The Future of Modeling in Material Handling Systems

Leon F. McGinnis
Keck Virtual Factory Lab
Georgia Tech

June 16, 2010

Abstract
Today, when we talk about "modeling" in the context of material handling systems, invariably we are referring to a mathematical or computational model for analyzing some aspect of the system, such as its throughput rate, response time, cost of ownership, required storage capacity, etc. Creating these kinds of models requires considerable knowledge in at least two domains—the material handling system domain, and the analysis methodology domain—and considerable skill in the "art of modeling" in order to express the former in the terms of the latter. The results can be somewhat *ad hoc*—e.g., two different modelers are likely to create two somewhat different simulation models of exactly the same material handling system. In the past, the situation in software development was very similar, with individual programming experts idiosyncratically driving software development. Over the past twenty years, however, computer scientists and software engineers have created a radically different approach to the process of software "modeling" called Model Driven Architecture, or MDA, that is used to create software for standard applications. The thesis of this paper is that MDA can be adapted to the kind of modeling done to support design and operational decision making in material handling systems. The paper describes MDA technologies in the context of material handling system modeling, and explains how adapting this approach to our context will transform the way we do research and the way material handling systems are analyzed and designed in practice.

1. Introduction

The history of engineering is replete with examples of "phase change" in how engineers perform calculations and, as a consequence, the scale and scope of calculations that are feasible to perform. Napier's work with logarithms in the early 17th century gave impetus to the invention of the first analog computing hardware, the slide rule, which was the workhorse of engineering computation for 300 years. In the mid 1970's, the slide rule was replaced by the digital "pocket calculator", and today, engineering computations are routinely performed using a variety of software solutions ranging from handhelds to laptops to desktops to supercomputers to "cloud computing".

As the hardware has changed, so has the software. Before the slide rule, computations were done long hand. Operating a slide rule required the engineer to have a fundamental understanding of those longhand computations. The widespread adoption of engineering calculators meant that users often did not know how to perform the fundamental calculations. Spreadsheet programs on personal computers provide an easy-to-use front-end to complex series of computations, which users may not really understand. Today, students learn high level languages for setting up computations using software like MatLab™ (http://www.mathworks.com/) or Mathematica™ (http://www.wolfram.com/), or for creating computational versions of mathematical models using simulation or optimization solvers; yet they may have a very limited understanding of how the computations actually are performed.

In each phase change, from long hand to slide rule to calculator to spreadsheet to high-level language, what has changed is the level of abstraction in how engineers think about computations. We've evolved from the abstraction of Arabic numerals and arithmetic to logarithms, to automated evaluation of single functions, to automated evaluation of simple systems of equations to automation of complex iterative computations.

Each of these phase changes has rendered obsolete the prior way of doing engineering computations. Each has enabled engineers to make more precise computations, covering larger and more complex systems. And each has created challenges in engineering education and research, as curricula and research programs developed for the previous computing paradigm were forced into sometimes painful restructuring.

The engineering of material handling systems, or more generally, discrete event logistics systems (DELS), will soon undergo another phase change as the technologies of *model driven architecture* (Miller and Mukerji 2003) enable fundamental changes in the way we think about and analyze these systems. This phase change will result in a higher level of abstraction and is enabled by four specific information and computing technology innovations:

1. Formal systems modeling languages enable the creation of high fidelity representations of large complex systems, based on object oriented modeling principles and using a mathematically formal schema;
2. Model transformation languages enable the translation of a formal model in one language to a formal model in another language, e.g., from a formal model in a system modeling language to a formal model in a simulation language or a formal model in an optimization language
3. Standards for data and information exchange enable the interoperability of applications through high level definitions of data content; and
4. Formal methods for specifying and controlling business process, scientific, and engineering workflows.

Collectively, these four innovations will allow us to change not only how we teach students about discrete event logistics systems, but also our fundamental mental model of our discipline.

Using formal languages, we will develop *domain specific languages* (DSLs) (see (anon 2010a) or (White 2009) for a good introduction to DSLs) for particular domains of application, such as warehousing, supply chains, or perhaps even more specific domains, such as conveyors. Actual or prospective systems will be specified using these DSLs, and model transformation methods will be used to automate the generation of a range of analysis models. Because analysis models will not be hand coded, it will be possible to do many more types and iterations of analysis, and these computational analyses will become an integral part of DELS design. The availability of a wide range of analysis tools will enable the creation of integrated design processes, which will be implemented using engineering workflow managers. The ultimate result will be that both students and practitioners will have ready access to powerful engineering tools for describing, analyzing, and designing material handling systems.

The remainder of the paper is organized as follows: section 2 introduces formal languages and focuses on SysML and its potential for modeling both the structure of DELS and their control; in section 3, the basic concepts of model transformation are introduced, and brief descriptions are given for two applications; section 4 briefly introduces data exchange standards; the basic concepts of workflow management is discussed in section 5, in the context of warehouse design; and finally, in section 6, the impact of these technologies is discussed in the context of DELS design and analysis.

2. Formal Languages and Domain Specific Languages

To say that a language is formal means, in essence, that the language has a mathematical specification, and therefore, statements in this language can be processed algorithmically. OMG has developed a four-layer architecture (anon 2010b) for specifying and using formal languages, which is illustrated in Figure 1 (taken from (Kwon and McGinnis 2010)).

