OOPM/RT: A Multimodeling Methodology for Real-Time Simulation

Kangsun Lee
University of Florida
and
Paul A. Fishwick
University of Florida

When we build a model of real-time systems, we need ways of representing the knowledge about the system and also time requirements for simulating the model. Considering these different needs, our question is "How to determine the optimal model that simulates the system by a given deadline while still producing valid outputs at an acceptable level of detail?" We have designed OOPM/RT(Object-Oriented Physical Multimodeling for Real-Time Simulation) methodology. OOPM/RT framework has three phases: 1) Generation of multimodels in OOPM using both structural and behavioral abstraction techniques, 2) Generation of AT (Abstraction Tree) which organizes the multimodels based on the abstraction relationship to facilitate the optimal model selection process, and 3) Selection of the optimal model which guarantees to deliver simulation results by the given amount of time. A more detailed model (low abstraction model) is selected when we have enough time to simulate, while a less detailed model (high abstraction model) is selected when the deadline is immediate. The basic idea of selection is to trade structural information for a faster runtime while minimizing the loss of behavioral information. We propose two possible approaches for the selection: integer programming based-approach and search-based approach. By systematically handling simulation deadlines while minimizing the modeler's interventions, OOPM/RT provides an efficient modeling environment for real-time systems.

Categories and Subject Descriptors: I.6.5 [**Simulation and Modeling**]: Model Development

General Terms: Modeling Methodology, Real-Time Simulation

Additional Key Words and Phrases: Model Abstraction, Model Selection, Real-Time Systems

1. INTRODUCTION

Real-time systems refer to systems that have hard real-time requirements for interacting with a human operator or other agents with similar time-scales. An efficient simulation of real-time systems requires a model that is accurate enough to accomplish the simulation objective and is computationally efficient [Garvey and Lesser 1993b; Garvey and Lesser 1993a; Lee and Fishwick 1998]. We define *model ac-*

Address: CISE P.O. Box 116120, University of Florida, FL 32611-6120

This work is supported by (1) GRCI (Incorporated 1812-96-20 (Gregg Liming) and Rome Laboratory (Steve Farr, Al Sisti) for web-based simulation and multimodeling; (2) NASA/Jet Propulsion Laboratory 961427 (John Peterson and Bill McLaughlin) for web-based modeling of spacecraft and mission design, and (3) Department of the Interior under ATLSS Project contract 14-45-0009-1544-154 (Don DeAngelis, University of Miami) for techniques for both code and model integration for the across-trophic-level Everglades ecosystem.

curacy in terms of the ability of a model to capture the system at a right level of detail and to achieve the simulation objective within an allowable error bound. Computational efficiency involves the satisfaction of the real-time requirements to simulate the system, in addition to the efficiency of model computation. In existing applications, it is a user's responsibility to construct the model appropriate for the simulation task. This is a difficult, error-prone, and time-consuming activity requiring skilled and experienced engineers.

Most CASE tools [Digital 1989] try to help the modeling process by providing an extensive library of functions, which allow a modeler to specify numerous aspects of an application's architecture. These tools deal with static models suitable for producing design documents, with limited facilities for simulating the models, analyzing the results of such simulations, running what-if questions, or translating the paper models to prototype code. However, these tools do not provide support for specifying the real-time constraints of an application's functions [Lark, J. S., Lee D. Erman, Stephanie Forrest and Kim P. Gostelow 1990; Burns, A. and Wellings, A. J. 1994].

Our objective is to present a modeling methodology with which the real-time systems can be modeled efficiently to meet the given simulation objective and time requirements.

One of the contributions of our research is that, with the ability to select an optimal model for a given deadline, we provide a semi-automatic method to handle real-time constraints for a simulation. In particular, we handle a time constraint out of the modeling processes; therefore, modelers are relieved from considering constraints that are not supposed to be part of modeling. Another contribution is that by generating a set of multiple methods through abstraction techniques and selecting the optimal abstraction degree to compose a model for the real-time simulation, we meet not only the real-time constraints, but also the perspective which modelers see the system for a given time-constraint situation. We expect that the proposed method can provide better sources of multiple methods for real-time computing groups.

This paper is organized as follows : In Section 2, we discuss several related research activities. We propose the OOPM/RT modeling framework in Section 3. In Section 4, we present our abstraction methodology to generate a set of models for a system at different levels of detail. In Section 5, we show how to organize the models in a way that facilitates the selection process. Two optimal model selection algorithms are presented in Section 6. A complete process from model generation to the selection of the optimal abstraction level is illustrated in Section 7 through an example. We conclude in Section 8.

2. MODELING OF REAL-TIME SYSTEMS

Real-time systems differ from traditional data processing systems in that they are constrained by certain non-functional requirements (e.g., dependability and timing). Although real-time systems can be modeled using the standard structured design methods, these methods lack explicit support for expressing the real-time constraints [Kopetz, H., Zainlinger, R., Fohler, G., Kantz, H., Puschner, P. and Schutz, W. 1991; Lark, J. S., Lee D. Erman, Stephanie Forrest and Kim P. Gostelow 1990; Burns, A. and Wellings, A. J. 1994]. Standard structured design methods

incorporate a life cycle model in which the following activities are recognized:

(1) Requirements Definition - an authoritative specification of the system's required functional and non-functional behavior is produced.
(2) Architectural Design - a top-level description of the proposed system is developed.
(3) Detailed Design - the complete system design is specified.
(4) Coding - the system is implemented.
(5) Testing - the efficacy of the system is tested.

For hard real-time systems, this has the significant disadvantage that timing problems will be recognized only during testing, or worst after deployment [Burns, A. and Wellings, A. J. 1994]. Researchers have pointed out that the time requirements should be addressed in the design phase [Burns, A. and Wellings, A. J. 1994; Lark, J. S., Lee D. Erman, Stephanie Forrest and Kim P. Gostelow 1990].

Two activities of the architectural design are defined [Burns, A. and Wellings, A. J. 1994]: 1) *the logical architecture design activity*, and 2) *the physical architecture design activity*. The logical architecture embodies commitments that can be made *independently* of the constraints imposed by the execution environment, and is primarily aimed at satisfying the *functional* requirements. The physical architecture takes these functional requirements and other constraints into account, and embraces the *non-functional* requirements. The physical architecture forms the basis for asserting that the application's non-functional requirements will be met once the detailed design and implementation have taken place. The physical architecture design activity addresses timing (e.g. responsiveness, orderliness, temporal predictability and temporal controllability) and dependability requirements (e.g., reliability, safety and security), and the necessary schedulability analysis that will ensure that the system once built will function correctly in both the value and time domains. Appropriate scheduling paradigms are often integrated to handle non-functional requirements [Burns, A. and Wellings, A. J. 1994]. The following issues arise :

—How to capture the logical aspects of the real-time systems?
—How to assess duration and quality associated with each model?
—How to resolve timing constraints?
—How to support both logical and physical activities under one modeling and simulation framework so that the resulting model is guaranteed to function correctly in both the value and time domains?

Several areas of research relate to these issues. Real-time scheduling focuses on how to deal with the physical requirements of the system. The main interest is to determine a schedule that defines *when to execute what task* to meet a deadline. Typical approaches to real-time scheduling assume that task priorities and resource needs are completely known in advance and are unrelated to those of other tasks, so that a control component can schedule tasks based on their individual characteristics. If more tasks exist than the system can process, the decision about which tasks to ignore is simple and local, usually based only on task priority [Ramamritham, K. and Stankovic, J. A. 1984; Stankovic, J. A., Ramamritham, K. and Cheng, S.

1985]. Our claim is that the resulting schedule of tasks does not reflect the real objective of the simulation when the selection is made based solely on task priority.

Real-Time Artificial Intelligence studies problem-solving methods that "given a time bound, dynamically construct and execute a problem solving procedure which will (probably) produce a reasonable answer within (approximately) the time available." [D'Ambrosio, B. 1989] Examples of this type are found in chess programs. Virtually all performance chess programs in existence today use full-width, fixed-depth, alpha-beta minimax search with node ordering, quiescence, and iterative-deepening for the real time problem solving. They make very high quality move decisions under real-time constraints by properly controlling the depth of search (or move) and having a good heuristic function that guides the search (or move). Research on the real-time problem solving through search methods can be found in Refs. [Korf 1990; Barr and Feigenbaum 1981].

The key to these approaches is to have *single* problem solving method that achieves a better result as the method is given more time. One of the problems is that these approaches rely on the existence of iterative refinement algorithms that produce incrementally improving solutions as they are given increasing amounts of runtime. Clearly, such algorithms exist for some problem cases, but also there are problems that will be difficult to solve in an incremental fashion [Garvey and Lesser 1993b; Garvey and Lesser 1993a]. An alternative to this approach is to have *multiple* methods to model the system which make tradeoffs in cost versus quality, and may have different performance characteristics in different environment situations. Garvey and Lesser proposed the *Design-to-Time* [Garvey and Lesser 1993a; Garvey and Lesser 1995] method, which is related to our approach. *Design-to-time* assumes that one has multiple methods for the given tasks and tries to find a solution to a problem that uses all available resources to maximize solution quality within the available time. They present an algorithm for finding optimal solutions to a real-time problem under task tree graph and task relationships [Garvey and Lesser 1995]. The algorithm generates all sets of methods that can solve the problem and prunes those superseded by other sets of methods that generate greater or equal quality in equal or less time. *Design-to-time* has a single model type and multiple methods are generated through approximation techniques; therefore, it concerns only the behavioral aspects of the system. In order to model a complex system, it's better to have different model types to efficiently characterize the different aspects of the complex system.

