SCALABLE AND USER-FRIENDLY SIMULATION

Adrian Rutle*, Hao Wang[†‡], Robin T. Bye [†§], and Ottar L. Osen[†‡§]
* Department of Computing, Mathematics and Physics,
Bergen University College, Norway.
Email: aru@hib.no
[‡] Big Data Lab and [§] Software and Intelligent Control Engineering Lab,
[†] Faculty of Engineering and Natural Sciences,
Aalesund University College, Norway.
Email: {hawa,roby,oo}@hials.no

KEYWORDS

Scalability; Parallel and GPU-programming; Agent-based modelling and simulation; Virtual Prototyping; Model-driven software development; Domain-specific languages.

ABSTRACT

Simulation is an important technique for integrating interacting models for predicting results of hypothetical scenarios. A typical application area for simulators is virtual prototyping (VP). In VP, simulators replace the real-world prototype. Hence, the quality of the virtual prototype depends on the quality of its simulations, which in turn are highly dependent on the quality of the models and the computational power, especially if visualization and/or real-time constraints are required. Unfortunately defining models is an error-prone activity which requires domain-experts to have knowledge about the implementation details and/or IT-technical concerns. In addition, the bigger the dataset, the more computational power is needed, which affects the cost, and in turn, the usability of today's simulators. To address both of these aspects, we propose a user-friendly, adaptive and scalable agent-based modelling and simulation framework with a hybrid CPU/GPU/FPGA high performance computing platform. The solution we describe provides domain-experts with a a scalable, adaptive, and efficient simulator and enables domain-experts to define high quality models without in-depth IT-knowledge. We use a running example from the particle transmission domain to illustrate our approach.

INTRODUCTION

In this paper we propose a user-friendly, adaptive and scalable agent-based modelling and simulation (ABMS) framework on a hybrid CPU/GPU/FPGA high performance computing (HPC) platform. An agent-based model (ABM) is a class of computational models in which multiple autonomous agents act and interact, typically guided by some set of simple rules, such that some collective behaviour emerges. Using this paradigm for simulation, it is possible to recreate and predict a number of complex phenomena observed in nature, e.g., a flock of birds, a herd of land animals, or a school of fish (Reynolds, 1987).

Address for correspondence: Adrian Rutle, Department of Computing, Mathematics and Physics, Bergen University College, Postboks 7030, NO-5020 Bergen, Norway.

Agent-Based Modelling and Simulation

In general, an agent-based simulation process is based on the emergence principle where the lower (micro) level of systems determines a higher (macro) level of behaviour (e.g., Bonabeau, 2002). Thus, simple behavioural rules generate complex behaviour. Typically, agents are characterised as rational; i.e., they are expected to be acting according to what they perceive as their own interests, such as reproduction, suitable environment, or economic benefit, using heuristics or simple decision-making rules (Bonabeau, 2002).

When designing an ABM, we need to consider the following:

- granularity of the agents: how many agents are needed to get a realistic behaviour.
- scalability: how well do the number of agents and the resolutions of time and space scale when compared with reality.
- decision-making rules and heuristics: what an agent does next based on a set of parameters given by the environment and the agent's interests.
- behaviour over time: do the agents learn or are they adaptive.
- topology: how the agents are related to each other and how they interact with each other.
- the environment: how the environment affects the agents' behaviour.

While ABMs perhaps are mostly associated with computer science, they are also used to model other domains such as biology, ecology and social science (Niazi and Hussain, 2011). Although ABM has good potential, it requires domain-experts to have know-how about the computational overheads involved with agents in terms of time and space and their distribution in networks or grids. Adding to the complexity is the fact that the location of the agents in the environment and their responsive behaviour are encoded in algorithmic form in computer programs.

Virtual Prototyping and Simulation

Virtual prototyping (VP) is defined as the computer-aided construction of digital product models and realistic graphical simulations for the purpose of design and functionality analyses in the early stages of a product's development process (Pratt, 1995). These prototypes are usually computer simulations of physical products that can be presented, analysed, and tested in various aspects like design and engineering, manufacturing, service and

Proceedings 29th European Conference on Modelling and Simulation ©ECMS Valeri M. Mladenov, Petia Georgieva, Grisha Spasov, Galidiya Petrova (Editors)
ISBN: 978-0-9932440-0-1 / ISBN: 978-0-9932440-1-8 (CD)

recycling, as if the prototypes were real physical models (Wang, 2002). One of the major aspects of VP consists of the representation (i.e., modelling) of various parts of the prototype. Using VP, one can easily construct (or manufacture) a set of prototypes with different configurations, behaviour and parameters. Another aspect is the simulation of these prototypes for analysis and experimentation purposes. VP engineers are experts in their domains, however, good abstraction of software technicalities, good modelling principles, and efficient simulation configurations are also necessary in order to make real advances in VP.

