Co-Simulation of Algebraically Coupled Dynamic Subsystems

by

Bei Gu

B.S., Mechanical Engineering
Shanghai Jiao Tong University, 1992

M.S., Mechanical Engineering
Arizona State University, 1997

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2001

Signature of Author_____
Department of Mechanical Engineering
August 25, 2001

Certified by_____
H. Harry Asada, Ford Professor of Mechanical Engineering
Thesis Supervisor

Accepted by_____
Ain A. Sonin
Chairman, Department Graduate Committee

Co-Simulation of Algebraically Coupled Dynamic Subsystems

By

Bei Gu

Submitted to the Department of Mechanical Engineering
on August 24, 2001 in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy in
Mechanical Engineering

ABSTRACT

In the manufacturing industry out-sourcing/integration is becoming an important business pattern (not a clear statement-integration still done in house—component design and manufacturing outsourced). An engineering system often consists of many subsystems supplied by different companies. Bridge between thoughts is weak. Object-oriented modeling is an effective tool for modeling of complex coupled systems. However, subsystem models have to be assembled and compiled before they can produce simulation results for the coupled system. Compiling models into simulations? is time consuming and often requires a profound understanding of the models. Also, the subsystem makers cannot preserve their proprietary information in the compilation process. This research is intended to address this problem by extending object-oriented modeling to object-oriented simulation called co-simulation. Co-Simulation is an environment in which we can simultaneously run multiple independent compiled simulators to simulate a large coupled system.

This research studies a major challenge of object-oriented simulation: incompatible boundary conditions between subsystem simulators caused by causal conflicts. The incompatible boundary condition is treated as an algebraic constraint. The high index of the algebraic constraint is reduced by defining a sliding manifold, which is enforced by a discrete-time sliding mode controller. The discrete-time approach fits well with the numerical simulation since it can guarantee numerical stability. A Boundary Condition Coordinator (BCC), which implements the discrete-time controller, makes the incompatible boundary condition compatible. Multi-rate sliding controllers are developed to guarantee the stability of the sliding manifold with any integration step size for the subsystem simulators. A multi-rate sliding mode scheme is specially devised to minimize information disclosure from the subsystem simulators and to facilitate pure numerical computation. The influence of the BCCs on the rest of the subsystem simulators is studied using the input-output linearization theory.

The Co-Simulation software environment is developed in Java. Subsystem simulators and BCCs run as independent processes in the Co-Simulation environment. Class templates containing all necessary functions for different types of subsystems are defined. Engineers can easily build a subsystem simulator by simply providing only the mathematical model, which will be hidden after the subsystem simulator is made. Integration engineers can assemble subsystem simulators into simulation of the large coupled system by merely making connections among subsystems. The object-oriented class design makes it possible to extend the Co-Simulation over the Internet or to compile subsystems into a single thread simulator.

Thesis Supervisor: H. Harry Asada
Title: Ford Professor of Mechanical Engineering

To Mom and Dad

Acknowledgements

I would like to express my sincere gratitude to my advisor, Professor H. Harry Asada, for his intellectual guidance, constant encouragement, and wide-ranging support. I would like to thank my committee members, Professors Kamal Youcef-Toumi and David R. Wallace, for their inspiring advice, and Dr. Sheng Liu for his invaluable guidance.

Our group is always a fun and dynamic place. I will remember Booho, Brandon, Carolyn, Danielle, Peter, Phil, Steve, Xiangdong, Yi, Yong and other members of the d'Arbeloff Laboratory for Information Systems and Technology for their help and friendship.

My family has been a great source of love and encouragement. My having the opportunity to write this thesis results from what I have learned from my parents. Most of all, thanks to my wife Stephanie. I will not be able to finish my research project without Stephanie's support.

Finally, I wish to thank my sponsors Daikin Industries, Home Automation and Healthcare Consortium, and Matsushita Electric Industrial Company.

Table of Contents

List of Figures and Tables

1 Introduction

1.1 Motivation

People have developed many modeling and simulation technologies for different purposes over the years [Sinha et al, 2001]. System modeling is an important tool used by engineers for studying and designing engineering systems. Simulation then makes it possible for engineers to test the engineering system in the virtual realm before building the hardware. Modeling and simulation are closely related. The model serves as the basis for the simulation while simulation computes the prediction of the model. Modeling and simulation reduce both the time and the cost associated with the product development.

Object-oriented modeling approaches were first introduced in the late 1970s to deal with complex and multi-disciplinary systems [Otter and Cellier, 1996]. Among many object-oriented properties, modularity is the most important. Reusable subsystem models, which are developed by various experts, can be assembled to represent different large system models. The typical procedure of simulation is as follows: modeling of subsystems, assembling subsystem models, compiling the overall system model and solving of compiled system model.

Use of simulators is not only a powerful engineering methodology for design and analysis but also an important tool with which engineers communicate to each other. In addition to traditional specifications and data sheets, engineers now communicate by exchanging simulators representing the whole behavior of the components and systems they develop. In the automobile industry, for example, carmakers request that suppliers provide simulators of supply parts, and evaluate them by connecting the supply part simulators to the engine simulator and body simulator of the automobile. In turn, carmakers provide the suppliers with simulators depicting the conditions of the automobile system so that the supplier can develop the right parts to meet the specifications. Today's carmakers can communicate with thousands of suppliers through a supply chain management system over the Internet. In the air conditioner industry, former competitors are now forming an alliance to use common components and integrate their products. Simulators representing detailed behavior of individual units are exchanged to streamline communications among engineers of alliance partners and complete thorough engineering analysis and product development in a limited time.

However, current object-oriented modeling based simulation method has limitations:

- It requires engineers to go through all of the compilation and solution procedure before the simulation results of the coupled system can be obtained;
- Some component makers may want to keep their intellectual property private, yet providing models of their components exposed many details of their design;
- Different components of a coupled system may require quite different solution methods which may further complicate the process of converting models to simulations.

Extending the concept of modularity a step further, i.e., using objects of subsystem simulators instead of objects of subsystem models, a new simulation environment is proposed. Termed as Co-Simulation, it is software environment for simultaneously running a collection of subsystem simulators. When subsystem simulators are connected, simulators will produce the simulation result of the large system without going through the complex procedure of compilation and of numerical solution. Subsystem component simulators coded by different suppliers will interact with each other using predefined inputs and outputs. Simulators can then work together if they have compatible inputs and outputs. The internal implementation of the simulators is solely decided by their designers. The system integrator only needs to know the interface of the subsystem simulators. Therefore, the aforementioned three drawbacks of the object-oriented modeling can be eliminated.

The proposed object-oriented simulation technology will be an essential tool to facilitate upstream and downstream cooperation along the supply chain. The downstream companies will be able to test the integrated system with component simulators supplied by the upstream companies. The component makers also will test their products to find how their components work with other components by running simulators of all of the components together. In this way, engineers from the entire supply chain can communicate by co-running their subsystem simulators. Ultimately, simulators become the modern specification sheets for any component, but with a much richer array of available information.

1.2 Thesis contribution

Unlike object-oriented modeling, which uses a declarative approach to describe the dynamic system, a simulation must use a procedural approach to obtain the numerical results. The procedural approach defines the relation between dependent variables and independent variables through assignments. Assignments are basically causal definitions. The declarative approach

defines the relationships between variables by using equations without causal information. Simulators are basically numerical solvers for certain models. Numerical algorithms are then usually realized by procedural computer programs. Therefore, when simulators are assembled to represent a large coupled system, causal conflicts may occur due to the fixed causal assignments. Causal conflict is therefore the major challenge of the object-oriented simulation.

An incompatible boundary condition resulting from the causal conflict is modeled as an algebraic constraint. The coupled system is considered to be a control problem and the algebraic constraint is treated as the desired trajectory. Thus the system described by the high index Differential Algebraic Equation (DAE) becomes a high relative-order feedback control system. A sliding manifold is used to reduce the high relative order. Since the simulators are discrete-time systems, a discrete-time sliding mode controller is used to enforce the sliding manifold. The Boundary Condition Coordinator (BCC), which implements the discrete-time controller, makes the incompatible boundary condition compatible. The major advantage of the discrete-time controller over the continuous time controller is that it is able to guarantee numerical stability. If a continuous-time controller were used, one would still have to solve it using a numerical method. In that case, even if the closed-loop system is stable in the continuous-time domain, the numerical simulation is not guaranteed to be stable.

Since algebraic constraints can have both fast and stable dynamics, it is logical to consider running the BCC faster than the rest of the subsystem simulators. Multi-rate sliding controllers are developed to guarantee the stability of the sliding mode with any integration step size of the subsystem simulators. The influence of the BCCs on the rest of the subsystem simulators is studied using the input-output linearization theory.

The discrete-time sliding mode control is a model-based control algorithm that can guarantee stability. However, the object-oriented simulation paradigm requires the subsystem simulators not to share their own models with other subsystem simulators. Therefore, the model-based controller should be built such that it relies on as little information from the subsystems as possible. The minimum information required for the discrete-time sliding mode controller is identifiable. An optimized multi-rate sliding mode scheme is especially devised to minimize the information disclosure from the subsystem simulators. This version of the multi-rate sliding mode scheme also facilitates the pure numerical computation required by the simulator.

A software environment for Co-Simulation is developed. This software environment, written in Java, defines the protocol for subsystem the simulators. Subsystem simulators and BCCs run as independent processes in the Co-Simulation environment. Class templates containing all necessary functions for the different types of subsystems are defined. Engineers can easily build a subsystem simulator by simply providing the mathematical model, which will be hidden after the subsystem simulator is compiled. Integration engineers can assemble subsystem simulators into simulation of the large coupled system by simply making connections among subsystems. The object-oriented class design makes it possible to extend Co-Simulation over the Internet or to compile subsystems into a single threaded simulator.

1.3 Thesis organization

This thesis is organized into eight chapters. In chapter 2, the major challenge of the object-oriented simulation—the causal conflict between subsystem simulators—is studied. The causal conflicts are modeled as algebraic constraints. The ODEs of the subsystems and the algebraic constraints form DAEs. Different ways of solving algebraically constrained system are compared. A nonlinear control-inspired realization approach is adopted. The Discrete-Time Sliding Mode (DTSM) method is proposed to build the Boundary Condition Coordinator. The BCC is an artificial dynamic subsystem simulator, which will converge to the algebraic constraint. By using the BCC, the causal conflict is resolved while the modularity of the subsystem simulator is preserved.

Chapter 3 is devoted to the details of the development of the DTSM method. The DTSM is achieved by a continuous control law, which avoids the discretization chatter problem. Under the DTSM control law, the vicinity of the sliding manifold is shown to be an invariant set. A Lyapunov type of proof is given to show that the invariant set is attractive, and the system trajectory will reach the invariant set within finite steps and will stay there forever. The bound of the minimum magnitude of the control input is given and its relation to the overall simulation step size is shown. In order to decouple the minimum magnitude of the control input and the step size of the subsystem simulators, the multi-rate DTSM method is developed. The multi-rate DTSM method guarantees the convergence of the sliding mode under any chosen subsystem simulator step size.

In Chapter 4, the effect of DTSM control on the subsystem simulators is investigated. These results indicate that a parameter in the sliding manifold definition affects the numerical stability of the subsystems, when the subsystem simulators are solved using explicit integration algorithms. The input-output linearization theory is used to investigate the role of the BCC. The close relationship between the index of the DAE and the relative order of the feedback control system is studied. This chapter suggests a procedure to prevent occurrences of the instability that may be caused by the BCC.

In Chapter 5, the implementation issues of the DTSM methods are discussed. A straightforward hybrid symbolic and numerical implementation is shown to be ineffective and not fully consistent with the requirements of the object-oriented simulation. The computational sequence is optimized to improve the efficiency. A variation of the multi-rate DTSM method is developed. The variation decouples the effect of the constrained input and the state variables on the sliding function. This new decoupled multi-rate DTSM method only requires the following information from the constrained subsystem simulators: the outputs, the time derivatives of up to one less than the relative order of the outputs and the time derivative of the relative order of the outputs as a function of the constrained input. The decoupled method is realizable using pure numerical operations, which will not cause problems if functions without closed-form expressions are used in the simulators.

Chapter 6 focuses on the software development for the Co-Simulation environment. The functional requirements of the Co-Simulation software environment are identified. The distributed architecture is chosen for the Co-Simulation environment. The Co-Simulation software environment is developed in such a way that it will be easy to extend to the Internet-based simulation while it is also given backward compatibility to object-oriented modeling. The subsystem simulator classes are coded in an object-oriented way. The class templates for different type of subsystems are provided for future use. A subsystem engineer can easily build a subsystem simulator by merely providing the mathematical model of that subsystem.

In Chapter 7, a real world problem consisting of a multiunit air-conditioning/heat pump system is simulated using the Co-Simulation software environment. The system consists of seven physical subsystem simulators and two Boundary Condition Coordinators. This coupled simulation is entirely based upon the software developed in Chapter 6.

Finally, remarks on the future research directions of the object-oriented simulation are given in Chapter 8.

II Incompatible Boundary Condition

2.1 Modeling of Subsystems

Modeling is the basis of simulation. Numerical simulation is simply a piece of software that solves a mathematical model. Therefore, we must discuss the modeling of large and complex systems before we can simulate them. Because of the importance of modeling, people have developed numerous modeling schemes, languages and software [Sinha et al, 2001]. Using different criteria, these modeling approaches can be divided into many categories. One of these criteria is whether the modeling approach is procedural or declarative. Many physical laws are declarative, meaning they only define the relationship among certain physical quantities. For example, Newton's second law

$$F = m \cdot a \tag{2.1.1}$$

does not tell us if the force causes the acceleration or if the change of momentum causes the force. The sign "=" is the equal sign, which only means that the two sides total the same quantity. If a modeling method exhibits such a property, the modeling method is declarative. The counterpart of the declarative modeling approach is the procedural modeling approach. In a procedural model, all relations are defined using assignments. In this case, the appropriate sign is ": =". A modeler has to choose if the force causes the acceleration

$$a := \frac{1}{m} F \tag{2.1.2}$$

or if the change of momentum causes the force

$$F := \frac{d}{dt}(mv). \tag{2.1.3}$$

Both declarative and procedural modeling languages have been developed. Declarative modeling methods are more flexible than procedural methods. It is easier to assemble subsystems modeled using a declarative approach into a large system model. However, numerical solvers are all procedural. The declarative models require post-processing before they can be numerically simulated. Post-processing of the assembled declarative models replaces

equal signs with assignment signs. The post-processing generates a numerical procedure relying on numerical integration. Numerical differentiation causes drift problem and is therefore avoided. Since we intend to build object-oriented and component-based simulations, which are not compatible with the post-processing of coupled models, we start our modeling effort using the procedural approach.

A bondgraph is a graphic procedural modeling tool [Karnopp et al, 1990; Hogan, 1987]. It is widely used in the modeling of multi-physics systems in which the subsystem components belong to different energy domains, e.g. mechanical, fluid, thermal, or electrical. Since the Co-Simulation software environment will be applied to coupled systems of different physical domains, we adopt some concepts of bondgraph theory, such as energetic bonds and junctions. An energetic bond is used to describe the coupling between two systems. Energy is the common physical property among all physical domains. Therefore, subsystems of different energy domain communicate with each other through energy transmission. Energetic coupling between physical systems implies that the coupling is bilateral [Karnopp et al, 1990]. The coupled subsystems influence each other, so that all connected subsystems transmit both the input and the output information through one bond.

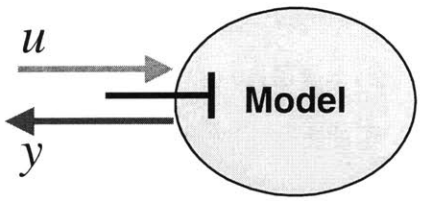


Figure 2.1.1 Bondgraph model of a subsystem

Figure 2.1.1 shows a subsystem model with one free bond, which is ready to be connected with other subsystems. For simplicity only one energetic coupling is considered here. The model of subsystem simulators in the differential equation form is

$$\dot{\mathbf{x}}_i = \mathbf{f}_i(\mathbf{x}_i, u_i, t) \tag{2.1.4}$$

$$y_i = y_i(\mathbf{x}_i), \tag{2.1.5}$$

where u and y are the input and the output variables of one bilateral energy coupling; subscript i denotes the i^{th} simulator of the Co-Simulation system.

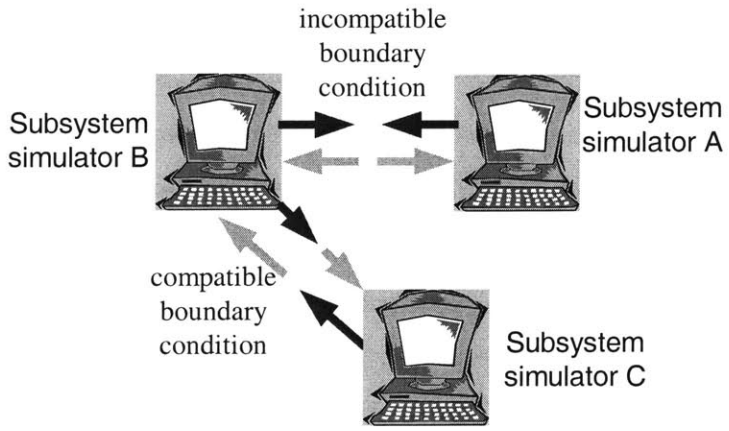


Figure 2.2.1 Co-Simulation of dynamic subsystems with different boundary conditions.

2.2 Algebraic Constraints

Power, the time rate of energy transfer, is the scalar product of the conjugate power variable—the flow variable f and the effort variable e. The physical boundary between two subsystems is modeled as a junction. By applying the first law of thermodynamics at the junction between two coupled subsystems, power flow into and out of the boundary should be equal since the boundary does not generate, store or dissipate energy. Using the conjugate power variable, the power balance for the junction is:

$$e_{in} f_{in} = e_{out} f_{out} . \tag{2.2.1}$$

We can also apply any one of the physical laws, such as Newton's third law, conservation of mass, conservation of momentum, conservation of electrical charge, etc., to the junction. Then boundary equation (2.2.1) can be written as

$$e(f_{in} - f_{out}) = 0 \tag{2.2.2}$$

or

$$(e_{in} - e_{out}) f = 0 . \tag{2.2.3}$$

Suppose that we have two subsystems coupled in the common effort case of Eq. (2.2.2). If the boundary condition is compatible, i.e. u_1, y_1, u_2, y_2 being e, f_{out}, f_{in}, and e, respectively, the two systems can be written in the ODE form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t), \tag{2.2.4}$$

where $\mathbf{x}$ is the combination of $\mathbf{x}_1$ and $\mathbf{x}_2$. The input is represented by the output of the other system and is not explicitly shown in Eq. (2.2.4). This type of coupling is shown in Fig. 2.2.1 between subsystem B and C.

However, if the boundary condition is incompatible, i.e. u_1, y_1, u_2, y_2 being e, f_{out}, e, and, f_{in}, respectively, the two systems cannot be written in a simple ODE form. Instead, we have to use an extra algebraic equation (2.2.6) to describe the coupled systems:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, z, t) \tag{2.2.5}$$

$$0 = y_1(\mathbf{x}, u, t) - y_2(\mathbf{x}, u, t) = g(\mathbf{x}, z, t), \tag{2.2.6}$$

where z, called a constrained input, is the input u_1 and u_2 of the coupled subsystems. g is the algebraic constraint that must be used to decide the constrained input z. This type of coupling is shown in Fig. 2.2.1 between subsystem the simulators A and B.

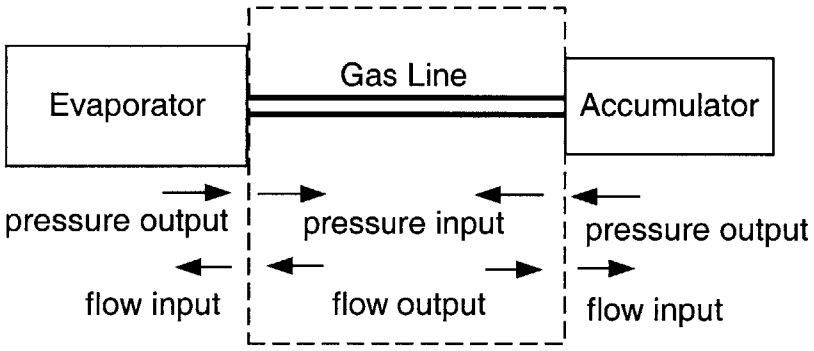


Figure 2.2.2 Incompatible boundary condition caused by numerical difficulty.

In some cases, the original models are compatible at the interface, such as shown in Fig. 2.2.2. However, when the connecting pipe is short and wide or the mass flow rate is low, the static equation that calculates the flow based on pressure difference becomes nearly singular. The proximity to the singularity makes the pipe equation very sensitive to pressure difference and can therefore be quite error prone. In practice, the pressure drop between the heat exchanger and the condenser is ignored to preclude these potential errors. Without the pipe, though, the two thermal equipment models have to interact across an incompatible boundary condition.

Incompatible boundary conditions are the most significant source of algebraic constraints in the modeling of dynamic systems [Campbell, 1995]. Algebraic equations can also be the result of model reduction or constrained dynamics, etc. We focus on the algebraic constraints caused by the incompatible boundary condition in this research. The challenge is to develop a method that can numerically solve differential equations coupled with algebraic constraints without compromising the object-oriented properties of the subsystem simulators.

2.3 Differential Algebraic Equation and Index Reduction

A differential system constrained by an algebraic equation, such as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, z, t) \tag{2.2.5}$$

$$0 = g(\mathbf{x}, z, t)\,, \tag{2.3.1}$$

is called a Differential Algebraic Equation (DAE). Assuming that the constraint equation (2.3.1) is sufficiently differentiable and that a well-defined solution for $\mathbf{x}$ and z exists with consistent initial conditions, i.e. initial condition satisfies the algebraic constraint equation, an important structural property of a DAE known as the index can be defined [Brenan et al, 1989]. The minimum number of times that all or part of the constraint equation (2.3.1) must be differentiated with respect to time in order to solve for $\dot{z}$ as a continuous function of t, $\mathbf{x}$, and z is the index of the DAE Eqs. (2.2.5) and (2.3.1).

Assuming that the DAE has an index r, by definition of the DAE index, the following set of algebraic equations must be satisfied by any exact solution of the DAE:

$$0 = g(\mathbf{x}, z, t) \tag{2.3.1}$$

$$0 = \frac{dg}{dt} \tag{2.3.2}$$

$$\cdots$$

$$0 = \frac{d^{r-1}}{dt^{r-1}} g\,. \tag{2.3.3}$$

A new constraint equation can be defined as a weighted summation of Eq. (2.3.1) through (2.3.3). This new constraint equation requires only one differentiation to solve for $\dot{z}$. If we use this new constraint equation to replace the original constraint Eq. (2.3.1), the resulting DAE is index one

and it is thus easier to solve. We define the following equation as the previously mentioned weighted summation:

$$s(\mathbf{x},z,t)=\left(\mu\frac{d}{dt}+1\right)^{r-1} g=0\,, \tag{2.3.4}$$

where $\mu > 0$. This is also known the Baugerte's stabilization [Baugarte, 1972]. This equation can be viewed as a sliding manifold as well. The sliding manifold definition places the poles of the constraint dynamics at $-\mu^{-1}$ [Gordon and Liu, 1998]. Now the DAE system (2.2.5) and (2.3.4) is index 1 and is much easier to solve.

Equation (2.3.4) represents a critically damped dynamic system. There are $(r-1)$ decoupled linear modes in this dynamic system. The time constant of each mode is the same and is equal to μ. If the sliding manifold equation is satisfied, all modes of the sliding manifold subspace decay to zero asymptotically with time constant μ. The derivatives of the original algebraic constraint, and the weighted summation of the constraint and the derivatives will remain zero all the time if the original algebraic constraint Eq. (2.3.1) is satisfied at the beginning of the simulation. Thus the original high index algebraic constraint can be replaced by the reduced index algebraic constraint Eq. (2.3.4).

2.4 Resolving of Algebraic Constraints

In the past 30 years, many DAE solution algorithms and packages have been developed [Brenan et al, 1989; Mattsson and Soderlind, 1993; Petzold, 1983]. If we use the existing DAE solution packages, we have to collect all dynamic and algebraic equations from all of the subsystems and feed them into the DAE solution package. This approach can solve object-oriented modeled large systems, but it should not be used in the Co-Simulation environment since every subsystem simulator of the Co-Simulation is an object-oriented component. If we dismantle the subsystem simulators and extract models out, we destroy the basic building blocks of Co-Simulation and the object-oriented simulation paradigm is destroyed.

Another way of dealing with the DAE system is to convert the DAE system into an ODE system. This procedure, known as DAE realization in the control community, may increase or decrease the dimension of the system. Figure 2.4.1 shows one example of two subsystems coupled through incompatible boundary conditions. Because of the boundary conditions, the

overall system equations contain algebraic constraints. Subsystem A takes an effort input and produces a flow output. If we can change the causal assignment of the R element of subsystem A, subsystem A can take a flow input and produce an effort output. Therefore, by manipulating the subsystem model, the algebraic constraint can be eliminated as shown in Fig. 2.4.2.

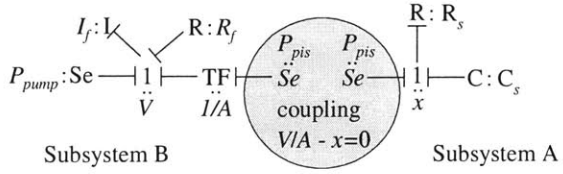


Figure 2.4.1 Algebraic constraint.

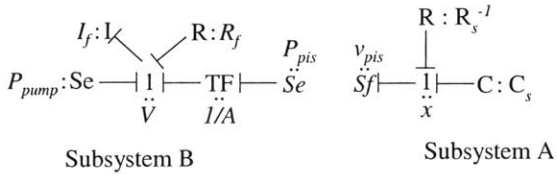


Figure 2.4.2 Algebraic constraint eliminated.

However, manipulating subsystem equations is against the functional requirements of the Co-Simulation since subsystems will be provided in the form of coded simulators. Also this method does not always work in practice. If R_s^{-1} is not given in a closed form expression, this method fails.

Figure 2.4.3 shows that two robots keep their endpoints in contact with each other. If we treat each robot as a subsystem simulator, we can simultaneously run two simulators of the robots to simulate the system. However, the boundary condition, which is the point of connection for the two endpoints, forms an algebraic constraint. We can add a fictitious spring and damper between the two endpoints empirically. This addition makes the original incompatible boundary condition compatible and thus removes the algebraic equation. Known as the stiff element method, this method does not change the subsystem simulators and the modularity is intact [Zeid and Overholt, 1995]. However, this approach requires an *ad hoc* procedure for tuning parameters. In practice, we can only adopt linear components for the fictitious spring and damper. The linear components substantially reduce the performance that the fictitious spring and damper may provide. Slightly damped linear stiff elements introduce

highly oscillatory modes while heavily damped linear elements converge to the algebraic constraint slowly.