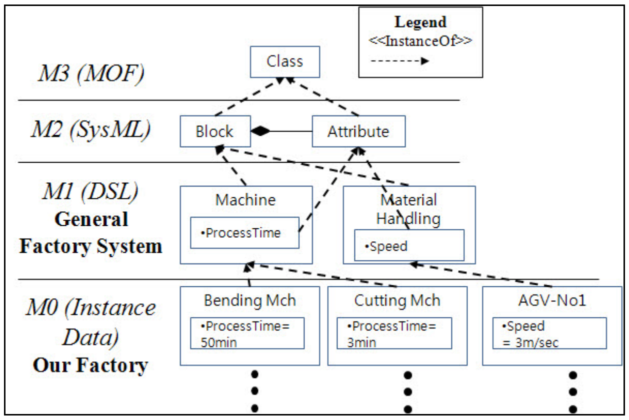


Figure 1. OMG Four-layer Architecture

Level M3 defines the Meta-Object Facility, or MOF, which is a self referencing language that can be used to define other languages. In the illustration, the concept of "class" is defined in M3. Level M2 is where a generic language (meta-model) is defined, such as UML or SysML. In the example, the concept of class is used to define two specific kinds of classes, called blocks and attributes. In addition, a relationship is defined to show that an attribute is a part of, or belongs to, a block.

In level M1, the generic language is used to define a domain specific language, or DSL. In the example of Figure 1, two kinds of blocks are defined—machines and material handling devices—and each of these specialized blocks has associated specialized attributes as parts. Finally, in level M0, the DSL is used to describe a particular situation, or *instance*, in the domain of interest. In figure 1, two types of machines are defined, and one type of material handling device.

SysML™ (http://omgsysml.org/) is a very expressive formal language for describing complex systems. It is both formal and graphical, i.e., the user of SysML can construct models by constructing diagrams to create the formal specification. Figure 2 illustrates the types of diagrams available. An excellent textbook on SysML is (Friedenthal et al. 2008).

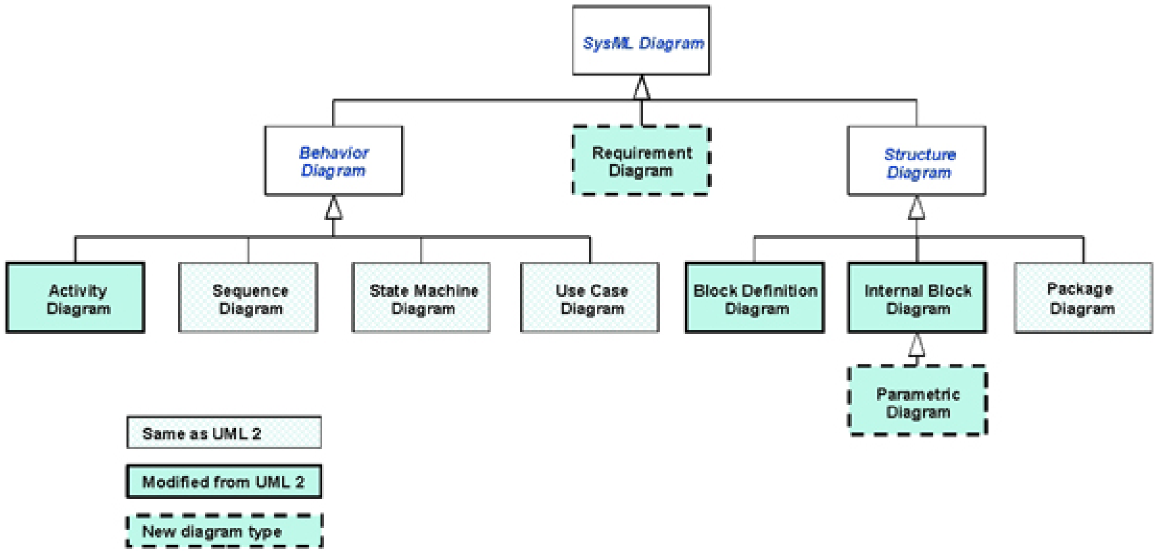


Figure 2. SysML Diagram Taxonomy (from http://omgsysml.org/)

The right side of Figure 2 identifies the diagram types that allow the specification of system structure, by identifying the system components (as "blocks") and by defining the internal structure of blocks (using Internal Block Diagrams or IBDs). Package diagrams provide a mechanism for organizing models, and parametric diagrams provide a mechanism for specifying mathematical relationships defined on or between blocks.

The left side of Figure 2 identifies the diagram types that allow the specification of system behavior. An Activity Diagram is much like a conventional flowchart, and can describe a complex process involving both control flows and object flows. An activity can be associated with a block (called the "classifier behavior" of the block). State machine diagrams are a way to represent both the states of a system component, the transitions between states, and the events that trigger these transitions. Associated with the entry, occupancy, or exit from a state can be an activity or action that involves other blocks.

Sequence diagrams are a way to specify in detail exactly how two or more blocks interact, through message passing. In particular, sequence diagrams capture timing relationships in interactions between or among blocks.

Huang and McGinnis (Huang and McGinnis 2010) argue that if a DELS can be modeled as a finite state machine (FSM)—often considered the most generic modeling paradigm for discrete event systems in general—then it can be modeled using a subset of SysML consisting of the BDD, IBD, State Machine, Activity, and Sequence diagrams. In fact, this subset of SysML can be used to define a domain specific language or DSL. The Keck Virtual Factory Lab at Georgia Tech has developed several such DSLs, for modeling wafer fabs (Huang et al. 2008), supply chains, and electronics manufacturing.

SysML is described as a graphical modeling language because the diagrams are the principle mechanism for constructing models. However, the model itself is actually a type of ontology, which is represented in the so-called "model tree". Any one of the diagrams provides an easy to comprehend view of some portion of the overall model.

