Received: 28 August 2018 | Revised: 4 September 2019 | Accepted: 5 September 2019

DOI: 10.1002/mrm.28020

FULL PAPER

Magnetic Resonance in Medicine

Portable and platform-independent MR pulse sequence programs

Cristoffer Cordes[1] | Simon Konstandin[1] | David Porter[2] | Matthias Günther[1,3]

[1]Fraunhofer Institute for Digital Medicine MEVIS, Bremen, Germany

[2]Imaging Centre of Excellence, College of Medical, Veterinary & Life Sciences, University of Glasgow, Glasgow, United Kingdom

[3]MR-Imaging and Spectroscopy, University of Bremen, Bremen, Germany

Correspondence
Cristoffer Cordes, Fraunhofer Institute for Digital Medicine MEVIS, Bremen, Germany.
Email: cristoffer.cordes@mevis.fraunhofer.de
Twitter: @cordesio

Funding information
FhG Internal Programs, Grant/Award Number: Attract 142-600172

Purpose: To introduce a new sequence description format for vendor-independent MR sequences that include all calculation logic portably. To introduce a new MRI sequence development approach which utilizes flexibly reusable modules.
Methods: The proposed sequence description contains a sequence module hierarchy for loop and group logic, which is enhanced by a novel strategy for performing efficient parameter and pulse shape calculation. These calculations are powered by a flow graph structure. By using the flow graph, all calculations are performed with no redundancy and without requiring preprocessing. The generation of this interpretable structure is a separate step that combines MRI techniques while actively considering their context. The driver interface is slim and highly flexible through scripting support. The sequences do not require any vendor-specific compiling or processing step. A vendor-independent frontend for sequence configuration can be used. Tests that ensure physical feasibility of the sequence are integrated into the calculation logic.
Results: The framework was used to define a set of standard sequences. Resulting images were compared to respective images acquired with sequences provided by the device manufacturer. Images were acquired using a standard commercial MRI system.
Conclusions: The approach produces configurable, vendor-independent sequences, whose configurability enables rapid prototyping. The transparent data structure simplifies the process of sharing reproducible sequences, modules, and techniques.

KEYWORDS
high performance computing, modular MR sequence development, platform-independent pulse sequence programming, portable, reproducible, vendor-independent MRI

1 | INTRODUCTION

Modern MRI sequences require sophisticated calculations to drive the MR hardware. These calculations can be very complex for techniques that exploit advanced physical models and vary a lot from one acquisition strategy to another. Many sequence patterns can be combined with each other, which is an active research topic and often done to find the best compromise between image quality, contrast, and acquisition time in various circumstances.

The development of MR sequences by an independent research group is often performed within a vendor-specific framework with a restricted selection of sequence patterns that can be utilized. A large portion of the sequence development work is to adapt a new technique to the specific environment's needs while patching side effects that the new technique has on all other used techniques. Adding a new feature to a fundamental technique which is used in many sequences generally requires a lot of effort.

Previous attempts at bringing reproducibility and vendor-independence to the task of MRI sequence development[1,2] have been useful for early-prototype studies. These tools are great solutions for creating prototype sequences and allow for an easy exchange and reusal of pulse modules. However, these tools have not provided the flexibility that is required for wider application in clinical research. In clinical practice, detailed parameter calculations need to be performed while the patient is being examined because the sequences depend on session-specific properties, such as changing field of view and imaging resolution. These previous approaches involve performing most sequence configuration, such as loop logic or pulse shape calculation, as a separate step prior to transferring the sequence onto the scanner devices. Once transferred to the scanner, it cannot be changed. All pulse shapes and loop counters are fixed. Our goal is to produce sequences that can be used and configured on the scanner with no additional hardware, effectively identical to standard product sequences from the scanner operator's point of view. This also includes the ability to change measurement protocol settings, such as resolution, slice positioning, field of view, repetition time, echo time, optional background, or fat saturation.

We developed a data structure that describes sequences in a self-contained and thereby portable manner. This sequence definition data structure has to be simple and easy to interface with, so that multiple commercial scanners can be supported with minimal integration effort.

Our second goal is to find a new way of designing and combining sequence parts. Some existing tools[3,4] require writing sequence code directly in C++ with a clearly defined and limited idea of what a reusable module might be. The programming paradigm is usually strictly object oriented, which rarely harmonizes with the thought process of an MRI sequence developer. It is more natural to express sequence modules and their interactions in terms of relationships and rules rather than assigning properties to modules in a specific order until all hardware events are fully parameterized. The latter requires the sequence developer to actively think about which parts of a sequence need to be calculated before other parts of other techniques that rely on it. The sequence developer therefore has to work on various implementation details of multiple logical parts of the sequence simultaneously. This often leads to sequence modules that have to be adapted or modified every time they are reused, which is poorly maintainable. Alternatively, this problem leads to modules with high number of responsibilities, which discourages reusability. The proposed approach solves this issue by employing a sequence definition strategy that is closer to the intrinsic rule and dependency-based thought process in MRI sequence development, completely removing the need to specify an order in which the sequence calculations have to be performed.

Some approaches[3-5] simplify the design process by employing a tree structure and more flexible parameter connectivity possibilities. However, the developer has to interact with these structures directly, which becomes confusing when many techniques are combined. This means that maintainability does not scale well as sequence complexity increases. The proposed way of defining sequences does not rely on the sequence developer interacting with these data structures directly. Instead, the sequence developer may specify rules and dependencies that the framework then translates to sequence module connections and calculation procedures.

The presented framework consists of a sequence definition data structure that can be supported by commercial scanners, and a sequence assembly process that encourages the development of flexible, reusable modules.

2 | METHODS

The methods presented in this paper solve multiple problems within the general topic of MRI sequences and their development. Modern sequences are complex and specialized, yet similar in many details. A sequence is comprised of many methods that may also be used by other sequences. Thus, the design process needs to be flexible, extensible, and amenable to reusable module generation. Moreover, the driver logic for the proposed sequences must be simple and easily maintainable to enable implementation on as many devices as possible. Therefore, as the connecting link, the sequence definition data structure that the driver uses to derive hardware events needs to provide a very simple interface for the driver while not diminishing the capability of realizing all concepts of modern MRI. Despite the goal of vendor independence, it is also important that the sequences are well-integrated into the standard workflow of the respective MRI scanner. It is therefore necessary that the sequence definition data structure has to be intrinsically adaptable to protocol changes and available hardware features.

An overview of the methods proposed in this work is given in Figure 1.

The first part elaborates the module definition approach. The second part of this section explains the sequence definition data structure that is transfered to the scanner. Finally, some implementation details of the hardware driver are provided.

