A data and code model for reproducible research and executable papers

Konrad Hinsen

A data and code model for reproducible research and executable papers

Konrad Hinsen

Centre de Biophysique Moléculaire (UPR4301 CNRS)
Rue Charles Sadron 45071 Orléans Cedex 2
France

Synchrotron SOLEIL
Division Expériences
B.P. 48, Saint Aubin
91192 Gif-sur-Yvette Cedex
France

E-Mail: konrad.hinsen@cnrs-orleans.fr
`http://dirac.cnrs-orleans.fr/~hinsen/`

1 June 2011

Abstract

This proposal describes how data, program code, and presentation can be stored together in a single file suitable for electronic publication and permitting the reproduction of computational results.. Universality, efficiency, platform-independence, automated verifiability, and provenance tracking are the major design criteria. Existing and well-tested technology is used as much as possible, the two major building blocks being the Hierarchical Data Format for storage and the Java Virtual Machine for platform-independent code representation and secure execution.

1 Introduction

One of the hallmarks of science is reproducibility: a scientific study must be documented to the point that another researcher can follow the same steps and obtain the same results. Computational science currently falls short of this goal because the programs and the input data sets for a computation are rarely

published. Recent efforts to improve this situation by creating suitable tools and awareness of the problem in the scientific community have been conducted under the somewhat provocative label "reproducible research" [1]. Executable papers carry this idea one step further: a single published digital object should contain an article describing a scientific study and its results as well as all the data and program code necessary for repeating the calculations and visualizing the results. Furthermore, this digital object must be suitable for long-time archival and for inspection on a large variety of computing platforms, present and future.

The work described here concentrates on the infrastructure required for making computational research reproducible and publishable. It addresses the question of how data, code, and human-readable text can be combined and stored efficiently in a single file that can be used on current and future computer architectures. The proposal uses already available technology as much as possible, pointing out where shortcomings of these technologies are likely to require improvements in order to make the proposed executable paper format suitable for production use in a wide range of computational science domains. Key features of the proposed format are

- platform independence through virtual machine technology
- automatic verification of all computational results for which code is provided by the authors, and authentication by electronic signatures of the results for which no code is provided
- protection against viruses and other malicious code
- provenance tracking for computational and experimental data with optional electronic signatures
- references to items in other executable papers allow re-use of previously published data and code

An aspect not addressed by this proposal is user interfaces, because it is important to clearly separate data representation and user interfaces. User interface choices depend on both personal preferences (e.g. some people prefer graphical user interfaces, others command line interfaces) and the hardware being used (a desktop computer and a smartphone have very different requirements). Data representation, on the contrary, must be as uniform and precisely defined as possible to guarantee portability and longevity of electronic documents.

A key design principle behind this proposal is the notion that "code is data". There is in fact no fundamental difference between executable programs and other data items: they are stored in memory or in files and can be transformed or analyzed by other programs. For both code and other data, clarity, portability and long-term usability require well-defined and carefully designed data models. The core of this proposal is exactly that: a data model for reproducible

research and for executable papers. The next section discusses the most important choices to be made: how to represent code and other data, and how to package data and metadata to make an executable paper.

2 Representation of data, code, and text

2.1 File storage

For reasons of integrity and for simplifying automated treatment, an executable paper must consist of a single file. The file format chosen must thus be able to accommodate raw data (including potentially very large data sets), metadata (for provenance tracking and for use in data models), executable code, text including markup for equations etc., and machine-treatable references to other executable papers (for corrections, updates, comments, etc.).

The Hierarchical Data Format 5 (HDF5) [2] fulfills all these requirements. It is a format specifically designed for scientific computing and addresses issues specific to this domain, e.g. the prevalence of large homogeneous data sets such as arrays. HDF5 is already widely used for implementing data models for experimental data (e.g. NeXus [3], used in neutron and X-ray scattering) or computational results (e.g. CGNS [4, 5], used in Computational Fluid Dynamics).