3. OOPM/RT : A MODELING METHODOLOGY FOR REAL-TIME SIMULATION

We have built OOPM/RT(Object-Oriented Physical Multimodeling for Real-Time Simulation) for aiding the user to meet arbitrary time and quality constraints imposed upon the simulation. OOPM/RT adopts a philosophy of *rigorous engineering design*, an approach which requires the system model to guarantee the system's timeliness at *design* time [Lark, J. S., Lee D. Erman, Stephanie Forrest and Kim P. Gostelow 1990]. OOPM/RT uses OOPM for the logical architecture design activity. OOPM is an implementation of OOPM (Object-Oriented Physical Multimodeling) [Fishwick 1997], an approach to modeling and simulation which promises not only to tightly couple a model's human author into the evolving modeling and simulation process through an intuitive HCI (Human Computer Interface), but also

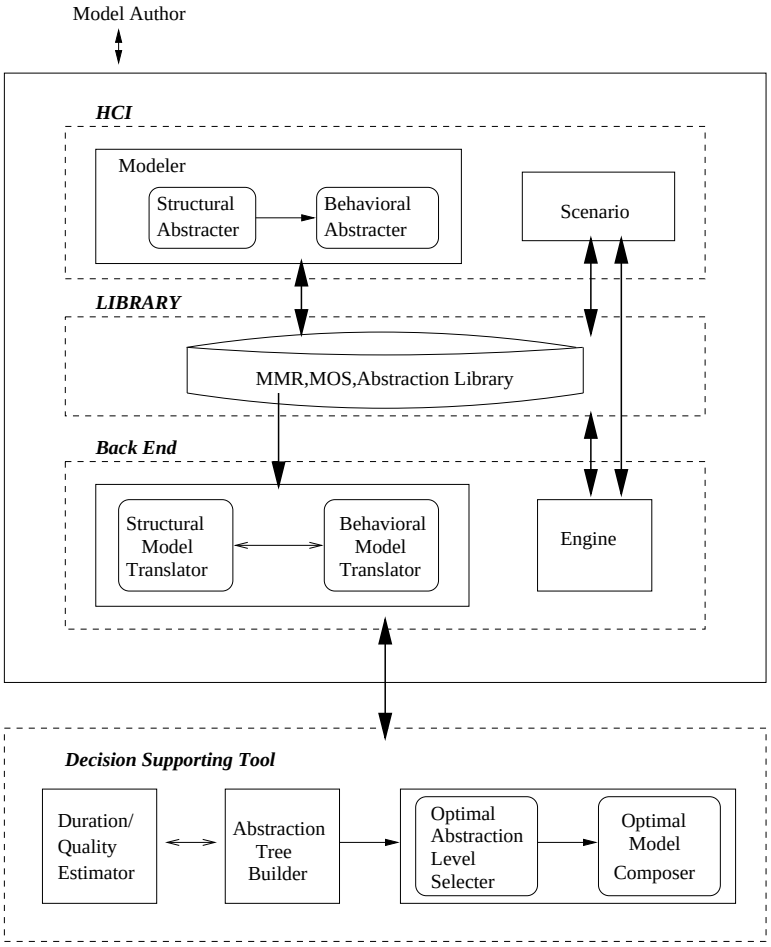


Fig. 1. OOPM/RT : OOPM is used for the logical architecture design activity. The Decision Supporting Tool is integrated to handle the physical architecture design activity. The three components of OOPM (HCI, Library, and Back End) are shown as boxes with dashed lines. Parts within each component are shown outlined with solid boxes: HCI (Human Computer Interface) appears within Modeler and Scenario; Back End appears within Translator and Engine; Library appears within MMR (OOPM Model Repository) and MOS (OOPM Object Store). Principal interactions are shown with arrows. The model author interacts with both parts of the HCI. The Decision supporting tool interacts with OOPM being hidden from a model author

to help a model author to perform any or all of the following objectives [Cubert and Fishwick 1997]:

—to think clearly about, to better understand, or to elucidate a model

—to participate in a collaborative modeling effort

—to repeatedly and painlessly refine a model with heterogeneous model types as required, in order to achieve adequate fidelity at minimal development cost

—to painlessly build large models out of existing working smaller models

—to start from a conceptual model which is intuitively clear to domain experts, and to unambiguously and automatically convert this to a simulation program

—to create or change a simulation program without being a programmer

—to perform simulation model execution and to present simulation results in a meaningful way so as to facilitate the prior objectives

By using OOPM for the sources of creating methods, we can model a system efficiently with different model types together under one structure. For time-critical systems, we may prefer models that produce less accurate results *within* an allowable time, over models that produce more accurate results *after* a given deadline. The key to our method is to use an abstraction technique as a way of handling real-time constraints given to the system. We generate a set of methods for the system at different levels of abstraction through a model abstraction methodology. When the constructed model cannot be executed for a different simulation condition, such as a tighter deadline, we change the abstraction degree of the model to deliver the simulation results by the given amount of time. The decision supporting tool is added to OOPM in order to take these constraints into account and determine the optimal abstraction degree for a given simulation deadline. Based on the determined abstraction degree, the optimal model is composed. The decision process is placed out of the modeling process, therefore modelers are relieved from considering time constraints that are not supposed to be part of modeling. The OOPM/RT structure is shown in Figure 1.

OOPM/RT is useful in the problem domain where

—Sacrificing solution quality can be tolerated so that the systems can be modeled by multiple solution methods that produce tradeoffs in solution quality and execution time

—Duration and quality associated with a method is fairly predictable

Our method has three phases :

(1) Generating a set of models at different abstraction levels.

(2) Arranging a set of models under the abstraction relationship and assessing the quality/cost for each model

(3) Executing model selection algorithms to find the optimal model for a given deadline

In the first phase, a set of methods is generated at a different degree of detail using an abstraction methodology. The second phase is to assess the expected quality and runtime of each method and organizes a set of methods in a way to facilitate

the selection process. Related research has been done for the estimation of runtime/quality [Marin G. Harmon and Walley 1994; Horst F. Wedde and Huizinga 1994; Rutledge 1995]. Our focus is not to propose a new estimation method. Instead, we assume that the execution time of each method is proposely measured by the available methods. In the third phase, we select an optimal model among the alternatives when the constructed model cannot be executed for a given amount of time. The selection is made by deciding the optimal abstraction level to simulate the system within a given amount of time. A more detailed model (low abstraction level) is selected when we have enough time, while a less detailed model (high abstraction level) is used when there is an imminent time constraint.

4. MODEL GENERATION

We generate a set of methods for the system by using model abstraction techniques. Model abstraction is a method (simplifying transformation) that derives a "simpler" model from a more complex model while maintaining the validity (consistency within some standard of comparison) of the simulation results with respect to the behaviors exhibited by the simpler model. The simpler model reduces the complexity as well as the quality of the model's behavior [Caughlin and Sisti 1997]. The proper use of abstraction provides computational savings as long as the validity of the simulation results is guaranteed. Our approach is to use the abstraction method when we need to reduce the simulation time to deliver the simulation results by a given deadline. A set of models are generated through abstraction techniques with different degrees of abstraction; each model simulates the system within a different amount of time, while describing the system with a different perspective. Therefore, a model that is selected for a given real-time simulation is useful not just because it meets a given deadline, but also because it suggests a perspective with which modelers view the system for a given time-constraint situation.

We have studied abstraction techniques available in many disciplines and created an unified taxonomy for model abstraction where the techniques are structured with the underlying characterization of a general approach [Lee and Fishwick 1996; Lee and Fishwick 1997a]. Our premise is that there are two different approaches to model abstraction: *structural* and *behavioral*. *Structural abstraction* is the process of abstracting a system in terms of the structure using refinement and homomorphism [Zeigler 1976; Zeigler 1990]. Structural abstraction provides a well-organized abstraction hierarchy on the system, while behavioral abstraction focuses only on behavioral aspects of the system without structural preservation. We organize the system hierarchically through the structural abstraction phase and construct an abstraction hierarchy with simple model types first, refining them with more complex model types later. Our structural abstraction provides a way of structuring different model types together under one framework so that each type performs its part, and the behavior is preserved as levels are mapped [Fishwick 1995]. The resulting structure becomes a *base model* which has the highest resolution to model the system. The problem is that selecting one system component from an abstraction level is dependent on the next lowest level due to the hierarchical structure. Each component cannot be *executed* at a random abstraction level, though it can be *viewed* independently. *Behavioral abstraction* is used when we want to simulate the base model at a random abstraction level. We isolate an abstraction level by

approximating the lower levels behavior and replacing them with a behavioral abstraction method. The method discards structural information of the lower levels, but the behavioral information is preserved in some level of fidelity. Possible techniques for behavioral abstraction are system identification, neural networks, and wavelets [Masters 1995]. Since our method involves less computation time by discarding lower levels structural information, it will be used when there is too little time to simulate the system in detail. The abstraction mechanism is implemented in OOPM, and the models produced by OOPM become the sources of methods for performing the real-time simulation. More detailed discussions on our abstraction methodology are found in Refs.[Lee and Fishwick 1996; Lee and Fishwick 1997a; Lee and Fishwick 1997b].

