Model-Based Computing for Design and Control of Reconfigurable Systems

Markus P. J. Fromherz, Daniel G. Bobrow, and Johan de Kleer

■ Complex electro-mechanical products, such as high-end printers and photocopiers, are designed as families, with reusable modules put together in different manufacturable configurations, and the ability to add new modules in the field. The modules are controlled locally by software that must take into account the entire configuration. This poses two problems for the manufacturer. The first is how to make the overall control architecture adapt to, and use productively, the inclusion of particular modules. The second is to decide, at design time, whether a proposed module is a worthwhile addition to the system: will the resulting system perform enough better to outweigh the costs of including the module? This article indicates how the use of qualitative, constraint-based models provides support for solving both of these problems. This has become an accepted part of the practice of Xerox, and the control software is deployed in high-end Xerox printers.

Complex electro-mechanical products are designed as composites of modules controlled by software. Design of these systems involves marketers who try to understand what is needed and would be bought by the market, hardware designers who create new components and combinations that help to satisfy identified needs, and software engineers that create the control code that allows the system to function as envisioned for the market. Current practice is for marketers to produce documents that describe their intuitions; hardware designers working from these informal descriptions to produce prototypes that match the functional requirements described; and then turning these over to software experts to make it all work together.

As products become more and more software enabled, manufacturers of equipment are facing the challenge that the proportion of development resources expended on the software component of a design and implementation is growing exponentially. Let us consider the challenges for the software engineer. The basic programming approach is to write one monolithic piece of control code that is cognizant of every module of the machine. The system controller for a reprographic system is the most challenging piece of software in the machine. In the past, the construction of such controllers has been a complex and labor-intensive task. Experienced software engineers started from the expected standard specifications of documents to be produced on the machine (e.g., all duplex sheets, or one simplex [cover] sheet followed by all duplex sheets, together with sheet sizes, etc.). They then identified a fixed set of operations that would produce sheets according to these specifications on a given configuration. This required analyzing the interactions of machine components during those operations and devised special case rules that would produce optimal schedules for most of the documents expected to be produced (e.g., A4-size sheets, all simplex or all duplex). The outputs of the analysis were detailed flowcharts that dictated which parametric template to use for scheduling under which circumstances.

Component interaction analysis is complex. Mapping this directly to control software leads to code that is difficult to understand, maintain, and extend. Most importantly, this practice of manual analysis and code development results in configuration-specific software. Reprographic machines are following the common trend toward plug-and-play systems, where the customers can buy and put together different machine modules to satisfy different needs. Yet for a machine that is configured by the customer, the system controller has to be itself compositional, something that is almost impossible to provide economically with the traditional approach.

The key breakthrough for a new paradigm is building constraint-based, qualitative, compositional models of system components. When these models are of sufficient fidelity, there is no need to develop specific control algorithms that schedule, monitor, and control the machine in response to tasks presented to it. Instead, the control software in the machine explicitly constructs a plan and monitors its execution in real time. This plan is formally derived from the description of the machine's modules and a description of the task at runtime. This approach has several powerful advantages: (1) The resulting schedules can be nearly optimal for all tasks. (2) The software engineer needs only build models and does not have to write any scheduling code for the entire machine. (3) The machine can be reconfigured in the field. Modules never foreseen at original manufacture time can be added to the system later.

The basic paradigm shift is that the software engineers become model builders and program execution is replaced by planning and constraint satisfaction.

Our goal is not only to increase the productivity of software developers, but also to improve the communication among different subsystem engineers (mechanics, electronics, software, etc.), to ensure the consistency across different engineering tasks (design, control, testing, diagnosis), and to enable automatic, modular configuration of the resulting systems of the controller. A model-based controller perspective supports other activities including the very earliest part of the design process—determining what kinds of modules would be worthwhile producing for the market. Using a model of the profile of jobs that are to be handled by the machine, we can see if adding a new module (say an additional sheet buffer) would increase the productivity of the machine sufficiently to add value to the end-customer. The ability to perform such configuration analyses for hypothetical machines has proven to be of significant value to designers.

The software for control described here has been deployed in Xerox high-end printers for the last five years. When we first started work with the business team, marketing had asked for plug-and-play capability as a sales advantage, but no approach to control such systems was available at the time; our methodology solved this problem for them and improved overall productivity in software development. In addition, Xerox is using this analysis methodology to decide between alternative configurations for future products.

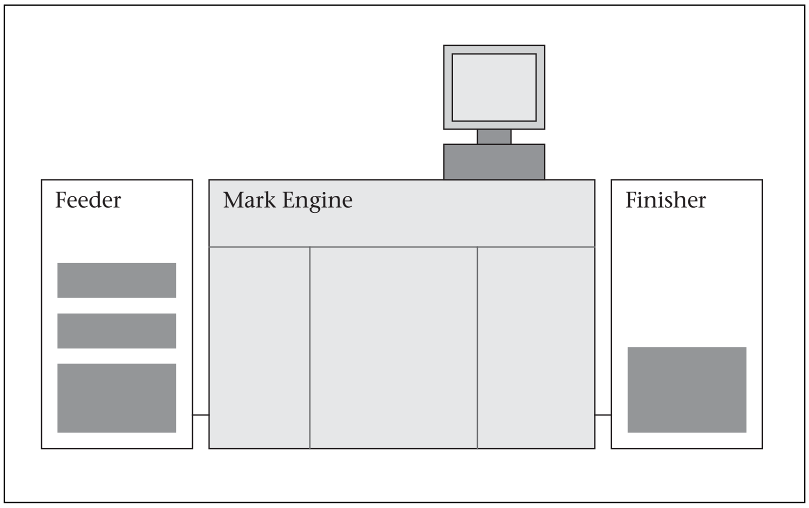

Figure 1. Schematic View of a Modern Reprographic Machine.