Domain-Specific Modelling Languages

A domain-specific modelling language (DSML) is a high-level computer language specialised to the application domain of modelling (e.g., see Fowler, 2010), with a focus on describing *the what*, not *the how*. We have designed a DSML which enables domain-experts to define models of agents without knowledge of technical details of the computations. Consequently, the domain-experts can focus on the description of the problem which they want to simulate. We use model-driven software engineering (MDSE) (Brambilla et al., 2012), in particular a metamodelling approach, for the definition of the DSML. Please note that the description of the DSML and the running example in this paper should be seen as a recipe for how to create a modelling language which is simple enough for domain-experts to deal with, and in addition formal enough for simulator engines to simulate models defined in these languages. That is, we outline here a conceptual framework for creating user-friendly DSMLs rather than presenting a full-scale modelling language.

Motivation and Aim

It is generally costly to develop ad-hoc simulators tailored for separate test cases, and current simulators are often too specific to be adaptable if we want to investigate different what-if scenarios. This is one of the reasons why simulation is not seeing a more widespread use in research projects although most researchers know the benefits and the insights one can get from simulation. Another reason is that deploying or even using simulators often require too much technical overhead which makes their usage less viable for the domain-experts. Our approach for tackling this problem is based on the development of a generic simulator able to abstracting the details away from the users. Among other features, the simulator should provide a user-friendly interface which enables domain-experts to define simulation scenarios without knowledge of the implementation details.

Another challenge is that of scalability. It is well-known that the result of a simulation is highly related to the granularity of the agents, the size/complexity of the existing data/environment, and generally the resolutions of the simulated time and space representations. If we have enough data about the environment, one can simulate the system in a near-realistic way. Employing analytical models is impractical because changes in settings have limitations in modelling important details and features of real world complex systems. Simulation models, however, provide the flexibility to accommodate arbitrary stochastic elements, and generally allow modelling of all the required complexities and dynamics of real world applications without resolving to undue simplifying assumptions. Obviously, however, a large amount of data and calculations will require a lot of processing power, which can be a challenge and may require optimisation at various levels.

In this paper, methods and approaches of simulation and of the solution of optimisation problems shall be linked to solve optimisation problems faster or make the obtained solutions more usable (under realistic conditions). Moreover, in most cases one wants to get feedback on changing different parameters almost in real-time. Our aim for addressing these challenges is based on using a hybrid CPU/GPU/FPGA HPC platform to design and develop a scalable simulator engine.

In the following, we outline our descriptions of a DSML with a running example and discuss aspects of user-friendly scenario definition; then we present our approach to a scalable simulator using a hybrid HPC platform, before we refer to some related work. Finally, we draw some conclusions and point to future work.

DOMAIN-SPECIFIC MODELLING LANGUAGE

First we introduce a running example, then we explain the modelling language which is used by the domain-experts to define their models and simulation scenarios. We adopt a MDSE methodology to develop a user-friendly DSML that enables domain-experts to define their models and simulation scenarios. MDSE is a branch of software engineering in which models are the first-class entities of all the phases of the development process. In MDSE one can define a modelling language as a metamodel and automatically generate support tools for the language, such as an editor and a syntax checker, using various language workbenches and frameworks (e.g., Epsilon (Kolovos et al., 2005), Eclipse Modeling Framework (EMF) (Steinberg et al., 2008), MetaEdit (MetaCase, 2007), or Diagram Predicate Framework (Rutle, 2010)). A metamodel is a model which defines the types and relationships between types used in a modelling language. We say that a model which is defined by the modelling language conforms to this metamodel.

Running example of Borgundfjorden Simulator

In the running example we consider the local fjord Borgundfjorden as a case-study. A big amount of data and a number of models already exist in different formats at different institutions in the geographical area surrounding the fjord. For example, the municipality of Aalesund has sewage data; a company called PatoGen has data about different kinds of viruses and micro organisms and their concentrations in the fjord; Havforskningsinstituttet has data on waves, temperatures and currents; and the Virtuelle Møre of Aalesund University College (AAUC) has data on oceanographic maps. Integrating all these models and creating a realistic model of the fjord is one of the important goals for many researchers and domain-experts in research areas such as biology, fish farms, particle transmissions, and urban management. We use a simulator for integration of these data, mainly because most of these data need to be put in context in order to get an overall picture of what is going on in the fjord. In addition, using a simulator can help us in finding answers to interesting research questions and hypotheses, and could be used to find patterns and relationships which we have not thought of yet.