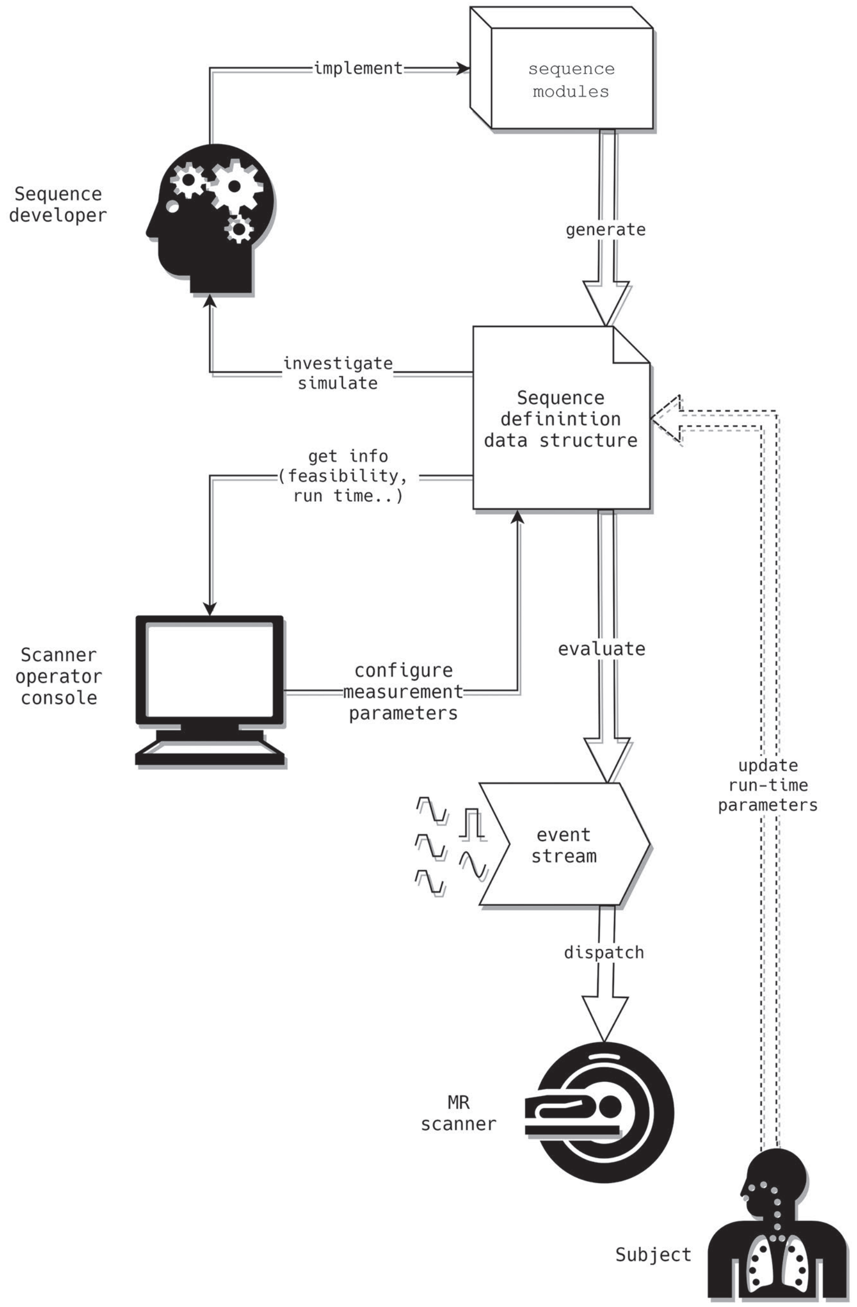

FIGURE 1 An overview of the main methods, data structures and interactions described in this paper. The sequence developer defines modules, called blueprints, in an intuitive high-level language. The sequence modules are used to programmatically generate the sequence definition data structure. This data structure describes the sequence in a self-contained and efficiently evaluable way. It provides feedback to the sequence developer to interactively aid the sequence design and development process. The data structure is portable and vendor-independent. Once the data structure is transfered to the MRI device, the scanner operator chooses protocol settings for the measurement and receives feedback, such as valid protocol parameter ranges and sequence run time. After preparation, the data structure can be evaluated to generate a stream of event instructions which is translated to vendor-specific hardware events. While the sequence is running, parameters that impact pulse shapes and timing change. This could be loop counters in the simplest case, but external parameters updates, such as patient position information or breathing state are also supported

2.1 | Sequence design process

Many sequence development frameworks require writing direct C++ code[3] or attempt to ease the process by requiring the developer to interact with a sequence tree.[3-5] The proposed sequence definition data structure is flexibly generated based on an extensible set of definition rules.

Tree structures and parameter connectivity graphs are not an elegant choice for direct interaction with the goal making a modification because it is difficult to keep an overview of them. The novelty of the approach used in the presented work is that complicated relationships do not have to be defined by the developer directly. Instead, the sequence definition data structure is generated programmatically as the result of reusable modules and physics-based rules that are defined by the sequence developer. Such definitions and rules are bundled in module descriptions that are called blueprints. The following paragraphs show some of the functionality that these blueprints may define. This assembly is performed in a high-level programmatic environment which is open for extensions. An overview of the sequence assembly process is given in Figure 2.

A blueprint is a set of definitions and rules which can be stored and referenced by other blueprints. Each blueprint represents a MR sequence technique or an isolated part of it. Blueprints are the MR sequence modules or building blocks of this work.

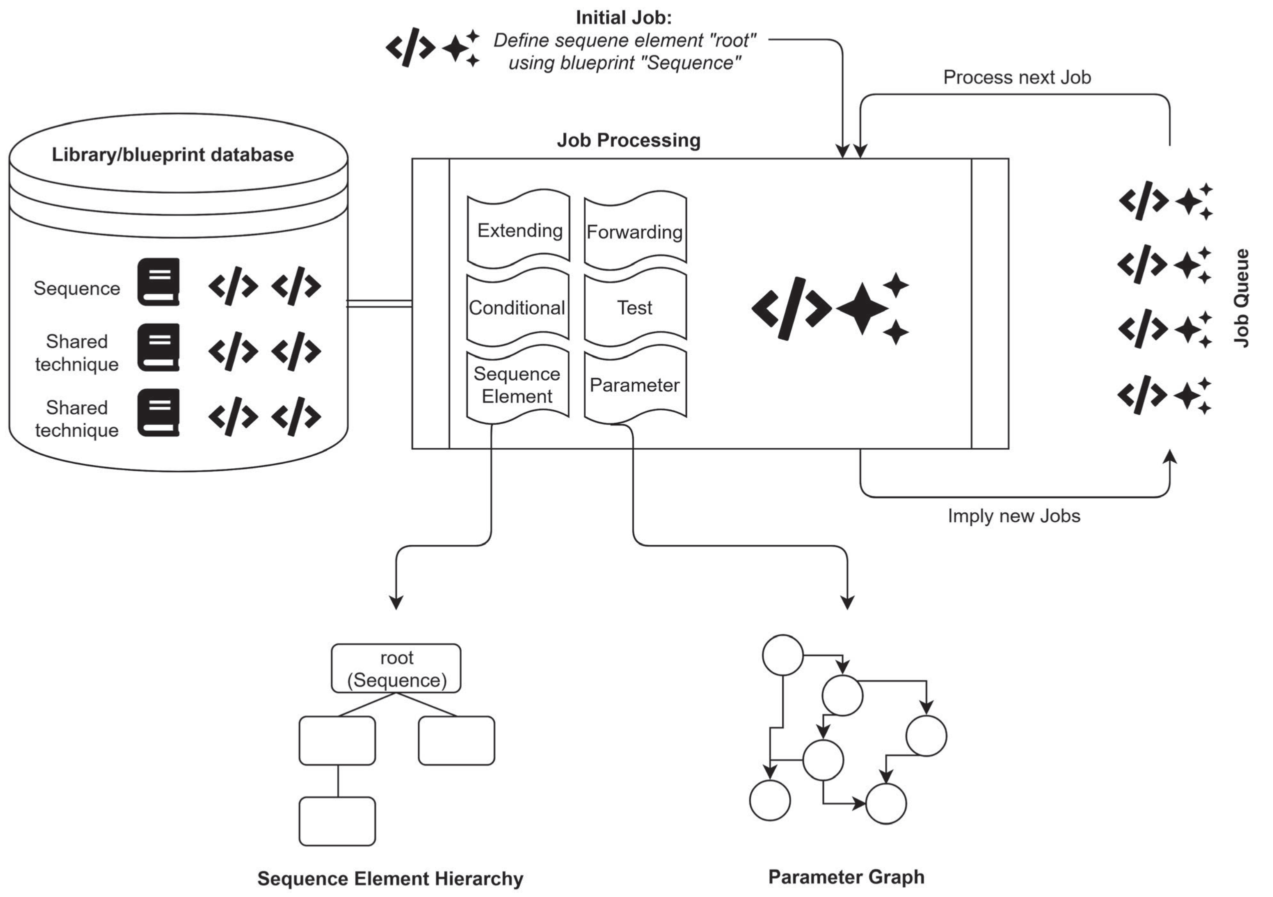

FIGURE 2 An overview of the sequence assembly process. The library contains blueprints, symbolized by the book icons. Each blueprint contains multiple definitions, symbolized by the angle brackets icon. The process is initialized with a sequence element definition job referencing a blueprint within the sequence library. Jobs are symbolized by the stars icon next to the angle brackets icon. Jobs are processed according to type-specific code. More details about the type-specific code can be found in the Supporting Information or in the help section of the sequence development software.[6] During processing, a job may imply further jobs which are then added to a queue to be processed later. Jobs of type sequence element and parameter generate parts of the sequence definition data structure as byproducts. The combination of sequence element hierarchy and parameter graph is the sequence definition data structure. Examples of sequences can be found in the sequence development software[6]

