RECONFIGURABLE AND LAYOUT-AWARE STORAGESYSTEM FOR NETWORK-BASED SIMULATIONMODELS IN THE SIMULATOR D^{3}FACT

Hendrik Renken[(a)], Felix A. Eichert[(b)], Markus Monhof[(c)]

[(a)(b)(c)]Business Computing, especially CIM
Heinz Nixdorf Institute
University of Paderborn
Fürstenallee 11
33102 Paderborn, Germany

[(a)]funsheep@mail.upb.de, [(b)]eichert@mail.upb.de, [(c)]monhof@mail.upb.de

ABSTRACT
Current simulation software typically provides limited types of storage components. But for accurate simulation results, warehouse operations have to be modeled close to reality. The static design of storage components found in current simulation software complicates the process of modeling. Especially the paths for workers or forklifts through a warehouse have to be defined manually. Furthermore, when the user makes changes to the warehouse layout he also has to adjust the paths. This process is cumbersome, time consuming and error prone. In this article we describe a warehouse system that can be customized to fit a diverse variety of storage types. Our implementation drives an automated mechanism, re-calculating paths when changes to the layout are made. Besides that, different storage types can be combined into one warehouse and the whole system may be reconfigured at runtime. At last we show the integration of the presented concepts into our research platform d^{3}fact.

Keywords: material flow systems, d3fact, simulation, warehousing

1. INTRODUCTION

Simulating a system to understand its behavior for certain inputs is a well-established scientific method and is broadly used in research as well as in the industry. Today's enterprise simulation software uses network-based modeling to implement processes occurring in a company. Examples are production systems, warehousing systems or inter-company logistics. For these areas simulation software typically offers a wide range of building blocks where each contains a specific function or behavior. The combination of those blocks allows easy modeling even of complex scenarios. One part of a company that needs specific and detailed treatment is the storage area, because the question "Where to store this specific part?" can get quite complex and hard to answer. Some software packages therefore offer special products to answer just this one question (Incontrol Simulation Software 2012). Thus modeling a plausible, complex and customizable storage is an important task. Our approach of a generic warehouse component allows us to define arbitrary warehouse layouts. This includes the combination of different storage types, e.g. block or rack storages. Furthermore, the storages can be located throughout the whole model.

When modeling a warehouse the definition of the drivable paths of the operating units (OU), e.g. forklifts or workers, consumes a major part of the deployed time. Especially, these paths have to be adjusted by the modeler every time the layout of the warehouse changes. In this paper we describe a solution that automatically adapts to layout changes and to reconfigured warehouse components at runtime. In our approach we integrate our generic warehouse component into a system for automated motion path finding (Fischer et al. 2010). This ensures that the OUs can be used throughout the model to transport goods. Due to the proposed design it makes no difference whether the goods are transported between machines, storages or, machines and storages.

In this paper we present our concept of a generic warehouse implementation with integration to a generic transportation service utilizing OUs like forklifts or workers. Furthermore we briefly lay out the implementation into our simulator d^{3}fact.

2. CONCEPTS

In the following we will briefly describe the scenario used throughout this paper and the ideas behind our concept. To illustrate one possible application of the described components we use the multi-floor building example, which was introduced in Fischer et al. (2010).

The building is a factory laid out on three floors with production areas on the top floor, a storage area on the middle floor and a distribution center at its base (see Figure 1). Forklifts or workers either transport the production work pieces. Therefore ramps and stairs connect the different floors. The factory produces wheel caps. Plastic pellets are pressed into unfinished components. These components are then further

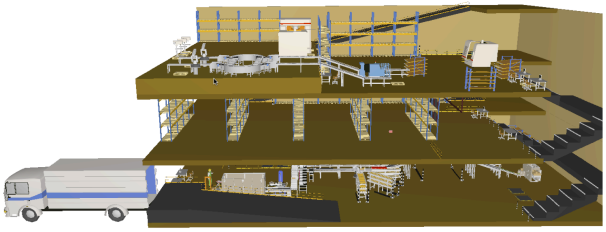

Figure 1: The Scenario: A Multi-Floor Factory Building.

processed into two different finished products. Customers can order the unfinished components (for individual processing) and of course the finished products. The model supports orders in large quantities as well as rush orders in small quantities. Products can be stored in large quantities on the middle floor. The lower floor is used as a packaging and shipping area. Forklifts transport large quantities of orders, while the workers can retrieve small quantities for rush orders or when parts are found to be missing during prepackaging. To model these dynamics the workers must have access to every storage area in the model, even the interim storages.