The presented example, the methodology and the data integration could easily be generalised for usage in other fjords, or even other environments. For example, we could build on our previous work (Alaliyat et al., 2013; Bye, Rutle, Stene and Yndestad, 2015), where we among other things used an agent-based model to simulate pathogen transmission between aquaculture sites in the Romsdalsfjord (see Figure 1). As noted in our literature review of VP in (Bye, Osen and Pedersen, 2015), VP can also be used to facilitate integrated planning and visualisation of large construction projects. Using our integrated simulator as a VP tool for the planning of new fish farms, public management and/or private companies could take into account factors such as pathogen transmission between fish farms related to ocean currents such as those in Figure 2 and be able to optimally position the new fish farms to reduce the risk of fish disease and provide optimal living conditions for the fish by simulating a large number of what-if scenarios involving pathogen particles, vessel traffic related to spreading of pathogen, more fish farms in the future, urban and industrial sewage, and so on (Bye, Rutle, Stene and Yndestad, 2015).

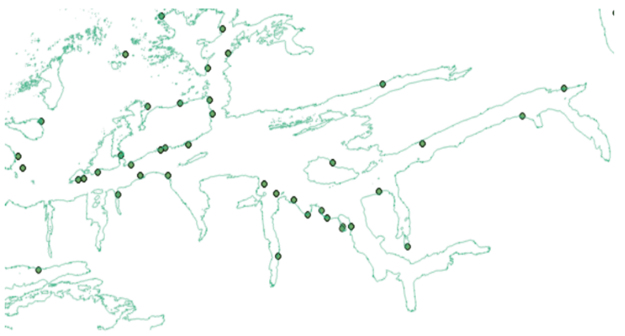
Figure 1: Aqua farms in Romsdalsfjorden. Courtesy of Alaliyat et al. (2013).

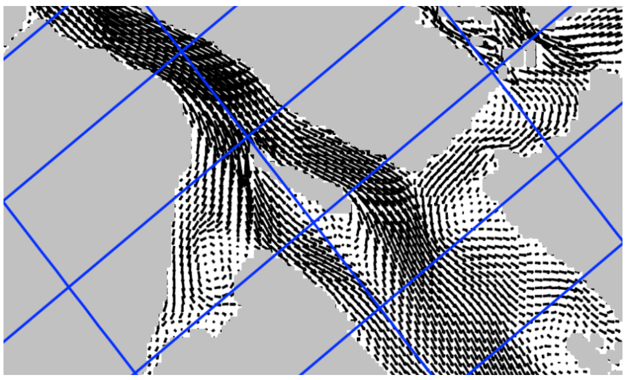
Figure 2: Snapshot of ocean currents in Romsdalsfjorden. Courtesy of Alaliyat et al. (2013).

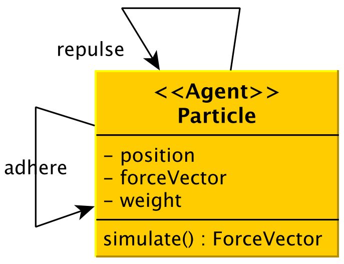

Figure 3: A metamodel of the sample modelling language used for modelling particles.

Metamodel

As a simulation case, a marine biologist may want to see the impact of a particular particle type or micro organism transmission through Borgundfjorden at a given time of the year. With the DSML and simulator we describe here, the domain-experts would only need to describe the particles of interest and their properties, for example their weight, their start position, and other properties like which particles/surfaces they are likely to adhere to or repulse, and then run the simulation for a given time into the future, or in the past. Designing the metamodel of the DSML is an activity which is usually delegated to the language designers or specialised IT-personnels who tightly cooperate with the domain experts. A simplified metamodel for the simulation case is shown in Figure 3. From this metamodel, we can generate a language editor so that domain-experts can define models representing simulation cases.