The core of the assembly process is a processing queue of definitions that is filled recursively, starting with the definition of the sequence's main blueprint. Definitions that are instantiated and added to the queue are called definition jobs. The sequence element hierarchy and the parameter graph are byproducts of the assembly process. Definition jobs can be postponed or updated based on properties and changes in the sequence element hierarchy, the parameter graph or the set of already processed definition jobs. This updating mechanism adds flexibility to the assembly process. For example, the default flip angle of an RF pulse is set or removed automatically, whenever a different flip angle is removed or set. As another example, the connections of a parameter in the parameter graph can be updated whenever a gradient or RF pulse is added or removed from a part of the sequence element hierarchy. Details about definition types are found in the next paragraphs, in the help section of the software,[6] and in the Supporting Information.

2.2 | Sequence element definition

The proposed approach utilizes a tree structure to represent the hierarchical properties of MR sequences. Nodes of that tree are called sequence elements within this work. A blueprint can hold the instruction to attach a sequence element to another one. This attached sequence element can be associated with another blueprint, effectively defining a sequence subtree recursively. For instance, a combination of two trapezoidal gradient pulse sequence element definitions and an RF pulse sequence element definition can form a refocused slice-selective excitation blueprint. This blueprint is then used itself as a sequence element definition and combined with a phase encoding and readout to yield a line readout blueprint. Sequence elements can be defined over multiple hierarchical levels, making it possible to inject functionality, such as a specific RF pulse, into different module that further configures itself based on that injected module. In practice, this allows the definition of blueprints that specify the overall structure of a sequence, containing all loop logic and timing between excitation, preparation and acquisition, while other modules specify the behavior of single sequence elements in an isolated manner. The hierarchy needs to be fully resolved prior to transferring the sequence to the scanner, which is done programmatically based on the blueprints that relate to a given sequence.

2.3 | Parameter dependency definitions

Sequence elements can possess properties or parameters, such as a start time or pulse amplitude. Blueprints can contain parameter definitions that specify a calculation instruction for a variable and its required input paths. It is not required to define parameters in a specific order. For instance, the start time of a pulse may be defined through the duration of another pulse, but the amplitude of the latter pulse might be determined by the amplitude of the prior pulse. None of the two pulses can be calculated without partial calculation of the other one. Since the sequence assembly step is separate from the definition process, this is not a problem in the presented platform. Circular definitions are identified at the time of sequence assembly, during the sequence development process.

2.4 | Simple algebraic equations

Some physical properties are related through simple algebraic relations. For example, bandwidth, duration and time-bandwidth-product of an RF pulse or ramp durations, plateau duration, and full pulse duration of a trapezoidal gradient pulse. During the sequence assembly process, once sufficient variables of the equation are resolved, the equation can be used to find an explicit formulation of the unknowns and also append them to the graph. Most notably, any two of the common timing parameters, start time, end time, duration, and center time, may be defined to have all of them available for further definitions. This allows the sequence developer to define a technique that is flexibly reusable, since the parameter calculation graph is resolved in different ways depending on its context. Non-existence or contradiction of such definitions are identified at the time of sequence assembly, during the sequence development process.

2.5 | Contextual parameter definitions

Some physical properties are best derived by using information about all sequence elements that are present in the sequence element (sub)tree. For example, to calculate the time of a spin echo or gradient echo, all pulses of a corresponding time interval have to be considered. This feature is realized programmatically by collecting all sequence elements of a certain type, e.g. gradient pulses, to then compose a formula based on their defining parameters.

2.6 | Tests

A parameter value assertion can be declared as a test and is then added to a test suite during the sequence assembly process. Such an assertion could be non-negativity of a pulse duration or conformity with hardware restrictions. Finally, all tests are combined into one parameter that asserts, that the sequence is parameterized properly. This test suite can then be evaluated by the scanner operator on the scanner console, either directly or indirectly while setting the measurement protocol. Thus, the sequence developer can decide which conditions have to be met in order for a module to function properly. Improper settings are aggregated and presented during the development process and prior to executing the sequence.

2.7 | Further rules and definitions

The assembly process produces the portable and self-contained sequence definition data structure which is explained in the next section. The blueprint concepts are open for extension. As long as that end-product of the assembly step does not need adaptation, the hardware drivers do not need to be modified, even when further complex definition logic is introduced.

2.8 | User interaction

The task of a sequence developer within the proposed framework is to produce desired MRI sequence functionality, represented as JSON objects that fully define suitable behavior. The user is given the option to either write and edit the JSON object directly, or to use graphical aids to generate the objects interactively. When working on an MRI sequence, information about the current state of the sequence is presented to the user to ease the development process. In case of a parameter dependency definition, this would be contextual information about parameters that are potential candidates as inputs, and a live evaluation of the calculation instruction based on the current state of the parameter graph. Demos are presented in the help section of the software.[6]

2.9 | Sequence definition data structure

This section explains how the rules and definitions of the previous sections can be used to create a self-contained and portable, vendor-independent description of the sequence that can be configured and run at the scanner. A JSON schema that defines the data structure can be found in the Supporting Information.

It is common practice to express pulse sequence elements in a tree structure.[3-5] This is a natural approach for two reasons: Simple elements are often combined to form more complex ones, and evaluation loop structures are nested and act only on partial sequence element groups. The proposed approach also uses such a structure. The novel addition is, that all calculations that are relevant to the sequence are not orchestrated by this data structure or coupled to it, but rather realized through a flow graph structure. A further difference with previous approaches is that leaves of the tree structure are not sorted chronologically, which impacts the algorithm required to run the sequence. This will be explained in a later section.

2.10 | Sequence element hierarchy

The sequence element hierarchy (see Figure 3) is a tree structure in which each node represents a sequence element. The driver uses the sequence element hierarchy and the special properties of the nodes to derive which hardware events have to be dispatched in which order. All calculations including timing and pulse shapes are then performed in the parameter graph which will be elaborated in the next section.

2.10.1 | Basic sequence element

Ultimately, the driver needs to dispatch hardware events. Basic sequence elements correspond to one hardware event each. In most common cases and throughout the examples in this paper, there are only three types of basic sequence elements: Gradient pulses, RF pulses, and ADC events. However, the concept is open for extension. The three basic types only differ in their set of defining parameters, and further basic types can be added, such as elastography hardware events, table movement, or triggers. However, the vendor-specific driver has to be extended to support new basic types.

2.10.2 | Loop sequence element

A loop sequence element within the sequence element hierarchy implies that all subordinated sequence elements are to be repeated a given number of times, which can be determined by evaluating the parameter graph node that is associated with that loop sequence element's number of iterations. The parameter graph will be explained in the next section. For each iteration of the subordinated basic sequence elements, the parameter of the parameter graph that is associated with the counter of that loop sequence element node needs to be set to the appropriate value.