An HDF5 file has an internal tree structure that resembles a file system as used by conventional operating systems, with group nodes playing the role of directories and datasets (leaf nodes) storing the actual data. Data sets can be simple values but more typically are arrays or tables of values. Data sets can be of fixed or variable size and use optional features such as compression. Individual values are defined by their data type, which can be a built-in standard type such as "16-bit unsigned integer" or "variable-length character string" or a user-defined (compound) type. There is also an "opaque" type for storing arbitrary byte sequences which is suitable for storing binary code representations. The wide range of data types and dataset types provides the flexibility required for storing any kind of data and choose a suitable representation that is the right compromise between file size, access time, and convenience.

Each group and dataset in an HDF5 file can have metadata attached in the form of named attributes. Attribute values can be of any HDF5 data type. A data type of specific interest for attribute values is the reference type, which stores a pointer to a dataset. Reference-type attributes can be used to store dependency information between datasets.

The HDF5 format is implemented as a highly portable library written in the C language. It is developed and maintained by the HDF Group, located at the University of Illinois. Interface layers for C++, Fortran, and Java (using the Java Native Interface (JNI)) are provided by the HDF group as well. Interface layers for a large number of other programming languages are available from other sources. A large number of generic tools for transforming and visualizing data stored in HDF5 files is available from the HDF Group and others.

Two important more recent developments are worth mentioning here because they ensure the utility of HDF5-based formats on future computing platforms. A parallel-I/O version of HDF5, available from the HDF Group, allows its efficient use in supercomputing environments. The OPeNDAP software [6] provides remote access to datasets stored in HDF5 (and other) files. Remote access is important for handling very large datasets that cannot be transferred completely to every user's computer, but also for browsing tools on portable devices such as smartphones and tablet computers.

2.2 Domain-specific data models

Many kinds of scientific data require non-trivial and non-obvious representations. For example, storing the configuration of a protein implies storing the positions of all atoms as well as sufficient information about the chemical structure of the protein to be able to identify the structural role of each atom as well as its most important chemical properties. The correct interpretation of such data relies on a well-defined data model.

Data models are necessarily domain-specific, and therefore cannot be part of a universal format for executable papers, with the exception of simple and frequently used data models e.g. for time series or mathematical equations. Each dataset or group of datasets has a metadata tag (an HDF5 attribute whose name is fixed by convention) indicating the data model to be applied.

It is expected that each electronic journal will select a set of data models relevant for its community and apply verification tools for those data models upon submission of an electronic paper. It is also expected that domain-specific support libraries will emerge that facilitate the use of HDF5-based data models and spare scientists and scientific software developers the effort of using the rather complex HDF5 programming API directly.

2.3 Dummy data sets

If a dataset can be reconstructed from other information in an executable paper, why store it at all? The main reasons are

- Time: recalculation of the data may be feasible, but too slow for interactive browsing.
- Space: the data set may be too big to be stored in memory, requiring on-disk storage prior to visualization.
- Simplicity of access: analysis or visualization software may not have the functionality to recompute the data.

In these situations, the explicit presence of a dataset is a form of caching. For long-term storage, where data size is the most prominent criterion, it may be preferable to delete such data sets and replace them by dummy data sets that retain only the metadata, permitting recalculation at a later time.

2.4 Executable code

The representation of code inside an executable paper is the most difficult aspect because of conflicting requirements. A suitable code representation must be platform-independent and stable over very long time spans in order to permit its use on future computer systems whose characteristics are still unknown. It should rely on a run-time system that can block erroneous or malicious behavior. It should have an unambiguous specification of the outcome of arbitrary computations. It should admit a wide range of programming languages, current and future. It should not stand in the way of efficient program execution on a wide range of computer architectures. Finally, it should permit the re-use of existing scientific software as much as possible. In the following I will examine existing code representations according to these criteria.

2.4.1 Source code

The availability of source code for computational procedures in an executable paper is highly desirable because it is the most precise documentation of the algorithms, and it permits readers to explore modifications. However, source code is not a good choice as the primary code representation in an executable paper. First of all, there is a very large number of programming languages in use, of which many have machine- or vendor-specific dialects. An executable paper format based on source code would have to allow only a well-defined (and thus fixed) set of programming languages, which would be an obstacle to future progress in software development. Moreover, many programming languages are intentionally ambiguous, leaving the precise meaning of some constructs undefined in order to allow compilers to apply machine-specific optimizations. Floating-point arithmetic is particularly concerned by this problem. As a consequence, the same source code compiled on different machines can yield different results, which violates the reproducibility requirement. Finally, the use of source code would require a runtime system for working with executable papers to include compilers or interpreters for all allowed languages.

