ODIN—Object-oriented development interface for NMR

Thies H. Jochimsen* and Michael von Mengershausen

Max-Planck-Institute of Cognitive Neuroscience, Stephanstr. 1a, D 04103 Leipzig, Germany

Received 4 December 2003; revised 4 May 2004

Abstract

A cross-platform development environment for nuclear magnetic resonance (NMR) experiments is presented. It allows rapid prototyping of new pulse sequences and provides a common programming interface for different system types. With this object-oriented interface implemented in C++, the programmer is capable of writing applications to control an experiment that can be executed on different measurement devices, even from different manufacturers, without the need to modify the source code. Due to the clear design of the software, new pulse sequences can be created, tested, and executed within a short time. To post-process the acquired data, an interface to well-known numerical libraries is part of the framework. This allows a transparent integration of the data processing instructions into the measurement module. The software focuses mainly on NMR imaging, but can also be used with limitations for spectroscopic experiments. To demonstrate the capabilities of the framework, results of the same experiment, carried out on two NMR imaging systems from different manufacturers are shown and compared with the results of a simulation.

PACS: 87.59.Pw

Keywords: NMR; Software; Sequence programming; Platform-independent; Pulse design

1. Introduction

Nuclear magnetic resonance (NMR) is a versatile tool to investigate physical properties of materials and living tissue. The flexibility of the NMR technique can be attributed to the fact that a wide range of experiments is designed by solely altering the software that controls the hardware during the measurement. With a given set of hardware components, various properties of the sample can be examined with different software-based experimental setups (i.e., pulse sequences). An important task of the NMR scientist who develops new NMR applications is therefore that of a software engineer. Provided a sophisticated programming interface for sequence design is available, advances in the field of computer science can accelerate the process of creating NMR applications.

Contemporary concepts like object-oriented design, polymorphism, and generic programming are used nowadays in software engineering to create modular, extensible, and easy-to-use software instead of procedural programming (an excellent overview of these programming paradigms and their implementation in C++ can be found in [1]). By contrast, NMR pulse sequences are usually programmed using the procedural approach. That is, the scientist provides a program that contains a list of sequential instructions to trigger hardware-events together with some calculations to achieve the required properties of the sequence (e.g., resolution, orientation, and contrast). This results in a non-modular, monolithic implementation of the sequence which seriously limits the reuse of certain parts in another sequence, except for duplicating the source code. A modern approach would describe the sequence as a composition of reusable, self-consistent objects that can be combined freely to develop new experimental setups.

Recently, a software architecture has been presented [2] which makes use of this approach by a double-layered design whereby the user interacts with an application framework written in Java [3] which is mapped to corresponding C++ functionality on the hardware.
controller and signal processing computer. The pro-
programming interface is provided not only for sequence
programming but also for developing work flows which
incorporate different measurement techniques for clini-
cal application. However, this framework is limited to
the devices of one manufacturer and its double-layered
design may impose a considerable overhead when add-
ing new functionality, for example custom real-time
feedback.

In contrast, ODIN, which is subsequently introduced,
concentrates on platform-independent sequence design,
and data processing with a single open-source code basis
in C++. The hardware-dependent components that
drive the different scanners are encapsulated into low-
level objects (pulses, gradients, and data-acquisition)
from which complex, platform-independent parts of the
sequence are constructed. The same source code is used
at all stages of sequence development, from simulation
on a stand-alone platform to play-out on a real-time
system. ODIN uses the native functionality of the
graphical user interface on each platform, allowing a
seamless integration of ODIN sequences. Although
ODIN is a relatively young software project, its se-
quence programming interface has been shown advan-
tageous in developing sophisticated functional magnetic
resonance imaging (fMRI) applications [4–6], in simu-
lations [7], and in the application of its module for pulse
design [8].

In this paper, the first section gives an introduction
into the ODIN sequence programming interface and its
underlying concepts. The design of radio frequency (RF)
pulses will be described in more detail as this is one of
the major strengths of ODIN. The next two sections
contain additional information about the internal rep-
resentation of the sequence within the ODIN library and
the mechanisms that are used to execute the experiment
in different hardware environments. After that, strat-
egies to visualize and simulate the sequence are presented,
and the data processing framework of ODIN is dis-
cussed. Finally, experimental results obtained with
ODIN on different platforms are shown and compared
with the results of a simulation.

