Enabling Cross-Domain Collaboration in Molecular Dynamics Workflows

Gergely Varga,
Janos Sallai
and Akos Ledeczi

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee 37212
Email: gergely.varga@vanderbilt.edu

Christopher Iacovella,
Clare McCabe
and Peter T. Cummings

Department of Chemical and Biomolecular Engineering
Vanderbilt University
Nashville, Tennessee 37212
Email: christopher.r.iacovella@vanderbilt.edu

***Abstract*—Molecular dynamics (MD) simulation is used increasingly often for materials design to reduce the costs associated with the pure experimental approach. Complex MD simulations are, however, notoriously hard to set up. It requires expertise in several distinct areas, including the peculiarities of a particular simulator tool, the chemical properties of the family of materials being studied, as well as in general C/C++ or Python programming. In this paper, we describe how MetaMDS, a web-based collaborative environment, allows experts of different domains to work together to create building blocks of MD simulations. These building blocks, capturing domain-specific knowledge at various levels of abstraction, are stored in a repository, and are shared with other users, who can reuse them to build complex simulation workflows. This approach has the potential to boost productivity in chemical and materials science research through separating concerns and promoting reuse in MD workflows.**

Keywords–Simulation; Metaprogramming; Online collaboration; Programming abstractions.

I. INTRODUCTION

Molecular dynamics (MD) simulation has become an important tool in various disciplines, including chemical engineering and materials science, to augment and partially replace the experimentalist approach in materials design. The driving force behind this trend is twofold. First, experiments provide only limited insight to the molecular scale phenomena, and second, the Edisonian approach that relies on experimentation is costly, in particular, when the design space includes a wide range of materials that has to be evaluated. Simulation provides the full spatial and temporal resolution of the system on the molecular scale, providing consider insight into the subtle mechanisms at work, and allows for precise modification of system topology and other parameters, making it possible to use a screening and optimization approach.

Today, several large-scale software packages exist for molecular dynamics simulation. They are conceptually very similar in the sense that they all implement an N-body simulation of a large number of particles and numerically solve Newton's laws of motion, where the forces are computed using functional forms describing the chemical interactions between the particles. These simulators, however, may differ in several aspects:

- the hardware platform they run on may range from desktops to supercomputers and from Central Processing Unit (CPU) to Graphics Processing Unit (GPU) to other specialized hardware;
- their scripting languages may be limited to only rudimentary control structures or could use a powerful language such as Python;
- they may address specific families of chemical compounds (proteins, crystalline structures, etc.) or focus on different chemical properties.

Setting up a molecular dynamics simulation is a notoriously hard task. It needs a user to be familiar with and have expertise in:

- the particular simulator platform, including the syntax of the simulator's scripting language, with the knowledge of how common MD concepts can be carried out using the simulator;
- the specifics of the chemical domain, i.e., how the interactions between the particles need to be parameterized (which may be very different in, e.g., proteins than in ionic liquids);
- the requirements of the particular application domain, e.g., batteries, nanolubrication in hard drives, self-healing paint, etc.;
- and general programming skills to generate input files describing complex systems of particles or to programmatically extract the quantities of interest from the simulation results.

Unfortunately, once a particular simulation workflow has been set up to run on a particular simulator, it is not trivial to retarget it to a different simulator package. There are no common Application Programming Interfaces (APIs), no well-defined abstractions that would allow simulations to be specified in a simulator-agnostic manner. Also, because of the monolithic and ad-hoc nature of the simulation code, code reuse is often merely accidental or nonexistent.

This paper is structured as follows. First, we explain the state of the art of the design-flow of molecular dynamics simulations, highlighting the ad-hoc and often one-shot nature of simulation design. Then, we describe how MetaMDS [1], a web-based metaprogrammable environment, allows for decoupling the roles of simulator experts, (chemical) domain

 ISBN: 978-1-61208-351-3

experts and end users through abstractions, in a way that they can work in parallel, creating reusable software artifacts and collaborate with each other when building complex molecular dynamics workflows. We present implementation details of the MetaMDS tool and conclude with a case-study demonstrating its use.