Language Editor

As a sample model, consider a set of particles A, B, C, D, and E, where A could adhere to B and E, while it is repulsed by C and D. In addition, B could adheres to D and C adhere to E. This information is then captured as instances of the metamodel shown in Figure 3. A sample language editor generated based on the metamodel in Figure 3 is shown in Figure 4. The figure shows a simplified view of the model and the language. The palette area shows the types which were defined in the metamodel, and the drawing canvas shows the particles and their relationships. Furthermore, using the language enables the domain-experts to see and define other interesting relationships in the model, for instance, how the relations between the particles A and B as well as B and C affect the relation between A and C.

Focusing on What Rather than How

Using this language, even the fact that the simulation is agent-based is hidden from the domain-experts. The focus of the experts is on the description of the domain, not how things are represented. Internally, these models are translated to agents which run in the simulator and predict the behaviour of the system. This translation to implementation is described in more details in later sections.

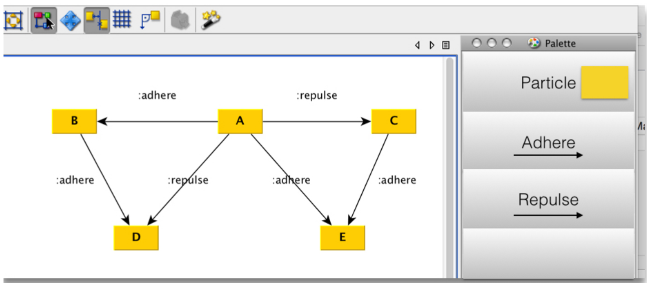

Figure 4: The language editor corresponding to the metamodel in Figure 3 and a sample particle model defined by the modelling language.

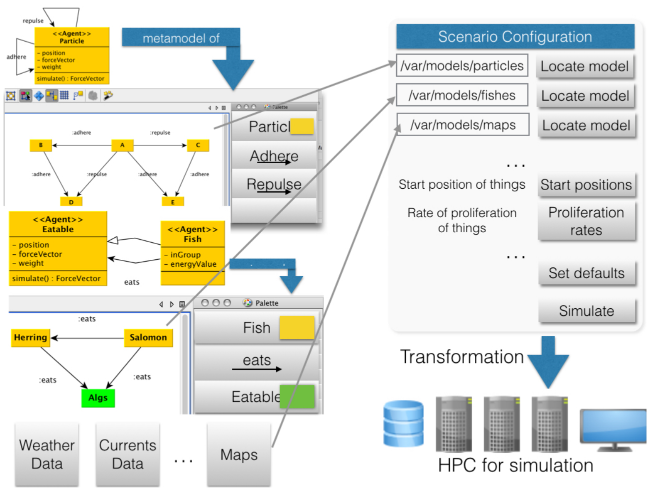

Figure 5: A general architecture for model integration and simulation.

USER-FRIENDLY SCENARIO DEFINITION

The responsibility and usage of the DSML is limited to the definition of the models. The simulator should also have an interface which enables domain-experts to integrate data and stage their scenarios. For instance, the simulator could have a module for choosing and uploading a weather model (forecast or historical data) for a given position in a given time period, a module for choosing wave and current models for that same time period and position, and one for choosing the oceanographic maps. In addition to these (relatively) static data, the domain-expert should also be able to upload the particle model (see Figure 4) as well as a model describing different kinds of fishes and their eating habits, and use the simulator interface to define a start position for the particles (perhaps with different positions for different particle types), a rate of proliferation (how many particles are put into the simulator per time unit, for how long, and when), and which start direction and speed the particles have. Figure 5 depicts a general architecture of the framework in which various models can be integrated and configured for simulation. The simulator should be able to generate all this information using high-quality pseudo-random number generators (PRNGs), as well as reproduce simulations by fixing the seed of a PRNG.

Next for the domain-expert is to choose which algorithms to be used for the calculation of the next positions for each particle, and how detailed and accurate the simulation must be run. Again, defaults should be proposed by the simulator. There are other applications of simulation which makes the design of a user-friendly interface for the simulator necessary. For instance, making observations on whether the simulation results match the real results from past real world experiments and data would strengthen to what extent one can trust the simulation results. This could be put in a machine learning context to train the simulator in order to come up with more precise results.

