Modeling for Parametric System-Level Design Optimization

Ravi Kapadia
Computer Science Department
Vanderbilt University
Nashville, TN 37235
email: ravi@vuse.vanderbilt.edu

Abstract

System-level design refers to a synthesis, analysis, and optimization process which reasons with the system in a holistic manner. We are developing a model-based approach to support parametric system-level design optimization. In this paper, we describe our modeling methodology based on the Environment Relationship net framework (Ghezzi *et al.* 1991) to represent a system for the purpose of design optimization. Specifically, we model a reprographic machine system (e.g., printer, photocopier) whose elements include hardware components and software processes. We discuss the issues that arise in modeling this system and the challenges that remain to be addressed.

Introduction

System-level design refers to a synthesis, analysis, and optimization process which reasons with the system in a holistic manner. The National Science Foundation's report Research Opportunities in Engineering Design (NSF 1996) observes that:

> "It is getting harder to improve system performance from advances in individual disciplines. The number of specialists is increasing, while the number of generalists, capable of doing system integration, is decreasing. The need is for more generalists in product design who can understand the big picture, not just some specialized problems."

To manage the complexity of the design process, designers recursively decompose functional specifications into subfunctions and focus their efforts on solving the subproblems and then, integrating their solutions. In the process, they often develop detailed designs of a subsystem without paying adequate attention to its dependencies with the rest of the system. With the advent of embedded computer systems that integrate hardware components with software computation elements (e.g., digital signal processors, digital printers), the choices available for decomposing functionality are far wider than in traditional systems, and the complexity of the interactions among the subsystems complicate their integration. The interactions between the system's components are often dynamic, i.e., they depend on different tasks performed by the system at run time. Designing the subsystems in isolation without taking into account their dynamic interactions and their inter-dependencies often necessitates backtracking (which lengthens the design cycle) or may lead to a non-optimal solution.

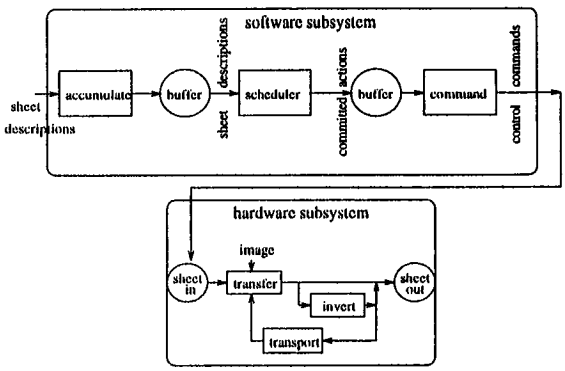


Figure 1: Reprographic machine: an integrated hardware and software system

Our research is targeted at supporting parametric system-level design optimization. Given a configuration of the system, we use a model-based reasoning approach to tune its parameters to optimize specified objectives, such as performance and manufacturing cost. Our methodology for system level parametric design optimization requires a model that incorporates design variables across the system and allows the designer to study and reason about their effects on optimization objectives (Kapadia & Biswas 1999). The model must incorporate not just a representation each subsystem, but also capture the interactions and dependencies that exist across subsystems. In this paper, we describe our approach for modeling a system whose components span multiple domains using the Environment Relationship net framework (Ghezzi *et al.* 1991).

Reprographic machine

As a test bed for our design optimization methodology, we use a digital reprographic machine (e.g., printer, photocopier), which is a computer-controlled electromechanical system that produces documents by manipulating images and sheets of paper. Given a configuration for this system, we are interested in tuning its parameters so that the designed machine optimizes job completion times and manufacturing cost, while meeting specified design constraints.

Fig. 1 shows a schematic description of the hardware and software subsystems of interest. The hardware subsystem is responsible for transporting sheets of paper in the machine. It prints simplex (one-sided) and duplex (two-sided) sheets. A sheet enters the machine through an input port. An image is transferred (or printed) to the sheet as it passes through the *transfer* component. A simplex sheet passes through without inversion on its way to the output port. A duplex sheet is *inverted*, routed to the *transport* along the duplex loop, an image is transferred on its back side (*transfer*), and the sheet is inverted again, before it is sent to the output port. Parameters that affect desired optimization criteria include the transit times of the components and capacities of buffers at the input and output (Kapadia, Biswas, & Fromherz 1997). To generate a document (i.e., an ordered sequence of simplex and duplex sheets) the transportation and printing of sheets must satisfy behavior constraints, for example, sheets must be manipulated such that they are available at the output in the specified order, and sheets must not collide with each other anywhere in the paper path.

