In Proceedings of the 2013 IEEE Symposium on Visual Languages and Human-Centric Computing © IEEE 2013

A Suite of Domain-Specific Visual Languages For Scientific Software Application Modelling

Mohamed Almorsy[*], John Grundy[*], Richard Sadus[*], Willem van Straten[+], David G. Barnes[#], Owen Kaluza[#]
[*] Centre for Computing and Engineering Software and Systems, Swinburne University of Technology, Hawthorn, Australia
[+] Centre for Astrophysics and Supercomputing, Swinburne University of Technology, Hawthorn, Australia
[#] Monash Monash Biomedical Imaging and Monash e-Research Centre, Monash University, Clayton Australia
malmorsy@ swin.edu.au, jgrundy@ swin.edu.au, rsadus@ swin.edu.au, wvanstraten@ swin.edu.au,
david.g.barnes@monash.edu, owen.kaluza@monash.edu

***Abstract*-Many advances in science now require sophisticated scientific software applications that facilitate data and computationally intensive experiments. However, the effective utilization of existing computational power e.g., grid and cloud platforms depends on the capabilities of scientists to implement parallel, scalable code for such experiments. Currently, tools aimed at supporting scientists are either very limited to specific domains, or require significant development using low-level code. We describe our work towards a more end user-friendly scientific applications development process, notations and toolset. We introduce a scientific application designer intended for use primarily by scientists to enable them in describing workflow, processes, entities, formulae, computation and ultimately realization code for different computing platforms. This is achieved via a set of integrated, domain-specific visual and textual languages (DSVLs). A web-based modeling tool supports definition of new DSVLs and modeling of these applications. We are currently extending our tool to support generation of multi-core and GPU implementations, and visualization of results.**

Keywords-component: **Domain-Specific Visual Language, Model-Driven Engineering, Scientific Applications, Scientific Workflow, High-Performance computing**

I. INTRODUCTION

Increasing IT infrastructure capabilities including computation, storage and communication has opened the door for scientists to try and address much more complex problems than previously attempted. New research areas have arisen that try to enable scientists to use greater computational power in their data-intensive and computationally-intensive experiments, including scientific computing [1], scientific workflow [2], high-performance computing [3], and model-driven engineering for scientific applications [4].

Developing new scientific applications most often requires deep involvement of scientists in writing low-level source code, as it is often difficult to use specialized programmers due to the deep scientific background required in developing such applications. Generally this means that scientists have to have good programming experience, most often with the C/C++ language, and software debugging and code optimization techniques. In addition, because many applications require very large data manipulation and computation, expertise with HPC and how to maximize the utilization of the computational power of the available computational platforms is usually required. This includes detailed use of HPC programming models embodied by APIs and languages such as MPI, OpenMP, OpenCL and CUDA. Finally, some expertise in software maintenance, in order to be able to modify existing programs, is often necessary. Working on source code level without high-level documentation of these scientific algorithms or bidirectional mapping to source code causes lot of consistency problems. Furthermore, new researchers find it very hard to understand and maintain code written by someone else. On the other hand, 90% of scientists did not study programming, yet spend more than 40% of their time developing software to support their experiments [8]. The main languages used are MATLAB, C/C++, and Python. These software solutions usually run for several days.

In this paper we describe our prototype scientific Domain-Specific Visual Languages (DSVLs)-based toolset. This is intended to provide scientists with a more human-centric approach to developing highly customizable domain-specific visual languages capturing their target domain concepts. These domain-specific visual languages are then used to develop their scientific applications. We illustrate multiple, integrated DSVL usage for exemplar scientific domain in our prototype web-based DSVL modeling tool.

