A New Approach for Integrating Discrete Element Method into Component-oriented System Simulations

Christian Richter

Technische Universität Dresden, Chair of Construction Machinery; *christian.richter1@tu-dresden.de*

SNE 27(3), 2017, 125 - 130, DOI: 10.11128/sne.27.tn.10381
Received: June 1, 2017 (Selected ASIM GMMS/STS 2017 Postconf. Publ.), Accepted: July 20, 2017
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The working process of construction and conveying machines is characterized by the interaction with granular and bulk materials. In order to allow prospective analysis of machine behaviour under real operating conditions, coupled simulations are increasingly used. While modelling the equipment happens within the scope of component-oriented system modelling, reproducing particle-mechanical behaviour is done with discrete element method. The work presented here introduces a new integrated approach which allows a closed modelling and simulation of system models and discrete element method. The creation and calculation of coupled simulations is thus facilitated by a multiple.

Introduction

In recent years the usage and importance of simulations has significantly increased in the field of construction and conveying machines. Both sectors have in common that some kind of machine is handling with some kind of granular material (e.g. sand, gravel, pellets). Typical processes are the digging of a hole with an excavator or the transport of material with a conveyor belt. Knowing the forces arising from these processes is very important. On the one side they cause strains on single parts, on the other side the can affect the entire machine behaviour. Let's consider a excavator for example. The forces coming from the digging process are acting on the bucket and causing stress and wear, but they will also lead to an increasing hydraulic pressure in the cylinders.

This in turn will affect the pump and engine activity. In order to make prospective statements about machine behaviour under real operating conditions it's necessary to simulate the machine as well as the process in common.

For simulating the equipments behaviour, component-oriented system models are often used. These kind of models are describing the machine as a network of components and subsystems which can be part of different domains like hydraulics, electrical or control engineering. One option for simulating bulk materials is using the discrete element method (DEM). With help of this method it's possible to reproduce the motion of granular materials as well as the strains they cause on mechanical parts.

Bringing both simulation techniques together isn't as easy as it seems. In addition to various methods of calculation, the modelling paradigms are also completely different. Arranging coupled simulations is for that very difficult and time consuming. This paper presents a new integrative approach for closed modelling and simulation control.

1 Basics

1.1 Discrete element method

The discrete element method (DEM) was presented first in 1973 by Cundall and Strack [1]. It's a numerical method for simulating the behaviour and motion of large numbers of discrete, interacting objects. In most cases, as done here, these objects are referred as particles. Basis of the method is the calculation of forces acting between the particles or between a particle and an adjacent surface. The basic calculation cycle should be explained briefly below.

After insertion every particle has an initial position and velocity. The simulation loop starts with collision detection. In this phase all particle-particle and particle-wall contacts are determined. After that the forces and torques acting on every particle must be calculated. These forces result on the one hand from field forces like gravity and on the other hand from the particle deformation as a consequence of collision. For that different force-deformation laws are used. By summing up all single forces and torques, the translational and angular acceleration of each particle can be obtained. The last step is solving the equations of motion. For that the new positions and velocities are resolved by integrating translational and angular acceleration two times (Equations 1 and 2). Figure 1 shows this loop. It is repeated until a predetermined number of iterations is reached.

$$\overrightarrow{F_k} = m_k \cdot \ddot{\overrightarrow{x_k}} \rightarrow \overrightarrow{x_k} = \int\int \ddot{\overrightarrow{x_k}}\, dt \quad k = 1,2,\dots,N \qquad (1)$$

$$\overrightarrow{M_k} = J_k \cdot \ddot{\overrightarrow{\phi_k}} \rightarrow \overrightarrow{\phi_k} = \int\int \ddot{\overrightarrow{\phi_k}}\, dt \quad k = 1,2,\dots,N \qquad (2)$$

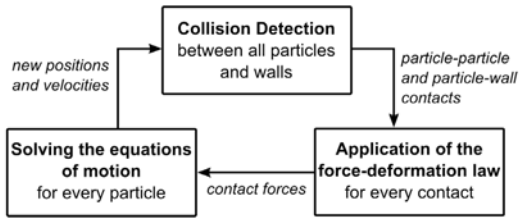

Figure 1: DEM Calculation Loop.