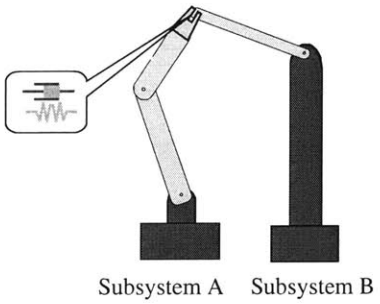


Figure 2.4.3. Fictitious spring and damper

A substantial improvement for the original *ad hoc* stiff element method is the Singularly Perturbed Sliding Manifold (SPSM) method [Gordon and Liu, 1998]. It is devised based on the nonlinear control theory. The algebraic constraint is treated as the desired trajectory and a model based feedback controller is designed to push the system onto the desired trajectory. Starting from the index-reduced DAE Eqs. (2.2.5) and (2.3.4), a boundary-layer-smoothed switching control law is used to enforce the sliding manifold Eq. (2.3.4)

$$\dot{s} = -Ksat\left(\frac{s}{\varepsilon}\right), \tag{2.3.5}$$

where $1 >> \varepsilon > 0$ and K is a positive constant [Slotine and Li, 1991]. Since both s and $\dot{s}$ are known, we can solve for $\dot{z}$ using Eq. (2.3.5):

$$\dot{z} = -K \cdot \left(\frac{\partial s}{\partial z}\right)^{-1} \text{sat}\left(\frac{s}{\varepsilon}\right) - \left(\frac{\partial s}{\partial z}\right)^{-1}\left(\frac{\partial s}{\partial t} + \frac{\partial s}{\partial \mathbf{x}}\mathbf{f}\right). \tag{2.3.6}$$

The ODEs in equations (2.2.5) and (2.3.6) are based on the realization of the DAE Eqs. (2.2.5) and (2.3.1). Figure 2.4.4 shows the block diagram of the SPSM realized DAE system [Gordon, B. W., 1999]. DAE realization using SPSM keeps the modularity of the subsystem simulators. The dynamic equation (2.3.6) can be viewed as the generalized fictitious spring and damper, which supplies the compatible boundary condition for the subsystem simulators.

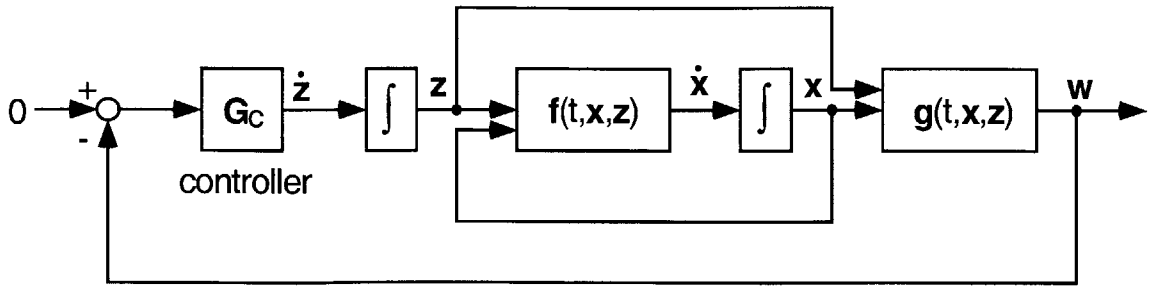


Figure 2.4.4. Modeling the solution of DAE as a control system.

However, by replacing Eq. (2.2.5) with Eq. (2.3.6), we are using a stiff ODE system to approximate the original DAE system. The resulting stiff ODE system may be even harder to solve than the original DAE (2.2.5) and (2.3.1) [Brenan et al, 1989]. We not only have to stabilize the sliding mode by choosing appropriate parameters of ε and K, but we must also guarantee the numerical stability of the resulting stiff ODE. The SPSM based control law is highly nonlinear and the subsystems are often nonlinear as well. Guaranteeing unconditional numerical stability for arbitrary nonlinear dynamic simulation is very difficult.

2.5 Pure Discrete-Time Boundary Condition Coordinator

The Singularly Perturbed Sliding Manifold method is a control-inspired realization method, which can provide the correct boundary condition stably. However, the SPSM method is a continuous-time approach. If we were to use the SPSM method in the Co-Simulation, we would have to check the numerical stability along with the stability conditions of the sliding mode since the Co-Simulation is a numerical simulation. Although subsystems of a coupled system are continuous systems, the subsystem simulators are all discrete-time systems. If the algebraic constraints are converted into stable discrete-time dynamic systems that provide the same boundary condition, we do not have to check the numerical stability.

Before we move on to the discussion of discrete-time systems, we first define a notation that will be required for our discrete-time analysis. All variables evaluated at time $k\Delta t$ are written with a subscript k. For example, the state variable $\mathbf{x}(k\Delta t)$ is written as $\mathbf{x}_k$ and the constrained input $z(k\Delta t + \Delta t)$ is written as z_{k+1}. All functions evaluated at $(\mathbf{x}_k, t_k, z_k)$ are written with subscript k. For example, the sliding mode $s(\mathbf{x}_k, t_k, z_k)$ is written simply as s_k.

In section 2.1, the model of the i^{th} dynamic subsystem is written as

$$\begin{aligned} \dot{\mathbf{x}}_i &= \mathbf{f}_i(\mathbf{x}_i, u_i, t) \\ y_i &= y(\mathbf{x}_i) \end{aligned} \quad . \tag{2.5.1}$$

The simulator form of the dynamic subsystem is the discretized version of the above model

$$\begin{aligned} \mathbf{x}_{i,k+1} &= \mathbf{x}_{i,k} + \Delta t \cdot \bar{\dot{\mathbf{x}}}_{i,k} \\ y_{i,k+1} &= y(\mathbf{x}_{i,k+1}) \end{aligned} \quad . \tag{2.5.2}$$

where $\bar{\dot{\mathbf{x}}}_{i,k}$ is the effective derivative, which is different for different numerical integration methods. For the sake of simplicity, we use Euler's forward integration formula. The effective derivative in this case is simply

$$\bar{\dot{\mathbf{x}}}_{i,k} = \dot{\mathbf{x}}_{i,k} = \mathbf{f}_i(\mathbf{x}_{i,k}, z_k, t_k). \tag{2.5.3}$$

Considering the case when two subsystems are in causal conflict configuration; the overall model of the coupled system is then

$$x_{k+1} = x_k + \Delta t \cdot \mathbf{f}(\mathbf{x}_k, z_k, t_k) \tag{2.5.4}$$

$$0 = g(\mathbf{x}_{k+1}) \tag{2.5.5}$$

where z replaces u to emphasize the algebraic constraint. Using Eq. (2.3.4), we define a sliding manifold to replace the original algebraic constraint Eq. (2.5.5). So, the algebraically coupled dynamic system with the reduced index in the numerical simulation form is

$$x_{k+1} = x_k + \Delta t \cdot \mathbf{f}(\mathbf{x}_k, z_k, t_k) \tag{2.5.4}$$

$$s(\mathbf{x}_{k+1}, z_{k+1}, t_{k+1}) = 0. \tag{2.5.6}$$

For this discrete-time system, the dynamic system is only meaningful at the sampling point t_k and the constraint equation needs to be satisfied only at the sampling point as well.

We want to design a special discrete-time dynamic subsystem, which enforces the constraint equation (2.5.6) by providing the constrained input. This special subsystem simulator converts the incompatible boundary condition into a compatible boundary condition. Called a Boundary Condition Coordinator (BCC), the special subsystem simulator is a linear dynamic system

$$z_{k+1} = z_k + v_k \,, \tag{2.5.7}$$

where z_k is the state variable of the BCC and v_k is the control input, which we will design based upon the output variable y_k's from the two subsystem simulators that are connected through the algebraic constraint. Applying the BCC to the system with the incompatible boundary condition in Fig. 2.2.1, we have resolved the incompatible boundary condition as shown in Fig. 2.5.5.

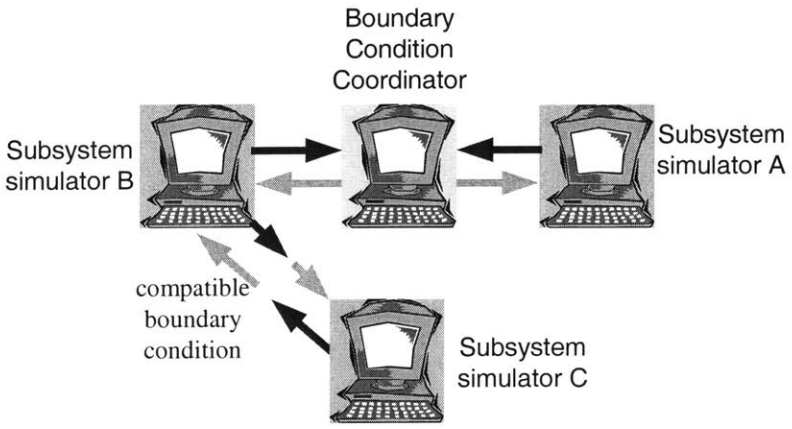


Figure 2.5.5. Boundary Condition Coordinator resolves incompatible boundary condition.

III Discrete-Time Sliding Mode

3.1 Introduction to Discrete-Time Sliding Mode Control

An intuitive approach to Discrete-Time Sliding Mode (DTSM) control is to derive control laws in the continuous-time domain and implement them directly in the discrete-time domain. However, this straightforward method leads to discretization chatter as shown in Fig. 3.1.1 [Yu, 1994]. The discretization, i.e. finite-frequency sampling, causes the discretization chatter. This type of chatter is not the result of the unmodeled dynamics. Figure 3.1.2 shows the sliding mode driven by an ideal infinite fast controller and shows no chattering problem. Because the straightforward approach to the discrete-time sliding manifold always causes the chattering phenomenon and the trajectory does not stay on the sliding manifold, this type of the discrete-time sliding mode is also called pseudo or quasi sliding mode [Delonga, 1989].

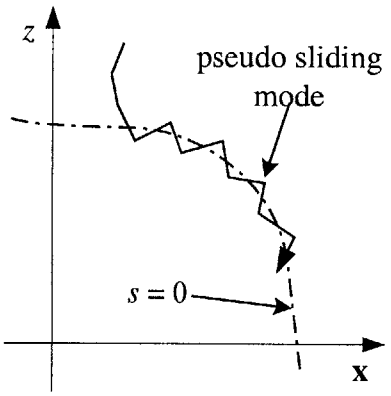


Figure 3.1.1 Pseudo sliding mode in the discrete-time domain.

For a continuous-time system, discontinuous control is required to make the dynamic system exhibit the sliding mode. However, discontinuous control is not required for a discrete-time dynamic system to exhibit the sliding mode motion [Drakunov and Utkin, 1992]. The definition of the discrete-time sliding mode is given by [Utkin et al, 1999]. In the discrete-time dynamic system,

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \tag{3.1.1}$$

where $\mathbf{x} \in R^n$, $\mathbf{u} \in R^m$, $\mathbf{f}$: $R^n \times R^m \rightarrow R^n$. The discrete sliding mode takes place on a subset Σ of the manifold $\sigma = \{\mathbf{x}: \mathbf{s}(\mathbf{x}) = 0\}$, $\mathbf{s} \in R^m$, if there exists an open neighborhood Ξ of this subset such that for each $\mathbf{x} \in \Xi$, it follows that $\mathbf{s}(\mathbf{x}_{k+1}) \in \Sigma$.

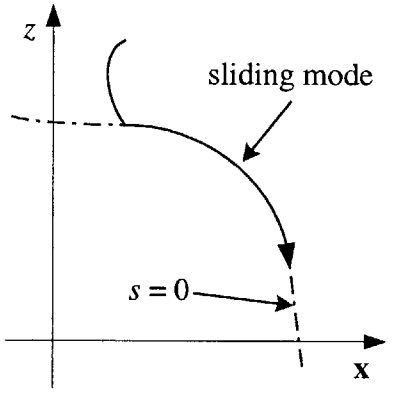


Figure 3.1.2 Sliding mode in the continuous-time domain.

3.2 Single-Rate DTSM

3.2.1 Dynamics of the Sliding Function

The design of the Boundary Condition Coordinator is viewed as a control problem. The dynamic system in discrete-time is

$$x_{k+1} = x_k + \Delta t \cdot \mathbf{f}(\mathbf{x}_k, z_k, t_k) \tag{3.2.1}$$

$$z_{k+1} = z_k + v_k, \tag{3.2.2}$$

and the system trajectory should stay at, or close to, an equilibrium point:

$$s(\mathbf{x}_{k+1}, z_{k+1}, t_{k+1}) = 0. \tag{3.2.3}$$

The relative order of this stabilization problem is zero.

Assuming the continuous function s is sufficiently differentiable around the point $s = 0$, s_{k+1} can be obtained as follows:

$$s_{k+1} = s_k + \frac{\partial s}{\partial \mathbf{x}}(x_k, t_k, z_k) \cdot (\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\partial s}{\partial t}(x_k, t_k, z_k) \cdot \Delta t + \frac{\partial s}{\partial z}(x_k, t_k, z_k) \cdot (z_{k+1} - z_k) + R_k. \tag{3.2.4}$$

where R is the second order remnant term. If we let $\mathbf{w} = [\mathbf{x}^T, t, z]^T$, then R_k is as follows:

$$R_k = \Delta \mathbf{w}_k^T \left[\frac{\partial}{\partial \mathbf{w}} \left(\frac{\partial s}{\partial \mathbf{w}} \right)^T \right]_{\mathbf{w}_k + \theta \Delta \mathbf{w}_k} \Delta \mathbf{w}_k , \tag{3.2.5}$$

where $0 \leq \theta \leq 1$, according to the Mean Value Theorem. Substituting in the dynamic equation (3.2.1) and (3.2.2), Equation (3.2.4) can be represented by using variables of step k, and we have

$$s_{k+1} = s_k + \mathbf{J}\mathbf{x}_k \Delta t \cdot \mathbf{f}(\mathbf{x}_k, z_k, t_k) + Jt_k \Delta t + Jz_k v_k + R_k . \tag{3.2.6}$$

We define

$$Jz_k = \frac{\partial s}{\partial z}(\mathbf{x}_k, z_k, t_k), \tag{3.2.7}$$

and

$$\alpha_k = \mathbf{J}\mathbf{x}_k \mathbf{f}(\mathbf{x}_k, z_k, t_k)\Delta t + Jt_k \Delta t \tag{3.2.8}$$

for the sake of notation clarity. The sliding function s at the step $k+1$ is

$$s_{k+1} = s_k + \alpha_k + Jz_k v_k + R_k . \tag{3.2.9}$$

By the definition of the DAE index, Jz_k is nonzero.

3.2.2 Continuous Control Input

From Eq. (3.2.9), we find that the discrete-time dynamic system s at the step $k+1$ is decided by four terms. Term s_k is the starting point for the next step. Terms α_k and R_k are external influences applied to the dynamic system Eq. (3.2.9). Term $Jz_k \cdot v_k$ is where active control can be applied to influence s_{k+1}. In order to push s_{k+1} towards the equilibrium point, we have to ensure that the control influences the system more than other sources. Therefore, it intuitively requires a minimum magnitude for the control input such that the control input is always strong enough to compensate for the external influences and to push the system trajectory towards the right direction. v_0 is denoted as the magnitude of the minimum control input. A domain $D \subset \Omega = \{\mathbf{x}, t, z \mid \mathbf{x} \in R^n, t \in R^+, z \in R\}$ can be defined, and D is the vicinity of the sliding

manifold and the index is fixed. $|Jz|^{-1}$, $|\alpha(\Delta t)|$ and $|R(\Delta t, v_0)|$ are assumed to be upper bounded by positive numbers. Then, the minimum control magnitude requirement is

$$v_0 > \sup_D(\left|Jz^{-1}\right|) \cdot \sup_D(|\alpha(\Delta t)| + |R(\Delta t, v_0)| + \delta), \tag{3.2.10}$$

where the constant δ is a small finite positive constant.

Since the term R_k cannot be exactly evaluated, we let the equivalent control O_k be our estimate of v_k that makes the next step of the sliding function, s_{k+1}, zero, giving

$$O_k = -Jz_k^{-1}(s_k + \alpha_k). \tag{3.2.11}$$

We choose a control law

$$v_k = v_0 \,\mathrm{sat}\left(\frac{O_k}{v_0}\right). \tag{3.2.12}$$

The saturation function sat() is used to limit the magnitude of the control input in order to guarantee the stability..

3.2.3 The Invariant Set around the Origin

We want to show that the closed-loop system with the control law of Eq. (3.2.12) possesses an invariant set

$$|s| \le \sup_D |R| \tag{3.2.13}$$

around the origin $s = 0$. When the system trajectory starts in the region that satisfies Eq. (3.2.13), we have

$$\left|-Jz_k^{-1}(s_k + \alpha_k)\right| < \sup_D \left|Jz_k^{-1}\right| \cdot \sup_D(|R_k| + |\alpha_k|). \tag{3.2.14}$$

The left hand side is actually the magnitude of the equivalent control O_k, so we have

$$|O_k| < \max\left|Jz_k^{-1}\right| \cdot \max(|R_k| + |\alpha_k|). \tag{3.2.15}$$

Adding one more term to right hand side and applying the inequality (3.2.10), we obtain

$$|O_k| < \max\left|Jz_k^{-1}\right| \cdot \max\left(|R_k| + \cdot|\alpha_k| + \delta\right) < v_0 . \tag{3.2.16}$$

This shows that the control law becomes

$$v_k = v_0 \,\mathrm{sat}\left(\frac{O_k}{v_0}\right) = O_k \tag{3.2.17}$$

under the condition Eq. (3.2.13). Substituting the control input (3.2.17) into the discrete-time dynamic system (3.2.9), we have

$$s_{k+1} = s_k + \alpha_k + Jz_k\left(-Jz_k^{-1}(s_k + \alpha_k)\right) + R_k = R_k . \tag{3.2.18}$$

This shows that Eq. (3.2.13) is an invariant set. Next we will show that the invariant set is attractive.

3.2.4 Proof of Convergence

To prove that the invariant set around the origin is attractive, we choose a Lyapunov function candidate

$$V(t_k, s_k) = V_k = |s_k| . \tag{3.2.19}$$

It is easy to find positive constants a and $b > 0$ such that

$$a|s_k| \le V_k \le b|s_k| . \tag{3.2.20}$$

This shows that the function V is positive define and decrescent.

We define the forward difference function ΔV as follows:

$$\Delta V_k = V_{k+1} - V_k = |s_{k+1}| - |s_k| . \tag{3.2.21}$$

If the trajectory starts not far from the invariant set of Eq. (3.2.13)

$$\sup_D \left|Jz^{-1}\right| \cdot \sup_D\left(|R|\right) < |s_k| < \sup_D \left|Jz^{-1}\right| \cdot \sup_D\left(|R| + |\alpha|\right), \tag{3.2.22}$$

it is easy to see that the sliding function s will reach the invariant set in one step of time. Now, let us start from

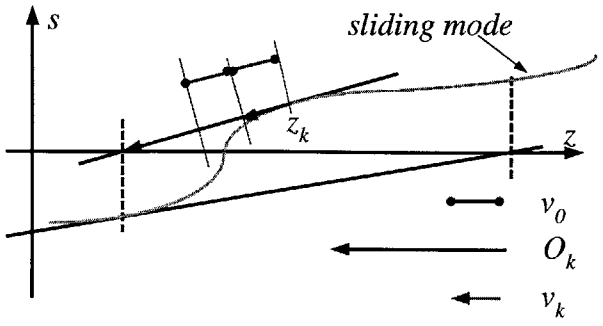


Figure 3.2.1 Illustration of the equivalent control, minimum control magnitude and control input.

$$|s_k| \geq \sup_D \left|Jz^{-1}\right| \cdot \sup_D \left(|R| + |\alpha|\right), \tag{3.2.23}$$

which is equivalent to

$$|O_k| > v_0 . \tag{3.2.24}$$

Substituting Eq. (3.2.12) into Eq. (3.2.9) and applying Eq. (3.2.11), we have

$$\begin{aligned} s_{k+1} &= s_k + \alpha_k + Jz_k v_0 \frac{O_k}{|O_k|} + R_k \\ &= s_k + \alpha_k + Jz_k v_0 \frac{-Jz_k^{-1}(s_k + \alpha_k)}{|O_k|} + R_k . \\ &= (s_k + \alpha_k)\left(1 - \frac{v_0}{|O_k|}\right) + R_k \end{aligned} \tag{3.2.25}$$

We substitute Eq. (3.2.25) into the forward difference of the Lyapunov function candidate and we have

$$\Delta V_k = |s_{k+1}| - |s_k| = \left|(s_k + \alpha_k)\left(1 - \frac{v_0}{|O_k|}\right) + R_k\right| - |s_k| . \tag{3.2.26}$$

Applying inequality (3.2.24), the forward difference function ΔV_k can be written as:

$$
\begin{aligned}
\Delta V_k &\le |s_k + \alpha_k| \cdot \left(1 - \frac{v_0}{|O_k|}\right) + |R_k| - |s_k| \\
&= |s_k + \alpha_k| - \frac{|s_k + \alpha_k| v_0}{\left|Jz_k^{-1}(s_k + \alpha_k)\right|} + |R_k| - |s_k| \\
&= |s_k + \alpha_k| - \frac{v_0}{\left|Jz_k^{-1}\right|} + |R_k| - |s_k| \\
&\le |\alpha_k| + |R_k| - \frac{v_0}{\left|Jz_k^{-1}\right|}
\end{aligned}
\tag{3.2.27}
$$

Inequality (3.2.10) implies

$$
|\alpha_k| + |R_k| - \frac{v_0}{\left|Jz_k^{-1}\right|} < -\delta . \tag{3.2.28}
$$

Combining Eq. (3.2.28) and Eq. (3.2.27), we finally arrive at:

$$
\Delta V_k < -\delta < 0 . \tag{3.2.29}
$$

This result guarantees that the system trajectory moves towards the invariant set (3.2.13) at a finite rate and that the invariant set is asymptotically stable.

According to the theorem of the discrete-time sliding mode, the closed-loop system exhibits a discrete-time sliding mode after s reaches the invariant set of Eq. (3.2.13). Once the sliding mode is established, the system trajectory will stay within the invariant set forever. The invariant set is also called the boundary layer of the sliding manifold of Eq. (3.2.3). Figure 3.2.2 shows the motion of the discrete-time sliding mode.

The meaning of the bound equation (3.2.10) can be explained by Eq. (3.2.9). The magnitude of the control input v_0 must be large enough to compensate for α_k and R_k to guarantee $|s_{k+1}|$ decreasing. Since α_k is a function of t_k, $\mathbf{x}_k$, z_k, and Δt, we can simply pick a v_0 to compensate for α_k. The nonlinear term R_k, however, is more difficult since R_k is a function of t_k, $\boldsymbol{x}_k$, z_k, Δt, and v_0. Although R_k is the second order small quantity with respect to Δt and v_0, R_k can still be large if s is highly nonlinear with respect to Δt or v_0. When R_k is large, we need a large v_0 on the left hand side of Eq. (3.2.10) to compensate for the R_k effect. A large v_0 will in turn cause a larger R_k. This may make finding a v_0 satisfies Eq. (3.2.10) impossible.

However, we can always find a v_0 that satisfies the inequality (3.2.10) by reducing the step size Δt. When Δt is small, Eq. (3.2.10) can be written as follows:

$$v_0 > \varepsilon + \gamma v_0^2, \tag{3.2.30}$$

where ε and γ are small positive numbers. This shows that we can always find v_0 that satisfies condition (3.2.10) and therefore the discrete-time sliding controller is able to stably enforce the algebraic constraint Eq. (3.2.3).

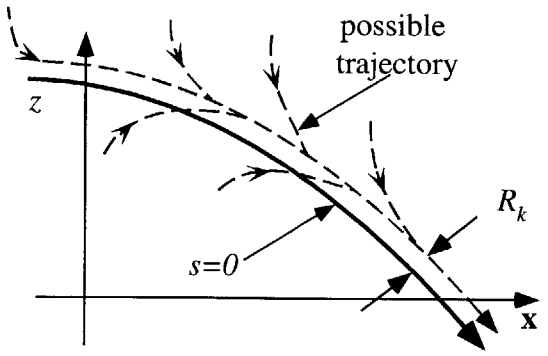


Figure 2.2.2 Phase portrait of the discrete-time sliding mode motion.

3.3 The Multi-Rate DTSM

3.3.1 Two-Time Scale Problem

The cost of using a small Δt to satisfy the convergence condition in Eq. (3.2.10) is high, especially when the dimension of **x** is large. We have to decrease the step size for the entire coupled system just to guarantee the stability of the sliding controller designed for the incompatible boundary condition. Although, the smaller step size will make the accuracy of the entire system better, the extra accuracy gained in **x** by the smaller integration steps may not be necessary. Thus it is of great interest to decouple the overall step size of the Co-Simulation from the stability requirement of the sliding mode controller.

In physical system modeling, very fast dynamics are sometimes modeled as algebraic constraints since the time constants of the fast dynamics are much faster than the dynamics of the rest of the system. If we are not interested in the dynamic properties of the fast dynamics that settle in a very short period of time but rather in the slower dynamics, we can treat the fast dynamics as algebraic constraints [Kokotovic et al, 1986]. This is the basis of the singular

perturbation theory. For such cases, using the fast dynamics Eq. (3.2.2) to replace the algebraic constraint Eq. (3.2.3) makes physical sense. Some algebraic constraints are not the result of ignoring fast dynamics. They are the result of redundant state variables. Although replacing this type of algebraic constraint is not based on physical reasoning, the sliding control method that we have just shown shows that the error generated by the substitution can be controlled and small. In either case, the algebraic constraint can be viewed as a dynamic system that settles to a constant in a short time period, which is much smaller than the smallest time constant of the rest of the dynamic system. This suggests that we can simulate the algebraic-constraints-converted fast dynamics using a time step smaller than the integration step size of the subsystem simulators. This leads to multi-rate simulation.

3.3.2 Implementation of DTSM

To implement the multi-rate simulation, we divide one step of the subsystem simulators into n small steps. Now the integration of the subsystem simulators is in the time scale k, while the algebraic constraint is simulated in the time scale i which is n times faster than the time scale k. Any variable evaluated at $\Delta t(k+i/n)$ instant is denoted using subscript $k+i/n$. When we calculate variables, such as $s_{k+\frac{i}{n}}$ and $Jz_{k+\frac{i}{n}}$, the value of the state variable $\mathbf{x}_{k+\frac{i}{n}}$ is required. The true value of $\mathbf{x}_{k+\frac{i}{n}}$ cannot be obtained for every i from the subsystem simulators since the state variables updates only at $\Delta t \cdot k$ instant. Instead, we use the values of $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ to interpolate the value of $\mathbf{x}_{k+\frac{i}{n}}$. In order to avoid clumsy notation, the variable name of $\mathbf{x}_{k+\frac{i}{n}}$ is used for these interpolated values.

Figure 3.3.1 compares the new multi-rate method with the original single-rate method. For z_{k+1} integration, we spend n small steps from z_k instead of just one step from z_k. By the configuration of this type of multi-rate scheme, we know that the sliding function s evaluated in the fast time scale can be used in the slow time scale. If we pick every $s_{k+\frac{i}{n}}$ out of all $s_{k+\frac{i}{n}}$ when i is zero, we obtain the s series in the large time scale. This shows that if the sliding controller in the fast time scale is stable, then the sliding controller in the slow time scale is also stable.