2. Platform-independent sequence design

An NMR experiment is basically a sequence of pe-
riods where the sample is exposed to different magnetic
field configurations, such as RF pulses and magnetic
field gradients, or periods where data are acquired.
From these basic sequence elements, complex exper-
iments can be composed which measure spectroscopic
properties, relaxation, and transportation processes of
the spins within the sample. Magnetic field gradients
extend these experiments to spatially resolved data sets,
i.e., images of these parameters. In addition, repetitive
measurements yield time series of physiological pro-
ces within living tissue, for example, neuronal activity
in the human brain.

The NMR sequence can be described in terms of the
physical properties of their elements and the arrange-
ment of these sequence elements as a function of time. A
simple NMR sequence is shown in Fig. 1. This level of
description is independent of the measurement device.
ODIN provides a programming interface in terms of a
C++ class hierarchy which reflects the physical aspects
of a sequence. A sequence program which is written
using this framework can be executed on different NMR
hardware. The system-specific actions are performed by
a library that transfers the sequence-specific requests to
the actual measurement hardware as depicted in Fig. 2.
The benefit of separating the physical logic of the ex-
periment from the peculiarities of the current hardware
is the portability of the sequence program. It can be
reused with other hardware, even from another manu-
facturer.

2.1. Sequence programming interface

In the following, the term basic sequence objects refers
to elements of the sequence that cannot be divided into
smaller elements from the physical point of view. Ex-
amples of such “sequence atoms” are periods of RF
irradiation, the application of temporary field gradients
or intervals of data acquisition. Each basic sequence
object is represented by a C++ class which handles its
physical properties, for example the duration. These
objects are constructed during the initialization of the
sequence according to the instructions given by the se-
quence programmer. From this collection, the sequence
is constructed by grouping the sequence objects into
container objects. To simplify the notion of composing
new container objects, the operators + and / are over-
loaded, i.e., they are redefined with sequence objects as
operands, and can be used to specify whether two se-
quence objects a and b should be played out sequentially
(a+b) or in parallel (a/b). As an example, the source
code for the simple sequence visualized in Fig. 1 is
printed in Fig. 3.

Besides this technique of building sequences from
scratch by grouping basic sequence objects together, the
ODIN library offers many predefined high-level se-
quence objects as C++ classes. For example, the object
\( a_{eq} \) in Figs. 1 and 3 is an acquisition window with the
simultaneous application of a gradient field that is used
in many imaging sequences for spatial frequency en-
coding. These more complex objects are constructed
from basic sequence objects within the library, using the
same mechanism of building container objects as the
sequence programmer would. In addition, the class of
these composite objects provides an interface that is
adjusted to its high-level concept. For instance, the ob-
ject acq has a member function that returns the point in time of the center of the acquisition window with proper consideration of the delayed onset due to the ramp of the simultaneous gradient.

2.2. Pulse design

A crucial part of the sequence is the application of RF pulses to generate a detectable signal from a limited spatial or spectral range of spins within the sample. The ODIN framework contains a flexible module for the generation and simulation of RF pulses. A wide range of pulses is supported by a plug-in style mechanism. The desired excitation profile, gradient shape, and frequency filter can be selected and modified separately to match the pulse optimally to the specific application. It can be easily extended by supplying the module with new plug-ins which generate k-space trajectories or calculate the RF waveform as a function of time or k-space coordinate. The following pulse types are already supported by existing plug-ins of the ODIN library:

- Slice-selective pulses (Sinc, Gauss), optionally with a VERSE [9] trajectory for reduced power excitation.
- Adiabatic pulses (Sech [10], WURST [11]).
- Spectrally and spatially selective pulses [12] for slice-selection with a predefined spectral profile (e.g., for fat suppression).
- Two dimensional (2D) pulses [13] with various excitation shapes and different spiral trajectories.
- Composite pulses [14] which are created by concatenating one of the above pulses with different transmitter phases and flip angles.