Fig. 1 shows a schematic of the software subsystem which comprises the following processes. The *accumulate* process receives sheet descriptions from an external source. State-of-the-art reprographic machines are equipped with a scheduler (illustrated in Fig. 2) that receives a stream of sheet descriptions, and dynamically determines optimal times at which individual actions must be initiated to produce the desired output by a process of heuristic search (Fromherz & Carlson 1994). The scheduler may employ different online algorithms (e.g., greedy methods, search with limited lookahead, etc.) which trade off the optimality of the schedule generated and computation time. To prevent the scheduler from being overwhelmed by large document descriptions, it may be designed to consider a fixed number of sheets (called its *lookahead* (L)) for any computation. In general, we expect larger values of lookahead to improve the prospect of determining an optimal schedule because the scheduler has access to additional information while making its decisions. However, the size of the search space explored by the scheduler increases with larger values of lookahead, which increases the software computation time (T) and consequently, the overall job completion time. We assume that T is a function of L, i.e., $T = f(L)$. At the termination of a computation, the scheduler may initiate the execution of all the actions computed in the schedule. Conversely, it may commit itself to only a predefined number of actions (we call this the *commitment* parameter (C)). This policy of non-commitment affords it greater flexibility; if more information about the document arrives later that makes an alternative schedule look more promising, the scheduler can reschedule the uncommitted sheets accordingly. The *command* process communicates control commands to the hardware subsystem for the actions committed to by the scheduler.

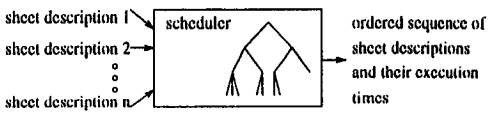


Figure 2: Schedule computation task

Table 1 shows an optimal schedule[1] for the document description consisting of one simplex sheet followed by two duplex sheets, and then a simplex sheet (i.e., s_1, d_2, d_3, s_4) which is completed in 14 time units. While generating this schedule, we assumed the following parameter values: printing an image on to a sheet requires one time unit, inverting a sheet takes two units, transporting a sheet along the duplex loop requires three units, bypassing inversion is instantaneous, software computation takes one unit, lookahead is two, and the commitment parameter is one. The job's completion time is a function of the system parameters (both hardware and software) and the schedule for the job. Optimizing the behavior of this system is particularly difficult because there is no predefined function that maps the job completion time to the design variables for any job. In (Kapadia & Biswas 1999), we have presented model-based reasoning techniques that start from a compositional model of the reprographic machine system and determine this mapping for a given job.

Modeling for optimization

A model of a system is a representation that is tailored towards addressing a specific set of tasks to be performed on the system. Creating a model for design optimization first requires that the designer must identify design variables and system optimization objectives. The model of system behavior must incorporate the design variables as system parameters and allow the designer to study and reason about their effects on optimization objectives. The designer must determine a level of detail for modeling that is appropriate for the design task. It may be necessary to combine different kinds of knowledge into a single model, e.g., a model may incorporate quantitative and qualitative relations among its parameters. Creating a veridical model of the system at the desired level of detail, is a difficult problem that requires considerable insight, experience

[1] An optimal schedule for a job description is one that completes the job in the shortest time.