We can now use our warehouse component and its adaptive logistics to model the (interim) storage on the different floors. On each floor, storage racks are located. In the front of most of the machines block storages for interim storage can be found. Since we are not modeling the different storages as different warehouses in terms of our warehouse component, we can access every stored work piece from everywhere at any time. Using our warehouse component we can even easily add, move or remove storage racks at runtime without caring about the motion paths of the forklifts or workers. These paths are adapted to the new layout during the simulation. Furthermore, changes made to the set of storage components are also directly adapted. This enables rapid prototyping of new ideas and easy optimization of storage layouts, even throughout the whole factory.

The warehouse component is based on the concept of separation of the storage of actual goods from the logistic operations, which are executed on the storage. Figure 2 shows the basic structure that results from this separation. There are two main components that make up a warehouse: The *Storage* and the *Logistic Component*. The first one holds the goods in a specific storage structure, e.g. in a block storage or rack storage. The latter handles the input and output operations of the warehouse by executing appropriate storage/ retrieval operations. These operations can involve OUs moving the goods. The paths taken by the OUs to process the operation must be set manually in current simulation software. Because the manual approach takes a lot of time, we propose the usage of an automatic motion planning system.

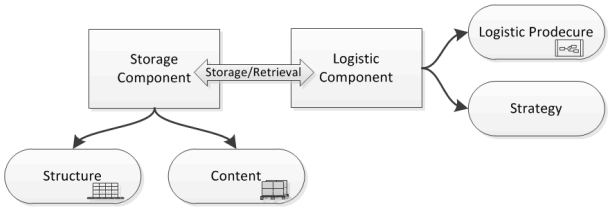

Figure 2: This Figure shows the Main Structure of the Warehouse Component in d^{3}fact.

3. RELATED WORK

While there are many applications of warehouse simulations the number of related works, addressing generic and easy-to-use modeling approaches for warehouse simulations, are very scarce.

Muller (1989) identifies components that need special attention when building an automated warehouse system. He describes three modeling approaches and notes that the modeling complexity differs for different objectives and uses. Unlike us he does not outline a generic component, which lowers the modeling time, but gives general advices what to consider when modeling a warehouse for different types of simulation results. He identifies, among others, the warehouse layout and control logic, which we call strategy, as important components to consider when modeling a warehouse.

Gunal et al. (1993) provides a simulation model for *Automated Storage and Retrieval Systems* (AS/RS) and gives conclusions about the general use of them. The authors found that most of their code they had written to simulate a particular warehouse could be reused for similar scenarios. Since they are focusing on AS/RS they do not provide a solution to change the layout of the warehouse. So their generalization is limited to characteristics of AS/RS like for example the number of aisles or the number of pick-and-drop stations.

Takakuwa (1996) is also focuses on AS/RS and utilizes a component (building block) approach. He presents predefined AS/RS and *Automated Guided Vehicles* (AGVs) components that can be combined to serve different applications. Due to the focus on AS/RS systems with AGVs the presented components for the warehouse are not as highly customizable as ours. The only layout, which is supported, is the aisle based rack layout of AS/RS, which is one building block for which some parameters could be set. So the approach of splitting a warehouse in different components is done in that way that AGVs and conveyors are part of warehouses. In our simulator there also exists components for AGVs and conveyors, but we do not limit their application to warehousing scenarios.

In contrast to that the problem of automated path finding in geometrical space is well researched. Motion planning in general is e.g. discussed in (Canny 1988) and (Brady 1982). de Berg et al. (2008) describes motion planning based on trapezoidal space partition, obstacle enlargement to support OUs with a size and how to support OUs with rotation. Latombe (1991) also addresses motion planning in general but furthermore,

he discusses several problems in depth, one has to deal with when creating a motion planning system. This includes e.g. how to manage obstacles, multiple moving objects and kinematic constraints. The system (Fischer et al. 2010) utilized in this paper uses an octree as a structure to partition the geometric space. It is based on a two dimensional approach by Chen et al. (1997). Also there are methods, which especially can be used in warehouses. Klaas et al. (2011) describe a knowledge-based approach on automated way finding for AGVs in dynamic warehouse environments (He also utilizes the method presented by Fischer et al. (2010)). A real-time motion planning method for highly dynamic environment with multiple participants is given in Vannoy and Xiao (2008).