5. CONSTRUCTION OF THE ABSTRACTION TREE

AT (Abstraction Tree) extends the tree structure to represent 1) all the methods that comprise the base model and 2) refinement/homomorphism relationship among the methods. Every method that comprises the base model is represented as a **node**. Each node takes one of three types : M_i, A_i or I_i.

—M_i - High resolution method. It takes the form of dynamic or static methods of OOPM. We have FBM (Functional Block Model), FSM (Finite State Machine), SD (System Dynamics), EQM (EQuational Model), and RBM (Rule-Based Model) choices for the dynamic method, and the CODE method for the static method

—A_i - Low resolution method. It takes the form of a neural network or a BJ (Box-Jenkins) ARMA (AutoRegressive Moving Average) model

—I_i - Intermediate node to connect two different resolution methods, M_i and A_i. I_i appears where a method i has been applied to behavioral abstraction, and the corresponding behavioral abstraction method has been generated for a low resolution method to speedup the simulation

The Refinement/Homomorphism relationship is represented as an **edge**. If a method M_i is refined into $N_1, N_2, N_3, ..., N_k$, an edge(M_i, N_j), for $j = 1, 2, ..., k$, is added to the AT. *AND/OR* information is added on the edge to represent how to execute M_i for a given submethod N_j, for $j = 1, 2, ..., k$.

—AND - M_i is executed only if N_j is executed, $\forall\ j$, $j = 1, 2, ..., k$

—OR - M_i is executed only if any N_j, $j = 1, 2, ..., k$, is executed

The decision of AND/OR is made based on 1) the node type of M_i and 2) the model type of M_i.

(1) Node type : An intermediate node I_i is executed either by H_i or L_i, where H_i is a high resolution method and L_i is the corresponding low resolution method. Therefore, I_i is connected with the *OR* relationship.

(2) Model type : If a method M_i takes the form of an FBM, and each of the block that comprises M_i is refined into $B_1, B_2, ..., B_k$, then the execution of M_i is completed when B_j, $\forall\ j$, $j = 1, 2, ..., k$, are executed. Therefore, an FBM method M_i is connected with the *AND* relationship. Other examples of the *AND* relationship are SD, EQM, and CODE method. However, other model

types can take the *OR* relationship. If a method M_i takes the form of an FSM, and each state of the FSM is refined into $S_1, S_2, ..., S_k$, then the execution of M_i is completed when any S_j, $j = 1, 2, ..., k$, is executed. The decision of j is made according to the predicate that the FSM method, M_i, satisfies at time t. Therefore, FSM method M_i is executed with the *OR* relationship. RBM is another example of the OR relationship.

Each node, T, in AT has **duration** $D(T)$ and **quality** $Q(T)$. $Q(T)$ summarizes three properties of the quality associated with node T.

(1) QA(T) - Degree of abstraction. Degree of abstraction represents how much information would be lost if the execution occurs at node T. The base model will not be executed at the associated leaf nodes when the method T is selected for the behavioral abstraction. QA(T) is defined by how many methods are being discarded if behavioral abstraction occurs at node T, comparing to the case where no behavioral abstraction is applied to the base model.

(2) QI(T) - Degree of interest loss. A modeler may have certain nodes that he/she wants to see with a special interest; If there are two nodes, A_1 and A_2 in a given AT, and a modeler has a special interest in A_1, it is preferable to take A_2 for behavioral abstraction. QI(T) is defined by how many interesting nodes are being discarded if behavioral abstraction occurs at node T, comparing to the case where no behavioral abstraction is applied to node T.

(3) QP(T) - Degree of precision loss. Degree of precision loss represents how accurately the behavioral abstraction method approximates the high resolution method for node T. The precision can be assessed by testing the trained neural network or Box-Jenkin's ARMA model. Several techniques for estimating error rates have been developed in the fields of statistics and pattern recognition, which include *hold out, leave one out, cross validation,* and *bootstrapping* [Weiss, S. M. and Kapouleas, I. 1989]. We use *holdout* method which is a single train-and-test experiment where a data set is broken down into two disjoint subsets, one used for training and the other for testing. QP(T) is estimated through the testing phase of *holdout*.

Based on the three quality properties, Q(T) is defined by :

$$\begin{aligned} QA(T) &= \frac{N(T)}{N} \\ QI(T) &= \frac{N_i(T)}{N_i} \\ QP(T) &= E(T) \\ Q(T) &= \frac{QA(T) + QI(T) + QP(T)}{q} \end{aligned}$$

where N(T) for the number of nodes in a subtree that has T as a root node, N_i(T) for the number of interesting nodes in a subtree that has T as a root node, N_i for the total number of interesting nodes in a given AT, and N for the total number of node in a given AT. E(T) is the normalized error rate of behavioral abstraction method

for node T. $N(T)$ of QA(T) is set to 1 for a leaf node. q represents the number of quality properties specified for a node, T ($1 \leq q \leq 3$). For an intermediate node, I_i

$$Q(I_i) = Q(H_i) - Q(L_i)$$

where H_i is the high resolution method for node I_i, while L_i is the low resolution method for node I_i. $D(T)$ is defined based on 1) AND/OR relationship and 2) node types. For a node type, I_i, and the corresponding two different resolution methods, H_i and L_i,

$$D(I_i) = D(H_i) - D(L_i) + \theta$$

where θ is the *system factor*; θ considers if the target simulation platform is different from the environment in which the execution time has been measured. Therefore, $D(I_i)$ represents the amount of speedup by replacing the high resolution model H_i with the behavioral abstraction model L_i, in any platform. For other node types, the duration of a node is defined based on its AND/OR relationship. For an AND related node,

$$D(T) = \sum_{j=1}^{k} N_j + \delta(T) + \theta$$

For an OR related node,

$$D(T) = Max\ (N_1, N_2, ..., N_k) + \delta(T) + \theta$$

where N_j, for $j = 1, ..., k$, is the method that T calls to complete its high resolution execution. k is the number of refined methods for T. $\delta(T)$ is the amount of time that method T takes for its own execution. For example, in the case of an FSM, checking the current state and determining the next state based on the predicates might take $\delta(T)$ time, while the execution of each state is assessed in the summation term. θ is a system factor as in the case of I_i node. D(T) of an OR node is set to the worst case execution time by taking the maximum duration of the possibilities. This worst case assignment securely guarantees the resulting model's timeliness. The quality and duration function are constructed recursively, until individual methods at the leaf level are reached.

Figure 2 shows an example of an AT. M_{21} is an AND related method that calls M_{31} and then M_{32} followed by M_{33}. M_{22} is an AND related method that calls M_{34} and then M_{35}. A method M_{11} calls M_{21} and then M_{22}. We suppose that behavioral abstraction methods $\{A_{11}, A_{12}, ..., A_{35}\}$ have been generated for each of the corresponding high level method M_{ij}. Each A_{ij} may take different model type according to the behavioral abstraction technique. The intermediate nodes relate a high resolution method to a corresponding behavioral abstraction. These nodes are symbolized by $\{I_{11}, I_{12}, ..., I_{35}\}$. If all $\{I_{11}, I_{12}, ..., I_{35}\}$ are executed by high resolution methods only, the resulting structure is the base model which has been constructed through the structural abstraction process. The quality and duration of

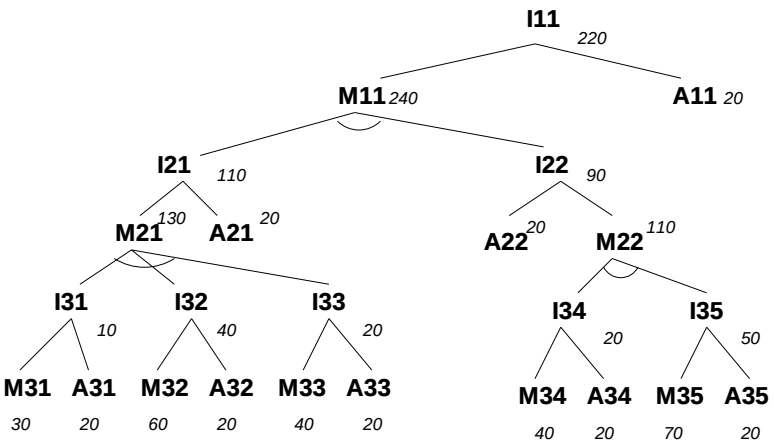


Fig. 2. Example Abstraction Tree

each node is determined by recursively applying the quality and duration equations. δ and θ are assumed to be 0 for each internal node.

6. SELECTION OF THE OPTIMAL ABSTRACTION MODEL

A *Task* is a granule of computation treated by the scheduler as a unit of work to be allocated processor time [Liu and Chingand 1991]. The scheduling problem is to select a task set each of which meets the given deadline. Partial orderings of the tasks should be met in the resulting schedule. Related research has shown the NP-completeness of this problem [Liu and Chingand 1991; Garey and Johnson 1979]. *Task* corresponds to *method* that comprises the base model, which has been constructed from the model generation phase. The problem of finding the optimal abstraction level translates to the scheduling problem, which is to find a schedule for a set of methods that yields maximal quality for a given amount of time, while preserving the partial temporal orderings among the given methods. In the following sections, we define three approaches for optimal abstraction level selection. An optimal abstraction model is built based on the determined optimal abstraction degree.

