Building Fast Multi-Agent Systems using Hardware Design Languages for High-Throughput Systems

Jannis Stoppe[1,2], Christina Plump[1], Sebastian Huhn[1,2], and Rolf Drechsler[1,2]

[1] Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2] Cyber-Physical Systems, DFKI GmbH, 28359 Bremen

Abstract. While being from various domains, Multi Agent Systems and Hardware Design Languages rely on the same core concepts: parallelism, separation of concerns as well as communication between independent entities. This paper introduces the idea of utilizing the benefits of HDLs, with their focus on fast simulation and correct timing, to implement multi agent designs. Consequently, the massive advantages can be taken into account, which have been gained by decades of research in the field of electronic design, concerning the analysis, the evaluation as well as the optimization.

1 Introduction

Since several years, *Multi Agent Systems* (MAS) have been frequently used to model logistics systems. They offer flexibility, a clear separation of concerns for the actors of a system, allow for a wide range of simulation parameters and system setups to be tested and quickly provide both, high-level and in-depth results for a given design. *Hardware Design Languages* (HDLs) have been in use for decades to design and test hardware systems – the development of state-of-the-art computers, embedded designs and integrated circuits of all kinds relies heavily on the workflow offered by HDL environments.

However, despite their conceptual differences, they hold some intriguing similarities, which suggest that they could be used to design systems. In fact, this application is the counterpart they were originally designed for. This paper introduces the idea of implementing MAS using established HDLs, which allow MAS designers to utilize decades worth of optimization from the electronic design industry while keeping the established multi agent paradigms.

2 Multi-Agent Systems

Since the 1980's, MAS have been part of various research activities and, thus, have gained more and more interest over the years. Mainly, this is due to the fact that they are interpreted as a software paradigm, which can easily deal with massive open distributed systems appropriately [10]. They are used in different contexts like logistic: Whenever it is necessary to model systems with several actors that, while acting autonomously, communicate and perceive their surroundings to determine their courses of action. Depending on the focus or the perspective of research, there have been a lot of different definitions for MAS. If one aims at simulating MAS to prove their validity, they all have the following things in common (cf. [9] or [3]):

- *Agents* act autonomously due to their tendencies (objectives, goals), perception of their environment (communication with other agents, passive recognition of changes in the environment) while this environment is only partially represented and their given resources.
- *Environments* constitute a dynamic system (without autonomous behaviour), which the agents are situated in, perceive their inputs from and output their actions to.
- *Coupling* of agents and environment to, finally, model the desired MAS. This mainly addresses the problem of relating agent's perceptions and actions with environmental changes in a time-accurate manner.

As MAS are often very complex, it can be difficult to verify their properties and correct behaviour formally [5]. Therefore, simulations of those MAS are strictly required to ensure that they work as intended. Since the beginning of the 1990's, various frameworks have been developed to simulate MAS in different domains. They aim usually at either the study of complexity or distributed intelligence or aim at Software MAS[9].

Nevertheless, as agents (and also changes in the environment) naturally happen simultaneously, a simulation of an MAS requires parallelism. Some work has been done in that regard with tools like Gensim [1], JADE [2] or Spades [7] as has been surveyed in [10] but they focus on software level.

Using MAS gives engineers the ability to (somewhat) easily model, simulate and evaluate arbitrary logistic processes. These applications range from small factory models over more complex, conceptually and novel manufacturing chains, e.g., a high throughput material developing approach [4] , to vast supply networks.

3 Hardware Design Languages

On the other hand, HDLs serve a very different and specific purpose: the design of electronic systems. In order to deal with the vast complexity of current hardware designs, these languages provide a layer of abstraction. This layer allows designers to implement modular systems at the so-called register-transfer-level.

Hardware modules can be defined as simple machines, which hold an internal state (i.e. some kind of storage). These machines processes an input usually referring to signals, i.e., connections to other modules, and produce some kind of output usually referring to various connections to other modules. Both, the current state and the output depend on the input as well as the previous state.

The advantage of this approach is, however, not just that dedicated synthesis tools realize the translation from the HDL to descriptions, which can later be manufactured, but also the simulation forming a core feature With hardware manufacturing processes being both, expensive and time-consuming, simulating a design in software is the state-of-the-art in order to quickly test certain changes or evaluate ideas. However, with current computers still being typically sequential machines, the simulation framework has to take care of (simulated) parallelism and the communication between modules.