2.4.2 Machine code

Machine code has the advantage of being unambiguous and able to exploit all features of a computer at maximum performance.. However, it is also inherently unportable, being specific to a processor or a processor family and in practice also to an operating system or even a precise version of an operating system. Machine code also poses security problems as it is very difficult to prevent malicious behavior.

Some of these problems can be circumvented by making a complete software installation, including the operating system, available through virtualization technology and run it in a secure environment. Moreover, emulators can be used to execute machine code for one machine on a different machine. However, both approaches lead to performance loss (which is severe in the case of emulators)

and are not suited for long-term archival because computer architectures will inevitably change in the future.

2.4.3 Virtual machine bytecode

Virtual machines executing bytecode (also called p-code) provide an intermediate representation between source code and machine code that is both portable and unambiguous. They also make it easy to restrict access to resources such as local files for security reasons. The main disadvantage of using a virtual machine is the loss of performance compared to optimized machine code. However, Just-In-Time (JIT) compilers have reduced the performance gap between virtual machines and native code, and this trend is likely to continue.

There are at the moment two widely used virtual machines that have good enough performance and sufficient programming language support to be candidates for use in an executable paper format: the Java Virtual Machine (JVM) [7] and the Common Language Infrastructure (CLI) [8]. In the following I will concentrate on the JVM because I am more familiar with it. However, the CLI might well be an equally good or even better choice.

The JVM is not an optimal choice for scientific computing. It wasn't designed for this field and therefore has a couple of shortcomings. It lacks value types (types whose data fields are inlined into a parent object or an array and don't require a separate object allocation with the associated garbage collection overhead), which would make many scientific data items (complex numbers, geometric data such as points or triangles, ...) more efficient in terms of memory use and CPU time. The JVM's handling of floating-point arithmetic has been criticized frequently because of its lack of some important IEEE 754 features. In modern JVMs, floating-point arithmetic is not platform-independent unless "strict math" is explicitly requested by the program; this feature was introduced mainly for improving performance on the popular Intel x86 architecture.

Perhaps the most significant obstacle to rapid adoption of any JVM-based infrastructure for executable papers is the lack of production-quality compilers for the most popular scientific programming languages (Fortran, C, C++) that produce JVM bytecode. However, proof-of-concept compilers for C [9] and Fortran [10, 11] already exist. The use of an x86 emulator such as JPC [12] may be of interest as a temporary solution for running legacy software, in spite of the significant performance cost.

Considering the numerous advantages of virtual machines (portable binary representation, portable mixed-language programming, automatic memory management, code optimization based on run-time information) and the fact that modern JIT compilers have almost eliminated their performance overhead, I expect the scientific computing community to adopt virtual machines in the long run for most computational tasks. Executable papers may well be one important motivation to move in this direction. New compilation tools such as LLVM [13] and VMKit [14] are an important technological innovation for such a move. They blur the frontier between native machine code and virtual machine bytecode, allow a better integration between the two worlds, and facilitate the

design and implementation of virtual machines. In the not-too-distant future, scientists will generate portable bytecode for publication and highly optimized native code for their supercomputer from the same source code.

2.4.4 Source code compilation

Even with virtual machine bytecode as the primary code representation in an executable paper, it is highly desirable for its users to have access to the source code as well, because it is the only complete description of the algorithms being used. This raises the question of ensuring that source code and bytecode are equivalent such that readers can trust that the source code they read corresponds to the computations they run.

