REACTIVE SCHEDULING SYSTEM IMPLEMENTATIONS USING A SIMULATED SHOP FLOOR

FRANK RIDDICK
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, Maryland USA

ABSTRACT

Developing approaches to integrating manufacturing applications is important for increasing manufacturing productivity. A reactive scheduling system architecture has been developed that defines how to integrate scheduling technology with manufacturing shop floor applications. Prototype implementations based on this architecture have been built using a simulation of a shop floor instead of a real shop floor. A composite representation of commonly available simulator technology is used as a basis for describing the common approach used for several prototype implementations.

Keywords: Scheduling, Discrete-Event Simulation, Manufacturing, Integration

INTRODUCTION

Enhancing manufacturing capability through the introduction of integrated manufacturing applications has proven to be a daunting task. While individual manufacturing applications now provide a vast array of powerful capabilities, they often become "islands of information" within an enterprise. This leads to vast storehouses of important information isolated because they cannot communicate or exchange this information with other applications. Many businesses have attempted to overcome this problem by implementing custom, point-to-point integration solutions. Such solutions have many negative characteristics. They are costly to implement, difficult to maintain, dependent on proprietary technologies, and difficult to integrate with other applications in the enterprise. Increasing manufacturing efficiency and production capability with integrated manufacturing applications requires a more flexible and cost-effective integration approach. Government, academia, hardware and software vendors, and manufacturing application customers must work together to develop such an approach.

At the National Institute of Standards and Technology (NIST), there is a program that seeks to address some of the problems associated with integrating manufacturing applications. The Systems Integration for Manufacturing Applications (SIMA) program is working with U.S. industry to develop potential information exchange and interface protocol standard solutions to manufacturing integration problems, and to foster the adoption of these solutions by manufacturing enterprises [1]. Some of the objectives of this program are to develop models, interfaces, techniques, and prototypes for integrating production system engineering, production management, simulated production facilities, and product data management systems with each other and with other manufacturing applications and support systems. Towards this end, a project is underway that seeks to develop methods for effectively integrating scheduling applications with shop floor operations. In this project, information models, generic interface specifications, and methods for describing shop floor entities and manufacturing processes are being developed. Also, significant events that modify those entities and processes during production are being defined.

The approach taken in this research is to develop a reactive scheduling system testbed that integrates commercially available scheduling technology with other shop floor applications. In a typical manufacturing scenario where a shop floor scheduling application is used, start and finish times for all shop floor operations are scheduled using the scheduling software. Most scheduling application vendors provide user controllable parameters that allow for schedule optimization with respect to minimizing work-in-progress, lateness, etc. The schedule information is then transferred to some form of shop floor controller. Production on the shop floor then proceeds according to the schedule. Once the schedule has been put into operation, events often happen which may "break" the optimization. After such events, production continues in an inefficient manner. This is unfortunate because scheduling applications typically have the capability to produce schedules that can effectively account for such events, but usually there is no mechanism for communicating the necessary information about those events back to the scheduling application.

With reactive scheduling, data about important events that occur during the execution of the schedule by the shop floor is collected. Mechanisms for updating the scheduling application's data with this information are implemented. Once the scheduler's data has been made up-to-date with respect to the current state of shop floor operations, a new schedule can be created that accounts for the production inefficiencies. The essential feature of

systems designed in this way is the ability to produce, and put into production, schedules that are based on the current state of manufacturing operations.

Several prototypes of reactive schedule systems based on these principles have been developed at NIST. These prototypes use a discrete-event simulation of shop floor operations. Through the use of a simulated shop floor, research using the reactive scheduling concept can be carried out with many different shop floor configurations without the expense of building, modifying, and maintaining a dedicated real shop floor.

In this paper, a reactive scheduling system in which a simulator executes manufacturing operations as directed by a schedule is discussed. First, a description of the conceptual entities necessary for understanding shop floor operations is presented. This will include a description of the status messages that define changes in the state of those entities. Next, the conceptual description of a reactive scheduling system is presented. Following that will be a more detailed examination of the system component that encapsulates the shop floor. Next, the modifications necessary to implement the simulated shop floor are presented. The modifications are discussed in terms of a "generalized" simulator, which is a composite description of the capabilities of several of the simulators used in building reactive scheduling system prototypes. The paper concludes with a brief summary.