Reordering concepts, such as centric reordering, are realized as part of the parameter graph. Within that graph, a function or array that maps indices appropriately is implemented

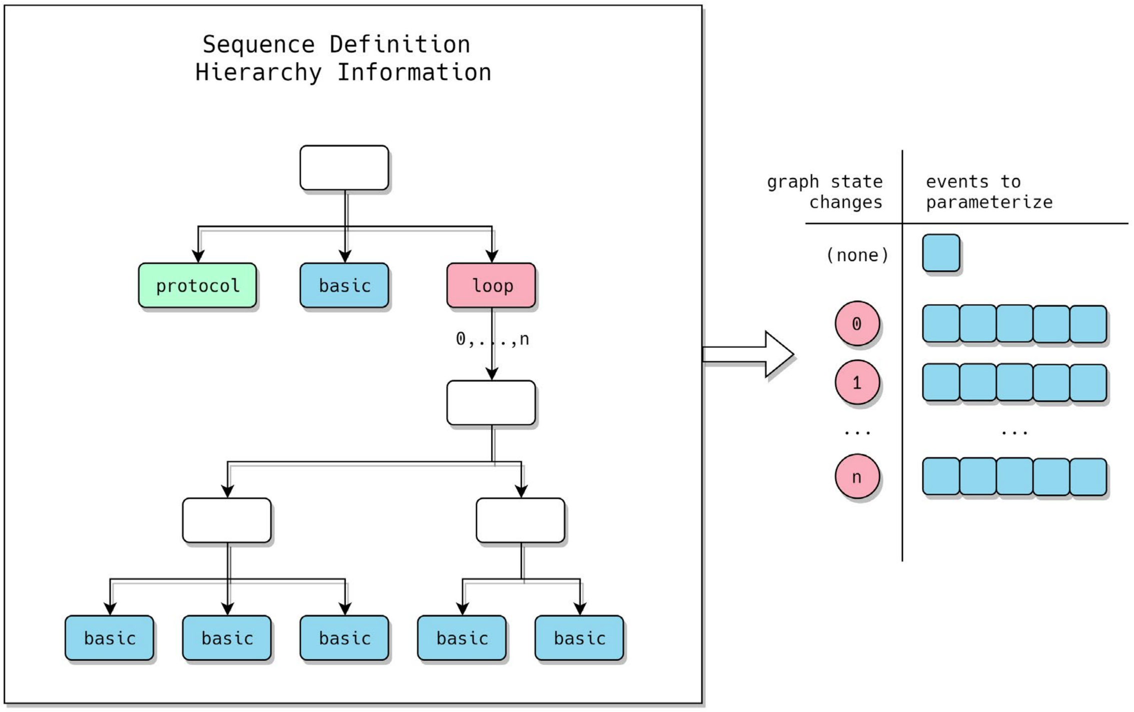

FIGURE 3 Sequence element hierarchy of an abstract sequence. Each node represents one sequence element. A basic sequence element corresponds to a hardware event, such as a gradient pulse. A loop sequence element implies a repetition of its subtree. The most straightforward way to run a sequence is to traverse the hierarchy depth-first and iterate at loop nodes while keeping track of the current loop counter values. For every basic sequence element that is visited, a corresponding event has to be dispatched. For that event, the loop nodes of the parameter graph (see Figure 4) need to be set correspondingly before calculation. An exemplary FLASH sequence element hierarchy is illustrated in Figure 6. All other sequence elements, such as the measurement protocol container node, do not affect the sequence tree traversal process

between the loop counter and the input parameters of the acquisition module. This way, reorderings can be exchanged easily.

2.10.3 | All other sequence elements

All other sequence elements are of no algorithmic relevance. They may be used as containers for system or protocol parameters corresponding with a convention to ease the development process, or as a group to ease the development process. Properties of these sequence elements can be connected to other sequence elements through the parameter graph that is explained in the next section.

2.11 | Parameter calculation graph

The portable sequence definition data structure contains a set of parameters. The parameters are arranged in an acyclic, directed calculation flow graph (see Figure 4). Each node has calculation instructions attached to it that may only use parameter values of the incoming edges. The calculation instructions can be of arbitrary type and the parameter values may consist of high-level data structures, such as pulse shapes. Within this work, this logic is implemented using the Lua scripting language.[7]

The acyclic nature of the parameter graph needs to be ensured during the module definition steps. Cycles and parameter nodes with missing inputs can be identified by recursively tracing the inputs of any parameter of the graph, which happens at their first calculation attempt. All parameters that are part of a cycle and parameters that cannot be calculated due to missing inputs are then communicated to the sequence developer during the module definition steps.

The whole parameter graph is guaranteed to yield consistent results, as long as each parameter calculation node yields a deterministic result based on its inputs, and as long as a valid calculation order is pursued. There are multiple possible calculation orders, which can be exploited to avoid calculations that are not needed for for specific tasks that do not require all parameters to be calculated. Choosing a specific calculation order may be beneficial during unit tests or binary search of valid protocol parameter ranges.

2.11.1 | Efficient calculation

For calculation efficiency, a parameter value cache is employed. Whenever a parameter is requested, its value is retrieved from the cache if it is valid, or calculated otherwise. The calculation of a parameter node recursively requests values from its input nodes, which ensures that only calculations that are essential for the currently requested parameter are performed. It suffices to only cache one value per parameter node. This means that the required memory for the calculation cache is independent of loop structure and the number of overall loop iterations.

2.11.2 | Efficient updating

Certain parameter node values, such as loop counters and protocol settings, need to change before or during the execution

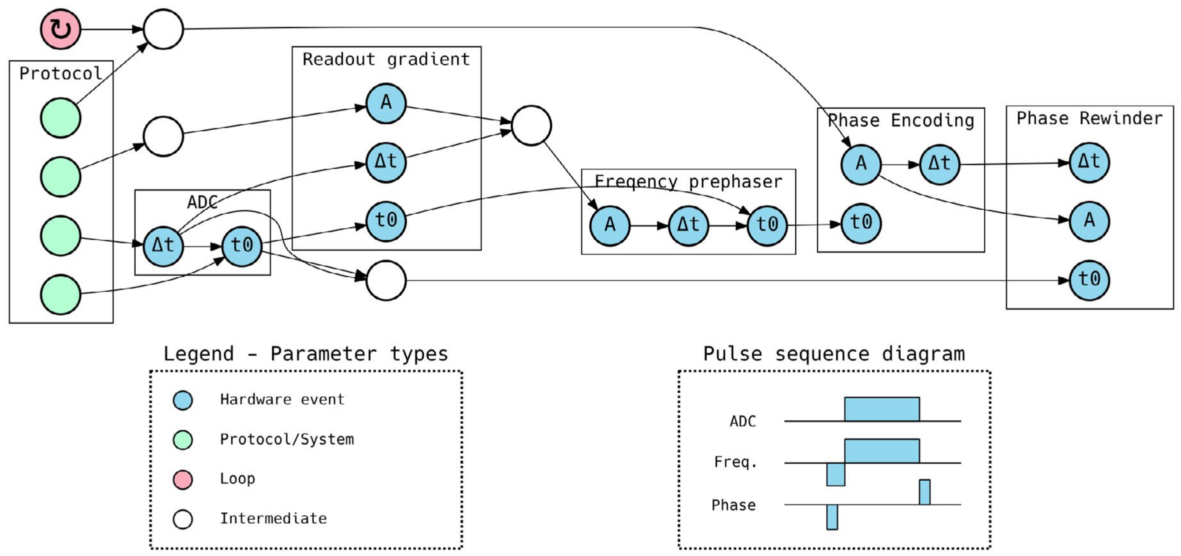

FIGURE 4 An illustration of a part of the calculation graph. Bounding boxes represent sequence elements. Contained parameters are associated with that sequence element. Gradient pulses are assumed to be rectangular with amplitude A, duration dt, and start time t0. The ADC is simplified to only require start time and duration. Parameters can depend on each other, beyond sequence element, or module boundaries. Certain calculations are unaffected by the loop counter change and can thus be cached and only need to be calculated once. The parameter graph of a complete and realistic sequence is illustrated in Figure 6

of the sequence. All of the parameter graph's values are deterministic when all of its nodes contain deterministic calculation routines. When a change in such a parameter value is instructed, it suffices to clear or update the cache recursively for all dependent parameter nodes. This means that no computation is performed at the time of parameter change. Calculations only occur when a parameter is requested.