Figure 3 is a screen shot of one SysML tool (http://www.magicdraw.com/) showing both the model tree (on the left of the screen) and a window containing a diagram (on the right of the screen).

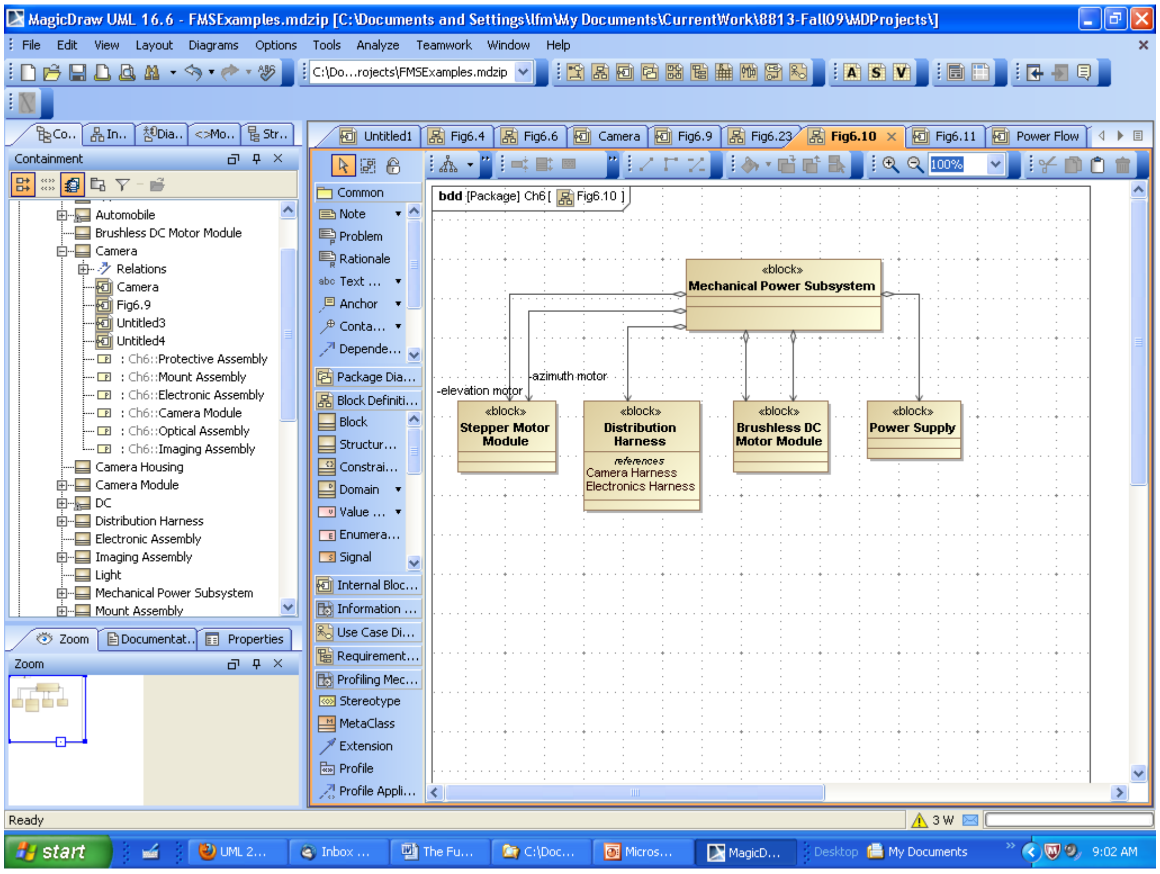


Figure 3. SysML Modeling Tool Screenshot

2.1. An Example

To illustrate the concept of formal language and the OMG four layer architecture, consider a very simple example—a flowshop system in which each workstation has a buffer with finite capacity and may have multiple machines in parallel. Jobs enter the system through the first workstation, and exit the system from the last workstation.

Figure 4 illustrates a simplified domain specific language for this domain, constructed using the TopCased (http://www.topcased.org/) implementation of SysML as the metamodel. In this DSL, the key construct is a “block” which may have “connectors” to other blocks, “ports” though which blocks may flow, and properties. Ports have a property called “direction”, which has a value type called “flow direction”. With this user model, a specific model can be created, with appropriate numbers of workstations, buffer capacities, number of parallel machines, etc.

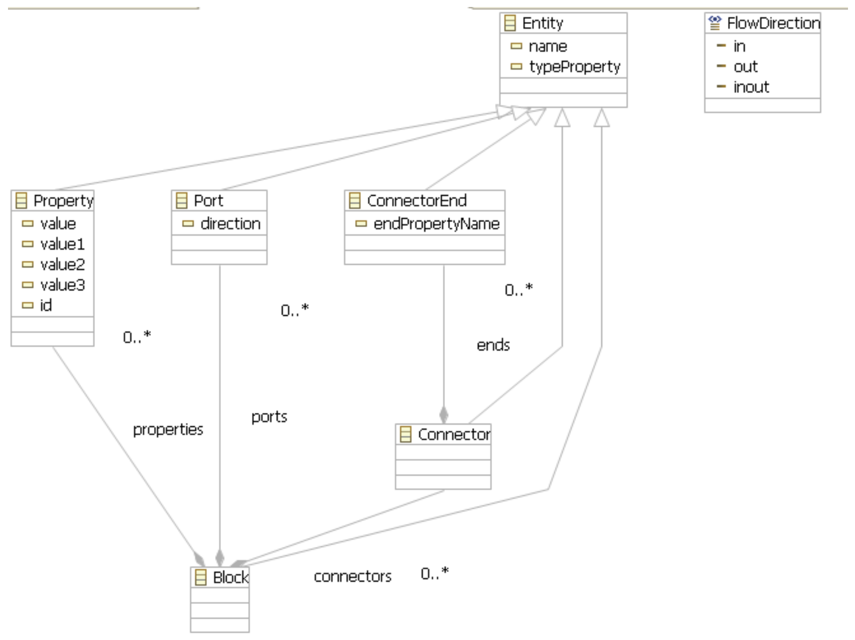


Figure 4. DSL for Flowshop with Parallel Processes
(from (McGinnis and Ustun 2009))