The problem is exactly the same one as for dependencies between data sets, and it can be solved using the same approach: bytecode datasets are tagged with a reference to the source code dataset from which they can be recovered by compilation. This requires only that compilers be integrated into the executable paper ecosystem in the same way as computational software is, i.e. compilers (and associated tools) must be available in the form of virtual machine bytecode. A few JVM-based compilers already fulfill this requirement (see e.g. [15], [16], [17]), and I expect their number to grow. There is also a unified compilation API for the JVM [18]. Once again, the adoption of a virtual-machine-based executable paper framework is likely to create the necessary motivation for writing compatible compilers.

2.4.5 Scripts

The above discussion has concentrated on the traditional way of writing scientific programs: source code is translated to bytecode or machine code, which is then run. Another mode of operation has been gaining popularity recently: the use of scripting languages at the highest levels of problem specification, making use of libraries written in lower-level languages for efficiency. Examples of popular scripting languages in scientific computing are Matlab, Python, and R. These languages are typically interpreted, with the interpreter proposing an interface to compiled low-level code.

In the proposed framework for executable papers, a scripting language can be made portable and unambiguous by including its interpreter (usually in the form of a reference, see section 2.7) in the paper itself as JVM bytecode. The use of scripting languages should even be particularly encouraged, because it makes the higher-level algorithmic logic of the computation clearer and easier to modify by the reader. Browsers for executable papers should include a script editor that permits readers to explore the influence of parameters and of algorithmic choices.

2.5 Text, tables, and figures

A traditional print-based scientific publication has as its prime ingredients text (including mathematical formulas), tables, and figures. Tables and figures are replaced by dynamically generated table views (i.e. code) and visualization scripts in executable papers. As for text, several useful representations are already in use, and tools for working with them are widely available. The transition to executable papers requires only one new feature: the possibility to include a reference to datasets and executable programs. This feature is readily provided by established hyperlink mechanisms.

The choice of a suitable text representation depends mainly on issues outside of the scope of this article: the needs of authoring tools and browsers. An obvious criterion is the suitability of the representation for algorithmic treatment, which excludes purely visual formats such as PDF. Candidates are light-weight content-oriented markup languages such as reStructuredText [19] or Markdown [20], complemented by an equation language, XML-based formats such as DocBook [21], or traditional typesetting languages such as TEX [22].

An aspect that deserves a special discussion is the representation of mathematical equations, which are widely used in many branches of science. Often they are used exclusively for the benefit of the human reader, for defining quantities and for documenting computational methods. They then have the same role as text and should be included in the latter using suitable markup. Equations that correspond directly to a program should be accompanied by a reference to the latter. It is, however, unrealistic to try to establish a direct correspondence between a program and an equation that documents it. In most practical situations, a program is described by more than one equation plus a context (definition of the quantities) provided in text form. Moreover, the algorithm implemented in the program usually deviates from the written equations for reasons of efficiency or data representation.

Mathematical equations take a more active role in publications in which their manipulation (in the form of symbolic computation) is an essential part of the paper's content. They should then be considered data and represented by a domain-specific data model just like other data. Their graphical display is a special case of visualization, to be handled by visualization scripts. It should be noted that the World Wide Web Consortium's MathML markup language [23] recognizes the two distinct roles of mathematical equations by defining separate markup languages for the display and the semantics of mathematical equations.

2.6 Visualization scripts

The most general specification for a visualization (defined here as any dynamically generated data display, i.e. including tables) is a program, meaning that the executable paper specification does not need to provide any special visualization features. However, it would be inconvenient to leave the choice of visualization routines fully to the authors of an executable paper. This would represent a burden for the authors and lead to a lack of coherent user inter-

face for readers. Visualization should be handled by the run-time system for working with executable papers (see section 4) and by its domain-specific extensions. Ideally, the run-time system provides a special scripting language for visualization which is then used in papers to specify interactive data displays.

2.7 References

In traditional print publishing, papers cite other papers using human-readable text, usually following some formatting convention. Such citations are notoriously difficult to treat electronically because of format variations. With the advent of electronic publishing, Digital Object Identifiers (DOIs) have replaced text-based references for use in databases and other electronic resources [24]. A DOI is a character string that uniquely identifies an electronic object. An organization issuing a DOI is responsible for maintaining access to the object and its associated metadata. DOIs are not limited to readable documents; for example, the DataCite consortium [25] promotes the use of DOIs for access to scientific data. DOIs are clearly a suitable way to refer to executable papers as well.