II. Approach

Our approach is inspired by Model-Integrated Computing (MIC) [2], a systems engineering methodology that focuses on building domain specific modeling environments, which allow for capturing the concept of the given domain at the level of abstraction that is most appropriate for the problem to be solved, hiding unnecessary level of detail from the end users. MIC focuses on creating Domain Specific Modeling Languages (DSMLs) via metamodeling: describing the DSML's concepts with a generic meta-language (which is, in fact, in itself a DSML). The modeling environment includes model interpreters that analyze the model, check constraints, verify properties, and generate code from the model, automating many of the time consuming, tedious and error prone programming tasks.

Molecular dynamics simulation scripts can be thought of as programs with linear control flow that describe how a simulation is initialized (loading particle coordinates from an input file, enumerating and parameterizing the interactions between the particles, defining a simulation cell, etc.), how the system evolves over time (changes in e.g., temperature, pressure, box size, etc.), as well as when and what quantities are logged or saved to file. We observed that the many of these simulation concepts (e.g., loading the input file, resizing the box, logging the potential energy) are supported in multiple simulators, and while the syntax in which they are defined can be very different, the *parameters* of these concepts (e.g., the *name* of the input file to load, the *dimensions* of the new size, or the *frequency* of the time steps when the potential energy should be logged) are more or less the same.

Therefore, we claim that it is possible to define a simulator-agnostic domain specific language (DSL) that can express these concepts as first class language elements. The end user can describe a simulation in terms of these concepts, instead of writing simulation scripts directly. The simulator scripts can then be automatically generated using a (simulator-specific) interpreter. The interpreter maps the concepts to their equivalent simulator-specific code snippets, and stitches them together to form a script understood by the target simulator.

We further observed recurrent patterns on how the basic concepts are used together, and have identified a number of steps (a series of actions that can be described with basic concepts) that are often present in multiple simulations within the same chemical domain. An example of such step is the initialization of a simulation (reading a data file and setting the parameters of the interactions), equilibrating the system (letting the system evolve at a given temperature for an extended amount of time), and even more complex groupings such as shearing a system to calculate the frictional properties. These simulation steps can be expressed in the domain specific language as a composition of the basic concepts. It is important to note that with our proposed approach no simulator-specific knowledge is required to define the simulation steps, aside from a rudimentary understanding of what needs to be in a simulation. Nevertheless, simulation steps can capture a tremendous amount of domain knowledge about specific compounds in a particular chemical domain. For instance, force-field parameters that define the interactions between the particles can be captured in a simulation step, and reused across several simulations involving compounds within the same family (e.g., dodecane and other length alkanes) or reused when calculate other properties (e.g., coupled to simulation steps that capture either phase coexistence or viscosity). We note that initialization of a chemical system and its interactions can be a difficult, error prone task for complex molecules and thus reuse of validated model parameters via simulation steps should not be an insignificant advancement.

The definition of simulation steps can further increase the level of abstraction at which the end user can define entire simulations. While relying on the simulator-agnostic basic concepts eliminates the need for the end user to have expertise in a particular simulator tool, building simulations from coarser-grained steps allows end users with no detailed knowledge of a specific aspect, e.g., force-field parameterization, to assemble and run simulation workflows. As such, this approach enables and encourages collaboration between those with different areas of expertise.

Naturally, however, several questions may arise regarding a *common* molecular dynamics simulation description language. Who defines this DSL and who creates the interpreters? Which concepts should the DSL contain, and which concepts should be excluded? Is it possible to create a one-size-fits-all set of concepts that meets the needs of *all* possible MD simulations? What parameters should a particular concept have? How does the DSL and the simulator specific interpreters evolve as new simulator tools are developed or existing ones are extended with new features.

Our answer to these questions is metaprogramming. MetaMDS, the platform we have developed, provides a way to define the basic concepts, along with their mappings to simulation-specific code in a simple way. This allows the DSL to be flexible and dynamically updated; either the basic steps, nor the higher level concepts are hard-coded into the MetaMDS tool. Instead, each working group may settle on the set of abstractions that best suit their needs, share them with each other, and use these abstractions as a means of interfacing between team members. That is, the DSL can include multiple methods for defining the same basic operation(s), depending on what is most convenient for the system being studied or for the group using the code. Also, this for composition of concepts into simulation steps allowing for increased flexibility and transparency, where again, groups can assemble basic operations into whatever steps most appropriate for the given system. Again, since the DSL is not hard-coded into the MetaMDS tool, it is trivial to add new routines as they are needed or become available, this makes the tool flexible and able to grow to meet the demands of new users, chemical systems, algorithms, and procedures.