Figure 5 below contains two SysML diagrams from an instance model created with the DSL defined in Figure 4. The first diagram is an IBD of a specific flow shop system with two workstations in series. The block construct is used to define arrival and departure processes and two different workstations. The second diagram is an IBD of the first workstation, showing that it has three machines in parallel, with a single input buffer.

3. Model Transformation

MDA depends in a fundamental way upon the ability to automate model transformation, i.e., to translate a formal model complying with a given metamodel into a different formal model complying with a different metamodel. OMG's QVT™ (http://www.omg.org/spec/QVT/1.1/Beta2/) is a standard supporting such transformation. For example, a formal model created using UML and describing a data-capturing operation might be transformed automatically into Java code. In the context of DELS, a formal model created in SysML and describing a manufacturing process might be automatically transformed into a formal model of a simulation created using AnyLogic™ (http://www.xjtek.com/). The essential requirements enabling this automated transformation are: the appropriate source and target metamodels, a metamodel for transformation, a mapping from the source metamodel to the target metamodel conforming to the transformation metamodel, and a transformation engine that uses the mapping, can read the source model, apply the mapping to the source model, and write the target model. Figure 6 illustrates the

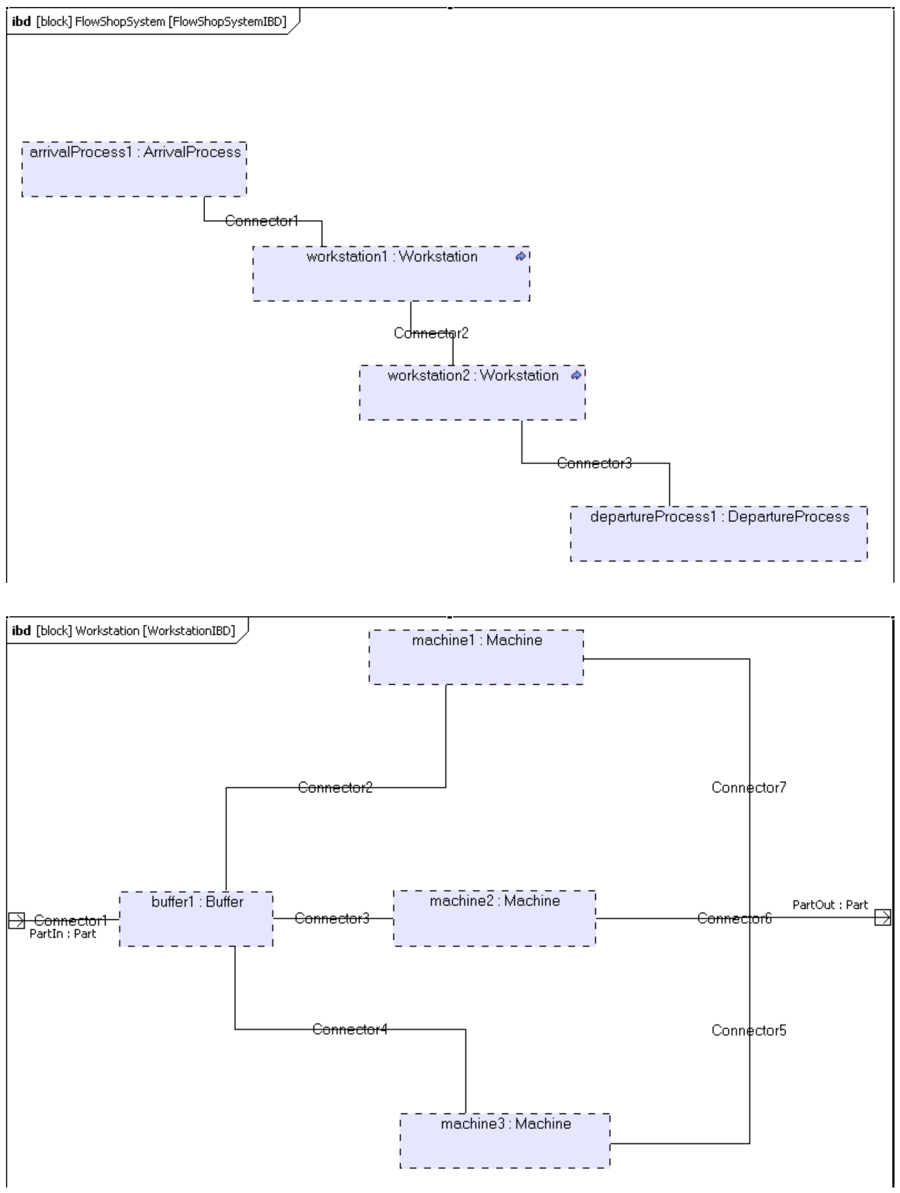

ibd [block] FlowShopSystem [FlowShopSystemIBD]
arrivalProcess1 : ArrivalProcess
Connector1
workstation1 : Workstation
Connector2
workstation2 : Workstation
Connector3
departureProcess1 : DepartureProcess
ibd [block] Workstation [WorkstationIBD]
machine1 : Machine
Connector2
Connector7
Connector1
PartIn : Part
buffer1 : Buffer
Connector3
machine2 : Machine
Connector6
PartOut : Part
Connector4
Connector5
machine3 : Machine

Figure 5. SysML Model of Flowshop System
(from (McGinnis and Ustun 2009))

components and processes for model transformation: MMa is the source metamodel, MMb is the target metamodel, QVT is the transformation metamodel, Ma is the source model, Mb is the target model and Tab is the transformation model. Once Tab is specified, ***any*** source model, Ma can be automatically transformed into the corresponding target model, Mb, using the transformation model Tab and the "Engine."