4. THE WAREHOUSE COMPONENT

As stated before, the warehouse component is based on the concept of separating the storage of the actual goods from the logistic operations, which are executed on the storage (cp. Figure 2).

4.1. The Storage Component

The *Storage Component* defines the storage's structure. One part of the structure is the position of the goods. This way, the *Storage Component* can determine the position of each good in space and the distance between them. This is needed, to compute the time needed to access a good and to proper visualize the storage.

The other part describes through specific rules which positions and therefore which goods are accessible. In the following we explain the rule system on the example of a block storage. However, it is easy to model a specific, e.g. custom storage type, by just replacing the rules. The block storage depicted in Figure 3 has a total space for eight goods, where goods occupy four spaces. Now the aforementioned access rules for block storages define the space *B1* as inaccessible, because *B2* is occupied. One has to remove the good from *B2* to access *B1*. Also it is physically not possible to store a good in *A2* without an occupied *A1* space. Therefore, *A2* is also inaccessible.

To implement a new storage type, the structure and access rules have to be defined and stored goods of to be mapped to certain positions. This makes it very easy to customize a warehouse for a specific scenario.

We further separated the goods from the actual space they are stored in (cp. Figure 2). This design enables the interchangeability of the storage types. The storage type can be modified by simply replacing a structure by another one and then remapping the stored goods to the new structure. Because the set of stored goods is not affected, this can be done even at runtime. The user now can start a simulation and change the parameters of the warehouse - including the storage type - while the simulation continues. This is a big advantage for rapid prototyping. Even while a simulation is running, a user can play around with the storage type and test the performance of each getting a direct feedback.

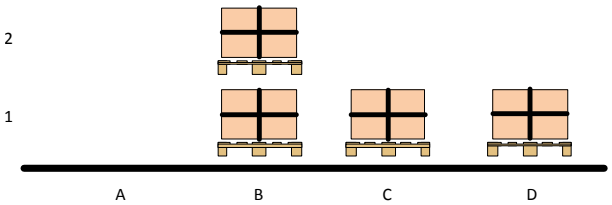

Figure 3: This is a Basic Representation of a Block Storage. There are Eight Spaces in this Storage. In Four of Them Goods are Stored.

4.2. The Logistics Component

Until now, we only discussed the static structure of our warehouse. In this section we will explain how the goods are transferred in and out of the *Storage Component*. The *Logistics Component* represents an interface to a storage and executes the input and output operations on the storage component.

The component itself is an abstract state machine and represents the generic input/output operation and the current state of such an operation. In our implementation an operation can be in the three states *Idle, Searching and Delivering*. A separate *Logistics Procedure* contains the logic how a particular good is stored or retrieved. The supervising component uses the procedure to compute the time needed for a particular operation. Such a procedure can e.g. implement the usage of OUs like forklifts or workers, represent a harbor crane, or an AS/RS. More abstract the procedure can even be directly related to the storage type and allow the implementation of physical processes like gravity for chute based storage types.

In our scenario of the multi-floor factory the racks on the second floor are embedded into the usual material flow using procedures that utilize forklifts to transport the goods. However, the interim storages in the top floor at the machines are embedded into the material flow using simple procedures with a static timed delay. Furthermore, to let workers from the lower floor also "see" and access these storages for rush orders, the interim storages have a second logistics component with a procedure utilizing the workers. In Section "MODEL-WIDE MOTION PLANNING INTEGRATION" we will cover the implementation of this system into a model wide transportation system in detail.

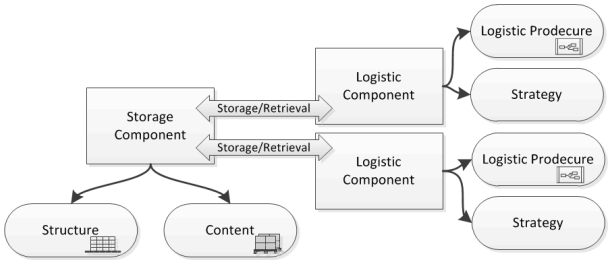

Figure 4: Several Logistics Access One Storage. Each Logistic has its own Production Definitions and Strategy.