A. Separation of roles

We can separate three different types of "actors" in the overall simulation workflow, as shown in Figure 1.

 ISBN: 978-1-61208-351-3

1. *Platform experts* are, as the name suggests, experts in the usage of a simulation platform. They are well versed in the given simulation tool, including the data file format, how a certain task or subtask is implemented, the syntax of the code used to control the simulation, what parameters/routines are needed (and in what order), etc. Examples of such simulation tools include Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS [3]), a classical MD simulator that is commonly run on high-performance computing (HPC) clusters, and Highly Optimized Object-oriented Many-particle Dynamics (HOOMD-Blue [4]), a general purpose particle dynamics simulations that takes advantage of NVIDIA GPUs to attain a level of performance on a single workstation.

Of course, the platform expert needs to have a strong knowledge of the basic principles of the field (e.g., chemical engineering), but they need not possess "domain" expertise related to the specific system of interest, as discussed below.

2. *Domain experts* are usually advanced researchers who are experts with regards to parameterizing a particular system and the general methodologies for carrying out the simulations of that system. We refer to *domain* as the solution for a specific problem, e.g., the models and procedures used to simulate grafted nanoparticles or to calculate the phase coexistence of a system. These users may have expertise in a given platform–i.e., how to implement those parameters/procedures in a specific simulation tool–but their domain knowledge should be considered applicable to any tool, as these concepts and parameters are general. In the next section we introduce two domains investigated during development of our tool, in order to facilitate a deeper understanding and better application of the concepts.

3. *Inquiry scientists (endusers)* perform virtual experiments, running simulations to determine the properties and behavior of the system of interest within a given domain. These inquiry scientists tend to be new researchers with limited domain expertise, with most expertise coming from knowledge of related work in the literature. The role of the inquiry scientist is to develop and/or test hypotheses for a system within a given domain, and to gain domain expertise.

Depending on the research, different types of knowledge are required, which is why domain experts are very important. They can design the flow of the simulations, what tasks need to be executed, what model parameters to use, and what properties need to be changed and checked. Platform experts can help implement these procedures correctly in a given simulation platform, noting that different tools may be best suited for different types of research. Finally, inquiry scientists perform the experiments, building upon the models and procedures developed by the domain experts, and codes implemented for a specific simulation tool by the platform experts. While the division between a domain expert and inquiry scientist is often natural due to seniority in a research group, separating-out platform expertise may be less trivial. The roles we have introduced follow the model that has successfully been applied in many experimental laboratories, where centralized microscopy facilities tend to exist, where "platform" experts in those tools (i.e., those that run the microscopy facility) assist domain experts and inquiry scientists in performing their measurements.

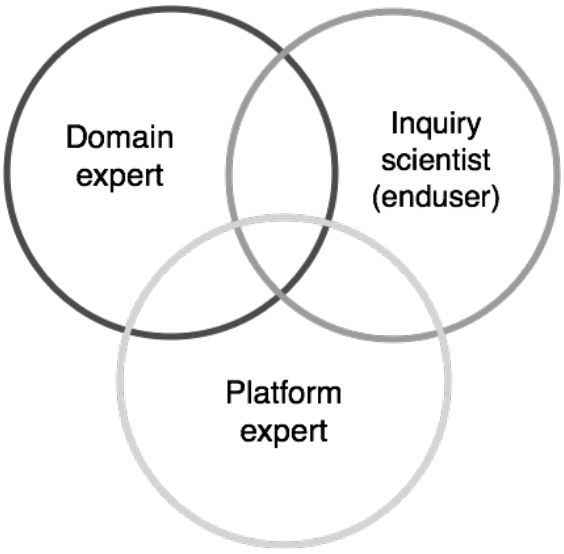

Figure 1. Knowledge domains and their relation. There is a need to know a little bit of everything to accomplish a non-trivial task.

One should note that considerable overlap may exist between these roles. That is, as inquiry scientists become more experienced, they will naturally gain domain expertise and become the domain experts. Also, a domain expert in one area may possess considerable platform expertise and thus can serve as a platform expert for problems outside of his/her domain. Furthermore, to accomplish complex simulations, workflows may need to be developed using multiple experts from different domains or even multiple platform experts if different tools will be used in conjunction. However, by considering the roles and relationships within this context, we can remove the burden that often falls solely on the inquiry scientist, enabling better collaboration and sharing of ideas between experts, which we believe will ultimately augment productivity and quality of the research.