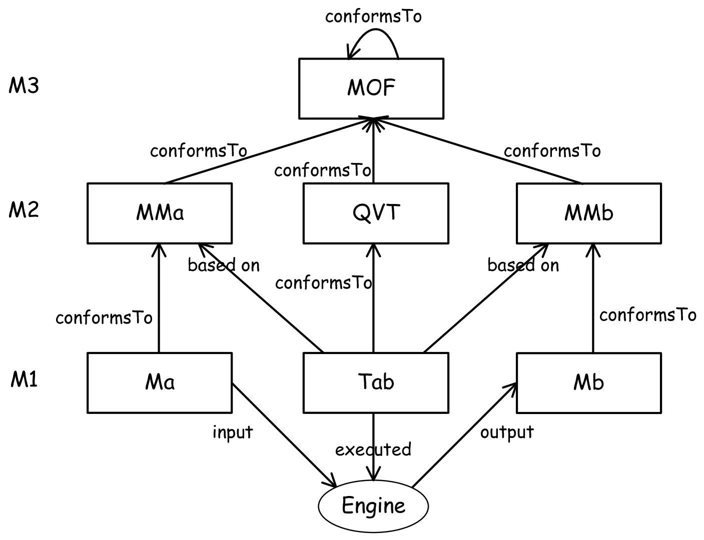


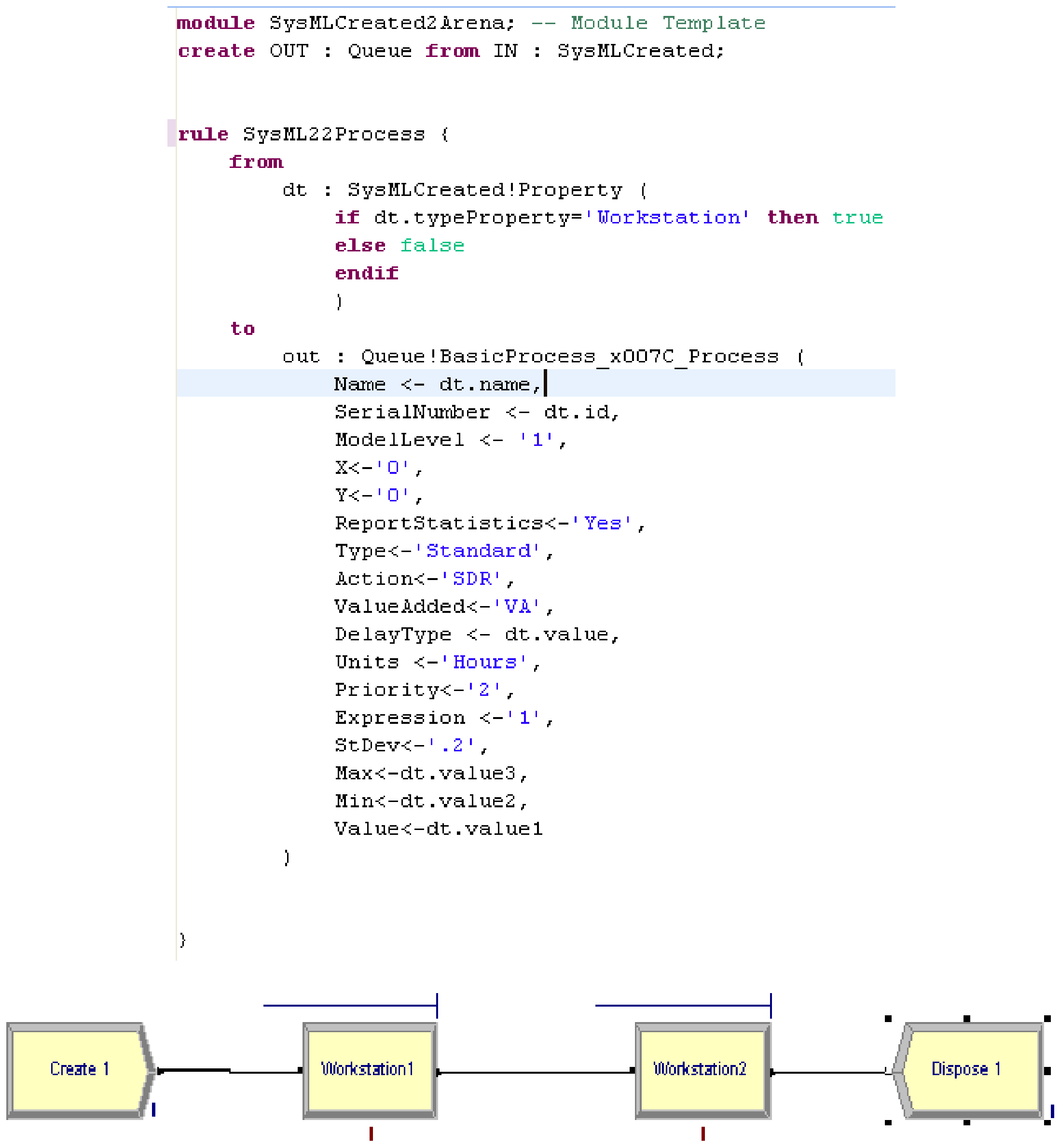
Figure 6. Basic structure of model transformation

