Improving Flow-based Modeling of Enterprise Systems and Modeling of Custom Warehouse Systems in d^{3}fact

Hendrik Renken and Wilhelm Dangelmaier
Heinz Nixdorf Institute, University of Paderborn, Fürstenallee 11, Paderborn, Germany

Keywords: Material Flow Simulation, Discrete Event Simulation, Material Flow Modeling, Warehouse Modeling.

Abstract: Due to a restrictive design, current enterprise simulation software has shortcomings when it comes to modeling complex business processes and custom warehouse systems. Often, standard processes are altered to add desired functionality. Also, custom (often complex) warehouse components are created by merging readily available components. Typically, this customization has to be done programmatically, which usually results in error-prone and hard to maintain simulation models. In this paper we present a concept to improve the modeling of complex business processes and warehouse systems. Our flexible design allows the combination of business-processes without any programming. It even allows a process to control several information flows at once. This comes in handy when implementing custom warehouse systems.

1 INTRODUCTION

Simulating a system to understand its behavior for certain inputs is a well established scientific method and is broadly used in both research and the industry. Innovative products as well as their corresponding manufacturing processes are to be reviewed at regular intervals in order to improve their efficiency and productivity. For this task enterprise simulation software is widely accepted as an optimization and decision making tool.

Enterprise systems are characterized by a set of processes (often also called *building blocks*) that are applied in a specific order on simple objects (*entities*). Standard-processes found in (almost) every enterprise building block library are e.g. entity sources and sinks, queues, services, buffers, etc. The building blocks are typically implemented as "black boxes", each providing a well-defined function. The blocks can be connected to form a network of functions through which the entities will flow during simulation. This allows for rapid and easy development of processing networks and is called *flow-based modeling*. In a material flow system typically tokens (the entities) flow through a network of production processes (the building blocks). Other examples are: A document-workflow in a company or pedestrians that move along a network to reach certain check points.

However, the *black box* design of the building blocks found in current simulation software can be very restrictive. Consider, that you want to extend a simple conveyor with a quality control process (cp. Figure 1). Since this is not an *of the shelf* building block, you would have to implement a new component with the desired behavior. To ease things, most simulation software allows to extend the integrated blocks. This is done using a specific programming language, e.g. Anylogic uses Java (XJ Technologies Company, 2012b) while Enterprise Dynamics uses its own Scripting Language called "4D Script" (Incontrol Simulation Software, 2012c). That means, you'll have to learn the programming language and the inner workings of the simulation framework you are using. This workflow is complicated and hard to learn. Custom building blocks are difficult to create and maintain. Usually, they are complex and cannot be reused in other models. To ease this problem, we propose a new building block design for flow-based modeling. Our design distinguishes between the location of an entity and the function that is applied to it while it resides in this particular location. For our conveyor example we would set up a *location* representing the conveyor belt and add two processes: The conveyor

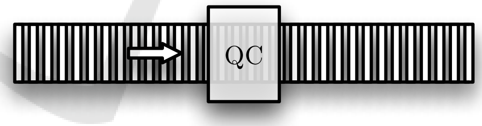


Figure 1: The desired building block: A Conveyor coupled with a quality control process (QC).

Renken H. and Dangelmaier W..
Improving Flow-Based Modeling of Enterprise Systems and Modeling of Custom Warehouse Systems in d3fact.
DOI: 10.5220/0004431900940101
In *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2013), pages 94-101
ISBN: 978-989-8565-69-3

logic to move the tokens within the location and the quality control process to test the bypassing tokens.

Another advantage is the possibility to associate a process with several locations at once. This feature is particularly useful when implementing custom warehouse systems. Then the process can act as a supervisor while local processes maintain the structural integrity of the storage locations.

In the following section we briefly present the concept and its usage. After that an overview over current flow-based implementations and warehouse modeling is given. In Section 4 the concept is presented in detail, followed by a section where the benefits for custom warehouse implementations are discussed. We will give details about the integration of the concept into our research simulation platform d^{3}fact and close with a summary and an outlook.