However, executable papers should permit more fine-grained references than DOIs in order to target specific datasets inside another executable paper. Such a mechanism permits the re-use (with proper attribution) of data and code published earlier. It can also be of interest to publish an executable paper explicitly in several pieces, i.e. to facilitate sharing of very large data sets among several publications. Since inside an HDF5 file, every dataset is uniquely identified by its path (the sequence of groups to be traversed in order to reach the dataset), which can be represented by a character string, a reference into another executable paper consists of two character strings: the DOI of the paper, and the path of the dataset inside the paper.

2.8 Provenance tracking

Data sets that are computed via code provided in an executable paper itself need no provenance information, as they can be recomputed at any time for verification. However, every paper must include information that cannot be regenerated automatically, as otherwise the paper would be of no interest. At the very least, a paper contains program source code and presentation text that are original work of its authors. Experimental data and input parameters for computations also fall into this category. Another category of data that cannot be reproduced is data taken over from legacy sources (print publications, databases) that cannot be referenced directly. Finally, there are data items that could in principle be reproduced but with means not easily available to every reader. This category covers computations that require a supercomputer, but also results obtained from proprietary or legacy software that cannot be included in an executable paper.

All the data items cited above should be accompanied by provenance information, and provenance information should be verifiable. A suitable verification

scheme for all the cases cited is an electronic signature, i.e. a digital fingerprint of the data encrypted using a public-key infrastructure. In the case of original work, the electronic signature identifies the authors, whereas non-reproducible computational results could be signed automatically by the batch processing system of a supercomputing center, using a signature identifying the computer system being used.

The digital fingerprints from the signatures can also be used to set up a database of published datasets, indexed by fingerprints, and consulted to detect basic forms of plagiarism. However, more sophisticated detection systems for plagiarism will be required, as it is rather straightforward to copy someone else's dataset and modify it slightly such as to change the fingerprint.

3 Executable papers

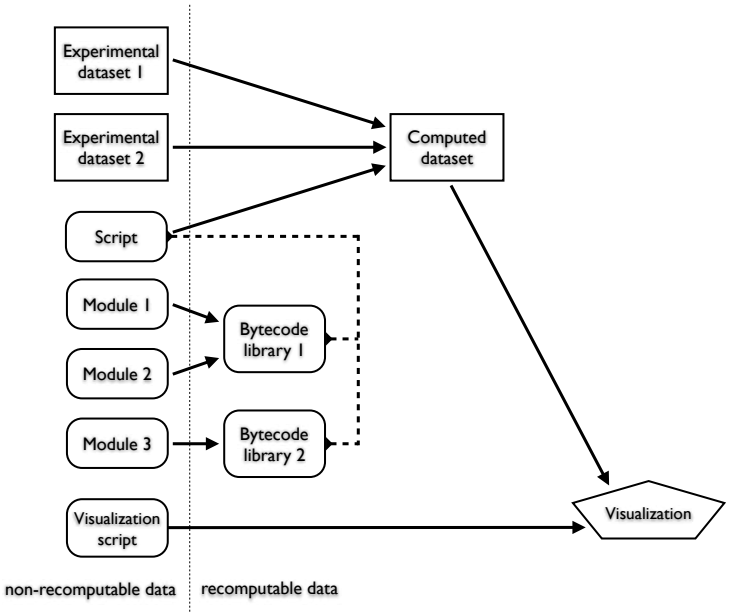

Figure 1: Data dependencies in an executable paper. The items on the left of the dashed line have no incoming arrows and cannot be recomputed. The script uses the bytecode libraries, but does not depend on them. However, data items that depend on the script indirectly also depend on the bytecode libraries.

In the preceding sections, I have explained the important design choices of my proposal for an executable paper format, but less relevant details were intentionally left out. In this section, I briefly present a concrete data layout to convey a clearer impression of what an executable paper looks like. Figure 1 illustrates the dependencies between data items in an executable paper.

An executable paper consists of a single HDF5 file. Published executable papers are identified by a Digital Object Identifier (DOI) and contain their own DOI in their metadata, permitting automated authenticity checks.