Finally, we note that domain-experts generally can not be expected to be programmers; they are cyberneticians, mechanical engineers, electrical engineers, zoologists, microbiologists, meteorologists, etc. They know their own domain, and in most cases work together in order to define simulation scenarios. We do not expect the domain-experts to use command-line, or to be familiar with artificial intelligence, machine learning or other algorithms. On the other hand, we do expect that domain-experts have a reasonable minimum knowledge on what simulation is, how it works, what could be expected from simulation, what is needed as input, and in which cases one can choose provided defaults, and so on. Thus, user-friendliness requires that the concepts and constructs in the modelling language are those which are used by the domain-experts. Usually, graphical languages are easier to understand by domain-experts, however, any language which has concepts that come directly from the domain may be considered user-friendly. In cases where a graphical syntax may add complexity, one may choose to add a textual syntax for the language. As additional features, domain-experts (or simulation-experts) could use meta-tools to reason about the results of their simulations, for instance, how accurate or realistic the simulations are, whether the results can be trusted, and whether the results be compared to/aligned with actual observations.

LARGE SCALE AGENT-BASED SIMULATIONS WITH HPC PLATFORM

In this section we outline our approach to the development of a scalable, generic simulator which is able to do high quality simulations efficiently. The approach is based on a fair and efficient scheduling algorithm to schedule computational resources in distributed simulator engines.

In order for ABMS to be (near-) realistic, the simulation needs to deal with large amount of agents, which incurs heavy computational workload. As we have observed in the HPC community, super computers have already been built with a hybrid CPU and GPU architecture to make use of the large pool of processing units in GPUs, e.g., Titan, a super computer built in Oak Ridge with a combination of GPUs and traditional CPUs, was ranked No. 1 in the Top500.org November 2012 list (TOP500.org, 2012). In addition, we have seen in recent years fast development of FPGAs, the performance of which starts matching that of GPUs with much less power consumptions (Kestur et al., 2010). For this reason, FPGAs are being adopted in new HPC architectures. More importantly, the new OpenCL 2.0 standard makes it much easier to program across different CPU, GPU, and FPGA architectures (Abdelfattah et al., 2014; Chen and Singh, 2012; Segal et al., 2014). We aim to experiment with different configurations considering the advantages and disadvantages of CPU, GPU, and FPGA.

Implementing an agent-based simulator runtime with disparate data sets requires complex scheduling decisions. CPU and communication resources needs to be measured, and the computation of agents moved to the appropriate cluster node.

The simulator engine must adapt dynamically to the hardware available, scaling the simulation according to current work load and real-time needs. By abstracting away much of the concurrency and communication complexities, the simulator can facilitate time and cost effective development of new simulators. These new simulators can run in parallel, separate or together, shining new light on complex problems.

Large scale simulations require massive amount of computing power. If the dataset grows in length (with time), the CPU requirement typically increases linearly, whereas adding more parameters, objects, agents, and so on, likely yield an exponential increase in the required computing power. In order to get sufficient resolution in its simulation results, the model must be able to incorporate more than 100 million concurrent agents. Scheduling algorithms can be utilized to ensure that the computation of an agent is localized to the cluster node most capable to its needs. Agents in need of high memory bandwidth and/or floating point calculations are automatically run on GPU powered nodes. The complexity of the distributed nature of the simulator core is abstracted away from the researcher, leaving them free to concentrate on the research problems.

Features of the simulator engine

Among the most important features of the simulator engine we can mention:

- Storage and management system for research data: Support storage of large datasets from various research groups. There will be a pluggable architecture to translate and import the various datasets into the simulator core. We design and develop a website and database system which can be used for upload, download, export and import of data.
- Simulator core: The simulator core will dynamically scale its performance according to the number and type of hardware nodes available. A particular simulation run on the simulator engine core is broken down into modules called agents. The communication and scheduling module will do real-time measurements of the current running simulation and dynamically redistribute the agents according to their calculation and communication needs.
- Support time-efficient implementations of new simulators: Concentrating on ease of use, the simulator toolkit will make simulations a viable research tool for a larger group of researches.
- Visualization tool: the simulation engine will utilize existing commercial and open 3D engines to support visualizations of the simulations. We will investigate various 3D engines in our new 3D visualization laboratory in Aalesund University Collage.

RELATED WORK

Modelling and simulation is gaining an increasing interest due to the access to powerful, cheap computers. The literature on modelling and simulation research could be divided into two subgroups: Those that focus on the software tools and APIs dedicated for running simulations, and those that focus on representation of data and modelling techniques. We will discuss some tools and frameworks related to these two groups below.