2 CONCEPT OVERVIEW

Earlier we mentioned that in general the black box design of current enterprise building blocks becomes a problem when it comes to customizing and extending readily available building blocks. In order to make the design more flexible and easier to extend, we wanted to make the different blocks easy to combine to form new ones. Also we wanted to be able to do this without having to program any code.

Reconsider the example of the conveyor which we want to extend with a quality control process (Figure 1). Unfortunately, such a building block is not available as a standard component in the simulation software. Therefore, we have to implement a building block with the desired behavior by ourselves. Let's assume, that each of the two distinct functionalities *Conveyor* and *Quality Control* (QC) are available as separate standard processes.

This opens up several possibilities:

- First of all, if we have access to the source code, we can copy it and merge the separate functionalities into a new component. Usually, to implement this approach we have to learn a new programming language and we need detailed knowledge about the inner workings of the simulation framework. Furthermore, it is error-prone as we would have to merge and handle code that is not our own.
- Second, we can attach the standard components to form the desired building block. This approach is depicted in Figure 2. The problem here is, that this setup does not match the original problem. Here the quality control instance is distinct from the conveyor and furthermore, there are two conveyors used. In this example this argument does not look like a big deal. However, a superior or customer may question this specific setup and because of that, the simulation results.

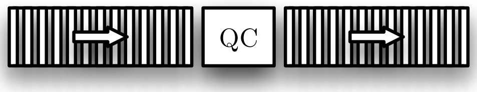


Figure 2: The second approach to implement a conveyor with a quality control.

However, with our approach where the location of an entity is separated from the process behavior, such a process implementation is quite simple. Because of the separation, we can associate several processes with a specific location. Therefore, we can combine the conveyor-process with the quality control into one component. This setup closely resembles the reality and can be implemented by simply combining standard components (without any programming).

By separating the storage (or location) of the entities from the actual processes we gain a new level of flexibility. Of course, this approach is not applicable in every situation. Also, this approach cannot be used to create any desired behavior just through combination. However, with a set of predefined generic processes, this approach allows for a more fine granular process and model customization. We can add new features to a model completely without programming and without copying code.

3 RELATED WORK

3.1 Flow-based Modeling

Today's simulation software packages offer very powerful solutions for the modeling, simulation and analysis of enterprise systems. Often coupled with a graphical user interface and a dynamic process animation, the user is allowed to model and analyze a system via drag and drop, e.g. ((Siemens, 2012; Rockwell Automation, Inc., 2011; XJ Technologies Company, 2012b)). In the following the general approach of most professional and commercial packages will be discussed. The simulation software suites "Enterprise Dynamics", build by Incontrol (Incontrol Simulation Software, 2012a), and "Anylogic" from XJ-Technologies (XJ Technologies Company, 2012a) will be used as examples. Modeling a specific system can be done in many different ways. The set of reasonable approaches often depends on the system that will be modeled and the used simulation software. To help the user getting started, current simulation software provides specialized products for different modeling areas, like “logistics”, “warehouse” or “airport” (Incontrol Simulation Software, 2012b). The mentioned software packages offer *flow-based* modeling as an approach to implement enterprise related systems. Flow-based modeling means to connect building blocks through defined inputs and outputs to exchange entities. Each block then stands for a function or a behavior that is applied to the entities flowing through it. The connected building blocks form a network of functions and behaviors.

3.2 Building Block Architectures

In “Enterprise Dynamics” the basic simulation object is called *atom*. An atom primarily is an unspecified object with certain properties, that can react on events occurring during a simulation run. Using a scripting language of their own (called “4D Script”) changes to the behavior of an atom can be made. A simple inheritance mechanism is available where a derived atom inherits the script code from the parent.

Anylogic ships with a library for modeling (generic) enterprise systems and is based also on the flow-based modeling paradigm. Extending the building blocks with a new features must be done by extending the building blocks using the normal java inheritance mechanism. To implement our “conveyor with quality control” example we would have to derive new java types from the available building block types and re-implementing the new quality control feature. More flexible simulation language frameworks, e.g. Simula or GPSS ((Nygaard and Dahl, 1978; Ståhl and AB, 2011)), offer a wider flexibility, since the model libraries are defined on a less detailed level. Since the programmer is aware of the entire programming interface (API), he is able to easily implement a new feature. Nevertheless, those frameworks base their model elements in most cases on a specific hierarchical type approach. As mentioned by (Gregory, 2009) and (Renken et al., 2011) these extension and customization technics can produce hard to maintain and hard to reuse code and model structures. Furthermore, they lower the users acceptance because typically the user has to learn a programming language and the created structures tend to be error-prone and hard to debug.