The file's root group contains three subgroups "data", "code", and "text". The "data" group contains groups and datasets that each have a metadata tag (HDF5 attribute) specifying the domain-specific data model they respect. Other attributes contain provenance information (for non-recomputable data), or references to dependencies plus a reference to the program (in the "code" subgroup) that re-calculates the dataset from its dependencies. There can also be a "legend" attribute pointing to a plain-text explanation for human readers.

The "code" subgroup contains any number of HDF5 opaque datasets storing a jar file per dataset. Attributes of these datasets store relevant metadata: provenance, reference to the source code, reference to the compilation script run to rebuild the jar file. The "code" subgroup also contains any number of scripts. A script is a character-string dataset whose attributes store references to the jar files making up the interpreter, and the JVM command for running the interpreter. Finally, the "code" subgroup can contain any number of source code files and groups containing source code files. A source code file is stored in a character-string dataset with provenance information in the metadata.

The "text" subgroup contains any number of character-string datasets representing text using a markup language. Each dataset has provenance information in its metadata. One dataset must be called "main" and contains the main presentation text that is shown to the reader upon opening the executable paper. References (hyperlinks) in each text can refer to other text items or to scripts/programs that produce visualizations. References can also point to datasets in the "data" section, allowing the user to open a data browser window for a specific dataset.

It is important to recall that references can point to items in previously published executable papers, using a combination of the DOI and the data item path inside the file. In fact, executable papers cover a much wider range of publishable items than traditional printed papers. Some important special cases of executable papers are:

- Raw data sets, usually from experiment or observation. Such papers contain data and text (for documentation), but no code.
- Program libraries, containing code and text (for documentation) but no data.
- Comments on published papers, containing only text and optionally visualization scripts.

Such items, which currently do not fit into the scientific publication system, become normal publications that have authors, a publication date, and which can be cited. They are automatically integrated into the archival systems operated by scientific publishers, ensuring their long-term conservation in unmodified

form, and thus the reproducibility of other papers that refer to them. Publishing such items separately also provides a means of asserting authorship on important contributions even if they are not in themselves scientific results.

4 Software infrastructure

Working with executable papers requires a variety of software tools. For successful adoption of an executable paper format, it is essential that it can be integrated into as many existing tools as possible, including

- interactive scientific computation environments
- workflow managers
- authoring environments for composing electronic papers
- software development environments
- visualization software

In addition, new functionalities are required that can be either integrated into existing tools or provided by new tools:

- interactive browsers for desktop computers that let readers explore the text and data as well as re-run the computations and modify parameters
- server-side tools with the same functionality, accessed via a standard browser interface by the user working on a desktop computer or a less powerful device such as a smartphone
- automatic verification tools that re-run all computations in the paper and compare the results to the ones supplied by the authors
- authentication tools that verify the electronic signatures

The current proposal is based on two existing and well-established technologies: the HDF5 format and library for data storage, and the Java Virtual Machine for code execution. The HDF5 format is already widely supported by scientific computing tools. The JVM has found its niches in scientific computing, but cannot be considered popular yet. This is partially due to its real or perceived defaults (see section 2.4.3), but mostly due to a lack of scientific libraries and compilers for the most popular scientific programming languages.

The most important support software that needs to be written for using executable papers is a run-time system for executing the code stored in such a paper. This run-time system must obviously include a JVM and the HDF5 library. It must be able to run programs from a paper in a secure environment in which access to local resources on the user's computer is limited to reading data from executable papers and writing data to a single HDF5 file specified

by the user. In addition, a good run-time system should provide support libraries for data management, visualization, etc. The generic run-time system would in practice be complemented by domain-specific extensions that handle domain-specific data models and access to domain-specific resources such as public databases.

5 Related work

The literate programming and reproducible research features [26] in Emacs org-mode [27] were a major inspiration for this work. They permit the storage of text, code, and data in a single plain-text file and provide an interface between code sections written in different languages. Org-mode is limited, however, by its plain-text representation; data sets cannot become very large and the code contained in such a file necessarily depends on external software (compilers and interpreters, usually also libraries) that cannot be included in the same package.