Most software for modelling and simulating discrete element models works command-oriented. The user tells the software what to do by typing single instructions into a command line tool or by loading an input script. Graphical user interfaces are very rare. After typing the commands they are processed sequentially. Typical representatives are open source solutions like LIGGGHTS® and Yade® or commercial tools like PFC3D®. This kind of modelling and user-software-interaction has historical reasons and is not very user friendly. The biggest problem is that the user has to have good knowledge about the commands and syntax. Furthermore its very complicated to adapt existing models onto new problems. Figure 2 shows an excerpt of a typical input script for LIGGGHTS®.

Most of these applications doesn't have any integrated post-processing tool. This is a disadvantage because visualization of results and particle data is very important. For that it's not seldom that users have to use third-party applications like ParaView®.

```
atom_style      granular
atom_modify     map array
boundary        mm mm mm
newton          off
echo            both
communicate     single vel yes

units si
neighbor        0.200000 bin
neigh_modify    delay 0
region          reg block -2.750 0.000 -2.200 2.200 -0.100 4.070 units box
create_box      2 reg

# Material Parameters
fix m0 all property/global youngsModulus peratomtype  2e9 2e9
fix m1 all property/global poissonsRatio peratomtype  0.3 0.3
fix m2 all property/global coefficientFriction peratomtypepair 2 0.5 0.4 0.4 0.3
fix m3 all property/global coefficientRestitution peratomtypepair 2 0.5 0.4 0.4 0.3
fix m4 all property/global coefficientRollingFriction peratomtypepair 2 0.5 0.4 0.4 0.3
fix m5 all property/global cohesionEnergyDensity peratomtypepair 2 0 5000 5000 10000

pair_style gran model hertz tangential history cohesion sjkr rolling_friction epsd2
pair_coeff  * *

variable dt equal 0.00001

fix integrator all nve/sphere
```

Figure 2: Input Script for LIGGGHTS®.

1.2 Network-based system simulations

Simulating complex machines - as already mentioned - is often done by using equation-based component-oriented system models. For the description of such models Modelica [2] has established as a kind of standard language in many areas. Meanwhile, there is a plurality of applications using and supporting Modelica out there (e.g. SimulationX® or Dymola®). Modelica allows the model description on several levels. One of the most common types of building up simulation models is the network-based or component-oriented modelling technique. Therefore configurable components and subsystems (network elements), which have clearly defined interfaces, are connected together in a network structure.

Apart from classic advantages of object-oriented approaches, such as encapsulation and re-usability through modularity and inheritance, this method also has other benefits. One is that building up and modifying models can be done very easily and rapidly. A further is the high degree of clarity, as the real and virtual structure often correspond with one another. Exemplary the top-level structure of an engine model is shown in Figure 3.

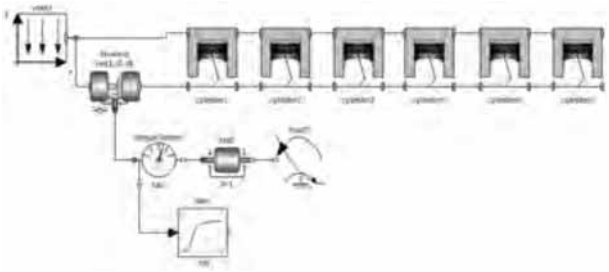

Figure 3: Network model of an engine with 6 cylinders.

1.3 Coupled simulations

As coupled simulation generally the calculation of coupled systems is meant. Coupled-systems consisting of two or more models from different domains, which have to exchange information with each other at simulation runtime. Furthermore the number of integrators and/or modelling tools must be greater than one. According to [3] there are different kinds of coupled simulations with different names as shown in Figure 4.

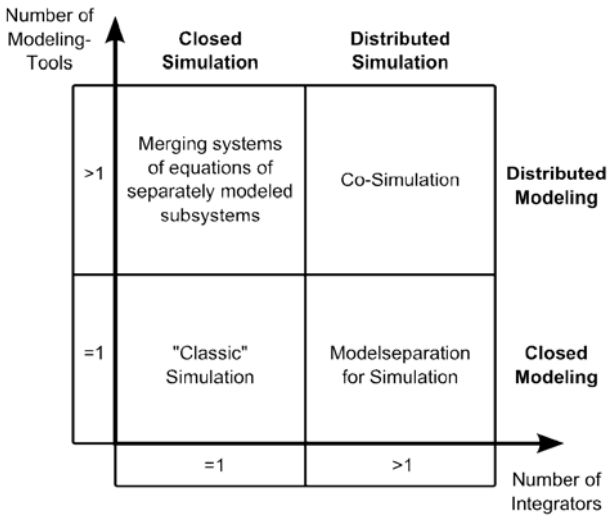

Figure 4: Classification of coupled simulations.