3.3 Warehousing

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 generic components which lower the modeling time, but gives general advices what to consider when modeling a warehouse for different types of simulation results.

(Takakuwa, 1996) is 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 *Automated Storage and Retrieval Systems* (AS/RS) systems with AGVs the presented components for the warehouse are not as highly customizable as ours. The only layout that is supported, is the aisle based rack layout of AS/RS which is one building block for which some parameters could be set. Therefore, splitting a warehouse in different components is implemented by making AGVs and conveyors parts of the warehouse. In our simulator there also exists components for AGVs and conveyors, but we do not limit their application to warehousing scenarios.

4 FLEXIBLE BUILDING BLOCK DESIGN

Let’s review the conveyor example from Section 1 in detail. The initial problem was to add a quality control element to an available conveyor component. However, due to a restrictive component design in most enterprise simulation software this setup has to be implemented as a custom component. This has to be done most likely in a specific programming language, making the whole workflow unintuitive and hard to master (Gregory, 2009; Renken et al., 2011).

To ease the development of custom components we propose a different base design for components for flow-based process networks. Our approach separates the location of an entity from the processes that are applied by a specific building block. Figure 3 shows the concept in general.

On the left side, a typical building block is displayed. It encapsulates a process (or function) and

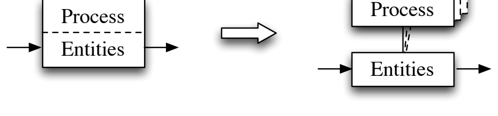


Figure 3: A typical building block (left side) consists of a function and a location to store entities. In our design (on the right side) they are separated.

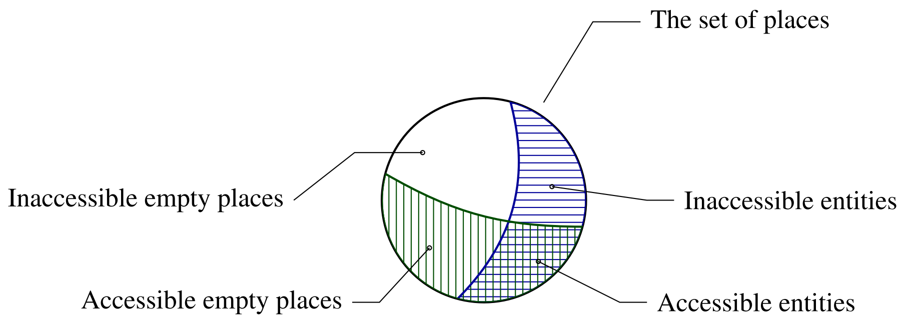


Figure 4: The structure of a location.

a data structure to store entities. During simulation the function is applied to all entities passing through. However, in our concept we deliberately separate the process and data structure (location). Such a location does not have any functionality. Instead, processes can be associated with a location, so that they apply certain functions to entities passing through.

Therefore, we represent the conveyor from the Introduction as a location with a conveyor process. The location resembles the conveyor belt, as the conveyor belt *stores* the entities as long as they move over the conveyor. The process implements the logic to *move* the entities over the conveyor belt. By simply adding the quality control to the location we can implement the solution discussed in Section 2. We will examine the different aspects of this solution in more detail in the upcoming paragraphs.

4.1 Locations

A location represents a place where entities can be stored until they are processed or requested by another location. The location concept is a generalization of storage concepts. It covers a wide range of storages like workplaces and -tables, warehouses and shelves. Consider a simple shelf that stores some boxes with unknown content. Some of the places in the shelf are occupied by boxes and some are free. Because the shelf is tightly packed, places in the back of the shelf can be inaccessible due to boxes in the front. Systematizing this example results in the following observations: A storage system of some sort has a specific number of places (may be infinite) to store entities. The places may be free or occupied by entities and they may be accessible or inaccessible. Combining these two attributes gives us four sets of places as depicted in Figure 4.

