An Agent-Based Coupling Platform for Complex Automata

Jan Hegewald[1], Manfred Krafczyk[1], Jonas Tölke[1],
Alfons Hoekstra[2], and Bastien Chopard[3]

[1] Technical University Braunschweig, Germany
{hegewald,kraft,toelke}@irmb.tu-bs.de
[2] University of Amsterdam, The Netherlands
alfons@science.uva.nl
[3] University of Geneva, Switzerland
Bastien.Chopard@cui.unige.ch

Abstract. The ability to couple distinct computational models of science and engineering systems is still a recurring challenge when developing multiphysics applications.

The applied coupling technique is often dictated by various constraints (such as hard- and software requirements for the submodels to be coupled). This may lead to different coupling strategies/implementations in case a submodel has to be replaced in an existing coupled setup.

Additional efforts are required when it comes to multiscale coupling. At least one of the submodels has to be modified to provide a matching interface on a specific spatial and temporal scale.

In the present paper we describe a generic coupling mechanism/framework to reduce these common problems and to facilitate the development of multiscale simulations consisting of a multitude of submodels.

The resulting implementation allows the coupling of legacy as well as dedicated codes with only minor adjustments. As the system is being build upon the JADE library, our platform fully supports computations on distributed heterogeneous hardware.

We discuss the platform's capabilities by demonstrating the coupling of several cellular-automata kernels to model a coupled transport problem.

Keywords: generic coupling, heterogeneous, JADE, multi-scale, mutual interactions.

1 Complex Automata

In order to be able to develop non-trivial computational models it is usually an essential prerequisite to identify the major elements of the target simulation system. In doing so, one can construct complex multi-science models and the same elements can be used to start with the software design.

Each contributing sub-model may use a different modelling technique, such as cellular automata (CA), a numerical kernel for PDEs or multi-agent based systems (MABS) (e. g. used for biomedical simulations [1]). In addition the models will most likely use varying abstractions of a shared item, e. g. different spatial or temporal scales of a shared domain.

In this work the resulting combined model will be termed Complex Automata (CxA) [2], a paradigm emerging from the EU funded project COAST [3].

2 Coupling Environment

The software design for a CxA should be able to inherit the concepts of the involved sub-models very closely to allow for better maintenance and reusability. This will result in separate software components for every sub-model, which have to be coupled to build the complete CxA, instead of a monolithic code for each individual CxA.

Since there is no limit to the number of sub-models (kernels) of a CxA, there should be some kind of middleware or library to aid the development and maintenance of complex CxA. This middleware should also embed the execution model of the CxA approach, so the implementation of each kernel can focus on the specific sub-model itself. This way we can easily exchange individual sub-model implementations and the reuse of existing sub-model code is greatly simplified.

2.1 Core Implementation

Though our CxA middleware implementation, the Distributed Space Time Coupling Library (DSCL), is a work in progress, we already have non-trivial CxA simulations up and running. The major core modules of the DSCL are implemented in the Java programming language. As of now, the library supports the coupling of Fortran, C and C++ kernels, next to pure Java codes. Multiple languages may be freely intermixed within a single CxA. A setup is configured via ASCII files which declare the CxA graph and the scale separation map (SSM). Each vertex in the graph represents an involved sub-simulation (e. g. CA and MABS) and the edges designate a sub-model coupling. The SSM describes the different spatial and temporal scales of the individual sub-models [4].

Because each sub-simulation has its own controlling task (i. e. thread), a CxA may run fully distributed in a distributes environment. Since the idea is to keep each sub-simulation implementation unaware of the whole CxA layout, the system can make use of distributed-memory environments. This is even possible on heterogeneous hardware setups.

The thread control is done via the MABS middleware JADE [5], where the threads belong to a software agent [6]. The JADE provides peer to peer message passing functionality which is currently used for the communications along the graph edges. A bandwidth comparison between JADE and MPICH MPI [7,8] is shown in Fig. 1. These bandwidth measurements have been performed on a Linux cluster, whose nodes (AMD 64, Debian) are directly connected via various network interfaces. For these tests the 100 Mbit and 1000 Mbit channels have been used where the nodes are connected via a Gbit capable network switch. See also [9] for further scalability and performance tests regarding JADE.