Assuming to this most recent approaches of coupling system simulations with discrete element method must be classified as co-simulation. One example for this is the approach presented in [4]. This approach starts with building up a machine model and exporting it as a functional mock-up unit (FMU). Afterwards a discrete element model is build up and connected to the FMU. The calculation is performed by two integrators with information exchange between both models at discrete time events. Building up and connecting the models causes a lot of work. This amount of work can be reduced by implementing a new functionality which allows closed modelling.

Generally, there are two possibilities for implementing such a functionality. Either it's attempted to integrate the DEM into component-oriented systems modelling or the other way around. From the points and facts mentioned above the first variant seems to be the better way for getting a high degree of usability.

2 Integrating DEM into Component-oriented System Models

2.1 Component library

For transforming the command-oriented modelling paradigm of classic DEM-tools into a componentorient-ed modelling technique several things have to be done. First of all, all relevant modelling functions must be identified. After this new abstracted components and parameters must be designed representing and implementing these functions. This components have to be very self-explaining and easy to understand for the user. Table 1 shows a selection of components and their corresponding function.

Component	Functions
SimulationBox	get total count and mass of all particles
ParticleSource	generate particles
ParticleSink	remove particles
ParticleFlowSensor	measure the number and mass of particles passing a surface
ParticleRegionSensor	measure the number and and mass of particles in a volumetric region
ParticleSet	loading existing data
RigidBody	interaction of a rigid body with the particles

Table 1: Components and corresponding functions.

As last step a translator must be implemented, which is capable to translate these components into commando sequences.

2.2 System structure

A component library as described before can be used in any Modelica-Tool. It allows the closed modelling of machine and process models. In order to perform a distributed simulation, models have to be subsequently separated. For a better understanding how this is done Figure 5 shows the basic structure of all simulation components. This structure is divided into two main areas - a front-end and a back-end.

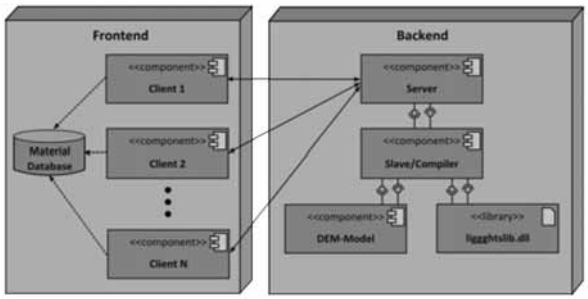

Figure 5: System structure.

The front-end consists essentially of the component library. Each of the components forms a client, which is capable to connect and communicate via TCP/IP to a server. Due to the fact that the presented solution was developed as part of a research project, the component library has some SimulationX®-specific features. One of them is the material selection via a database interface.

The server the components can connect to forms the root node of the back-end-structure. It receives the messages from the components and forwards them to a specific compiler. It's possible that components belonging to different simulation models can connect to the server. For that reason the server can handle more than one compiler at the same time. The compiler uses the information coming from the server to build up a copy of the current component structure. With help of this model commando-sequences are generated which can be executed by a specific DEM-tool. The software used here was LIGGGHTS® compiled as a shared library to make information exchange easier. Holding back the whole component structure at the back-end is necessary for the translation process because some commands can only be created with information coming from two ore more components.

The division into front-end and back-end communicating via TCP with each other seems at first sight a little bit complicated but brings some advantages. One of them is the fact that distributed computations are made possible. That means solving the system model can be done on a normal computer while discrete element simulation runs on a workstation or computer cluster. Especially for time consuming and expensive DEM simulations that's a plus.

2.3 Communication

At this point, communication between front-end and back-end is briefly explained. At the beginning of every simulation - that means during initialization phase of the system model - all components respectively clients connect to the server. Required connection settings, such as server address and port, are coming from the so called SimulationBox-component. The following code snippet shows this.

```
1 outer SimulationBox simbox "simBox";
2 Boolean isConnected(start=false);
3 ExternalObject client=TcpClient();
4 equation
5   when initial() then
6          isConnected=connectToServer(
7                 client,
8                 simbox.address,
9                 simbox.port);
10  end when;
```

The *SimulationBox* is a kind of superordinate parent or world object. Besides basic simulation settings it contains information about the spatial domain for DEM and gravity forces acting on all particles. All other components can access this information.