Factory Entities and Status Messages

The goals of a reactive scheduling system may be characterized as: to produce a feasible initial schedule that is suitable to be used in production; to monitor shop floor status as it changes during the course of production; to use current shop floor status to assess the effectiveness of the schedule; and to react to changes in the shop floor environment by producing an improved schedule based on current shop floor status. A salient feature of this statement of goals is the concept of shop floor status. Shop floor status is the collective state of all of the essential shop floor entities that are being affected by shop floor operations. In a reactive scheduling system, the shop floor entities for which status must be tracked are of four types: loads, resources, buffers, and materials.

Loads represent the groups of parts that are being transformed by shop floor operations into finished products. Each load has attributes whose values define its characteristics. These characteristics include a unique identifier, the planned start and due date, the related product, the current job step being executed, the starting number of pieces, and the current and previous processing state for the load. The processing state for the load, called the load status, defines whether a load has started processing a particular job step, has been held or interrupted because of a production problem, or has completed all of its processing.

Resources represent the machines, operators, tools, and fixtures necessary to carry out production on the shop floor. Each resource has attributes whose values define its characteristics. These characteristics include a unique identifier, the load currently being processed, the amount of time the resource has been in use since its last maintenance event, the current setup, and the current and previous processing state for the resource. The processing state for the resource, called the resource status, defines whether a resource is available, busy, in preventive maintenance, broken, undergoing a setup or tear down process, or not scheduled to do any processing during this work shift.

Buffers represent locations in a production facility where unfinished loads are temporarily stored. Each buffer has attributes that define a unique identifier for the buffer and a list of the loads contained in the buffer.

Materials represent quantities of substances that are not associated with loads but are required for and are consumed during manufacturing operations. Examples of materials are gallons of paint or a quantity of loose 1-inch flat head screws. Each material has attributes that define a unique identifier for the material and the quantity of the material.

Loads, resources, buffers, and materials define a minimal set of factory entities useful for describing the current state of production on the shop floor. They are not intended to exhaustively describe all aspects of factory operations, but to provide a common conceptual model of the important elements of a shop floor that is sufficient to allow the communication of important production related events to the scheduler. A more in-depth discussion of the attributes associated with factory entities can be found in [2].

To facilitate the communication of changes to the factory entities, status messages have been defined. These messages represent the information necessary for the creation, modification, or destruction of each of the different factory entities. Each message contains information that associates it with a particular factory entity and defines a new state for the associated factory entity. Manufacturing applications that use the factory entity definitions as a basis for understanding the state of the shop floor can use similar status messages to keep their representation of shop floor operations up-to-date. For a more in-depth discussion of status messages and how they define state transitions on the factory entities see [3].

While information models for four factory entities and twelve status messages have been defined, the project currently focuses on the load and machine resource factory entities and the six status messages that define transitions on them. The remainder of the paper only addresses issues related to these factory entities and status messages.

The Conceptual Reactive Scheduling System

To accomplish the goals stated in the previous section, the functionality required for such a system has been partitioned into six subsystems called components. Each component has specific responsibilities and coordinates with the other components to accomplish its goals and the goals of the system. Figure 1 shows the components and the inter-component data flow of a reactive scheduling system. Below is a brief discussion of the functional responsibilities of each component.

The Scheduler takes in order information and produces schedules that are executed on the shop floor. Note that in the lexicon of the reactive scheduling system, an order is just a group of one or more loads related by a common order identifier. Any other information that might commonly be associated with order, such as customer or cost related information, is beyond the scope of this discussion. Before producing schedules, the Scheduler is responsible for gathering current information about the state of operations on the shop floor so that it can make informed decisions when producing a new schedule. The output of the scheduling process is a new schedule and a list of orders upon which the schedule is based. Each line of the schedule defines the machine resource, associated load, job step name, and start and completion times for a production step.

