2018

A Framework for Modeling Material Handling with Decentralized Control

Kai Furmans
Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, kai.furmans@kit.edu

Kevin R. Gue
Department of Industrial Engineering, University of Louisville, kevin.gue@louisville.edu

A Framework for Modeling Material Handling with Decentralized Control

Kai Furmans
Institute for Material Handling and Logistics
Karlsruhe Institute of Technology
Karlsruhe, Germany
kai.furmans@kit.edu

Kevin R. Gue
Department of Industrial Engineering
University of Louisville
Louisville, KY, USA
kevin.gue@louisville.edu

***Abstract*—Despite decades of research in material handling, the academic community still has no accepted way of describing material handling requirements in a way that machines and algorithms can process them. Such a "way of describing" requires a language with which to describe requirements, objects, relationships between objects, and so on. We propose a modeling framework that differs from existing research in two ways: First, we address material handling modeling from the bottom up rather than from the top down, meaning we define a set of elementary functions and then combine them into processes and more complex relationships that allow us to describe any material handling activity or requirement. Second, the framework assumes material handling devices and objects can make decisions and therefore that control can be decentralized, as might be required in an Industry 4.0 environment.**

The ultimate goal is to be able to create truly flexible material handling systems, where expansion or redesign of the system is feasible and easy. This requires a system architecture, where the design of the systems components, the software architecture, and the language are congruent.

***Index Terms*—framework, modeling, material handling, decentralized control**

I. Modeling material handling

As Leon McGinnis [1] has argued for several years, the mark of a mature academic discipline is an accepted set of tools with which scholars can describe its subject matter, problems, and solutions. For material handling research, the lack of such a set has meant creating and re-creating tools on an as-needed basis, and those tools are necessarily difficult to share or to improve upon directly. As our colleague Jeff Smith once asked, "How many times has a PhD student started a warehousing research project by creating his own unique version of `class pallet`?"

Previous research has approached the modeling problem from a top down perspective, starting with a warehouse, for example, and then breaking down requirements and design questions into ever smaller parts. We pursue the problem from the bottom up, starting with specific things (classes, to use the language of object-oriented programming) and a set of elementary functions to describe their activities and finishing with a desired system. We believe this approach has at least two benefits. First, it facilitates the integration of entirely new categories of material handling devices into existing models and operating material handling systems. As long as a new device can be described as a particular *type* of thing, it can be integrated into the framework, even if it has never been seen in its particulars (think: the introduction of Kiva robots and mobile shelving). Second, the bottom up approach enables the same modeling framework to be used across the boundaries of research, development, and implementation. Our goal is to develop a framework that describes a "this" as a type of "that," which then allows software objects created with the language to communicate in a way that emulates a real material handling system. Ultimately, we wish to embed the language in devices to allow them to communicate in real operations. Thus the same language could be used to design, model, and operate the material handling system.

As we describe in [2], a standardized communication protocol is a design pattern required to build truly adaptable and expandable systems. The communication protocol we describe below is a suggestion for this requirement. Our approach is also limited: we rely on standardized building blocks and decentralized decision making based on decentralized information. This might lead to sub-optimal solutions and also to redundant storage of information. Because solving real material handling systems optimization problems in real time is very difficult, and because prices for hardware are continuing to drop, we assume that these shortcomings are acceptable.

A primary feature of our research is decentralized control, meaning devices can make decisions and the ability to communicate with other devices such that centralized, external control is unnecessary or at least minimally necessary. Decentralized control is a particular interest area of the authors and a feature of systems designed for Industry 4.0, but the framework could also be used, perhaps with some modification, for systems with centralized control.

II. Literature Review

Several authors have proposed modeling frameworks for supply chain [3]–[5], logistics [6], and warehousing systems. The latter group is most closely related to our framework. Several papers offer formal frameworks for warehouse design [1], [7]–[9]. Abdoli [10] and McGinnis [9] describe object-oriented models for warehousing systems. McGinnis's work in particular mentions "embodiment design" as a means of

accomplishing functional requirements, the so-called "how" to the "what" of warehouse operations.

