Proceedings of the 2004 Winter Simulation Conference
R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds..

AUTONOMOUS PREDICTIVE-ADAPTIVE SIMULATION FOR OPERATIONS SUPPORT

Peter C. Bosch

Highpoint Software Systems, LLC
S42 W27451 Oak Grove Lane
Waukesha, WI 53189, U.S.A.

Majdi Rajab

M-Solutions, Inc.
15718 Island Grove
Houston, TX 77079, U.S.A.

ABSTRACT

This paper describes a simulation system that monitors operations on a production floor, periodically creating a model of those operations, and running a simulation that predicts the next several shifts' worth of events, providing operators with new predictive analysis capabilities.

As a set of procedures is carried out in the model and the real world, deviations are introduced by the variations between the expected activities and the actual occurrences. More deviations arise from explicit adaptations undertaken by operations staff in response to already-observed anomalies. With each cycle, those deviations are integrated into the model, heuristics are applied to estimate the likely future course of events, and after a simulation run, a new set of predictions is generated from that model, and it compares the new predictions with the last run's predictions. Differences between the pre-loop prediction and the post-loop prediction serve to indicate whether the situation is improving or degrading.

1 INTRODUCTION

Simulation is often used for strategic decision making, because the cost of collecting the necessary data, creating and validating the model, running the simulation, and then interpreting the results can be weighed against the larger benefits that are realized from a strategic decision.

Tactical simulation (simulation of fine-grained operations) has received less attention because the savings to be reaped from a given tactical decision are usually significantly less than for a typical strategic decision, and the costs must still be borne for the creation, configuration and provisioning of the model.

The equalizer in these economics, though, is that there are many more decisions made in day-to-day operations than at the strategic level. If the cost of creation and configuration can be amortized across a large number of such decisions, the expense of creating such a model can be far offset by the multitude of smaller gains resultant from many better-informed decisions.

The challenge that must be met in order to reap a positive financial return from such an effort is two-fold. First is that of developing a modeling system that can integrate with on-line data sources. To avoid the cost of implementing the necessary services, some or most of the data sources should probably already exist, or be a planned upgrade apart from the modeling effort. These data sources should be rich enough to establish the current state of the model (including resources and their current states), the current procedures in progress (and in what stages those procedures currently exist), and to provide a reasonable assessment of anticipated future procedures. Second, the system must be able to use all of that information to build a reasonable model (especially in the face of frequently-changing plans,) run the model, and report on the results of that simulation in a manner that average operators can readily understand, and act upon.

This paper describes such a system, including both the generalized framework we built to solve the abstract problem, and the specific implementations we layered on top of that framework. The whole system runs on top of the HighMAST framework we described in (Bosch 2003).

In section 2, we will describe the thought that went into making the business case for this application, and where else we might see similar justification. In section 3, we present a big-picture representation of the system, including interactions with surrounding subsystems, and the simulation's lifecycle. In sections 4, 5, and 6 we will discuss the surrounding infrastructures we leveraged from the areas of manufacturing recipe modeling, data feeds from execution systems such as automated control systems, inventory management systems, material tracking systems, and the ancillary systems such as scheduling and material inventory systems. In section 7, we will describe the anomalies that can arise to complicate the matter of creating a new tactical model, followed by section 8, where we we will describe the mechanism we used to blend the production plan, the execution system data and the stored manufacturing recipes into a tactical model for simulation. In section 9, we discuss the framework for running the newly-created model. In section 10 we describe how we interpret the data, and in section 11, we describe the benefits we expect to see in the coming months from operations based on this system. Finally, in section 12, we will summarize the system with its inputs, transformations and outputs and describe some of the future efforts and feature growth we expect in this system.

2 THE BUSINESS CASE

There are three key aspects of a decision to undertake a project of this nature. These are (1) political environment, (2) technical achievability and (3) expected value.

The question of political environment is one not to be taken lightly, although we will not delve deeply into the subject here. An organization should be at least tolerant of change, with a majority of key stakeholders on board with the remaining two criteria. Project management must be politically skilled.