The Structure of the Machine

Reprographic machines consist of a source of paper and images, a complex paper path that brings these together at the right time, place and orientation, and finishing components that collate, sort, staple and bind the resulting, marked sheets. The software control system coordinates this activity for not just a single job, but a stream of jobs that come in at random intervals. The paper path is a central element of this structure. Our primary example shows how model-based computing enables flexible generation of control code for moving paper along this path.

Large machines are typically split into modules such as "feeder," "mark engine," and "finisher." Feeders, housing several sheet trays, serve as sheet sources. The mark-engine module processes images and transfers them onto sheets. Finishers sort sheets, collect them in bins, and staple or bind them. High-end configurations typically consist of multiple feeder and finisher modules, connected in series and with a mark engine module in between. Figure 1 shows a typical configuration of a mid-size print engine with three modules: a feeder module with three feed trays, a mark engine module that is able to produce single and double-

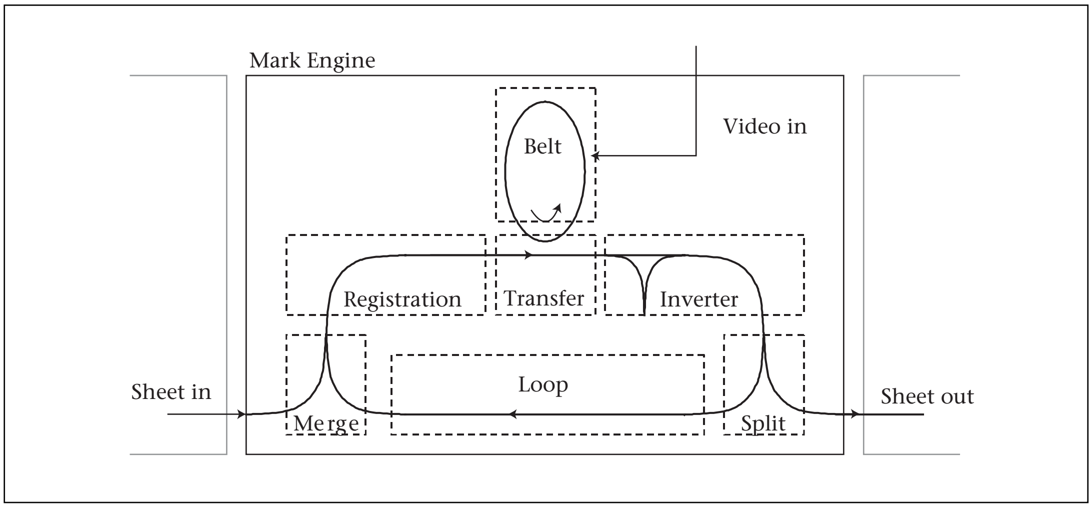

Figure 2. Schematic Decomposition of a Machine's Paper Path.

sided prints (simplex and duplex sheets), and a finisher with two output trays.[1]

Modules themselves consist of multiple components. The mark engine module (figure 2), for example, consists of a photo-receptor belt, an image transfer component, a sheet inverter, path merging and splitting components, and two simple sheet transport components (registration and duplex loop). The transfer component prints an image onto one side of the sheet from a continuously revolving photo-receptor belt onto which images are laid in the form of charged toner particles.

The inverter has two modes of operation (figure 3); because it will be our running example in this article, we explain it in more detail. In one mode of operation, the sheet is guided by the inversion gate G from the input rollers R_{in} down into the inversion rollers R_{inv}; when its trailing edge clears the gate, the sheet is stopped and then moved in reverse direction up and through the output rollers R_{out}. In the other mode, the sheet is moved from the input rollers, guided by the inversion gate in its down position, directly to the output rollers. Thus, the first mode inverts a sheet from face-up to face-down orientation or vice versa, while the second mode bypasses this operation and leaves the sheet's orientation unchanged. The other components of the mark engine module primarily move sheets along the paper path.

In this configuration, a simplex sheet is produced as follows. A sheet is fed from a feeder tray into the mark engine module and moved to the registration component, while a video image is received and laid down onto the continuously revolving photo-receptor belt. As this image and the sheet meet in the transfer component, the image is printed onto the sheet. The sheet is then moved along the bypass path to the output on the right and from there into an output tray. For a duplex sheet, two video images are received. After the first image has been printed onto the sheet as for a simplex sheet, the sheet is inverted in the inverter and moved through the duplex loop back to registration. In the meantime, the second video image is laid down onto the belt so that it can be transferred onto the sheet's back side when the sheet passes through the transfer component. Afterwards, the sheet is inverted again in the inverter (so that it is face up) and moved out.