II. BACKGROUND

New solutions try to address scientific applications development using DSVLs that capture domain concepts and help scientists in developing their experimental software. Existing domain-specific tools, such as LAMMPS[1] for molecular simulation and MeVisLab[2] for medical image processing and analysis, deliver a set of predefined capabilities that can be reused as building blocks to develop new scientific experiments in these domains. However, most such solutions are black-box with limited domain applicability and limited ability to reuse and extend the platform. Some more general, customizable solutions do exist, mainly based on scientific workflows with a focus on data flow [5], such as Kepler [6] and Microsoft Trident workbench [7]. However, these tools also have limited capabilities to help scientists address new problems through more highly customized solutions. Extending these solutions requires detailed programming knowledge, as mentioned in Section I, and integration with

[1] http://lammps.sandia.gov
[2] http://www.mevislab.de

often complex frameworks. Palyart et al. [4] introduce a DSVL to help in specifying and modeling HPC applications. They focus on specification of solution parallelism. However, they did not show how their language could be used to generate HPC code.

We had multiple brainstorming sessions with different scientists from three different domains including molecular simulation, magnetic resonance imaging, and Astrophysics about their scientific applications' development approaches. All of them stated that they start with pen and paper specifications of their problems and then develop a simple single-processor version of their solutions. Once they have a working prototype, they start porting their solution to multi-core GPUs and CPUs platforms. This requires rewriting most of the program to use MPI, OpenCL, or CUDA. We also did informal survey with some of the new researchers who are expected to extend such applications to serve their new research problems. The feedback was they had to spend lot of time to understand and modify the existing code.

As a next step, we asked several scientists to try and develop descriptions of their target scientific problems. Figure 1 shows snapshots of problem descriptions that we received from our scientists. Figure 1-A shows a molecular simulation experiment modeled as formulae and pseudo-code. Figure 1-B shows radio telescope analysis modeled as scientific workflow. This motivated us to consider an approach of supporting definition of a variety of DSVLs for scientists and use these DSVLs to model and generate complex scientific applications. This helps in minimizing the amount of time spent by scientists in developing and updating applications as well as the amount of HPC and other IT experience required.

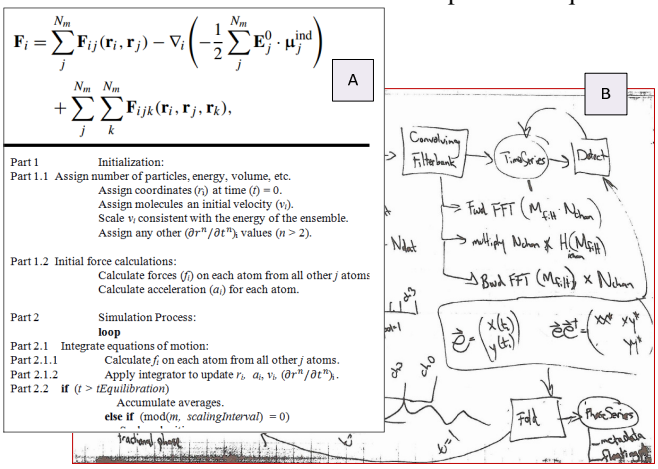

Figure 1. Example problem definitions modelled by scientists

III. Approach

Motivated by this examination of the current scientific application engineering approach, we are developing a novel model-driven scientific applications engineering approach, as outlined in Figure 2. (1) Users define various DSVLs for modelling aspects of the problem domain at varying levels of abstraction. Meta-models and DSVLs should be developed by experienced engineers with the help of domain experts and scientists. (2) Scientists model a particular application of interest with existing and/or new DSVLs (e.g. workflow, science model, formulae). (3) They then refine and annotate their specifications with intended parallelism and structured pseudo-code. (4) Either they or computing platforms' experts model available HPC platform components and characteristics. (5) They assign computation and data to the platform. (6) At this stage they generate, reuse and/or implement GPU code suitable to realize their desired implementation. (7) Experiments are then run by deploying the application to a suitable target HPC platform and results are gathered. (8) Finally, results are then visualized for the user by visualization-specific DSVLs. The intended users of this approach are both experienced scientists who spend most of their time in development although it is a side task of their job; and scientists who develop algorithms with a simple prototype and wait for a development team to parallelize this work on GPUs and CPUs.