This design of the logistics component has several advantages. Due to the decoupling of the logistics from the storage, it is easy to change the properties, like the storage capacity or type - especially at runtime. Furthermore, several logistic components can serve one storage component at the same time as you can see in Figure 4. In that figure you can see an example of two *Logistic Components* accessing one storage object. This enables, e.g. the modeling of several harbor cranes unloading a container ship or several forklifts serving one storage object at the same time.

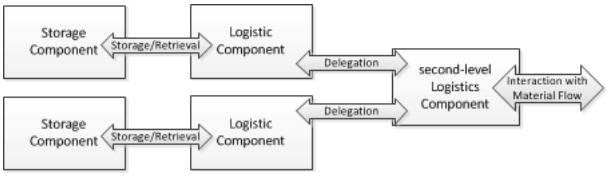

Figure 5: A *Second-Level Logistics Component* Delegates Operations to Two First-Level Components.

To allow the combination of all kinds of storage types into one "logical" warehouse we use *second-level Logistics Components*. These components aggregate several first-level components and the storage structures they are attached to as it can be seen in Figure 5. If an input/output-operation has to be executed the second-level component delegates the operation to the appropriate first-level component. This allows the integration of the first-level components into the material flow as usual and furthermore provides the same interface for the overall warehouse. Depending on the logistics component responsible for the current operation, the state of the second-level component is adapted. This ensures parallel operations for the different first-level components but also sequential operations on a particular logistics component. Depending on the state of the first-level components one could also choose different strategies, although we didn't explore this further. Besides the structural planning, the warehouse strategy usually is the focus of optimization in real life. Since changes made to the strategy usually do not result in expensive investments, it is a usual practice to optimize this part first. A strategy in the warehouse component determines a free storage space for a good to be stored in the component. Due to the volatility of the strategy, it is a separate component in the *Logistic Component*. This allows the strategy to be exchanged with customized implementations for static models and during runtime. It is even possible to have the strategy exchanged automatically, e.g. triggered by events generated from the simulation.

5. LAYOUT-ADAPTIVE WAREHOUSE LOGISTICS

Utilizing an automated motion planning system to access a storage object allows the implementation of layout-adaptive warehouse logistics. Up to this point, we assumed that the storage structure of a warehouse is static. The layout of a warehouse component was interchangeable, but the layouts themselves are static. This circumstance prevents the optimization of some important aspects of warehouses and their layouts, especially during a simulation run. A modeler might want to try different values for the gap between racks in a storage object to optimize space utilization and access by forklifts. Goetschalckx and Ratliff (1991) explain an approach to calculate optimal lane depths in block storage systems and compare their results to traditional concepts where all lanes have the same depths. With our system, a simulation-based comparison could be easily done with several different lane configurations.

The current, static implementation results in a try-and-error approach, which makes the optimization of a warehouse layout slow and cumbersome. One reason for this is, that by changing certain values of a warehouse, many other values have to be changed as well. For example, when using OUs to access a warehouse the *Logistic Procedures* need to know the exact positions of the goods to calculate the time needed to move. When the position of a storage object is changed or the layout of the scene changes, the motion paths to the storage spaces can get compromised. In current implementation the modeler has to manually adapt the paths, which is a complex and cumbersome task.

With the integration of an automated motion planning system, we enable the modeler to freely reposition storage structures. This enables him to easily test different settings. Furthermore, due to the proposed separation of the different components, changing the location of a storage object does not affect the stored goods or current operation. This means, the modeler can adjust the layout during a simulation run, getting direct feedback. How we integrated the motion planning into a model-wide transportation system is explained in the next section.

6. MODEL-WIDE MOTION PLANNING INTEGRATION

An important factor for simulation analysis is realistic motion paths through a defined factory layout. Typically, these paths have to be modeled manually, which is a cumbersome, time consuming and error prone task. Besides that, these paths have to be manually maintained when changing the layout.

To solve this problem Fischer et al. (2010) present a motion planning framework for automatic route calculation in three-dimensional environments. The framework is capable of automatic analysis of a given factory layout and of computation of motion paths for moving objects like fork lifts or workers (cp. Figure 6). At first the scene is divided into small cubes, called voxel. These voxel then are analyzed for a driving surface for a specific moving object, taking into account the size and the supported slope.