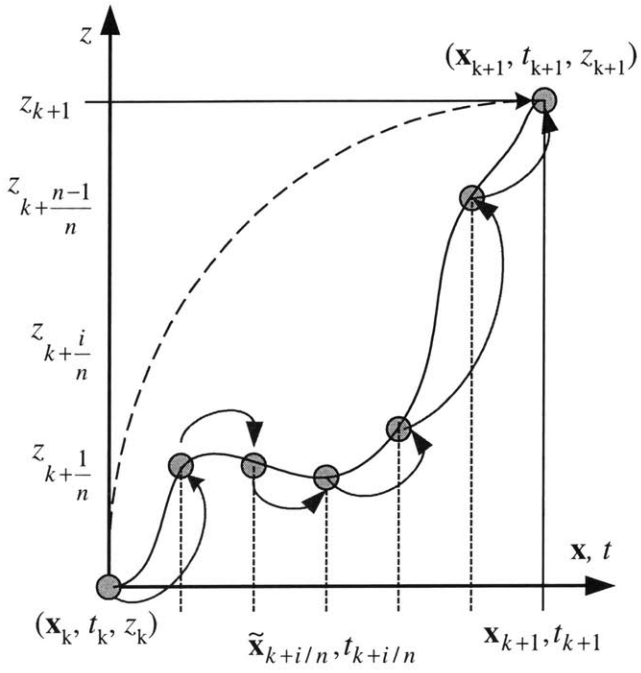


Figure 3.3.1 Possible implementations of the multi-rate algorithm.

Similar to the previous proof, we can define a Lyapunov function candidate

$$V_{k+\frac{i}{n}} = \left| s_{k+\frac{i}{n}} \right|, \tag{3.3.1}$$

and this function also has the property of Eq. (3.2.5). The forward difference of the Lyapunov function candidate is

$$\Delta V_{k+\frac{i}{n}} = \left| s_{k+\frac{i+1}{n}} \right| - \left| s_{k+\frac{i}{n}} \right|. \tag{3.3.2}$$

$s_{k+\frac{i+1}{n}}$ is obtained by the local linearization and the interpolation of $\mathbf{x}$, as follows

$$\begin{aligned} s_{k+\frac{i+1}{n}} = s_{k+\frac{i}{n}} + \left.\frac{\partial s}{\partial \mathbf{x}}\right|_{k+\frac{i}{n}} \frac{\Delta \mathbf{x}_k}{n} + \left.\frac{\partial s}{\partial t}\right|_{k+\frac{i}{n}} \frac{\Delta t}{n} + \left.\frac{\partial s}{\partial z}\right|_{k+\frac{i}{n}} \Delta z_{k+\frac{i}{n}} \\ + R_{k+\frac{i}{n}}\left(t_k + \frac{i}{n}\Delta t, \mathbf{x}_k + \frac{i}{n}\Delta \mathbf{x}_k, z_{k+\frac{i}{n}}, \frac{\Delta t}{n}, \Delta z_{k+\frac{i}{n}} \right) \end{aligned}. \tag{3.3.3}$$

Similar to Eq. (3.2.9), the sliding mode at $k+\frac{i+1}{n}$ step is

$$s_{k+\frac{i+1}{n}} = s_{k+\frac{i}{n}} + \alpha_{k+\frac{i}{n}} + Jz_{k+\frac{i}{n}} \cdot v_{k+\frac{i}{n}} + R_{k+\frac{i}{n}}. \tag{3.3.4}$$

The equivalent control is

$$O_{k+\frac{i}{n}} = -Jz^{-1}_{k+\frac{i}{n}}\left(s_{k+\frac{i}{n}} + \alpha_{k+\frac{i}{n}} \right), \tag{3.3.5}$$

and the control law is

$$v_{k+\frac{i}{n}} = v_0 \,\mathrm{sat}\left(\frac{O_{k+\frac{i}{n}}}{v_0} \right). \tag{3.3.6}$$

We can also show that

$$|s| \le \sup_D(|R|). \tag{3.3.7}$$

is an invariant set if the system is under feedback control law (3.3.6). We need to find what condition v_0 has to satisfy in this multi-rate scheme. Starting from the initial condition

$$\left| O_{k+\frac{i}{n}} \right| > v_0, \tag{3.3.8}$$

and using the procedure similar to Eqs. (3.2.25) to (3.2.27), it requires that

$$\left| \alpha_{k+\frac{i}{n}} \right| + \left| R_{k+\frac{i}{n}} \right| - \frac{v_0}{\left| Jz^{-1}_{k+\frac{i}{n}} \right|} < -\delta \tag{3.3.9}$$

is always satisfied in order to guarantee the convergent condition

$$\Delta V_{k+\frac{i}{n}} = \left| s_{k+\frac{i+1}{n}} \right| - \left| s_{k+\frac{i}{n}} \right| < -\delta . \tag{3.3.10}$$

Therefore, the minimum control magnitude must satisfy

$$v_0 > \sup_D \left(\left| Jz^{-1} \right| \right) \cdot \sup_D \left(\left| \alpha\left(\frac{\Delta t}{n} \right) \right| + \left| R\left(\frac{\Delta t}{n}, v_0 \right) \right| + \delta \right) \tag{3.3.11}$$

Comparing equation (3.3.11) with Eq. (3.2.10) shows that increasing n plays the same role as decreasing Δt in ensuring the existence of v_0. Therefore, updating the discrete-time sliding control at a faster rate than the subsystem simulators can guarantee that the system trajectory converges to the sliding manifold Eq. (3.2.3). The multi-rate approach just runs the Boundary Condition Coordinator faster without changing the step size of other subsystem simulators. This is a major advantage over the single-rate approach.

One should note that convergence is achieved under the assumption that all the subsystem simulators are stable. The effect of the sliding controllers on the subsystem simulators will be discussed in Chapter 4.

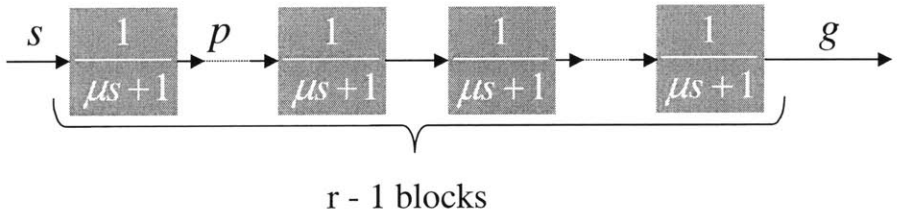


Figure 3.4.1. s – g relation viewed as linear filters.

3.4 The Accuracy of the Original Algebraic Constraint

Since the sliding function is defined as

$$s = \left(\mu \frac{d}{dt} + 1 \right)^{(r-1)} g , \tag{3.4.1}$$

where r is the index of the coupled system, the algebraic constraint g can be viewed as a filtered version of the sliding function s. Figure 3.4.1 shows that the sliding function s passes $(r - 1)$

filter blocks of $\frac{1}{\mu s+1}$ and becomes the original high index algebraic constraint g. Note that s in the linear filter blocks denotes the Laplace operator.

Let the output of the first filter be p, and we have

$$p(t)=\frac{1}{\mu}\int_0^t e^{-\frac{t-\tau}{\mu}} s(\tau)d\tau . \tag{3.4.2}$$

Once the system trajectory reaches the boundary layer of the sliding manifold, it exhibits the sliding mode and stays within the boundary layer forever. So the sliding function satisfies

$$|s| \le \sup_D(|R|), \tag{3.4.3}$$

where R can be defined for the single-rate or the multi-rate DTSM method. Then the magnitude of the algebraic constraint is

$$|p(t)| \le \frac{|s(\tau)|}{\mu}\int_0^t e^{-\frac{t-\tau}{\mu}} d\tau = \frac{|s(\tau)|}{\mu}\mu\left(1-e^{-\frac{t}{\mu}}\right) \le \sup_D(|R|). \tag{3.4.4}$$

After (r - 1) filter blocks, we have the accuracy of the algebraic constraint as follows:

$$|g| \le \sup_\Omega(|R|). \tag{3.4.5}$$

3.5 Example

3.5.1 Model

An example of Co-Simulation is given for the system shown in Fig. 3.5.1. The system consists of two subsystems: a hydraulic subsystem and a mechanical subsystem. The oil pump creates high pressure that drives oil to flow through a long and narrow pipe. The oil-activated piston is connected to a fixed elastic object. Bondgraph models of the two subsystems are shown in Fig. 3.5.2. Since the two subsystem simulators are coded independently and both simulators take an effort as input, there is a causal conflict, i.e. algebraic constraint, at the interface. It is possible to map the flow inertance onto the mechanical domain, and combine the two inertances to eliminate

the causal conflict. However, this approach destroys the modularity of one of the two subsystem simulators.

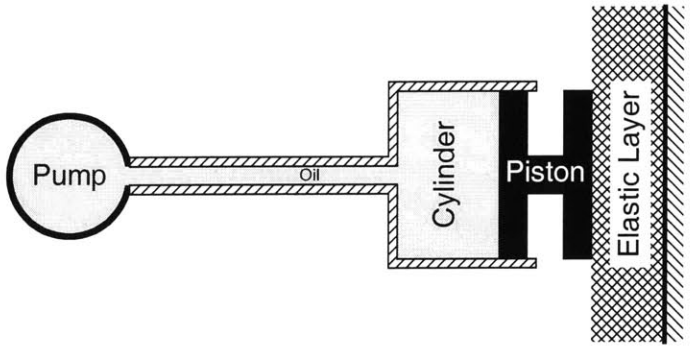


Figure 3.5.1 Coupled hydraulic and mechanical systems.

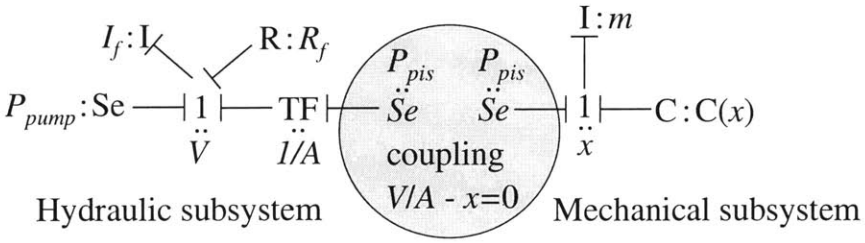


Figure 3.5.2 Causal conflict between two subsystems.

The state space model for the hydraulic subsystem is

$$\begin{aligned} \dot{V} &= Q \\ \dot{Q} &= -\frac{R}{I}Q + \frac{P_{pump}}{I} - \frac{P_{pis}}{I} \end{aligned}, \tag{3.5.1}$$

and the state space model for the mechanical subsystem is

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -\frac{k}{m}x^3 + \frac{A}{m}P_{pis} \end{aligned}, \tag{3.5.2}$$

where V, Q, x, v, P_{pump}, P_{pis}, R, I, m, k and A are the cylinder volume, flow rate, piston displacement, velocity, pump pressure, interface pressure, flow resistance, flow inertance, piston

mass, nonlinear spring constant, and piston cross-section area, respectively. The generalized displacement variables satisfy the continuity equation at the interface:

$$\frac{V}{A} - x = 0. \tag{3.5.3}$$

We put Eqs. (3.5.1), (3.5.2) and (3.5.3) together to obtain an index 3 DAE. We define $\mathbf{x} = [V\ Q\ x\ v]^T$ and $z = P_{pis}$, and the multi-rate DTSM control law Eq. (3.3.5) is used to construct the Boundary Condition Coordinator.

3.5.2 Results

The subsystems are integrated by using the simple forward Euler's method. The step size for the subsystem simulator is 0.1. The parameter μ is 0.1. The minimum magnitude of the control input v_0 is 1000. Figure 3.5.3 shows the simulation results of an inconsistent initial condition case. The initial values of all state variables and constrained input are zero, except that V starts from 0.0001. The Boundary Condition Coordinator is running at the same rate as the subsystem simulators, i.e., $n = 1$. The results show that the BCC is able to provide the boundary condition that makes the two subsystem simulators converge to the algebraic constraint. Larger n makes the sliding mode converge to the sliding manifold faster as expected. The oscillation of s is hardly noticeable at $n = 5$.

Figure 3.5.4 shows the results of another set of simulations. In this case, the initial condition is consistent, but the pump generates a sinusoidal pressure fluctuation throughout the simulation. Other conditions of the simulation remain unchanged except that we have to use a large n to stabilize the sliding function s. The simulation explodes when n is smaller than 16. Plot D shows that the accuracy of algebraic constraint increases with increasing n.

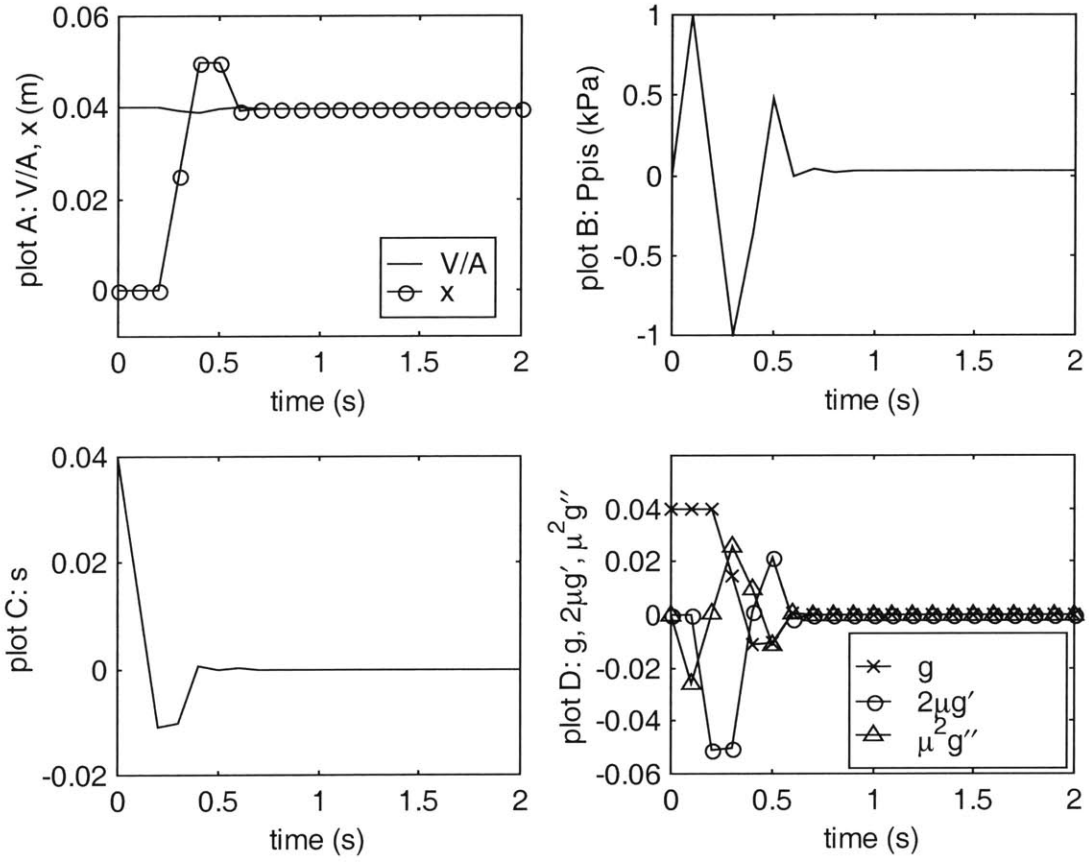


Figure 3.5.3 Results of the inconsistent initial condition.

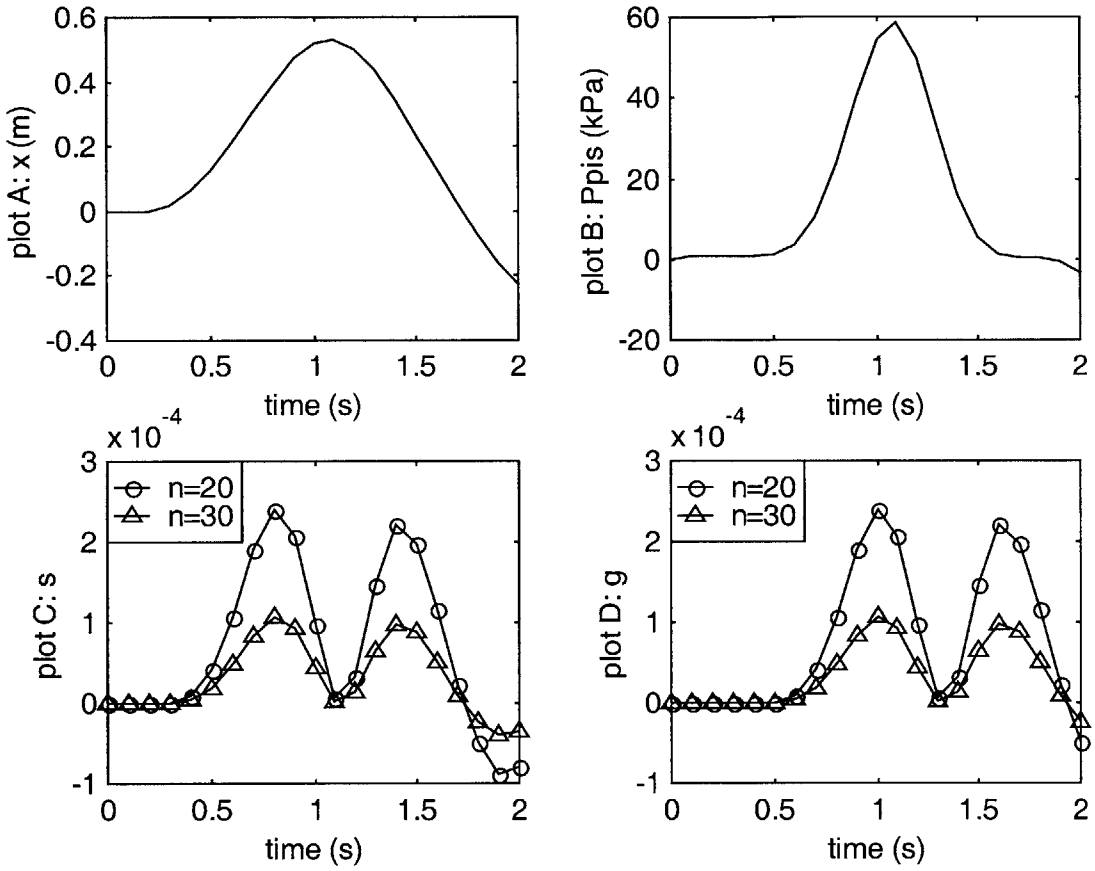


Figure 3.5.4 Results of consistent initial condition.

3.6 Extension to Multiple Subsystems

So far, the Co-Simulation of two subsystems coupled by a single algebraic constraint is studied. This section shows that the DTSM method can be extended to multiple subsystems having multiple constraints. However, in order to keep the mathematical notation simple and focus on the important findings, the following chapter will be based on two conflicting subsystems.

It is assumed that all the subsystems are physical systems connected through energetic junctions [Karnopp et al, 1990]. There are two types of junctions and they can be described by the following junction equations:

$$e\sum_{j}^{m} f_j = 0 \tag{3.6.1}$$

and

$$f\sum_{j}^{m} e_j = 0, \tag{3.6.2}$$

where e and f are effort and flow variables, respectively. Equation (3.6.1) describes the common effort junction, of which the effort variables, such as force, pressure, voltage, etc., satisfy the continuity condition. The flow variables of the common effort junction satisfy the compatibility condition. This is a generalization of Kirchhof's current law. Equation (3.6.2) defines the common flow junction, which is symmetric to the common effort junction. The common effort junction is used in the following discussion since the common flow junction exhibits similar behavior.

It is convenient to use bond graph to represent the energetic junctions and causal relations of the connected subsystems (Karnopp et al, 1990). Figure 3.6.1 shows a common effort junction with m connected subsystems.

If one and only one subsystem provides an effort to the junction, the junction is free of causal conflict. Figure 3.6.1 shows such a case. Subsystem 1 provides an effort to the junction and the junction passes the effort to all other connected subsystems.

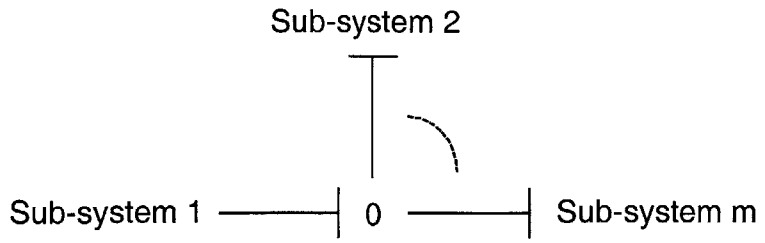


Figure 3.6.1 A common effort junction with compatible boundary conditions.

If no subsystem dictates the effort of the junction, such as the case shown in Fig. 3.6.2, we have to use the compatibility property of the flow variables to find out the common effort variable. The compatibility equation

$$g = \sum_{j}^{m} f_j = 0 \tag{3.6.3}$$

is used to determine one unknown common effort e. This is a scalar case of causal conflict, and the unknown common effort e is the boundary variable z.

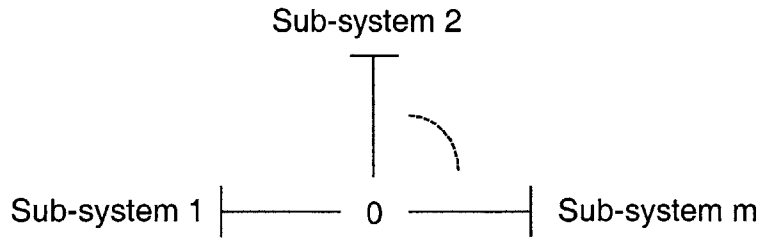


Figure 3.6.2 A common effort junction without a subsystem dictating the effort variable.

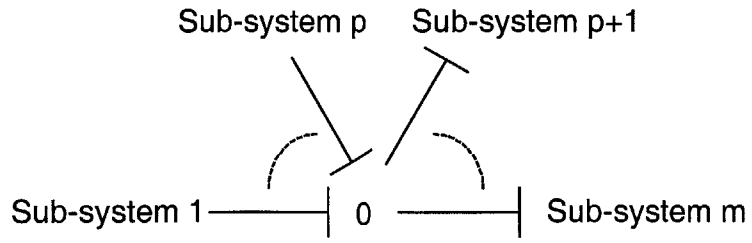


Figure 3.6.3 A common effort junction with multiple subsystems providing the effort variables.

On the other hand, if multiple subsystems provide effort outputs to the same junction, a different type of causal conflict occurs. As shown in Fig. 3.6.3, subsystems 1 through p all provide effort outputs, which must be the same. We have to determine p flow variables to supply to subsystems 1 through p so that all p subsystems provide the same effort output satisfying the continuity condition. We have p unknown flow variables treated as boundary variables $z_1, \ldots, z_p$. The continuity condition is given by

$$e_1 = e_2 = \cdots = e_p . \tag{3.6.4}$$

These include $p - 1$ independent constraint conditions. One choice of such $p - 1$ constraint equations are

$$\begin{aligned} g_1 &= e_j - e_1 = 0 \\ &\vdots \\ g_{j-1} &= e_j - e_{j-1} = 0 \\ g_j &= e_j - e_{j+1} = 0 \\ &\vdots \\ g_{p-1} &= e_j - e_p = 0 \end{aligned} , \tag{3.6.5}$$

where j can be any one of the p subsystems. The compatibility condition provides another constraint

$$g_p = \sum_j^m f_j = 0 \tag{3.6.6}$$

or

$$g_p = \sum_{j=1}^{p} z_j + \sum_{j=p+1}^{m} f_j = 0 . \tag{3.6.7}$$

Therefore, we have p constraint equations and p unknown boundary variables z's. This is a vector case of causal conflict. The index number of the algebraic constraint formed by the compatibility equation is one, while the indices of the constraints formed by the continuity

equations depend on the p subsystems. The index 1 constraint, Eq. (3.6.7), can be used for reducing the number of z variables. Solving it for $z_p = -\sum_{j=1}^{p-1} z_j - \sum_{j=p+1}^{m} f_j = 0$.

To deal with multiple constraint, the sliding variable must be extended to a vectorial variable

$$\mathbf{s} = [s_1, s_2, \ldots, s_j, \ldots, s_p]^T. \tag{3.6.8}$$

Each component of **s** is defined as given by Eq. (2.3.4) with its own index number. The DTSM method can be extended to a class of multiple constraint problems as long as the vector index exists. The sensitivity Jacobian **Jz** becomes a Jacobian matrix and a matrix norm must be used for the convergence condition. If the algebraic constraints are defined as Eqs. (3.6.5) to (3.6.7), the Jocobian is sparse and can easily be inverted.

IV Input-Output Linearization Analysis of Co-Simulation

4.1 The Effect of μ

4.1.1 Example

The example in Chapter 3 demonstrates that the Boundary Condition Coordinator is able to enforce the algebraic constraint in a stable manner through appropriate choice of values for n and v_0, as predicted by the multi-rate DTSM method. However, the effect of the DTSM controllers on the subsystem simulators has not been studied yet. Both the single-rate and the multi-rate DTSM analyses presented so far are based on the assumption that all subsystem simulators are stable. Therefore the stability of subsystem simulators when the algebraic constraint is replaced by the DTSM-based Boundary Condition Coordinator will be investigated in this chapter.

Using the exemplary system in Chapter 3 as a starting point, a set of simulations are performed, starting from a consistent set of initial conditions with the coupled system excited by a sinusoidal input from the pump. The simulations are run three times under the same conditions, except for the choice of parameter μ which defines the internal dynamics of the sliding manifold, is different. The step size Δt, the minimum magnitude of the control input v_0 and the multi rate parameter n is 0.1, 1000, and 20, respectively. The parameter μ in Eq. (2.3.4) takes on value of 1, 0.1 to 0.04 in the three simulations and the results are given in Figs. 4.1.1, 4.1.2, and 4.1.3, respectively.

Comparing the results shown in Fig. 4.1.1 and Fig. 4.1.2, we find that smaller μ leads to smaller errors in both the sliding function s and the algebraic constraint g. However, there is a lower threshold for which this observation holds: the simulation becomes unstable when an even smaller μ ($\mu = 0.04$) is used as shown in Fig. 4.1.3. There is no indication of instability of any cause for the sliding function for different values of parameter μ in the analysis of DTSM. Thus, the source of instability must come from the subsystem simulators.