Control Software Architecture and Control Process

Our new model-based control software architecture is hierarchical. This mirrors the architecture of the machine itself, which is made up of a number of independent modules, with a top-level module controller. Each machine module comes with its own microprocessor, memory, etc., and with software that controls the module's operations. Part of a module controller's job is to integrate the operations of the machine module into complete functions. For instance, a mark engine module controller may export exactly two functions, namely printing a simplex or duplex sheet. When told to execute one of these functions at a certain time, the controller will autonomously start and monitor the necessary operations at the right times. Another part of a module controller's job is to mask local variances in image and sheet processing, in particular timing variances. In other words, under feedback con-

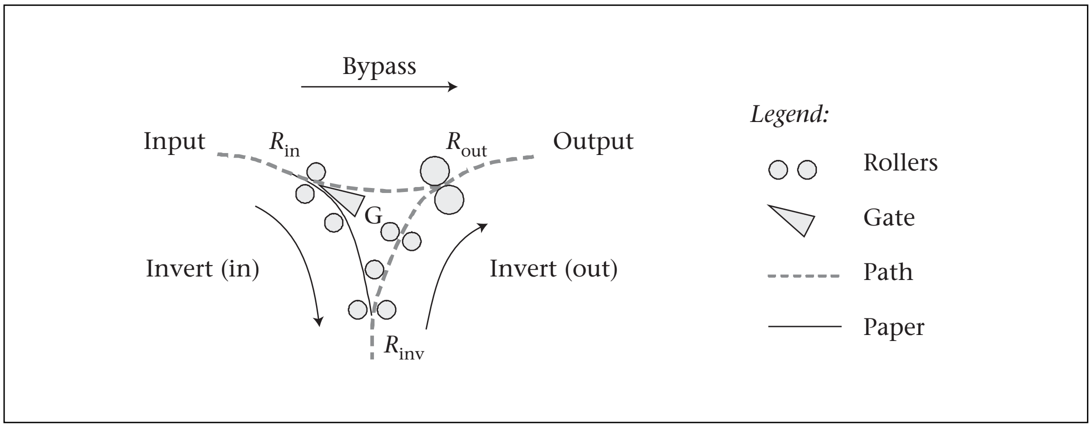

Figure 3. Schematic View of the Inverter in the Paper Path.

trol, a module controller abstracts away many of the local deviations from the expected behavior and thus makes the module's functions predictable.

The system controller breaks the system's functions down into module functions and coordinates the modules in order to produce the desired documents. For example, in order to deliver a set of simplex sheets in a desired output tray, the system controller will tell the feeder, mark engine and finisher modules to feed, print and finish the sheets at certain times such that together a complete document is produced. This print-engine system controller receives a potentially continual stream of document specifications from a variety of sources, such as the network, the scanner, and the fax input. A document specification only describes the desired output, e.g., "five collated, stapled, double-sided copies of a 10-page document." This specification is mapped into a sequence of sheet specifications, with specific images on each side, that must reach the output in a certain order. The system controller's job is to determine the operations that will produce this sequence, while optimizing machine productivity. From the sheet specifications, the system controller first plans the module operations that need to be executed for each sheet, and then schedules these operations. (By planning, we mean the decision of what operations to execute in what order. By scheduling, we mean the decision of when to execute these operations.) Scheduling operations as close together as possible, even interleaving them when feasible, enables the controller to keep the machine as busy as possible, maximizing productivity for the customer. Thus, the system controller has to be able to both generate and commit to schedules incrementally.

Satisfying the Machine Constraints

When planning and scheduling the machine's operations, the controller has choices in selecting and interleaving operations. It tries to optimize certain criteria, such as the start and completion times of a document, while honoring the modules' physical and computational constraints. The transportation and printing of sheets and images is constrained in various ways by the physics of the machine. For example, for a machine to operate properly, sheets are transported in almost continuous movement along the paper path, and the timing of sheets is determined by the lengths and speeds of the transport components; images can be placed on the photo-receptor belt only at certain places (e.g., because a seam in the belt must be avoided); and sheets and images have to be synchronized in the transfer component. The properties of both components and sheets may impose constraints on the execution. For example, an inverter may only be able to invert sheets that don't exceed a certain length.

Furthermore, it would be simple to transport and print one sheet at a time, but productivity can be improved significantly if multiple sheets are printed in tight succession. In this case, the controller has to make sure that sheets never collide. For example, the time it takes a sheet of paper to be inverted in the inverter is longer than the time for it to just pass through. Thus, if a sheet to be inverted at the inverter is followed by a sheet that is not to be inverted, the controller has to schedule a gap between the two sheets in order to avoid having the second sheet "catch up" with the first one and jam. The length of the gap depends both on the inversion time (which is proportional to the length of the inverted sheet) and

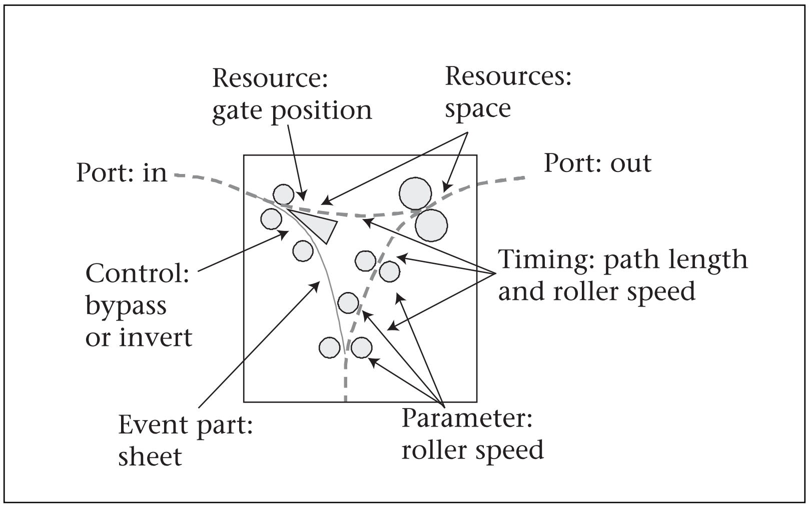

Figure 4. Constraints Are Derived from Physical Model of the Inverter.