Component/Process	Place	Time														
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
accumulate	p_2	s_1	d_2	d_3	s_4											
schedule	p_4			d_2	d_3	s_1,s_4										
command	p_5			d_2		d_3			s_1						s_4	
sheet in	p_6			d_2		d_3			s_1						s_4	
transfer	p_7				d_2		d_3			s_1	d_2		d_3			s_4
invert	p_8					d_2		d_3		s_1		d_2		d_3		s_4
transport	p_9							d_2		d_3						
sheet out	p_{10}									s_1			d_2		d_3	s_4

Table 1: Integrated hardware and software behavior for the job sequence s_1, d_2, d_3, s_4

and, often, trial and error. Compositional modeling approaches, particularly those that model the behavior of the system from domain principles and component descriptions, help to simplify this problem (Fromherz & Saraswat 1995). Furthermore, the model must facilitate the performance of tasks that are used for design optimization, e.g., behavior generation and analysis. Behavior generation for a system which combines subsystems from disparate domains must support the study of the interactions that occur among these subsystems. Behavior analysis must be holistic, i.e., it must permit reasoning about the effects of design variables on optimization objectives in a system-wide manner.

Abstract models of components and processes. (DeKleer & Brown 1984) model the behavior of a physical system in terms of material, components, and conduits. In our application, the system's components span the electromechanical hardware and the software domain, and we are interested tracking the movements of sheets and data in the system at discrete time points. For each component (as shown in Fig. 3), we model its *structure* (i.e., input and output ports) and, with the demands of our application in mind, its *temporal behavior* defined in terms of spatial locations and time stamps of material and data (i.e., the time it takes for the material to flow through the component from an input port to an output port).

(Gupta & DeMicheli 1993) model software processes in terms of primitive *operations*, i.e., assignments, conditional tests, loops, etc. Their software model represents the time required for the execution of each operation and temporal constraints among the operations. (Thomas, Adams, & Schmidt 1993) choose a more abstract representation for their software subsystems by merging sets of operations into *software processes*. We select a process level representation where each process, which represents a collection of primitive operations, is modeled as an executable "black box". Data flow through a process is analogous to material flow in a hardware component. The time taken by a software process to perform its specified computation is a function of the nature of the task and the design variables that affect the process. Given that we are primarily interested in tracking the movement of materials and data

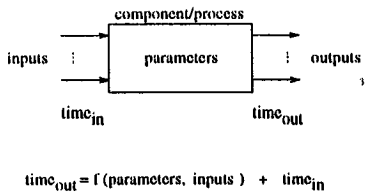


Figure 3: Modeling hardware components and software processes

in the system at discrete time points, software processes and hardware components in the reprographic machine system may be modeled at a uniform level of abstraction as shown in Fig. 3.

Environment Relationship nets. We are developing a modeling methodology that can represent our hardware and software subsystems in a uniform manner and can support reasoning about the dependencies among them. We have adapted the Environment Relationship (ER) Net (Ghezzi *et al.* 1991) framework, a timed extension of basic Petri nets which supports tokens with properties, as the basis of our system modeling methodology. Petri nets have been used to model discrete systems because of their graphical nature, their ability to represent system structure and dynamic behavior, and the availability of mathematical analysis techniques. A key advantage of this framework is that we can use the same language to represent both hardware and software subsystems. We describe the properties of our ER net-based modeling methodology below.

Material. In the ER net framework, a token is a collection of attribute-value pairs (ID, V), where ID is a set of identifiers, and V is a set of values. Among the attributes is the token's *time* stamp, i.e., the time when the token is created. Tokens are described by the following notation.

- A place name (e.g., P) stands for any token in P.
- If y is a token, and x is an attribute, then $y.x$ stands for the value of attribute x in token y.

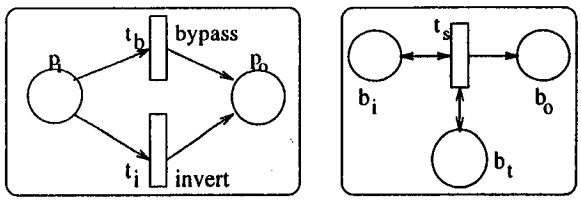
In the hardware subsystem, sheets represent material that flows through the machine. A sheet is represented by a token with the following attributes: *type* $\in$ {s,d} (s for simplex, d for duplex); *position* in document; number of remaining passes (*pass*) through the transfer component, initially if *type* = s, *pass* = 1; if *type* = d, *pass* = 2; and a *time* stamp of the token that represents current time.