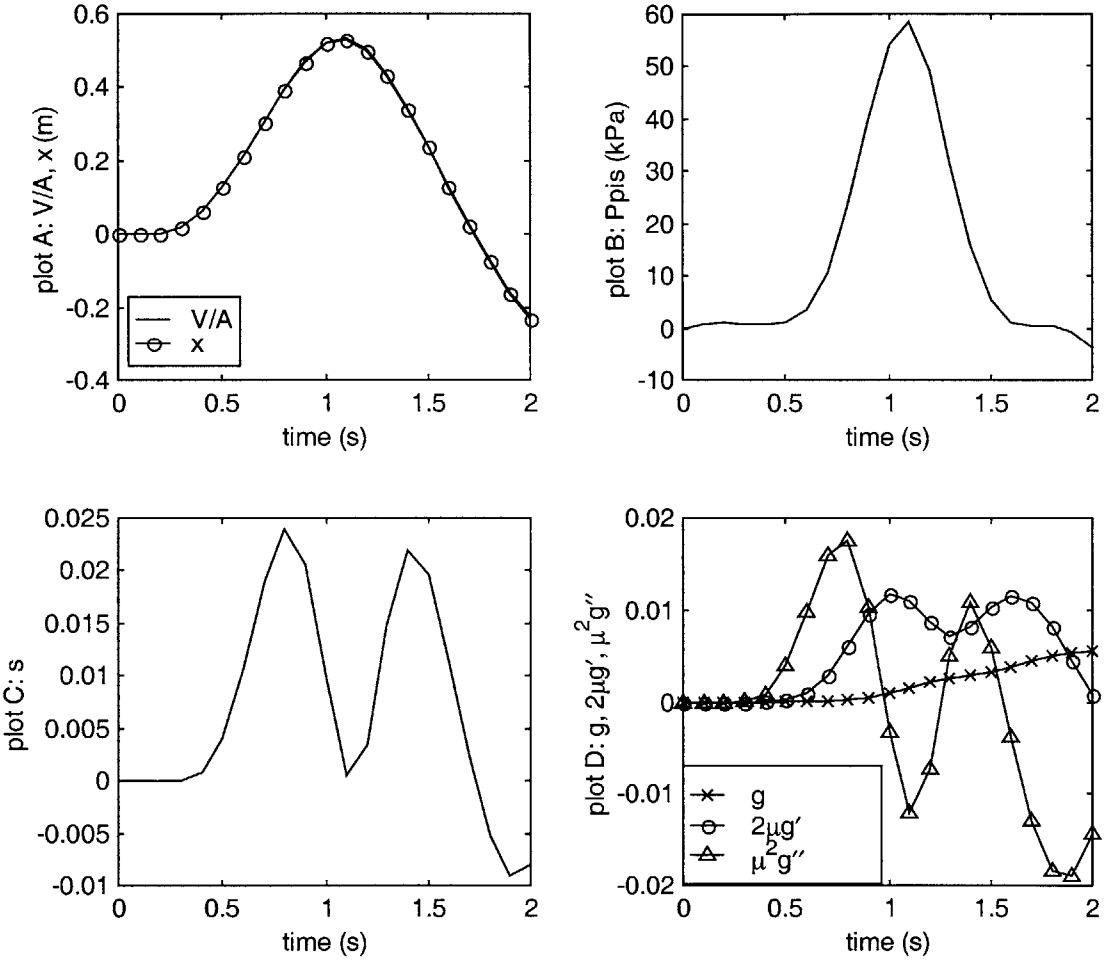


Figure 4.1.1. Effects of parameter μ ($\mu = 1$).

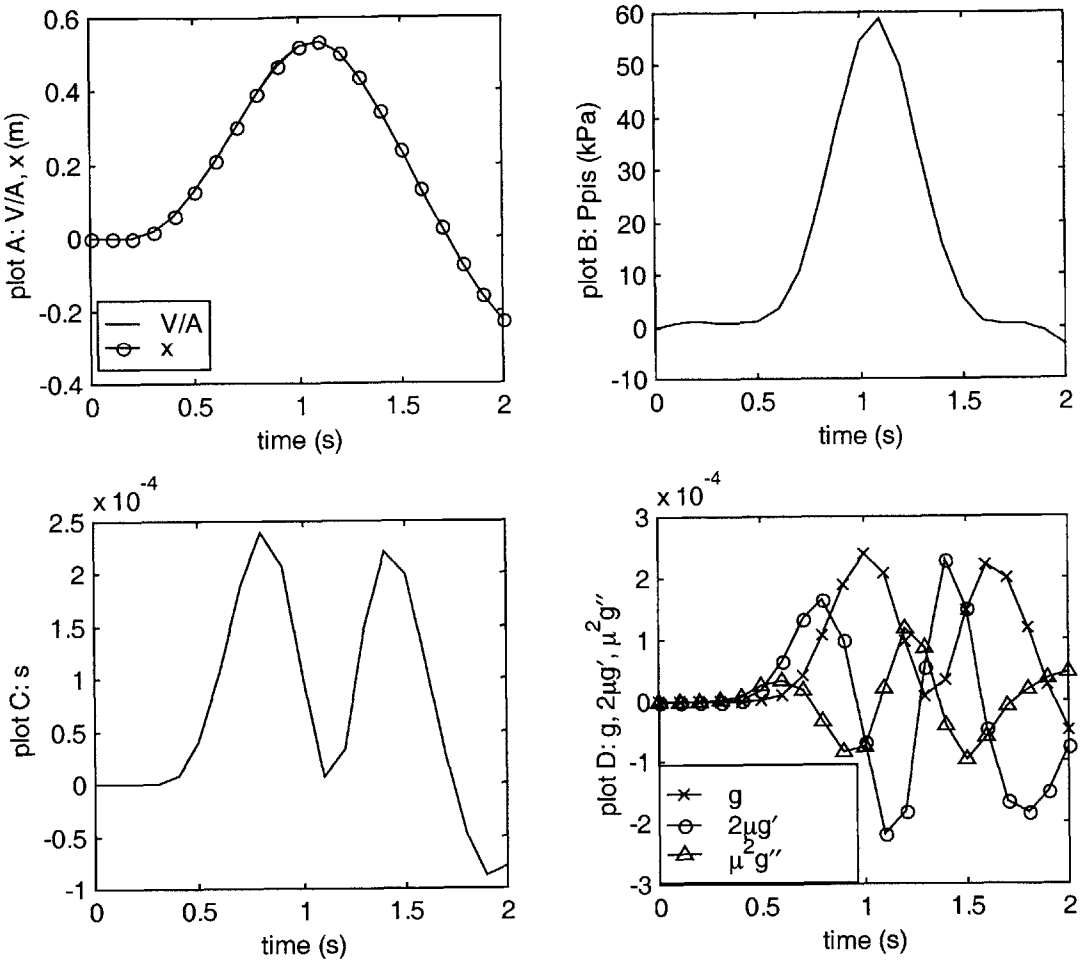


Figure 4.1.2. Effect of parameter μ ($\mu = 0.1$).

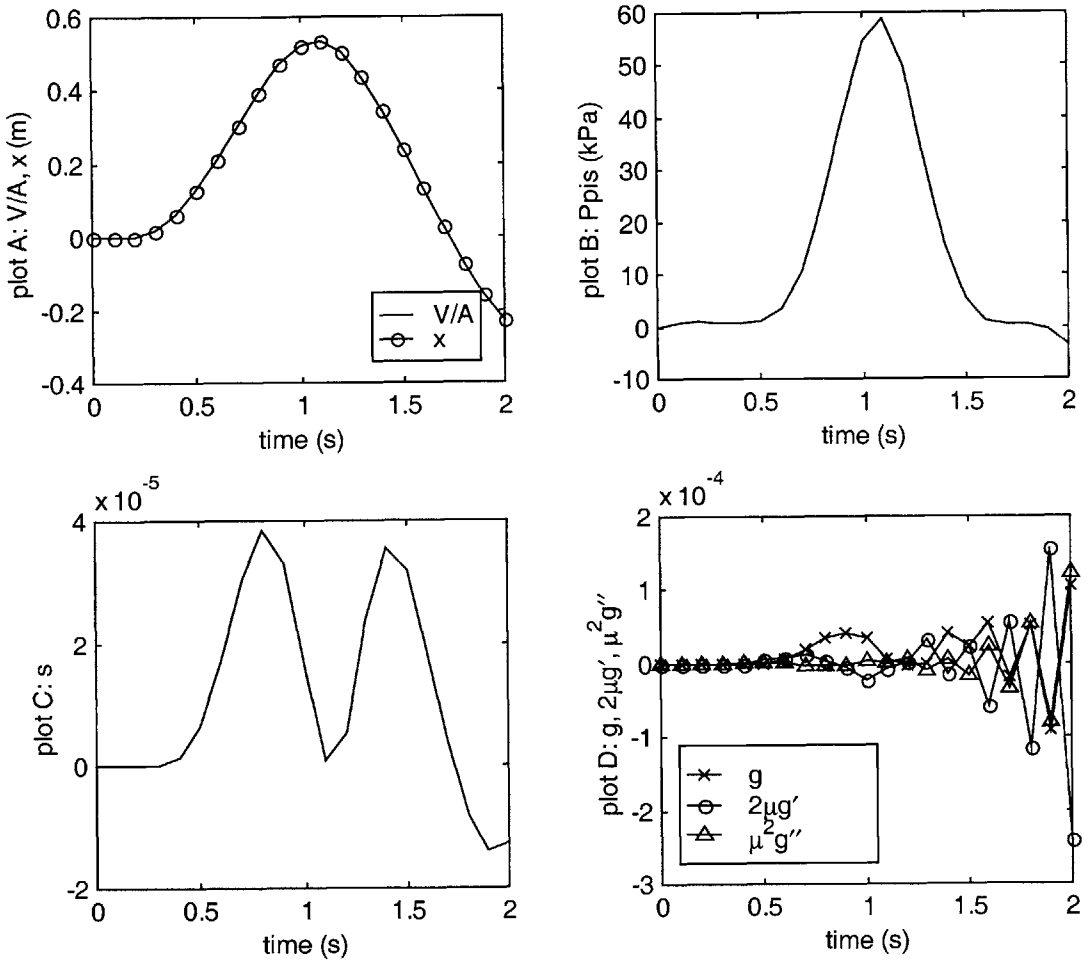


Figure 4.1.3 Effect of parameter μ ($\mu = 0.04$).

4.1.2 A Linear Case

In order to learn the cause of the instability, a simple linear case is investigated first. Consider the following index 3 linear DAE system:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}z \\ g(\mathbf{x}) &= \mathbf{C}\mathbf{x} = 0 \end{aligned}, \tag{4.1.1}$$

The values of the coefficient matrices of the system are given by:

$$\mathbf{A} = \begin{bmatrix} -20 & 1 & 1 & 0 \\ 4 & -10 & 0 & 0 \\ 15 & -20 & -50 & 5 \\ -2 & 3 & 5 & -9 \end{bmatrix}, \tag{4.1.2}$$

$$B = [0 \quad 1 \quad 0 \quad 0]^T \tag{4.1.3}$$

and

$$C = [1 \quad 0 \quad 1 \quad 0]. \tag{4.1.4}$$

This system can be viewed as resulting from two algebraically coupled subsystem simulators.

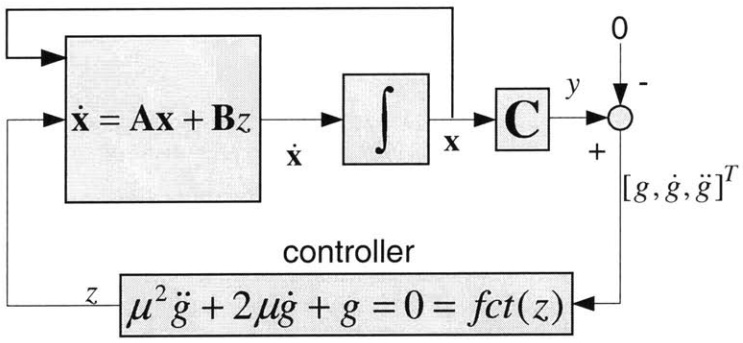


Figure 4.1.4 The block diagram of a linear system.

A sliding manifold is defined using Eq. (2.3.4) and the sliding function is enforced by a DTSM controller. The DAE realization can be seen as a control problem with a block diagram as shown in Fig. 4.1.4. The DTSM controller block effectively moves two closed-loop poles to $-1/\mu$ without adding any nonlinearity to the system. A plot of the closed-loop poles with

parameter μ varying from 0.025 to 0.01 is shown in Fig. 4.1.5. Since the system is a fourth order system, it has 4 poles. Two of the 4 poles are located on the right side of the plot and they do not move when the parameter μ changes. The two other poles are coincident and are thus shown with one pole on top of the other. The two coincident poles move from –40 to –100 when μ changes from 0.025 to 0.01. Clearly, imposing dynamics of the sliding manifold Eq. (2.3.4) to the system changes the eigenvalues of the closed-loop system. This shows that the Boundary Condition Coordinator based on the DTSM controller changes the dynamics of the subsystems.

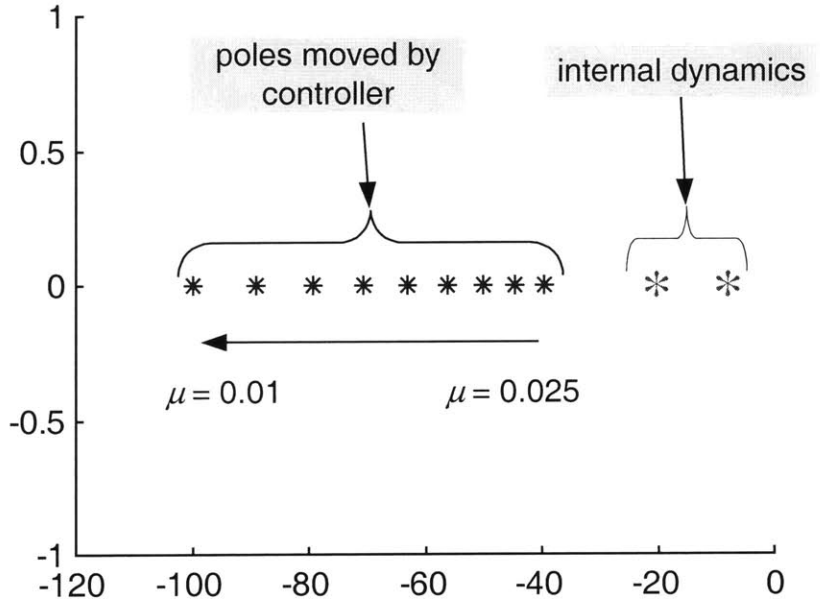


Figure 4.1.5 Effect of μ in a linear system.

If the study of the closed-loop system were undertaken in an ideal continuous domain, the conclusion would be that small μ is preferred to make the system converge to the algebraic constraints faster and with smaller error. However, some subsystems may be simulated using certain explicit numerical integration methods which do not provide for absolute numerical stability. For instance, to stably simulate a dynamic system with fast modes requires a very small step size. If the step size is not small enough, the fast dynamics will cause numerical instability. Therefore, guaranteeing the stability of the DTSM controller is not enough for ensuring the stability of the entire Co-Simulation. The fast dynamics of the sliding manifold

may make the overall system numerically unstable. Hence, it is necessary to ensure that the choice parameter μ does not result in instability of the subsystem simulators.

4.2 Influence of DTSM on Stability of Subsystems

Consider the two algebraically coupled subsystems shown in Fig 2.5.5. Ignoring subsystem C, the remaining subsystem simulators A and B and the Boundary Condition Coordinator can be represented in a control block diagram as shown in Fig 4.2.1, assuming the index is 3. Since the purpose of this chapter is to study the stability issue caused by the additional discrete-time sliding controller, the instability problem arising from the interconnection of the two subsystems is not considered here. Two stable subsystems connected in a feedback configuration may become unstable then the simulation should reflect this instability. Therefore, we can replace the dynamic subsystem A with a time function $y_A(t)$ and we can assume that $y_A(t)$ and it time derivatives are small. The block diagram of Fig. 4.2.1 becomes the block diagram shown in Fig. 4.2.2 and the DTSM controller pushes the output of subsystem B, $y_B(t)$, to track the desired trajectory $y_A(t)$. The DAE realization problem is thus converted into a trajectory tracking problem. From the definition of the sliding manifold:

$$s(\mathbf{x},z,t)=\left(\mu\frac{d}{dt}+1\right)^{r-1} g=0\,, \tag{2.3.4}$$

note that this problem is essentially equivalent to an Input-Output linearization problem.

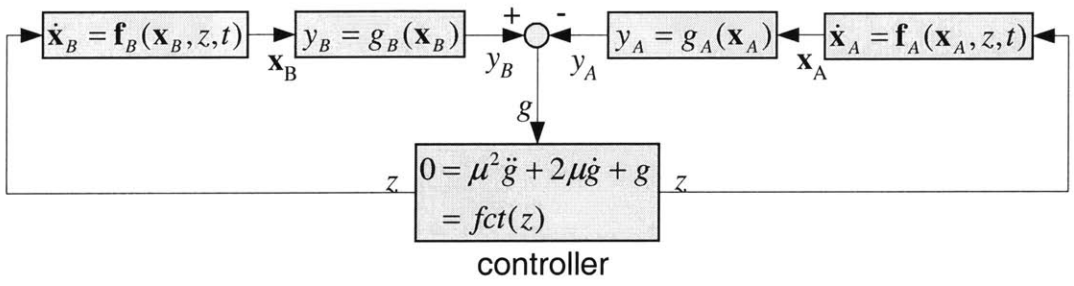


Figure 4.2.1 Block diagram view of realization of algebraically coupled systems.

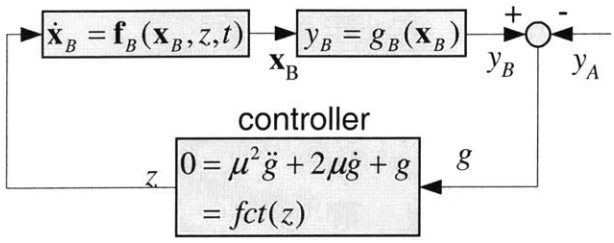


Figure 4.2.2 Block diagram of Input-Output linearization control.

The feedback linearization techniques of nonlinear control theory use the feedback controller to convert a nonlinear system into a linear or a partially linear one [Khalil 1996; Vidyasagar 1993]. The sliding manifold Eq. (2.3.4) can be viewed as a special case of an exponentially stable tracking error dynamics. In general, the tracking error dynamics may have different eigenvalues. Since only part of the system dynamics, i.e. the output, is linearized, this feedback linearization is an input-output linearization. The internal dynamics of the input-output linearized system is often nonlinear and we have to guarantee that these internal dynamics will not explode due to the external feedback.

4.2.1 Subsystems in Partial Controller Canonical Form

Many physical systems such as the one shown in Fig 4.2.3 are modeled in a special way. The following equations are the system model for the system shown in Fig. 4.2.3:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f_2(\mathbf{x}, z, t) \\ \dot{x}_3 &= f_3(\mathbf{x}, t) \\ &\vdots \\ \dot{x}_n &= f_n(\mathbf{x}, t) \\ y &= x_1 \end{aligned} \qquad (4.2.1)$$

Variables x_1, x_2, and $\dot{x}_2$ is displacement, velocity and acceleration of the mass m_1, respectively. The displacement x_1 is the output of interest of the subsystem and z is the input that we can design. The input z does not affect the rest of the system $\dot{x}_3 \sim \dot{x}_n$, so the subsystem ($\dot{x}_1 \sim \dot{x}_2$) represents the external dynamic of this input-output pair. The external subsystem is in controller canonical form, in which only the last differential equation of the system can be nonlinear.

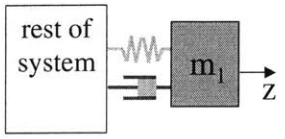


Figure 4.2.3 Example of systems in partial controller canonical form.

The control problem is to design a force input z to force the displacement output, y, to follow a desired output y_d. Defining the tracking error as:

$$e = y - y_d\,, \tag{4.2.2}$$

and using positive constants k_1, k_2 and k_3 so that the error dynamics are always stable:

$$k_1\ddot{e} + k_2\dot{e} + k_3 e = 0\,. \tag{4.2.3}$$

for the choice of constants k_1, k_2 and k_3 equal to μ^2, 2μ and 1, respectively, the error dynamics reduces to that of the sliding manifold given by Eq. (2.3.4). The error dynamics can be solved for the input z. The DTSM methods are thus able to generate the input z that forces the system trajectory to stay on the manifold.

Substituting the input z into Eq. (4.2.1) and closing the loop, we obtain

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \ddot{y}_d - \frac{2}{\mu}(x_2 - \dot{y}_d) - \frac{1}{\mu^2}(x_1 - y_d) \\
\dot{x}_3 &= f_3(\mathbf{x}, t) \\
&\vdots \\
\dot{x}_n &= f_n(\mathbf{x}, t)
\end{aligned} \tag{4.2.4}$$

The closed-loop system, Eq. (4.2.4), can be divided into two parts: a linearized external input-output dynamics ($\dot{x}_1 \sim \dot{x}_2$), and a nonlinear internal dynamics ($\dot{x}_3 \sim \dot{x}_n$) that is not directly influenced by input z. Since the input-output dynamics is now completely linear, we can place its poles at arbitrary location. Also, because of the linearity of the input-output dynamics, it is easy to find a parameter μ that guarantees numerical stability of the dynamics for a given explicit integration algorithm with a given step size Δt. The internal dynamics is not directly involved in the feedback loop and its stability can be inferred by studying the zero dynamics of the system.

The zero dynamics is defined as the internal dynamics ($\dot{x}_3 \sim \dot{x}_n$) when the output of the system y is constantly kept at zero. Zero dynamics is an intrinsic system property and thus is independent of control laws that keep the output constantly zero [Slotine and Li, 1991]. We can show that the closed-loop system is able to follow any desired trajectory, y_d, if the zero dynamics is exponentially stable in the sense of Lyapunov, and if the desired trajectory y_d and its relevant time derivatives are small.

Using a new vector $\mathbf{w}$ to represent state variables $\dot{x}_3 \sim \dot{x}_n$ and substituting Eq. (4.2.2) into Eq. (4.2.4), the closed-loop system becomes:

$$\dot{\mathbf{e}} = \mathbf{A}\mathbf{e} \tag{4.2.5}$$

$$\dot{\mathbf{w}} = \mathbf{r}(\mathbf{e} + \mathbf{m}, \mathbf{w}), \tag{4.2.6}$$

where vectors $\mathbf{m}$ and $\mathbf{e}$ are defined as:

$$\mathbf{m} = \left[y_d, \dot{y}_d, \cdots, y_c^{(r-1)}\right]^T \tag{4.2.7}$$

and:

$$\mathbf{e} = [e, \dot{e}]^T, \tag{4.2.8}$$

respectively. Assuming Euler's method is used, the discrete-time form of the system is:

$$\mathbf{e}_{k+1} = \mathbf{e}_k + \Delta t \cdot \mathbf{A}\mathbf{e}_k = \mathbf{B}\mathbf{e}_k \tag{4.2.9}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k). \tag{4.2.10}$$

Function $\mathbf{r}$ is locally Lipschitz since $\mathbf{f}$ and $\mathbf{x}$ are smooth. Therefore, we have:

$$\left\|\mathbf{r}(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k) - \mathbf{r}(\mathbf{0}, \mathbf{w}_k)\right\| \le \Delta t^{-1} \cdot k_1\left(\left\|\mathbf{e}_k + \mathbf{m}_k\right\|\right). \tag{4.2.11}$$

Since the zero dynamics

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{0}, \mathbf{w}_k) \tag{4.2.12}$$

are locally exponentially stable, the converse Lyapunov theorem guarantees the existence of a Lyapunov function U such that:

$$k_2\|\mathbf{w}_k\|^2 \le U(\mathbf{w}_k) \le k_3\|\mathbf{w}_k\|^2 \tag{4.2.13}$$

$$\Delta U_k = U(\mathbf{w}_{k+1}) - U(\mathbf{w}_k) \le -k_4\|\mathbf{w}_k\|^2 \tag{4.2.14}$$

$$\|\Delta U_k\| \le k_5\|\Delta \mathbf{w}_k\|, \tag{4.2.15}$$

where k_1, k_2, k_3, k_4, and k_5 are positive numbers.

Let V be a candidate Lyapunov function:

$$V_k = \mathbf{e}_k^T\mathbf{P}\mathbf{e}_k + k_6 U(\mathbf{w}_k), \tag{4.2.16}$$

where $\mathbf{P}$ is a positive definite matrix satisfying

$$\mathbf{B}^T\mathbf{P}\mathbf{B} - \mathbf{P} = -\mathbf{I} \tag{4.2.17}$$

The forward difference of the Lyapunov function candidate is

$$\begin{aligned}
\Delta V_k &= V(\mathbf{e}_{k+1} + \mathbf{m}_{k+1}, \mathbf{w}_{k+1}) - V(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k) \\
&= \mathbf{e}_k^T\mathbf{B}^T\mathbf{P}\mathbf{B}\mathbf{e}_k - \mathbf{e}_k^T\mathbf{P}\mathbf{e}_k + k_6[U(\mathbf{w}_{k+1}) - U(\mathbf{w}_k)] \\
&= -\|\mathbf{e}_k\|^2 + k_6 U(\mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k)) - k_6 U(\mathbf{w}_k) \\
&= -\|\mathbf{e}_k\|^2 + k_6 U(\mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{0}, \mathbf{w}_k)) + k_6 U(\mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k)), \\
&\quad - k_6 U(\mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{0}, \mathbf{w}_k)) - k_6 U(\mathbf{w}_k) \\
&= -\|\mathbf{e}_k\|^2 + k_6[U(\mathbf{w}_{k+1}) - U(\mathbf{w}_k)] \\
&\quad + k_6[U(\mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k)) - U(\mathbf{w}_k + \Delta t \cdot \mathbf{r}(\mathbf{0}, \mathbf{w}_k))]
\end{aligned} \tag{4.2.18}$$

Applying Eqs. (4.2.14) and (4.2.15), we have

$$\Delta V_k \le -\|\mathbf{e}_k\|^2 - k_6 k_4\|\mathbf{w}_k\|^2 + k_6 k_5 \Delta t\|\mathbf{r}(\mathbf{e}_k + \mathbf{m}_k, \mathbf{w}_k) - \mathbf{r}(\mathbf{0}, \mathbf{w}_k)\|. \tag{4.2.19}$$

Applying Eq. (4.2.11), we have

$$\Delta V_k \le -\|\mathbf{e}_k\|^2 - k_6 k_4\|\mathbf{w}_k\|^2 + k_6 k_5 k_1(\|\mathbf{e}_k\| + \|\mathbf{m}_k\|). \tag{4.2.20}$$

We can choose k_6 and a positive constant k_7 such that

$$k_6 k_5 k_1 + k_7 < 1, \tag{4.2.21}$$

and finally we arrive at

$$\Delta V_k \le -k_7\|\mathbf{e}_k\|^2 - k_6k_4\|\mathbf{w}_k\|^2 + k_6k_5k_1\|\mathbf{m}_k\|. \tag{4.2.22}$$

The forward difference of the Lyapunov function is negative when $\|\mathbf{e}_k\|$ and $\|\mathbf{w}_k\|$ are large and the desired trajectory $\mathbf{m}_k$ is bounded. This implies that $\|\mathbf{e}_k\|$ and $\|\mathbf{w}_k\|$ are also bounded [Slotine and Li, 1991].

4.2.2 General Nonlinear Subsystems General nonlinear subsystems that do not conform to the partial controller canonical form can be converted into normal form first. Applying a coordinate transformation, a general nonlinear system

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, z, t) \\ y &= g(\mathbf{x}) \end{aligned} \tag{4.2.23}$$

is converted into the normal form:

$$\begin{aligned} \dot{\zeta}_1 &= \zeta_2 \\ &\vdots \\ \dot{\zeta}_{r-2} &= \zeta_{r-1} \\ \dot{\zeta}_{r-1} &= v(\zeta, \eta, z, t) \\ \dot{\eta} &= \mathbf{w}(\zeta, \eta, t) \\ y &= \zeta_1 \end{aligned} \tag{4.2.24}$$

The tracking error and the sliding manifold are defined as before:

$$e = y - y_d \tag{4.2.25}$$

$$s = \left(\mu\frac{d}{dt} + 1\right)^{r-1} e = 0, \tag{4.2.26}$$

where $(r - 1)$ is the relative order. A multi-rate DTSM controller provides the input z to be used in Eq. (4.2.24).