4 Merging the Concepts

The interesting point is the similarity of these core features of the framework. While the original application is quite different, the core issues of simulating

either hardware or multi agent systems is very similar: independent entities, which operate concurrently, must be simulated on a single sequential machine. Even the required parts for the actual simulation are quite similar on a second glance: retrieving some kind of (sensory) input, sometimes storing something depending on this input and finally acting (setting the output) based on both, inputs and the internal state. Thus, it should be possible to map one of these concepts to the other.

Hardware development has always been a hot research topic, with countless hours having been poured into the various tools and workflows. Additionally, HDLs have always had performance as a core issue. Current hardware systems consist of billions of gates, which have to be simulated in order to generate test sets being applied in both pre- as well as post-silicon tests and, at least important, realizes various verification tasks. Thus, theoretically, HDLs should be a promising foundation, if it comes to the execution of fast simulations.

This remainder of this paper introduces the concept of *MAS-via-HDLs* (MvH), implementing concepts of MAS and simulations using the established HDL frameworks.

With concepts such as schedulers, events, event queues, communication etc. all being readily available, HDLs contain everything, which is required for an fully operational framework. With modularization, processes, modules etc. , HDLs provide structure similar to what MAS libraries implement to build the agents themselves.

One major difference between HDLs and MAS is that the former explicitly prohibit altering any structural features after the simulation has started. This not only means that, assuming agents are considered to be mapped to module structures, i.e., no further agents can be instantiated after the simulation has commenced. but this extends to the communication structures: While wires (or signals) are considered to be part of the (fixed) hardware in HDLs, i.e., they can not to be altered on a finished design, MAS often rely on communication being initiated and then later finished and cut between two or even more agents. However, this shortcoming can be addresses by introducing the required maximal amount of agents before the simulation starts. This is like a pool containing enough members, which are activated and deactivated as required and all these member are connected according to the given profiles – if needed using, e.g., bus-like communication structures establishing the message exchange between parties.

Apart from this conceptual difference, however, the concepts are indeed quite similar allowing HDLs to be orchestrated by MAS frameworks effectively.

5 Case Study

As a case study, the Party example from the JADE [2] framework was adapted to SystemC [6]. SystemC is a high-level system design framework, which is based on C++, which has become an industry standard [8] for electronic system design.

The Party example illustrates how agents communicate and how this mechanism is used to distribute information. Two agent types are implemented as follows:

- The *Guests* are being introduced to each other. Out of all guests, one starts to share a rumor with the party. As soon as two guests are introduced of which, at least one, knows the rumor, the rumor is spread among them.
- The *Host* steadily introduces guest agents to each other and tells initially a guest the rumor, which has been randomly selected by the host.

The scenario is done as soon as all guests are aware of the rumor.

The Java classes for the agents inherit the existing JADE classes:

```
1  public class HostAgent extends Agent { ... }
2  public class GuestAgent extends Agent { ... }
```

The implementations for the SystemC adaption on the other hand implements agents as hardware modules, relying on the according C++ macros:

```
1  SC_MODULE(HostAgent) { ... }
2  SC_MODULE(GuestAgent) { ... }
```

More precisely, JADE comes with the so-called DF, the Directory Facilitator, which operates as a hub between agents (and is often referred to as the system's "yellow pages"). While SystemC does not provide a functionality that works exactly like JADE's DF, it *does* have a strong focus on communication between modules, often in the form of buses or more complex structures. The adapted implementation gives the host the role of the hub, requiring guests to register at the host. As communication connections have to be modeled explicitly, the host requires as many communication sockets as there are potential guests:

```
1  tlm_utils::simple_initiator_socket<HostAgent> socket[COUNT];
```

TLM is short for Transaction Level Modeling, a framework integrated into SystemC, which is used to pass messages between modules in a generic way while providing better simulation performance than pin-accurate signals. Most importantly, TLM messages contain arbitrary "payload" fields, allowing users to easily attach any given data to the message.