The Agent Modeling Platform

The Agent Modeling Platform (AMP) (Agent Modeling Platform, n.d.) is an Eclipse Project which focuses primarily on providing tools for ABM. In AMP, extensible frameworks and exemplary tools are provided for representing, editing, generating, executing and visualising ABMs and any other domain requiring spatial, behavioural and functional features. The two main AMP components are Agent Modeling Framework (AMF) used for modeling of agent systems; and (AXF, AGF and Escape) used for execution and exploration of those agent systems. AMP may generate code (from models) for target platforms like Repast Symphony, Escape and Ascape (Macal and North, 2009; North and Macal, 2007; Inchiosa and Parker, 2002) , and other platforms could be supported with minor efforts. However, AMP does not follow any of the FIPA (Poslad, 2007) guidelines and standards.

The Java Agent Development Environment

Java Agent Development Environment (JADE) (Bellifemine et al., 2007) is a completely distributed middleware system with a flexible and extensible infrastructure. The framework provides a run-time environment for agent-based applications that implements the life-cycle support features required by agents, the core logic of agents themselves, and a rich suite of graphical tools. Since JADE is written in Java, it benefits from Java language features and third-party libraries, and thus offers a rich set of programming abstractions allowing developers to construct JADE multi-agent systems with relatively minimal expertise in agent theory. However, while JADE is only a middleware system facilitating implementation of agents, it does not provide a simulation infrastructure/engine for running the simulations. Furthermore, JADE is designed to be used by (Java) programmers, meaning that other scientific staff must rely on programmers and agent engineers to develop and run simulations.

Modelica

Modelica is an object-oriented equation-based modelling language suited for modelling complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, control, etc., subcomponents (Fritzson, 2015). Modelica comes with a large set of reusable models and functions from various domains. In addition, models defined using the Modelica language can be simulated with a large number of both commercial and free simulation environments. Furthermore, simulation of complex environments containing a large number of agents (especially when the agents are supposed to be simulated using different simulation tools) is identified as a challenge which modelling and simulation environments like GridLAB-D try to solve using Functional Mockup Units (FMU) (Elsheikh et al., 2013).

Our Approach

In our approach, we abstract away from programming details, FIPA standards and simulation infrastructure by proposing DSMLs suitable for the domain-experts. Then, we use model transformation techniques, based on our knowledge on MAS and HPC simulations, to generate agent code which will be following FIPA standards and well-proved simulation guidelines. Some of the relevant work that we can build upon for our approach is listed below.

Hybinette et al. (2006) developed a middleware paired with a standard parallel discrete event simulation kernel so as to realize distributed and parallel ABMS. Parker (2007) created a distributed Java-based ABM for disease transmission and managed to simulate 300 millions agents with 38 GB memory in about 2 hours. Mengistu and Troger (2008) analysed large scale ABMS in grid execution infrastructures.

Recently, GPU and FPGA are attracting attention in the simulation community. Oak Ridge National Lab advocates the use of GPU and realized ABMS using a cluster of GPUs using a combination of MPI and CUDA (Perumalla, 2009; Aaby et al., 2010), they obtained 30x speedups compared to a CPU-based implementation. Cui et al. (2011) used FPGA and achieved a speedup of 290x with 2 millions agents, compared to the C implementation.

CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a conceptual framework for the definition of domain-specific modelling languages (DSMLs) which are of particular usage for modelling multi-agent-based systems. Along with a user-friendly configuration of simulation scenarios, we believe that scientists with no particular knowledge in programming can benefit from using our methodology for both definition of complex models and simulation of these models. A hybrid CPU/GPU/FPGA HPC platform makes large scale simulations with millions of agents feasible.

By the use of our proposed framework it will be easy to define models that for instance may be used in Virtual Prototyping (VP). VP is not limited to testing a thing or a physical object prior to building it, it can as well be applied to a technique or a method. In our specific case, the Borgundfjord, our model could be used with VP techniques to find good cleanup procedures or to find perfect locations for fish farms.

Another research question is related to the use of the gathered and integrated data from experiments to search for patterns, which may or may not be thought of by the domain-experts. For instance, there may be a relationship between a certain type of boat traffic and the transmission of certain diseases, or there may be a relationship between a certain weather condition and a certain pattern of transmission, etc. This is a problem addressed by the big-data domain, which, given an event-log, one can find possible relationships between different events, or between causes, effects and consequences of different events. We would like to address this idea in future work.

Data and models from different domains inevitably will come in a variety of formats and level of details. Integration of these models into a common simulation scenario represents a challenge. We consider two options for transforming the knowledge described in these models; either (i) a single simulation system with distributed agents in a grid, or (ii) several simulation systems with distributed agents which communicate using co-simulation techniques. Techniques involving a functional mockup interface (FMI) and function mockup units (FMUs) may prove useful or even necessary and will be investigated in future work.