The transformed system, Eq. (4.2.24), is in the controller canonical form and the zero dynamics in the discrete-time form

$$\eta_{k+1} = \eta_k + \Delta t \cdot \mathbf{w}(\mathbf{0}, \eta_k, t_k) \tag{4.2.27}$$

can be studied using input-output linearization analysis as described in the previous section. However, since the transformation between the **x** space and the (ζ, η) space is nonlinear, a stable choice of μ and Δt in the (ζ, η) space for a given explicit integration method does not guarantee the stability in the **x** space for the same integration method.

From the assumption that all algebraic constraints are caused by causal conflicts, the input and output considered here are always collocated. Therefore the zero dynamics in the continuous-time domain is always stable since the input-output relation is minimal phase. The task is then to seek a choice of μ and Δt such that the numerical simulation of the system is stable for given explicit integration algorithm. Finding a Lyapunov function for a general nonlinear system is often difficult. Hence, it may be more efficient to find μ and Δt by trial and error.

A stable choice of μ and Δt will be stored as properties of the subsystem in the simulator. When two subsystem simulators are algebraically coupled, the Co-Simulation environment will compare parameters μ from of the subsystems and use the larger one to form the sliding function. This guarantees that the subsystem simulators are stable when they are connected through a Boundary Condition Coordinator. This together with the findings in Chapter 3 guarantees stability when we convert the algebraic constraints into the Boundary Condition Coordinator.

4.3 Subsystem Relative Order and the Index of DAE

The function g in the algebraic constraint $g = 0$ is a linear combination of all subsystem outputs that are constrained algebraically. Each output has its own relative order, which is an intrinsic property of the subsystem. When the constraint equation is successively differentiated with respect to time, the constrained input z will appear sooner or later in one or all of the outputs. The index of the DAE then equals one plus the number of differentiations after which at least one of the output functions yields the constrained input z. Therefore, the relative order of all input output pairs of a subsystem is stored in that subsystem's simulator. When subsystem simulators are connected through an algebraic constraint, the index number can then be easily obtained.

V Co-Simulation with Minimum Information Disclosure

5.1 Hybrid Implementation of Boundary Condition Coordinator

In Chapter 3, the Discrete-Time Sliding Mode methods of converting the incompatible boundary conditions into compatible boundary conditions were developed. The effect of the DTSM-based Boundary Condition Coordinators on the subsystem simulators was investigated. Therefore we know how to simultaneously run the subsystem simulators coupled with algebraic constraints without changing the stability property of the overall interconnected system. We also know what types of information are required to set up and run the Co-Simulation. This knowledge is now used to implement the Co-Simulation.

Figure 5.1.1 summarizes how the Boundary Condition Coordinator is constructed and run. Complete models of the algebraically coupled subsystem simulators are needed to construct the BCC. The model of a dynamic subsystem includes state equations and output equations. The coupling condition which describes exactly how the subsystems connected is also required. The state equations and the output equations of any subsystem are the properties of that subsystem, and they do not change when the subsystem is used as part of various large systems. The coupling condition, which can be different, is supplied by the system integration engineer who builds the coupled simulation based on the subsystem simulators.

Reducing the indices of the differential algebraic systems involves symbolic differentiations and equation manipulations. Further similar operations are required to construct the Discrete-Time Sliding Mode based controller. A symbolic procedure called the System Analyzer can be built to automate the construction of the Boundary Condition Coordinator. Maple, the symbolic mathematical language, is used to build such symbolic procedures. The System Analyzer takes the entire models stored in the subsystem simulators and produces the discrete-time dynamic system that provides the compatible boundary condition. The System Analyzer also obtains parameter μ and step size information from the subsystem simulators.

During the operation of the Co-Simulation, the Boundary Condition Coordinator requires the state variables from the algebraically coupled subsystem simulators. This is because s, $\mathbf{Jx}$, Jt, and Jz are functions of the state variables, the constrained input and time.

Since one of the terms of the DTSM control law is given by:

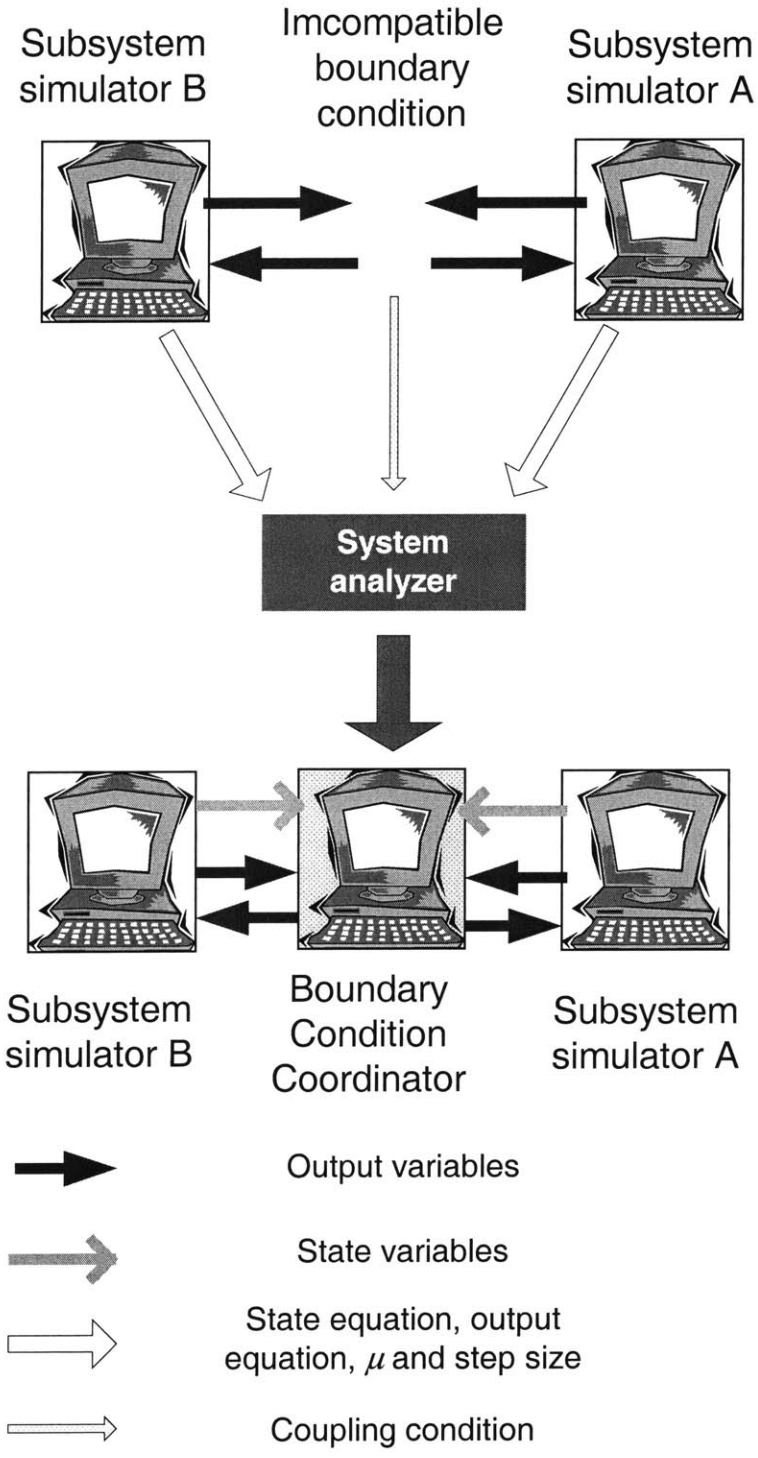


Figure 5.1.1. Hybrid implementation of the Co-Simulation.

$$\alpha_k = \mathbf{J}\mathbf{x}_k \Delta t \mathbf{f}(\mathbf{x}_k, z_k, t_k) + J t_k \Delta t, \tag{3.2.11}$$

the term $\mathbf{f}(\mathbf{x}_k, z_k, t_k)$ is calculated in both subsystems and in the Boundary Condition Coordinator. This redundant computation is inefficient, especially for large or complex subsystems. To increase the computational efficiency of the simulation, the results of the subsystem simulators can be used in the Boundary Condition Coordinator computation:

$$\alpha_k = \mathbf{J}\mathbf{x}_k (\mathbf{x}_{k+1} - \mathbf{x}_k) + J t_k \Delta t. \tag{5.1.1}$$

The Boundary Condition Coordinator appears to be using future information due to substitution. However, if we always update the dynamic subsystems before the Boundary Condition Coordinator, $\mathbf{x}_{k+1}$ will be readily available for use by the Boundary Condition Coordinator. Table 5.1.1 shows the computational sequence of the example system shown in Fig. 2.5.5. In order not to perform redundant computation, the BCC should be always updated after subsystems A and B since the updated information from subsystem A and B can be used in updating the BCC. However, subsystem C, which is not connected to the BCC, can be updated at any time during one step of the Co-Simulation.

Table 5.1.1 Computational sequence of the Co-Simulation.

Sequence	State Variable Updating Equation
Subsystem A	$\mathbf{x}_{A,k+1} = \mathbf{x}_{A,k} + \Delta t \cdot \mathbf{f}_A(\mathbf{x}_k, z_k, t_k)$
Subsystem B	$\mathbf{x}_{B,k+1} = \mathbf{x}_{B,k} + \Delta t \cdot \mathbf{f}_B(\mathbf{x}_k, z_k, t_k)$
Subsystem C	$\mathbf{x}_{C,k+1} = \mathbf{x}_{C,k} + \Delta t \cdot \mathbf{f}_C(\mathbf{x}_k, t_k)$
BCC	$z_{k+1} = z_k - v_0 \,\mathrm{sat}\left(J z_k^{-1}(s_k + \mathbf{J}\mathbf{x}_k(\mathbf{x}_{AB,k+1} - \mathbf{x}_{AB,k}) + J t_k \Delta t) v_0^{-1}\right)$

This substitution trades flexibility of the execution sequence for better efficiency. A certain execution sequence is always required for the Co-Simulation systems, which consist of algebraic subsystems as well as dynamic subsystems. The air conditioner system is one such example in which the expansion valves and the compressor are modeled as algebraic systems.

Algebraic systems cannot predict future states; they only can produce results at time t_{k+1} when inputs at time t_{k+1} are supplied. This requires the dynamic subsystem simulators supply the inputs for the algebraic subsystems before updating the algebraic subsystems. Further, even if the Co-Simulation system is a pure dynamic system, freedom of execution order is limited to a single step because we do not intend to apply any asynchronous integration algorithms in this research. Therefore, these extra requirement associated with removal of the redundancy does have any substantial drawbacks.

5.2 Disadvantages of Hybrid Implementation

The DTSM methods described in previous chapters may be improved for the purpose of Co-Simulation. The major drawback is that the previous DTSM method requires full information about the subsystems that are connected through the algebraic constraints. Subsystem simulators have to supply state equations, output equations, etc to build the DTSM-based Boundary Condition Coordinator before the simulation starts. This requirement forces the subsystem simulators to disclose their complete models. Subsystem simulator makers cannot keep their proprietary information secret under these conditions. This contradicts the concept of object-oriented simulation since the models of the subsystems are the implementation details and should be hidden from other objects.

In previous implementations of DTSM methods, the sliding mode and the Jacobians were obtained by symbolic manipulation. Symbolic processes can be automated using symbolic math languages, such as Maple. In the early stages of this research, a Maple V procedure was designed. It produced a dynamic approximation of any algebraic constraints using the SPSM method. However, when this symbolic procedure was applied to a simple nontrivial system [Asada et al, 2000], it produced 700 kB C of source code that contained the differential equation approximation of two algebraic constraints, without any numerical integration code. That symbolic procedure can be adapted to use the DTSM methods with little effort, but the code efficiency will be equally low.

Some subsystems contain functions that do not have a closed-form expression. Examples can be found in air-conditioner system components, where thermodynamic properties are supplied by lookup tables or curve fitting. Since the DTSM methods require Jacobian calculation, lookup tables or curve fitting will make the symbolic procedure impossible to use.

Therefore, the need to develop a pure numerical implementation of the DTSM methods exists. Such a numerical approach will address non-closed-form problems and greatly improve efficiency for subsystems with very complex closed-form representations.

5.3 Constructing Pure Numerical DTSM Using Minimum Information

The objective of improving the DTSM methods is to reduce information disclosure by subsystem simulators and to make the new algorithm more suitable for pure numerical computation. The computational efficiency is also a factor. Actually, allowing the use of the state variables at time t_{k+1} supplied by subsystem simulators makes the design of DTSM algorithms much more flexible. This feature may be exploited to improve efficiency and more importantly to minimize the information required by the Boundary Condition Coordinator from its neighboring dynamic subsystems. This will allow hiding of information, which is an important feature of the Co-Simulation environment.

The equivalent control is at the core of sliding controllers. Its definition is given by:

$$O_k = -Jz_k^{-1}(s_k + \alpha_k). \tag{3.2.14}$$

All three components of the equivalent control. (s, Jz and α) are functions of the state variables, the constrained input, and time. The current goal is search for alternative ways of obtaining these three components.

There are two ways of calculating the sliding function s. The way used in the hybrid implementation is to form s as a symbolic function of the independent variables first and then to evaluate s using the variables $\mathbf{x}$, t, and z. However, as described in Chapter 2, the algebraic constraints considered in this research all result from causal conflicts. Hence, the high index algebraic constraint term, g, is a linear combination of m output variables of the subsystems that are algebraically constrained:

$$g(\mathbf{x}, z, t) = [b_1 \quad b_2 \quad \cdots \quad b_m] \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}. \tag{5.3.1}$$

The sliding function is nothing but a weighted summation of the original algebraic constraint g and it time derivatives:

$$s = \left(\mu \frac{d}{dt} + 1\right)^{r-1} g = \sum_{j=1}^{m} b_j \left(\mu \frac{d}{dt} + 1\right)^{r-1} y_j, \tag{5.3.2}$$

where r is the index of the differential algebraic system. Therefore, the other way of evaluating the sliding function s simply depends on its definition, assuming all the algebraically constrained subsystems supply their output variables and their time derivatives to the $(r - 1)^{th}$ order. If the relative orders of the subsystems are different, the index number of the coupled system is taken to be one plus the smallest relative order of the subsystems.

Requiring the time derivatives of the system outputs from the subsystem simulators does not disclose more information about the subsystem simulators than the outputs themselves. After all, the time derivatives of the outputs can always be numerically calculated based on the outputs.

Another term that is required in the DTSM methods is the Jacobian Jz. In the hybrid implementation, its expression is obtained by symbolic differentiation of the sliding function with respect to the constrained input. The goal is to determine whether this Jacobian can be easily evaluated using pure numerical methods. Knowledge of s as a function of z is needed to numerically evaluate the Jacobian Jz. It seems that the entire expression of the sliding function is needed. However, further study of the sliding function definition, Eq. (5.3.2), and the definition of the DAE index reveals that only $g^{(r-1)}$ is a function of the constrained input z. Other components of the sliding function are only functions of the states $\mathbf{x}$ and time t. Therefore, in order to evaluate Jacobian Jz numerically, the subsystem simulators only need to supply $y^{(r-1)}$ as a function of the constrained input z.

Ideally, subsystem simulators should only provide predefined outputs for any given input. The Jacobian Jz can be obtained from an ideal subsystem simulator if different constrained inputs are supplied. Therefore, the subsystem simulators should be programmed to provide $y^{(r-1)}$ as functions of z at each time step. This will result in some additional computation and transmission load, but no additional information about the subsystem simulators need to be disclosed.

The α term requires much computation to construct symbolically and evaluate numerically since it consists of Jacobian $\mathbf{Jx}$. However, examination of its definition:

$$\alpha_k = \frac{\partial s}{\partial \mathbf{x}}(x_k, t_k, z_k)\cdot(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\partial s}{\partial t}(x_k, t_k, z_k)\cdot \Delta t\,, \tag{5.3.3}$$

reveals that α_k is closely related to $s(\mathbf{x}_{k+1}, t_{k+1}, z_k)$ and $s(\mathbf{x}_k, t_k, z_k)$:

$$s(x_{k+1}, t_{k+1}, z_k) - s(x_k, t_k, z_k) = \frac{\partial s}{\partial \mathbf{x}}(x_k, t_k, z_k)\cdot(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\partial s}{\partial t}(x_k, t_k, z_k)\cdot \Delta t + h.o.t. \tag{5.3.4}$$

The sliding function evaluated at $(\mathbf{x}_{k+1}, t_{k+1}, z_k)$ is not physically obtainable, but it can be calculated in a numerical simulation. Using the sliding function at $(\mathbf{x}_{k+1}, t_{k+1}, z_k)$ eliminates the need to symbolically manipulate and numerically evaluate equations to obtain **Jx**. Because only the highest time derivatives $y^{(r-1)}$ are functions of z, $(\mathbf{x}_{k+1}, t_{k+1}, z_k)$ can be calculated using subsystem outputs $y_{k+1}^{(0)}, y_{k+1}^{(1)}, \cdots, y_{k+1}^{(r-2)}$ and function $y_{k+1}^{(r-1)}(z)$.

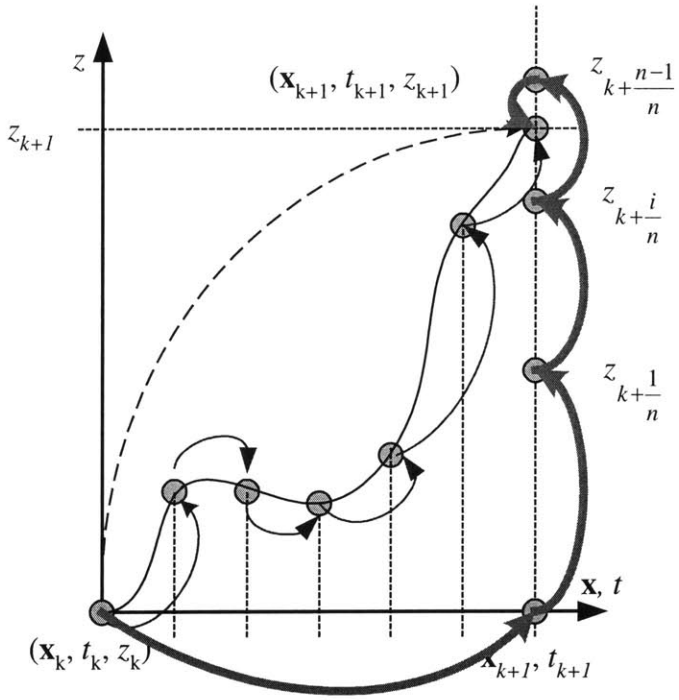


Figure 5.4.1 Different ways of enforcing the sliding function.

5.4 Decoupled Multi-Rate DTSM

5.4.1 Integration path

Figure 5.4.1 compares the newly proposed multi-rate DTSM algorithm with previously discussed methods. The integration path of the new algorithm is drawn in thick lines. Instead of sampling the system in physical time, the new algorithm uses variables evaluated at different times taking advantage of the numerical simulation. This decouples the effect of the constrained input z from the other variables. This new scheme is a type of multi-rate algorithm since the constrained input z is updated n times more frequently than the state variables $\mathbf{x}$.

The original single-rate DTSM method generates the constrained input z by minimizing s_{k+1} using only one step. The previous multi-rate DTSM method equally divides Δx_k into n equal, small steps, and the n-th step provides z at t_{k+1}. By simply enforcing

$$\left|s\left(\mathbf{x}_{k+i+1/n}, t_{k+i+1/n}, z_{k+i+1/n}\right)\right| < \left|s\left(\mathbf{x}_{k+i/n}, t_{k+i/n}, z_{k+i/n}\right)\right|, \tag{5.4.1}$$

in the small time scale i, it is possible to guarantee

$$\left|s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+1}\right)\right| < \left|s\left(\mathbf{x}_k, t_k, z_k\right)\right| \tag{5.4.2}$$

in the large time scale k.

However, the decoupled approach is more complicated. Even if we succeed in making the magnitude of $s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+i+1/n}\right)$ smaller than the magnitude of $s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+i/n}\right)$, we do not necessarily have

$$\left|s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+1}\right)\right| < \left|s\left(\mathbf{x}_k, t_k, z_k\right)\right|, \tag{5.4.3}$$

since the sliding mode control in the fast time scale starts from $s\left(\mathbf{x}_{k+1}, t_{k+1}, z_k\right)$ and not $s\left(\mathbf{x}_k, t_k, z_k\right)$, as illustrated in Fig. 5.4.2. Therefore, defining

$$\beta = \left|s\left(\mathbf{x}_{k+1}, t_{k+1}, z_k\right)\right| - \left|s\left(\mathbf{x}_k, t_k, z_k\right)\right|, \tag{5.4.4}$$

and by requiring the solution to satisfy

$$\left|s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+1}\right)\right| < \left|s\left(\mathbf{x}_{k+1}, t_{k+1}, z_k\right)\right| - \beta, \tag{5.4.5}$$

inequality (5.4.3) can be satisfied.

Condition Eq. (5.4.5) will be satisfied if we push the s function at least β/n at each time step in the faster time scale. Therefore the minimum control magnitude has to satisfy

$$v_0 > \max\left|Jz^{-1}\right| \cdot \left(\frac{\max|\beta|}{n} + \max\left|R\left(v_0^2\right)\right| + \frac{\delta}{n}\right). \tag{5.4.6}$$

Defining the equivalent control as:

$$O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right) = -Jz\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)^{-1} \cdot s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right). \tag{5.4.7}$$

The control law is:

$$v_{k+\frac{i}{n}} = v_0 \,\mathrm{sat}\left(v_0^{-1} \cdot O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)\right). \tag{5.4.8}$$

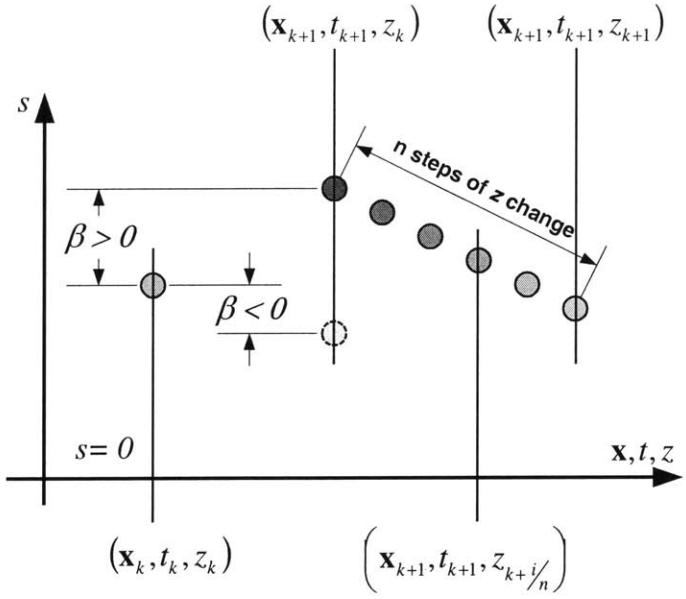


Figure 5.4.2 Schematic of the decoupled multi-rate DTSM.

5.4.2 Convergence in a Small Time Scale

The first step is to show that:

$$|s| < \max\left|R\left(v_0^2\right)\right|, \tag{5.4.9}$$

is an invariant set.

Starting from within the invariant set

$$\left|s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)\right| < \max\left|R\left(v_0^2\right)\right|. \tag{5.4.10}$$

We have the following

$$\begin{aligned}\left|Jz\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)^{-1} s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)\right| &< \max\left|Jz^{-1}\right| \cdot \max\left|R\left(v_0^2\right)\right| \\ &< \max\left|Jz^{-1}\right| \cdot \left(\frac{\max|\beta|}{n} + \max\left|R\left(v_0^2\right)\right|\right)\end{aligned}, \tag{5.4.11}$$

and this inequality is equivalent to

$$\left|O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)\right| < v_0. \tag{5.4.12}$$

Under this initial condition, the control input is

$$v_{k+\frac{i}{n}} = O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right) = -Jz\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right)^{-1} \cdot s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}}\right), \tag{5.4.13}$$

and the s function at the next small time step is

$$s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i+1}{n}}\right) = R\left(\Delta z_{k+\frac{i}{n}}\right). \tag{5.4.14}$$

Therefore we have

$$\left| s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i+1}{n}} \right) \right| < \max \left| R\left(v_0^2\right) \right|. \tag{5.4.15}$$

This proves that Eq. (5.4.9) is an invariant set.

If we start outside of, but close to, the invariant set satisfying

$$\max \left| R\left(v_0^2\right) \right| < |s| < \max \left| Jz^{-1} \right| \cdot \left(\frac{\max|\beta|}{n} + \max \left| R\left(v_0^2\right) \right| + \frac{\delta}{n} \right), \tag{5.4.16}$$

the control input is still Eq. (5.4.13). Therefore the trajectory will move into the invariant set after one step. If the initial condition is far from the invariant set satisfying

$$|s| \geq \max \left| Jsz^{-1} \right| \cdot \left(\frac{\max|\beta|}{n} + \max \left| R\left(v_0^2\right) \right| + \frac{\delta}{n} \right), \tag{5.4.17}$$

this initial condition can be written as

$$\left| O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}} \right) \right| \geq v_0. \tag{5.4.18}$$

Under this condition, the control input is:

$$v_{k+\frac{i}{n}} = v_0 \cdot Jz\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}} \right)^{-1} \cdot s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}} \right) \cdot \left| O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}} \right)^{-1} \right|. \tag{5.4.19}$$

The s function at step $\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+i/n} \right)$ is:

$$\begin{aligned} s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i+1}{n}} \right) = s\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}} \right) \cdot \left(1 - v_0 \cdot \left| O\left(\mathbf{x}_{k+1}, t_{k+1}, z_{k+\frac{i}{n}} \right) \right|^{-1} \right). \\ + R\left(\Delta z^2_{k+\frac{i}{n}} \right) \end{aligned} \tag{5.4.20}$$

Taking absolute values and considering the initial condition, we have

$$\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i+1}{n}}\right)\right|$$

$$\leq\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|\cdot\left(1-v_0\cdot\left|O\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|^{-1}\right)+\left|R\left(\Delta z^2_{k+\frac{i}{n}}\right)\right| \qquad .(5.4.21)$$

$$=\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|-v_0\cdot s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\cdot\left|O\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|^{-1}+\left|R\left(\Delta z^2_{k+\frac{i}{n}}\right)\right|$$

Substituting in the equivalent control again, we arrive at:

$$\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i+1}{n}}\right)\right|\leq\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|-v_0\cdot\left|Jsz\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|^{-1}+\left|R\left(\Delta z^2_{k+\frac{i}{n}}\right)\right| \qquad .(5.4.22)$$

Since the minimum control magnitude satisfy condition Eq. (5.4.6), the following is guaranteed:

$$\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i+1}{n}}\right)\right|-\left|s\left(\mathbf{x}_{k+1},t_{k+1},z_{k+\frac{i}{n}}\right)\right|<-\frac{\beta}{n}-\frac{\delta}{n}<0. \qquad (5.4.23)$$

Equation (5.4.23) shows that the invariant set Eq. (5.4.9) is attractive and the convergence rate is larger than $(\beta+\delta)/n$ per step in the fast time scale.