The Dispatcher maintains the information necessary for shop floor operations and provides it to the shop floor when requested. Two kinds of information are maintained. The load release list is a list of information related to each load that is to be released for production. Each element of the list contains information necessary for the shop floor to create a load, the time and date that the load should be released to production, and whether the load has been released or has completed all of its processing. The Dispatcher also maintains information

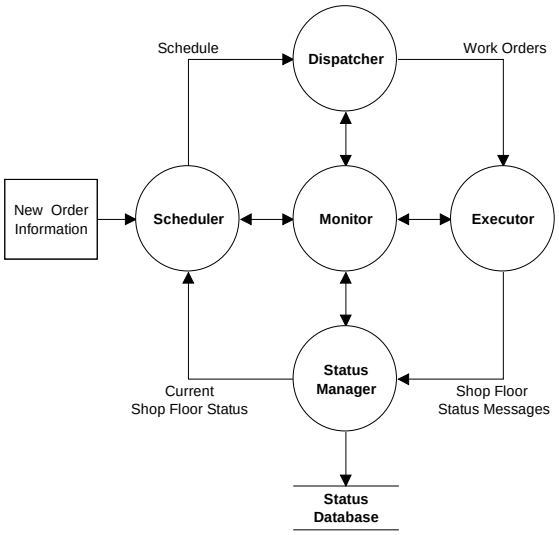

FIGURE 1
REACTIVE SCHEDULING SYSTEM COMPONENTS

for each machine resource called a dispatch list. The dispatch list defines the order in which the jobs associated with a machine resource are to be processed. A job defines one step in the processing of a load and contains a job step, a load identifier, and an order identifier. The dispatch lists information comes from reordering schedule information, primarily by resource and secondarily by start time. The dispatcher is also responsible for merging order list and schedule information in with the current load release and dispatch list information when new information is produced by the Scheduler.

The Executor is the component that represents the manufacturing capabilities of the system. The shop floor is a part of this component. The Executor coordinates with the Dispatcher for job and load release information, executes the manufacturing operations specified by that information, and reports on the status of manufacturing operations to the Status Manager. A detailed discussion of its makeup and operation will be presented in the next section.

The Status Manager is responsible for encapsulating and providing access to a current "snapshot" of the state of operations of the shop floor. It contains a database of information about all of the loads and resources that are being used in production. This database is kept current by receiving the status messages generated by the Executor and merging the new information in with the current data. It also provides an interface for accessing load and resource information.

The Monitor is responsible for analyzing shop floor performance. It gathers information from the Status manager and other components and uses it to determine if the system is operating within acceptable performance parameters. If the system is not operating acceptably, it initiates the creation of a new schedule.

The Conceptual Executor Component

Figure 2 is a Unified Modeling Language (UML) [4] class diagram of the Executor component of the reactive scheduling system. The classes in this diagram are meant to represent typical elements of real manufacturing environments. The two sub-components of the Executor are the *Shop Floor* and the *Shop Data Collector* (the names of the sub-components of the Executor in this and future UML diagrams will be in presented in Italics). The *Shop Data Collector* is used to collect information about the *Shop Floor* and report it to other components in the reactive scheduling system. The *Shop Floor* is the means by which raw materials are turned into finished products. Associated with the *Shop Floor* are *Resources* and *Loads*.

Loads are groups of parts that are transformed into finished products through the processing of *Resources*. *Resources* are the elements of the *Shop Floor* that perform processing on the parts contained in *Loads*. Although they generally represent machines, *Resources* may be used to represent any *Shop Floor* work area that is used to transform *Loads* from an unfinished state of

processing to a more finished state. Each *Resource* must be a part of a *Resource Family*. These are groups of one or more *Resources* with identical capabilities, combined with an *Input Queue* to contain *Loads* awaiting processing and an *Output Queue* to contain *Loads* awaiting transfer to the next *Resource* that is to process them. The important state information about *Resources* and *Loads* is defined by the resource and load entities that were discussed earlier.