For each physical sheet in machine, we model a corresponding *sheet description* datum in the software subsystem. A sheet description is modeled by a token with the following attributes: *type*, *position*, *pass*, *time*, and *execution*. The first three attributes are used to identify the token, while *time* represents the token's time stamp and *execution* records the time at which the physical sheet corresponding to the sheet description is introduced at the hardware subsystem's input port for execution. A *schedule* for a set of sheets is represented as a token with two attributes: an ordered *list* of sheet description tokens (the tokens are in ascending order of their execution time attribute), and a measure of *optimality* that expresses the designer's qualitative belief in the optimality of the schedule, which is related to L, i.e., *optimality* $\propto L$.

Components and processes. In the ER net framework, each transition has an associated action that maps tokens in its input places to tokens in its output places. Associated with an action is a predicate that must be satisfied by tokens in the input places for the transition to be activated. A transition is enabled in a marking, if and only if, for every input place of the transition there is a token that satisfies the predicate of the action. An enabled transition fires by removing a token in each of its input places and producing one in each output place. The values of the attributes of tokens in the output places are determined by the action.

Our abstractions of hardware components and software processes (Fig. 3) allow us to model them in a uniform way. Hardware components are modeled as a set of transitions with one or more input and output places corresponding to the component's ports. Software processes are modeled as a set of transitions with places corresponding to input and output buffers where data can be stored for any length of time. Sometimes a place may act as both input and output for a component, e.g., a buffer which is updated (i.e., both read from and written to by a process). Actions associated with transitions correspond to functions that the component or process performs, i.e., the transformations that the material (data) undergoes as it passes through the component (process), and its behavior, i.e., how the component (process) transforms the material (data) flowing through the component (process).

Formally, a process or component is modeled by a tuple $\langle P, T\rangle$ where $P = \{p_1, \ldots, p_m\}$ is a set of places and $T = \{t_1, \ldots, t_n\}$ is a set of transitions. The notation



(a) inverter (b) schedule

Figure 4: Modeling hardware components and software processes as ER nets

for describing a transition is:

$$t_i = \{\langle \text{input places, output places}\rangle | \text{actions}\}.$$

Fig. 4(a) shows an ER net fragment for the *inverter* component which performs two functions: it either inverts a sheet or lets it pass through without inversion. p_i and p_o are the input and output of the inverter, respectively. Its behavior is represented by the following transitions:

1. t_b represents bypassing inversion in the inverter. We assume that simplex sheets bypass inversion instantaneously. $t_b = \{\langle p_i, p_o\rangle | p_i.type = s \textbf{ and } p_o.time \leftarrow p_i.time\}$.
2. t_i represents the inversion of a sheet in the inverter. We assume that inversion requires k_2 units of time, and only duplex sheets undergo inversion. $t_o = \{\langle p_i, p_o\rangle | p_i.type = d \textbf{ and } p_o.time \leftarrow p_i.time + k_2\}$.

Fig. 4(b) shows an ER net fragment for the reprographic machine's *scheduler*, which computes execution times for each sheet description. The *scheduler* has the following input places:

- buffer b_i stores sheet description tokens that have not been scheduled for execution, and
- place b_t stores the current *time*; initially, $b_t.time = 0$.

Its output places are:

- buffer b_o which contains a schedule of operations, i.e., a list of sheet description tokens ordered according to their execution times, and
- place b_t so that $b_t.time$ can be read and updated.

The scheduler (represented as transition t_s) is invoked when there are L sheet description tokens in b_i, where L is the lookahead parameter[2]. It computes a schedule spending T units of time in this process, where $T = f(L)$. We assume that this function can be established empirically from past experience.

To formally define transition t_s, we first define the following auxiliary functions.

- *compute*(*Place*) computes a schedule for the list of tokens in *Place* and returns a list of sheet description tokens with their assigned *execution* times. The

[2]Once the complete job description is available at its input, the scheduler is invoked even if there are fewer than L tokens.

list is in chronological order of the *execution* times (which as demonstrated in Table 1, may differ from the specified order of the sheets, e.g., d_2 is introduced first at time 2, while s_1 is introduced at time 7).

- *commit*$(List, C)$ commits to the execution of the first C actions in $List$.
- *maximum*$(Place)$ returns the maximum time attribute for all tokens in $Place$.
- *max*(X, Y) returns the maximum value of two integers X and Y.