Kwon and McGinnis (Kwon and McGinnis 2010) investigate the application of model transformation to model-based systems engineering, and explain why some adaptation is required, relative to the seminal applications in software engineering. In software, the domain specific source model is a description of an application, and this model is transformed into a software code for a specific platform (e.g., Java running on a Linux operating system). Transformation at this level is sufficient, since the instance data to be processed by the software is important only in terms of its format and semantics, not in terms of the specific values of the data; appropriate interfaces for the subject data source can be easily defined.

For systems engineering in general, and DELS in particular, however, the situation is somewhat different. Consider, for example, the creation of a factory simulation model using the MDA approach. Some of the factory description may be in a CAD package, some may be in a spreadsheet, and some in a relational database. The target simulation model will be structured to reflect some of the factory information (e.g., the layout), and will use some of the factory information as input (e.g., process plans and processing times). Thus, transformation from source model(s) to a target model must consider the instance data of the source model.

3.1. Example Transformation

McGinnis and Ustun (McGinnis and Ustun 2009) describe a model transformation for the example presented earlier, i.e., from a SysML model of a specific flowshop to an Arena simulation model. The transformation was complicated by the fact that Arena is not object oriented, and there is no Arena metamodel. This difficulty was overcome by using the Access model import/export capability of Arena. A partial metamodel for Access was developed and the associated Access data schema was reverse engineered in order to establish the necessary mapping from the SysML DSL to Access. The transformation was accomplished using ATL (http://www.atl-pro.com) in the Eclipse domain (http://www.eclipse.org/). Figure 7 shows a portion of the transformation script from ATL and the resulting Arena representation of the flowshop.



```
module SysMLCreated2Arena; -- Module Template
create OUT : Queue from IN : SysMLCreated;


rule SysML22Process {
    from
        dt : SysMLCreated!Property (
            if dt.typeProperty='Workstation' then true
            else false
            endif
            )
    to
        out : Queue!BasicProcess_x007C_Process (
            Name <- dt.name,
            SerialNumber <- dt.id,
            ModelLevel <- '1',
            X<-'0',
            Y<-'0',
            ReportStatistics<-'Yes',
            Type<-'Standard',
            Action<-'SDR',
            ValueAdded<-'VA',
            DelayType <- dt.value,
            Units <-'Hours',
            Priority<-'2',
            Expression <-'1',
            StDev<-'.2',
            Max<-dt.value3,
            Min<-dt.value2,
            Value<-dt.value1
        )


}
```

Figure 7. Example Transformation Script and Simulation Model
(from (McGinnis and Ustun 2009))

4. Data Exchange Standards

There are two types of data exchange standards that are important for the future of DELS modeling and analysis: W3C's XML (http://www.w3.org/XML/) (eXtensible Markup Language), and OMG's XMI (http://www.omg.org/spec/XMI/) (XML Metadata Interchange). The well-known XML is a standard that enables the exchange of data between processes that may use the same data but perhaps not the same format for internally representing the data. For example, XML encoding allows documents to be exchanged between Microsoft Office™ and OpenOffice. In essence, XML associates a tag with each data element, and the tag identifies the semantic meaning of the data element.

Less well known is XMI, which is an OMG standard for the exchange of metadata whose metamodel can be expressed in MOF. The most common use of XMI is for the exchange of UML models, although other model types also can be serialized.

Both XMI and XML are important in the application of the MDA strategy for DELS. XMI is needed in creating target models through model transformation, because the transformation engine must deal with source and target metadata. XML is needed because either or both of the transformation engine and the resulting target models will need to read data from disparate sources.

5. Engineering Workflow Management

The final technology component is the computational implementation of workflow management, i.e., computational processes that manage the sequence and execution of steps in a workflow. The best known examples of workflow management today come from the integration of business process workflows. For example, OMG's Business Process Modeling Notation (BPMN) (http://www.bpmn.org/) and the OASIS Business Process Execution Language (BPEL) (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel) provide, respectively, a standard notation for describing business processes, and an execution platform for executing the description. BPMN is a *domain specific language* used by business analysts to create models of business processes, while BPEL is a solution platform more likely to be used by IT specialists to implement a business process. White [x] describes an example of both, and the mapping (transformation) from BPMN to BPEL.

Workflow management also is a concern in science and engineering. Taverna Workbench (http://www.taverna.org.uk/) is a tool developed to support designing and executing workflows, primarily in the bioinformatics domain. Kepler (https://kepler-project.org/) is a similar tool developed under the Ptolemy project http://ptolemy.eecs.berkeley.edu/) at Berkeley, with a somewhat richer model of computation. In the engineering domain, Model Center (http://www.phoenix-int.com/) supports graphical representation of engineering design processes incorporating commercial-off-the-shelf and proprietary analysis tools along with logic, branching and looping constructs. The resulting workflows can be automated.

At their most abstract or conceptual level, workflows are simply activity networks. However, actually implementing workflows requires attention to the computational details of data transport between processes and the launching, running, and terminating of computational processes that may be hosted on a variety of platforms and distributed globally over the internet.