Among established warehouse modeling frameworks, our work lies most closely to the Control Layer described by Sprock et al. [11], which is the lowest level addressed by the authors. [12] also discuss a "conceptual framework for operational control in manufacturing systems."

A significant difference between our work and existing research on warehouse modeling is that we do not address design decisions such as, "what should the warehouse look like in order to meet operational requirements?" By "operational requirements," this question usually refers to aggregate performance measures such as throughput capacity or average time to respond to an order. These are important issues, of course, but they are not the focus of our research. We assume the design of the warehouse is given, and then ask how to model the devices inside such that specific orders and other material handling jobs are accomplished to meet their individual requirements. We are especially interested in the control layer referred to in some of the literature but not described in detail.

III. The Framework

The modeling framework defines material handling devices, which we call *modules*. The modules have the capability to perform one or more physical functions related to handling or storing of material, which we call *objects* (not to be confused with software objects). Modules are capable of storing relevant information, contributing to decision making, and then controlling the execution of these decisions. Decision making and coordination between the modules takes place with cyber- (or information-) functions. Objects are described by a set of attributes, which include an identifier and any requirements for storage and handling, such as size, weight, and orientation.

Modules consist of all the ordinary material handling devices such as lift trucks, AGVs, shelving units, storage racks, and so on. Embedded in the cyber functions and processes is the ability for modules to control their own activities. If a module happens to be handled by the material handling system, it also gets the properties of an object.

Figure 1 illustrates the framework for material handling activities. At both the physical and cyber (or information) levels, a very limited set of functions can be combined to form processes. These processes can then be combined to define and accomplish material handling jobs.

IV. Physical Functions

In our framework, *functions* are the most basic elements of material handling activity. We contend that all material handling operations are comprised of the following physical functions:

- **Store**: To store an object is simply to hold or possess it. For example, a slot in pallet rack *stores* a pallet; a flow rack *stores* totes; and so on. With store we bridge the time from the availability of an object until it is requested and retrieved.

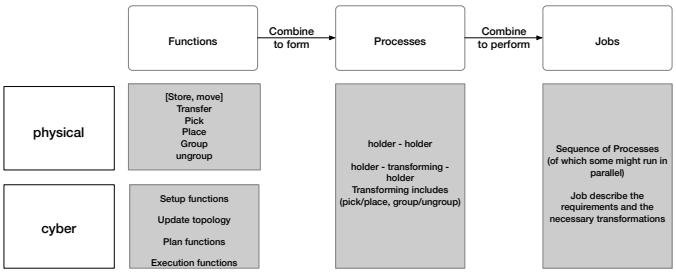

Fig. 1: The framework for material handling with decentralized control.

- **Move**: A move changes the location or the orientation of an object without changing the container or holder. There are two types of moves: (1) to move along a track or conveyor (to bridge a distance), and (2) a device can move itself. For example, a section of powered roller conveyor changes the locations of cartons on itself; a lift truck moves a pallet (and itself) from receiving to another location; and an empty lift truck moves itself from one location to another.
- **Transfer**: The transfer function moves an object from one module to another module. This could be the transfer from one conveyor onto the next or the transfer of a pallet from the lift truck or an AS/RS-machine to a storage location. Transfer is distinct from move in that it implies a change of possession from one device to another.
- **Pick**: To pick an object is to select and remove it from its current container or holder. For example, we pick a carton from a pallet, an item from a carton, or a package from a tote. Pick is different than move in the sense that the object was "inside" a container of some sort before the pick, and after the pick some objects were left behind.
- **Place**: To place an object is to put it into a container or holder. For example, workers place items into bins at a put station; robots place items onto a pallet at a unitizing station.
- **Unitize**: Unitizing involves a physical restraint that allows movement as one entity, and therefore not separately any more. A reason for unitizing might be a change of transportation or storage device. For example, to stretch wrap or band cartons such that they are moved only as a larger unit is to unitize them.
- **Separate**: To separate a unitized object allows access to individual objects, which are inside of the previously existing unit. For example, to break stretch wrap so that cartons can be moved individually and not as a group is to separate them.