A problem arises from the usage of these moving objects with a warehouse like the one from our scenario (cp. Figure 1). Because the racks are located throughout the factory, transportation tasks vary with the location

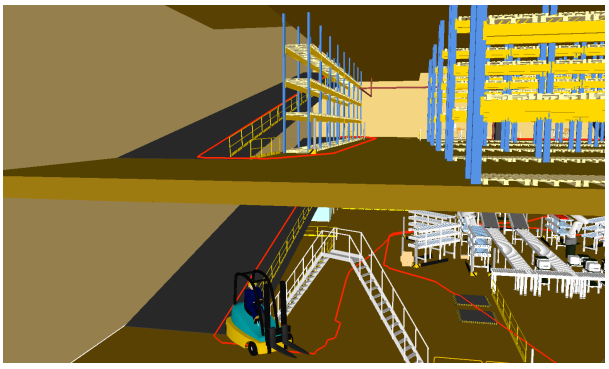

Figure 6: Motion Planning in a Production Plant Modeled in d^{3}fact.

of the object carrying out the task. Taking these variations into account are very important for the overall performance of such a simulation model.

To solve this problem we use a supervisor that schedules transportation tasks to minimize the overall processing time. The system is implemented in a very generic way so that we can use it to carry out transportations from a machine to particular storage space or even to drive housekeeping processes. The supervisor is a core component of the model-wide transportation system. It collects transportation jobs and schedules them to the operation units. A job always contains a start and an end position. Based on these positions a pick-up and a drop-off time can be estimated. These are updated whenever the scheduled plan changes due to new or updated jobs. As stated earlier, the *Logistics Procedures* are the interfaces between the warehouse logistics and the transportation system. While the logistics component itself is integrated into the material flow. Consider, that one of the machines on the top floor of our example finishes a product. Now a worker has to transport the product to a rack. The logistics component, connected to the machine, receives the information about the finished product. It asks the procedure, how long does it take to transport the item. The procedure itself issues a new job with the start position being the machine and the drop-off position being the rack to the transportation system. To control which OU processes which job, only certain OUs capable of handling the issued jobs are linked to the *Logistics Procedure*. The transportation system then integrates the job into its plan, considering the different available OUs and their positions. The procedure then notifies its parent of the estimated completion time, which changes its state from *Idle* to *Delivering*. When the job is finished the component returns into the *Idle* state.

7. IMPLEMENTATION INTO D^{3}FACT

In our discrete event simulation software d3fact (Renken et al. 2011) we use a concept called *composition and aggregation* to remove the need for a static type hierarchy in our component definitions. While static inheritance type hierarchies are simple to understand since it is natural for us humans to arrange objects taxonomically (Sommerville 2004, Shaw and Garlan 1996), they can become hard to maintain because of the limited possibilities for enhancement. With the two patterns *composition* and *aggregation* a type hierarchy can be simplified or completely removed (Gregory 2009, Deacon 2005). The benefit of this concept is exposed when it is used to combine entirely different subsystems into one object. In d^{3}fact an object is represented by a dynamic set of "properties". While there are very different approaches to identify a set of properties, we use a generic container concept to sustain the object-oriented approach common in current simulation model architectures. The container type provides methods to manage its properties (add, delete, get by key, etc.).

Simple properties like numerical values or strings are passive, meaning they do not react to state changes and also do not cause them. These properties are aggregated (the weak ownership): The container object owns them but they are not bound to the life cycle of the container object. The "logic" instead, is an active property, because it does react to state changes. E.g. when an event is caught, the "logic" initiates the processing of the event as a perception. Composition strongly binds a property to a container, which means the property is bound to the container's life cycle and also receives a reference to it. Through this reference the property can access the container, the simulation core, the model and other objects within the model.

We are able to attach so called "listeners" to various objects and also properties. Through these properties, objects are able to get notified when various events or changes happen in the simulated system, for example when other properties are changed. The listener concept allows for handling dynamics in the system, as well as implementing measuring figures such as throughput of a simulation model.