6.1 IP (Integer Programming)- Based Selection

Operations research (OR) is a field that is concerned with deciding how to best design and operate systems under conditions requiring the allocation of scarce resources. Our optimal level selection problem falls under the OR umbrella, since the selection should be made for the best model that has an optimal abstraction level to simulate a given system under conditions requiring the allocation of scarce resources, such as time and accuracy. The essence of the OR activity lies in the construction and use of the mathematical models. The term *linear programming* defines a particular class of programming problems when the problem is defined by a linear function of the decision variables (referred to as the *objective function*), and the operating rules governing the process can be expressed as a set of linear equations or linear inequalities (referred to as the *constraint set*). IP refers to the

class of linear programming problems wherein some or all of the decision variables are restricted to integers [Ravindran and Solberg 1987]. *Pure integer programming* is a category where all the decision variables are restricted to be integer values. Our problem is a special case of integer programming, where the decision variables take *binary values* to indicate whether or not to select the examining node of a given AT for the behavioral abstraction. We formulate the problem as two IP models: *IP1* and *IP2*. A simple example is given to analyze the IP models.

6.1.1 *Formulation of IP-Based Selection.* Let a binary integer variable L_{ij} denote the decision to select or not to select the node I_{ij} for the behavioral abstraction.

$$L_{ij} = \begin{cases} 1 \text{ if behavioral abstraction occurs at the } I_{ij} \text{ node} \\ 0 \text{ otherwise} \end{cases}$$

Then, the objective function for *IP1* selection is defined in equation 1

$$Minimize \sum_{i=1}^{l} \sum_{j=1}^{n_i} L_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{l} \sum_{j=1}^{n_i} a_{ij} L_{ij} \ \leq \ a_c \tag{2}$$

$$\sum_{i=1}^{l} \sum_{j=1}^{n_i} t_{ij} L_{ij} \ \geq \ t_c \tag{3}$$

and, for each parent node L_{ik} of a given AT

$$\sum_{k=1}^{n_i} L_{i+1,k} \leq n_i(1 - L_{ik}), \text{ for each } i \tag{4}$$

where l is the maximum level of the AT and n_i is the number of nodes at level i. a_{ij} represents the accuracy loss and t_{ij} represents the expected duration. a_c defines the accuracy constraint given to the system, while t_c is the amount of desired speedup to meet a given deadline, D. Therefore, t_c = *execution time of the base model* - D.

The objective function of *IP1* reflects the fact that the smaller number of behavioral abstraction methods is desirable as long as the resulting model satisfies the given time deadline and the accuracy constraint.

The objective function for *IP2* selection is defined in equation 5

$$Minimize \sum_{i=1}^{l} \sum_{j=1}^{n_i} a_{ij} L_{ij} \tag{5}$$

subject to

$$\sum_{i=1}^{l} \sum_{j=1}^{n_i} t_{ij} L_{ij} \ \geq \ t_c \tag{6}$$

and, for each parent node L_{ik} of a given AT

$$\sum_{k=1}^{n_i} L_{i+1,k} \leq n_i(1 - L_{ik}), \text{ for each } i \tag{7}$$

The objective function of *IP2* minimizes the quality loss while satisfying the timing constraint defined in equation 6. *IP2* does not minimize the number of behavioral abstraction methods as long as the resulting model minimizes the accuracy loss. For instance, if a model, $\mathcal{A}$, that cuts out three structural components, is expected to produce more accurate results than a model, $\mathcal{B}$, that cuts out only one structural component , *IP2* keeps $\mathcal{A}$ for a candidate of the optimal abstraction model.

6.1.2 *Analysis of IP-Based Selection.* Consider AT in Figure 2. We associate each I_{ij} node with a binary variable L_{ij} discussed in the previous section. Then the objective function of *IP1* for a given AT is defined as :

$$Minimize\ (L_{11} + L_{21} + L_{22} + L_{31} + L_{32} + L_{33} + L_{34} + L_{35}) \tag{8}$$

For simplicity, we assume that any behavioral abstraction method takes 20 units. θ is assumed to be 0, which means real-time simulation will be performed in the same platform with which the expected duration has been measured. Then, (t_{11}, t_{21}, t_{22}, t_{31}, t_{32}, t_{33}, t_{34}, t_{35}) is defined as (220, 110, 90, 10, 40, 20, 20, 50), respectively.

For the simplicity of illustration, we consider only QA(I_{ij}) to assess the quality loss, a_{ij}, when the corresponding node I_{ij} is selected for the behavioral abstraction. Then, a_{ij} is simplified as follows:

$$a_{ij} = \frac{C_{ij} + \sum_{k=1}^{ic} C_{i+1,k}}{N}$$

where C_{ij} is the number of children that L_{ij} has. Since the behavioral abstraction at this level discards all the structural information of the lower levels, we believe that the accuracy loss is proportional to the number of descendants that a node has. The right hand side of a_{ij} is to find out the number of descendants that node L_{ij} has. For a given AT in Figure 2, (a_{11}, a_{21}, a_{22}, a_{31}, a_{32}, a_{33}, a_{34}, a_{35}) is defined as (7/8, 3/8/, 2/8, 1/8, 1/8, 1/8, 1/8, 1/8), respectively.

For a given accuracy loss, a_c, the accuracy constraint of *IP1* is defined as :

$$(a_{11}L_{11} + a_{21}L_{21} + a_{22}L_{22} + a_{31}L_{31} + a_{32}L_{32} + a_{33}L_{33} + a_{34}L_{34} + a_{35}L_{35}) \leq a_c \tag{9}$$

Also, for a given deadline, t_c, the speedup constraint is defined as :

$$(t_{11}L_{11} + t_{21}L_{21} + t_{22}L_{22} + t_{31}L_{31} + t_{32}L_{32} + a_{33}L_{33} + a_{34}L_{34} + a_{35}L_{35}) \geq t_c \tag{10}$$

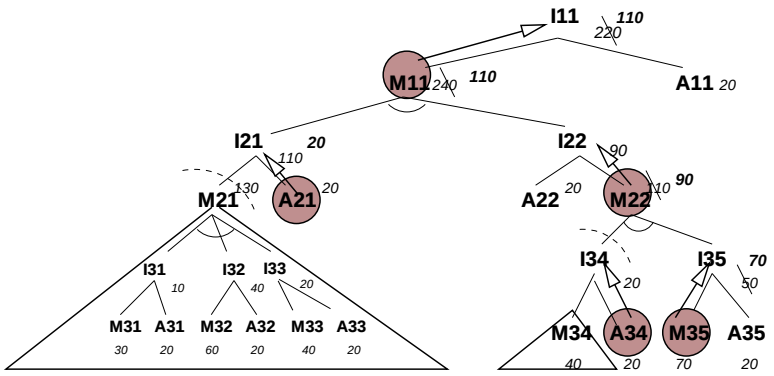


Fig. 3. Optimal abstraction tree decided from the integer programming approach

To define *parent and child* relationships in the given AT, we have a set of equations for all the child nodes of the given AT.

$$\begin{aligned} L_{31} + L_{32} + L_{33} &\geq 3(1 - L_{21}) \\ L_{34} + L_{35} &\geq 2(1 - L_{22}) \\ L_{21} + L_{22} &\geq 2(1 - L_{11}) \end{aligned} \tag{11}$$

Then, the IP1 selection of the optimal abstraction level for a given AT is to solve the objective function defined in equation 8 subject to the constraints defined in equations 9- 11.

Figure 3 shows the result when we have 120 units for a deadline ($t_c = 120$) and (50%) for an accuracy constraint ($a_c = 0.5$). The resulting AT concludes that applying behavioral abstraction at I_{21} and I_{34} gives the optimal abstraction level which simulates the system within a given deadline and the accuracy constraint with the minimum loss of the structural information. I_{21} node is executed by its behavioral abstraction method, A_{21}. Also, I_{34} node is executed by its behavioral abstraction method, A_{34}, instead of the high resolution method, M_{34}. Other intermediate nodes (I_{35}, I_{22}, I_{11}) are executed by the high resolution methods. The intermediate nodes, I_{31}, I_{32}, and I_{33} are not considered , since the behavioral abstraction occurs at the parent node, I_{21}, as defined in equation 11. The execution time of the AT is saved by the speedup amount that A_{21} and A_{34} yield. The execution of *IP2* is same to *IP1* with the objective function and the time constraint defined as :

$$Minimize(a_{11}L_{11} + a_{21}L_{21} + a_{22}L_{22} + a_{31}L_{31} + a_{32}L_{32} + a_{33}L_{33} + a_{34}L_{34} + a_{35}L_{35})$$

subject to

$$(t_{11}L_{11} + t_{21}L_{21} + t_{22}L_{22} + t_{31}L_{31} + t_{32}L_{32} + a_{33}L_{33} + a_{34}L_{34} + a_{35}L_{35}) \geq t_c$$

6.2 Search Based Selection

Our hypothesis is that *as we need tighter time constraints, we tend to employ more behavioral abstraction methods.* We increase the number of behavioral abstraction

Algorithm 1. Optimal Abstraction Level Selection

```
1:  nodes ⇐ at.ConstructAT(fid)
2:  baseCost ⇐ nodes[0].getCost()
3:  if baseCost ≤ deadline then
4:    return(0)
5:  endif
6:  at.CollectOrNodes(nodes,orNodes)
7:  size ⇐ at.SelectOneByDeadline(orNodes,deadline)
8:  if size > 0 then
9:    at.SelectByAccuracy(orNodes)
10: OptimalAbstraction ⇐ orNodes[0].getName()
11: else
12:   degree ⇐ OptimalAbsNumber(orNodes,deadline,baseCost)
13:   if degree == -1 then
14:     return(-1)
15:   else
16:     OptimalSet ⇐ OptimalCombination(ornodes,deadline,baseCost,degree)
17:     OptimalCost ⇐ OptimalSet.cost
18:     OptimalQualityLoss ⇐ OptimalSet.qualityloss
19:   end if
20: end if
21: return(0)
```

methods as we require more stringent deadlines for the heuristics. The selection algorithm starts from one behavioral abstraction. If this abstraction satisfies the time constraint, we stop and do not go further to examine other possibilities, with the hope that increasing the number of behavioral abstraction methods will result only in a less accurate model. If the time cannot be met by one behavioral abstraction method, we examine how many behavioral abstraction methods will be needed for a given time constraint. This is done by examining r fast behavioral abstraction methods. If combining r fast behavioral abstraction methods satisfies the time constraint, then the optimal abstraction level will be determined by r behavioral abstraction methods. At this point, we start to pick r behavioral abstraction functions until the most accurate combination is found while still satisfying the given time constraint. Algorithm 1 shows the overall method in detail.