In general, physical functions have as many as three requirements: *power, control* and *space*. Power refers abstractly to "that which causes movement." For example, a conveyor cannot move cartons unless it is plugged into a source of electricity, and non-powered skatewheel conveyor cannot move cartons without a human to push them. Control refers to how power is delivered to the device, and therefore is always

necessary in the presence of power. For example, a human applies the power residing in a lift truck by pushing the throttle. All physical functions have a space requirement. For example, a robotic arm cannot place a carton in a tote if the tote is full or too small.

V. Material Handling Modules

At an abstract level, a material handling module (lift truck, bot, AS/RS) is simply a collection of one or more functions, but we believe that defining classes of material handling modules that embody these functions is helpful for practical reasons. Below is a set of four classes which we believe is both comprehensive and extensible:

- Holder: The primary function of a holder module is simply to store one more more objects. Examples include a pallet position in pallet rack, a slot in carton flow rack, or a section of flooring on which products can be stored.
- Mover: A mover moves or transfers objects from one location to another. Examples include lift trucks, conveyors, bots, and humans.
- Picker-Placer: The primary function of a picker-placer is to perform these two operations. Examples include humans and robotic arms.
- Unitizer: A unitizer transforms an object such that its handling unit changes. Examples include stretch wrap machines and depalletizers. (Notice that a "unitizer" may also separate an object into many smaller objects.)

In general, material handling devices are either stationary, movable, or mobile. A *stationary* device is bolted to the floor or very difficult to move; a *movable* device has wheels or can be easily moved (as on a roller table), but has no ability to move itself; a *mobile* device contains its own power and control, and therefore is fully autonomous with respect to moving itself. Each class embodies functions, each of which has requirements. Only when all requirements are met can a function be performed.

We intend these categories to be flexible enough for modelers to use them in the way that seems best. In many cases a physical device could be considered in more than one class. For example, a mobile robotic arm that picks and then transports items to a collection bin is both a mobile picker-placer and a mobile mover, and if it stops for a while, perhaps even a mobile holder. Nevertheless, *we believe that all current or imagined material handling devices can be defined by the physical functions they perform.*

We also define a *collection* to be a set of modules working together as a permanent or semi-permanent entity to accomplish unique functions, processes, or jobs. For example,

- An A-frame is a collection containing many Stationary Holders (slots or cartridges), a Stationary Mover (conveyor) for totes, and multiple Stationary Movers (ejection mechanisms) for eaches.
- A tilt-tray sorter is a collection that contains a Stationary Mover (circular conveyor) and many Stationary Movers (diverters) that move cartons into chutes.
- An AS/RS is a collection of Stationary Holders (racks, slots) and Mobile Movers (cranes).
- A shuttle-based storage and retrieval system is a collection of Stationary Holders (racks, slots) and Mobile Movers (shuttles, vertical lifts).

Because modules might be limited by the size or weight of the objects they can handle, the framework requires an *object-to-module compatibility matrix* that defines allowable operations. Because not all modules are compatible with each other when transferring an object, the framework requires another compatibility matrix describing the relationship between modules, the *module-to-module compatibility matrix*.

VI. Cyber Functions

A number of cyber or information functions are required to enable these physical functions and processes. Before discussing these functions, observe that information functions presuppose the ability to communicate. We assume that material handling modules are of two types—those with the ability to communicate and those without. Communicating modules (AGVs, lift trucks, humans via handheld devices, conveyors with control, etc.) possess the ability to send and receive messages with other communicating devices via physical or wireless network connectivity. Modules unable to communicate (slots in pallet racking, floor space in block storage, etc.) have *caretakers* responsible for communicating on their behalf.

A caretaker is any module that has the ability to perform a physical function with the non-communicating module. For example, caretakers for a slot in pallet rack would be all lift trucks and other devices that can store or retrieve pallets from that location. Any device executing a *transfer* (physical function) with that slot broadcasts an update on the status of the slot to all other caretakers. This way, caretakers know the status of all modules under their care at all times.