ACKNOWLEDGEMENT

The authors wish to thank Arne Styve at Aalesund University College for his valuable insight and comments in the preparation of this manuscript.

REFERENCES

Aaby, B. G., Perumalla, K. S. and Seal, S. K. (2010), Efficient simulation of agent-based models on multi-GPU and multi-core clusters, SIMUTools 2010: 3rd ICST International Conference on Simulation Tools and Techniques, ACM, pp. 29:1–29:10. http://dl.acm.org/citation.cfm?id=1808143.1808181

Abdelfattah, M. S., Hagiescu, A. and Singh, D. (2014), Gzip on a Chip : High Performance Lossless Data Compression on FPGAs using OpenCL, IWOCL 2014: 2nd International Workshop on OpenCL, ACM.

Agent Modeling Platform (n.d.), *Project Web Site*. http://www.eclipse.org/amp/.

Alaliyat, S., Osen, O. L. and Kvile, K. O. (2013), An Agent-Based Model To Simulate Pathogen Transmission Between Aquaculture Sites In The Romsdalsfjord., ECMS, pp. 46–52.

Bellifemine, F. L., Caire, G. and Greenwood, D. (2007), *Developing Multi-Agent Systems with JADE*, Wiley.

Bonabeau, E. (2002), Agent-based modeling: Methods and techniques for simulating human systems, *Proceedings of the National Academy of Sciences* 99(suppl 3), 7280–7287. http://www.pnas.org/content/99/suppl_3/7280.abstract

Brambilla, M., Cabot, J. and Wimmer, M. (2012), *Model-Driven Software Engineering in Practice*, 1st edn, Morgan & Claypool Publishers.

Bye, R. T., Osen, O. L. and Pedersen, B. S. (2015), A simulator for intelligent virtual prototyping of offshore cranes, Proceedings of the 29th European Conference on Modelling and Simulation (ECMS'15). To appear.

Bye, R. T., Rutle, A., Stene, A. and Yndestad, H. (2015), Nautilus21: A generic, integrated, and scalable 3D ocean simulator for scientific exploration and management of Norway's coastal waters and fjords, Proceedings of the 2nd Fjord Conference (Fjordkonferansen 2014), Fjordkonferansen, To appear.

Chen, D. and Singh, D. (2012), Invited paper: Using OpenCL to evaluate the efficiency of CPUS, GPUS and FPGAS for information filtering, FPL 2012: 22nd International Conference on Field Programmable Logic and Applications, IEEE, pp. 5–12.

Cui, L., Chen, J., Hu, Y., Xiong, J., Feng, Z. and He, L. (2011), Acceleration of Multi-Agent Simulation based on FPGA, FPL 2011: International Conference on Field Programmable Logic and Applications, IEEE, pp. 470–473.

Elsheikh, A., Awais, M., Widl, E. and Palensky, P. (2013), Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface, Modeling and

Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on, pp. 1–6.

Fowler, M. (2010), *Domain-Specific Languages*, Addison-Wesley Professional.

Fritzson, P. (2015), *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*, 2 edn, Wiley, Hoboken, NJ.

Hybinette, M., Kraemer, E., Xiong, Y., Matthews, G. and Ahmed, J. (2006), SASSY: A Design for a Scalable Agent-Based Simulation System using a Distributed Discrete Event Infrastructure, WSC 2006: the Winter Simulation Conference, pp. 926–933.

Inchiosa, M. E. and Parker, M. T. (2002), Overcoming design and development challenges in agent-based modeling using ascape, *Proceedings of the National Academy of Sciences* 99(suppl 3), 7304–7308.

Kestur, S., Davis, J. D. and Williams, O. (2010), BLAS Comparison on FPGA,CPU and GPU, ISVLSI 2010: IEEE Computer Society Annual Symposium on VLSI, IEEE Computer Society, pp. 288–293. http://caxapa.ru/thumbs/282284/ISVLSI_FINAL.pdf

Kolovos, D., Rose, L. and Paige, R. (2005), *The Epsilon Book*. http://www.eclipse.org/gmt/epsilon/doc/book/.