The location lacks rules or logic to handle model dynamics, which are added through processes. Processes that are attached to the location can make changes to the four sets by adding(removing) entities or by setting a place (in-)accessible. The conveyor process from our example treats places of the location as discrete places on the conveyor belt. This means, it has a specific order of the places, treating the first place as the beginning of the conveyor and the last as the end. By moving a token from its current place to the next one (if it's not blocked) the conveyor mechanics are simulated.

4.2 Processes

Processes are associated with one or more locations. That means, that a process can alter the state of every place of these locations and may create new entities and put them into a specific place. A process that is associated with a location also is informed about all changes made. Therefore, it is able to react on changes made by other associated processes.

The conveyor process works as follows: As depicted earlier, it has a certain ordering of the places within the location. The first place, resembling the beginning of the conveyor is always accessible. Tokens from other locations will arrive here. Arriving tokens are then moved along the ordered list of places, reflecting the movement of the conveyor belt. Also, the last place (which depicts the end of the conveyor) is always accessible. That means, that tokens that arrive at this place can be picked up by adjacent locations for further processing.

The initial problem was to add a quality control to the conveyor belt (cp. Figure 1). Implemented as a process within our concept, this component can be associated with the conveyor location and furthermore with a specific place, e.g. as depicted in Figure 1 with the one in the middle. Now the conveyor process will move every token past the quality control, where it can test the token for defects. If a token fails the control the component can simply remove the token from the location, making space for the next token.

4.3 Connecting Locations

We use a system that is similar to the *channels* used by Incontrol in "Enterprise Dynamics" to connect their building blocks (Incontrol Simulation Software, 2009). But instead of connecting port objects, we are aware of the different places of a location. Therefore we can connect the places of the locations (cp. Figure 5). Several connections are then packed into one *channel*.

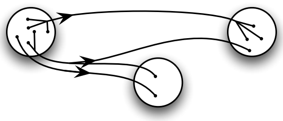


Figure 5: Three locations (circles) are connected through *channels* (lines) with each other. A channel can contain several connections (indicated by the number of ends on each channel side).

A channel moves entities from one location to another (in an atomic transition) only if three conditions are met:

1. All places on both ends of a channel must be accessible.
2. All places at the beginning of a channel must contain an entity.
3. All places at the end of a channel must be empty.

Because of these rules, a process can manage how and when entities move from one location to another just by managing the state of the places. As shown in Figure 5 even more complex channel setups are possible, connecting three locations at once (through one channel).

Of course, it is possible to implement channels with additional conditions. This e.g. allows to implement channel-based forks, where a simple switch controls the destination for the next tokens.

4.4 Analytics Made easier

Another advantage of the representation of arbitrary storages by a generic location concept is an easy implementation of model analytics. With the *black box* design found in common simulation software packages, arbitrary analytics components are needed for the different building blocks (Incontrol Simulation Software, 2009). However, because of the generic representation and full access to the ongoings within such an entity location, analytics of common ratios like *lead time*, *work in process*, *throughput* or *utilization* can be easily implemented. For example, the *work in process* of a set of building blocks simply is the sum of all entities contained in the associated locations. Furthermore, the ratios can be reused throughout different problem domains as long as the entity storages can be represented by the generic location concept.

A very deep analysis of the ongoings in a warehouse are also possible. For example, the utilization for every single place in the warehouse. With the full access to the entity movement, we are capable of tracking the movement for every single entity.

5 CUSTOM WAREHOUSE BUILDING BLOCKS

The previous section laid the focus on the combination of different processes into one component. As mentioned before, our concept also allows the association of one process with several locations simultaneously. In this section we want to explore this possibility and its benefits when implementing custom warehouse processes.

5.1 A Simple Block Storage

Let's start with a rather simple example - that of a block storage. A block storage usually holds palletized goods of some sort. There is no supporting structure like a rack, the goods are just stacked upon each other.