In addition, these pulses can be filtered either in k-space or in the time domain using a filter plug-in. The benefit from separating the pulse shape and the trajectory into different plug-ins can be illustrated by considering the generation of 2D pulses: each of the excitation profiles (point, box, disk, and user-defined list of points) can be used in combination with any of the 2D trajectories in order to generate a pulse profile that is well adjusted to the requirements. For example, an excitation profile that consists of a chain of adjacent points together with a
slew-rate optimized trajectory is useful for curved slice imaging [15]. Because the pulse module is a regular sequence object, it can be integrated seamlessly into any NMR sequence. For example, the object pulse in Figs. 1 and 3 is a slice-selective specialization of this module using the Sinc plug-in for the pulse shape. In addition, a graphical user interface (Fig. 4) which acts as a front-end to the pulse module can be used for interactive pulse design and monitoring of the corresponding excitation profile.

2.3. Loops and vectors

An essential aspect in most NMR experiments is to repeat certain parts of the sequence unchanged or with different settings. Examples are the repetition of a gradient-echo with different strength of the phase-encoding gradient in conventional Fourier imaging as used in the sequence of Fig. 1, or the repetition with different pulse frequencies for multi-slice acquisition.

To use this technique in a uniform manner, ODIN introduces the concept of vector objects and loop objects. Vector objects are elements of the sequence that are used repeatedly with different settings. The following predefined vector classes, derived from a common base class Seq Vector, are available to the sequence programmer:
- Gradient pulses with different gradient strengths for phase encoding or diffusion weighting.
- Sequence objects that drive the transmitter (RF pulses) or receiver (acquisition windows) contain two vector objects for frequency and phase switching to be used for multi-slice experiments or phase cycling.
Delay objects with a variable duration, which is changed for each iteration.

A list of user-defined rotation matrices that can be attached to gradient-related objects to alter their direction subsequently.

A container object that holds a list of other sequence objects which are played out sequentially for each repetition.

Although this set of specialized vector classes is probably not exhaustive, the last class may be used to easily
extend this list by storing sequence objects for each repetition into the container. This emulates the behavior of a built-in vector class.

To specify which parts of the sequence will be repeated and which vectors will be modified at each repetition, loop objects play a central role in sequence design with ODIN. They possess a function-like syntax (functors) when used within a sequence:

```
loop (kernel) [vector1][vector2]...
```

With this line of source code, the loop object `loop` is used to repeat the sequence part `kernel` while incrementing the properties of the vector objects `vector1`, `vector2`, ... that are located within `kernel`. Instead of using a vector object, an integer number `N` can also be given as an argument to the loop, which will then repeat the sequence part `N` times unchanged. By using this common notation for all variable aspects of a sequence, new applications can be implemented rapidly without dealing with the specific aspects of the hardware.

### 2.4. Sequence parameters

Normally, each sequence has a set of parameters which specify the actual experiment, for example, the sampling rate for data acquisition or the duration of the RF pulse. The sequence parameters are edited interactively within the user interface of the measurement device, and the sequence is recalculated according to the new settings. Within ODIN, these parameters are members of the C++ sequence class, allowing transparent access to their values in the member function that prepares the experiment. Well-known data types (integer numbers, floating point numbers, and Boolean values) can be used as sequence parameters. They are designed to be used exactly like built-in types of the C++ language, resulting in understandable source code.

Whenever possible, the native user interface of the measurement device is used to present the set of parameters specified by the sequence programmer. Thereby, the parameter values are exchanged between the native user interface and the ODIN library. If no native mechanism for parameter editing exists (e.g., on a standalone platform), ODIN provides its own set of widgets using the Qt library [16] to edit the parameters interactively (Fig. 5). After the measurement, the parameters are stored on disk in JCAMP-DX format [17] together with the raw data. In the post-processing step, the parameters and the raw data are then read from disk.