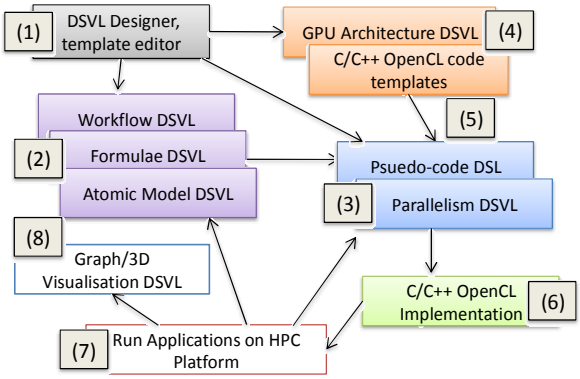

Figure 2. High-level components and steps of our approach

IV. Scientific Applications DSVL Designer

We have developed a detailed meta-model for scientific software applications based on analysis of a range of application domains. We have developed a prototype web-based DSVL designer (meta-tool) and integrated DSVL editors (modeling tools) based on this meta-model. This allows scientists to model their applications at varying levels of abstraction from abstract mathematical model down to code level. Our visual language designer meta-model, outlined in Figure 3, is built on a common entity *Concept*. Each *Concept* entity has a reference to a visualization element which may be a shape such as cloud, rectangle, triangle, square, etc. It may also be an image, or an XMLShape which may be a stencil or HTML tags. A Concept may be one of two domain entities (Data and Task); one of two control flow constructs (IF and Loop operations); or a constraint specification note.

A *Data* entity in any scientific application DSVL represents the definition and structure of data used in the application. For example, if the domain has a matrix as an input, then we need to define how data looks like in terms of attributes (#rows, #cols, and values as a 2D array). Complex data entities may have attributes of other data entities – e.g., an image data entity is made up of pixels.

A *Task* entity represents data processing or other tasks in a scientific application. Each Task has a number of *inputs* (InputPorts), *outputs* (OutputPorts), and a set of ordered *commands* to apply on the inputs. Each input and output has type, name, and multiplicity. Each command has a commandType, which may be a source file, a function, code, or a mathematical formula. In the case of function code, the user has a text editor (e.g. in Figure 4-C, D) where they can write and debug (Pseudo)-code. In the case of mathematical formula, users are provided a formula editor (e.g. Figure 4-B), where they can define their own formulas. We are developing a formula-to-code transformer that generates realization code.

Control flow is covered by two key entities: *IF* and *Loop*. The IF entity has a condition to be checked, and based on the value of this condition we decide which block to execute. The IF entity covers if- and switch-type control flow statements, with a list of conditionally guarded blocks and Tasks to be executed for satisfied conditions. The *Loop* entity has initial value (initialization of loop variable(s)), loop condition (termination), loop step (increment in each iteration), and loop body, a list of Tasks to be repeated.

The *Constraint* entity represents condition(s) to be applied on DSVL entities individually or over grouped elements of the DSVL as a whole. These constraints are expressed using the Object-Constraint Language (OCL) by the DSVL designer.

An example of our DSVL designer meta-tool being used to specify a workflow DSVL is shown in Figure 4-A. Each time a scientist defines a new scientific DSVL using our DSVL designer, they develop a language meta-model (tasks, data, relationships) relevant to the domain. This language definition is retained in a repository and loaded at runtime to instantiate the DSVL's modeling tool. Our DSVL designer specifications are designed in the same way we design other scientific applications DSVLs. Language details are captured as XML that is loaded, parsed, and rendered at runtime. This makes the whole language highly extensible to incorporate new concepts or attributes that we did not cover at this phase, or even scientists can add their concepts that need to be available in designing other DSVLs.

We have implemented a prototype web-based DSVL designer and DSVL modeling tool based on an existing open source web modeling tool[3]. We have also reused a scientific formula editor, text editor for (pseudo)code, and visualization component. Our tool can be downloaded from here.