5.4.3 Convergence in a Large Time Scale

The analysis in the fast time scale shows that the system exhibits a convergent discrete-time sliding mode starting from $(\mathbf{x}_{k+1},t_{k+1},z_k)$. If the sliding function evaluated at $(\mathbf{x}_{k+1},t_{k+1},z_k)$ point is within the boundary layer of the sliding manifold $s = 0$, the system trajectory will stay within the boundary layer at $(\mathbf{x}_{k+1},t_{k+1},z_{k+1})$. If the system at $(\mathbf{x}_{k+1},t_{k+1},z_k)$ is outside of the boundary layer, the system trajectory is either pushed into the boundary layer before or at $(\mathbf{x}_{k+1},t_{k+1},z_{k+1})$ point, or the system trajectory moves a distance of $(\beta+\delta)/n$ closer to the sliding manifold.

When the system is analyzed in the large time scale, i.e., time scale k, it is also seem to exhibit a convergent sliding manifold. If s_k starts within a boundary layer of the origin, $s(\mathbf{x}_{k+1},t_{k+1},z_k)$ may be closer to the origin than s_k. So, the sliding mode remains within the

boundary layer in the small time scale i and s_{k+1} will be within the boundary layer. If $s(\mathbf{x}_{k+1}, t_{k+1}, z_k)$ is farther away from the origin than s_k, the sliding control in the small time scale is guaranteed to compensate for the difference β, and s_{k+1} will be within the boundary layer. This shows the boundary layer, Eq. (5.4.9), is also an invariant set in the large time scale. If s_k starts outside of the invariant set, the sliding control in the small time scale either brings the system trajectory into the invariant set or at least δ closer to the invariant set. Hence, the invariant set in the large time scale is attractive as well. The speed of convergence to the invariant set is finite, and at least equal to δ. Finally, the system trajectory will reach the invariant set within a finite number of steps in the slow time scale.

5.4.4 Numerical Examples

The two examples will use the following coupled system including two dynamic subsystems:

$$\begin{aligned} \dot{x}_{A,1} &= x_{A,2} \\ \dot{x}_{A,2} &= -x_{A,1} - x_{A,2} - x_{A,3} + \text{atan}(z) \\ \dot{x}_{A,3} &= x_{A,1} - x_{A,2} \end{aligned} \tag{5.4.24}$$

and

$$\begin{aligned} \dot{x}_{B,1} &= x_{B,2} \\ \dot{x}_{B,2} &= -x_{B,1} - 2x_{B,2} - x_{B,3} \\ \dot{x}_{B,3} &= x_{B,1} - x_{B,2} - z \end{aligned} . \tag{5.4.25}$$

This system is subject to the algebraic constraint

$$y_{A,1} - y_{B,1} = 0, \tag{5.4.26}$$

where the two outputs are $y_{A,1} = x_{A,1}$ and $y_{B,1} = -x_{B,1}$, respectively. This is an index 3 DAE. The subsystems are simulated using Euler's forward method with $\Delta t = 0.1$ and the algebraic constraint is enforced by the Multi-Rate DTSM given by Eqs. (5.4.7) and (5.4.8).

Large v_0 Is Unstable

The function atan(z) is chosen to demonstrate the nonlinear effect on the choice of v_0. $x_{A,1}$, $x_{A,2}$, and $x_{A,3}$ start from -1 while $x_{B,1}$, $x_{B,2}$, and $x_{B,3}$ start from $+1$. The initial value of z is 10. This leads to an inconsistent initial condition $s \neq 0$. Parameter μ equals 0.5.

The remnant $R(v_0^2)$ is nonzero when the sliding variable s is nonlinear with respect to the boundary variable z in the multi-rate DTSM. When v_0 is 20, the nonlinearity of $R(v_0^2)$ makes z diverge as shown in Fig. 5.4.3 and s cannot be stabilized as shown in Fig. 5.4.4. When v_0 is reduced to 1, the nonlinearity $R(v_0^2)$ is suppressed and stable results are obtained.

Small $n \cdot v_0$ Is Unstable

All conditions remain the same from the above example except that different n and v_0 will be used. This is to demonstrate that the control action has to be large enough to compensate for the change in s caused by the change of **x** and t.

Both **x** and t are the exogenous inputs to the s dynamics. In Multi-Rate DTSM, $n \cdot v_0$ needs to be large enough to counteract the effects of **x** and t in order to stabilize the sliding variable s. Figure 5.4.5 shows that $n \cdot v_0 = 10$ is able to stabilize s while $n \cdot v_0 = 3$ is not. Figure 5.4.6 shows that the boundary variable changes too slowly when $n \cdot v_0$ is small. The sliding variable becomes negative around $t = 1$ second, but boundary variable is still in the far negative region trying to correct the positive s in the past. This leads to unstable BCC eventually.

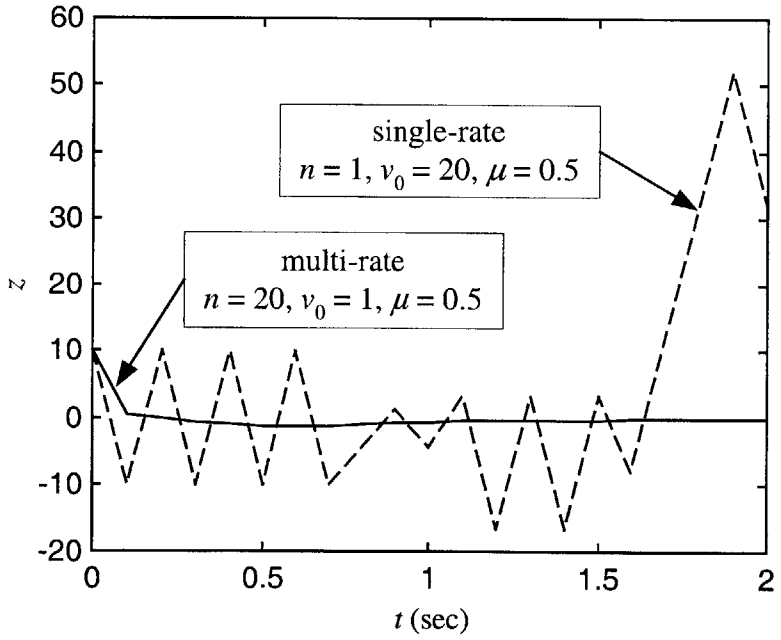


Figure 5.4.3. Boundary variable for different v_0.

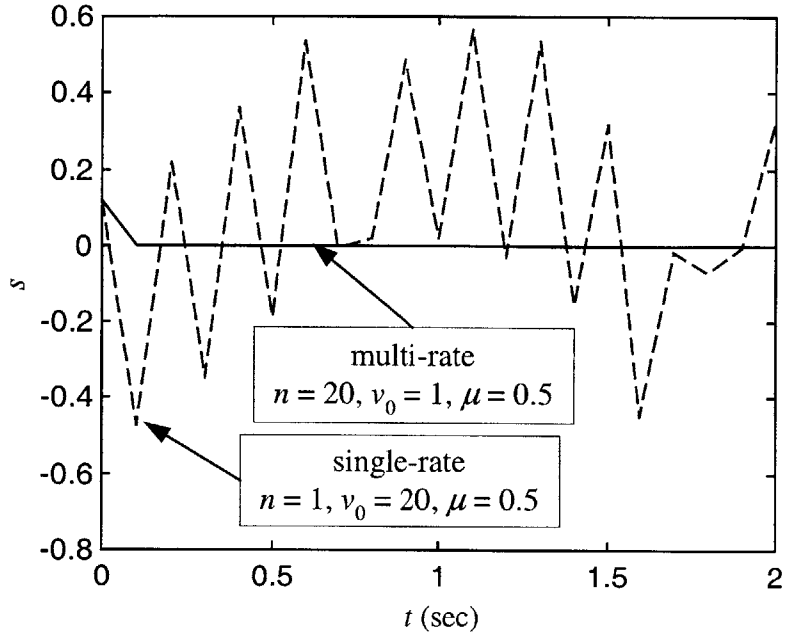


Figure 5.4.4. Sliding variable for different v_0.

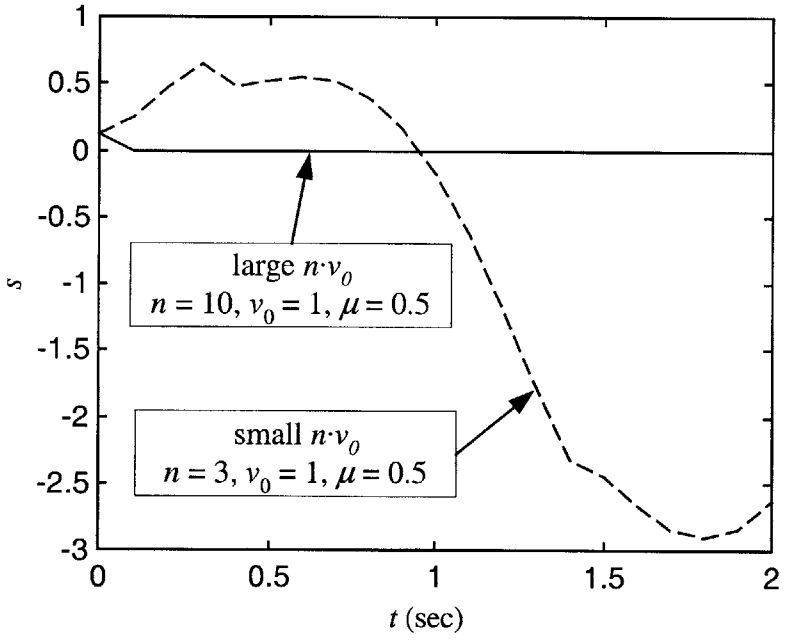


Figure 5.4.5. Sliding variable for different $n \cdot v_0$.

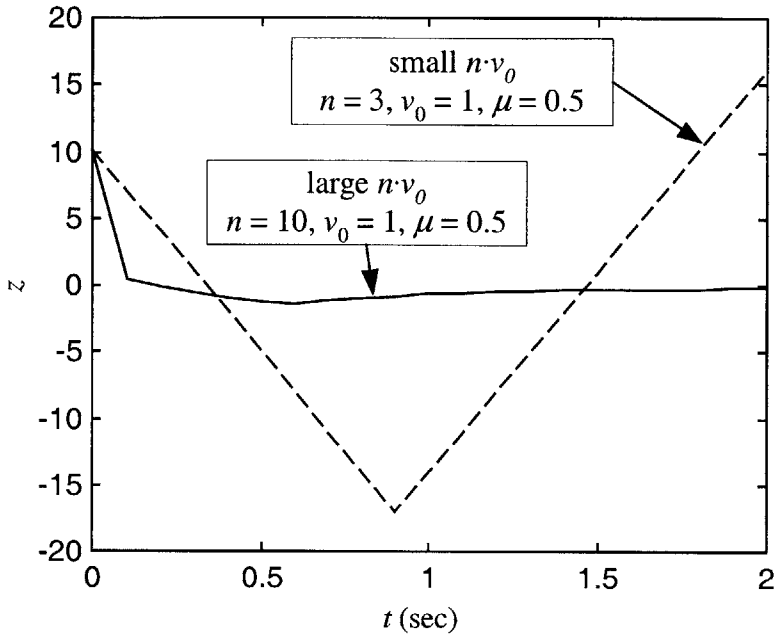


Figure 5.4.6. Boundary variable for different $n \cdot v_0$.

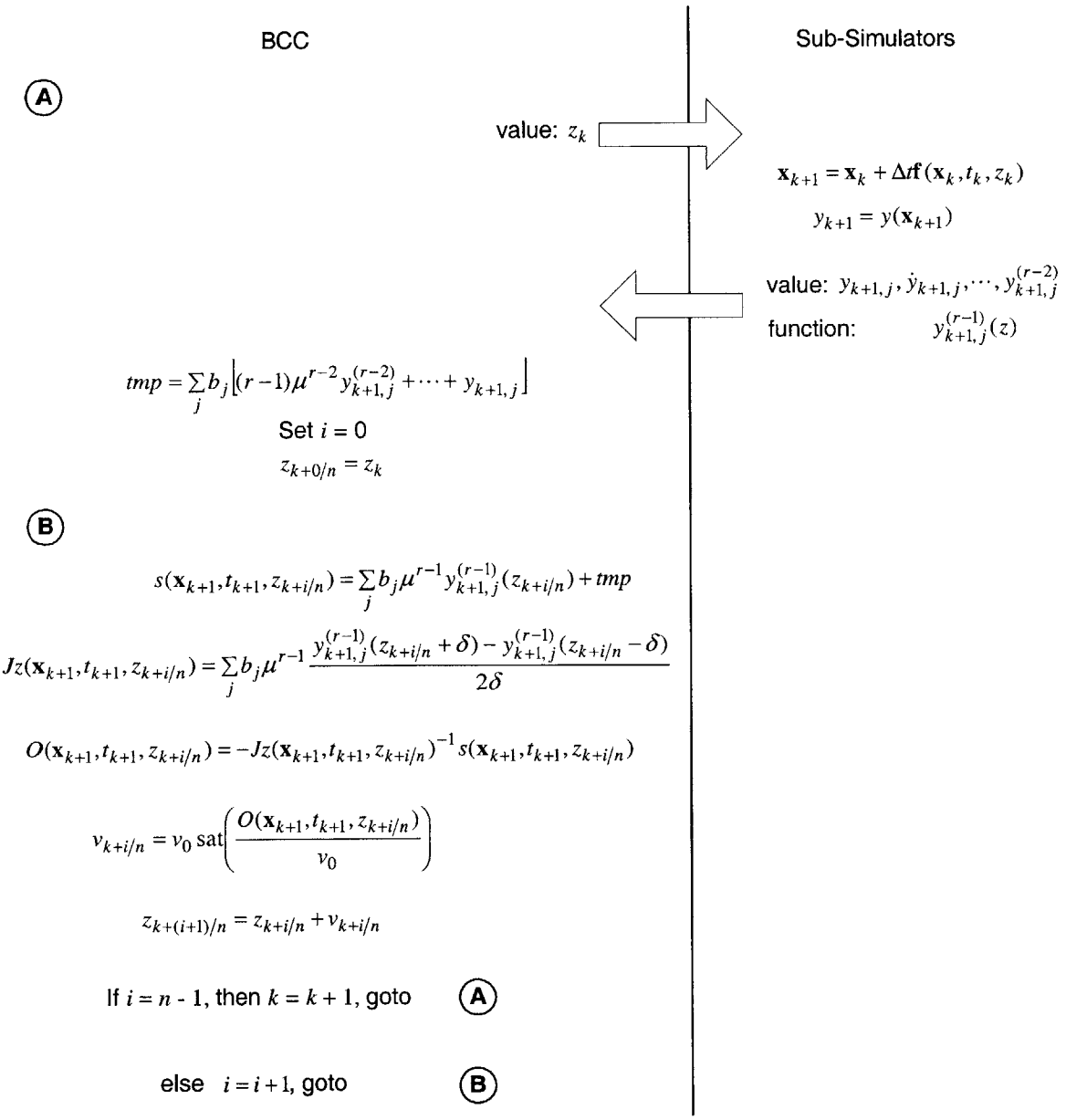
5.5 Separation of BCC and Dynamic Subsystems

Pseudo codes are used to illustrate the computations taking place in a BCC and its neighboring subsystem modules. Table 5.5.1 recapitulates the computation done in dynamic simulators and the BCC, and the communication between the BCC and its neighboring simulators. The initial values of the state variables and the boundary variables for subsystem modules and BCCs are set using the Co-Simulation software environment. The computation starts at time step k from point A. The BCC supplies the boundary variable z_k to its neighboring simulators. Using z_k and the stored $\mathbf{x}_k$'s, the simulators integrate one step. The simulators calculate outputs and their derivatives y_{k+1}, $\dot{y}_{k+1}$ ... $y_{k+1}^{(r-2)}$, and then send these values together with the functions $y^{(r-1)}(z)$ to the BCC.

Having received the information, the BCC calculates a temporary variable, which is not affected by z, before going into the fast time scale computation. The temporary variable is a part of the sliding variable s. This is to avoid redundant computation. The coefficient b defines the sign of connected subsystems. Starting from point B, the boundary variable z is updated n times. The procedure is a straightforward application of the control law Eqs. (5.4.7) and (5.4.8). After the BCC completes n steps of computation for z, the computational thread goes back to point A

and the BCC sends the newly updated z to connected simulators. This completes one time step of computation.

Table 5.5.1. Communication among subsystem simulators and BCC.



VI Software Development

The software developed in this project realizes the idea of the object-oriented simulation. The object-oriented simulation is a level higher than object-oriented modeling [Cellier and Elmqvist, 1993; Haier and Wanner, 1991; Andersson, 1990; Mattsson et al, 1998]. With object-oriented modeling, subsystem models are reusable components that possess some object-oriented properties. One can use subsystem models to form a large system model by simply connecting the subsystem models together. No modification of subsystem models is required. When one wants to solve the coupled system numerically, one has to compile the coupled model into a simulator of the coupled system. This compilation and solution process is not often a simple task. Also, information hiding is not applicable in object-oriented modeling since the function of the model is to provide a description of the subsystem.

In this project, we apply object-oriented paradigm to the level of simulation. Each subsystem simulator is an object. Users can run these subsystem simulator objects with minimum knowledge of the subsystems. Subsystem simulator objects can be connected to simulate a large coupled system without any modification. More importantly, these simulator objects do possess the information hiding property. Users can obtain the simulation results but cannot access the models of the subsystems.

The most difficult part of the object-oriented simulation is to deal with causal conflicts. We have discussed the mathematical solution in the previous chapters. In this chapter, the decoupled multi-rate Discrete-Time Sliding Mode method will be applied to develop the Co-Simulation software environment in order to treat the algebraic constraints caused by causal conflicts.

Most of the subsystem simulators including the Boundary Condition Coordinator are written in Java [Sun, 2001].

6.1 Software for Different Functional Requirements

6.1.1 Co-Simulation Running on One Computer

The basic functional requirement for the Co-Simulation software environment is that every subsystem is an independent simulator. Once the inputs to a subsystem simulator are supplied, the subsystem simulator will produce outputs. Ideally, the users of the subsystem simulator

should not be required for anything other than providing the inputs, and the users should not be able to find out the detailed implementations of the subsystem simulators. For subsystem simulators that are connected in causal conflicts, we have to supply slightly more information.

In order to connect subsystem simulators to form a large system, we need a piece of software, in which we can define the interconnections among subsystems. In this stage, all subsystem simulators will be located on one single computer as shown in Fig. 6.1.1.

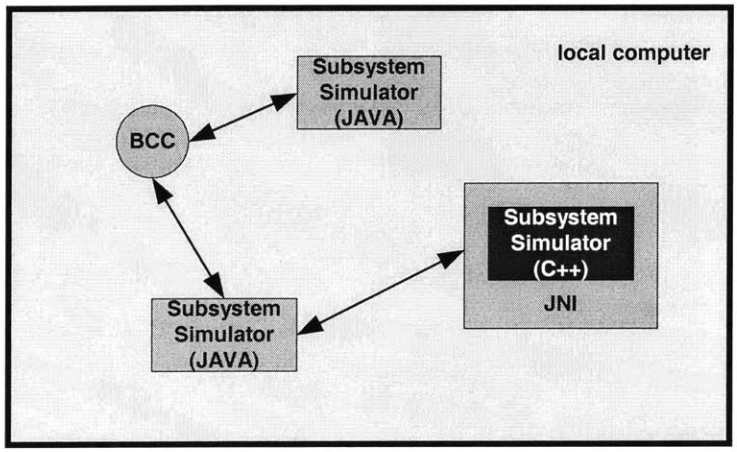


Figure 6.1.1 Structure of Co-Simulation.

6.1.2 Co-Simulation Running over The Internet

The goal of Co-Simulation is to make sharing of simulators easier. The current version of the Co-Simulation software requires that all subsystem simulators run on one computer. With the fast growth of information technology, future versions of Co-Simulation should not be confined to a single computer.

However, Co-Simulation is different compared to other Internet-based simulation software, such as DOME [Wallace et al, 2000; Senin et al, 2000]. The purpose of the Co-Simulation discussed here is to simultaneously run many dynamic subsystem simulators in order to form a large dynamic system simulator. Since the coupled system is a dynamic system, its components have to exchange information at every time step. When the time step is small, the information transmission speed is essential to the speed of the Co-Simulation. To the author's knowledge, no existing Internet-based simulation software deals with dynamic simulation; they only have to send and receive information at the start and end of the simulation. Therefore, it is

necessary to verify that Internet-based dynamic simulation is feasible for now and the near future. The following calculation shows that the raw speed of 1 M bits per second is equivalent to 16384 double numbers per second.

$$1 \text{ Mbps} \div 8 \text{ (bits/byte)} = 131072 \text{ Bps (byte per second)}$$
$$= 131072 \text{ (Bps)} \div 8 \text{ (bytes/double)}$$
$$= 16384 \text{ double numbers per second}$$

The result is promising since a moderate T1 (1.544 Mbps) connection can ideally transmit 25297 double numbers per second. The real transmision rate will be lower than this estimate since no overhead has been considered.

Although implementation of an Internet-based Co-Simulation is not a part of this thesis work, we would like to make our software structure compatible with an Internet-based implementation. Figure 6.1.2 shows the schematics of an Internet-based Co-Simulation. The basic structure resembles the local version shown in Fig. 6.1.1. The difference is that some subsystem simulators run on different computers, and some components of the coupled simulation are connected through the Internet.

6.1.3 Single-Thread Simulation

We have to consider yet another scenario. For instance, a large engineering company has many divisions. Each division manufactures one or more components, and all the components will be assembled to form the final product. Since all the components are made within that company, the subsystem simulator makers do not need to hide the details about their subsystem simulators. On the other hand, the company wants the computation of the coupled simulation to be as fast as possible.

Simultaneously running multiple simulators is not the best choice for this scenario. Because every subsystem simulator is independent, each simulator sends and receives information at each time step. Comparing to lumping all the differential equations into a large set of equations and solving them at once, this distributed approach spends some overhead time in sending and receiving information. Therefore, it makes sense to consolidate all the equations and solve them together if the protection of the intelectual property is not a concern. This requires that the internal structure of the subsystem simulator is carefully designed such that the model equations, the integration algorithm, and other necessary functions are clearly separated. This will make it easy for an optional piece of software that extracts all the equations out of the

subsystem simulators and compiles them into a single-thread simulation. This functional requirement of the object-oriented simulation can be viewed as the backward compatibility to object-oriented modeling.

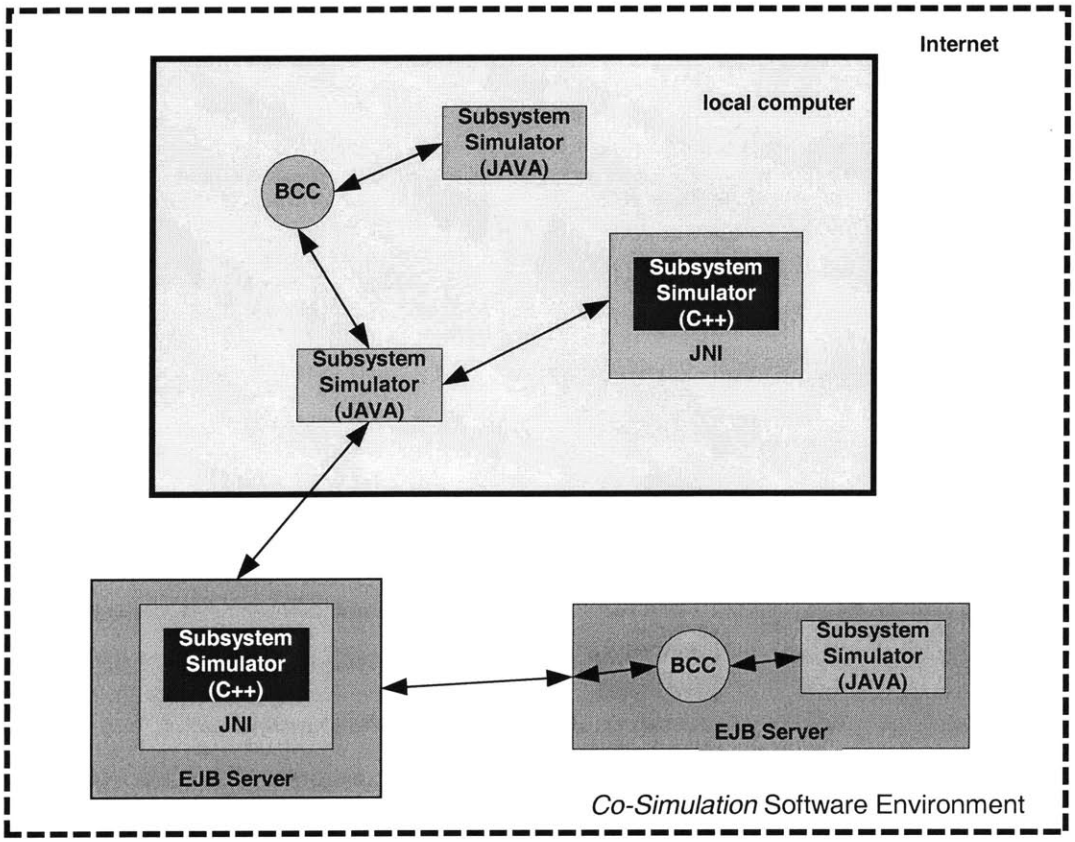


Figure 6.1.2 Structure of the Internet-based Co-Simulation

6.2 Software Architecture

The previous section discussed the functional requirements of the Co-Simulation software environment. We will design the architecture of the Co-Simulation software environment to match the functional requirements. The most important functional requirement is that every subsystem simulator must be independent. Every subsystem simulator is an independent thread of process. Therefore a distributed architecture is used for the Co-Simulation software environment. Communication among subsystem simulators is done through messages. The subsystem simulators are programmed as object-oriented objects and their states can only be affected by sending messages to them. The Co-Simulation software performs distributed

computation without any central coordination. At the current stage, independent subsystems run in the domain of a local computer. It will be easy to expand the domain to the entire Internet in the future.

Since we are solving differential equations in order to produce simulation results at every time step, synchronization must be maintained throughout the simulation. It is possible to apply certain asynchronous numerical integration algorithms to solve differential equations. However, the purpose of asynchronous differential equation solving is to utilize parallel computers. Therefore, our intentions to adopt distributed computation are rather to achieve modularity, information hiding, and user collaboration. We are not using distributed computation as a means for better computational speed in this project. Therefore, asynchronous differential equation solving is not considered and it is out of the scope of this thesis work.

With independent subsystem simulators running in a distributed manner, we need a software environment in which we can search for available subsystem simulators, define interconnections among chosen simulators and obtain results for the coupled simulation. The system integrator is the piece of software designed for this purpose. The architecture of the Co-Simulation software environment is similar to a server/client structure. The system integrator, which can be loosely compared to a web browser, provides an interface between subsystem simulators and the system integration engineer. The subsystem simulators are like web servers, providing the information that the user requests from the system integrator. The communication among the subsystem simulators is transparent to the user when the Co-Simulation is running. The system integrator does not coordinate the execution of the subsystem simulators.

Since subsystem simulators need to solve differential or algebraic equations, send/receive information, and synchronize with other subsystem simulators—the internal structure of a subsystem simulator is complex. We do not expect subsystem simulator designers to know the detailed software implementation and multi-thread programming. We therefore predefine some base classes that include all these functionalities. The subsystem simulator designer will only need to provide the ODE models of the subsystems and some additional information to prepare for a possible causal conflict situation.