Technical achievability hinges primarily on there being existing systems in place, or planned, that can be interfaced to and depended upon for *reasonably* clean data feeds indicating current system status. In a manufacturing operation, this would imply a networked execution system, as well as probably a networked inventory control and/or material handling system and at least an electronic production schedule. In a hospital operations implementation, it would imply a networked bed control system, as well as perhaps an admissions and equipment management system. In addition, standard operating procedures need to be definable in a manner such that they can be used piecemeal to synthesize a model that may consist of many already-running partial procedures. This enables us to use a single recipe as not only a template for an entire batch or procedure, but as a source for many mini-templates for reordered and repeated executions of subparts of the whole. We have found that for a manufacturing application, at least, a hierarchical task graph is well suited to this requirement.

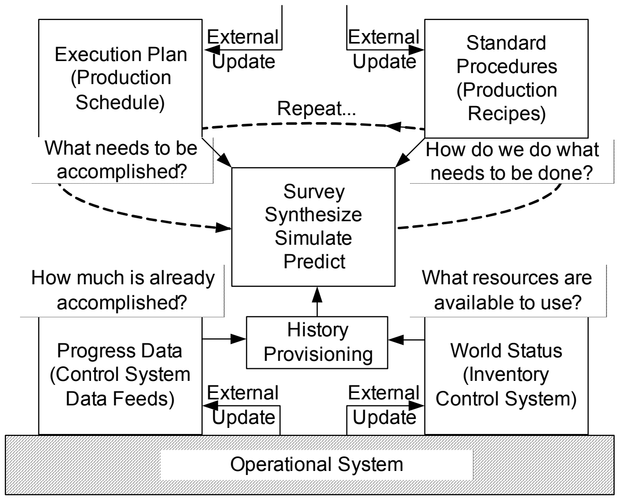
Against the cost of implementing this project we expect to see primary benefits in the areas of increased shift productivity, increased product quality, improved planning, and a central point of connection for a range of tools that will serve to decrease the level of chaos inherent to a large and busy manufacturing floor. See section 11, Expected Benefits, for a more in-depth treatment of these benefits.

There is a class of operational areas that are well-suited to this approach. These are the areas in which similar, but not identical, procedures are executed many times over, most interestingly, with overlapping effects in such areas as equipment and material utilization. These procedures need to have a scope that is large enough that an average operator either cannot easily visualize the effects of local decisions (such as allocating a piece of equipment in a large manufacturing operation) or does not have all of the data necessary to support an optimal decision (such as supporting staffing or procedural decisions in the context of a large hospital.)

Current systems in many of these areas are characterized by reliance on a few “expert” operators or analysts having enough “horse sense” to (often orthogonally to complex documented procedures) do the right thing most of the time. The primary pitfall to this reliance is the failure to realistically institutionalize knowledge and situational awareness & monitoring. The opportunity cost of not being able to move these experts up the food chain to higher-level decision making, whether tactical or operational, is high, and the impact of losing such an expert (to a variety of loss scenarios) is even higher. Furthermore, putting in place a system that encourages thorough monitoring of an operational process is very likely to move the host organization strongly in the direction of higher process discipline, visibility, quality and controllability.

3 THE BIG PICTURE

Figure 1 below shows a representation of a generic system of data and behavioral sources surrounding the core engine, and closely parallels the structures of our manufacturing implementation as well as our healthcare prototype.



Execution Plan (Production Schedule)
External Update
External Update
Standard Procedures (Production Recipes)
Repeat...
What needs to be accomplished?
How do we do what needs to be done?
Survey Synthesize Simulate Predict
How much is already accomplished?
What resources are available to use?
History Provisioning
Progress Data (Control System Data Feeds)
World Status (Inventory Control System)
External Update
External Update
Operational System

Figure 1 : System Topology

The central block, annotated “Survey / Synthesize / Simulate / Predict” reads the production needs, current operational progress and resource availability as well as a library of standard manufacturing procedures. From these, it constructs a model representing the current state of the operational system. It then performs one or more simulations using that model (enabling a Monte Carlo analysis,) and issues predictions on upcoming behaviors, conditions or perhaps limitations. A rules engine may be used to issue recommendations for corrective actions. This cycle is repeated as frequently as necessary to present meaningfully changing, but still timely, updates in predictions. In the current implementation of our production system (large, site-wide pharmaceutical manufacturing operations), we perform 10-15 cycles per hour, although the system performance would support updates at ten to twenty times this rate.