5.1. Workflow Example

Engineering workflow management presents some challenges that are different from typical business processes. Consider, for example, the workflow associated with designing a warehouse. In the process of designing a warehouse, the designer may want to utilize Access to perform database manipulation and statistical analysis of customer orders, replenishment deliveries, inventories, etc. Later, the designer may wish to optimize the partition of products into families, perhaps using an optimization algorithm, and still later, use a simulation code to evaluate the labor required for the peak month of operation. After each kind of analysis, the designer may either specify some element of the warehouse design (technology selection, product grouping, operational policies, physical arrangement, etc), or change a previous specification. Engineering workflows involve execution of analysis software and interaction with the user to both specify and modify designs.

A conceptual warehouse design workflow, first proposed by (Sharp et al. 2008), is represented in Figure 8 as a SysML activity diagram. In this workflow, the initial step is to "profile" the data describing the warehouse and to specify an initial "function flow network", which specifies the key warehouse functions and how goods flow between the functions. Subsequent steps in the workflow specify additional aspects of the warehouse design, and after each step, there is the possibility of returning to some previous tep in the workflow. Himmeroeder (Himmeroeder 2010) compared this workflow to a structured warehouse design practice followed by the Fraunhofer Institute for Material Flow and Logistics (IML), and found that the resulting warehouse designs were quite similar.

6. Future of MHS Analysis and Design

Today, we see our role, and that of our students as they go into the discipline as researchers or practitioners, as the "modelers", i.e., as the experts who interact with the problem owner to "define the problem" and then apply our special expertise to create a (specific) model to solve the client's (specific) problem. That is an "in-line" view of what we do—we are in between the problem owner and the problem analysis.

In the future, this role will not disappear, but it will not be the routine experience for us or our students. Rather, in the future, our role will be much more an "off line" role, involving two kinds of activities: (1) we work with problem owners to define or refine a formal domain specific language which the problem owners can use to author, edit, and maintain an appropriate description of their problem—***in its own terms***—and we implement the DSL in specific authoring tools; and (2) we create appropriate model transformations which convert the problem owner's problem description into an appropriate computational analysis model, send that model to a commercial-off-the-shelf (COTS) solver, and return appropriate solution information to the problem owner.

Technology enables this phase change, but what will drive the phase change is basic economics. Our current approach to modeling DELS creates very limited learning for the problem owner, although it may allow the modeler to develop a set of skills and specific knowledge that is transferable and, in a limited sense, reusable. From the problem owner's perspective, the current approach is expensive, and the result has a relatively short shelf life. If something changes in the problem, often the modeler must be re-engaged to "update" the model, with attendant delays and costs. As a result, many of our models end up being disposable, they are used once or for a limited time and then discarded. More fundamentally, because our modeling languages (e.g., simulation or optimization) are so far removed from the problem owner's language (i.e., how the problem owner talks about or thinks about the problem) validation has always been a major hurdle.

By the same token, it is incredibly expensive for a single researcher to create a full "solution" for any class of problems, and so today we see virtually no truly system level computational resources in any of our courses on material handling or DELS. As DSLs, model transformation, data exchange standards and workflow managers become the norm, this also will change. The emergence of domain specific languages for significant domains, e.g., warehousing or supply chains, will allow many researchers to contribute to the refinement of the DSL, which will become fundamental to education and practice. A DSL is a natural organizing framework for the wealth of narrow analytic models available in the archival literature, and data interchange standards will enable the interoperability needed to utilize these models collectively. Finally, workflow managers will enable the creation of reusable analysis and design workflows, which can access analysis tools wherever they may be hosted on the internet. Since model transformations, data exchange, and workflows all are based on standards, they will be easily shared within the communities of researchers, educators, and practitioners.

The ultimate result of these innovations will be that our students will experience a much richer analysis and design education, which they will be able to translate directly into practice. They will be able to model and analyze realistic case studies, and because they can experiment with different design approaches, they will begin to develop the intuition about specific DELS domains that is essential for engineering decision making, and that makes them valuable employees and capable entrepreneurs.

This is, fundamentally, an optimistic view of the future of DELS research, education, and practice. It is a view that embraces the potential for dramatic and beneficial change to the status quo. However, realistically, the future described in this view is hostage to the feasibility of adopting and adapting technologies and methods that are not traditional in the field, and perhaps most difficult, the fostering of a community-wide collaboration, based on shared and persistent models and standards.

7. Conclusion

In this paper, I've identified a set of technologies that enable MDA, and described a future in which deploying these technologies fundamentally alters the way we do research in the DELS domain, how we teach DELS, and how DELS are designed and operated. The technologies are not conceptual—they are in routine use in software engineering, and making inroads into systems engineering.

Over the past four years, research in the Keck Virtual Factory Lab at Georgia Tech has been focused on better understanding the potential roles of these technologies, and demonstrating their deployment across a broad spectrum of DELS, including warehousing, semiconductor manufacturing, and global supply chains. We believe these demonstrations are compelling evidence for the feasibility of the MDA approach to DELS modeling and analysis.

However, fully realizing the promise of this approach will require engaging a much larger community of researchers, because there is so much research and development required. Key tasks include: developing and promulgating domain specific languages for significant sub-domains, such as warehousing, discrete parts manufacturing, assembly, distribution systems, and supply chains; developing metamodels for major analysis model types, such as simulation and optimization; developing transformations from domain specific languages to analysis specific models; and developing a range of large scale demonstrations or case studies.

