Platform dependency

The Java-based MRUI software is only platform dependent through calls to native code via the Java Native Interface [22], as is required for certain quantitation and preprocessing methods (see for example Figs. 3 and 6). That is to say, the GUI part of MRUI is in pure Java code and thus should be really platform independent.

In order to illustrate the latter we have made screen captures of some MRUI-GUI windows, when running on various platforms. For instance, in Fig. 8 the MRUI main window is shown, as captured via Windows 95. It also contains the child window for choosing the MRUI working mode. Moreover, in the next figure (see Fig. 9) the same working mode window is depicted again, this time captured via the Linux operating system. It can be concluded that the same Java class bytecode can be used on both computer platforms, although small differences are visible in the two graphical presentations.

Concerning installing the Java-based MRUI on the various computer platforms the following can be said. Once the libraries for the native (Fortran) code have been generated for the platform concerned, most of the work has been done. After that the whole MRUI file system can be stored into an archive file. For instance, for the Windows 95/NT system the current version of MRUI can be stored into a ZIP file of about 5 MB. Extraction of the files on the local computer takes of the order of 30 s. This yields a file system with about 8 MB of Java related files and about 4 MB of
Fortran related dynamic linking libraries (DLLs). After extracting the MRUI files the only other action to be done is adding in a `run.bat` batch file the correct path to the local JDK software. In fact, to be more exact, on the target machine only the Java runtime environment (JRE) part of the JDK is needed [15] in order to be able to run MRUI.
3.4. A quantitation example: separating overlapping spectral components of unknown shape

Sometimes it is of interest to quantitate NMR peaks that are overlapping with spectral components having unknown line shapes. In those cases the unwanted overlapping background features can be taken care of by modeling them, in a sort of mathematical way, as sums of exponentially decaying sinusoids. To that end SVD-based methods [5,33] are quite convenient to be employed since they do not require any starting values or prior knowledge.

In Fig. 10, we show, as an example, the MRUI result obtained by applying the HLSVD method [30] to a simulated MRS signal. The complication of the signal is that it contains a small NMR component (peak) at 3.13 ppm which is completely covered by a large background peak centered around 2.35 ppm. This background peak deliberately was given a deviating decay function, leading to a kind of triangular shape in the frequency domain. As a consequence the background peak is to be decomposed into a sum of exponentials. Nevertheless, the parameters obtained for the peak of interest are not too poor, considering the heavy overlap problem. For instance, the amplitude was estimated to be 1.4 whereas the true value is 1. This has to be seen in the light that the overlapping background peak has an amplitude being about 40 times larger than that of the peak to be quantitated.

Another sign that the black box quantitation by means of HLSVD was a success is the fact that the residue between the fitted model function and the simulated MRS signal contains only noise.
It appeared that the computational time, actually needed for carrying out the HLSVD analysis of the example signal, is about a few seconds on a Gateway Notebook Solo 9100 working under Windows 95. This time was found to be comparable to that used by a standalone Fortran 77 HLSVD program, when running on a Gateway2000 G6-450 XL Pentium II PC with the Linux operating system. Very recently it has been established, when comparing the performance for technical computing of Java, Fortran and C++, that “Java will soon be a serious contender for numerically intensive computing” [34]. For the floating-point-intensive benchmarks, investigated in the paper, it was found that Java can achieve at least 80% of the peak Fortran performance.

4. Conclusions

We conclude that:

- The work on the Java-based MRUI graphical user interface (GUI) confirms that the Java programming language is indeed well suited for developing highly interactive graphical software applications. Particularly, the Swing part of the Java Foundations Classes [13,14] appears to be a rich user interface class library.
- Through the Java native interface (JNI) approach [22] we can embed code, generated in Fortran, into our software system. To that end an additional interface function written in ANSI C is to be used since the currently applied JNI does not support direct interfacing to Fortran.
- Analyzing UML class diagrams [19,20], produced by the Together/J visual UML modeling tool [31], is helpful in better understanding and improving the development of systems like the MRUI-GUI.
- Platform independency of the Java class bytecode helps in simplifying the software installations on the various computer platforms.

5. Summary

5.1. Introduction

Since about half a decade a magnetic resonance spectroscopy (MRS) software package exists, called the magnetic resonance user interface (MRUI). This package allows MR spectroscopists to easily perform quantitative time-domain analysis of in vivo/medical MRS data. In the first version of MRUI its graphical user interface (GUI) employed the user interface control objects, provided by the commercial MATLAB programming environment. In this work we present the design of a new GUI for MRUI, now completely written in the Java programming language. Since the standard software of the Java development kit (JDK) can be downloaded for free from the Internet, we were able in this manner to undo the need of using the commercial software.

5.2. Methods

We have used the Java programming language to build the MRUI-GUI, since it benefits the concepts of object orientation. Furthermore, with Java one can create applets that can be downloaded
from any WWW server, Java bytecode is claimed to be platform independent, the Java library contains a complete collection of GUI components and Java supports interfacing to native code.

In order to be able in a more systematic way to describe and interpret the various parts of our software system, we have used the concepts of working with diagrams of the unified modeling language (UML).

5.3. Results and conclusions

We have used Pentium III and Windows NT based PCs for developing and testing our MRUI-GUI software. Also we have used Linux based PCs for testing the Java-claimed platform independency. We could indeed confirm that our software system is only platform dependent through calls to native code, as is sometimes required for employing certain existing quantitation methods. At present all functionalities, already accessible via the MATLAB based GUI, are also embedded in the new GUI. Moreover, we have added new features, including simultaneously quantitating all signals of time-series MRS and generating output of quantitation results in the form of WWW-suited code. We could conclude that the Java programming language is well suited for developing highly interactive graphical software applications.

Acknowledgements

The authors thank Dr. S. Cavassila of the University of Lyon and Dr. L. Vanhamme of the University of Leuven for helpful discussions concerning the MRUI-GUI. This work is supported by the European Union TMR/Networks programme (project number ERB-FMRX-CT97-0160).

References


Alexandre Naressi obtained his Engineering degree in Computer Science of the Institut National des Sciences Appliquées, Lyon, France in 1998. He was one of the developers and builders of the graphical user interface (GUI) of the Java-based MRUI package.

César Couturier obtained his Engineering degree in Computer Science of the Institut National des Sciences Appliquées, Lyon, France in 1998. He was one of the developers and builders of the graphical user interface (GUI) of the Java-based MRUI package.
Igor Castang is studying at the Institut National des Sciences Appliquées, Lyon, France, for getting Engineering graduated in Computer Science. He has worked on the Java-based MRUI package in the area of automatic conversion of datafile structures of commercial MR scanners.

Ron de Beer received his Ph.D. degree in physics in 1971 from the Applied Physics Department of the University of Technology Delft, Netherlands. In 1989 he was appointed Associate Professor at this Institute. His present fields of interest are quantitative data analysis of in vivo magnetic resonance (MR) signals and integration of MR signal processing and data analysis techniques into Java-based software systems.

Danielle Graveron-Demilly is Engineering graduate of the Institut National des Sciences Appliquées, Lyon, France, 1968; she got her Ph.D. in 1970, Lyon and her D.Sc. in 1984, Lyon. Since 1968, she has been an Engineer of Research in the NMR Laboratory, now CNRS UMR 5012, at Université Claude Bernard, Lyon I. She supervises the signal processing group. Her research interests are signal processing for in vivo magnetic resonance (MR) spectroscopy and MR imaging. She co-ordinates the development of the Java-based MRUI software in the context of the European project, TMR, FMRX-CT97-0160.