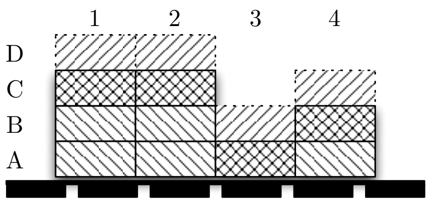
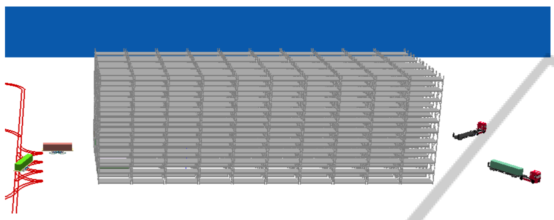


Figure 6: Side view of a block storage. Each rectangle is a place. The pattern describes the state of the place.

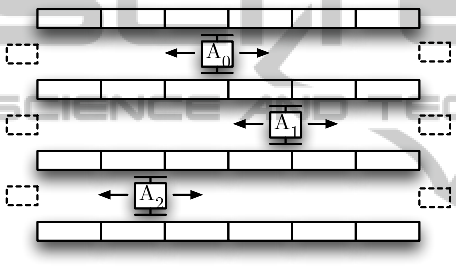
Figure 6 depicts the state of a block storage. Some goods (e.g. *A1, B1, A4, ...*) are inaccessible while others (*C1, C2, A3, B4*) can be access (e.g. with a forklift). However, as with the goods, some empty places are inaccessible, namely *C3, D3, D4*), and also there are places where additional goods can be stored (*D1, D2, B3, C4*). This concept perfectly matches with the structure of our locations introduced earlier (Figure 4). The places of the location are simply mapped to specific three dimensional coordinates of the block storage space. A specific *block storage process* describes the dynamics of the storage, e.g. that a palette cannot be stored in midair. Consider that we add a new palette to *B3*, then the process would set *A3* inaccessible and *C3* accessible.

5.2 A Complex Rack Storage Setup

Another advantage of the separation of the locations from processes, is the simultaneous association of a process with several locations. Figure 7 depicts a model of a container rack storage with four racks, three AS/RS and three load/unload areas on each side. The whole model is described in (Asef-Vaziri and Khoshnevis, 2000).



(a) The simulated rack storage.



(b) Diagram of the rack storage.

Figure 7: A rack storage for maritime containers. It consists of four racks, three AS/RS and three load/unload areas on each end (dotted rectangles).

The storage is used as a buffer in a maritime container terminal. Container unloaded from a ship are stored in the racks until trucks become available to transport them inland. The same applies for the other way round. In this scenario the AS/RS take containers from both ends and store them in one of the two racks accessible to them. When a ship or truck arrives specific containers are brought to the appropriate load/unload stations. As depicted, each rack can be accessed from both sides and therefore from two AS/RS.

In a previous attempt standard components where used to model the setup. A single AS/RS had to be implemented as two different entities, one for each reachable rack. Also, the racks had two rows instead of one so that two AS/RS could not get into conflict when accessing a storage place. This approach didn't match the originating setup (cp. Figure 7(a)) and furthermore, the synchronization of the two entities representing one AS/RS was very cumbersome and error-prone. In the end we had to reprogram this storage from scratch to make it functioning as intended.

However, with the architecture described in this paper such a complex warehousing component can be implemented much more easily. The most interesting part in this implementation is the communication of the AS/RS with each other to avoid conflicts in accessing the storage and load/unload areas. As we will see, this communication can be done solely through state changes on the locations. The initial setup for A_1 is shown in Figure 8.

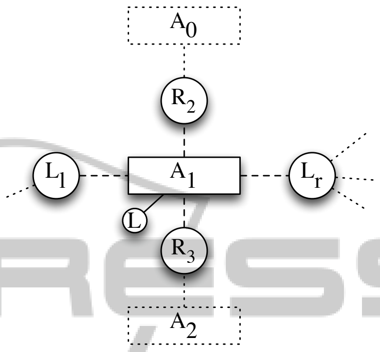


Figure 8: The setup for AS/RS A_1. It is connected to five locations: Two racks (R_2, R_3) and two load/unload areas (L_l, L_r) and a location to store the currently carried container (L).