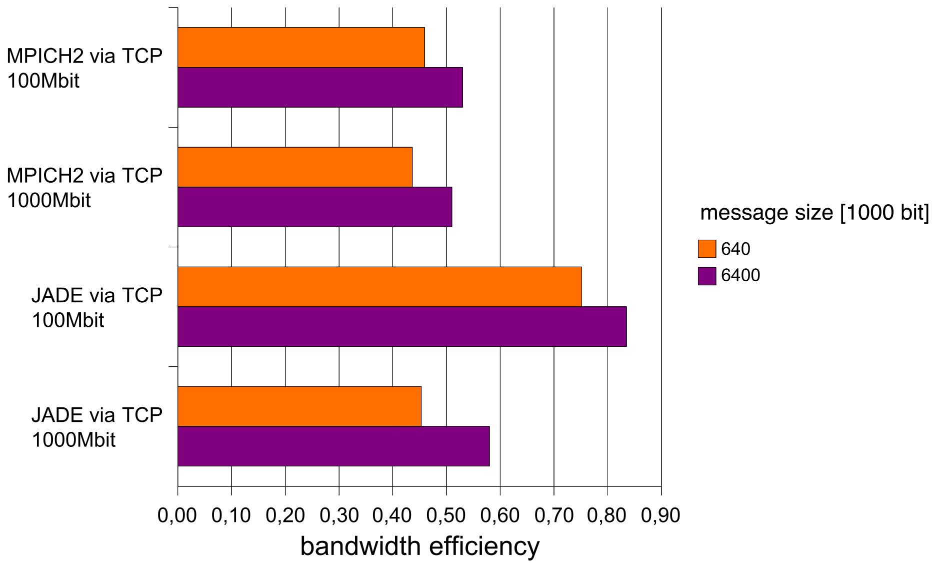


Fig. 1. Comparison of JADE and MPICH MPI bandwidth using TCP

MABS are well suited for building large software systems. Complex software systems, like CxA, consist of several more or less tightly coupled subsystems that are organized in a hierarchical fashion i. e. the subsystems can contain other subsystems. It is argued that MABS is well suited to reduce the complexity of building such software systems [10]. The topic is influenced by ongoing research in areas such as parallel and distributed discrete event simulation [11] and object oriented simulation [12].

2.2 Reusability and Coupling Interface

Since the connected kernels of a CxA have no information of the whole graph, we can keep the kernel implementations separate from a specific CxA application. This allows us to reuse the kernels in different CxA setups, even without the need for recompilation. To achieve this, there has to be some coupling knowledge where data transfer takes place at the coupling interfaces (edges). In the DSCL these smart edges are called “conduits” which act as a kind of network (or local) pipe with a data sink and a data source. In its simplest form such a conduit is an unidirectional pipe and can pass data within a single machine or transfer it across a network. For a generic coupling approach we have to map data from the sending side to the required input format of the receiving kernel. This involves scale mapping and also data conversion, if necessary. Using this technique it is possible to connect a kernel k_1 to multiple kernels at runtime $(k_2, k_3 \ldots)$, which may require different I/O data formats e. g. due to different scales required by the sub-models.

Another benefit of the conduits is the flexible substitution of kernels which provide functionality for the same vertex in the CxA graph. If, for example, a kernel $k_{0,\alpha}$ is used to represent vertex v_0, the kernel may be exchanged with another implementation $k_{0,\beta}$. This is even possible at runtime.

In order to reduce the effort to use legacy code as a participating kernel in a CxA setup, each kernel remains full control over its main time-loop. All communication calls to the conduits are issued from the responsible kernel. Except for booting, the kernel implementation does not need to provide a public interface.

We allow the CxA to influence the local time-loop execution of a kernel by means of blocking calls from kernel to conduit. This can be compared to blocking MPI send/receive calls which often can simply be replaced by the DSCL equivalents (highlighted lines in Fig. 4). The time-loop synchronization is implicitly synchronized this way.

To make a kernel available for the DSCL, it has to be glued to a Java class which is (derives from) a JADE agent (Fig. 2).

```
public class PseudoKernelWrapper extends CAController {

    protected void addPortals() {
        // add data inlets/outlets
        addExit(reader = new ConduitExit(...));
        addEntrance(new ConduitEntrance(...));
    }

    protected void execute() {
        // launch kernel
        bootKernel();
        // done
    }

    private void bootKernel() {
        // kernel main-loop
        for(time = 0; !stop(); time += INTERNAL_DT) {
            // read from our conduit(s) at designated frequency
            if(time % DT_READ == 0)
                data = read();

            // process data

            // dump to our conduit(s) at designated frequency
            if(time % DT_WRITE == 0)
                write(data);
        }
    }
}
```

Fig. 2. Pseudo code of a kernel agent

3 Transport Problem Example

During the initial development phase, a testbed with two distinct simulation kernels was developed from an originally single-kernel flow solver. This existing code is a preliminary kernel to simulate river channel erosion and sediment transport within a fluid flow. The solver uses a modified Lattice-Boltzmann automaton [13] to simulate incompressible flow where terms to simulate buoyancy where added. Currently the automaton works as a single monolithic solver on uniform grids. The research done on this sediment-erosion-model is part of another research project [14] and is written in the C++ programming language.