A sheet of paper enters through port "in," either going directly across to port out if the resource "gate position" is in the bypass position, or going down if the gate is in the invert position. Other parts of the model include various parameters and timing constraints.

on the time it takes to switch the inverter gate. To facilitate controller development and enable modular machines, we set out to understand the constraints on document production imposed by machine components, to formalize the reasoning that has hitherto been done informally, and to develop algorithms that perform that reasoning in a plug-and-play system. In the remainder of this article, we will develop a domain theory for a model-based version of the print-engine controller.

Modeling

Our domain theory is driven by our primary task, control. By using a declarative modeling approach, however, we are able to extend this theory and thus use our models for other tasks. Conceptually, reprographic machines may be thought of as multi-pass assembly line machines, where parts (e.g., sheets and images) are moved along the assembly line (e.g., paper path, photo-receptor belt), manipulated, and put together, until a desired output is produced. Intuitively, each component is a transducer of timed streams of sheets and images. Thus, it may receive a sheet at a certain time at its input port, transform the sheet as directed by a control command, and produce the result as output at a certain later time at its output port. For example, when directed to invert a sheet, an inverter receives a sheet at its input port, changes its orientation, and forwards the otherwise unchanged sheet to its output port. For a complete machine, producing an output sheet consists of executing a set of component operations that together, if performed in the right sequence, transform input sheets and images into printed output sheets.

Capabilities

Each distinct operation of a component is modeled as a capability. Components can have several capabilities. A component capability is defined by the transformation it performs (e.g., inversion changes a sheet's orientation from face-up to face-down and vice versa); constraints on the features of sheets and images (e.g., the sheet length has to be less than 436 mm for inversion); its timing behavior (e.g., the time it takes to move a sheet from input to output, with inversion, is the bypass path length plus sheet length divided by component speed); and any requirements on internal resources (e.g., only one sheet can be inverted at a time, and inversion requires that the inversion gate is switched to the inverting position).

A component often has multiple capabilities. For example, an inverter can also forward a sheet unchanged through the bypass path (figure 4). Typically, each capability has its own constraints on features and timing of sheets and images, but uses some shared resources. A resource may be the space between transport rollers or on a belt, a gate that can be switched into different positions, or a bin that has a limited capacity. For example, while the inverter's bypass capability does not change the sheet and requires less time to move the sheet from input to output, the sheet still needs the rollers while moving through, and the inversion gate has to be in the bypass position. By modeling rollers and gate as shared resources and use of rollers and gate as allocation constraints for these resources we are able to model the interactions between capabilities in a modular fashion as illustrated in the inverter model in figure 4.

Task

This theory of components and their capabilities is based on a detailed analysis of the requirements of print-engine control. There are two parts to this analysis, conceptualization and representation, discussed in the following.

First, our models need to provide the necessary information to enable the control task. As described, print-engine control consists of two stages, planning and scheduling. Planning consists of identifying the component capabilities that in sequence produce a desired document sheet. The result is a plan of selected component capabilities for one sheet, or simply a sheet plan. Scheduling means finding feasible timings for the capabilities in such a plan, resulting in a sheet schedule, a timed sheet plan, that forms the basis for the control commands (e.g., "feed at time 1500 in tray 1, move at time 3000 in transport 2," etc.). The total set of sheet schedules at a specific time is simply called the schedule. While a sheet plan usually is determined independently for each sheet specification, a schedule may interleave these plans and therefore must honor additional consistency and resource constraints.

For each sheet, the inputs to the planning phase are a specification of the desired features of the output sheet, as well as input ports that provide blank sheets and new images (e.g., feed trays and video input). While the machine's paper and image paths are defined by its components and their connections, planning is somewhat complicated by the fact that there may be multiple alternative paths and loops, and not all components may be able to handle all sheets. In order to find a sheet plan, the planner takes into account each capability's transformation and feature constraints: the plan must correctly produce the desired output sheet from the input sheet and image(s), and none of the feature constraints (e.g., constraints on sheet size) must be violated. For example, given a well-defined input orientation, the sequence of capabilities along the paper path will determine the output orientation of the sheet by composing the changes of the sheet's orientation feature along the way. If there are multiple possible plans for a given output specification, either the planner or the scheduler may decide on which one to choose.

Given such a plan (or multiple plans) for each sheet, together with constraints on the sheets' output order, the scheduler then has to find a time for each selected capability that satisfies all timing constraints and possibly optimizes some objective function (e.g., productivity measure).

Representation

The most important guiding rules have been to start from first principles, and to heed the "no function in structure" principle (deKleer and Brown 1984). While this is common credo in model-based reasoning, it was harder to convince software engineers of the benefits. We illustrate this in two examples.