### 3. Internal representation of the sequence

Any NMR sequence has a nested structure, that is, basic sequence objects can be grouped together to form
296 logical units, which in turn can be collected to build
more complex units. This leads to an internal repre-
298 sentation of the sequence as an ordered tree of sequence
299 objects. The leaves of this sequence tree are the basic
300 sequence objects (RF pulses, gradients, acquisition
301 windows, and evolution delays). The sequence contain-
302 ers are represented by nodes of the tree. They contain a
303 list of references to their members in the same order as
304 given by the sequence programmer. The nodes can
305 contain additional information, e.g., a loop object con-
306 tains the number of repetitions besides the elements of
307 the sequence that are repeated.
308
The tree is constructed during the preparation phase
309 of the experiment according to the instructions of the
310 sequence programmer. Each sequence has its special
311 tree. As an example, Fig. 6 depicts the sequence tree
312 structure for the sequence of Fig. 3. The created se-
313 quence tree is the central data structure that is used in
314 further steps of the experiment. If a certain operation
315 has to be performed for the sequence, e.g., calculating
316 the total duration of the experiment, the sequence tree is
317 traversed recursively, querying each object for a value
318 (in this case its duration), or requesting a certain oper-
319 ation from the object. Thereby the starting point is the
320 root of the sequence tree. At each node that contains an
321 ordered list of other sequence objects, these sub-objects
322 are in turn requested to perform the operation. This
323 recursion in each branch terminates at the leaves, if a
324 basic sequence object is reached. The two following
325 sections describe how this technique of traversing the
326 sequence tree is used to control the measurement device
327 or to visualize and simulate the sequence.
328
The whole sequence (i.e., the root of the tree) is in
329 itself a container object, represented by a C++ class,
330 which is implemented by the sequence programmer.
331 This class is derived from a base class that acts as an
332 interface between the sequence and the ODIN library.
333 By the mechanism of virtual functions in C++, a set of
334 sequence-specific member functions must be provided
335 by the sequence class that will be called during initiali-
336 zation, preparation, and data processing of the experi-
337 ment. With this technique, all sequence modules share a
338 common interface which can be used by the library in a
339 uniform manner.
4. Hardware-specific implementation

In this section, two examples show how the ODIN sequence tree can be utilized to drive the hardware of two scanners from different manufacturers:

Platform A (Bruker Medspec, 3 T) is driven by a pulse program which is an ASCII file that contains a list of sequential instructions for the hardware and controlling structures (loops, jumps) to repeat certain parts of the sequence. To perform an experiment, a set of parameters must be provided that contains the detailed settings for the measurement. The pulse program and the parameter set cover all characteristics of the experiment on this platform. ODIN maps its internal representation of the sequence to the device by traversing the sequence tree and generating an entry in the pulse program for each sequence object. In addition, each sequence object is asked to make an entry into the parameter set. After transferring the generated files to the system controller, the sequence can be executed. Because the pulse program is generated externally on the workstation, the limited memory and speed of the system controller is not an issue. Even better, different variations of the pulse program, which would usually be implemented by conditional statements in the pulse program itself (if-then-else instructions), are handled by ODIN. Therefore, a minimal pulse program is generated for each experiment containing only the necessary instructions, thereby reducing the code size which is actually processed by the system controller.

On platform B (Siemens Trio, 3 T), the system components are driven directly by a C++ program in real time. The corresponding source code must be provided by the sequence programmer. It contains instructions to trigger hardware events (RF pulses, gradients) at specified points in time. The experiment is performed during run-time of this program. On this platform, ODIN executes a sequence by traversing the sequence tree at run-time, querying each sequence object for a corresponding event. An internal counter takes care of the correct starting time of each event. Although the additional level of indirection when using ODIN to trigger the hardware events decreases execution speed a little, it was still fast enough to execute all ODIN sequences, which were tested so far, in real time. An additional amount of memory is required for the ODIN library (typically 5 MB) which can be easily accommodated in the free memory (approximately 18 MB) of the used system.