The system runs on a server, displays in a control room, and is monitored by shift supervisors. The display summarizes the activities and their timing or relationship to the critical path, as well as predicting material outages, equipment bottlenecks and staff tasking.

4 RECIPE MODELING

A campaign is a sequence of batches whose aim is to produce a desired quantity of a given product. Generally, a batch runs according to a given recipe, though recipes may change in subtle ways during the execution of a single batch. We represent a recipe as a hierarchical task graph, with the recipe as the root-level task, registered with the model's executive to be commenced at a given time, or per a given dependency, probably with another batch. The units, abstractly representing equipment, act as containers for equipment task sequences, and are represented in parallel under the recipe. The equipments' task sequences are in parallel under the unit, with the tasks in sequence under each unit. Tasks may be hierarchical, with several subtasks required, perhaps, in order to accomplish a super task. See **Figure 2** below for a representation of this structure.

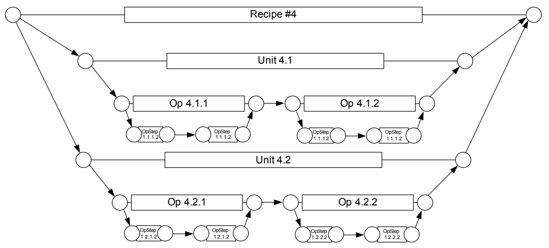


Figure 2 : Recipe Structure

Synchronization and sequencing are handled by special constructs that are part of the task graph, and may be applied to units, tasks or subtasks. Start and finish timing for any task, if known, can be attached to that task. Since the recipe is represented as a directed acyclic graph, it can be analyzed for critical path and related timing data. Loops and branches are unrolled, so that the analysis may still be performed.

In theory, each recipe can host multiple simultaneous batches (execution instances), since the recipe is stateless and the batch is the container of state (Gamma 1995). We employ this approach in the scheduling and model design aspects of the application suite, but chose not to do so in the shift view application itself. This is because in shift view, a given model may contain multiple variations in recipe structure within the same campaign, and even within the same batch. It was simpler to provide each batch with its own recipe, when there was the possibility of each being different anyway.

A model may contain many batches running against many recipes, as well as many resources and resource pools, material types, reaction definitions and pieces of equipment, all of which, together, represent the states of the plant, the product and the process. See **Figure 3** below for details.

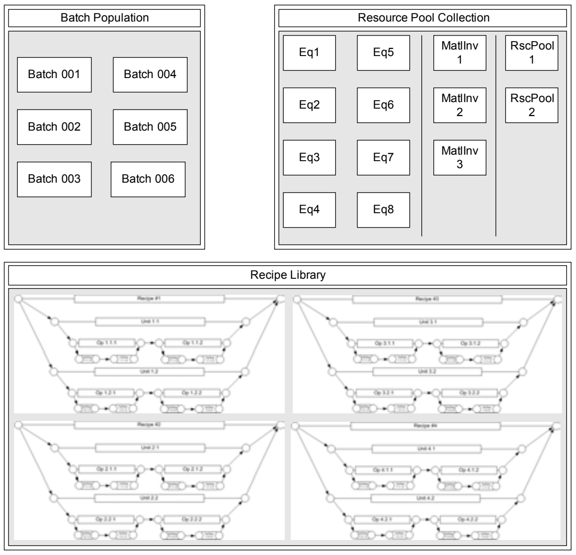


Figure 3 : Principal Model Participants

Individual behavioral objects perform resource allocation & release, material transfers, enable chemistry and thermodynamics calculations, and impose certain dictated or calculated time delays. Tasks of different types orchestrate these behaviors according to differing protocols. There are currently four different types of tasks, all built on the HighMAST task graph object model.

5 MANUFACTURING CONTROL SYSTEMS

In the last 10 years we have witnessed major growth in operations and execution management systems in many industries. This has had the positive effect of moving execution data from the paper to the digital medium. **Figure 4,** below shows a typical taxonomy of such digital information.

Furthermore, several market drivers have convinced many industries to standardize the format of their execution data through consortiums. Example market drivers are the need to exchange execution data between multi-vendor systems and the need to consolidate execution data from multiple vendors to feed higher level management systems. Examples of such standards are ISA's S88 and S95 [ISA SP88] which provide general formats for batch manufacturing production records. The standards define abstract data entities such as campaigns, batches, procedures, operations, materials and equipment.