V. Usage Example

We briefly describe a usage example of our prototype scientific DSVL specification and modeling tool on the N-Body Simulation problem (a simulation of a dynamical system of particles under the influence of physical forces [8]). To address problem, we have conducted a preliminary analysis of the domain to come up with an initial DSVL for molecular simulation. The DSVL toolbox reflects key entities for molecular simulation. This includes Atom (attributes: coordinates, velocity, forces), Atoms Lattice (2D array of Atoms), Atom Forces' Calculator, Atom Velocity and Distance Calculator, Update Simulation. Figure 4-E shows an example of molecular simulation workflow. Given a lattice of Atoms, we start by initializing the atoms attributes. Then, we loop on all atoms to calculate atom forces, velocity, and coordinates. After iterating on all atoms, we reflect new atoms' coordinates on current atoms' visualization. These steps are repeated for X-number of simulation rounds.

[3] https://www.draw.io

VI. Summary

We introduce a new, integrated scientific applications DSVL designer that assists scientists in developing their own application DSVLs. We described a usage example from the molecular simulation domain. We are extending our tool into a scientific applications development platform. This platform uses the DSVL models as above and provides semi-automated support for transforming them into fine-tuned applications using suitable programming models and computational platforms.

ACKNOWLEDGEMENTS

This research is supported by the Australian Research Council under Discovery Project DP120102653.

References

[1] D. F. Kelly, "A Software Chasm: SW Engineering & Scientific Computing," *Software, IEEE,* vol. 24, pp. 120-119, 2007.

[2] A. Barker and J. Hemert, "Scientific Workflow: A Survey and Research Directions," in *Parallel Processing and Applied Mathematics*. vol. 4967, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds., ed: Springer Berlin Heidelberg, 2008, pp. 746-753.

[3] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers* vol.7:CRC Press, 2010.

[4] M. Palyart, D. Lugato, e, "HPCML: A Modeling Language Dedicated to High-Performance Scientific Computing," In Proc. 1st Int Workshop on Model-Driven Engineering for High Performance and CLoud computing, Innsbruck, Austria 2012.

[5] M. Sonntag, D. Karastoyanova, and E. Deelman, "Bridging the Gap between Business and Scientific Workflows: Humans in the Loop of Scientific Workflows," in *e-Science (e-Science), 2010 IEEE Sixth Int. Conference on*, 2010, pp. 206-213.

[6] I. Altintas, C. Berkley, E. Jaeger, et al., "Kepler: an extensible system for design and execution of scientific workflows," in *16th Int. Conf. on Scientific and Statistical Database Management,* 2004, pp. 423-424.

[7] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan, "The Trident Scientific Workflow Workbench," in *IEEE 4th Int. Conf, on eScience*, 2008, pp. 317-318.

[8] G. Marcelli and R. J. Sadus, "Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials," ed.

[9] J. F. Schenck, "The role of magnetic susceptibility in magnetic resonance imaging: MRI magnetic compatibility of the first and second kinds," *MEDICAL PHYSICS-LANCASTER PA-,* vol. 23, pp. 815-850, 1996.