The engineers originally formulated the inversion constraint as: "if a duplex sheet is followed by a simplex sheet, the simplex sheet has to be fed with a delay that is equal to the inversion time of the duplex sheet." Engineers found it useful to "compile" the system's expected behavior into a constraint between simplex and duplex sheets, simply because traditionally duplex sheets were always inverted while simplex sheets were not. Thinking about it this way made it easier to write the case-based control software. However, newer machines can deliver documents either face-up or face-down, and thus simplex sheets are sometimes inverted, while duplex sheets sometimes are not inverted after the second image transfer. So it seemed, in order to be more general, that the inversion constraint should be stated as a constraint between inverted and non-inverted sheets. Today's machines, however, also process multiple sheet sizes (e.g., an 11-by-14 inch [A3] sheet that is folded as an insert in an 8.5-by-11 inch [A4] magazine). Since inversion time is proportional to sheet length, it turns out that the inversion constraint also holds between two inverted sheets if the first one is longer than the second one. This generalization was "discovered" only when we developed our first inverter model from first principles.

Many constraints, in particular timing constraints like the inversion constraint, intuitively are expressed as constraints between interacting components and/or interacting sheets and images. However, this formulation quickly makes it awkward or even impossible to model machines in a modular way. If the configuration changes only slightly, e.g., if another inverter is added to the paper path (say, before the image transfer), the interaction analysis has to be redone to account for the accumulating delays. What is worse, the inversion constraint may become different for each inverter, depending on the relative position of the inverter in the configuration. Formulating physical constraints as constraints between multiple sheets and images also runs counter to the requirement that the controller be able to schedule sheets incrementally, sheet by sheet, and would further complicate the model as well as the controller implementation. High-end machines, for example, are able to process up to ten different sheet sizes. Analyzing and keeping track of all possible interactions is not an attractive option.

In summary, the original formulation of the inversion constraint was not robust when any of the configuration, the sheet behavior, or the sheet properties changed. Using a modeling approach that derives constraints from the physical structure of devices as shown in figure 4 provides a better basis for reusability and compositionality.

Starting from first principles, we find, for example, that performing the invert capability has a certain duration depending only on the sheet's features, and that the capability requires and competes for certain component resources such as roller space and gate position. Similarly, the bypass capability has a certain duration and competes for the same resources. Neither capability has to mention how it interacts with multiple executions of itself or another capability. Instead, we rely on the constraint systems to manage these interactions.

The abstraction provided by the low-level module controllers makes it possible to represent capability execution as discrete events with predictable durations and transport times. We can further model all velocities as either constant or, when required, as changing discontinuously.

Modeling Language

Abstractly we can think of a model in terms of a set of connected components, each of which has a structural description and a behavioral description. The structural description specifies entry and exit ports, internal resources shared by its capabilities, and the parameters of the model. Behaviorally, each component is modeled by a set of capabilities, where each capability is a distinct operation of the component, typically on a single sheet or image. A capability is described by the tuple *<U, I, O, C>*, where *U* is its control command (naming the capability, with reference time), *I* and *O* are sets of input and output events, and *C* are its feature and timing constraints, C_f and C_t. An event is a triple *<P, S, T>*, where *P* is an entry or exit port, *S* is a sheet or image entering or exiting through *P*, and *T* is the time of entry or exit.

Sheets and images are represented through their features (e.g., length, width, color, orientation, and images). The feature constraints C_f are constraints on and between the sheet's or image's features (combined in *S*), while the timing constraints C_t are constraints on and between the timing variables *T*. C_f includes constraints that represent the capability's transformation. C_t includes resource allocation constraints. A composite configuration is defined as a set of components with connections between their ports. When capabilities of two connected components are selected and composed for a sheet plan, the output event of the first component's selected capability becomes the input event of the second component's selected capability. Thus, both feature and timing constraints are propagated within a sequence of selected capabilities. In particular, the transformational constraints on sheet and image features are accumulated from input to output, providing a complete specification of the output sheet produced by a sheet plan.

This composition of capabilities and propagation of constraints is done during the planning step or can be done ahead of time in a precompilation step. Solving the timing constraints is then part of the scheduling step. Planning here is similar to forward simulation with discrete events and event propagation in other formalisms, in particular discrete event simulation languages (Al-Aomar and Cook 1998; Banks, Carson, and Nelson 1995; Vaidyanathan, Miller, and Park 1998) and Petri Nets (Ghezzi et al. 1991; Peterson 1981). However, these languages are generally restricted to simulation, performance evaluation, and reasoning about specific software properties such as freedom from deadlocks.

Concretely, we provided a modeling language that can be thought of as a declarative specification of the input/output constraints, but one that looked familiar to the engineers who were intended to use it. This language, CDL (Component Description Language), has a syntax that is based on the syntax of C++, with the intention of lowering the barrier for use. CDL has become part of the practice of the engineers in Xerox since 1995 and is an integral part of a generic, reusable machine control tool kit. However, to provide reasoning capabilities with a good semantic basis, we translate this language into a concurrent constraint programming (CCP) language, a well-defined framework (Fromherz, Gupta, and Saraswat 1997) that with the appropriate libraries of constraint solvers supports the simulation, partial evaluation, abduction, and general reasoning that we need. We would like to emphasize that we consider both this higher-level modeling language and the lower-level CCP language that provides its foundation important elements of our approach. They both serve important purposes, one to support human communication and the other to support computer processing.