In this paper we will examine execution data that is typical for a chemical multi-product batch manufacturing operation.

A tactical simulation solution must be able to consume two main types of data from such a system: 1) Batch Recipe Data; and 2) History Data. The following is an example diagram that shows the major components of a recipe:

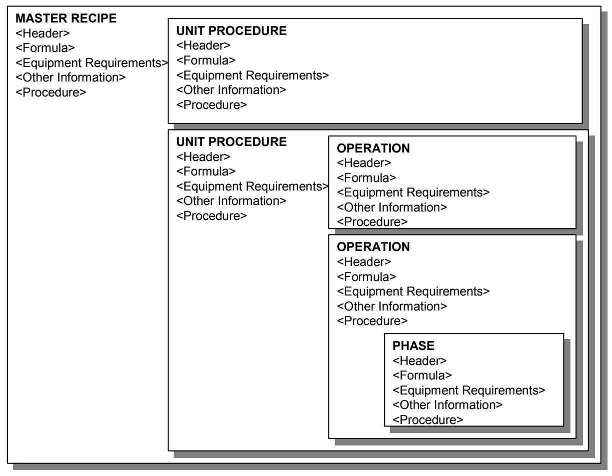


Figure 4 : Execution System Data Taxonomy

Execution System history data can be divided into six general categories – examples of the contents and format of each data record follow:

a) Equipment Measurements

TimeStamp	EquipmentTag	Value	Quality

b) Batch Events

TimeStamp	RecipeID	UnitID	UserID
	ProcedureID:#	OperationID:#	
	PhaseID:#		

where # is a sequence number

c) Control System Alarms

TimeStamp	EquipmentTag	AlarmString

d) User Actions

TimeStamp	EquipmentTag	ActionString	UserID

e) Equipment Failure Events

TimeStamp	EquipmentTag	FailureID	FailureString

f) Material Records

TimeStamp	LotID	MaterialID	UserID
	TargetEquipmentID	Quantity	

Each of these records is expressed differently with different standards, lately, typically in a text stream containing snippets of XML such as Batch Markup Language (World Batch Forum 2003). Legacy systems are typically constructed of proprietarily-coded packet streams.

6 ANCILLARY SYSTEMS

The systems deemed ancillary to this project are the scheduling and material inventory systems. They are helpful to the goals of the project, but neither are they critical, nor do they have a great effect upon the architecture or execution path of this project.

The scheduling system is a portion of a large application that is used to create a master production plan, with information on campaigns, batches, and material consumption & production. The schedule also contains macroscopic expectations of start and finish times – that is, such times are specified to the batch level. This system is considered ancillary because it stores the schedule to a database, and our provisioning infrastructure reads from that database – therefore we need not integrate with the scheduling system, and for our purposes, anything from a spreadsheet to a commercial ERP package could serve our needs with respect to scheduling.

The material inventory system is considered ancillary because while it is not strictly a part of the execution system, its functionality is a part of the execution system data feed. The data feed, where possible, decorates the control events with the material quantities it can read from the material inventory system. Without the material inventory systems, we might lose some of the benefits of predictive/adaptive simulation, but we would still be able to achieve most of its goals.

7 COMPLICATING FACTORS

Recall from **Figure 2** that a recipe is a hierarchical structure of tasks that take time, but some of which may proceed in parallel. A schedule has a similar structure, as seen in **Figure 5** below (we represent the detail under one campaign only, for the sake of the diagram's size.) This similarity allows us to represent the entire schedule as a hierarchical structure with entities ranging from "Schedule" at

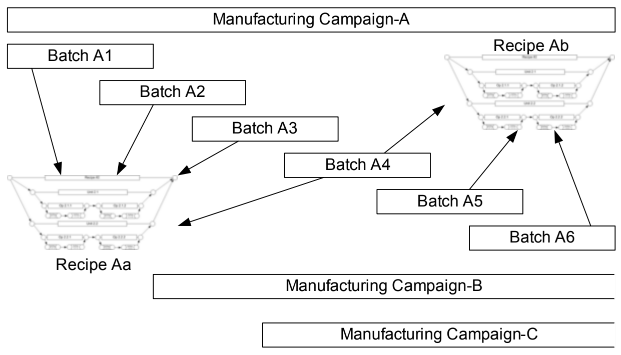


Figure 5 : Schedule Structure

the root, all the way down to "operation steps" at the bottom (leaf nodes). Each entity has a start and finish time (milestone). When a milestone is past (i.e. it has already occurred in the plant, and we have control data for either it or a successor,) the time represents a historically observed or inferred time. If it is a future milestone, the time represents a predicted time.