Cyber functions can be thought of in three phases of material handling activity: setup, planning, and execution.

A. Setup Functions

After the physical design is established, material handling modules must establish connections among themselves. Setup begins by creating a means for communication, which might be physical via wires or virtual via a wireless network. After physical connections are established, the following functions are available:

- **Announce**: Announce the presence of the module in the network.
- **Establish immediate neighborhood**: Make connections to physical neighbors, if appropriate. We assume that modules can determine which other modules are either directly connected or close enough to perform joint functions and processes.
- **Disestablish immediate neighborhood**, which might be used when changing a configuration. For example, an AGV or robot that docks at a transfer station would establish and then disestablish an immediate neighborhood.

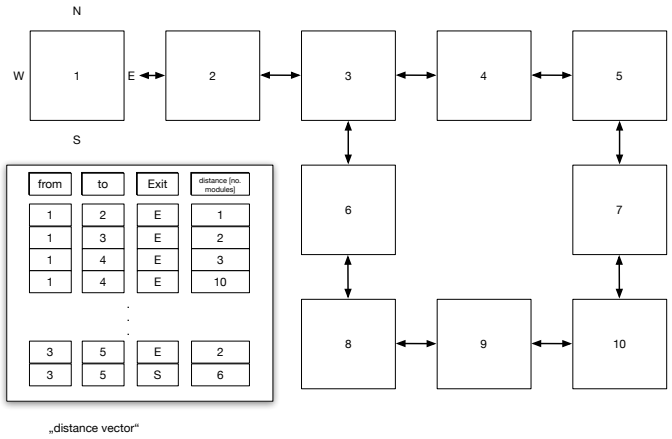

Fig. 2: Example of a network with portions of the associated distance vector for Modules 1 and 3.

- **Disconnect**: Announce to neighbors that the module is not present anymore.
- **Disconnect detected**: The neighbor detects that a module is no longer connected.

Each module requires knowledge about the modules connected to it. We say such modules are in the same "neighborhood." Modules also require knowledge of how objects they contain can reach any other allowable module in the system. Specifically, the host module must know through which of its exits to send the object and what is the distance to the destination module. Notice that there may be many possible paths to a destination module; therefore we keep several entries per destination module in this list. Although this results in a dynamic list, which each module maintains, we refer to the result as the "distance vector."

When modules change their location in the system (via ordinary travel, for example) or leave the system by becoming inactive, other modules need to know where they are in order to calculate distances and plan routes. Therefore, at regular intervals the distance vector is updated by exchanging the topological information. This also serves as a sort of "heartbeat," where the absence of a new distance vector indicates that a module is no longer available.

The following function updates the distance vector:

- **Update distances**: The distance vector associates with each other module how far it is to this module, if the connection is made through a particular neighbor. It indicates to immediate neighbors which other modules can be reached directly through a module and the known distances to other modules. See Figure 2 for an example.

B. Planning Functions

The main task of material handling systems is to move things and store them until they are needed. Other tasks are related to the transformation of the items themselves, such as packaging. For customer orders several items have to be picked, collected and then packed into one or several handling units. On the receiving side, items are received in units perhaps unfit for storage, in which case they must separated, possibly repackaged, and then stored in smaller quantities.

Because our proposed modules are only able to perform a limited number of tasks and are distributed over a warehouse, distribution center, factory or similar facility, it is necessary to identify, find, and enlist the modules required to do a job, because only a combination of modules is able to perform the job.

We begin with the simple movement jobs, which fall into three categories:

- **Source known, destination known**: The object and its current location are known; therefore we must find modules able to transport from the source to the destination and reserve them for the job.
- **Source known, destination unknown**: The object and its location are known, but a suitable storage location must be found and all the modules required to transport the object must be identified and booked for the job.
- **Source unknown, destination known**: The object type (e.g., part number), required quantity, and destination (e.g., packing area) are known; the current storage location(s) of the corresponding objects must be found and then all modules required to transport the objects from their current locations to the destination must be found and booked.