The manufacturing capabilities of the *Shop Floor* are defined by the *Shop Floor Configuration*. Contained in this collection of information are descriptions of: how *Resources* are allocated to *Resource Families*; the processing capabilities of each type of *Resource*; and the routing of *Loads* from *Resource Family* to *Resource Family*.

The *Shop Foreman* represents the people and software applications that enable the *Shop Floor* to perform manufacturing operations based on the schedule produced by the Scheduler. Its responsibilities include: communicating with the Dispatcher to get information about load creation and release; communicating with the Dispatcher to get the next job that each *Resource* is to process; and ensuring that *Loads* move from *Resource Family* to *Resource Family* according to the routing specified by the *Shop Floor Configuration*. It also works with the *Shop Data Collector* to communicate the current status of the *Shop Floor* to other components in reactive scheduling system.

In the reactive scheduling system, the elements of the Executor collaborate with the Dispatcher and Status Manager components to carry out production according to the schedule. Each *Resource*, with the aid of the *Shop Foreman*, requests the next job that is to process from the Dispatcher. If the *Load* specified by that job has not arrived at the *Resource's* associated *Input Queue*, the *Resource* waits until that *Load* is available. It does not attempt to process another *Load* because that would undermine the sequencing of jobs defined by the schedule. The *Shop Foreman* coordinates the movement of *Loads* from *Resource* to *Resource* based on a *Product Routing* defined in the *Shop Floor Configuration*. When important production events occur such as the arrival of a *Load* at a *Resource Family*, the completion of processing of a *Load* by a *Resource*, or the breakdown of a *Resource*, the *Shop Data Collector* creates a status message describing the event and sends it to the Status Manager. In this way, production occurs according to the schedule produced by the Scheduler, and production status information is reported so that updated schedules may be produced based on the current state of the shop floor.

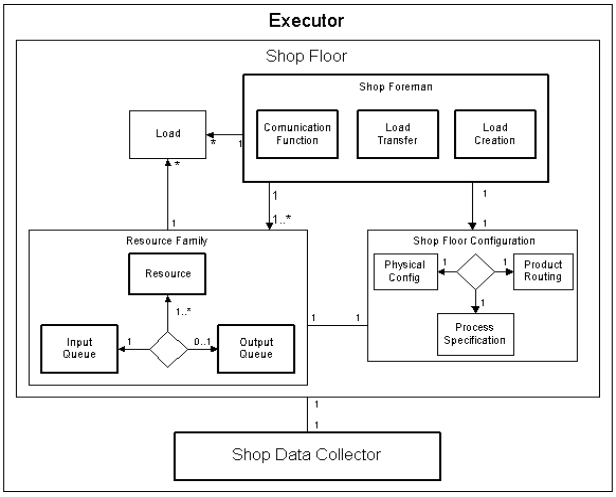

FIGURE 2
CONCEPTUAL VIEW OF THE EXECUTOR

The Executor Implementation Using A Generalized Simulator

The conceptual models of the reactive scheduling system and the Executor component are meant to be a guide to building real systems based on the described concepts. Several prototype implementations of the Executor were done using different commercially available discrete-event simulation tools. Following is a discussion of the Executor implementation based on a "generalized" simulator. The *Generalized Simulator* combines the common functional elements of the commercial simulators that were used in the project prototypes. A detailed discussion of an implementation using a specific simulator can be found in [5].

Figure 3 shows a UML class diagram of an implementation of the Executor. The most prominent feature of this implementation is the *Generalized Simulator*, which has three main sub-components. The *Simulation Engine* provides the capability to perform discrete-event simulations of manufacturing operations. The *Visualization Engine* coordinates with the *Simulation Engine* to provide a 2D or 3D visual representation of the simulated manufacturing operations. In addition, one or more *Integration Mechanisms* are provided to allow for the dynamic introduction of information during simulation execution. *Integration Mechanisms* include the ability to read and write files, communicate over pipes or sockets, make system calls, or execute programs.