CDL provides behavioral statements (constraints) akin to those available in a typical CCP language, together with constructs for the specification of structural elements not usually available in a constraint language. As a concrete example, consider again the inverter component (figure 4). Structurally, the inverter has two ports, in and out, through which sheets enter and leave. We model the rollers at entry and exit ports as (unary-capacity) resources r_{in} and r_{out}, because only one sheet is allowed in a port at any one time. We also model the inverter switch as a (state) resource r_{inv} that has to be in either "bypass" or "invert" position while the sheet is moving through. (Note that "bypassing" and "inverting" denote values of state variables.) Finally, the model is parameterized by the length of the path from entry to exit ports (in mm), and by the speed of the rollers (in m/s). (Parameters may be instantiated either when defining an instance of the component, when composing components, or when selecting component capabilities at runtime.)

The inverter's two capabilities are modeled as follows. A sheet *s* to be passed through without inversion (first capability) will enter the component at time *t_in* and exit at time *t_out*. Only sheets of width 285 mm or less can be handled by the inverter. It will take a certain amount of time *d_byp* to move from entry to exit, and the sheet will be in the entry and exit rollers for a duration *d*. These times are determined by the length and speed of the component as well as the length of the sheet. Finally, the two roller resources are busy while the sheet is in the rollers, and the switch resource has to be in bypassing state for the whole time. The component controller associated with the inverter will be instructed to perform this capability with the command *Bypass(t_in)* (name and reference time).

A sheet *s_in* to be inverted (second capability) will be transformed to an output sheet *s_out* that is identical to the input sheet except for its orientation, which is reversed. In addition to the width constraint, sheets are limited to a length of 436 mm. Also, the time between entry and exit increases by the time it takes to invert the sheet, which is proportional to its length. Resource allocations correspond to those of the bypass capability. The corresponding control command is *Invert(t_in)*. The complete CDL model of the inverter is defined as in figure 5.

```
Component Inverter(int length, int speed) {
   EntryPort in;                        // ports
   ExitPort out;
   UnaryResource r_in, r_out;           // declarations
   StateResource r_inv;
   IntVariable t_out, d, d_byp, d_inv;
   FeatureVariable s, s_in, s_out;

   Capability Bypass(IntVariable t_in) {
      in.Input(s, t_in);                // input/output events
      out.Output(s, t_out);
      s.width <= 285;                   // feature constraint
      t_in + d_byp == t_out;            // event time constraints
      d_byp == length/speed;
      d == s.length/speed;
      r_in.Allocate(t_in, d);           // resource constraints
      r_out.Allocate(t_out, d);
      r_inv.Allocate(t_in, d_byp, "bypassing");
   } // Capability Bypass

   Capability Invert(IntVariable t_in) {
      in.Input(s_in, t_in);             // input/output events
      out.Output(s_out, t_out);
      s_in.width <= 285;                // feature constraints
      s_in.length <= 436;
      s_out == s_in except {orientation}; // sheet transformation
      s_in.orientation == 1 - s_out.orientation;
      t_in + d_inv == t_out;            // event time constraints
      d_inv == (length+s_in.length)/speed;
      d == s_in.length/speed;
      r_in.Allocate(t_in, d);           // resource constraints
      r_out.Allocate(t_out, d);
      r_inv.Allocate(t_in, d_inv, "inverting");
   } // Capability Invert
} // Component Inverter
```

Figure 5. The Complete CDL *Model of the Inverter.*

A module is modeled by specifying its components and their connections, and by defining itineraries, the mappings from module commands to component commands. A module integrates the control of its components into higher-level commands: when the control module receives a command, it sends the required component commands to its components. Composite modules currently do not have their own resources (because we did not find a need for it). They may pass parameters through to the components.

Model-Based Applications

The primary problem that pushed us to using modeling was building a system control framework for systems that can be configured out of different modules, some even being installed long after the system has left manufacturing. This control program had to continue to work without change to take into account the particular configuration. At initial power-up after a machine has been physically reconfigured, the machine adjusts its own machine model. The modules pass up their module models to the system controller, where the models are composed to a machine model as explained above. (Some customization, such as instantiation of the speed parameter, may be done at this time.) At run time, given this machine model and the specifications for desired document sheets, the system controller plans the module operations that need to be executed for each sheet, and then schedules these operations.

Planning

Given an output specification defined by feature constraints on a sheet of paper that is to appear at a particular output port, the task is to find one or all plans of machine capabilities that produce the specified output. Given a machine model, and reasoning about capabilities, connections, and the desired output port and specification, a plan is a sequence of modules visited in order that start from a source of sheets of the right type, and provide the appropriate transformations of the features of the sheet. No timing information is included in the plans. Given that the plans depend primarily on a few sheet parameters and the configuration, it is possible to cache the results for various plans, making this a very efficient part of the process.

Scheduling

The scheduler receives the stream of sheet plans, to which it typically adds further constraints, such as precedence constraints between output times (to enforce the correct sheet output order) and between the current (real) time and a plan's input times (to force the plans to be scheduled in the future) (Fromherz and Conley 1997). The scheduler then solves these timing constraints in order to find a feasible schedule, while maximizing productivity.