We have:

$$t_s = \{(\{b_i, b_t\}, \{b_t, b_o\}) | \textbf{ and } b_o \leftarrow \text{commit}(\text{compute}(b_i), C) \textbf{ and } b_t.time \leftarrow \max(b_t.time + T, \text{maximum}(b_i) + T) \textbf{ and for each token in } b_o, b_o.time \leftarrow \max(b_t.time + T, \text{maximum}(b_i) + T) \textbf{ and } b_o.optimality \propto L\}.$$

Once t_s is invoked, it must not be invoked again for T units of time (i.e., it must not be preempted). ER nets allow us to impose the condition that no transition can be fired before the *time* attribute of any token that is consumed by the transition. In this model, each subsequent firing of t_s is delayed until $\max(b_t.time + T, \text{maximum}(b_i) + T)$.

Model composition. We represent connections between system components by sharing places. For example, we model the connection between two hardware components by sharing the output port of one with the input port of the other, and the communication between two software processes or a software process and a hardware component by sharing buffers. Combining the hardware and software component models by sharing p_5 results in the composite model of Fig. 5 for the reprographic machine of Fig. 1.

Discussion

Our research is targeted at developing a methodology to support system-level design optimization (Kapadia & Biswas 1999). In this paper, we presented a framework for modeling the behavior of a system with components that span multiple domains. Developing a uniform representation for such a system allows for transparent communication of relevant design information and decisions between members of a design team working concurrently on the problem and the application of global reasoning mechanisms to different aspects of system design.

Supporting design optimization. Presently, we have developed techniques that utilize our models for the following tasks.

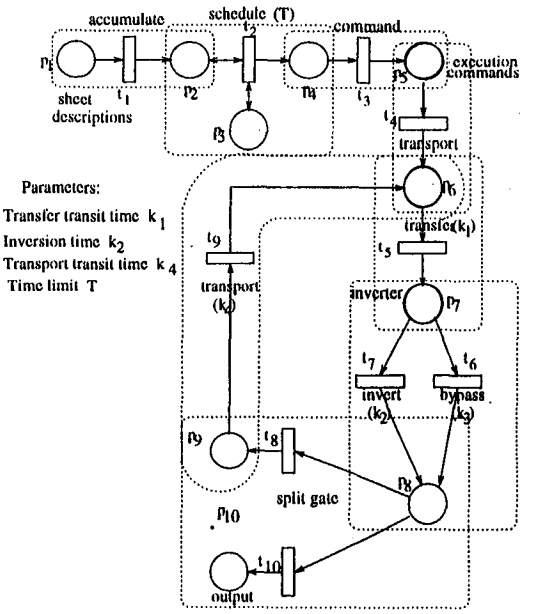


Figure 5: Reprographic machine system model with ER nets

Behavior generation. We can use our ER net model to generate system behavior as shown in Table 1. Generating integrated hardware and software behavior facilitates an accurate and holistic analysis of the system.

Modeling temporal constraints between system events. (Fromherz & Saraswat 1995) employ compositional models of system components to derive temporal constraints that are used for controlling system behavior. In a similar manner, our ER net system model may be used for scheduling operations in the reprographic machine system (Kapadia, Biswas, & Fromherz 1997).

Generating optimization relations. As mentioned before, the relation between a job's completion time and the system design variables may not be known explicitly because it depends on decisions made by the scheduler at run time. We have developed an optimization methodology that derives an *event* model from the ER net system model and a given job description. An event model is a directed acyclic graph where each vertex represents an event (informally, an event is the arrival of a token in a place in the system model) and a directed edge between two vertices represents a precedence relation between the corresponding events. Formally, we use the notation $t(r)^i_j$ to represent the event that the i^{th} token is in place j on it's r^{th} pass through the machine. Fig. 6 shows the event model for the ER net system of Fig. 5 and the behavior depicted in Table 1. We have used event models to derive relations between optimization objectives and design variables for different job descriptions (Kapadia & Biswas 1999).