There are two software deliverables from this project. One is the object-oriented class design for the subsystem simulators. Another deliverable is the system integrator, which is a thin

application with a friendly graphic user interface. The details of the two deliverables are provided in the following section.

6.3 Object-Oriented Design of Subsystem Simulators

6.3.1 Object-Oriented Design

A computer program is a software copy of a real world object. We have to study the real world object and use the most fitted software structure and data structure to represent the object in the virtual realm. If the structure of the real world object and the software copy does not match, usually the resulting software code is not effficient and error prone, and the software program will be difficult to code and read.

For many software projects, the object-oriented paradigm fits real world objects very well. Figure 6.3.1 shows the structure of an object-oriented software object [Sun, 2000]. The object consists of two layers. The core is the state of the object and the shell is the behavior of the object. The state of the object describe its properties, such as the speed of a car, temperature of a room, etc. The behavior of the object describe how it interacts with the rest of the world. An object can apply influence to other objects, such as cold air cooling off people in a room. An object can also be influenced by other objects, such as the speed of a car changing because of the wind. The state of the object usually should not be changed by other objects directly.

The object-oriented paradigm fits with the subsystem simulators quit well. The core of a subsystem simulator consists of state variables and other variables. It is not physically realistic to change a state variable of a subsystem directly from outside. For example, we cannot change the speed of a moving object directly. Instead, we have to apply a force in order to change its speed. The shell layer of a subsystem simulator consists of the input/output functions of the subsystem model. The state variables of a subsystem simulator will be influenced by inputs that the subsystem receives.

The object-oriented paradigm also features encapsulation that can be used to achieve information hiding, one of the most important functional requirements of the subsystem simulators. Subsystem simulators maker can keep their system models secret. The detailed implementation of a subsystem simulator is also proprietary to the subsystem simulator maker. The users of the subsystem simulator can only supply inputs and obtain outputs from the simulator. The builder of subsystem simulators should be able to make some states or

parameters open to the users and keep some data inaccessible from the users. The users may select some of the available data from subsystem simulators for post-processing.

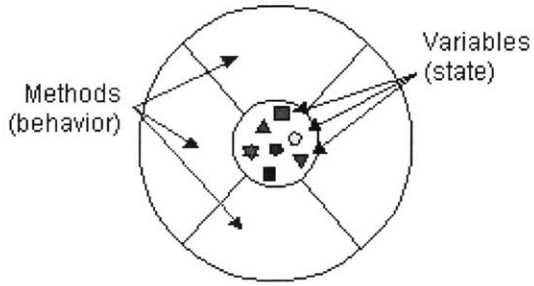


Figure 6.3.1 The structure of an object.

Inheritance is another feature of the object-oriented program design and it plays an important role in subsystem simulator development. The basic element of Co-Simulation is the subsystem simulator. In a coupled simulation, there are various types of subsystem simulators. The most common subsystem simulator is the dynamic subsystem simulator. We have to use the Boundary Condition Coordinator as a subsystem simulator whenever there exists causal conflict. Some subsystems may simply be algebraic relations. We even use some special subsystems that only provide the boundary condition. These subsystem simulators behave differently. However, they share some common features. With inheritance, we can simply define a parent class for the general subsystem simulators, and all different special subsystems can use that feature without implementing it by themselves. Inheritance can be passed on to several layers. For example, all dynamic subsystem simulators have to solve differential equations. The solution methods for differential equations can be defined in the dynamic subsystem class. All specialized dynamic subsystem simulators that share the solution inherit the solution method from the dynamic subsystem class. At the same time, they also inherit properties and behaviors common to all subsystem simulators from the subsystem base class.

6.3.2 Synchronization

The basic function of all types of subsystem simulators is to get inputs and generate outputs. Therefore input/output behavior is common to all subsystem simulators. Since most subsystem simulators require inputs before they can calculate the outputs, the timing of the operation of an individual subsystem simulator must be regulated.

The Co-Simulation environment adopts a software architecture of distributed multi-processing without central coordination. The important task of synchronizing the dynamic simulation is carried out by the individual subsystem simulators. Every subsystem simulator takes inputs from connecting subsystems simulators at every time step. However, every subsystem simulator is an independent process and knows nothing about other subsystem simulators. If one simulator tries to get information from the other simulators without any regulation, the subsystem simulator may get old information from the last time step or future information for the next time step. Neither case causes beneficial results. Therefore, subsystem simulators must keep track of their inputs and outputs.

The real issue here is that two independent processes have to access a single data object. The two processes must operate on the data object one by one in an interlaced sequence. Simultaneous access is not allowed. The data object must keep track of its own status. If the data are ready for the first process, the first process will be allowed to operate on the data. If the data are ready for the first process but the second process wants to access the data, the data object will tell the second process to wait until the first process completes its operation.

If a subsystem simulator communicates with only one other subsystem simulator, the possible conflict is easy to solve. The subsystem simulator that provides output can utilize a flag for its output data. If the output has already been read by the other subsystem simulator and the data acquisition changes the state of the flag, the output subsystem can proceed to replace the output data object with an updated one and set the flag again. However, subsystem simulators usually communicate with several other subsystem simulators. If one subsystem simulator needs to obtain inputs from several other subsystem simulators and receives data from output providers one by one in a loop, and if the first data provider is not ready to provide the data, the rest of the simulators have to wait even if they are ready. When Co-Simulation is running over the Internet, data transmission may account for a considerable amount of time. The waiting for data and sequential data transmission may contribute to lower overall speed of the Co-Simulation.

The solution is to launch an independent thread for each *send* function of a subsystem simulator, so data will be transmitted concurrently. Figure 6.3.2 shows an illustration of data transmission among three subsystem simulators. Every subsystem has an output data storage object for each output port of that subsystem. The output data storage objects are part of every subsystem simulator. The *send* function of the subsystem simulator is a multi-thread function.

There is one thread for each output data storage object, and each thread calls the *put* function of its output data storage object. The *put* function will wait if the existing data of the output data storage object have not been read. The *put* function will set the output data storage object with the new data and return if the existing data have been read. After all the *put* functions of output data storage objects return, the *send* function of the subsystem simulator returns and the subsystem simulator's main process goes on. The *receive* function of subsystem simulators functions in a similar manner.

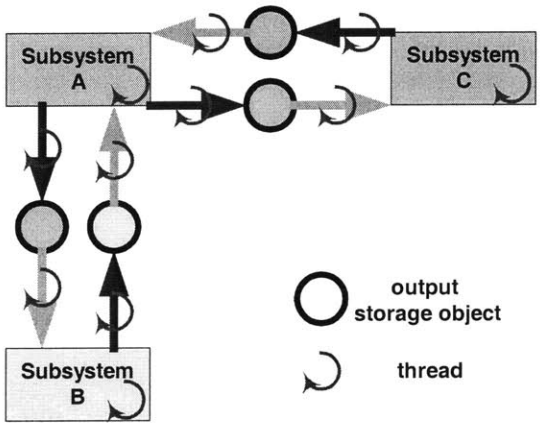


Figure 6.3.2 Mulit-thread input/output of subsystems

6.3.3 Computational Logic

Depending on different subsystem simulator categories, subsystem simulators generate outputs in different ways. A general subsystem is modeled as follows:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{y}_{k+1} &= \mathbf{y}(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) \end{aligned}, \tag{6.3.1}$$

where **x**, **u**, and **y** are states, inputs and outputs, respectively. If the subsystem is a causal dynamic system, the feedforward term $\mathbf{u}_{k+1}$ is zero. Such systems only require input information at time t_k to produce result at t_{k+1}. Causal dynamic systems have state variables, which require memory and initial conditions. For pure algebraic subsystems, there is no state variable so they do not need memory or initial conditions. They do not predict the future state and they produce output at time t_{k+1} when the input at time t_{k+1} are provided. Boundary Condition Coordinators using the decoupled multi-rate DTSM method discussed in Chapter 5 are non-causal dynamic

systems. They have memory but also require information at t_{k+1} to produce results at t_{k+1}. Table 6.3.1 summaries all possible subsystem characteristics.

Table 6.3.1 Subsystem characteristics.

Subsystems	Memory (Initial Condition)	Require input at t_{k+1} to generate output at t_{k+1} ?
Causal Dynamic	Yes	No
Non-Causal Dynamic	Yes	Yes
Algebraic	No	Yes
Source	No	No

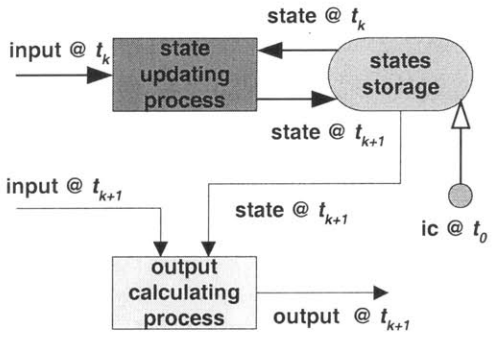


Figure 6.3.3 Common computational logic

The flow chart for the general subsystem is shown in Fig. 6.3.3. The flow chart is consistent with Eq. (6.3.1). All of the different subsystem simulators are based on this common computational logic.

In every subsystem simulator, there is a main loop that consists of the mathematical computation, information receiving and information sending. However, for different subsystem simulators with different characteristics, the structure of the main loop is different. Figure 6.3.4 shows the main loop for causal dynamic subsystem simulators. The loop starts from the small circle, which supplies the initial condition. When the dynamic subsystem simulator begins normal operation, the simulator goes through the send, receive, state updating process, and

output calculating process blocks in the loop. The send block and the receive block use the multi-thread *send* and *receive* function discussed in previous section, respectively.

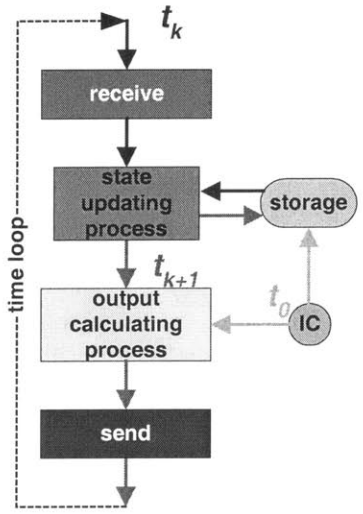


Figure 6.3.4 Subsystem simulator logic of causal dynamic systems.

The logic of the Boundary Condition Coordinator is shown in the flow chart of Fig. 6.3.5. The BCC requires an initial value for the constrained input z, which can be viewed as the state variable of the BCC. The BCC is also a non-causal dynamic system so it requires input at time t_k and t_{k+1} to produce the result z_{k+1}. Therefore, the BCC takes input at the start of the simulation (t_0) and stores it in its memory. With the initial condition of z, the BCC can send the output to other subsystem simulators at t_0. After the initialization period, the BCC first takes inputs $\mathbf{u}_{k+1}$, and then calculates output z_{k+1} with the stored z_k and $\mathbf{u}_k$ from the last step.

The simulator logic of the pure algebraic systems and source systems are very simple, and their flow charts are given in Fig. 6.3.6 and Fig. 6.3.7, respectively.

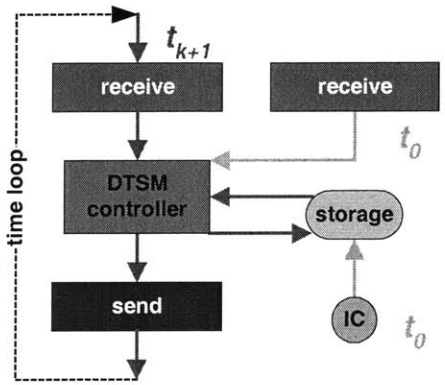


Figure 6.3.5 Subsystem simulator logic of BCCs.

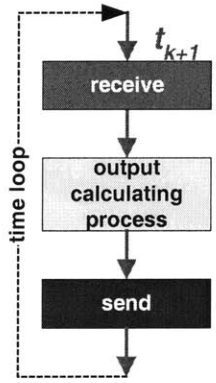


Figure 6.3.6 Subsystem simulator logic of pure algebraic systems.

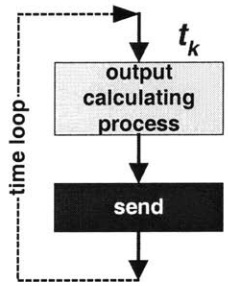


Figure 6.3.7 Subsystem simulator logic of source systems.

6.3.4 Class Tree

Now that important functions and subsystem simulator logic have been explained, we are ready to show all the classes that we have developed in an inheritance tree structure shown in Fig. 6.3.8. The base class for subsystem simulator is `Subsystem`, which implements a runnable interface

to become an independent thread of process. Behaviors and properties common to all different subsystem simulators are defined in class `Subsystem`. Among many common properties defined in this class are arrays of the state variables, input variable, output variable, and output storage objects. The common behaviors include the multi-thread *send* and *receive* methods discussed earlier in this chapter. A *setConnection* method is defined to connect one of the subsystem simulator's output port to another subsystem's input port. Abstract methods *setDescriptions* and *setPublishedData* are defined to make a derived class always select some data to publish and to provide descriptions of the published data. The `subsystem` class also provides two default methods for the two aforementioned abstract methods so that the derived classes have an option to simply use the default information.

We derive four classes from the `Subsystem` classes. They are `DynamicSub`, `AlgebraicSub`, `BCC` and `Source`. These four classes are all abstract classes, which means that they can only be used as class templates for deriving subclasses. Abstract classes cannot be instantiated into objects because they are abstract generalizations without detail implementation.

The `DynamicSub` class is a widely used class since there are many dynamic subsystems in a coupled system. The most important method of the `DynamicSub` class is the *run* method. The basic structure of the *run* method has been already shown in Fig. 6.3.4. The *send* and *receive* methods used in the *run* method are inherited from the base class `Subsystem`. The *nextStep* method, which utilizes a generic `IntMethod` class to perform numerical integration, updates the states in the *run* method. The subclasses of `DynamicSub` will override the `IntMethod` class using classes that perform different numerical integration algorithms, such as Runge-Kutta, Euler, etc. The `DynamicSub` class also defines many abstract methods, such as the differential equations *diff*, the highest order derivative as a function of the constrained input *ydd*, and the initial condition *ic*. Subsystem simulator builders will design their specific simulator classes for different dynamic subsystems based on the `DynamicSub` class. Different *diff*, *ydd* and *ic* methods set different subsystem simulators apart. The `DynamicSub` class overrides the *setConnection* method defined in the base class `Subsystem` since the dynamic subsystem simulator may connect to the Boundary Condition Coordinator, which requires not only output variables but also time derivatives of the output variables to run.

Being a subclass of `Subsystem`, the `BCC` class's most important method is also *run*. However, the detailed implementation of the `BCC`'s *run* method is much different than `DynamicSub`'s. The `BCC` class's *run* method adopts the decoupled multi-rate DTSM algorithm for solving the constrained input z. Method *jacobian*, *s_cal* and *dtsm_m2* calculates Jacobian Jz, sliding function s, and constrained input z, respectively. The `BCC` class is a very detailed class. Thus, its subclasses only need to define a few parameters, such as the minimum magnitude of the control input v_0 and the multi-rate parameter n.

The `AlgebraicSub` class is very simple. Its *run* method realizes the flow chart given in Fig. 6.3.6. It predefines an abstract method *output*, which can be defined by its subclasses to represent different algebraic relation between the input and the output. The `Source` class is very simple as well. Its *run* method realizes the flow chart given in Fig. 6.3.7. It predefines an abstract method *output*, which can be defined by its subclasses to supply different outputs as the functions of time.

Subsystem simulators derived from these four abstract classes are not part of the Co-Simulation software package. These classes will be defined by builders of different subsystem simulators. Subsystem builders can fully utilize all predefined class templates without writing a single line of code. They can also make their own abstract class to funciton a subsystem simulator template, such as the `HeatExchanger` class that is subclassed by `IndoorUnit` and `OutdoorUnit` class shown in Fig. 3.6.8.

Through all abstract classes shown in this inheritance tree structure, the basic functional requirement of the Co-Simulation software environment is achieved. Although the basic Co-Simulation software only does multi-process simulation on one computer, future extension of the Co-Simulation software is very easy due to the carefully designed object-oriented software structure. If we want to do Internet-based Co-Simulation, we only have to change the implementation of the *send* and *receive* methods in the `Subsystem` class. All subclasses derived from the `Subsystem` class require no modification from the subsystem simulator builders' point of view. If we want to solve the coupled system in a single native process, we can simply extract the models from the subsystem simulators and compile the subsystem models into a large simulator. The compiler will be easy to design since the subsystem simulator classes contain almost nothing beyond the system models.

6.4 User-friendly System Integrator Design

The other software component of the Co-Simulation software environment is the system integrator. It actually serves two different purposes in the architecture of the Co-Simulation software environment. On one hand, the system integration environment provides the subsystem-human interface so that we can connect subsystem simulators, run the coupled simulation and get the simulation results. On the other hand, the system integration environment implicitly serves as the server environment, on which subsystem simulators run. For future Internet-based Co-Simulation, we have to divide these two functionalities since the system integration environment and subsystem simulator servers are not usually located at the same place.

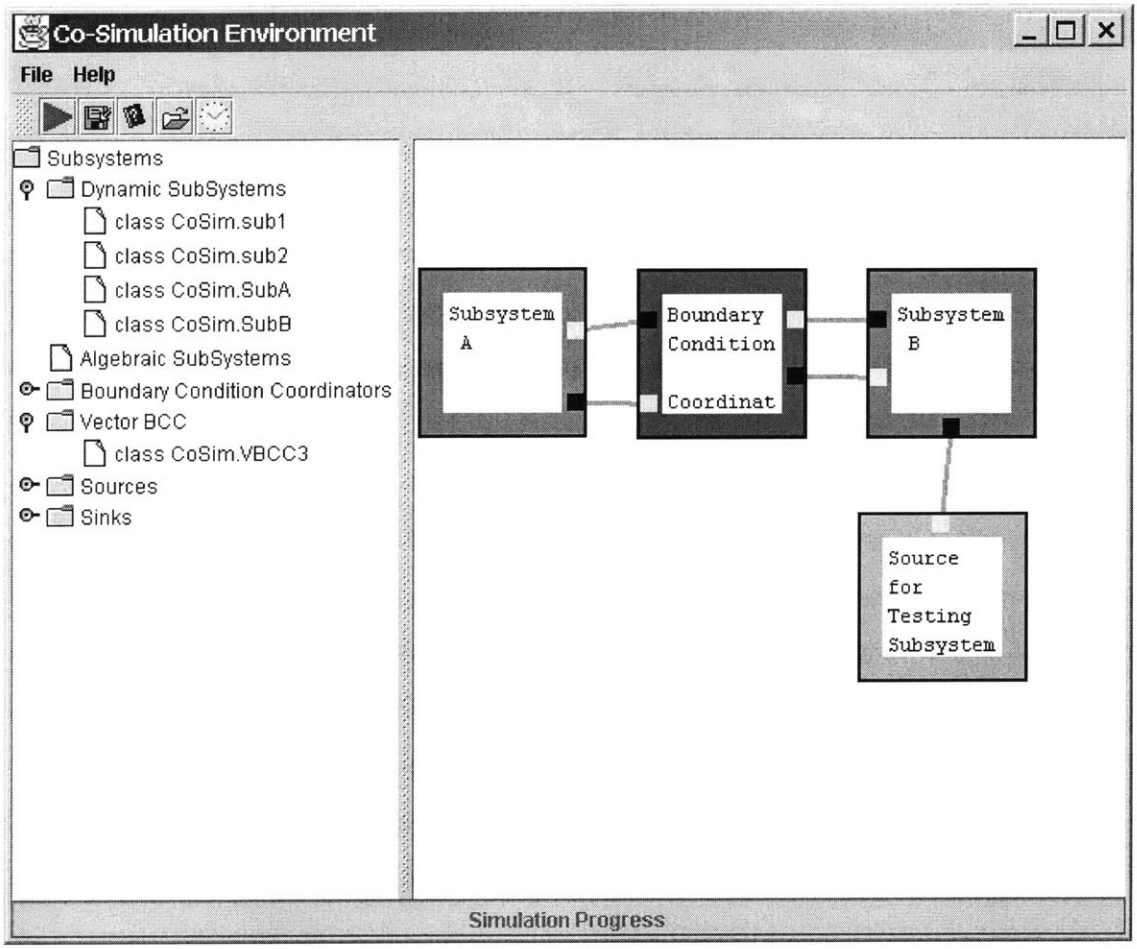
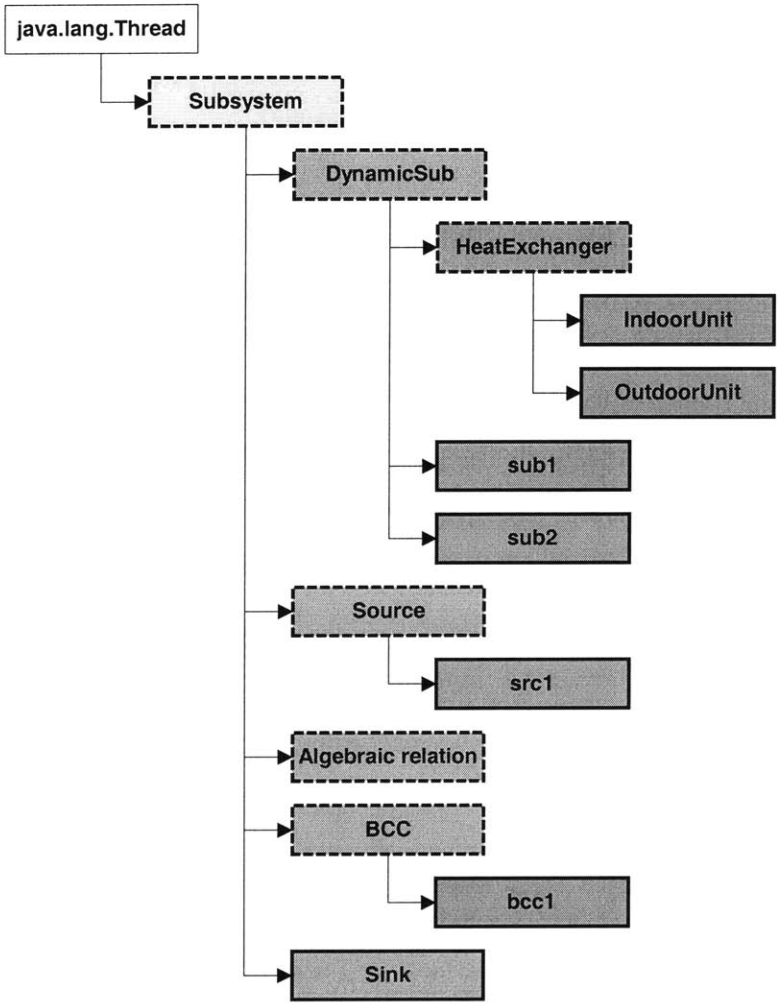


Figure 6.4.1 System integration environment.

For the current version of Co-Simulation, the server function is very simple and we will mainly introduce the system integration part. All available subsystem simulators are shown in the left pane organized by subsystem type. The system integration engineer can pick subsystem simulators shown on the left and place in the system integration pane on the right. Subsystem simulators appear as blocks in the system integration pane. Input and output ports are displayed. The system integration engineers can use the mouse to make connections among subsystem simulators. The integration environment carries this out by calling the *setConnection* method of subsystem simulators. Right clicking the subsystem simulator blocks, the system integrator can select available data to save and plot. Users can also use the mouse to move subsystem blocks and input output ports around to make the connected subsystems and connecting lines in order. The tool bar buttons make it easy for users to start the coupled simulation, save results, etc.



	Base class
	Subsystem base class
	Base classes with different behaviors
	Classes made by subsystem designer
	Instantiatable classes
	Abstract classes

Figure 3.6.8 Class tree of subsystem simulators.

VII Co-Simulation of a HVAC System

7.1 Introduction of Building Energy System

The building energy system includes heating, ventilation and cooling systems in the commercial and residential buildings. The building energy system is a large-scale system that consists of many subsystems. These subsystems may belong to different energy domains. Although the Heating, Ventilation, and Air-Conditioning (HVAC) system appears to be a simple building energy system, it can be complex as well. Figure 7.1.1 shows an HVAC system for small commercial buildings. The manufacturers of the HVAC equipments often work together to supply an HVAC system. The Co-Simulation software environment is a very good tool for cooperation among companies that are involved in the HVAC business. The component makers and system integrators can simulate a large HVAC system to provide more information for design, operation and maintenance. The Co-Simulation provides a good environment for these engineers to share their subsystem simulators and to study different system configurations.

An example of the multi-unit commercial air-conditioning/heat pump system with two indoor heat exchangers is studied in this chapter. When subsystem simulators of the multi-unit air-conditioning/heat pump system are connected, causal conflicts occur between the indoor units and the accumulator. The Boundary Condition Coordinators are used to resolve these causal conflicts.

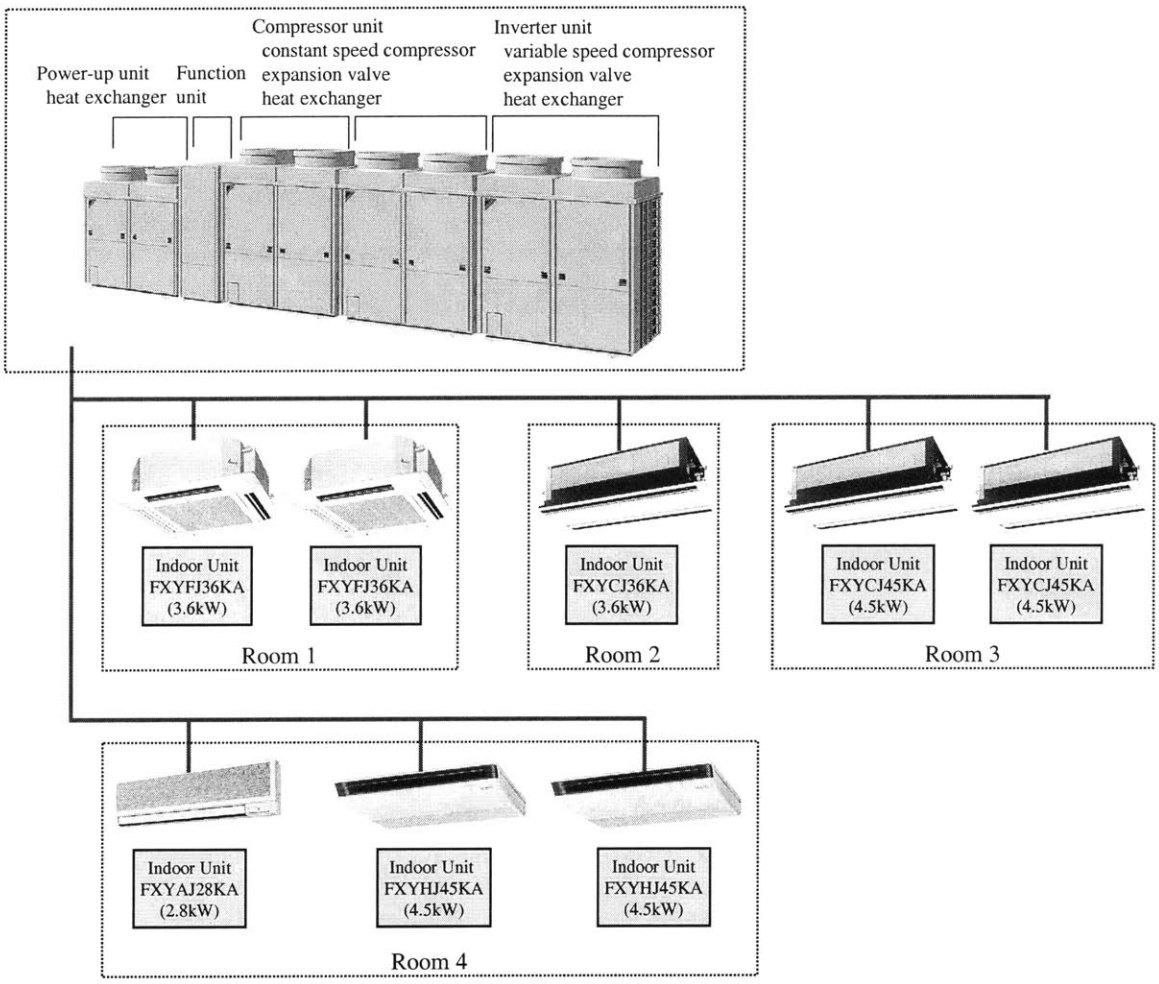


Figure 7.1.1 A commercial building HVAC system (courtesy Daikin).