The SHARE system [28] also uses virtual machine technology (coupled with network access through a browser) to provide platform-independent and secure access to research code and data. It differs in using system virtualization to store the researcher's entire working environment, including the operating system, the whole file structure, and programs in the form of machine code. The obvious advantage of this approach is the possibility to use existing software and data formats. On the other hand, system-level virtual machines are impractical for long-term storage because of their size and their narrowly defined hardware requirements. Moreover, data and code stored in this way are not reusable in other work.

Various existing scientific computing tools, including workflow managers such as VisTrails [29] or Kepler [30], interactive scientific computation environments such as Matlab [31] or Spyder [32], or automated logging systems such as Sumatra [33], handle much of the dependency information that makes the difference between a collection of code and data and an executable paper. Another interesting approach to dependency handling that also permits the unification of code and data is the use of software build tools for computation. An example is the use of SCons [34] in the Madagascar tool suite [35]. All of these tools could probably be modified easily to work with the data model described in this work.

6 Conclusion

The preceding sections propose a data and code representation model that support reproducibility in computational science and the publication and archival of reproducible computational results. It addresses many of the open issues in the design of executable papers:

Executability The proposed system permits authors to provide the full code implementing their computational procedures with their publication. It

also permits the publication of code libraries that other papers can build on.

Short and long-term compatibility The use of a virtual machine (JVM) ensures code portability on current and future platforms. The HDF5 library ensures the portability of data storage. The long-term compatibility of the format relies on the long-term maintenance of the JVM and the HDF5 library, or on translation mechanisms for converting first-generation executable papers to an eventual future format. Long-term archival by scientific publishers of all data and code guarantees long-term reproducibility of results.

Validation The proposed format permits an automatic validation of all results for which code is provided, and the automatic verification of electronic signatures for all other data.

Universality Domain-specific data models and domain-specific extensions to the run-time system ensure the applicability to all domains of computational science. The use of virtual machine bytecode as the principal code representation admits a wide range of programming languages.

Use of supercomputers Results that cannot be easily reproduced due to exceptional system requirements (i.e. the use of supercomputers) are treated like experimental data: they are supplied with provenance information and an electronic signature certifying their origin. Program code for reproducing them can still be supplied and used in the future if the required computing power becomes more easily available.

Data size The HDF5 library can handle large datasets efficiently and provides support for parallel I/O. Large datasets can be published separately and accessed from analysis papers through references. Remote access to subsets of a large dataset stored on a server eliminates the need to transfer large datasets completely to every user's computer.

Provenance The origin of each data item can be documented by provenance metadata and certified by an electronic signature. Computational steps are documented by their program code and can be reproduced at any time.

Security The use of a virtual machine (JVM) permits the execution of code contained in a paper in a secure run-time environment that protects users against viruses, trojans, and spyware.

The biggest obstacle to the rapid adoption of this proposal is the requirement for all computational code to be based on a virtual machine, which excludes many currently popular programming languages and tools. However, there is no fundamental difficulty with writing the required tools, and the adoption of executable papers by major scientific publishers is likely to encourage such developments.

The implementation of the basic run-time system requires only a modest development effort because most of the technology already exists. Implementing domain-specific extensions represents a much larger effort, because of the large number of domains with specific requirements, and also because many domains of computational science did not yet develop well-defined data models.

Acknowledgements

This work was supported by the Agence Nationale de la Recherche (Contract No. ANR-2010-COSI-001-01).

References

[1] Sergey Fomel and Jon F. Claerbout. Guest editors' introduction: Reproducible research. *Computing in Science & Engineering*, 11(1):5–7, JAN-FEB 2009.

[2] The HDF Group. Hierarchical data format version 5. `http://www.hdfgroup.org/HDF5`.

[3] P Klosowski, M Koennecke, JZ Tischler, and R Osborn. NeXus: A common format for the exchange of neutron and synchrotron data. *Physica B*, 241:151–153, DEC 1997. International Conference on Neutron Scattering, Toronto, Canada, Aug 17-21, 1997.