This algorithm reads abstraction information about a given base model. The information contains methods, parent/child relationships between the nodes, and duration/quality information for each method. Based on the given information, the algorithm constructs an AT as in Line 1. The execution time of the base model that has no behavioral abstraction methods is calculated in Line 2. Lines 3 - 5 examine whether we need to employ behavioral abstraction methods to meet a given deadline. If the calculated duration of the base model is less than equal to the given deadline, we don't have to employ behavioral abstraction methods; thus, the algorithm terminates. If the duration of the base model is greater than the given deadline, then it becomes necessary to use behavioral abstraction methods. Upon recognizing the necessity of behavioral abstraction methods, the algorithm collects *OR* nodes that contain the information about the behavioral abstraction methods and then starts to increase the number of behavioral abstraction methods. Line 8 examines whether one behavioral abstraction method will resolve the timeliness

requirement. If the returned *size* is greater than 0, as in Line 9, we know that *one* behavioral abstraction is enough to meet a given deadline. Then, the algorithm looks up the most accurate method to ensure that the selected method will have the best quality while satisfying the given deadline. If one behavioral abstraction is not enough to achieve the given deadline ($size \leq 0$), the algorithm determines how many behavioral abstraction methods can achieve the given deadline. We know that behavioral abstraction methods will bring more savings to the execution time of the resulting model. Our objective is to minimize the use of behavioral abstraction methods as long as the resulting model meets the given deadline. A simple method defining *degree* in line 12 is :

(1) $i = 2$

(2) select i behavioral methods that will bring the maximum time savings to the given base model;

(3) if the use of i behavioral methods cannot resolve the required speedup, increase i by 1 and go to step 2;

At this point, the algorithm knows how many behavioral abstraction methods will be needed for a given deadline. If the returned *degree* is -1, it means the given deadline cannot be met even if we use all available behavioral abstraction methods. Lines 16 - 18 look for the best combination that will lead to the most accurate model. The algorithm examines all nCr combinations, where n represents the number of behavioral abstractions available to the given base model and r is the calculated *degree* in Line 13.

6.3 Experiments

We implemented the proposed integer programming solutions with *solver* on Excel. For a small problem space as in the case of Figure 2, *solver* of Excel might be a good choice. However, if the problem size is large, we can use CPLEX [CPLEX 1995] which is an optimization callable library designed for large scale problems.

For the exact solution method, we use the *branch and bound method*, which is a classical approach to find exact integer solutions. The *branch and bound method* is an efficient enumeration procedure for examining all possible integer feasible solutions [Ravindran and Solberg 1987; CPLEX 1995]. Through the *branch and bound method*, the binary variable L_{ij} is either 0 or 1. Table 1 shows some results from the experiments of *IP1, IP2* and search-based approach.

When the base model meets a given deadline, as in case 1, no behavioral abstraction is suggested. Also, when a given deadline is immediate, the entire AT is behaviorally abstracted into one method as in case 12. Other cases employ one or two behavioral abstraction methods to meet a given accuracy constraint while satisfying a time constraint. Note that *IP2* selects equal or more number of nodes comparing to *IP1*. Case 7 shows that the objective of *IP1* is to meet both accuracy and time constraint, while minimizing the loss of structural abstraction methods. Case 11 shows that the objective of *IP2* is to minimize the expected quality loss rather than to minimize the loss of structural abstraction methods.

Given an optimal abstraction level determined by the selection algorithms, the *Optimal Abstraction Model Composer* looks at the method names which comprise

Table 1. Experiment results from *IP1*, *IP2* and search-based approach

No	Deadline	a_c	t_c	*IP1*	*IP2*	Search
1	240	0.5(50%)	0	n/a	n/a	n/a
2	220	0.5(50%)	20	L_{34} $a_c = 0.125(13\%)$ deadline = 220	L_{34} $a_c = 0.125(13\%)$ deadline = 220	L_{34} $a_c = 0.125(13\%)$ deadline = 220
3	200	0.5(50%)	40	L_{35} $a_c = 0.125(13\%)$ deadline = 190	L_{35} a_c =0.125(13%) deadline = 190	L_{35} a_c =0.125(13%) deadline = 190
4	180	0.5(50%)	60	L_{22} $a_c = 0.25(25\%)$ deadline = 150	L_{32}, L_{34} a_c =0.25(25%) deadline = 180	L_{22} $a_c = 0.25(25\%)$ deadline = 150
5	160	0.5(50%)	80	L_{22} $a_c = 0.25(25\%)$ deadline = 150	L_{22} $a_c = 0.25(25\%)$ deadline = 150	L_{22} $a_c = 0.25(25\%)$ deadline = 150
6	140	0.5(50%)	100	L_{21} $a_c = 0.375(38\%)$ deadline = 130	L_{22}, L_{33}, L_{35} $a_c = 0.375(38\%)$ deadline = 80	L_{21} $a_c = 0.375(38\%)$ deadline = 130
7	120	0.5(50%)	120	L_{21}, L_{34} $a_c = 0.5(50\%)$ deadline = 110	L_{22}, L_{32} $a_c = 0.375(38\%)$ deadline = 110	L_{22}, L_{32} $a_c = 0.375(38\%)$ deadline = 110
8	100	0.5(50%)	140	L_{21}, L_{35} $a_c = 0.5(50\%)$ deadline = 80	L_{22}, L_{32}, L_{33} $a_c = 0.5(50\%)$ deadline = 90	L_{21}, L_{35} $a_c = 0.5(50\%)$ deadline = 80
9	80	0.5(50%)	160	L_{21}, L_{35} $a_c = 0.5(50\%)$ deadline = 80	L_{21}, L_{35} $a_c = 0.5(50\%)$ deadline = 80	L_{21}, L_{35} $a_c = 0.5(50\%)$ deadline = 80
10	60	0.7(70%)	180	L_{21}, L_{22} $a_c = 0.625(63\%)$ deadline = 40	L_{21}, L_{22} $a_c = 0.625(63\%)$ deadline = 40	L_{21}, L_{22} $a_c = 0.625(63\%)$ deadline = 40
11	40	0.9(90%)	200	L_{11} $a_c = 0.875(88\%)$ deadline = 20	L_{21}, L_{22} $a_c = 0.625(63\%)$ deadline = 40	L_{11} $a_c = 0.875(88\%)$ deadline = 20
12	20	0.9(90%)	220	L_{11} $a_c = 0.875(88\%)$ deadline = 20	L_{11} $a_c = 0.875(88\%)$ deadline = 20	L_{11} $a_c = 0.875(88\%)$ deadline = 20

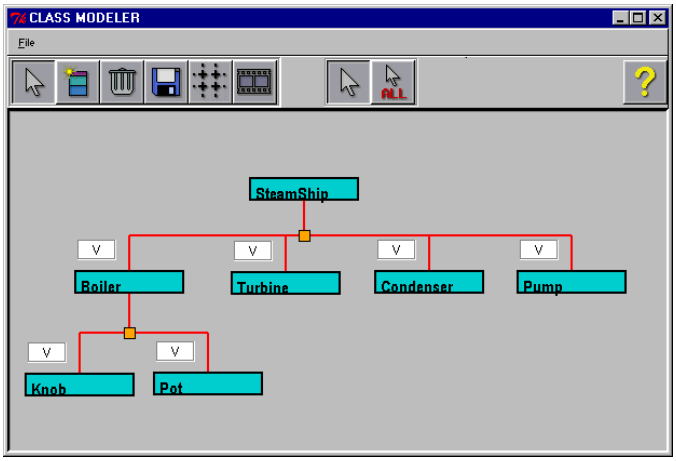


Fig. 4. Conceptual model of FULTON: FULTON is modeled within OOPM Conceptual model is designed in terms of classes, attributes, methods (dynamic method and static method) and relationships of classes (inheritance and composition)

the optimal abstraction level for a given AT. The optimal abstraction model is composed by observing the partial temporal orderings of the selected methods.