We must track these data points from one run of the simulation to the next. We chose to represent this as two separate hierarchies, mirroring the schedule structure. In these two hierarchies, the planned and observed structures exist separately. In summary, there is a hierarchical structure to a recipe, a hierarchical structure to a schedule (each of which references one or more recipes) and a pair of hierarchical data structures that mirror the schedule structure, for tracking time data.

Figure 5 describes the first, and largest, complicating factor of the system. Since recipes can be modified while a batch is in process (notice batch A.4, which starts out running on recipe Aa, and mid-run, switches to recipe Ab) our builder needed to be able to construct a model using parts of two different recipes in the same batch.

Another complicating factor arises from the fact that we do not know a priori the number of times a looping operation (such as a titration/sample loop) will execute. Therefore we may unexpectedly branch or not branch, and must be able to correctly construct the model in the face of such historical data.

The final, and most intricate requirement is that the model construction must gracefully react to unexpected presence or absence of history data. User domain analysis led to the following rules.

1. If an additional, unexpected instance of a past operation is the last observed historical event, it is to be executed, but considered an aberration. The predicted execution path is to continue with the next operation following the last expected operation.
2. If an unexpected instance of a future operation is the last observed historical event, it is to be executed, and considered an ad-hoc jump forward. The predicted execution path is to continue as though the jump forward were planned. This accommodates the overt jump forward, as well as an apparent (but erroneous) jump forward due to missing data.
3. Expected operations, whether the result of a linear execution, or a loop or branch, simply serve to advance the pointer to the next expected operation.

Note that historical events define the observed execution path – it is only the last observed event (and the state of the expected events list) that drives the predicted future execution path. In the case where a predicted future path is incorrect, two eventualities preside – first, the next update of the system will have more history data, and therefore may more reasonably predict the future, and second, additional rules are relatively easy to inject into the algorithm, so as we observe and learn, we can update the algorithm.

8 MODEL SYNTHESIS

Figure 1 depicts a central box containing the label, "Survey, Synthesize, Simulate, Predict." This section describes the manner in which the system surveys the current state of the system, and synthesizes a model. The model it creates represents a time slice from very recent past to some time in the future, in which the modeled state associated with current time will closely represent that of the real world under study. This allows the user of this model to treat the progression from the current time forward as a prediction of near-term plant conditions.

The problem of model synthesis is at the heart of this system. There is a set of recipes, each of which represents an ideal process for manufacturing a product. There is a stream of control system data that represents what is really going on in one's plant. There is a production plan that represents the way we would like to apply our recipes to a set of production goals. How do we merge them together into a single executable model that can predict with reasonable accuracy, the near-term future activities in the plant? We will examine first the structure of these data, and then the algorithm applied to synthesize the model from them.

The ShiftViewBuilder reads the schedule as a series of milestones. With each starting milestone of a batch, a database is consulted and the nominal start and finish times of each of the constituent tasks in that recipe are read. The milestones are reflected into the schedule with the appropriate offsets contributed by the starting time of the batch in question, such that the result is an interlaced sequence of milestones, each of which refers to its campaign, batch, recipe, unit, and task of origin.

The execution system produces a similar data stream, with three exceptions. First, there is a range of data points also provided that represent physical system parameters such as the mass or temperature of material in a vessel or inventory point. Second, the data stream proceeds only up to the current time, and third, the milestone times represent actual observed times, rather than nominal calculated times.

The third data source used in the generation of a model is the recipe. While the execution of a recipe can involve cycles, recipes are a hierarchical task graph which is a directed acyclic graph. Each node represents a milestone, and each edge represents a task with duration, and that usually affects system state. Cycles are represented as decorations on tasks detailing branch targets & typical branch counts.

Each unit and each task in a recipe can be reconstituted separately, and its relationships reestablished by the ShiftViewBuilder. Relationships are represented by an ordered list of participants' version independent unique identifiers (VIIDs), and a relationship type.