The AS/RS itself has one location L where it stores the one container that it currently carries. Furthermore, it has access to the two load and unload areas at both sides, depicted as L_l and L_r. And, of course, it can access the two racks R_2 and R_3 - as the other two AS/RS can (also shown in the figure).

5.2.1 Handling the Load/unload-Areas

The load/unload-areas serve as gateways to the rest of the model. At any point in time trucks can arrive, blocking the areas until they have been served by the AS/RS. However, due to simulation of a warehouse with particular orders, a supervisor will deploy jobs for the AS/RS as needed. Such a job contains specific instructions how to handle a certain container or how to serve a specific truck.

5.2.2 Accessing the Racks

The places within a rack can be accessed from both sides and therefore by two independent AS/RS. Therefore, a synchronization has to take place to avoid conflicts during access. For example, each of the two AS/RS might want to store a container in the same place. This conflict can be resolved in several ways: First of all, one can make the AS/RS more intelligent, allowing them to directly communicate with each other. Due to the asynchronous setting such a communication protocol can be come very complex and hard to maintain. Another approach would be to implement a supervisor for the warehouse system. This leads to a very complex supervisor which has the same drawbacks as the solution before.

Instead, in our implementation both AS/RS can easily communicate through the states of the places. The places can be seen as a resource that has to be locked before it can be accessed. An AS/RS can indicate the (future) usage of a particular place simply by setting a place inaccessible. Either to store a container (if it's an empty place) or to handle the container that is stored in that place.

The communication through the states of the places within the locations has the advantage that no explicit communication is required, that is is asynchronously and that no external supervisor or conflict solver is needed. This makes this approach simple, clean and easy to understand and implement.

6 IMPLEMENTATION DETAILS

As mentioned earlier, we implemented our concept into our research simulation platform d^{3}fact. Here we report on some interesting details of the implementation.

Let a location have n associated processes $p_0, \dots, p_n$. Now an arbitrary process $p_s \in \{p_0, \dots, p_n\}$ makes changes to the location. This creates an update u_s containing all information about the changes, i.e. added and removed entities, now accessible or inaccessible places. Unfortunately u_s starts an update cascade. That means, another process p_k responds with its own update u_k to the update u_s. Now u_k again causes another process p_r to respond, and so on. A trivial update mechanism would inform each of the n processes of every update. This can lead to an efficiency problem. For example one process occasionally creates new entities while another process destroys them to replace them with completely new entities. Every associated process now gets both updates, even if the entities created in the first place do not last long or have been destroyed already. Carrying out each update can lead to a huge overhead when updates contain oppositional or obsolete information. Also, processes would have to be capable of determining the obsolete information in the update or the differences between the last known and the current state of the location. This would make the implementation of new processes more difficult, especially for new users.

Instead of informing all processes about all updates, we accumulate updates. This reduces the times a process is informed about updates to a minimum and the updates do not contain stale information. The update mechanism we came up with can be found in Listing 1. It resides in the location implementation. Changes made by a process to the location trigger the `UPDATE()` method. Here the parameter `p` is the process initiating the changes and `u` is the update initiated by `p`.

```
#p is the source process and u the update
UPDATE(p, u)
   put(u) → stack

  #inform processes in front of p about u
  0 → i
  peek(stack) → u
  while (P[i] ≠ p)
  {
    u → inform(P[i])
    i+1 → i
  }

  #do not inform p about u
  if (|stack| > 1)
  {
    #merge u with previous update
    poll(stack) → v
    poll(stack) → w
    put(v ∩ w) → stack
  }
  else
  {
    #there is only one update left on the stack
    i+1 → i
    peek(stack) → u
    while (i < |P|)
    {
      u → inform(P[i])
      i+1 → i
    }
    poll(stack) #clear stack
  }
```

Listing 1: The Update-algorithm.

We inform the processes in the order they were added. If one process happens to start a new update during a running update we suspend the current update and start from the beginning (with the new update).