After a successful connection, the communication between front-end and back-end starts. It's divided into three phases:

1. During initial phase all front-end components send their parametric information to the server. These are, for example, initial positions and orientation, geometrical data or material values. All this information are collected at the back-end and used to generate executable commando sequences.
2. At simulation runtime information are exchanged in regular intervals. Here mainly new calculated positions of rigid bodies are transmitted from front- to back-end and forces as well as torques are returned.
3. At the end of simulation an information is sent to the server, which tells this that calculation is over. This will reset the back-end simulation.

Simplified code, containing the complete communication algorithm, is shown below.

```
1 parameter Real tc=0.0001;
2 Boolean commTrigger=sample(0, tc);
3 algorithm
4   if isConnected then
5     when initial() then
6          ...send initial data...
7     elsewhen commTrigger then
8          ...send and receive data...
9     elsewhen terminal() then
10         ...send final data...
11    end when;
12  end if;
```

3 Bucket Elevator

3.1 Analytical considerations

For functional testing and evaluating the new solution a bucket elevator was modelled and simulated. A bucket elevator is a mechanism for hauling flow able bulk materials (e.g. grain or sand) vertically. For that it is often used in hoppers.

Of decisive importance in analysing and constructing bucket elevators is material deflation. Basically there are two ways of emptying the bucket at the upper turning point. Either the material is thrown out by centrifugal forces or the material falls out in reaction to gravity forces. The second variant should be avoided because there's a high risk that particles fall back into the elevator housing.

To figure out which kind of material deflation is dominating point *P* is constructed like shown in Figure 6.

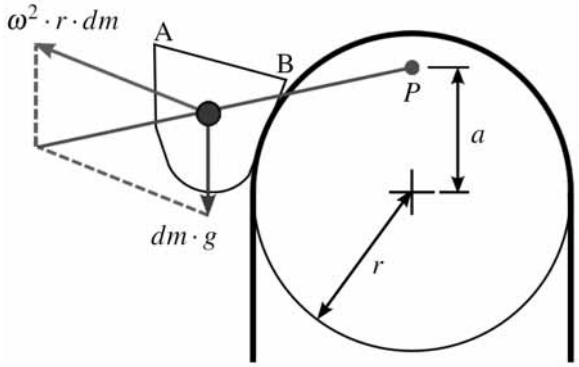

Figure 6: Construction of point *P* and distance *a*.

Thus particles are thrown out over edge *A* there must be $a < r$ [5]. For achieving this the minimal rotational velocity can be calculated like in Equation 3.

$$\frac{g}{\omega^2} = a; a < r \rightarrow \omega > \sqrt{\frac{g}{r}} \qquad (2)$$

What is completely ignored in this calculation is the internal friction of the material and the friction between material and bucket wall. So there's no possibility to get reliable prospective statements if the bucket is emptied the right way. Furthermore it's very difficult to investigate the bucket filling process or dynamic forces acting on the drive chain. That's where the discrete element method can help.

3.2 Modelling

First step to simulate the bucket elevator was to create a simple machine model, including buckets, housing and a simplified drive train. After this was done it was extended by adding DEM specific components. For continuous filling a particle source was inserted at the lower housing aperture. This particle source has an output rate of 35000 particles every second. At the upper outlet a particle sensor was added for measuring the mass and particle flow leaving the bucket elevator. Additionally another particle sensor was added for measuring the number of particles falling back into the housing. The complete model is shown in Figure 7.

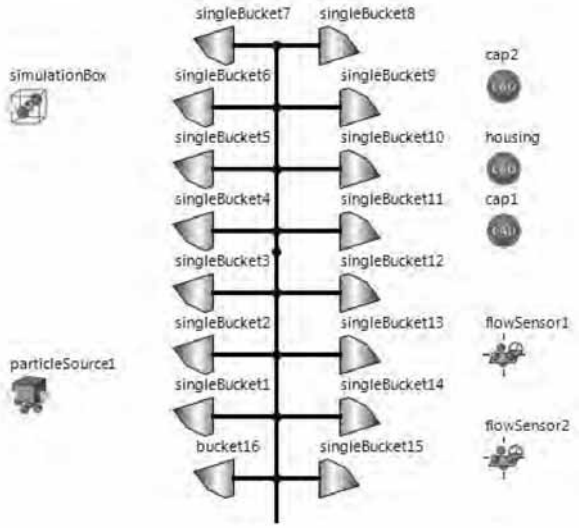

Figure 7: Structure view of the bucket elevator model in Simulation X®.