In the procedures described above, the connection between the ODIN library and the current platform is realized by a set of so-called hardware drivers, as illustrated in Fig. 2. These hardware drivers are implemented by C++ classes. Each basic sequence object uses a hardware driver to execute itself on the current platform. Thereby, the hardware drivers are interchangeable, depending on the hardware to operate. For example, an RF pulse uses different hardware drivers on each platform: the driver for platform A is responsible for an entry in the pulse program and for the pulse-specific parameter settings. The driver for platform B is...
responsible for preparing and triggering hardware
events to execute the RF pulse. The internals of the
drivers are hidden behind a common interface (abstract
virtual base class in C++) so that there is little coupling
between the drivers and the rest of the library. With this
design, the code to deal with the peculiarities of each
platform is located only within a small set of C++
classes. In the case of porting ODIN to a new platform
or in the case of a software update by the manufacturer
which is accompanied by a considerable change of the
sequence programming interface, only these driver
classes have to be implemented or updated, the rest of
the library and the ODIN sequences remain unchanged.
The benefit is straightforward portability to new system
types and minimum effort in case of a software update.

Usually, the sequence programmer is responsible for
manually adding code to calculate the total duration of
the sequence or estimating the RF power deposition for
safety control in human or animal studies. With the
sequence tree in ODIN, which holds all information
about the sequence, this tedious process can be com-
pletely automated by the library which traverses the
sequence tree and queries the objects at each branch for
their properties (duration, power deposition). Thereby,
simple but error-prone programming tasks are trans-
ferred to the ODIN library, allowing the sequence pro-
grammer to concentrate on the important features of the
sequence.

5. Sequence visualization and simulation

Even on computers where no NMR device is at-
tached, the ODIN framework can be useful for de-
veloping sequences. On a stand-alone platform, the time
courses of the different channels (RF, gradients, and
receiver) can be displayed, or a simulation of the se-
quence acting on a virtual sample can be performed.
This is achieved by giving all basic sequence objects the
capability to generate a digitized version of themselves,
i.e., a function that returns the values of each channel
for equally spaced points in time.

To generate a digitized version of the whole sequence
for visualization, the container objects can combine them
recursively, traversing the sequence tree until the whole
sequence is processed. The result can then be displayed
graphically. For simplicity, this is currently realized by
generating a multi-channel audio file which is then dis-
played using conventional sound editors. In addition,
predefined functions exist which calculate important as-
pects of the sequence numerically using the digitized se-
quence, for example gradient moments, the strength of
diffusion weighting or the k-space encoding of different
coherence pathways in a multi-pulse sequence.

For the simulation, a virtual sample that holds spa-
tially resolved NMR-specific properties (spin density,
relaxation rates $T_1$ and $T_2$, and frequency offset) is re-
quired. It can be created by means of a graphic editor or
a special ODIN sequence that measures these properties
of a real sample with a high resolution. The latter will be
used in the experimental section of this work to compare
the simulation to actual measurements. The digitized
version of each sequence object is then used to simulate
its effect on the sample. By traversing the sequence tree,
the simulation is performed in the same order as the
sequence objects would be played out on a real NMR
device. An exact solution of the Bloch equations for
piecewise constant fields [18] is utilized for the calcula-
tion: It transforms the magnetization vector at each
point of the sample recursively according to the set of
values within the digitized arrays of the sequence object.
During acquisition periods, a virtual NMR signal is
generated by integrating over the transverse component
of the magnetization vector for all points within the
virtual sample. The result of the simulation is then a
synthetic NMR signal that can be post-processed with
the same algorithm as the real signal would be pro-
cessed.

This simulation strategy is most useful for analyzing
imaging sequences. Because it is limited to ensembles
of isochromatic spins with single-quantum coherences and
interactions simplified by $T_1$ and $T_2$ (e.g., quadrupolar
coupling, spin–spin coupling), other tools [19–21] are
more appropriate to generate virtual spectra of samples
with different nuclei, to simulate higher-order quantum
coherences or explicit interactions. Another limitation is
given by the finite spatial size of the volume elements:
The simulation does not account for static intra-voxel
dehasing due to field inhomogeneities ($T_2^*$).