The ShiftViewBuilder runs the following algorithm.

```
// First, process history.
foreach ( IEvent evt in history.Milestones ) {
     SequenceBuilder sb =
                         GetSequenceBuilder(evt);
    sb.Process(evt);
}
```

Each SequenceBuilder is responsible for assembling a sequence of tasks that, along with its relationships, is integrated into the final model. Sequence builders maintain a list of expected tasks, an under-construction recipe sequence (which will eventually be merged into the actual execution model), and the VIID of the recipe that contains the sequence it is modeling. A sequence typically corresponds to a given piece of equipment, and the operations that run on it.

With the GetSequenceBuilder(evt) method above, the ShiftViewBuilder retrieves the SequenceBuilder that corresponds to the recipe and sub-recipe (i.e. sequence) to which the historical event belongs. If the SequenceBuilder does not yet exist, it is created and initialized.

Initialization of a SequenceBuilder involves reading the sequence steps from the stored recipe, creating an empty list to contain executable operations as they are created, and creating a list of expected operations. It is in the reading of the recipe and the creation of this list of expected operations that loops are unrolled and branches speculatively executed. This means that upon completion of model construction, that the model is still a directed acyclic graph, against which critical path analysis can be performed. The list of expected operations is equipped with a cursor, initially pointing at the first element in the list, that represents the next expected operation. This cursor moves as historical events are observed, and is used to predict the future course of events, once no more historical data exist.

Once created and initialized, the SequenceBuilder executes the following code on each event:

```
// - SequenceBuilder.Process(IEvent event) -
if ( IsRelevant(evt) ) {
  if (RecipeChanged(evt)) {
      LoadAndSyncNewRecipe(evt);
  }

  if ( AccordingToPlan(evt) ){
      // History unfolding according to plan.
      m_rsuc.Add(NextPlannedOperation());
      AdvanceNPECursor(evt);
  } else if ( FailedToJump(evt) ){
      m_rsuc.Add(GetOperationForEvent(evt));
      AdvanceNPECursor(evt);
  } else if ( JumpedBackward(evt) ) {
      m_rsuc.Add(GetOperationForEvent(evt));
  } else if ( JumpedForward(evt) ) {
      m_rsuc.Add(GetOperationForEvent(evt));
      AdvanceNPECursor(evt);
  } else {
      // Report an error.
  }
}
```

The SequenceBuilder only reacts to events that contribute to the building of the recipe – this filters out extraneous events such as valve operations, data reads, and other sub-operations that, while relevant to the stream of operations, are below the level of what is modeled in our system.

Once it has determined that an event is relevant, the first thing the SequenceBuilder does is to check the recipe that originated that historical event, and if it has changed from the one that originated the last observed event, loads the new recipe. Loading a new recipe basically consists of deleting events from the expected events list that have not been read from history, and replacing them with the corresponding events from the new recipe.

The recipe sequence under construction always receives a new operation, corresponding to the history event that was just read. The sequence builder then determines if the new history event represents the expected train of execution (the observed event matches the expected event, which is the event under the cursor in the list of expected operations.) If it does, then that cursor is advanced.

Once all history events have been read, the model has been constructed up to the representation of current time in the real world. At this point, each SequenceBuilder is directed to complete its sequence, which involves creating one executable operation at a time from the template operation in the expected operations queue, until no more operations remain. At this point, all sequences contain a mix of observed historical, and best-guess anticipated future, events.

The final step is for the ShiftViewBuilder to reconcile relationships between sequences – these represent transfers, charges, discharges and other similar relationships. They are provisioned into the model as a simple token exchange protocol, where a token is offered and subsequently accepted. The synchrony of the relationship is determined by whether the offering is a blocking operation or a non-blocking one.

9 MODEL EXECUTION

Once a model has been built, the next step is, of course, to execute it. This execution is performed by external control (a portion of the ShiftView application that is outside of our model builder), but the current algorithm controlling that execution is that the model re-runs every five minutes unless there is new historical data since the last synthesis was begun. A synthesis/execution cycle may not be interrupted by new data. This way, a model does not become stale in the presence of no activity (which may signal an issue, if activity was expected), but it also does not thrash in the face of a steady stream of data arriving. This cycle may be throttled by specifying that synthesis may not begin until, for example, at least three minutes after the last cycle began. This algorithm is encapsulated so that it may be easily modified or replaced.