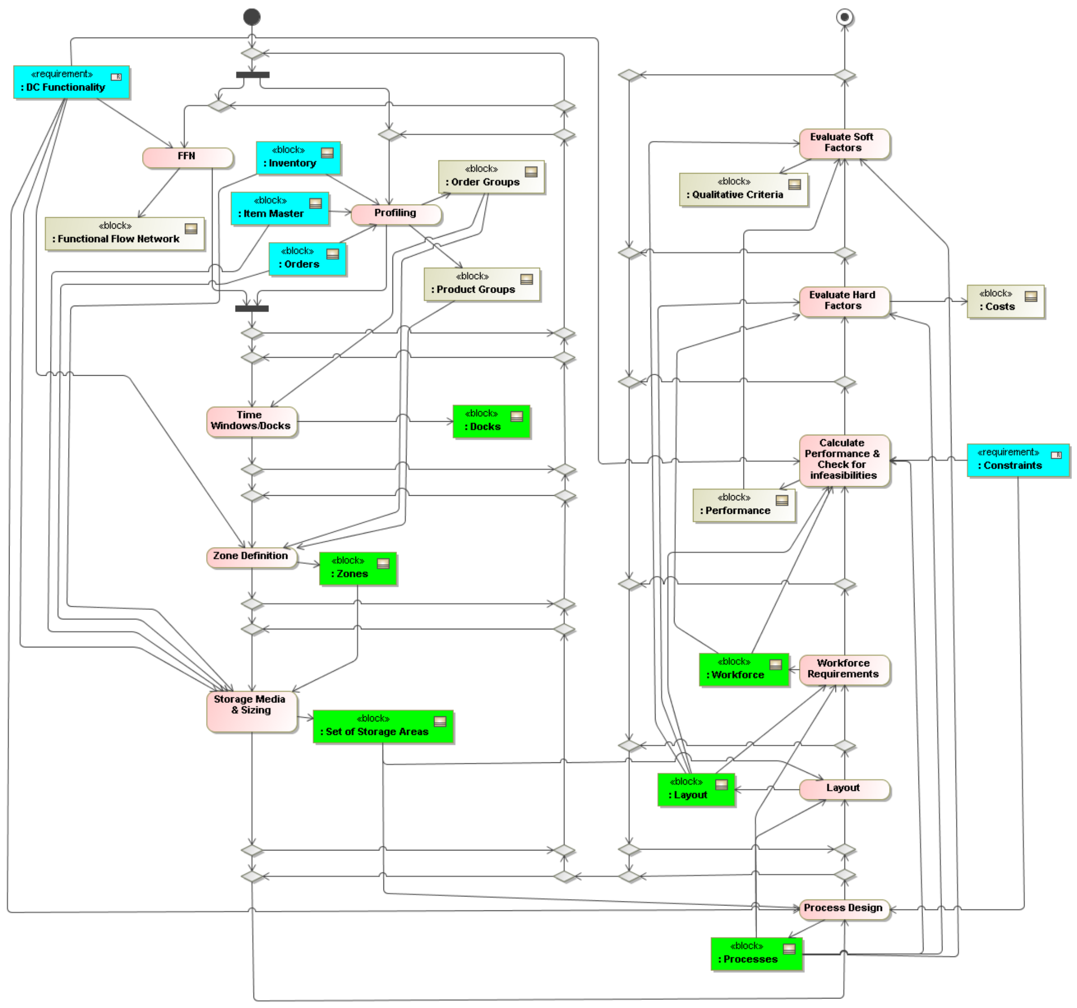


Figure 8. Example Warehouse Design Workflow

8. Acknowledgements

For many years, I have been interested in “reusable” and “reference” models, but it wasn’t until I was introduced to SysML that I found the key technology for creating such models. My colleagues at Georgia Tech—Russell Peak and Chris Paredis—have been mentors and collaborators in the SysML related work in the Keck Virtual Factory Lab. Sandy Friedenthal has both sponsored our research and been a keen promoter. I have been very fortunate to work with students who know much more than I about UML and SysML, specifically Edward Huang, our “go to guy” on UML, and Ky Sang Kwon, whose prior experience and expertise in CAD and product data management has been invaluable, and who is leading our efforts to formalize model transformation.

9. References

anon. (2010a). "Domain-specific language." Wikipedia.
anon. (2010b). "Meta-Object Facility." Wikipedia.
Friedenthal, S., Moore, A., and Steiner, R. (2008). *A Practical Guide to SysML*, Elsevier.
Himmeroeder, B. (2010). "Demonstration and Evaluation of a Warehouse Design Workflow " Project Report, Georgia Tech.
Huang, E., Kwon, K. S., and McGinnis, L. "Toward On-Demand Wafer Fab Simulation using Formal Structure and Behavior Models." *Winter Simulation Conference,* Monterey, CA.
Huang, E., and McGinnis, L. (2010). "On modeling discrete event logistics systems with formal languages." Georgia Institute of Technology.
Kwon, K. S., and McGinnis, L. (2010). "Instance Data Integration in Model-Driven Systems Engineering using Recursive Model Transformation." *Journal of Systems Engineering (submitted).*
McGinnis, L., and Ustun, V. "A simple example of SysML Driven Simulation." *Winter Simulation Conference*, Austin, TX.
Miller, J., and Mukerji, J. (2003). "MDA Guide Version 1.0.1." OMG.
Sharp, G., Goetschalckx, M., and McGinnis, L. (2008). "A systematic warehouse design workflow: focus on critical decisions." Working Paper, Georgia Tech.
White, J. (2009). "Improving Domain-Specific Language Reuse with Software Product Line Techniques." *IEEE Software* 26(4), 47-53.