6. Data processing

In a typical NMR experiment, the RF signal that is
induced by the magnetization of the sample and received
by the coil is post-processed to obtain interpretable
data. This can be a frequency analysis for spectroscopic
applications or the reconstruction of spatially resolved
parameter maps for imaging. In general, the data pro-
cessing algorithm is specific to the NMR sequence which
was used to acquire the raw data. This step is supported
by a software layer that integrates external numerical
libraries consistently into ODIN.

After the measurement, the raw data is processed by a
function of the same sequence module that was used for
the experiment. Because this function is implemented as
a C++ member function, all parameters of the mea-
surement are directly accessible. The external numerical
libraries can be used within this function. After the
processing step, the final data is written back to disk.

When dealing with large datasets, e.g., for fMRI, the
problem arises that the whole record cannot be held in
memory for analysis at once. ODIN addresses this by the use of memory paging mechanisms of the underlying operating system (mmap/munmap functions under Linux/UNIX) so that the array can be accessed transparently, even if it is too large for the main memory.

6.1. Integration of external libraries

As a basis for further integration of external libraries into ODIN, the expression-template based multidimensional array type provided by the Blitz++-library [22] is used to hold the NMR data during the different processing steps. Many useful functions that operate on multidimensional arrays are already made available by Blitz++. However, more complex numerical operations must be added separately as they are not part of Blitz++. Therefore, an interface to the following libraries has been implemented so that they always operate on the array type of Blitz++ and add the described functionality to it:

- NewMat [23]: Supports various matrix types and matrix calculations.
- GSL (GNU Scientific Library) [24]: Non-linear least-square fitting, interpolation.
- FFTW (Fastest Fourier Transform in the West) [25]: Fourier transform for multidimensional arrays.

For example, an FFT of arrays with arbitrary dimensionality can be programmed in one line of C++-code with this integration of external libraries:

```c++
blitz_fftw(data(all,0,all));
```

This instruction will perform a complex in-place FFT over the first and third dimension of the array data for all values with index 0 in the second dimension.

7. Experiments

Two sequences were executed with the same subject and the same settings on platform A and B. Fig. 7 shows the reconstructed images from a power-reduced variant of the modified driven equilibrium Fourier transform (MDEFT) sequence [26]. Although the position of the brain within the slice differs due to different positioning of the subject within the magnet, both images show the same spatial pattern and comparable contrast with a signal-to-noise ratio of 30.5 (platform A) and 25.1 (platform B) in white matter.

In Fig. 8, spin-echo EPI [27] experiments are compared with the result of a simulation which was performed using high-resolution maps of the NMR parameters (spin density, \(T_1\), \(T_2\), and frequency offset). These maps were acquired on platform A during the same session. The simulation was then carried out offline on a Linux PC to generate a synthetic signal using the same sequence code that was used for the measurements. The images are similar in terms of contrast and image quality, but show slightly different field-of-views in phase encoding direction which is very sensitive to frequency offsets due to the small bandwidth. The mismatch may therefore be caused by non-optimal compensation of the field inhomogeneities (shimming) or eddy-currents modifying the phase encoding blips. This otherwise undesired discrepancy could be used here to study the effects of field variations and gradient imperfections. However, the general similarities between the result of the simulation and the actual experiments indicate that the simulation can be used to reproduce the measurement and that it is feasible to develop and test sequences on a stand-alone platform.

8. Availability and licensing

The software package is published under the terms of the GNU General Public License. It can be obtained as source code and binary packages for different platforms (Linux, IRIX, Windows, and VxWorks) from the web.
more than one scanner exists, or to exchange sequences between research facilities with different hardware infrastructure. With the ODIN data processing framework, a consistent interface to reliable open-source libraries for calculating the final data is provided. The internal representation of the experiment by the sequence tree is adequately matched to the application domain and allows easy extensibility when porting the framework to new platforms.

Acknowledgments

The authors thank Robert Trampel, Markus Körber, and Andreas Schäfer for improving the software and Harald E. Möller for helping with the manuscript.

References


Fig. 8. Spin-echo EPI images from platform A (top), platform B (middle), and the simulation (bottom) with a matrix size of 64 × 64, 100 accumulations, 100 kHz sweep-width and the same slices as in Fig. 7. The phase encoding direction is aligned vertically.