To perform its task, the *Simulation Engine* requires a *Simulation Model*. The *Simulation Model* describes: the manufacturing elements that are to be simulated (the products and resources of the shop); their relationships to each other (machine A is connected to machine B); and the standard behaviors that the elements of the simulation should exhibit given specified circumstances (machine C processes part A1 for 10 seconds). To enable the *Generalized Simulator* to act as a component of the Executor, the *Simulation Model* has been extended into what is called the *Enhanced Simulation Model*. The relationship between the *Simulation Model* and the *Enhanced Simulation Model* is described in detail later.

The final element needed for implementing the Executor is the *Communications Manager*. It coordinates with the *Integration Mechanisms* to implement the sending and receiving of messages to and from the other reactive scheduling system components. This collaboration provides the capabilities needed to implement the communication requirements of both the *Shop Foreman* and the *Shop Data Collector* of the conceptual Executor.

The Enhanced Simulation Model

The *Simulation Models* that are typically executed by *Simulation Engines* do not allow for the implementation of the complex behaviors needed by the Executor like status message processing, Dispatcher driven product creation, and *Resource Family* processing of *Loads*. To implement these behaviors, it was necessary to enhance the *Simulation Model*. Figure 4 shows how the *Enhanced Simulation Model* is derived from the *Simulation Model*. An essential characteristic of the *Simulation Model* and the *Enhanced Simulation Model* is that they represent a characterization of the common elements of the input models of several simulation tools. The modifications therefore are characterizations of modifications that have been made to the different simulators represented by *Generalized Simulator*.

A *Simulation Model* has several components. *Product Descriptions* describe the parts that are to be produced by the simulated factory resources. *Resource Descriptions* describe the factory resources that are to be used to define the simulated manufacturing facility and to simulate production. The resources are things like machines, conveyers, pallets, etc. *Logic Descriptions* specify the behaviors of the simulated factory resources. They define how the manufacturing processes are carried out by the simulated resources. They are specified as sequences of pre-defined or user-defined behaviors. Some form of scripting or programming language is supplied to allow for this specification. The manner in which the simulation is initialized may also be modified by user-defined behaviors. *Configuration Data* contains information that relates the *Logic Descriptions* to the *Resource Descriptions* and *Product Descriptions*. Information about how parts are to be routed and how the *Shop Floor* is to be presented by the *Visualization System* are specified here.

The *Enhanced Simulation Model* is a *Simulation Model* in which modifications have been made to each component to enable the *Generalized Simulator* to behave as required by the Executor. To enable the parts defined in the *Product Descriptions* to be used as *Loads*, each part is augmented with *Load Attributes*. To enable the factory resources defined in the *Resource Descriptions* to behave as *Shop Floor Resources* of the Executor, *Resource Attributes* are associated with each resource defined in the *Resource Descriptions*. The *Logic Descriptions* are modified to initialize the *Load Attributes* and *Resource Attributes*, and to keep them updated during simulation execution. In addition, new *Logic Descriptions* were created to define four new kinds of factory resources. *Input Queue* and *Output Queue Resources* coordinate with machine *Resources* to implement *Resource Families*. The *Load Release Resource* implements the subset of the *Shop Foreman's* responsibilities related to creating *Load* representations. It interacts with the Dispatcher to create *Loads* at the appropriate time and to put them into production. The *Transfer Agent Resource* implements the *Load Transfer* function of the *Shop Foreman*. It uses *Product Routing* information to facilitate the movement of *Loads* from *Resource Family* to *Resource Family*. It