B. Example domains

During the development of the tool, we have examined the protocols and procedures used in the study of two different domains, simulation of the coexistence properties of polymer grafted nanoparticles and the structural and frictional properties of polymer monolayers.

1) Grafted Nanoparticles: Polymers grafted to the surface of nanoparticles, have been used as a means to control the aggregation behavior of nanoparticles, in order to tune the system structure and properties. For example, tuning the graft length can result in transitions from dispersed nanoparticles to string and sheets [5] and properties such as fracture toughness can be increased by many orders of magnitude for polymer grafted nanoparticles as compared to the polymers alone [6]. Understanding how to control the aggregation/dispersion of these systems, via polymer graft length, polymer surface density and nano particle interactions, is of great importance to creating predictive framework for the use of nanoparticles. We have focused on quantifying the aggregation/dispersion behavior of alkane grafted silica nanoparticles, by means of calculating the phase coexistence, as a function of graft properties and relative interactions. In this case, one can consider

 ISBN: 978-1-61208-351-3

this study to be the intersection of two different domains, grafted nano particle simulation and coexistence calculation, with platform expertise related to the HOOMD-Blue simulator.

2) Monolayers: Self-assembled monolayers have been proposed as a means to lubricate and protect surfaces interacting at the nanoscale, such as those surfaces found nano- and micro-electromechanical systems (NEMS and MEMS). Designing lubricants for such systems is not necessarily straightforward, as the behavior may strongly depend on many factors including the chemical composition (and mixtures) of the monolayer molecule(s), length of the molecule, density of the molecules, and surface structure. Several different domains intersect, including domain experts in the ares of (1) monolayer assembly, (2) monolayer simulation under equilibrium conditions, and (3) non-equilibrium simulation (to calculate frictional behavior) with platform expertise related to the LAMMPS simulator.

If we want to accomplish a relatively non-trivial task using the state of the art technologies in MD simulations, there is a need to have knowledge of the three actors: we need the domain knowledge to design the concept of a simulation, we need the knowledge how to implement the steps that we want to run in a specific platform, and last, but not least, we have to create the data file and all the physical/dynamic properties that could be crucial to succeed with our experiments. In our work, we have broken down how a simulation is accomplished from the beginning to the end, creating a universal tool that helps different field-experts collaborate with each other.

Our collaborative tool helps experts to work together and design simulations from ground-up in an interactive way. To use the tool for different simulations from different domains creates the situation for users that the more they use the tool, the less work they need to invest in it. Building a shared library of basic building blocks and using them as basic concepts for different domains becomes trivial in our approach. Of course, in the beginning platform experts need to work very close with domain experts where they are asked to implement those concepts that are needed to be used for the specific simulation. However, using our system's flexible definition engine they can use as many parameters as they want so that certain concepts/blocks can be reused later (e.g., loading a data input file, exporting trajectories of particles, defining interactions between n particles, etc), which will also serve the same goal in a completely different concept (domain). After defining initially the simulation basic operations (which can be also called the shared API), domain experts can start building logically integrated blocks from them (i.e., simulation steps), defining the relevant parameters.

To create a connection between simulation parameters and simulation results, we support *programmable workflows*, meaning domain experts can setup simulations that change their parameters automatically until a certain condition is met, enabling steered simulations and allowing input to optimization schemes – this goes well beyond what is typical in done in state-of-the-art MD-simulations.

Inquiry scientists and platform experts also need to collaborate once a platform is put into use, to ensure proper implementation. As we advance in the development of this tool there are more extra requirements that arise related to an easy-to-use interface, which is able to allow endusers to design complex, yet abstract particle structures and workflows. On the other hand, inquiry scientists can collaborate with domain experts to augment their knowledge and ultimately transition into domain experts themselves.

C. Advantages

The approach we introduced has several advantages compared to today's research habits. First, domain experts do not need to be experts with regards the simulators themselves, in particular, they do not need to know the coding syntax and the low-level tips and tricks of a specific platform; this is increasing important has the number of freely available, full featured simulators continues to grow. Instead, they can focus on their own tasks more directly related to the scientific goals. This is also advantageous, as documentation of freely available, ever developing simulators can often be sparse and difficult to understand for non-experts; this is where the knowledge of platform experts is particularly relevant. Additionally, by working in the MetaMDS framework, domain experts and inquiry scientists can avoid easy-to-make errors, like syntax errors or bad parameter order; this is a very important because researches can wait days or even weeks in queues, waiting for their jobs to execute, wasting considerable research time if a syntax error is made in the final simulation script.