Therefore, three different tasks must be solved, using the cyber functions of the modules in our material handling system:

- **Find route**: find a suitable route from the source to the destination and make reservations for an object to be moved along this route.
- **Find object**: identify a set of modules where objects satisfying these characteristics are held (or stored).
- **Find storage location**: identify a set of modules that satisfy the characteristics and are not currently holding anything.

For routing, we must find a path from the source to the destination and make the reservations that allow the movement of the object without creating deadlocks or livelocks. The "logical time" method for doing so is described in [13].

Routing is an iterative, bilateral negotiation cycle between modules in the same immediate neighborhood. Negotiation is possible because we assume cyber functions are much faster than physical functions. Furthermore, bilateral negotiations allow a decentralized implementation of all functions in the framework. The basic logic is: ask a neighbor that has not yet been asked and which is the next best solution, whether it is willing to participate in the moving of an object. This request is passed from module to module until the destination is reached. Then a cascade of *willing* messages are sent along modules connecting the destination and the source. These messages are followed by a cascade of *commit* messages in the opposite direction [14]. The cyber functions used for routing are:

- **Find next module**: an internal function of each module, including those acting as caretakers. This function identifies based on its distances vector the next best, but not yet requested module on the route.
- **Request**: send request to the selected module to see if it is available to participate in the process at a future point in logical time.
- **Check request**: an internal function of each module, including those acting as caretakers, that determines the earliest (logical) time that a request can be fulfilled. It might be necessary to send requests further downstream before the earliest time can be determined.
- **Reply to request**: Returns *willing* if the request can be fulfilled, possibly with a logical time at which the request can be fulfilled. Returns *not willing* if the request cannot be fulfilled.
- **Commit**: Sends a *commit* message to a selected module if it answered with *willing*.
- **Cancel request**: Sends a cancel request message to all modules requested but not committed to. This message releases all modules that responded with *willing* but were not selected because a better solution was found. All reservations made by requested modules receiving *cancel request* are canceled.

To find a suitable and empty storage location, we use broadcast messages. Five functions are required:

- **Find storage location**: an empty storage location for an object with specified characteristics is requested.
- **Empty storage location available**: responds to a request with yes or no, and possibly with an available time. Yes implies that the holder is empty and that no previous reservation before that time has been accepted.
- **Commit to storage location**: notifies a storage location (a holder) that it should book the reservation at the appointed time.
- **Cancel request for storage location**: informs the storage location that it will *not* be used.
- **Process requests**: an internal function that keeps track of requests and responses.

Because the distance vector allows a module to determine which locations are closest in time, the routing task starts with the closest destination.

Searching for products, or more specifically for holders or caretakers responsible for them, is analogous to searching for locations. Modules use a similar set of functions.

The physical functions *unitize* and *separate* have a cyber counterpart:

- **Group**: To group objects is to consider them as a single object. Grouping is frequently associated with the physical function of unitizing. It is important not to lose the information about the objects, which are now inside of a group, in case they have to be accessed after a later ungrouping.
- **Ungroup**: To ungroup objects is to consider them as separate objects for cyber processes.

To see how these problems might be addressed in practice, consider the activity of receiving, which is the task of admitting a product into the system, finding a location for storage, and transporting it to that location. Upon first detection and identification, say by scanning at the trailer door, the device performing the scan recognizes the product type and determines via the object-to-module compatibility matrix the preferred means of storage. Suppose the item (a pallet) must go to bulk storage. The handheld scanner performs the function *Find storage location* and broadcasts a *request* to all caretakers of pallet locations. One or more caretakers responds with *Empty storage location available* on behalf of empty locations. The scanner module responds to its preferred storage location with *Commit to storage location*, perhaps by using a closest open location algorithm and executes *Cancel request* to all other requested caretakers.

Once the storage location is reserved, the handheld starts to plan a route to the selected storage location by executing the *Find next module* function. It then *Requests* that neighbor, which, if it can execute the entire task, responds with *Willing*; otherwise, it the request is passed along the shortest route until it reaches the caretaker of the selected storage location. Once the request reaches the destination successfully, a *Willing* message is passed all the way back to the scanner module, which then answers with a *Commit* message back to the destination location. If the shortest path cannot be booked, the second best path is tried from the first module at which a *not willing* message was received. This process ends when a route has been found or if all neighbors have answered with not willing. If the latter has occured, after a random timeout the same process is started again (similar to the CSMA/CD-protocol).