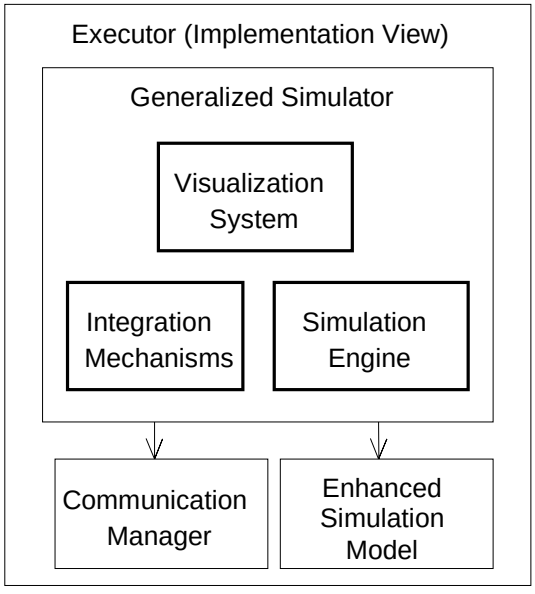

FIGURE 3
IMPLEMENTATION VIEW OF THE EXECUTOR

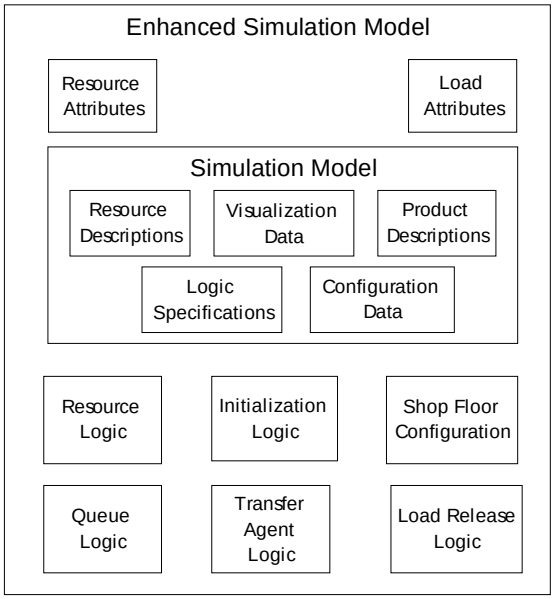

FIGURE 4
THE ENHANCED SIMULATION MODEL

can accomplish this because it is connected to the *Input Queue* and *Output Queue* of all *Resource Families* in the *Shop Floor*. The *Product Routing* information is available because the *Simulation Model's Configuration Data* has been augmented with *Shop Floor Configuration* information, and new *Initialization Logic* been added to provide for loading and accessing that information. With the *Enhanced Simulation Model*, *Generalized Simulator*, and *Communication Manager* all of the required functionality of the Conceptual Executor can be implemented.

Conclusion

In this paper, the conceptual architecture for the reactive scheduling system was presented. An approach for implementing the shop floor functionality using a simulated shop floor was described. While the products of this research must be eventually applied to real machines doing real production, the use of a simulated shop floor facilitates a broader examination of the reactive scheduling concept while the research is in its initial stages. The description of the implementation is in terms of a *Generalized Simulator*, which represents the common characteristics of the commercially available discrete-event simulators used to develop prototypes based on this concept and acts as a guide for future implementations.

Acknowledgments

Work described in this paper was sponsored by the U.S. Navy Manufacturing Science and Technology Program and the NIST Systems Integration for Manufacturing Applications (SIMA) Program. The work described was funded by the United States Government and is not subject to copyright.

References

[1] E. Barkmeyer, T. Hopp, M. Pratt, and G. Rinaudot (Editors), *Background Study: Requisite Elements, Rationale, and Technology Overview for the Systems Integration for Manufacturing Applications (SIMA) Program, NISTIR 5662*, National Institute of Standards and Technology, Gaithersburg, *MD*, 1995.

[2] A. Jones, F. Riddick, and L. Rabelo, Development Of A Predictive-Reactive Scheduler Using Genetic Algorithms and Simulation-based Scheduling Software, *Advanced Manufacturing Processes, Systems, and Technologies Conference Proceedings*, AMPST96, 1996, 589 - 598.

[3] F. Riddick and A. Loreau, Models For Integrating Scheduling And Shop Floor Data collection Systems, *16th IASTED International Conference on Modelling, Identification and Control, Innsbruck, Austria*, 1997, 276-279

[4] H. Erikson and M. Penker, *UML Toolkit*, (New York; Wiley, 1998)

[5] F. Riddick, *Using Simulation As A Proxy For A Real Shop Floor And Data Collection System, NISTIR 6173*, National Institute of Standards and Technology, Gaithersburg, MD, 1998.