Secondly, we are able to seamlessly keep all the platform-specific code up-to-date while still providing the ability to run using older versions; this allows new versions to be validated to ensure errors were not introduced as a result of the update. Similarly, this allows a standard suite of tests to be performed across multiple different simulators, to ensure consistency in their outputs. Furthermore, different simulators often perform better at certain tasks and worse at others, depending on the numerous factors such as the hardware available, nature of the chemical system, system size, etc. Switching seamlessly between platforms, we can optimize performance and decrease simulation runtime; this enables results to be achieved faster and at reduced cost to the end user (centralized high performance computing resources charge users based on runtime). Our approach focuses on flexibility: extending our system with a new platform is easy and relatively fast (if you have the right platform expert) and can improve the efficiency massively, e.g., by taking advantage of high performance GPUs, and by allowing the examination of systems in new domains (e.g., biophysics simulations).

III. Implementation

MetaMDS is a web application with a complex server-side backend system. We utilized the *JavaScript* scripting language both on client- and server-side. In the browser we built a standalone application using the *backbone.js* framework [7] which is based on the Model-View-Controller (MVC) pattern and is easily extendible and customizable. On the server side we use the *node.js* platform [8] (that is built on Chrome's JavaScript runtime) and *MongoDB*, a document-based, NoSQL database [9] to store our data.

Based on our introduction of our research on how a typical workflow is designed, we created a multi-level data-hierarchy that can provide an easy collaboration between different field-experts. On the top level, endusers interact with

 ISBN: 978-1-61208-351-3

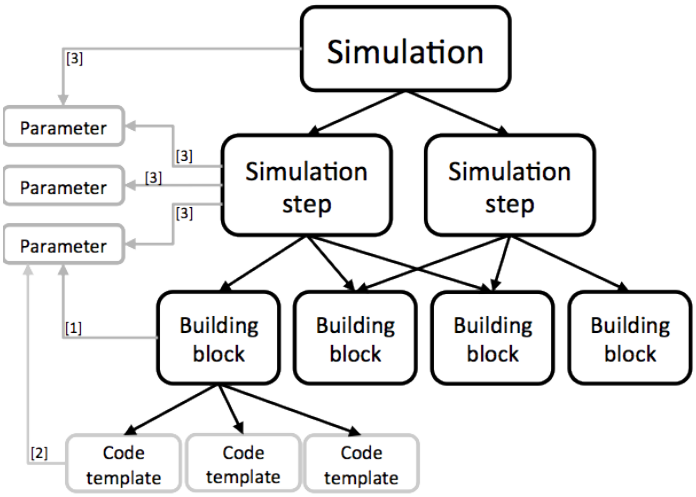

Figure 2. MetaMDS data model. The data structure we use for modeling MD-simulations. Parameter references: (1): Building blocks include parameter definitions; (2): In code templates, code references to parameters; (3): Simulations and simulation steps can set or override parameter values.

a visual programming language that provides them a clean user interface and supports not just importing the pieces of the concepts modeled by the domain experts, but they can build a complete control-flow with branches, conditions (if/else), loops (for/while), return values and of course setting parameter values. While manipulating program elements graphically rather than defining them textually, scientist can design simulations on a high-level, only having to find the suitable parameter-value combinations. Also, they are able to automate simulations (e.g., keep repeating this simulation with increasing a parameter value until a certain condition is met, e.g., *average potential energy* is less than a threshold). Endusers use built-in programming language elements that represent the basic language constructs and additionally use custom elements that were designed by the domain experts. These elements are called in our system *simulation* and *simulation step*.

In Figure 2, we introduce how our data is structured. Simulation steps are well-defined, reusable units that are used to construct simulations. As an example, domain experts can define simulation steps such as *Initialize TNP system*, *Adjust temperature* or *Collect data*. *Simulations* are defined with a limited visual language that doesn't allow control-flow statements, just setting parameter types and values. It is important to note that at this level, domain experts are able to handle the control allowed to the endusers by defining *variable-type* parameters or hardcoding in values. This means that only the data-type of the variable parameter (e.g., integer, double, boolean, string) needs to be defined, where setting the numerical value of the parameters is done one level higher (which can be set as a constant value or variable). With this approach we can provide a very flexible system that allows domain experts and inquiry scientists collaborate in a very unique way, where domain experts provided a specific template to the inquiry scientists.