Main elements of the simulation are:

- advection of the sediment in the fluid
- diffusion of the sediment in the fluid
- sediment concentration in the fluid
- morphological changes of the river channel (erosion, deposition)

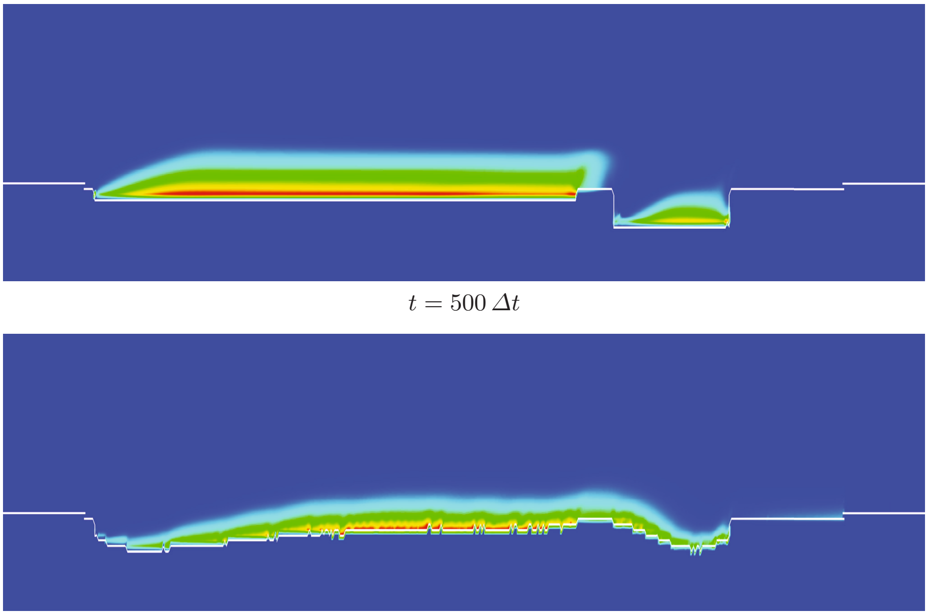
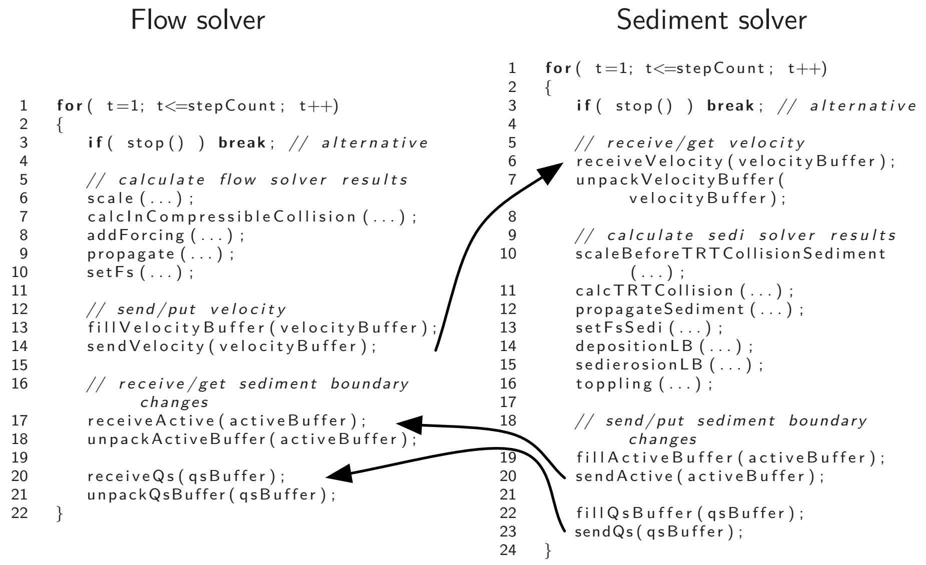


Fig. 3. Sediment concentration and boundary changes due to erosion

In Fig. 3 we show two different snapshots of such a simulation where the current sediment concentration is displayed. Some parts of the bedrock are subject to erosion (removal) of sediment, where in other areas deposition (adding of sediment) takes place.

To couple two distinct automata with the first prototype of the COAST coupling mechanism, the existing sediment erosion kernel was taken apart into two mutually interacting solvers: one Lattice-Boltzmann automaton to simulate the fluid flow and a second automaton to simulate the sediment advection/diffusion/erosion processes. Both kernels rely on the same uniform grid, so there is no scale separation here [4].