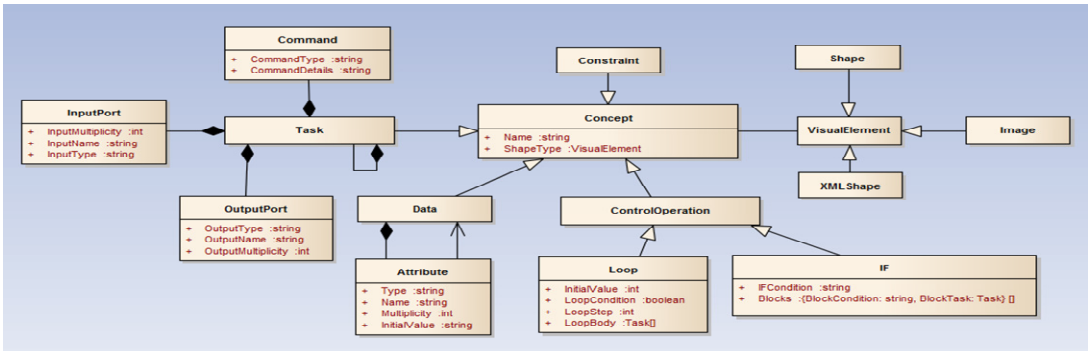

Figure 3. Our DSVL meta-model

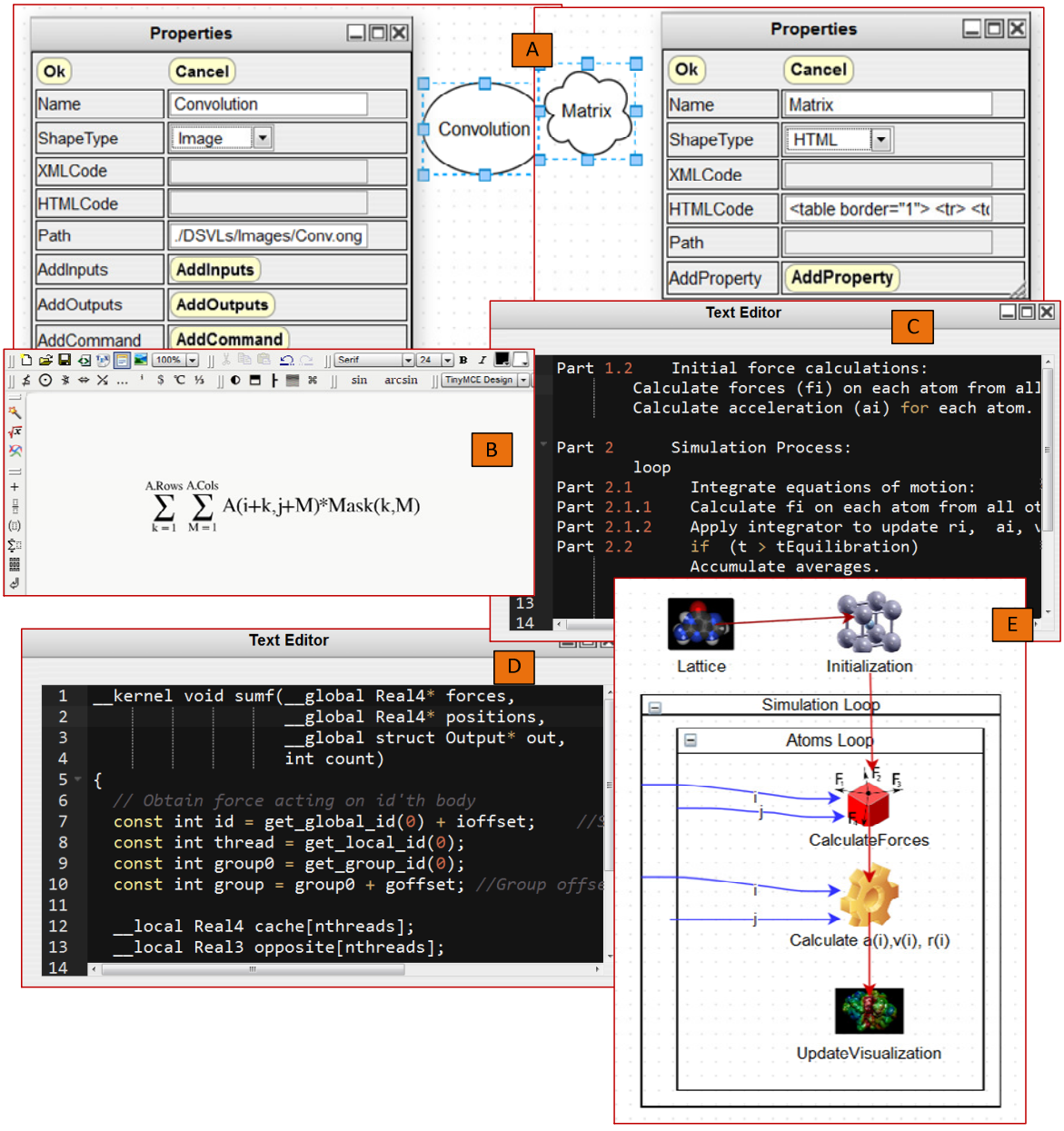

Figure 4. Snapshots from our DSVL designer