At the base level of our data hierarchy we use the concept of *basic building blocks*. These blocks are defined by the domain experts on high-level and are implemented by the platform experts. A basic building block contains its basic properties (id, name, description), a list of parameters – where each parameter has its own identifier/name and a thorough description that can be used as a help while assembling higher-level constructs to locate/recognize what is it exactly for – and the actual platform-specific implementations. This implementation contains the actual code in the platform's language using its syntax and parameter notation. These code implementations work as code templates, which are evaluated by the server-side backend system (either before submitting the simulations to the chosen server/cluster, or when users want to download generated code to run on their local systems). After a platform expert has implemented the simulator specific code template to handle the parameters defined by the domain expert, it is used to build up the aforementioned simulation steps. In Figure 2, we highlighted three different types of pointers that point towards the parameters, which need further explanation. *Type 1* pointers mean a definition/containment: we define the descriptors of a parameter in a basic building block, that consists of its name, description, data type, default value, and a simulator identifier. In some cases there are building blocks that express the same concept but, due to simulator implementation differences, have additional parameters that are simulator-specific, thus we need to maintain the last property. *Type 2* pointers show *reference*-type connection: in a code template we need to reference to previously defined parameters (e.g., load $filename$ where $filename$ is a $string$ type parameter). *Type 3* connections are the most advanced references. It expresses that an entity contains a specific parameter with either a value (constant) or indicates that this parameter is a variable. In the variable case we also use the parameter's other properties to perform validations.

In an object-oriented world, we could map the items from our concept to the following entities: basic building blocks as code statements; simulation steps as functions that contain some/several statements; simulations as classes that contain functions and variables defined; our simulation repository as a class library; and the workflows that are built and submitted to simulator servers as programs/applications.

IV. Case Study

In this case study, we demonstrate how MetaMDS and its flexibility can save significant time for researchers. Let's suppose we wish to run simulations with a simple mono atomic fluid. In this case, we wish to determine under what conditions it would be most efficient to run LAMMPS vs. HOOMD-Blue. These two platforms are similar enough to express the same concepts, however their performance vary depending on the hardware and system studied. HOOMD-Blue supports GPU-based calculations, which tend to provide considerable performance over CPUs, while LAMMPS is designed to scale efficiently on large numbers of CPUs. To measure the performance of the different simulators, we use another metric that reflects their efficiency: time-per-step (TPS). We can define this metric as the time needed to perform one timestep during the simulation. The test simulation (*Test mono LJ*) consists of two simulation steps and we had to define 12 distinct building blocks to set up these simulation steps. For a platform expert, implementing these 12 code templates for HOOMD-blue and LAMMPS was accomplished within a few hours (including understanding the concepts we were using here). Only a single simulation needs to be constructed in

 ISBN: 978-1-61208-351-3

TABLE I. TIME-PER-STEP VALUES FOR DIFFERENT SIMULATOR PLATFORMS.

Num. of particles	TPS (ms) LAMMPS	TPS (ms) HOOMD	%
500	0.11736	0.16658	70.45131
1000	0.18470	0.18053	102.30979
5000	0.70105	0.21098	332.27690
10000	1.32672	0.23454	565.66885
50000	6.71378	0.73771	910.08082
100000	14.77878	1.48404	995.84767
500000	108.06365	7.04629	1533.62483

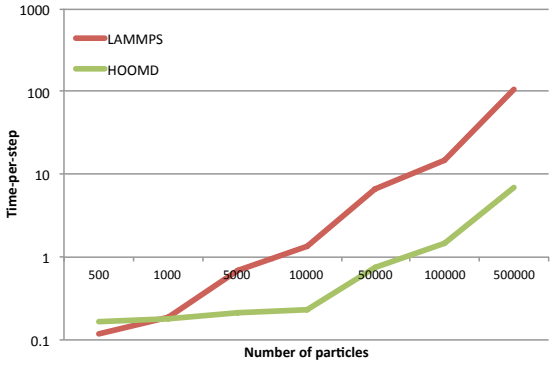

Figure 3. Comparison of the TPS values of different simulators.