The logic of using the observer pattern for parameter or attribute calculation has been introduced previously.[5] The approach presented here enhances this idea by only performing calculations when they are requested, rather than updating all dependent parameters each time a change has occurred. The benefits of this variation are explained in the discussion.

2.12 | Evaluating the sequence definition data structure

The sequence definition data structure can be used to directly calculate sequence properties such as run time through the parameter graph and thereby provide feedback while the scanner operator determines the acquisition-specific settings. Afterward, the sequence definition data structure generates the corresponding stream of hardware events.

The following paragraphs explain how the stream of events is calculated for a sequence that does not foresee external feedback during its runtime. External feedback that merely influences pulse shapes does not affect the validity of this approach, but external feedback that changes the timing of the pulses requires a slightly modified approach which will be explained in the discussion section of this work.

Running a sequence, i.e. to generate the stream of hardware events, involves two steps. In the first step, the sequence element hierarchy is analyzed to identify how often and with which loop counter value states the basic sequence elements have to be evaluated. The second step then applies the loop counter value changes to the parameter graph and then retrieves the defining parameters of the basic sequence elements. The retrieved parameters values completely specify basic sequence elements which are then used by the scanner driver to dispatch corresponding hardware events.

2.12.1 | Extracting counter values and basic sequence element iterations

The tree structure of the proposed approach is not required to contain temporal information. Pulses of different subtrees can be interwoven temporally. Consequently, all start times of all hardware events generally have to be calculated during the first step. To remove the need to calculate the timing of every single pulse beforehand, the sequence developer can set subtrees to be *atomic*, meaning that it may only overlap partially with its directly neighboring atomics. For example, an atomic could either be a pulse of basic sequence element, a preparation procedure, or a full iteration of a line acquisition. The timing of MR events within each atomic may change from loop counter to loop counter.

After all atomics are determined, their start times are calculated. Atomics that are subordinated to a loop are registered multiple times with their respective set of loop counters. Start time parameters are calculated through the parameter graph structure and as such processed with no unnecessary overhead and redundancy. Using this traversal approach, it is not guaranteed that all atomics are registered in chronological order. Therefore, they need to be sorted before the next step to provide a chronological hardware event stream. The number of atomic executions is usually of the same order as the number of acquisition lines. As such, the sorting does not introduce significant computational overhead. Since pulses and timing within an atomic are calculated during run-time, overlap violations may occur unless test routines are implemented as part of the parameter graph in a prior step. An alternative evaluation strategy that circumvents this issue is explained in the discussion section. All steps are performed on the scanner hardware.

Besides loop length and atomic start time parameters, no calculations are performed in this step. In particular, no pulse shapes are calculated at this time. Note that loop lengths and atomic start times can depend on loop counters. The processing of these calculation is performed through the parameter graph, equivalent to the calculations of pulse shapes or other parameters.

2.12.2 | Running atomic executions

The result of the first step is an ordered list of start times for atomic sequence element hierarchy subtrees together with their associated loop counter values. The memory requirements of this is roughly of the same order as the header information of all ADC events within the sequence. The second step calculates the corresponding hardware events in chronological order.

To calculate the hardware event descriptions of an atomic execution's basic sequence elements, the parameter graph needs to be set to the appropriate state by setting loop counter variables in the parameter graph. Afterward, all defining parameters of the atomic's subordinated basic sequence elements can be retrieved through the parameter graph.

Since intermediate results of the calculations are cached, only the differences between the parameter calculation graph states are calculated.

These basic sequence element parameterizations ideally describe the hardware events directly. They are then passed on to the sequence driver to perform the vendor-dependent interfacing steps. Gradient shapes are stored as time-amplitude pairs in the base data structures, assuming linear interpolation between the points. There is no distinction between trapezoidal and non-trapezoidal pulses in the base data structures. The vendor-dependent interface can either process gradient pulses with shapes containing four or less points in a specialized manner, or merge pulses that are contained in a given time span, depending on hardware restrictions and performance. RF pulse shapes are generally cached and mirrored in the vendor-specific interface.

2.13 | Implementation

The sequence evaluation logic is implemented in Lua.[7] Lua is a lightweight and high-performance scripting language that was designed to be embedded into C/C++ code flexibly with minimal requirements. The sequence definition data structure can be transferred directly onto the scanner without a compilation or hardware-specific preparation step. The vendor-specific driver implementation sets and gets hardware properties, sequence properties, and measurement protocol parameters. All evaluation logic beyond this is part of the vendor-independent Lua codebase. Lua can be extended with C/C++ code easily, but this was not necessary throughout this work, since we have not run into performance issues or limitations of Lua.

After the sequence is fully prepared, the driver requests atomic executions sequentially through a pulse event stream. The atomic execution structures can either be split according to the basic sequence elements, or provided as one pulse per channel chunks (RF, physical gradient directions), as time-value pairs, that are scaled in time and amplitude to match the driver requirements. It also contains further information that is directly required to configure the hardware events such as the ADC measurement header and RF/ADC frequency or phase modulations. The ADC measurement headers are provided in the vendor-independent ISMRMRD[8] format, which needs to be mapped to vendor-specific acquisition properties if the vendor-specific reconstruction is to be used.

Sequences can alternatively be exported to the Pulseq format[1] to be then run on devices for which no other dedicated driver exists. However, the exported sequences cannot be further configured since they contain static pulse shape definitions.

To embed the sequence evaluation logic, it is necessary to provide functionality for loading the sequence file, and to transform a stream of basic hardware event descriptions to actual events, which can be done via a C/C++ API or by directly interfacing with Lua.

An optional GUI for modifying sequence protocol parameters in addition to the vendor protocol setting interface was also implemented, which can be run directly on the scanner operator console. The GUI is presented in the software release.[6]

2.14 | Software

The sequence development software is released.[6] It requires a backend server that performs sequence assembly and provides a communication interface to the sequence development frontend. The frontend that can be run on a contemporary browser. The frontend contains the same Lua codebase that is used in the MR hardware driver to calculate MR events and configure the measurement protocol. Assembled sequences can be exported in seqeunce definition data structure form, and alternatively as read-only variants to be run without the backend server, pertaining information about blueprints and detailed information about the steps that were performed during sequence assembly. It is possible to use the tool, even in read-only-mode, to export a parameterized pulse sequence to the Pulseq format[1] and to run custom Lua scripts to interactively explore the sequence and Lua core capabilities.

The source of a reference implementation for evaluating the sequence definition data structure is publicly available. The file format specification for the sequence definition data structure can be found in the Supporting Information. The source for a C++ program that interfaces with the Lua code of the frontend is also provided, such that custom scripts that are written within using the frontend can be evaluated without the use of a browser. The Lua interpreter is the only external dependency of the C++ interface. The sequence development backend server and frontend are provided as binaries.

2.15 | Example

The techniques described above were used to implement and run a 2D FLASH[9] sequence as an illustrative example.

The 2D FLASH sequence consists of one base loop that repeats the line acquisition sequence element, offset by the repetition time. This reduces the effort of the first step of the sequence running process to a mere accumulation of repetition times. The line acquisition has three subordinated sequence elements, which are set to be the atomics of the sequence, for excitation, phase/frequency encoding, and spoiling. Sequence element hierarchy and parameter calculation graph are illustrated in Figures 5 and 6. Further more complex examples can be found in the software.[6] A simplified abstract example can be found in the Supporting Information.