Dependencies among subsystems. Integrating different subsystems requires considering dependencies

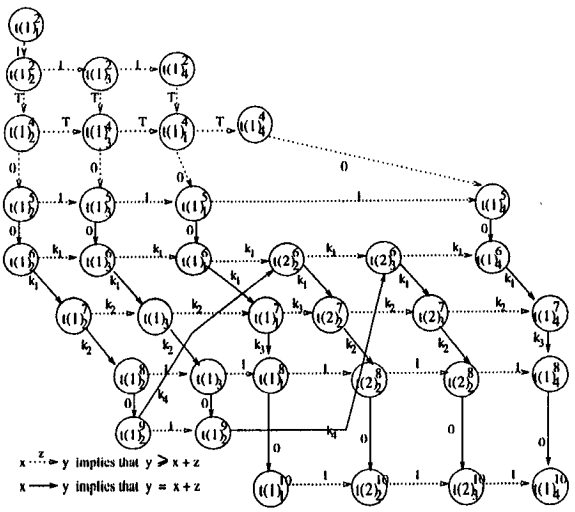


Figure 6: Event model for Table 1

among their parameters and events in order to find optimal design solutions.

An example of a dependency among parameters in our reprographic machine system is the maximum lookahead constraint for making an optimal scheduling decision which captures the dependency between software parameters lookahead (L) and commitment (C), and transit times for hardware components for printing an image to a sheet (k_1) and transporting a sheet along the duplex loop (k_4). By combining intuition and strong domain knowledge (Kapadia 1999), we have discovered that the following constraint guarantees optimal scheduling decisions:

$$L = C - 1 + \lfloor \frac{k_4 + k_1 - 1}{k_1} \rfloor.$$

An example of a dependency among the system's events is the issue of synchronizing the computation and commitment of sheet descriptions in the software subsystem and the execution of the corresponding operations in the hardware subsystem. If the commitment parameter C is too small and the software computation time T is too large, it is possible that the hardware subsystem may be largely idle while the software subsystem is always busy. This is undesirable because it results in poor utilization of the hardware subsystem.

We are investigating techniques to derive these dependencies from our system models rather than from our intuitions of system behavior. We anticipate that inferring these dependencies may require augmenting traditional model-based reasoning with techniques from machine learning.

Modeling software in richer detail. Presently, we do not model the working of the scheduling algorithm used by the control software. Instead, we employ qualitative and empirical associations (e.g., $optimality \propto L$ and $T = f(L)$) which capture our intuition and experience regarding software computation. We are trying to develop a coherent set of "first principles" in the software domain and reasoning techniques that will help us to use these principles to derive such associations.

Acknowledgements

The author thanks Gautam Biswas and Markus Fromherz for their help in developing and refining the ideas presented in this paper.

References

DeKleer, J., and Brown, J. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 7–83.

Fromherz, M., and Carlson, B. 1994. Optimal incremental and anytime scheduling. In *Proc. Workshop on Constraint Languages/Systems and their Use in Problem Modeling*, 45–59.

Fromherz, M., and Saraswat, V. 1995. Model-based computing: Using concurrent constraint programming for modeling and model compilation. In *Proc. Concurrent Programming*, 629–635. Springer-Verlag. LCNS 976.

Ghezzi, C.; Mandrioli, D.; Morasca, S.; and Pezze, M. 1991. A unified high-level petri net formalism for time critical systems. *IEEE Trans. on Software Engg.* 160–172.

Gupta, R., and DeMicheli, G. 1993. Hardware software cosynthesis for digital systems. *IEEE Design and Test of Computers* 29–41.

Kapadia, R., and Biswas, G. 1999. Model-based support for parametric mutable design optimization. To appear in Proc. of AAAI99.

Kapadia, R.; Biswas, G.; and Fromherz, M. 1997. Hybrid modeling for smart system design. In *Proc. Tenth Intl. FLAIRS*, 111–115.

Kapadia, R. 1999. Model-based support for system-level mutable parametric design optimization. Technical Report TR-99-02, CS Dept. Vanderbilt University, Nashville TN 37235.

NSF. 1996. Research opportunities in engineering design. NSF Strategic Planning Workshop Final Report.

Thomas, D.; Adams, J.; and Schmidt, H. 1993. A model and methodology for hardware-software codesign. *IEEE Design and Test of Computers* 6–15.