C. Execution Functions

As soon as the route has been successfully established, the physical transactions are started. Since the implementation of the physical function requires the control of drives and sensors in a real-time environment with a bilateral coordination between adjacent modules, execution is delegated to a lower level and encapsulated in a *begin transaction* and *end transaction* sequence. The initiating module sends a *begin transaction* message, then control is handed over to the physical layer, while both modules are in the busy state. Once the receiving module has firmly established by its sensors that the object is completely transferred to the receiving module, it triggers the *end transaction* function and the sending module sets its state to idle.

VII. Cyber-Physical Processes, Material Handling Areas, and Jobs

The jobs that a material handling system has to perform define the capabilities it must to have in order to satisfy requirements. In the functional design of a material handling system (see [9]), specific areas of the material handling system are associated with specific requirements. The embodiment

design then describes how these requirements are implemented. This requires the selection of the suitable modules (for storage, movement, packaging, etc.) and the arrangement and connection of these modules, thus creating an area of the material handling system associated with providing the required operations for that job.

In the design process, a requirement is mapped onto an area of the material handling system and thus associated with the process for which that area is designed. We assume that the description of a job contains the required processes.

At a fundamental level, material flow is movement of objects between two places of rest, which we call *buffers*. In the language of modules, a buffer is a collection of holders. With this view of material handling movement, processes such as merge, split, sort, sequence, and time-specific release are simply outcomes of coordinated route planning. For example, cartons on two lengths (modules) of conveyor might flow into a common length of conveyor (itself a module), each with a specific destination. Because routes for each carton are coordinated with logical time, there is no collision of material at the merge point and the process appears to the eye to be a simple "merge." Similarly, sequencing is the task of coordinating routes such that departures through a common module (the exit point) are ordered as required. Sortation assigns to each object a specific destination associated with a criterion, such as all parcels for a specific route being sent out through a specific module.

The existence of a buffer at the beginning and the end of a process implies that the process is decoupled from its predecessors and successors. Therefore it is sufficient to plan and execute the process as a series of physical and cyber functions between two buffers.

A job materializes at a module that represents a boundary of the material handling system. For an outbound job, this module is probably part of the shipping area. After analyzing the requirements of the job, the required areas of the material handling system are identified and a planning process, analogous to the one from module to module is executed on an area level. Then the result is handed over to the areas for the planning from buffer to buffer. In the most simple case, when we store unit loads and retrieve them as unit loads, collect them in a shipping area per truck, and then load the trucks, we would have two processes, one retrieving from the storage location, sortation per truck ramp and buffering there. Once this is done, the next process would take care of retrieving the pallets from the buffer and loading them into the truck.

Let us now assume we have two storage areas, one in a pallet high-bay warehouse and another area for tote storage with shuttles. The customer requests unit loads as well as consolidated pallets with titles, which are shrink wrapped and then consolidated with the unit loads before shipping. The design might result in five areas:

- a unit load warehouse with the process that retrieves pallets and brings them to the incoming buffer of a sortation area,
- a shuttle-system that stores and retrieves totes and brings them to a palletizing area,
- a palletizing area, where the totes are grouped, placed on a pallet and unitized, and then sent to the incoming buffer of the sortation area,
- a sortation area, where incoming pallets are sorted according to destination and then sent to the shipping buffer, and
- a shipping area, where buffered pallets are loaded into the designated truck.

Each of these areas contributes to the customer requirement and jointly fulfills the requirements. The processes are planned and executed sequentially, started and concluded with **start process** and **end process** messages between the areas.

VIII. Conclusions

We believe the framework described in this paper lies beneath, so to speak, other modeling frameworks more focused on design. As such, the cyber and physical functions could be viewed as an operating system for decentralized control in material handling.