2.16 | Sequence implementation

To assess the feasibility of the approach, various sequences were implemented. The result section showcases 2D FLASH,[9] EPI,[10] SE-EPI, and RARE[11] sequences. During the design process of those sequences, blueprints were implemented to be flexibly reusable. For instance, the RARE sequence did not require the implementation of any module that directly relates gradient, RF, ADC, protocol, system or k-space calculation components. It sufficed to use parts of the 2D FLASH and SE-EPI sequence and connect them in an alternative loop and refocusing strategy. Many sequences contain GRAPPA functionality, which all use the same GRAPPA blueprint. The modules that connect device properties and common protocol

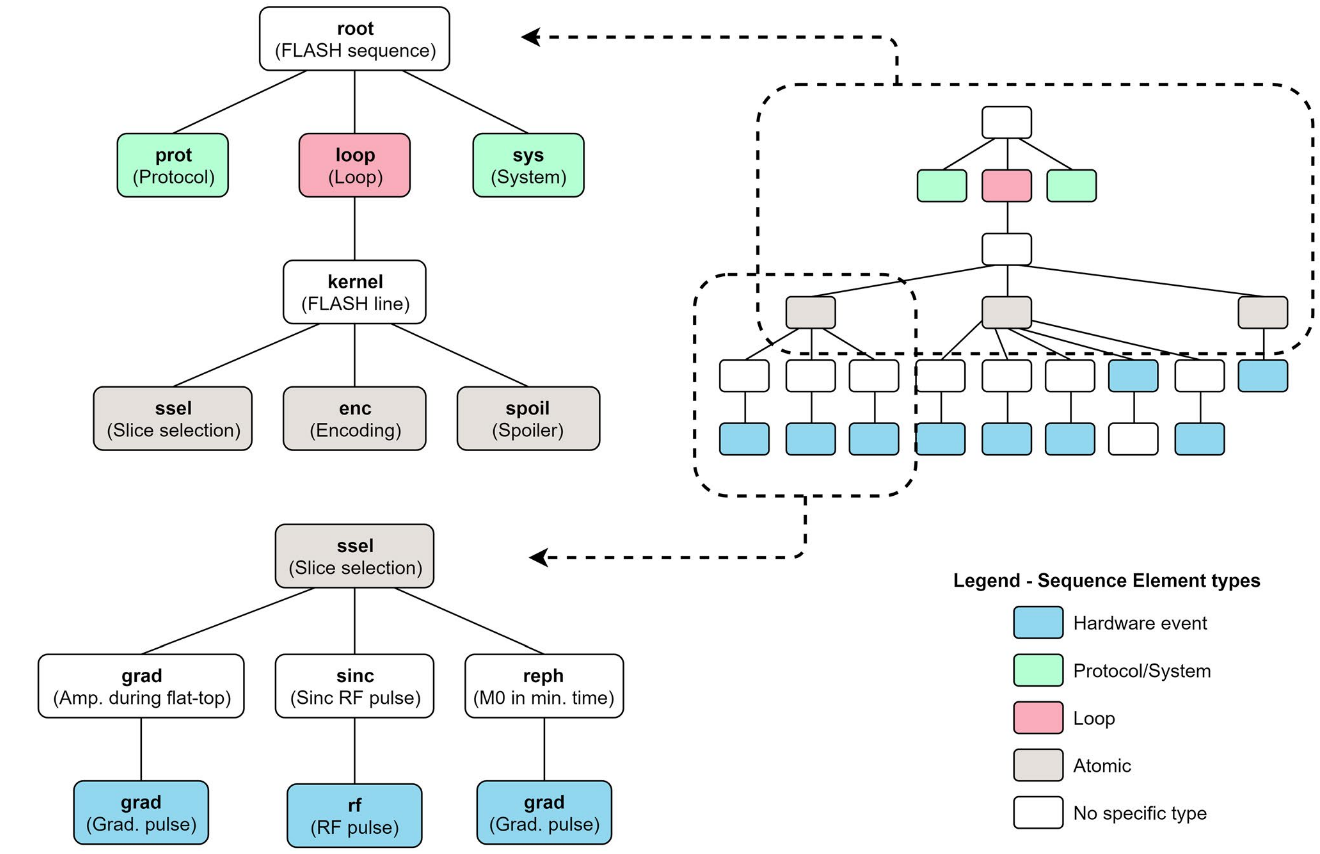

FIGURE 5 Illustration of the sequence hierarchy of a simple 2D FLASH sequence. Protocol and system sequence elements (green) are subordinated to the sequence root. The single loop (red) has one subordinated sequence element that represents a single FLASH line excitation, readout, and spoiling. These three parts are subordinated and set as the atomic sequence elements of the sequence. They each define children with lower level logic down to base hardware events (blue). Note that no information about the temporal order is part of the sequence hierarchy. A demo FLASH sequence can be explored interactively in the software[6]

settings to the rest of the sequence were also shared, such that no extra work was required to provide a seamless configuration experience to the scanner operator. Sequences and their implementation details can be explored interactively in the software.[6]

2.17 | Phantom and in vivo measurements

Images were acquired using a standard commercial MRI system, operating at 3T (MAGNETOM Skyra, Siemens Healthineers, Erlangen, Germany)

Sequence protocol parameters are as follows: FLASH[9]: TR = 100 ms, TE = 5 ms, flip angle = 30°, bandwidth = 500 Hz/px, fov = $(256\,\text{mm})^2$, matrix size = 256^2. single-shot EPI[10]: TE = 65 ms, flip angle = 90°, bandwidth = 2000 Hz/px, fov = $(256\text{ mm})^2$, matrix size = 128^2. Single-shot spin echo EPI: Identical to EPI, except for TE = 120 ms. RARE[11] TR=4 s, TE = 43 ms, flip angles = 90° and 180°, bandwidth = 1000 Hz/px, lines/shot = 8, fov = $(256\,\text{mm})^2$, matrix size = 256^2.

3 | RESULTS

3.1 | Phantom and in vivo measurements

The framework was used to implement a set of common sequences. Measurements acquired with these sequences were then compared to their product sequence counterparts in Figures 7 and 8. Protocol parameters are listed in Section 2.16.

The images are visually similar. Some image artifacts are different, most notably in the EPI and SE-EPI sequences. The images are further discussed in Section 4.1.

4 | DISCUSSION

The presented work introduced a novel MR sequence definition data structure containing a parameter graph and sequence hierarchy. This data structure and its interaction concepts provide an easy interface to the driver. The sequences are still completely configurable, even after being ported onto