The scheduler may solve the timing constraints repeatedly in order to generate a schedule incrementally. A sheet plan's constraints may be solved immediately after receiving it, or when several plans are available. Solving the constraints instantiates the reference time variables of the control commands, which are then sent to the module controllers in order to execute the schedule. There is a spectrum of scheduling algorithms and architectures that can be based on this framework. We have implemented algorithms that range from efficient search engines using pre-compiled versions of the constraints (Saraswat et al. 1996), to flexible, reactive optimizers based on a generic constraint solver (Fromherz and Conley 1997) (both running in launched products), to integrated planning and scheduling algorithms. These algorithms further allow for various choices such as the amount of look-ahead in the sequence of selected capabilities, and the timing of when to commit to parts of the incremental schedule (Fromherz and Carlson 1993). Implementation descriptions may be found in a related paper (Fromherz, Saraswat, and Bobrow 1999).

Configuration Analysis

Typically, market studies are used to identify the "average" user's workload distribution, and the system is designed to optimize performance with respect to this average distribution. But individual users are likely to subject the system to workload distributions that vary from the average. Ideally, the design process would be sensitive to this uncertainty about the application context of a product family.

Given a machine model with a set of design variables whose values are unknown (e.g., the velocities and lengths of transport modules), the design and analysis task consists of determining consistent values for these variables such that performance and cost of the resulting design are optimized. We used the model-based techniques described here to develop declarative, multi-use models of machines, with certain parameters open and only constrained by ranges. Using these models, we can analyze design solutions in the face of both exact and qualitative workload distribution knowledge. Given exact knowledge about expected jobs, the model-based scheduler can simply determine and compare the productivity of a set of configurations (often finding counterintuitive solutions, such as in one instance showing how a slower component actually increased overall productivity) (Kapadia and Fromherz 1997).

When given only a qualitative classification for workload distribution (with "frequent" and "infrequent" jobs), we can compute the expected deviations of different configurations from optimal productivity and determine a convex hull that shows which designs are best for which workload distributions (Kapadia and Fromherz 1997). This helps the designer understand how different workload distributions influence the generation of optimal designs. Furthermore, this analysis may be used to avoid commitment to a specific design in the absence of accurate workload distribution until such information becomes available. In particular, one can predetermine a set of designs (machine parameters) with associated distribution boundaries and later, given concrete knowledge about expected jobs, look up the best design and adapt machine parameters accordingly.

Implementation Concerns

Our scheduling task is representative of a class of reactive controllers that have a model of the controlled system and thus can predict the system's behavior. All other information, such as information about documents, is disclosed incrementally. We found three points particularly relevant for the scheduler's constraint solver:

First, the solver constantly alternates between adding constraints, searching for solutions, and committing to partial solutions (e.g., the next sheet to be printed) (Fromherz and Carlson 1993) (which makes memory management, in particular garbage collection in the constraint network, more difficult).

Second, the solver has to distinguish between temporary decisions (for search) and committed decisions (parts of a schedule that are being executed).

Third, the solver has to manage the relation between timing variables and real time (e.g., the timings for any given sheet have to be greater than or equal to the current real time—which is a moving target—until the sheet is being printed).

We used two guiding principles in our solver design. First, all low-level constraint operations (propagation, search, garbage collection) should be as incremental and distributed over time as possible in order to minimize their effects at any one time and to allow for trade-offs between memory and processor usage. Second, the scheduling algorithm should be able to make use of its application knowledge, which, together with a well-defined model of reactive computing, helps the solver manage its resources effectively. We also provide special functions for our reactive scheduling task, such as the ability to constrain variables with respect to the current real time when required and easily remove this constraint when appropriate.

An on-line scheduler, one that makes choices in parallel to job submission and schedule execution, is inherently suboptimal, because it makes decisions based on incomplete information about the future. Besides the implementation of the underlying constraint solver, there are various task-level choices that can affect the efficiency and optimality of a generic on-line scheduler.

A traditional constraint-optimizing scenario is to assert the constraints and then search for a solution for all variables, optimizing an objective function. The scenario of on-line scheduling requires repeatedly finding a solution for only a small subset of the variables, such as those in the sheet plan to be executed next. While this solution should be part of an optimal solution for all variables (all known sheets), we want to keep the other variables open in case more information about future sheets becomes available. In other words, the optimizer is to return a solution for the next sheet only, but guarantee that it is part of a currently optimal solution for all known sheets. We found that this approach, full optimization with minimal commitment, leads to the best overall productivity in an on-line scenario (on average about 5% worse than the theoretical optimum for randomly generated jobs). This idea can be encapsulated in a new enumeration primitive that generates an optimal solution for all variables, but instantiates only those variables that have to be returned for execution of the next sheet plan (Fromherz and Carlson 1993).

The scheduler has been tested successfully for configurations with as many as 12 machine modules, between 2 and 40 capabilities each, for a total of thousands of system capabilities (ways of producing printed sheets). Several characteristics of our application make constraint-based scheduling tractable even for real-time. First, although finding a solution for a constraint network is in general exponential in the number of variables, the constraints for typical configurations and jobs we have encountered so far result in constraint graphs that have a tree structure. Also, our application does not impose deadlines on jobs, i.e., there is always a feasible schedule. Finding an optimal schedule for general sheet sequences (e.g., with mixed simplex and duplex sheets in the same document) is still exponential. For homogeneous sheet sequences and typical machine configurations, however, the first solution is guaranteed to be optimal. In other words, for a set of typical machines, we can always find a solution in polynomial time, and for a set of typical jobs, we can also find the best solution in polynomial time. Finally, we can restrict the complexity of CDL models to a class of "max-closed" constraints, an algebraic closure condition that ensures tractability.