Figure 9 shows an exemplified update cascade. Horizontal lines indicate which process is informed about which update at which point in time. Horizontally the processes $p_0, \dots, p_n$ are shown. The time line is displayed vertically. A back pointer indicates the accumulation of two updates. Basically the procedure starts with an update u_s. Now we consider p_k to be the next process responding to the changes made by u_s. That means, that u_s is applied to all processes before process p_k. Upon informing p_k about

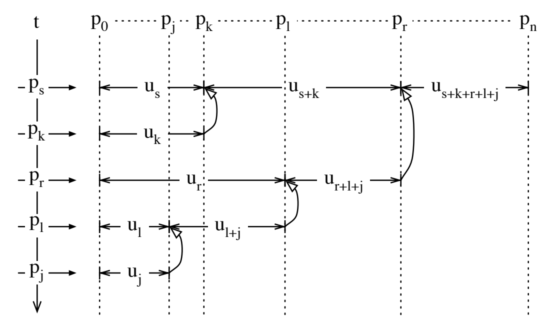


Figure 9: Generic example showing how our update method works. Horizontally all processes are displayed while the time line is plotted vertically.

u_s it triggers a new update u_k. As stated before we now start from the beginning, informing all processes before p_k (cp. Listing 1, Lines 5-12). After doing so, all processes $p < p_k$ are informed about the updates u_s and u_k. Before informing the processes $p > p_k$ we merge both updates into the new update u_{s+k} eliminating all oppositional information (Line 20, Listing 1). As depicted in Figure 9 this update algorithm can also handle recursively triggered updates.

7 CONCLUSIONS

In this paper we report on a concept that enhances the flow-based modeling found in current enterprise simulation software. The presented concept splits the current building block design into a location where entities are stored and the building block behavior (*process*). Through this separation an arbitrary number of processes can be associated with a location, enabling the combination of them. Another advantage is the possibility to associate a process with a set of locations. This enables the easy implementation of management logic like those used in warehousing systems. Other advantages are the simple analysis of the systems behavior and the code portability and flexibility. We showed that our location design covers a wide range of storage types, like working places, trucks, machines, shelfs, block storages, etc.

REFERENCES

Asef-Vaziri, A. and Khoshnevis, B. (2000). Performance analysis of automated technology in maritime container terminals. *Progress in Material Handling Research*, pages 165–178.

Gregory, J. (2009). *Game engine architecture*. A K Peters, first edition.

Incontrol Simulation Software (2009). Tutorial ed 8.

Incontrol Simulation Software (2012a). Enterprise dynamics. Accessed April, 2012. http://www.incontrolsim.com.

Incontrol Simulation Software (2012b). Enterprise dynamics products. Accessed April, 2012. http://www.incontrolsim.com/en/products.html.

Incontrol Simulation Software (2012c). Enterprise dynamics technical overview. Accessed April, 2012. http://www.incontrolsim.com/en/ed-platform/technical-over view.html.

Muller, D. (1989). As/rs and warehouse modeling. In *Proceedings of the 21st conference on Winter simulation*, pages 802–810. ACM.

Nygaard, K. and Dahl, O.-J. (1978). The development of the simula languages. *SIGPLAN Not.*, 13(8):245–272.

Renken, H., Fischer, M., and Laroque, C. (2011). An easy extendable modeling framework for discrete event simulation models and their visualization. In *Proceedings of The 25th European Simulation and Modelling Conference - ESM'2011*.

Rockwell Automation, Inc. (2011). Arena simulation software by rockwell automation. Accessed Feb. 1, 2011. http://www.arenasimulation.com/.

Siemens (2012). Simulation & testing. Accessed April, 2012. http://www.industry.siemens.com/industry solutions/global/en/cross_industry_solutions/automat ion_it/simul_test/Pages/Default.aspx.

Ståhl, I. and AB, B. (2011). Webgpss - simulation made simple. Accessed Feb. 1, 2011. http://www.webgpss.com/.

Takakuwa, S. (1996). Efficient module-based modeling for a large-scale as/rs-agv system. In *Proceedings of the 28th conference on Winter simulation*, pages 1141–1148. IEEE Computer Society.

XJ Technologies Company (2012a). Discrete event simulation modeling tool. Accessed April, 2012. http://www.xjtek.com/anylogic/approaches/discreteevent/.

XJ Technologies Company (2012b). Why anylogic simulation software? Accessed April, 2012. http://www.xjtek.com/anylogic/why_anylogic/.