MetaMDS, even though we wish to run using two different simulators. Table I reports the TPS calculated as a function of system size, where we note that smaller values of TPS are preferred. Figure 3 shows the graph comparing the two simulator systems tested. There is a clear transition at 1000 particles, for which HOOMD-Blue out performs LAMMPS, where we observe a 3 order of magnitude speed increase by using the GPU-enabled HOOMD-Blue. The first column of Table I indicates the number of the particles in our system, while the second and the third column show the time-per-step values for both simulators we tested (in milliseconds) and the last column shows the performance gain we can have switching from LAMMPS to HOOMD-Blue. While this is a relatively trivial example, the concept is generally applicable to benchmarking more complex systems and algorithms where GPU performance is less significant.

A. Code reusability

Another advantage of the approach used in MetaMDS is code reusability and interchangeability. While the example above focused on a system with only a single particle type, we can reuse the basic simulation workflow for a system composed of two particle types. To accomplish this, we only had to define a new system initialization step that handles two different types of particles. To use this simulation step, we need only to exchange this initialization step in our first simulation instance and then be able to perform simulations using a binary system. The changes are shown on Figure 4. Again, this is a relatively simple example, but the concepts demonstrated are general. For example, a more complex simulation workflow, with many different simulation steps and procedures could be developed and used for one system and then reused for a different system by only changing the step that initializes

Figure 4. Simulation initialization steps for mono- and binary-atomic systems. The rest of our simulation is untouched while we can run different simulations in different domains with very small changes.

the system. This is the general idea that facilitates the collaboration of domain experts; e.g., a domain expert could design the workflow needed to calculate the viscosity of an alkane system, which could then be trivially merged with the system initialized by an expert in grafted nanoparticles to facilitate the calculation of viscosity of grafted nanoparticles. This example also demonstrates the power of being able to present the end user with abstract representations. While the binary system is still simple, two additional interactions were needed to be defined to initialize the system; a simple model of a grafted nano particle [10], would require three more pair interactions, two additional bond parameters, one angle parameter and one dihedral parameter whereas a self-assembled monolayer system requires in excess for 50 additional model parameters. As such, this approach facilitates reusability of not only the general workflow, by e.g., swapping in a different model, but also would enable reuse of models, which is particularly important as system complexity grows.

We performed our tests on a desktop computer with an Intel Core i7 4820K CPU and an NVIDIA GeForce GTX780Ti - 3GB - EVGA Superclocked graphics card.

V. RELATED WORK

Within the MD simulation domain, the objectives of MDAPI [11] are closest to those of our system. MDAPI aims at creating a unified application programming interface that shields the specifics of the particular simulator tools. It is designed for biophysical simulation, where the interface and computational engines are separated, and consequently the API is geared toward the needs of that domain. The most important differences between MDAPI and our work is that a.) MDAPI defines a fixed API, while the set of concepts that serve as the interface between the simulator and the domain expert may evolve over time; and b.) MetaMDS provides two distinct abstraction levels (the basic operations describing the basic concepts, and simulation steps representing high-level

 ISBN: 978-1-61208-351-3

operations that capture and hide knowledge specific to the chemical domain), while MDAPI offers only one.

The Atomic Simulation Environment (ASE) [12] is a Python-based tool that can connect to many different simulation codes as *calculators* you plug into the environment. Its primary target is quantum mechanical calculations and thus, it is not directly applicable for most MD simulations. The power of ASE lies in its tool integration capabilities: through the use of the Python programming language it it possible to link different toolkits together, e.g., plotting and visualization libraries, etc.

Etomica [13] is a molecular simulation code written in Java, enabling it to be easily used and distributed via the web. Etomica is similar to our approach in a sense that it defines a molecular simulation API that hides the low-level details of running MD simulations, which allows simulations to be build from predefined pieces. However, it does not offer the flexibility that MetaMDS provides through its metaprogramming functionality. Creating simulations in Etomica requires Java programming expertise: the end users are limited to executing prewritten modules with custom parameter settings.