Given our framework, many interesting research questions might be addressed. For example, the communications burden in a decentralized system is frequently a concern, especially when objects are attempting to find shortest paths among several alternatives at the same time. What approach obtains the best operational performance subject to constraints on the number of messages passed? Is it better to issue parallel requests or better to inquire for resources one by one? Other questions could be imagined.

Our framework is intended not just for modeling and analysis, but also for control of real devices in decentralized or Industry 4.0 environments. It is our hope that the framework could become a starting point for a common language spoken from the research laboratory to the warehouse floor.

IX. Acknowledgment

The authors are thankful for the generous support of Toyota Material Handling North America. We also thank our students Hannah Bachus, Benedikt Fuß, Gang Hao, and Allison Lloyd for their help.

References

[1] L. McGinnis, M. Schmidt, and D. Spee, "Model based systems engineering and warehouse design," in *Efficiency and Innovation in Logistics*, U. Clausen, M. ten Hompel, and J. F. Meier, Eds. Cham: Springer International Publishing, 2014, pp. 161–178.

[2] K. Furmans, F. Schönung, and K. R. Gue, "Plug-and-work Material Handling Systems," *Progress in Material Handling Research*, no. 1, pp. 132–142, 2010.

[3] S. Biswas and Y. Narahari, "Object Oriented Modeling and Decision Support for Supply Chains," *European Journal of Operational Research*, vol. 153, no. 3, pp. 704 – 726, 2004, eURO Young Scientists. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221702008068

[4] M. Fayez, L. Rabelo, and M. Mollaghasemi, "Ontologies for Supply Chain Simulation Modeling," in *Simulation Conference, 2005 Proceedings of the Winter*. IEEE, 2005, pp. 7–pp.

[5] M. D. Rossetti and H.-T. Chan, "Supply Chain Management Simulation: a Prototype Object-oriented Supply Chain Simulation Framework," in *Proceedings of the 35th conference on Winter simulation: driving innovation*. Winter Simulation Conference, 2003, pp. 1612–1620.

[6] M. Glöckner and A. Ludwig, "Ontological structuring of logistics services," *Proceedings of the International Conference on Web Intelligence - WI '17*, 2017. [Online]. Available: http://dx.doi.org/10.1145/3106426.3106538

[7] L. McGinnis, E. Huang, K. S. Kwon, and V. Ustun, "Ontologies and Simulation: A Practical Approach," *Journal of Simulation*, vol. 5, no. 3, pp. 190–201, 2011.

[8] L. McGinnis, "Integrating Analysis Into a Warehouse Design Workflow," in *Proceedings of the 12th IMHRC in Cincinnati, Ohio, USA*, ser. Progress in Material Handling, MHI. MHI, 2014.

[9] ——, "An Object Oriented and Axiomatic Theory of Warehouse Design," in *Proceedings of the 12th IMHRC in Gardanne, France*, ser. Progress in Material Handling Research,, MHI. MHI, 2012.

[10] S. Abdoli, S. Kara, and B. Kornfeld, "Application of Dynamic Value Stream Mapping in Warehousing Context," *Modern Applied Science*, vol. 11, no. 1, p. 76, Oct 2016. [Online]. Available: http://dx.doi.org/10.5539/mas.v11n1p76

[11] T. Sprock, A. Murrenhoff, and L. F. McGinnis, "A Hierarchical Approach to Warehouse Design," *International Journal of Production Research*, vol. 55, no. 21, pp. 6331–6343, 2017.

[12] T. Sprock and L. F. McGinnis, "Analysis of Functional Architectures for Discrete Event Logistics Systems (DELS)," *Procedia Computer Science*, vol. 44, pp. 517–526, 2015.

[13] Z. Seibold, "Logical Time for Decentralized Control of Material Handling Systems," Ph.D. dissertation, KIT, Karlsruhe Institute of Technology, Karlsruhe, Germany, June 2016 2016.

[14] K. R. Gue, K. Furmans, Z. Seibold, and O. Uludağ, "Gridstore: a puzzle-based storage system with decentralized control," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 429–438, 2014.