7. FULTON EXAMPLE

Consider a steam-powered propulsion ship model, named FULTON. In FULTON, the furnace heats water in boiler: when the fuel valve is OPEN, fuel flows and the furnace heats; when fuel valve is CLOSED, no fuel flows and the furnace is at ambient temperature. Heat from the furnace is added to the water to form high-pressure steam. The high-pressure steam enters the turbine and performs work by expanding against the turbine blades. After the high-pressure steam is exhausted in the turbine, it enters the condenser and is condensed again into liquid by circulating sea water [Gettys and Keller 1989]. At that point, the water can be pumped back to the boiler.

7.1 Model Generation

A conceptual model is constructed on OOPM. It is designed in terms of classes, attributes, methods, and relationships between classes (inheritance and composition). Figure 4 shows the class hierarchy of FULTON, which follows the physical composition of a steamship. Classes are connected by a *composition* relationship as denoted by the rectangular boxes in Figure 4. *V* denoted in the white box specifies 1 for the cardinality of the associated class. In Figure 4, *Ship* **has** *a Boiler*, *a Turbine*, *a Condenser*, and *a Pump*. Class *Boiler* **has** *a Pot* and *a Knob*. Each class has attributes and methods to specify its dynamic behaviors.

7.1.1 *Structural Abstraction of FULTON.* Figures 5, 6, and 7 show structural abstractions of FULTON. Since FULTON can be configured with 4 distinct physical components and a functional directionality, we start with FBM. The FBM is located in class *Ship*, as shown in Figure 5. Figure 5 has 4 blocks: L_1 for the function of

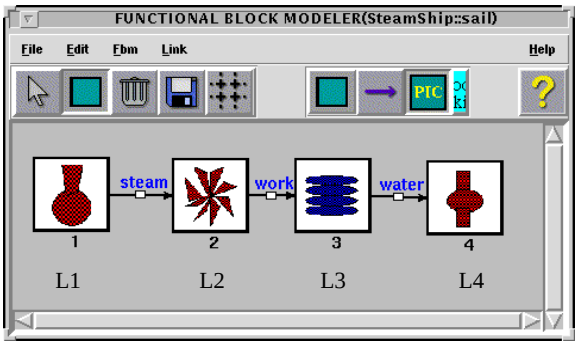


Fig. 5. Top level : structural abstraction for FULTON

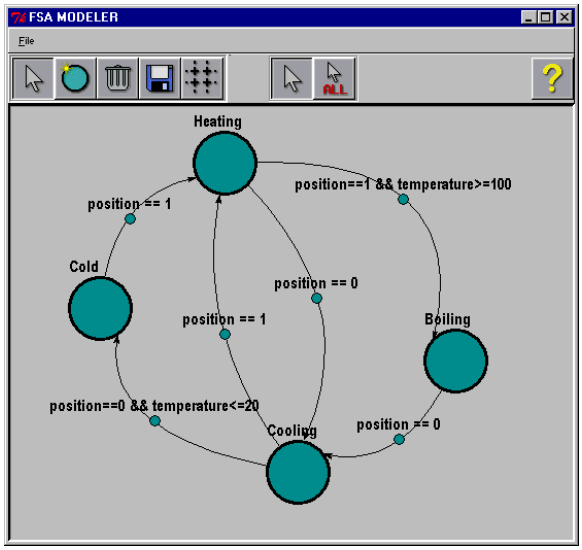


Fig. 6. Structural abstraction of M_7: FSM has 4 states (Cold ($M13$), Cooling ($M14$), Heating (M15) and Boiling (M16))

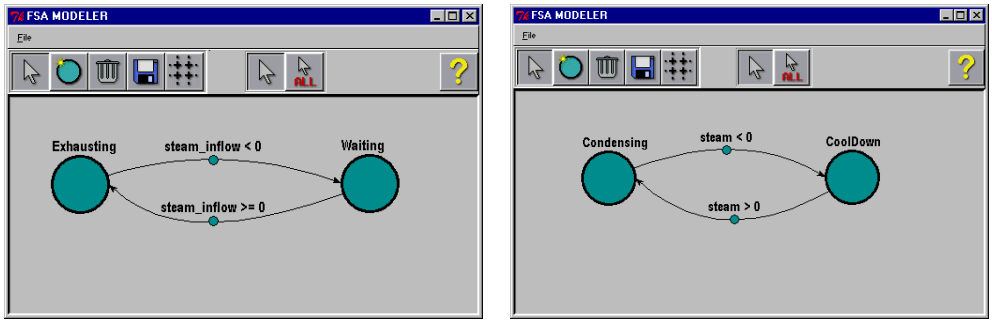
a boiler, L_2 for turbine, L_3 for condenser, and L_4 for pump. Boiler assembly (L_1) has distinct states according to the temperature of the water. L_1 is refined into :

(1) B_1 : method of class $Knob$, $M6$, which provides fuel to the boiler

(2) B_2 : FSM, M_7, in Figure 6, which determines the temperature of the boiler and makes state transitions according to the temperature

(3) B_3 : provides high-pressure steam, defined in a CODE method, $M8$

Each state of Figure 6 (Cold (M13), Heating (M14), Boiling (M15), and Cooling (M16)) is refined into an algebraic equation, which calculates the temperature based on the position of the knob (Open, Closed). Each state of B_2 is refined into a *CODE* method that defines the temperature equations with C++ syntax.



(a) Structural abstraction of M_{10} (b) Structural abstraction of M_{12}

Fig. 7. Structural abstraction of M_{10} and M_{12}

L_2 is refined into two functional blocks: $M9$ and $M10$. $M9$ gets the high pressure steam from the boiler. $M10$ is decomposed into two temporal phases: *Exhausting* ($M17$) and *Waiting* ($M18$). If there is no steam to exhaust, $M10$ resides in the waiting state. Otherwise, $M10$ exhausts steam to generate the work of the steamship. Figure 7 shows the FSM of the turbine. L_3 is also refined into two functional blocks: $M11$ and $M12$. $M11$ gets the exhausted steam from the turbine. $M12$ has two distinct temporal phases: *Condensing* ($M19$) and *Cooldown* ($M20$), in Figure 7. *Condenser* decreases the temperature in *Cooldown* state, waiting for the turbine to send more steam. Otherwise, $M12$ resides in *Condensing* state where the steam from the turbine turns into liquid again.

7.1.2 *Behavioral Abstraction of FULTON.* We start with the observed data set of (input, output) from the simulation of the base model. With this *prior* knowledge, the method of behavioral abstraction is to generate a C++ procedure which encodes the input/output functional relationship using a neural network model (MADALINE, Backpropagation) or a Box-Jenkins model.

We abstract the multimodel method of $M7$, $M10$ and $M12$ with Box-Jenkins models. Given three behavioral abstraction methods for $M7, M10$ and $M12$, 8 (2^3) new models can be generated with different degrees of abstraction. Table. 2 shows the possible combinations of the behavioral abstraction methods. For example, $C6$ uses two behavioral abstraction methods for $M7$ and $M12$. Therefore, the structural information associated with $M7$ and $M12$ methods, which are both FSMs, are abstracted.

When modelers create a behavioral abstraction method, they pick the dynamic function to abstract. Figure 8 shows the Box-Jenkins abstraction process for $M7$. Based on the learning parameters of the Box-Jenkins, we learn the input/output functional relationship of M7. Once the performance of the Box-Jenkins model is accurate enough, we generate a behavioral abstraction method based on the resulting weight and offset vector.

Table 2. New multimodels of FULTON with behaviorally abstracted component(s): A capital letter represents a full-resolution model, while a small letter represents a low-resolution model. The low resolution model is generated through behavioral abstraction

no	Model	Abstracted Method	Abstracted Component
C_1	BTCP	N/A	N/A
C_2	BTcP	$M12$	Condenser
C_3	BtCP	$M10$	Turbine
C_4	BtcP	$M10$, $M12$	Turbine, Condenser
C_5	bTCP	$M7$	Boiler
C_6	bTcP	$M7$, $M12$	Boiler, Condenser
C_7	btCP	$M7$, $M10$,	Boiler, Turbine
C_8	btcP	$M7$, $M10$, $M12$	Boiler, Turbine, Condenser