10 DATA INTERPRETATION

After a model has been executed, the next task is to correlate the newly-predicted milestone times with those same times as they were predicted in the last iteration. Since the model is hierarchical, and each element has data associated with it, a tree structure is used to hold timing data. We use two such structures, one to represent the times that we plan (or planned) to reach the milestone, and one for the observed time that we actually did reach the milestone.

After a simulation run, we populate the tree structure with the new data, and record also into the milestone data the change we observed in the time the milestone was reached from the last simulation run to this one.

The two trees are tied together by a binder that allows us to iterate through the trees in time sequence, pulling data from the historical tree, switching to the predicted tree as the model time passes current world time. This construct makes it easy to visualize the entire timeline as one, from both the observed history and the predicted future perspectives.

11 EXPECTED BENEFITS

The primary benefits we expect to see are threefold:

1. Increased shift productivity. Knowing what is coming up ahead of time enables better human and equipment resource planning. Also having shift targets improves accountability of shift personnel to the production goals.
2. Increased quality. Having a mechanism to identify "critical path" tasks in order to focus resources on those tasks enables manufacturing to become more consistent which leads to better quality.
3. Improved planning. Having a mechanism to identify in real-time when a production slow-down has impacted a future delivery date will enable planners to have more time to develop alternative plans. The sooner the scheduling problem is identified the easier and cheaper the problem is to deal with.

An additional class of benefit that parallels, rather than leverages, the simulation aspects of this system is that of providing a point of chaos reduction.

Chaos reduction represents the fact that there is one well-engineered central point of information and connection for shift changeover expectations (the predicted event stream) & notes, staff task assignments (through an electronic work instruction system), and material tracking and a host of other data management systems.

Benefits that could also be realized in a system of this type would come from situations where a what-if decision must be made in response to an exception case. In our healthcare prototype, we are exploring using a set of simulations to explore the cost effectiveness and supportability of each event path associated with a specific class of decisions.

12 SUMMARY AND FURTHER READING

ShiftView represents a new simulation paradigm, wherein a simulation engine, integrated with an execution system, is used to provide a continuously adapting and updating prediction of a near-future train of operations. With this foundation, operations personnel can react to developing situations before they become critical, or sometimes before they even become visible.

ShiftView is built on the HighMAST™ library. This library represents an acknowledgment that simulation development and application development are often one and the same, and takes a significant step toward bringing simulation development up to the state of the art in application development.

For more information on ShiftView or HighMAST™, please visit `<http://www.highpointsoftware.com>`.

REFERENCES

Bosch, Peter. 2003. *Introduction to HighMAST.* In Proceedings of the 2003 Winter Simulation Conference, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1852-1859. Piscataway, New Jersey. Institute of Electrical and Electronics Engineers.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. 1995. *Design Patterns*. Reading, MA: Addison-Wesley.

Instrumentation, Systems & Automation Society. 2002. *ISA-SP88 Batch Record Specification Draft 5* [online]. Available online via `<http://www.isa.org>` (Accessed August 22, 2004).

World Batch Forum. 2003. *Batch Markup Language Version 02* [online]. Available online via `<http://www.wbf.org>` (Accessed August 23, 2004).

AUTHOR BIOGRAPHIES

PETER C. BOSCH is a founder of Highpoint Software Systems, a small and attentive decision-support technology firm in the upper Midwest. He holds a BSEE from the State University of New York, and is a Certified Java Developer and Microsoft Certified Solution Developer. Pete has published numerous technical articles on object-oriented development in these environments. Pete has been designing and building simulations since 1991 for Fortune 100 firms in aerospace, medical imaging, pharmaceutical manufacturing and investment banking. He has been leading large software projects since 1995.

MAJDI RAJAB is a senior associate with Highpoint Software Systems, and the managing director of M-Solutions, Inc, a firm that specializes in several types of software development, including .Net application and component development. Majdi holds a Bachelor's degree from the University of Liverpool, and a Master's degree in Management Science from the University of Texas. Majdi has more than 15 years' experience in software, much of it in the areas of industrial process automation & control and distributed systems architecture.