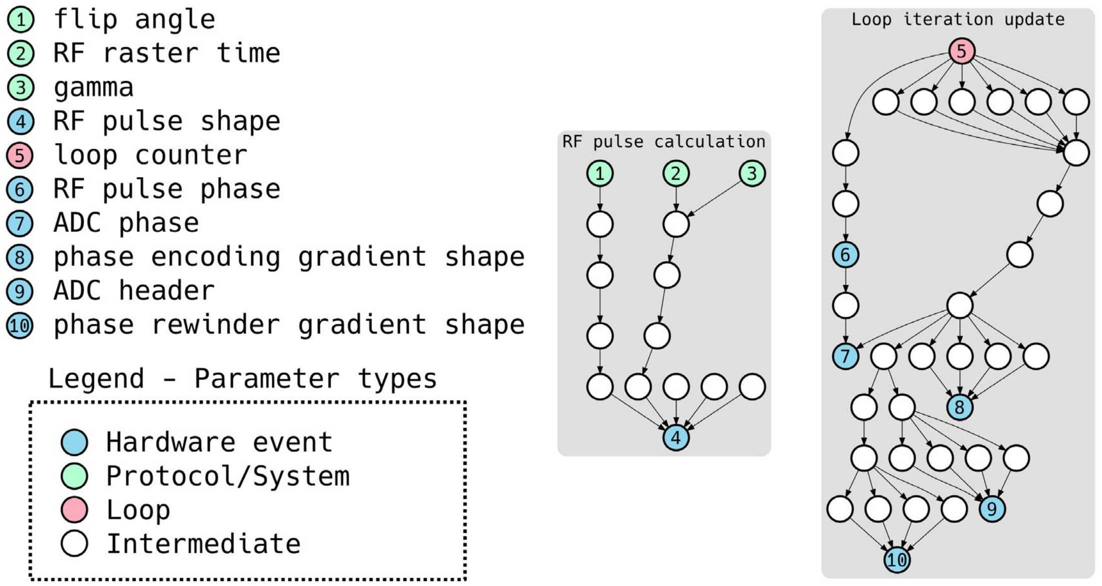

FIGURE 6 Illustration of a part of the sequence definition data structure's parameter graph of a 2D FLASH sequence. Each node corresponds to one parameter which can be calculated deterministically, using parameter values of connected input parameters. The arrow direction reflects the input/output relationship. Whenever a loop counter is changed, only a small subset of parameters need recalculation, illustrated by the right cluster box. Some parameters that are affected by a loop counter change, such as phase encoding gradient amplitude, do not result in a recalculation of other parameters of the affected sequence element, such as phase encoding gradient start time. Their calculation is triggered individually. The RF pulse shape calculation is highly independent of most other calculations of the sequence, illustrated by the left cluster box. If none of the affected input protocol and system parameters change, this calculation is never performed a second time. A demo FLASH sequence can be explored interactively in the software[6]

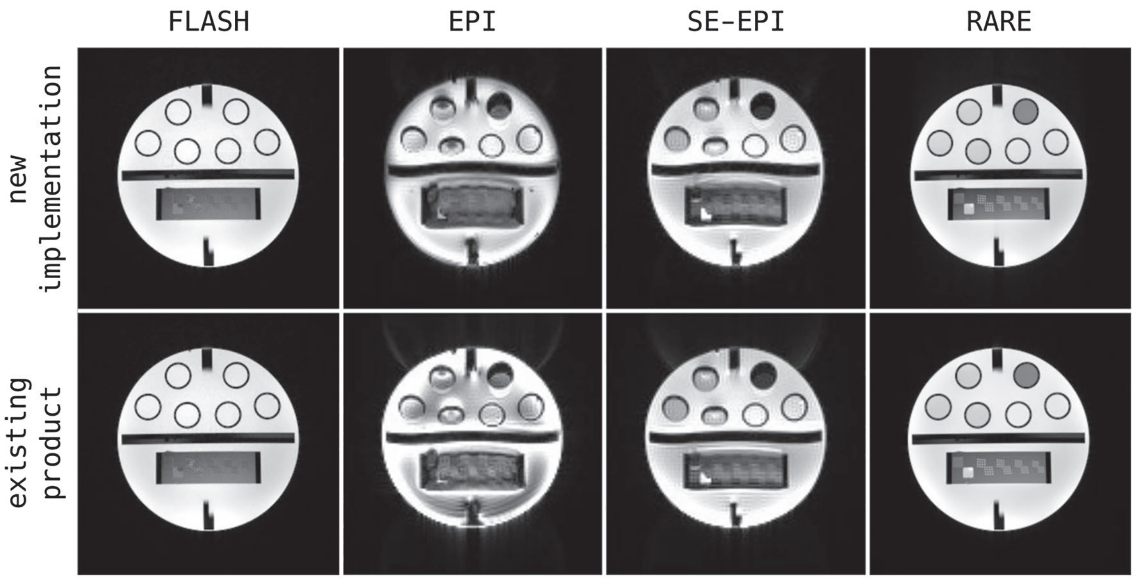

FIGURE 7 Comparison of images acquried with sequences implemented within the proposed framework and their counterparts acquired with sequences provided by the manufacturer. The images were reconstructed on the manufacturer's hardware using the manufacturer's reconstruction framework. Measurement protocol parameters are listed in Section 2.16

the scanner, without requiring any additional hardware or external preprocessing steps. The strategy for all calculations is based on a graph structure that ensures the shortest possible path and thus optimal calculation efficiency. The sequences are generated within a high-level programming framework that spares the sequence developers from interacting with the low-level structures, but rather allows them to combine rules and define dependencies in a way that is natural to the sequence design process.

It is a matter of taste what an inutitive sequence definition workflow should be comprised of. Furthermore, sequence development tasks can differ greatly in their

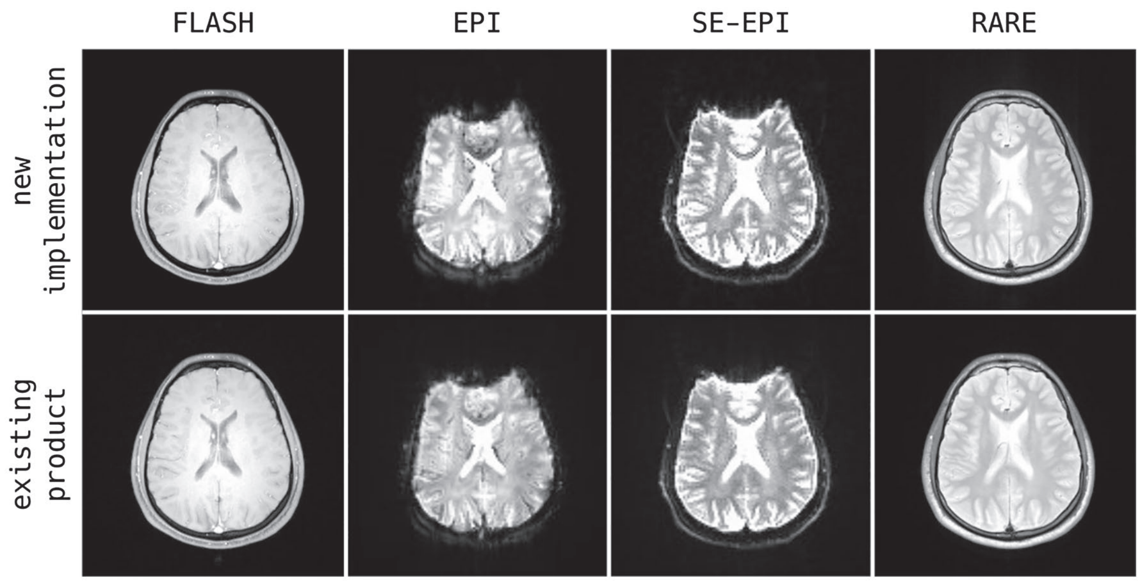

FIGURE 8 Comparison of further images acquired with sequences implemented within the proposed framework and their counterparts acquired with sequences provided by the manufacturer. The sequences are identical to those of Figure 7, except for an added fat saturation pulse within the EPI and SE-EPI sequences. Measurement protocol parameters are listed in Section 2.16

underlying thought process. The design process of isolated excitation module is very different from that of a loop and reordering strategy. The presented approach offers a concept of bundling such design logic in reusable modules that capture the essence of such thought processes. How these modules are created, modified, and debugged by the user in an intuitive way are a matter of further research. In particular, we do not propose that a sequence developer interacts directly with the parameter graph structure.

4.1 | Phantom and in vivo measurements

The images acquired with sequences of the proposed framework are justifiable alternatives to those provided by the manufacturer. The images are visually similar. Objective comparisons with SNR measurements and joint histograms were omitted because mimicking product sequences was not a goal of this project. Differences were due to deliberate choices made during the implementation process. For example, the manufacturer version of the EPI sequence that was used employs ramp sampling, while the sequence developed using the proposed framework does not. This could however be implemented. The artifact differences of the EPI and SE-EPI sequences can be explained with the employment of ramp sampling and the implied different echo spacing and gradient amplitudes.