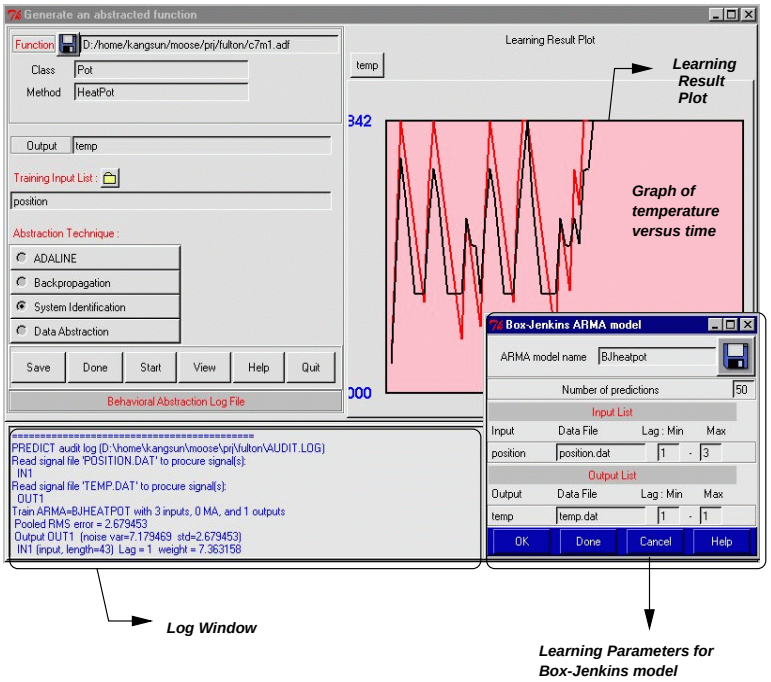
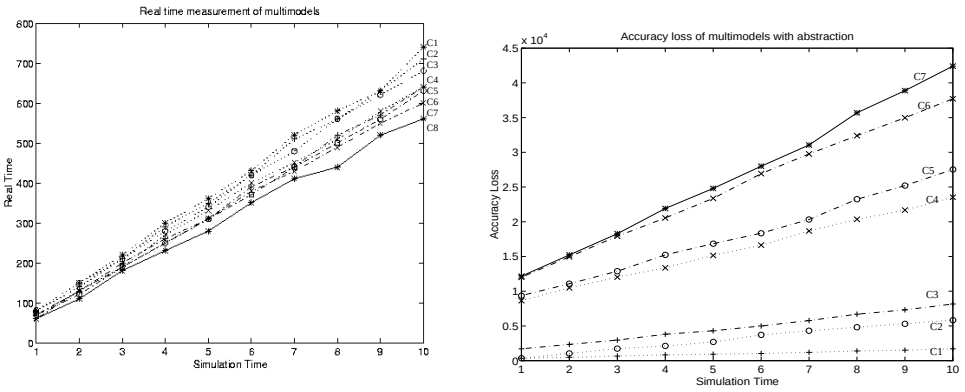


Fig. 8. Behavioral abstraction : The user selects a dynamic function to abstract, Pot::HeatPot, which is an FSM. States and their transitions will be lost, but behavioral information will be preserved to some level of fidelity. In the learning process, the user gives several parameters, for example, *lag* for input and output variables



(a) Execution time of multimodels

(b) Accuracy loss of multimodels

Fig. 9. Execution time/ accuracy loss of models: Behavioral abstraction yields shorter elapsed times. As we increase the level of abstraction by using more behavioral abstraction methods, the model execution time decreases. Accuracy is lost as we increase the number of behavioral abstraction methods in the base model

7.2 Construction of the Abstraction Tree

The model sets in Table 2 provide alternatives that vary in the expected accuracy and in the computation time; it thus allows us to investigate the effects of model structure on model accuracy and on computation time. We measured the execution time of each model by varying the simulation logical clock from 50 to 500 using a time step of 50. As shown in Figure 9, the most detailed model, C_1, takes the longest time, and the least detailed model, C_8, runs faster than the other models.

The cumulative accuracy loss of each model is also measured in Figure 9. The accuracy loss of the Box-Jenkins model is measured by examining the sum of the squared error through the Box-Jenkins testing process. As simulation proceeds, the cumulative accuracy loss increases for each model. The least detailed model, C_8, has the maximum accuracy loss, while C_2 shows the minimum accuracy loss over time.

Figure 10 shows the AT of FULTON. We applied the Box-Jenkins behavioral abstraction technique to three methods, ($M7$, $M10$, $M12$), and produced ($A7$, $A10$, $A12$), respectively. I_i node is positioned where the two associated different resolution methods reside. Intermediate nodes I_i are connected to the children by the *OR* relationship.

The execution time of leaf methods can be measured by extensive experimentation [Lark, J. S., Lee D. Erman, Stephanie Forrest and Kim P. Gostelow 1990], or assessed by available execution time prediction techniques [Marin G. Harmon and Walley 1994; Horst F. Wedde and Huizinga 1994]. To simplify the illustration, we assume that the execution time of each leaf method ($M6$, $M13$, $M14$, $M15$, $M16$, $M8$, $M9$, $M17$, $M18$, $M11$, $M19$, $M20$, $M5$) in the AT is properly assessed to (2,2,8,4,6,2,2,4,3,2,2,4,2), respectively by using the available analytic tools. Also,

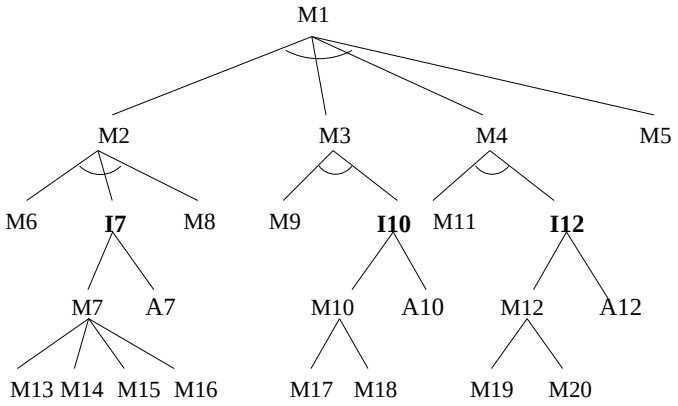


Fig. 10. Abstraction Tree of the FULTON example: Intermediate node I_i is introduced to connect the high resolution method M_i and low resolution method (behavioral abstraction method), A_i

we assume that (A_7, A_{10}, A_{12}) takes (4,2,2), respectively. Non-leaf nodes iteratively look for child's execution time and calculate its own execution time by applying the execution time assessment equations discussed in Section 5. System factor, θ, is assumed to be 0; therefore, the target platform in which the model will be executed is the same one in which the model execution time is measured. We assume that there is no special interesting class for the simulation. The precision loss of A_7 is assumed to be 0.4, while (A_{10}, A_{12}) is assumed to be (0.1, 0.2), respectively. Then, the quality loss of each method, a_7, a_{10}, and a_{12} is defined by $0.325, 0.125$, and 0.175, respectively.

7.3 Selection of the Optimal Abstraction Model

The base model of FULTON takes 26 units to complete the simulation. Suppose we have 20 units for a deadline. Upon receiving the time constraint, we immediately know that the behavioral abstraction is needed to make the simulation faster. The optimal abstraction level is determined by *IP1, IP2* and the search-based algorithm discussed in Section 6.2.

For a given AT in Figure 10, the objective function of the *IP1* is defined as:

$$Minimize\ (I_7 + I_{10} + I_{12}) \tag{12}$$

subject to

$$\begin{aligned} & a_7 I_7 + a_{10} I_{10} + a_{12} I_{12} \leq a_c \\ & t_7 I_7 + t_{10} I_{10} + t_{12} I_{12} \geq t_c \\ & a_7 = 0.375,\ a_{10} = 0.125,\ a_{12} = 0.175 \\ & t_7 = 4,\ t_{10} = 2,\ t_{12} = 2 \end{aligned} \tag{13}$$

Then, the *IP1* selection of the optimal abstraction level is to solve the objective function defined in equation 12 with the constraints defined in equation 13. Since the desired speedup to be achieved for a given deadline is $26 - 20 = 6$, we assign 6 to t_c. To find out the most accurate combination, we assign 1.0 to a_c. Therefore, the accuracy is not constrained to a certain bound.

The objective function of the *IP2* approach is defined as :

$$Minimize\ (I_7 * a_7 + I_{10} * a_{10} + I_{12} * a_{12}) \tag{14}$$

subject to

$$\begin{aligned} & t_7 I_7 + t_{10} I_{10} + t_{12} I_{12} \geq t_c \\ & a_7 = 0.375, a_{10} = 0.125, a_{12} = 0.175 \\ & t_7 = 4, t_{10} = 2, t_{12} = 2 \end{aligned} \tag{15}$$

Then, the *IP2* selection of the optimal abstraction level is to solve the objective function defined in Equation 14 with the constraints defined in Equation 15.

The search-based algorithm increases the number of behavioral abstraction methods to be used for the deadline. The algorithm examines whether one behavioral abstraction method will resolve the time constraint. Neither of the candidates meets the deadline. Therefore, the algorithm increases the number of behavioral abstraction methods to use for the simulation. The fastest behavioral abstraction A_7 achieves the deadline if either of A_{10} or A_{12} is combined with A_7. Therefore, the algorithm concludes that using 2 behavioral abstraction methods will resolve the timeliness requirement. At this point, the algorithm starts to find the most accurate combination. (A_7, A_{10}) meets the deadline with the maximum accuracy. Therefore, the algorithm declares (A_7, A_{10}) as the optimal abstraction degree for a given AT and a deadline of 20. Figure 11 shows the optimal abstraction level of the given AT. The execution of (I_7, I_{10}, I_{12}) is made by (A_7, A_{10}, M_{12}), respectively. Then, the optimal abstraction model is composed of the sequence $(M_6, A_7, M_8, M_9, A_{10}, M_{11}, M_{12}, M_5)$. Note that the FSM models of *Boiler* and *Turbine* are cut off to save simulation time. The corresponding scheduling diagram is shown in Figure 12.

Table 3 shows other selection examples. *IP2* produces a different answer for a deadline of 22. *IP1* and the search-based methods minimize the number of behavioral abstraction methods in order to minimize the loss of structural information. When modelers want to minimize the loss of structural information (to preserve the base model structure as much as possible), behavioral abstraction occurs at I_7. However, if the simulation objective is to minimize the expected quality loss, we apply behavioral abstraction at I_{10} and I_{12}, as suggested from *IP2*.