MetaMDS can be thought of as a science gateway for MD simulators. In this sense, the most closely related tool to our work is the Nanohub [14]. The Nanohub provides a VNC-based interface to a variety of simulation tools hosted as cloud instances. The complexity of the variety of simulators is addressed through simplified user interfaces: the user is only presented with a limited subset of options to help guide the simulations. Most of the modules have a consistent look and feel, so the learning curve is reasonable. Visualization and plotting tools are often built into the GUIs. Jobs are submitted to clusters and the results copied back to the Nanohub space. Unfortunately, Nanohub has its set of limitations. The VNC-based user interface is not very responsive. User-level customization is not supported. The user can only change the parameters that Nanohub includes in its simplified interface. There is no interaction supported between various tools: the output of one simulator cannot be trivially fed to the input of another. Similarly, the primary mode of operation is interactive, since most tools have been developed with education in mind, and thus submitting a large set of jobs is not easily accomplished.

VI. Conclusion

We have demonstrated the design principles associated with our MetaMDS collaborative simulation environment, built upon the ideas of model integrated computing. This tool enables the creation of a flexible "API" for molecular simulation, allow any number of simulation platforms to be run with only a single simulation workflow. Furthermore, our tool provides ways to group common procedures and concepts used within molecular simulations, to enable the reuse and sharing of simulation models and procedures. These common simulation blocks can be pieced together into larger simulation templates to accomplish tasks within a specific domain. This approach enables experts in the simulator platforms to work with experts in a give domain to great simulations for use by inquiry scientists that perform virtual experiments. By their construction, these simulation templates can be used to limit the number of parameters available to end users, to provided an error-free, guided experience. With the flexible simulation parametrization we can also use MetaMDS as tool for teaching both the concepts of simulation and for enabling students to use simulation as a means of understanding molecular level interactions in systems. After setting up several simulations with variable-type parameter values, professors can allow their students or new researchers in their groups to access domain-specific parameter values, enabling efficient simulation of systems within a targeted range of variables. Overall, our tool enables the collaboration between users with different areas and levels of expertise, allows for the seamless integration of different simulator toolkits, and collaboration between end users by creating a platform for the reuse of simulation models and procedures.

Acknowledgments

The work presented in this paper was supported by the National Science Foundation grants NSF CBET-1028374 and NSF OCI-1047828.

References

[1] G. Varga, S. Toth, C. R. Iacovella, J. Sallai, P. Volgyesi, A. Ledeczi, and P. T. Cummings, "Web-based metaprogrammable frontend for molecular dynamics simulations," in 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), Reykjavik, Iceland, 07/2013 2013.

[2] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," Proceedings of the IEEE, vol. 91, no. 1, 2003, pp. 145–164.

[3] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," Journal of Computational Physics, vol. 117, no. 1, 1995, pp. 1 – 19. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002199918571039X

[4] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," Journal of Computational Physics, vol. 227, no. 10, 2008, pp. 5342 – 5359. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999108000818

[5] P. Akcora, H. Liu, S. K. Kumar, J. Moll, Y. Li, B. C. Benicewicz, L. S. Schadler, D. Acehan, A. Z. Panagiotopoulos, V. Pryamitsyn, V. Ganesan, J. Ilavsky, P. Thiyagarajan, R. H. Colby, and J. F. Douglas, "Anisotropic self-assembly of spherical polymer-grafted nanoparticles," Nat Mater, vol. 8, no. 4, 04 2009, pp. 354–359. [Online]. Available: http://dx.doi.org/10.1038/nmat2404

[6] T. Song, S. Dai, K. Tam, S. Lee, and S. Goh, "Aggregation behavior of two-arm fullerene-containing poly(ethylene oxide)," Polymer, vol. 44, no. 8, 2003, pp. 2529 – 2536. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0032386103001071

[7] "Backbone.js javascript framework," http://backbonejs.org [accessed 05/2014].

[8] "Node.js platform," http://nodejs.org/ [accessed 05/2014].

[9] "MongoDB document database," http://www.mongodb.org [accessed 05/2014].

[10] C. Iacovella, G. Varga, J. Sallai, S. Mukherjee, A. Ledeczi, and P. Cummings, "A model-integrated computing approach to nanomaterials simulation," vol. 132, no. 1, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00214-012-1315-7

[11] "MDAPI web page," http://www.ks.uiuc.edu/Development/MDTools/mdapi [accessed 05/2014].

[12] "Atomic Simulation Environment web page," https://wiki.fysik.dtu.dk/ase/ [accessed 05/2014].

[13] "Etomica web page," http://etomica.org/ [accessed 05/2014].

[14] "NanoHUB web page," http://www.nanohub.org/ [accessed 05/2014].

 ISBN: 978-1-61208-351-3