4.2 | Features of the proposed approach

The following sections focus on the features that are enabled by the evaluation data structure. These features are impossible or hard to realize without the proposed graph structure.

4.2.1 | Quick random access pulse calculation during development

The pulse calculations of most MRI frameworks are required or assumed to be run strictly sequentially, following a single, dedicated, full MRI sequence calculation routine. The proposed framework was designed to not have that restriction. When only small parts of the sequence are of interest during the sequence development process, such as the pulses during a single repetition time, then only the elements contained in that timespan are calculated or refreshed. This provides instant visual feedback when investigating the effect of a protocol parameter or a structural change, when plotted as a pulse sequence diagram. This is enabled by the fact that all calculations are deterministic and cached, and thus either stay valid or are efficiently updated after a modification.

4.2.2 | Valid protocol parameter ranges

The range of physically feasible measurement protocols is often of interest and non-trivial to determine. Protocol parameters are often defined individually and the scanner operator receives feedback in the form of valid ranges for a single parameter of interest.

Tests that ensure physical feasibility are added to the parameter graph. To investigate a protocol parameter value for feasibility, the protocol parameter can be set correspondingly, followed by evaluating the test parameters that then then ensure that the protocol parameter choice is valid. This process is accelerated by caching of intermediate results which is intrinsic to the graph approach - parameters that do not depend on the protocol parameter in question are only calculated once. It can be further streamlined by prioritizing tests that failed in a previous attempt. This ensures the shortest calculation path for the tests that are likely to fail.

This approach requires test results to be independent of parameters that change during execution, such as loop counters. Invalid parameters that depend on specific loop counter values can only be identified when the graph is set to a failing loop counter configuration. Iterating through all loop counter configurations essentially calculates all hardware events and is thus computationally expensive and a poor choice for testing many potential candidate protocol settings. Performing a full test run can be triggered by the scanner operator or sequence developer, and a failing full test run that depends on specific loop counters is an indicator of poor test design.

The graph approach does not provide a full solution to finding the complete set of feasible protocols. Arbitrary parameter calculations cannot be inverted and arbitrary parameters do not belong to a finite-dimensional or convex space. This issue is independent of sequence implementation or framework. The general mathematical formulation has no directly applicable, simple solution. But a certain degree of back-propagation for selected parameters is reasonable and may lead to insight about the hidden protocol parameter constraints. In particular, many timing relations that are omnipresent in sequences correspond to systems of linear equations which can be analyzed confidently. This insight could then lead to new approaches in sequence optimization or machine learning.

In any case, the forward problem of parameter validity check is streamlined by exploiting the aforementioned properties of the graph and caching. This results in faster checks and feedback which may be critical in a clinical setup.

Finding a valid protocol when given an invalid protocol state is an essential problem of clinical routine. For this task, the graph structure can help automate the process by reducing the set of parameters that need to be considered for a change to those that cause the breaking tests. The sequence developer can add protocol resolution logic as vendor-independent core logic to the Lua code base.

4.2.3 | Parameter updates during sequence execution

The parameter calculation graph can receive additional feedback or modification instructions during runtime, such as slice settings generated by a tracking system. Any part of the graph, including all pulse shape calculations, may depend on those additional changing parameters.

If the externally updated parameter does not influence start time parameters of atomic sequence elements or hardware events, then the change is directly supported by the approach explained above, without the need for any modification to the sequence definition data structure evaluation algorithms. The parameter update recursively invalidates or updates the cache of all dependent parameters. Upon the next parameter retrieval request, the non-cached parameter values will be calculated based on the new state of the parameter calculation graph.

If the externally updated parameter influences a start time parameter of an atomic sequence element or hardware event, or a loop length parameter, then it must be considered that order and number of all hardware events is not known at the time the measurement is started. This challenge can be mitigated by adding a sequence element hierarchy traversal order to the sequence definition data structure. This additional order removes the need to calculate all start times as an initial step after the measurement is started, at the cost of increased development effort.

With the addition of this feature, the need to keep a list of all atomic execution start times and corresponding loop counters in memory, which has a size of the order of all image acquisition lines, would vanish.

5 | CONCLUSION

The sequence definition data structure has a clear protocol parameter structure and hardware event stream interface, and thus hardware-specific driver implementations can be slim, and roughly require the same steps to implement and support as related vendor-independent pulse sequence description approaches.[1,2] This does not diminish its flexibility because all sequence logic is resolved as part of the parameter calculation flow graph, and many remaining features can be implemented in Lua without requiring compilation. The resulting sequences have an efficient logic for handling protocol changes by the scanner operator by only performing affected calculations in their most efficient way. Informational feedback, such as run time, and sequence unit test results are provided to the scanner operator.

Any device with a compatible driver can evaluate the sequence, even if the development process changes. Other open pulse sequence design tools require writing raw C++ code,[3,4] interacting with complex evaluation structure,[4,5] or leave the process of writing the sequence evaluation structure to the developer altogether.[1,2] The proposed framework allows MRI techniques to be specified on a higher level, such that the sequence developer can directly define relations of physical models, instead of having to specify hardware events, step by step, from protocol parameter to pulse shape. The evaluation structure contains no raw pulse shapes and thus scales well with sequence complexity since its memory and calculation footprint scale well, as illustrated by the demo tool.

The proposed approach is a step toward reproducible sequence descriptions that may enhance comparability of MRI measurements in the context of multi-center clinical studies.

ACKNOWLEDGMENTS

The authors would like to thank Saulius Archipovas for consultancy and fruitful discussions.

ORCID

Cristoffer Cordes http://orcid.org/0000-0001-6389-4572

REFERENCES

1. Layton KJ, Kroboth S, Jia F, et al. Pulseq: a rapid and hardware-independent pulse sequence prototyping framework. *Magn Reson Med*. 2017;77:1544–1552.
2. Nielsen JF, Noll DC. TOPPE: a framework for rapid prototyping of MR pulse sequences. *Magn Reson Med*. 2018;79:3128–3134.
3. Jochimsen TH, von Mengershausen M. ODIN-object-oriented development interface for NMR. *J Magn Reson*. 2004;170:67–78.
4. Magland JF, Li C, Langham MC, Wehrli FW. Pulse sequence programming in a dynamic visual environment: sequenceTree. *Magn Reson Med*. 2016;75:257–265.
5. Stöcker T, Vahedipour K, Pflugfelder D, Shah NJ. High-performance computing MRI simulations. *Magn Reson Med*. 2010;64:186–193.
6. Cordes C. gammaSTAR—http://gamma-star.mevis.fraunhofer.de—https://ipfs.io/ipns/QmQYBinMaYbMFsSQ2TwrYEvKk5BBnan1EsjVcUdDnupYBW, 2019.
7. Ierusalimschy R, de Figueiredo LH, Filho WC. Lua—An extensible extension language. *Software: Practice Experience*. 1996;26:635–652.
8. Inati SJ, Naegele JD, Zwart NR, et al. ISMRM Raw data format: a proposed standard for MRI raw datasets. *Magn Reson Med*. 2017;77:411–421.
9. Haase A, Frahm J, Matthaei D, Hanicke W, Merboldt K-D. FLASH imaging. Rapid NMR imaging using low flip-angle pulses. *J Magn Reson (1969)*. 1986;67:258–266.
10. Mansfield P. Multi-planar image formation using NMR spin echoes. *J Phys C*. 1977;10:L55–L58.
11. Hennig J, Nauerth A, Friedburg H. RARE imaging: a fast imaging method for clinical MR. *Magn Reson Med*. 1986;3:823–833.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section.

How to cite this article: Cordes C, Konstandin S, Porter D, Günther M. Portable and platform-independent MR pulse sequence programs. *Magn Reson Med*. 2020;83:1277–1290. https://doi.org/10.1002/mrm.28020