The sediment solver depends on the current fluid velocity at each discrete location, whereas the flow solver depends on the changing sediment boundary. These two kernels where now coupled with the first implementation of DSCL. Calculation results and integrity of the CxA implementation could successfully be validated against the original monolithic solver.



```
for( t=1; t<=stepCount; t++)
{
    if( stop() ) break; // alternative

    // calculate flow solver results
    scale(...);
    calcInCompressibleCollision(...);
    addForcing(...);
    propagate(...);
    setFs(...);

    // send/put velocity
    fillVelocityBuffer(velocityBuffer);
    sendVelocity(velocityBuffer);

    // receive/get sediment boundary
        changes
    receiveActive(activeBuffer);
    unpackActiveBuffer(activeBuffer);

    receiveQs(qsBuffer);
    unpackQsBuffer(qsBuffer);
}
```

```
for( t=1; t<=stepCount; t++)
{
    if( stop() ) break; // alternative

    // receive/get velocity
    receiveVelocity(velocityBuffer);
    unpackVelocityBuffer(
          velocityBuffer);

    // calculate sedi solver results
    scaleBeforeTRTCollisionSediment
          (...);
    calcTRTCollision(...);
    propagateSediment(...);
    setFsSedi(...);
    depositionLB(...);
    sedierosionLB(...);
    toppling(...);

    // send/put sediment boundary
          changes
    fillActiveBuffer(activeBuffer);
    sendActive(activeBuffer);

    fillQsBuffer(qsBuffer);
    sendQs(qsBuffer);
}
```

Fig. 4. Pseudo code of coupled sedi and flow kernels

4 Conclusion and Outlook

The Complex Automata Simulation Technique describes a method to couple multi-science models in order to simulate multi-scale problems. To facilitate the design and implementation of a coupled simulation software, we developed the Distributed Space Time Coupling Library which implements the ideas developed in the COAST project (i. e. CxA graph and scale separation map) using an agent based approach. The library aims to provide a flexible and powerful way for model developers to integrate new kernels as well as legacy code.

The different software kernels of a CxA are strictly separated which guarantees their reusability for future CxA projects. This way CxA implementations may evolve with upcoming (re)implementations of kernels (vertices).

Within the COAST project we will focus on the complex simulation of coronary artery in-stent restenosis [15] as a demonstrator application to validate the CxA modeling concept and the coupling library. These first simulations will include multi-science models from physics, biology and biochemistry, acting on different spatial and temporal scales. The results of these simulations will be subject to future publication.

References

1. Walker, D.C., Southgate, J., Holcombe, M., Hose, D.R., Wood, S.M., Neil, M.S., Smallwood, R.H.: The epitheliome: Agent-based modelling of the social behaviour of cells. Biosystems 76(1-3), 89–100 (2004)
2. Hoekstra, A.G., Chopard, B., Lawford, P., Hose, R., Krafczyk, M., Bernsdorf, J.: Introducing complex automata for modelling multi-scale complex systems. In: Proceedings of European Complex Systems Conference, European Complex Systems Society, Oxford (2006)
3. Mission of coast – complex automata, `http://www.complex-automata.org/`
4. Hoekstra, A.G., Lorenz, E., Falcone, J.L., Chopard, B.: Towards a complex automata framework for multi-scale modeling: Formalism and the scale separation map. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 922–930. Springer, Heidelberg (2007)
5. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology. Wiley, Chichester (2007)
6. Davidsson, P.: Multi agent based simulation: Beyond social simulation. In: Moss, S., Davidsson, P. (eds.) MABS 2000. LNCS (LNAI), vol. 1979, pp. 97–107. Springer, Heidelberg (2001)
7. MPICH Home Page, `http://www-unix.mcs.anl.gov/mpi/mpich1`
8. Message Passing Interface Forum: Message passing interface (mpi) forum home page, `http://www.mpi-forum.org`
9. Scalability and Performance of JADE Message Transport System, AAMAS Workshop, Bologna (2002)
10. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of the ACM 44(4), 35–41 (2001)
11. Fujimoto, R.: Parallel and distributed simulation. In: Winter Simulation Conference, pp. 122–131 (1999)
12. Page, B.: Diskrete Simulation. Eine Einführung mit Modula-2. Springer, Berlin (1991)
13. Geller, S., Krafczyk, M., Tölke, J., Turek, S., Hron, J.: Benchmark computations based on lattice-boltzmann, finite element and finite volume methods for laminar flows 35, 888–897 (2006)
14. Stiebler, M., Tölke, J., Krafczyk, M.: An Advection-Diffusion Lattice Boltzmann Scheme for Hierarchical Grids (Computers and Mathematics with Applications) (in press)
15. Mudra, H., Regar, E., Klauss, V., Werner, F., Henneke, K.H., Sbarouni, E., Theisen, K.: Serial Follow-up After Optimized Ultrasound-Guided Deployment of Palmaz-Schatz Stents: In-Stent Neointimal Proliferation Without Significant Reference Segment Response. Circulation 95(2), 363–370 (1997)