Macal, C. M. and North, M. J. (2009), Agent-based Modeling and Simulation, A. Dunkin, R. G. Ingalls, E. Yücesan, M. D. Rossetti, R. Hill and B. Johansson, eds, Proceedings of the 2009 Winter Simulation Conference, WSC 2009, Hilton Austin Hotel, Austin, TX, USA, December 13-16, 2009, WSC, pp. 86–98. http://dx.doi.org/10.1109/WSC.2009.5429318

Mengistu, D. and Troger, P. (2008), Performance Optimization for Multi-agent Based Simulation in Grid Environments, CCGRID 2008: 8th IEEE International Symposium on Cluster Computing and the Grid, IEEE, pp. 560–565.

MetaCase (2007), Metaedit+ workbench. http://www.metacase.com/mwb/

Niazi, M. and Hussain, A. (2011), Agent-based computing from multi-agent systems to agent-based models: a visual survey, *Scientometrics* 89(2), 479–499. http://dx.doi.org/10.1007/s11192-011-0468-9

North, M. J. and Macal, C. M. (2007), *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*, Oxford University Press, Inc., New York, NY, USA.

Parker, J. (2007), A flexible, large-scale, distributed agent based epidemic model, WSC 2007: Winter Simulation Conference, IEEE, pp. 1543–1547.

Perumalla, K. S. (2009), Switching to high gear: Opportunities for grand-scale real-time parallel simulations, DS-RT 2009: IEEE International Symposium on Distributed Simulation and Real-Time Applications, pp. 3–10.

Poslad, S. (2007), Specifying Protocols for Multi-agent Systems Interaction, *ACM Trans. Auton. Adapt. Syst.* 2(4). http://doi.acm.org/10.1145/1293731.1293735

Pratt, M. J. (1995), Virtual prototypes and product models in mechanical engineering, *Virtual Prototyping–Virtual environments and the product design process* 10, 113–128.

Reynolds, C. W. (1987), Flocks, Herds, and Schools: A Distributed Behavioral Model, Proc. ACM SIGGRAPH Computer Graphics, Vol. 21, ACM, Anaheim, California, pp. 25–34.

Rutle, A. (2010), Diagram Predicate Framework: A Formal Approach to MDE, PhD thesis, Department of Informatics, University of Bergen, Norway.

Segal, O., Margala, M., Chalamalasetti, S. R. and Wright, M. (2014), High Level Programming Framework for FPGAs in the Data Center, FPL 2014: 24th International Conference on Field Programmable Logic and Applications, IEEE, pp. 1–4.

Steinberg, D., Budinsky, F., Paternostro, M. and Merks, E. (2008), *EMF: Eclipse Modeling Framework 2.0 (2^{nd} Edition)*, Addison-Wesley Professional.

TOP500.org (2012), November 2012 List. www.top500.org

Wang, G. G. (2002), Definition and review of virtual prototyping, *Journal of Computing and Information Science in engineering* 2(3), 232–236.

AUTHOR BIOGRAPHIES

ADRIAN RUTLE[1] holds PhD and MSc degrees in Computer Science from the University of Bergen (UiB), Norway. Rutle is associate professor at the Department of Computing, Physics and Mathematics at the Bergen University College (HiB), Norway. Rutle's main interest is applying theoretical results from the field of Model-driven Software Engineering to practical domains. He also conducts research in the fields of modelling and simulation of various physical environments using Multi-Agent Systems. His main expertise is the development of formal modelling frameworks as well as domain-specific modelling languages, based on multi-level metamodelling and formal, diagrammatic constraint definitions. He has also experience in using graph-based logic for reasoning about static and dynamic properties of models; as well as in using model transformations for the definition of semantics of modelling languages.

HAO WANG [2] holds a PhD and a BEng, both in Computer Science. He is currently an associate professor and the head of big data lab with the Faculty of Engineering and Natural Sciences at the Aalesund University College. He is interested and has worked in big data, distributed systems, safety-critical systems, real-time systems, and security. He has extensive collaborations with industrial companies and academic institutes.

ROBIN T. BYE[3] graduated from the University of New South Wales, Sydney with a BE (Hons 1), MEngSc, and a PhD, all in electrical engineering. Dr. Bye began working at the AAUC as a researcher in 2008 and has since 2010 been an associate professor in automation engineering. His research interests belong to the fields of artificial intelligence, cybernetics, and neuroengineering.

OTTAR L. OSEN is MSc in Cybernetics from the Norwegian Institute of Technology in 1991. He is the head of R&D at ICD Software AS and an assistant professor at AAUC.

[1]www.rutle.no
[2]www.haowang.no
[3]www.robinbye.com