The components presented in are implemented using this object-oriented approach. The *Storage Component* and the *Logistic Component* each are implemented as logic properties combinable in a container object. For commencing operations on the storage, the *Logistic Component* has a reference to the *Storage Component*. While the basic operations are composed into these containers, the specific behavior of the storage and the logistic is each implemented into a separate property. These are used by the logic properties to process events that are triggered by other parts of the scene.

8. CONCLUSION

We presented a generic and flexible warehouse implementation. The presented concept allows the modeling of different types of storages with minimum effort. The components can be combined to form bigger, logical warehouses. This allows the usage of model spanning processes like the fork lifts and workers retrieving goods from our example. Through the separation of the goods and the storage structure we are even able to reconfigure the components at runtime,

enabling rapid prototyping. In combination with a motion planning component, it is possible to change the layout of the warehouse without caring about the motion paths for the operating units.

REFERENCES

Brady, M., 1982. Robot motion: Planning and control. The MIT Press.

Canny, J., 1988. The complexity of robot motion planning. The MIT Press.

Chen, D.Z. and Szczerba, R.J. and Uhran, J., 1997. A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment. *IEEE Trans. Robotics Automat.* 13(5):. 668 – 681.

de Berg, M. and van Kreveld, M. and Cheong, O. and Overmars, M., 2008. *Computational Geometry: Algorithms and Applications*. Springer-Verlag Berlin Heidelberg.

Deacon, J., 2005. Object-Oriented Analysis and Design. ADDISON-WESLEY.

Fischer, M. and Renken, H. and Laroque, C. and Schaumann, G. and Dangelmaier, W., 2010. Automated 3D-Motion Planning for Ramps and Stairs in Intra-Logistics Material Flow Simulations. *Proceedings of the 2010 Winter Simulation Conference (WSC 2010)*. IEEE, Omnipress, 1648 – 1660.

Goetschalckx, M. and Ratliff, H., 1991. Optimal lane depths for single and multiple products in block stacking storage systems. IIE TRANSACTIONS, 23, no. 3, 245–258.

Gregory, J., 2009. Game engine architecture. A K Peters, first ed.

Gunal, A. and Grajo, E. and Blanck, D., 1993. Generalization of an AS/RS model in SIMAN/CIMENA. *Proceedings of the 25th conference on Winter simulation*. ACM, 857–865.

Incontrol Simulation Software, 2012. *Enterprise Dynamics Products*. Available from: http://www.incontrolsim.com/en/products.html [April, 2012].

Klaas, A. and Laroque, C. and Dangelmaier, W. and Fischer, M., 2011. Simulation aided, knowledge based routing for AGVs in a distribution warehouse. *Proceedings of the 2011 Winter Simulation Conference*. IEEE, 1668– 1679.

Latombe, J.C., 1991. Robot motion planning. Springer.

Muller, D., 1989. AS/RS and warehouse modeling. *Proceedings of the 21st conference on Winter simulation*. ACM, 802–810.

Renken, H. and Fischer, M. and Laroque, C., 2011. An Easy Extendable Modeling Framework for Discrete Event Simulation Models and their Visualization. *Proceedings of The 25th European Simulation and Modelling Conference - ESM'2011*.

Shaw, M. and Garlan, D., 1996. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.

Sommerville, I., 2004. Software Engineering. Addison Wesley, seventh ed.

Takakuwa, S., 1996. Efficient module-based modeling for a large-scale AS/RS-AGV system. *Proceedings of the 28th conference on Winter simulation*. IEEE Com- puter Society, 1141–1148.

Vannoy, J. and Xiao, J., 2008. Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes. *Robotics*, IEEE Transactions on, 24, no. 5, 1199–1212.

AUTHORS BIOGRAPHY

Hendrik Renken studied computer science at the University of Paderborn, Germany. Since late 2007 he is a research assistant at the Heinz Nixdorf Institute. His research interests are multi-domain simulation engines and in particular material flow simulation models.

Felix A. Eichert studies business computing at the University of Paderborn, Germany. Since early 2008 he is a student assistant at the Heinz Nixdorf Institute. In his bachelor thesis, he designed the warehouse component described in this paper.

Markus Monhof studies business computing at the University of Paderborn, Germany. Since mid 2011 he is a student assistant at the Heinz Nixdorf Institute. He helps to extend the simulator d^{3}fact to new problem domains.