Conclusion

As the demand for more functionality and lower cost increases, the processes used in designing and controlling electro-mechanical systems become both more important and more difficult. We have sketched here an approach to the design, control, and evaluation of complex systems that leverages our work in model-based computing. This methodology starts with declarative, compositional models that enable both qualitative and quantitative reasoning at design and run time. Models describe the local behaviors of components, stating constraints and transformations on parts moving through the components as well as constraints on the timing of resource allocations. Some aspects of the model constraints are qualitative in that they deal with symbolic properties of the sheets as they are transformed by the various components. Others are qualitative by virtue of their use of interval constraints. The final schedule for a particular is of necessity numeric, since precise times must be chosen for each control action. By using this combination of symbolic, qualitative, and numeric constraints, and connecting component models to describe entire machines, we can reason about the behavior of the composite configuration. This approach enables a variety of applications, and model-based product development has become an integral part of practice at Xerox.

Note

1. This example is realistic, but simplified, and this (and all other examples) should not be taken as describing an existing or future product.

References

Al-Aomar, R., and Cook, D. 1998. Modeling at the Machine-Control Level Using Discrete Event Simulation (DES). Paper presented at the 1998 Conference on Winter Simulation, 13–16 December, Washington, D.C.

Banks, J.; Carson, J.; and Nelson, B. 1995. *Discrete-Event System Simulation*. New York: Prentice-Hall.

de Kleer, J., and Brown, J. S. 1984. A Qualitative Physics Based on Confluences. *Artificial Intelligence* 24(1–3): 7–83.

Fromherz, M. P. J., and Carlson, B. 1993. Optimal Incremental and Anytime Scheduling. Paper presented at the Workshop on Constraint Languages/Systems and Their Use in Problem Modeling at ILPS'94, 13–17 November, Ithaca, New York.

Fromherz, M. P. J., and Conley, J. H. 1997. Issues in Reactive Constraint Solving. Paper presented at the Workshop on Concurrent Constraint Programming for Time Critical Applications—COTIC 97, CP'97, 27–28 October, Linz, Austria.

Fromherz, M. P. J.; Gupta, V.; and Saraswat, V. A. 1997. CC—A Generic Framework for Domain Specific Languages. Paper presented at the POPL Workshop on Domain Specific Languages, 18 January, Paris, France.

Fromherz, M. P. J.; Saraswat, V. A.; and Bobrow, D. G. 1999. Model-Based Computing: Developing Flexible Machine Control Software. *Artificial Intelligence* 114(1–2): 157–202.

Ghezzi, C.; Mandrioli, D.; Morasca, S.; and Pezze, M. 1991. A Unified High-Level Petri Net Formalism for Time Critical Systems. *IEEE Transactions on Software Engineering* 17(2): 160–172.

Kapadia, R., and Fromherz, M. P. J. 1997. Design Optimization with Uncertain Application Knowledge. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems 1997—IEA-AIE'97,* eds. D. Potter, M. Ali, and M. Matthews, 421–431. Atlanta, Ga.: Gordon and Breach.

Peterson, J., ed. 1981. *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, N.J.: Prentice-Hall.

Saraswat, V. A.; Bobrow, D. G.; Fromherz, M. P. J.; Lindholm, T. G.; Berlandier, P. C.: and Conley, J. H. 1996. Print sequence scheduling system for duplex printing apparatus. U.S. Patent No. 5,504,568.

Vaidyanathan, B. S.; Miller, D. M.; and Park, Y. H. 1998. Application of Discrete Event in Production Scheduling. Paper presented at the 1998 Conference on Winter Simulation, 13–16 December, Washington, D.C.

Markus Fromherz is a principal scientist in the Systems and Practices Laboratory at the Palo Alto Research Center. His research interests are in the domain of model-based computing, including constraint-based modeling languages, constraint-based planning, scheduling, and control, and model-based design analysis and optimization. He received his Ph.D. in computer science in 1991 from the University of Zurich (Switzerland). Fromherz has published widely in the areas of transformational program development, constraint-based modeling, logic-based reasoning, online scheduling, embedded constraint solving, and design optimization. He also has coauthored more than 30 patents. He is a member of the American Association of Artificial Intelligence.

Daniel G. Bobrow is a research fellow in the Systems and Practices Laboratory at the Palo Alto Research Center and manager of the Scientific and Engineering Reasoning Area. He received his Ph.D. in artificial intelligence from the Massachusetts Institute of Technology. Bobrow has published over 100 technical papers and written and/or edited 13 books on a number of subjects, ranging from programming languages and operating systems, through object-oriented languages and metaobject protocols, to artificial intelligence, model-based programming, computer-supported collaborative work, and systems that bring together social and technical design in a single participatory framework. Bobrow is a fellow of the Association for Computing Machinery and a former president and fellow of the American Association for Artificial Intelligence.

Johan de Kleer is manager of the Systems and Practices Laboratory at the Palo Alto Research Center, an interdisciplinary lab conducting research ranging from social science to robotics. He champions work to develop methodologies and technologies to support knowledge creation, use, and sharing in organizations. He received his Ph.D. in artificial intelligence from the Massachusetts Institute of Technology. De Kleer has published widely in the areas of qualitative physics, model-based reasoning, truth maintenance systems, and knowledge representation and coauthored three books. He received the Computers and Thought Award at the International Joint Conference on Artificial Intelligence in 1987 and is a fellow of the Association for Computing Machinery and a fellow of the American Association for Artificial Intelligence.