8. CONCLUSIONS

We demonstrated a semi-automated methodology to build a model that is right for the simulation objective and real-time constraints. The key to our method is to use the model abstraction technique to generate multiple methods of the system

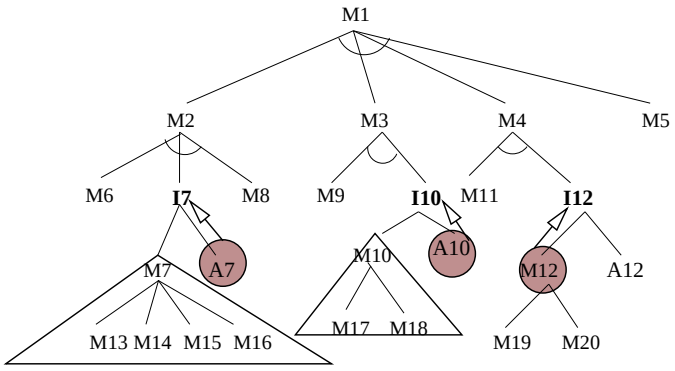


Fig. 11. Optimal abstraction level for a deadline of 20

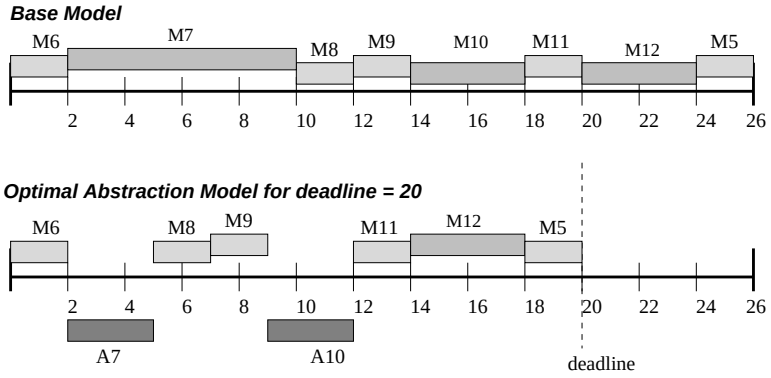


Fig. 12. Scheduling diagram for a deadline of 20

Table 3. Selection examples of three algorithms for FULTON

deadline	20	22	24
IP1	I_7, I_{10}	I_7	I_{10}
	$a_c = 0.5$	$a_c = 0.375$	$a_c = 0.175$
IP2	I_7, I_{10}	I_{10}, I_{12}	I_{10}
	$a_c = 0.5$	$a_c = 0.3$	$a_c = 0.175$
Search	I_7, I_{10}	I_7	I_{10}
	$a_c = 0.5$	$a_c = 0.375$	$a_c = 0.175$

which involve tradeoffs in runtime versus accuracy. Modelers construct the abstraction hierarchy through a structural abstraction phase, and we use the abstraction hierarchy for the source of information where the optimal abstraction degree is determined. By applying the proposed algorithms that determine the optimal abstraction level to simulate the system for a given deadline, we find position(s) where the behavioral abstraction technique is applied. Behavioral abstraction yields time savings of the simulation by discarding detailed structural information, though accuracy is sacrificed. The resulting model simulates the system at an optimal level of abstraction to satisfy the simulation objective for a given deadline so as to maximize the tradeoff of model execution time for accuracy. One of our assumptions was that quality and execution time of the abstraction methods are fairly predictable. Predicting the execution time is possible, in general, by using the available research on runtime estimation techniques; however, assessing the method's quality is difficult. Especially, when the result from a method decreases the quality of other methods, the estimation becomes more complicated. One of the possible solutions is to monitor the selected model's execution under the real-time simulation [Garvey and Lesser 1993b]. When the selected model takes longer time than expected, or the solution quality is lower than expected during the real-time simulation, it is reported to the monitor to take an action. Then, the appropriate actions can be taken to adjust the problems that have been caused by under-estimated/over-estimated duration or quality.

ACKNOWLEDGMENTS

We would like to thank the following funding sources that have contributed towards our study of modeling and implementation of the OOPM multimodeling simulation environment: (1) GRCI Incorporated 1812-96-20 (Gregg Liming) and Rome Laboratory (Steve Farr, Al Sisti) for web-based simulation and multimodeling; (2) NASA/Jet Propulsion Laboratory 961427 (John Peterson and Bill McLaughlin) for web-based modeling of spacecraft and mission design, and (3) Department of the Interior under ATLSS Project contract 14-45-0009-1544-154 (Don DeAngelis, University of Miami) for techniques for both code and model integration for the across-trophic-level Everglades ecosystem. Without their help and encouragement, our research would not be possible.

REFERENCES

Barr, A. and Feigenbaum, E. A. 1981. *The Handbook of Artificial Intelligence.* William Kaufmann.

Burns, A. and Wellings, A. J. 1994. Hrt-hood: A structured design method for hard real-time systems. *Real-Time Systems 6*, 73–114.

Caughlin, D. and Sisti, A. 1997. A summary of model abstraction techniques. In *Proceedings of SPIE97* (1997), pp. 2–13.

CPLEX. 1995. *Using the CPLEX Callable Library.* CPLEX Optimization, Inc.

Cubert, R. M. and Fishwick, P. A. 1997. Moose: An object-oriented multimodeling and simulation application framework. *Simulation 6.*

D'Ambrosio, B. 1989. Resource bounded-agents in an uncertain world. *IJCAI.*

Digital. 1989. *CASE for Real-Time Systems Symposium.* Digital Consulting, Andover, MA.

FISHWICK, P. A. 1995. *Simulation Model Design and Execution: Building Digital Worlds.* Prentice Hall.

FISHWICK, P. A. 1997. A Visual Object-Oriented Multimodeling Design Approach for Physical Modeling. *University of Florida Technical Report 9.*

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and intractability.* W.H. Freeman and company.

GARVEY, A. J. AND LESSER, V. R. 1993a. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics 23*, 6, 1491–1502.

GARVEY, A. J. AND LESSER, V. R. 1993b. A survey of research in deliberative real-time artificial intelligence. *UMass Computer Science Technical Report 93-84.*

GARVEY, A. J. AND LESSER, V. R. 1995. Design-to-time scheduling with uncertainty. *UMass Computer Science Technical Report 95-03.*

GETTYS, E. AND KELLER, F. 1989. *Physics.* McGraw-Hill.

HORST F. WEDDE, B. K. AND HUIZINGA, D. M. 1994. Formal timing analysis for distributed real-time programs. *Real-Time Systems 7*, 1, 58–90.

KOPETZ, H., ZAINLINGER, R., FOHLER, G., KANTZ, H., PUSCHNER, P. AND SCHUTZ, W. 1991. The design of real-time systems: from specification to implementation and verification. *Software Engineering 6*, 72–82.

KORF, R. E. 1990. Depth-limited search for real-time problem solving. *Real-Time Systems 2*, 7–24.

LARK, J. S., LEE D. ERMAN, STEPHANIE FORREST AND KIM P. GOSTELOW. 1990. Concepts, methods, and languages for building timely intelligent systems. *Real-Time Systems 2*, 127–148.

LEE, K. AND FISHWICK, P. A. 1996. Dynamic Model Abstraction. In *Proceedings of Winter Simulation Conference* (1996), pp. 764–771.

LEE, K. AND FISHWICK, P. A. 1997a. A semi-automated method for dynamic model abstraction. In *Proceedings of AeroSense 97* (1997), pp. 31–41.

LEE, K. AND FISHWICK, P. A. 1997b. A methodology for dynamic model abstraction. *Transactions of the society for computer simulation international 13*, 4, 217–229.

LEE, K. AND FISHWICK, P. A. 1998. Generation of multimodels and selection of the optimal abstraction level for real-time simulation. In *AeroSense 1998* (1998), pp. 164–175.

LIU, S. AND CHINGAND, Z. 1991. Algorithms for scheduling imprecise computations. *IEEE Computer 24*, 5, 58–68.

MARIN G. HARMON, T. B. AND WALLEY, D. B. 1994. A retargetable technique for predicting execution time of code segments. *Real-Time Systems 7*, 2, 130–159.

MASTERS, T. 1995. *Neural, Novel and Hybrid Algorithms for Time Series Prediction.* John Wiley and Sons, Inc.

RAMAMRITHAM, K. AND STANKOVIC, J. A. 1984. Dynamic task scheduling in distributed hard real-time systems. *IEEE Software 1*, 3, 65–75.

RAVINDRAN, D. P. AND SOLBERG, J. J. 1987. *Operations Research.* John Wiley and Sons.

RUTLEDGE, G. W. 1995. Dynamic selection of models. *Ph.D Dissertation, Department of Medical Information Sciences, Stanford University.*

STANKOVIC, J. A., RAMAMRITHAM, K. AND CHENG, S. 1985. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Transactions on Computers 34*, 12, 1130–1143.

WEISS, S. M. AND KAPOULEAS, I. 1989. An experimental comparison of pattern recognition, neural nets, and machine learning classification methods. In *In Proceedings of IJCAI-89* (1989), pp. 781–787.

ZEIGLER, B. P. 1976. *Theory of Modelling and Simulation.* John Wiley and Sons.

ZEIGLER, B. P. 1990. *Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems.* Academic Press.