[4] D. Poirier, S. R. Allmaras, D. R. McCarthy, M. F. Smith, and F. Y. Enomoto. The CGNS system. American Institute of Aeronautics and Astronautics Paper 98-3007, 1998.

[5] CFD general notation system (CGNS). `http://cgns.sourceforge.net/`.

[6] OPeNDAP: Open-source Project for a Network Data Access Protocol. `http://www.opendap.org/`.

[7] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Prentice Hall, 1999.

[8] ECMA Standard 335: Common Language Infrastructure CLI. `http://www.ecma-international.org/publications/standards/Ecma-335.htm`.

[9] David Given. Clue: an ANSI C compiler targeting high level languages. `http://cluecc.sourceforge.net/`.

[10] G Fox, XM Li, QA Zheng, and ZG Wu. A prototype of Fortran-to-Java converter. *Concurrency - Practice and Experience*, 9(11):1047–1061, NOV 1997. Simulation and Modelling Workshop II - Java for Computational Science and Engineering, Las Vegas, NV, Jun 21, 1997.

[11] f2j, a Fortran to Java compiler). http://icl.cs.utk.edu/f2j/.

[12] Ian Preston, Rhys Newman, Jeff Tseng, Chris Dennis, Guillaume Kirsch, and Mike Moleschi. Pure Java x86 emulator. http://jpc.sourceforge.net/.

[13] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. *IEEE/ACM International Symposium on Code Generation and Optimization*, 0:75, 2004.

[14] N. Geoffray, G. Thomas, J.Lawall, G. Muller, and B. Folliot. VMKit: a substrate for managed runtime environments. In *Virtual Execution Environment Conference (VEE 2010)*, pages 51–62, Pittsburgh, USA, March 2010. ACM Press.

[15] Arno Unkrig and Matt Fowles. Janino, a super-small, super-fast Java compiler. http://docs.codehaus.org/display/JANINO/Home.

[16] Groovy, an agile dynamic language for the Java platform. http://groovy.codehaus.org/.

[17] Rich Hickey. Clojure, a dynamic programming language that targets the Java Virtual Machine. http://clojure.org/.

[18] The Apache Software Foundation. Apache commons JCI, a Java compiler interface. http://commons.apache.org/jci/.

[19] reStructuredText. http://docutils.sourceforge.net/rst.html.

[20] Markdown. http://daringfireball.net/projects/markdown/.

[21] The OASIS consortium. The DocBook schema version 5.0. http://docs.oasis-open.org/docbook/specs/docbook-5.0-spec.html.

[22] Donald Ervin Knuth. *The TeXbook*. Addison-Wesley, London, 1984.

[23] The World Wide Web Consortium. Mathematical Markup Language (MathML). http://www.w3.org/Math/.

[24] The International DOI Foundation. The DOI system. http://www.doi.org/.

[25] The DataCite consortium. http://www.datacite.org/.

[26] Eric Schulte and Dan Davison. Active documents with org-mode. *Computing in Science & Engineering*, 13(4):in print, 2011.

[27] Carsten Dominik and et al. Org-mode for emacs. http://orgmode.org/.

[28] Pieter van Gorp and Paul Grefen. Supporting the internet-based evaluation of research software with cloud infrastructure. *Software and Systems Modeling*, pages 1–18, 2010. 10.1007/s10270-010-0163-y.

[29] VisTrails. http://www.vistrails.org/.

[30] The Kepler Project. https://kepler-project.org/.

[31] The MathWorks, Inc. MATLAB. http://www.mathworks.fr/.

[32] Pierre Raybaut. Scientific PYthon Development EnviRonment (Spyder). http://packages.python.org/spyder/.

[33] Andrew Davison. Sumatra, a tool for managing and tracking projects based on numerical simulation or analysis. http://neuralensemble.org/trac/sumatra/.

[34] Steven Knight, Chad Austin, Charles Crain, Steve Leblanc, and Anthony Roach. Scons software construction tool. http://www.scons.org/.

[35] Madagascar, an open-source software package for multidimensional data analysis and reproducible computational experiments. http://www.reproducibility.org/wiki/Main_Page.