3.3 Results

For evaluation the bucket elevator was simulated for 4.0 seconds. After about 3.0 seconds, a steady state is achieved, at which the number of particles inserted is approximately equal to the number of particles leaving the bucket elevator. At this point of time there are about 65000 particles in the system. Figure 8 shows the 3d-simulation-view at this point of time.

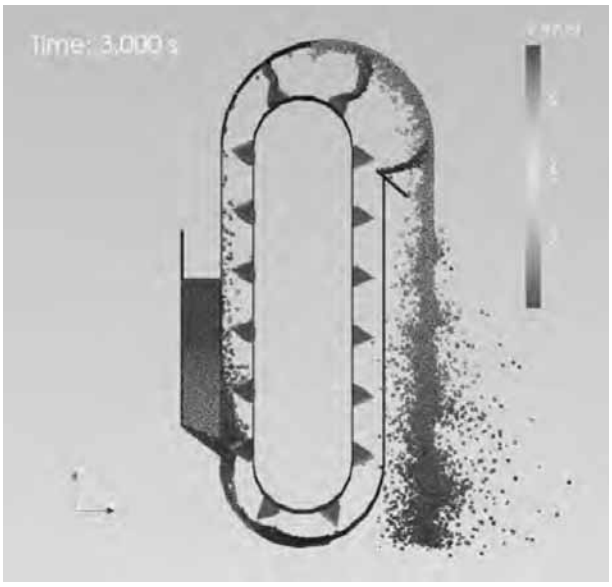

Figure 8: 3d-simulation-view of steady state.

It has been found that at a peripheral speed of 4 m/s nearly no particles fall back into the housing. Furthermore the forces acting on all buckets can be measured and used for optimizing the bucket design and drive chain. The temporal course of forces acting on a single bucket is exemplary shown in Figure 9.

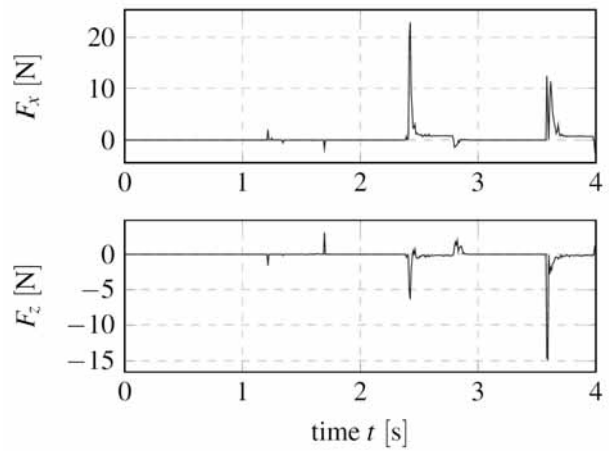

Figure 9: Forces on single bucket.

4 Conclusion

In this work, a new concept was presented allowing the closed modelling of machine models and discrete element systems in one simulation tool. For that the command-oriented modelling technique many DEM-applications work with was transferred into an object-oriented approach. Especially for new users which are not familiar with discrete element method but also for old experienced users creating models and coupled simulations is getting very easy.

References

[1] Cundall P, Strack ODL. A discrete numerical model for granular assemblies.

[2] Otter M. Modelica-A Unified Object-Oriented Language for Physical Systems Modeling-Language Specification, 2000.

[3] Geimer M, Krüger T, Linsel P. Co-Simulation, gekoppelte Simulation oder Simulatorkopplung? Ein Versuch der Begriffsvereinheitlichung, O+P Ölhydraulik und Pneumatik, pp. 572-576, 2006

[4] Kunze G, Katterfeld A, Richter C, Otto H, Schubert C. Plattform- und softwareunabhängige Simulation der Erdstoff-Maschine Interaktion In: 5. *Fachtagung Baumaschinentechnik*, Dresden, 2012

[5] W. Noack. Der Entleerungsvorgang bei Becherwerken, In: Agrartechnik, 1955