Similarly, when the guests are instantiated, they have to be connected to the given socket in order to be able to exchange messages with the host:

```
1  initiator = new HostAgent("HostAgent");
2  guests    = new GuestAgent*[COUNT];
3  for(int i = 0; i < COUNT; i++) {
4    guests[i] = new GuestAgent("GuestAgent");
5    initiator->socket[i].bind( guests[i]->socket );}
```

Beside this loop, which connects the host's ith socket to the ith guest's socket, the rest of the modeling remains conceptually similar to the JADE blueprint. The most obvious difference is due to the fact that hardware simulation frameworks (such as SystemC) do not allow modules to be removed from the design after the simulation has started. This means that the modules have to keep track by themselves whether they are currently active or not. In this example, this is not an issue as agents are only removed when the final state (of all agents knowing the rumor) has been reached, at which point the simulation is finished anyway. However, while this may be an issue concerning how the simulation is planned and executed (due to dynamic instanciation being restricted), this should not pose an issue as *a)* user-defined types may still be instanciated at will and *b)* sensible communication protocols should be able to disregard inactive parts. The only issue that might arise is memory, as all participating parts must be created before the simulation starts and held in the RAM afterwards.

When these designs are executed (with the JADE implementation being modified to no longer rely on user inputs but instead running by itself), the difference between the given approaches is striking: JADE required 6.895 seconds to spread the rumor across 1000 guests, the SystemC adaption merely 0.054 seconds (executed on an Intel i5-3320M with 12 GB RAM). While this example of course only provides a single measurement, scalability should not be an issue as long as the system is run on a single host system. As HDLs are built very

performance-centred while at the same time focusing on reproducibility, they do not provide measures to route communications across separate machines – thus, as soon as this would be required for larger simulations, scalability might become a problem.

Keep in mind that this comes at the cost of increased implementation effort: There is no graphical user-interface to investigate the communication, although SystemC does offer sophisticated logging features. Furthermore, the C++ foundation does not offer garbage collection, thus, the designers have to free allocated memory by themselves, and the communication channels have to be modeled explicitly. However, the proposed MvH technique leads to a significant performance boost compared to conventional MAS development workflows – in particular, a computation time less than ten seconds are perfectly fine for most the investigated use cases. This means that the application of the proposed MvH technique has to be considered, especially, when building systems with lots of agents or a vast amount of communication, which -in the first place- allows to design highly optimized MAS.

These optimized MAS are strictly required, if the system relies on a high throughput of elements (such as manufacturing chains of very small, independent, interacting items) and, therefore, should be deliberately analysed

Future work should look into the details of whether or not certain criteria have to be fulfilled for the difference to remain this vast, if the performance-gaining points of SystemC are projectable to other HDLs as well. Furthermore, it should be investigated whether or not these driving forces can be transferred into a dedicated MAS framework that is built on top of these faster frameworks.

6 Acknowledgment

Financial support of subprojects P02 "Heuristic, Statistical and Analytical Experimental Design" and P01 "Predictive Function" of the Collaborative Research Center SFB1232 funded by the German Research Foundation (DFG), the Reinhart Koselleck project DR 287/23-1 (DFG), University of Bremen's graduate school SyDe, funded by the German Excellence Initiative and BMBF grant SELFIE, no. 01IW16001 are gratefully acknowledged.

References

1. John Anderson. A generic distributed simulation system for intelligent agent design and evaluation. In *in Proceedings of the Tenth Conference on AI, Simulation and Planning, 2000, Society for Computer Simulation International*, pages 36–44.
2. Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. *Jade — A Java Agent Development Framework*, pages 125–147. Springer US, Boston, MA, 2005.
3. Roberto A. Flores-Mendez. Towards a standardization of multi-agent system framework. *Crossroads*, 5(4):18–24, June 1999.
4. Sebastian Huhn, Heike Sonnenberg, Stephan Eggersglüß, Brigitte Clausen, and Rolf Drechsler. Revealing properties of structural materials by combining regression-based algorithms and nano indentation measurements. In *IEEE Symposium Series on Computational Intelligence*, 2017.
5. Muaz A. Niazi, Amir Hussain, and Mario Kolberg. Verification & validation of agent based simulations using the VOMAS (virtual overlay multi-agent system) approach. *CoRR*, abs/1708.02361, 2017.
6. O.S.C. Initiative. IEEE Standard SystemC Language Reference Manual. *IEEE Computer Society*, 2006.
7. Patrick Riley and George Riley. SPADES — a distributed agent simulation environment with software-in-the-loop execution. In S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, editors, *Winter Simulation Conference Proceedings*, volume 1, pages 817–825, 2003.
8. Carsten Schulz-Key, Markus Winterholer, Thomas Schweizer, Tommy Kuhn, and Wolfgang Rosentiel. Object-oriented modeling and synthesis of SystemC specifications. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 238–243. IEEE, 2004.
9. Adelinde M. Uhrmacher and Danny Weyns. *Multi-Agent Systems: Simulation and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.
10. Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.