7.2 Modeling of Multi-unit Air-Conditioning/Heat Pump System

A simple air-conditioning/heat pump system consists of a compressor, an outdoor heat exchanger, an expansion valve, an indoor heat exchanger and an accumulator. Multiple indoor units can be installed in different rooms and the outdoor heat exchanger, the compressor and the accumulator are shared. The multi-unit system considered here comprises seven components, which are shown in Fig. 7.2.1. Among the seven components of the system, there are four types of different modules. The mathematical models for the different types of modules are given in the following section. Only the state variables, inputs and outputs are introduced here. The complete models can be found in [Asada, 2001]. Note that the subscripts used in the following sections are local to each section.

7.2.1 Two-Node Heat Exchangers

There are three components that are modeled as the two-node heat exchangers. These heat exchangers can be simplified into lumped parameter models with four state variables despite they are all distributed parameter systems [He, 1996]. These heat exchangers can operate in either evaporator or condenser mode. In the cooling mode, the indoor units function as evaporators and the outdoor unit functions as a condenser. In the heating mode, outdoor and indoor units exchange their roles.

The model of the two-node heat exchanger is the following

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x},\mathbf{u}) \\ \mathbf{y} &= \mathbf{y}(\mathbf{x}) \end{aligned}, \tag{7.2.1}$$

where

$$\mathbf{x} = [P, \gamma, T_w, T_s]^T, \tag{7.2.2}$$

$$\mathbf{u} = [\dot{m}_{in}, \dot{m}_{out}, h_{in}, T_a]^T \tag{7.2.3}$$

and

$$\mathbf{y} = [P_{out}, h_{out}, P_{in}]. \tag{7.2.4}$$

The variables that make up vectors **x**, **u** and **y** are the average evaporating or condensing pressure, mean void fraction, tube wall temperature, mean refrigerant temperature of the superheated or

sub-cooled section, inlet mass flow rate, outlet mass flow rate, inlet specific enthalpy, surrounding air temperature, outlet pressure, outlet specific enthalpy and inlet pressure, respectively. These parameters are different for the evaporating and condensing mode.

7.2.2 Accumulator

The accumulator is also a heat exchanger but simpler than the indoor and outdoor units, and it has only three state variables. The model of the accumulator is very similar to Eqs (7.2.1) to (7.2.4), but the state vector is defined as

$$\mathbf{x} = [P, \gamma, T_w]^T . \tag{7.2.5}$$

The definitions of the variables and of the input and output ports are the same as those for the four-note heat exchangers.

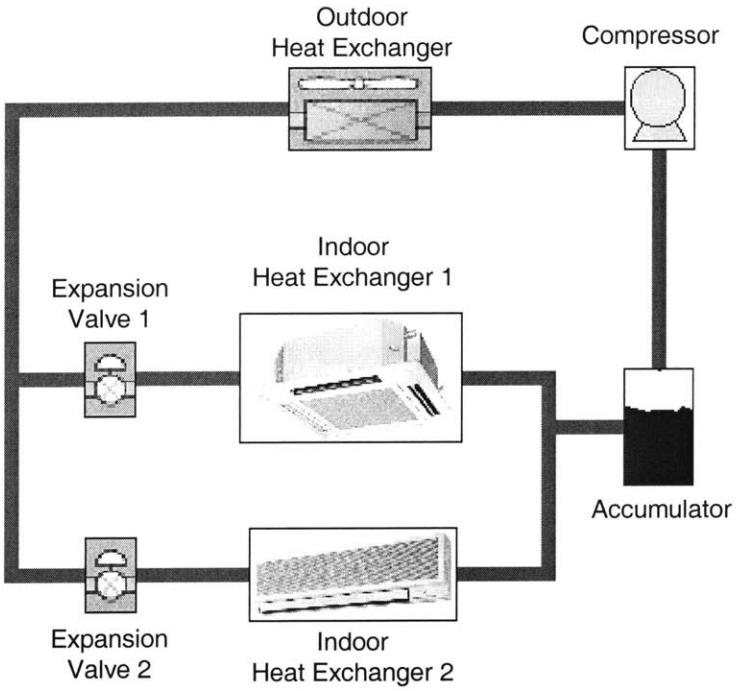


Figure 7.2.1 The two-indoor-unit air-conditioning/heat pump system.

7.2.3 Compressor

The main physical process in the compressor is the momentum transfer, which is much faster than the heat transfer dynamics in the heat exchangers. Therefore, the compressor is modeled as

an algebraic system. The mass flow rate output of the compressor is obtained by empirical curve fitting

$$\dot{m}_{out} = fct(P_{in}, P_{out}, \omega), \tag{7.2.6}$$

where $\dot{m}_{out}$, P_{in}, P_{out} and ω is the outlet mass flow rate, inlet pressure, outlet pressure and compressor speed, respectively.

The outlet enthalpy is obtained by the isentropic outlet enthalpy corrected by isentropic compression efficiency. We have

$$h_{out} = h_{in} + \frac{h_{out,s} - h_{in}}{\eta_s}, \tag{7.2.7}$$

where h_{in}, h_{out}, $h_{out,s}$, and η_s is the inlet specific enthalpy, outlet specific enthalpy, outlet isentropic specific enthalpy and isentropic compression efficiency, respectively.

The input and output vector is defined as follows:

$$\mathbf{u} = [P_{in}, P_{out}, \omega, h_{in}] \tag{7.2.8}$$

and

$$\mathbf{y} = [\dot{m}_{out}, \dot{m}_{in}, h_{out}]^T. \tag{7.2.9}$$

The definitions of the variables are the same as the previous section.

7.2.4 Expansion Valve

It takes the refrigerant a very short period of time to flow through the expansion valve. Comparing it to the speed of the heat transfer processes in the heat exchangers, the dynamics of the expansion valve can be treated as an algebraic relation. The expansion is an adiabatic process since the heat transfer is not significant in that time scale. Therefore the enthalpy of the flow does not change, and we have

$$h_{out} = h_{in}. \tag{7.2.10}$$

The mass flow rate output can be calculated by applying the orifice equation, and we have

$$\dot{m}_{out} = c_v A_v \sqrt{\rho(P_{in} - P_{out})}\,, \tag{7.2.11}$$

where c_v, A_v and ρ is the orifice coefficient, expansion valve opening and average density, respectively. The input and output vectors are defined as follows:

$$\mathbf{u} = [P_{in}, h_{in}, P_{out}, A_v] \tag{7.2.12}$$

and

$$\mathbf{y} = [\dot{m}_{in}, \dot{m}_{out}, h_{out}]^T\,. \tag{7.2.13}$$

7.3 Co-Simulation Setup

7.3.1 Subsystem Simulator Coding

The subsystem simulators of components of the air-conditioning system are built based on the models introduced in the previous section. Class `HeatExchanger` for the two-node heat exchanger modules is derived from the base class `DynamicSub`. This heat exchanger class can run in both evaporator and condenser mode. The `HeatExchanger` class is an abstract class, which defines all behaviors of the two-node heat exchanger. However, the geometric and heat transfer parameters are not defined in the `HeatExchanger` class. We use the `HeatExchanger` class as the template to develop `IndoorUnit` class and `OutdoorUnit` class by simply specifying the geometric and heat transfer parameters. Other modules modeled as two-node heat exchangers can use the `HeatExchanger` class as a class template as well.

Since we only use one type of accumulator in the simulation, the `Accumulator` class is developed as a regular class, which can be instantiated. This class has similar input and output port definition as the `HeatExchanger` class. However, since multiple indoor units may connect to the accumulator in different system configurations, the input port of the inlet mass flow rate, the input port of the inlet specific enthalpy and the output port of the inlet pressure can be split into multiple ports.

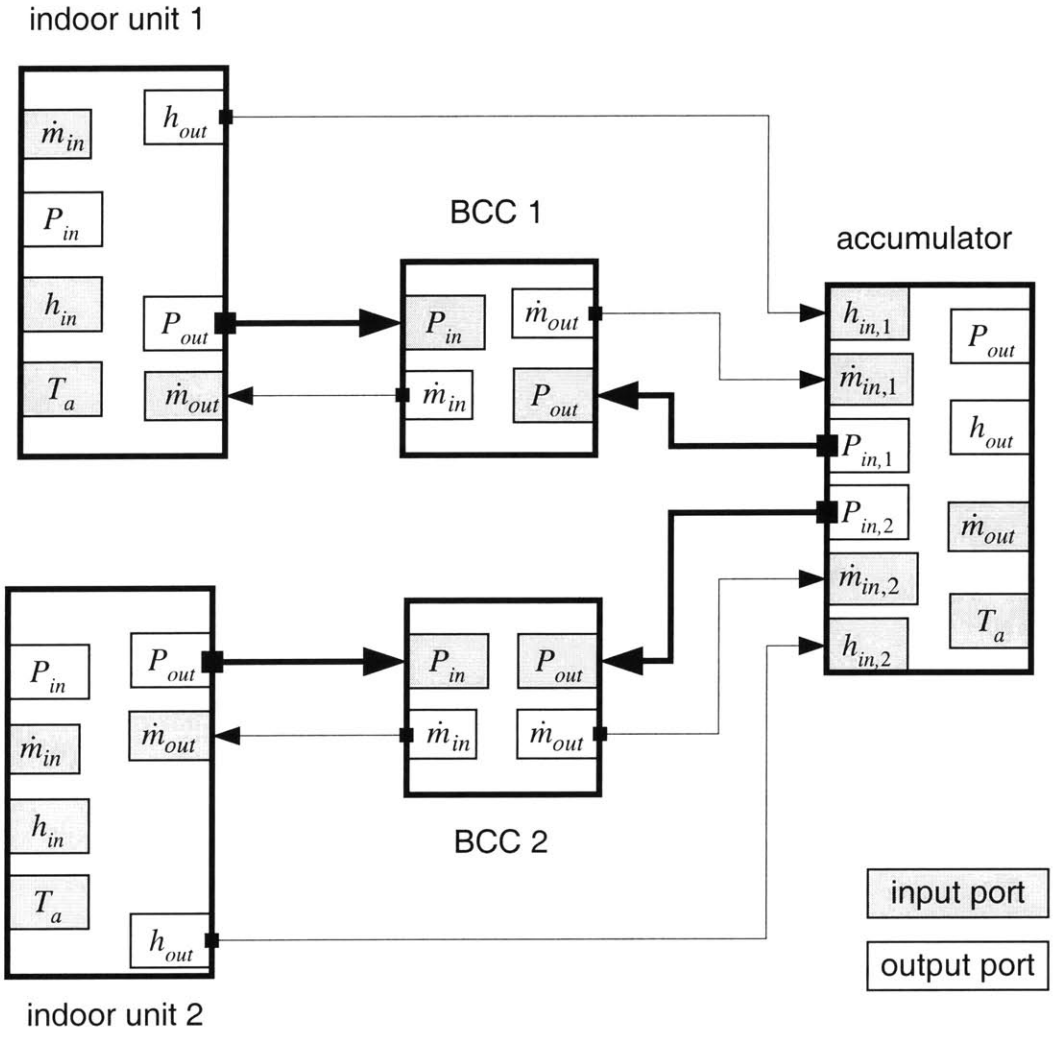


Figure 7.3.1 BCCs connecting the indoor units and the accumulator.

The two algebraic systems, i.e. the expansion valve and the compressor, are easily coded into Class `ExpValve` and Class `Compressor` based on the class template `AlgebraicSub`. We also defined a boundary condition subsystem simulator based on the class template `Source`. The boundary condition subsystem provides the indoor air temperatures, outdoor air temperature, compressor speed, and expansion valve openings.

7.3.2 System integration

To demonstrate the coupled simulation of multi-unit air conditioning/heat pump system, we use two indoor units in the system. Figure 7.2.1 shows the seven physical modules in the air conditioning/heat pump system. However, there are causal conflicts between the two indoor units and the accumulator. The accumulator and the indoor units all take pressure as the input and produce the mass flow rate as the output. Therefore, coupled simulation has to satisfy two algebraic constraints.

The first algebraic constraint is

$$\dot{m}_{out,1} = \dot{m}_{in,1}, \tag{7.3.1}$$

where $\dot{m}_{out,1}$ and $\dot{m}_{in,1}$ are the outlet mass flow rate of the first indoor unit and the first inlet mass flow rate of the accumulator, respectively. The other algebraic constraint is defined similarly between the second indoor unit and the second inlet of the accumulator. If we differentiate the algebraic constraints with respect to time twice, we will find the time derivative of the pressure of indoor units and the accumulator [Gordon, 1999]. Therefore, the algebraic constraints are index 2.

When the multi-unit air-conditioning/heat pump system is simulated in the Co-Simulation software environment, we have seven subsystem simulators based on physical modules, two BCCs for the algebraic constraints and one source type subsystem simulator providing the boundary conditions. The ten subsystem simulators are connected by matching the input and output ports of the subsystem simulators. In order to show the connections of the BCCs clearly, Figure 7.3.1 only shows the two indoor units, two BCCs, accumulator and connection lines among these five subsystem simulators. Other ports are also connected since the entire loop shown in Fig. 7.2.1 is simulated.

The thin connection lines in Fig. 7.3.1 represent the data transfers that are defined by input and output ports. However, the thick connection lines represent the transmission of the data objects that consist of the pressure and the time derivative of the pressure as a function of the mass flow rate. This capacity is predefined in the `HeatExchanger` and `Accumulator` classes. When these two subsystem simulators are connected to the BCCs, the system integrator software automatically sets the data type that will be transmitted.

7.4 Results

The system is set to run in the air-conditioner mode. The Co-Simulation starts from an inconsistent initial condition. The pressure in the accumulator is 650 kPa and the pressures in the indoor units are 700 kPa. The two indoor units are located in two different rooms. The air temperature in the two rooms is 27 °C and 30 °C, respectively. The simulation results are shown in Fig. 7.4.1 through Fig. 7.4.6.

Since two BCCs are used in this simulation, there are two sliding functions. The sliding functions are driven to the sliding manifolds quickly, so Fig. 7.4.1 only shows the simulation from time 0 to 0.8 second. The outputs of the two BCCs are the mass flow rates, which are shown in Fig. 7.4.2. Figure 7.4.3 shows the state variable pressure of the four heat exchangers. As a result of the algebraic constraints, the two indoor units and the accumulator should have the same pressure. Although the starting pressures are different in the accumulator and the indoor units, they are forced to satisfy the algebraic constraints within a certain boundary layer. Figure 7.4.4 shows the state variable mean void fraction in four different heat exchangers. The state variable tube wall temperature is shown in Fig. 7.4.5. Figure 7.4.6 shows only three curves of the refrigerant temperature since the accumulator has only three state variables. The refrigerant temperature is the superheated temperature in the evaporators and is the sub-cooled temperature in the condenser.

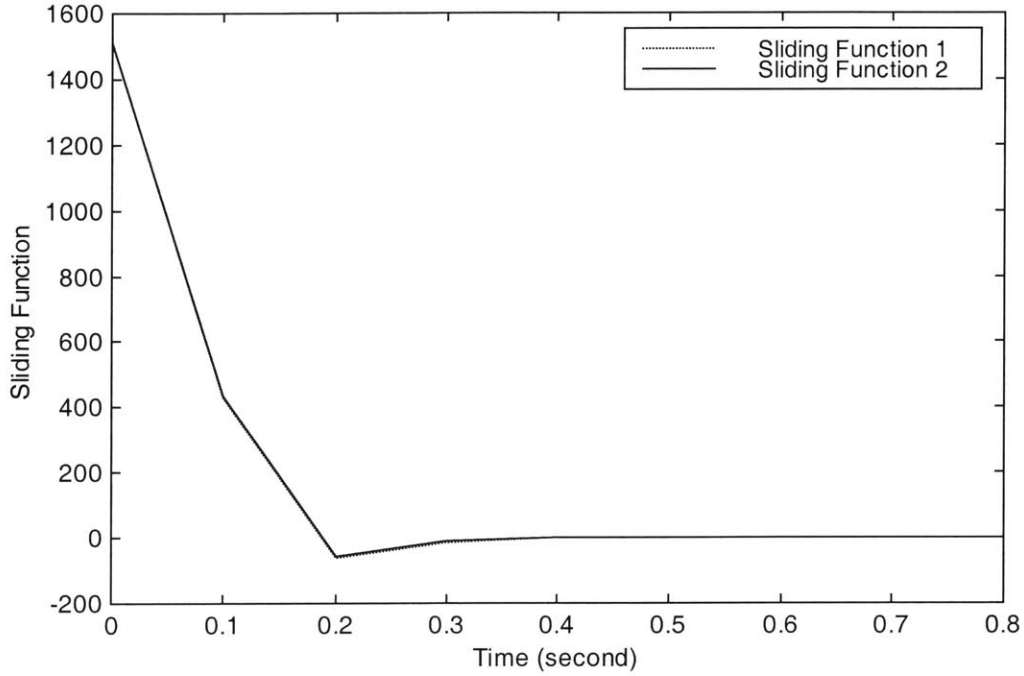


Figure 7.4.1 Plot of sliding function s.

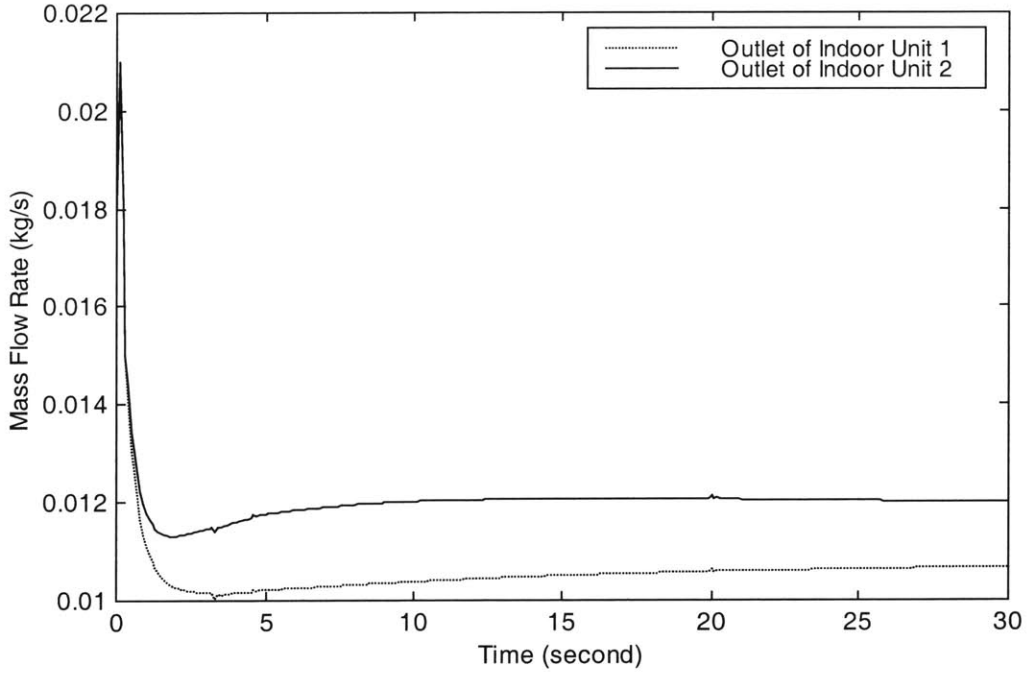


Figure 7.4.2 Plot of mass flow rate $\dot{m}_{out}$.

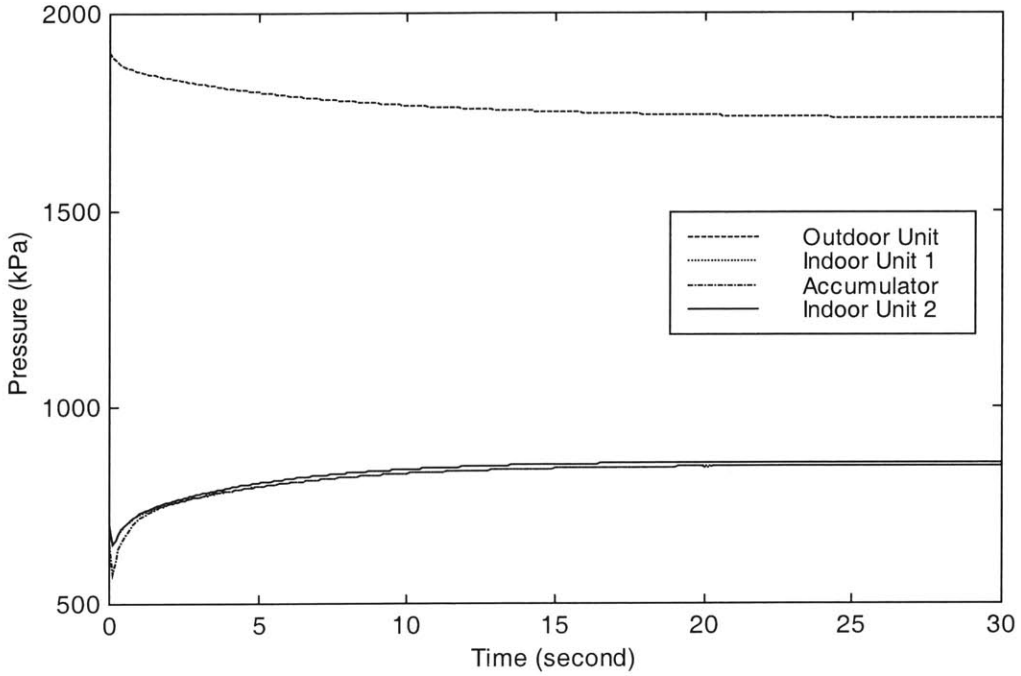


Figure 7.4.3 Plot of Pressure P.

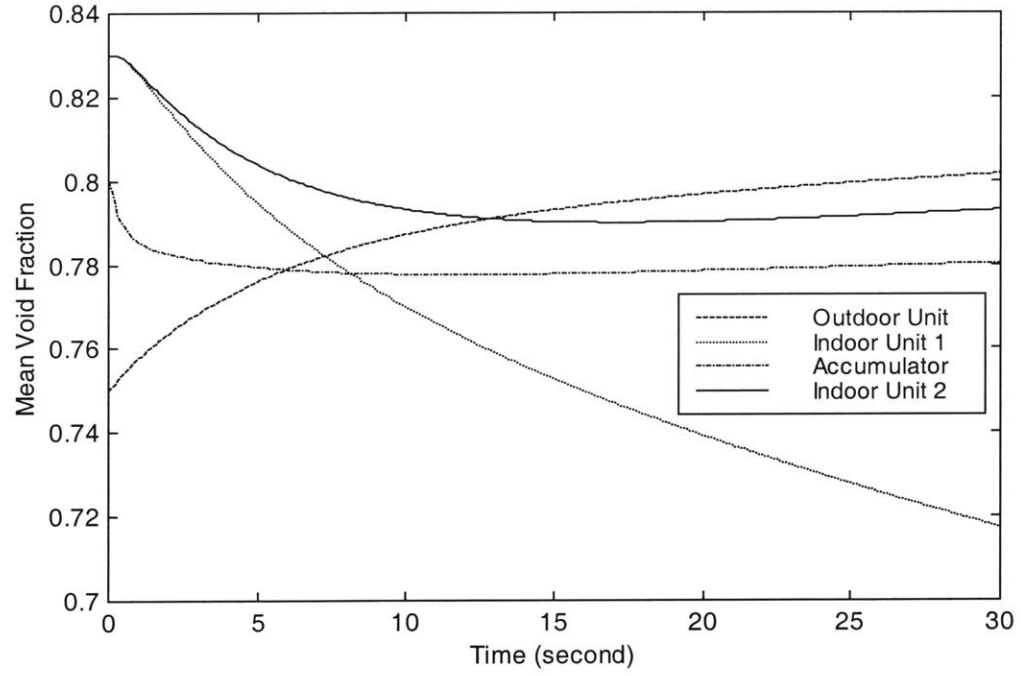


Figure 7.4.4 Plot of Mean void fraction γ

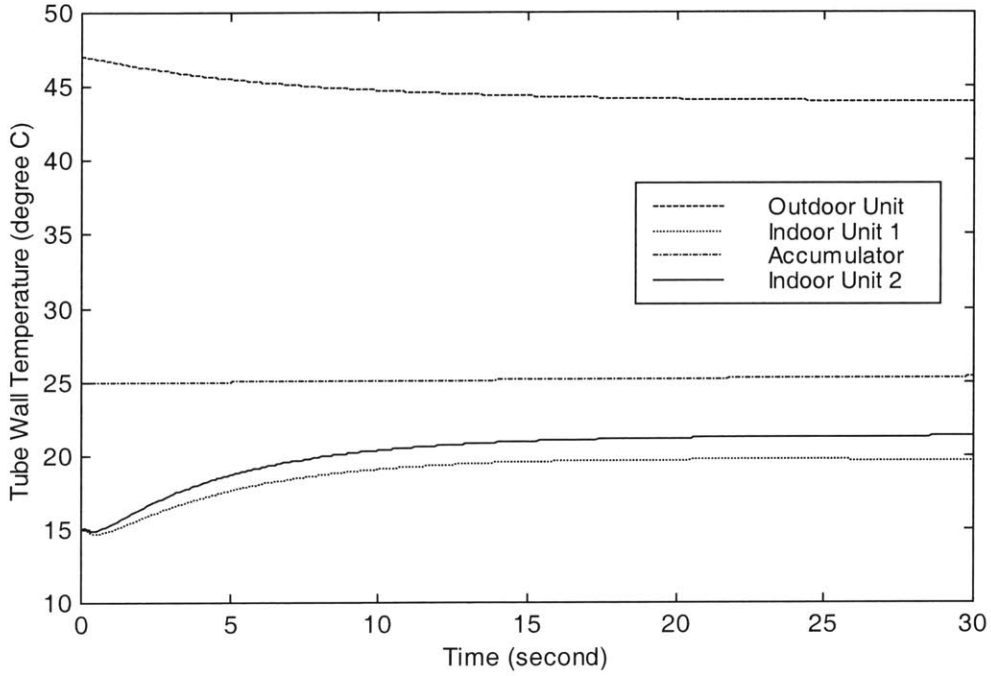


Figure 7.4.5 Plot of tube wall temperature T_w.

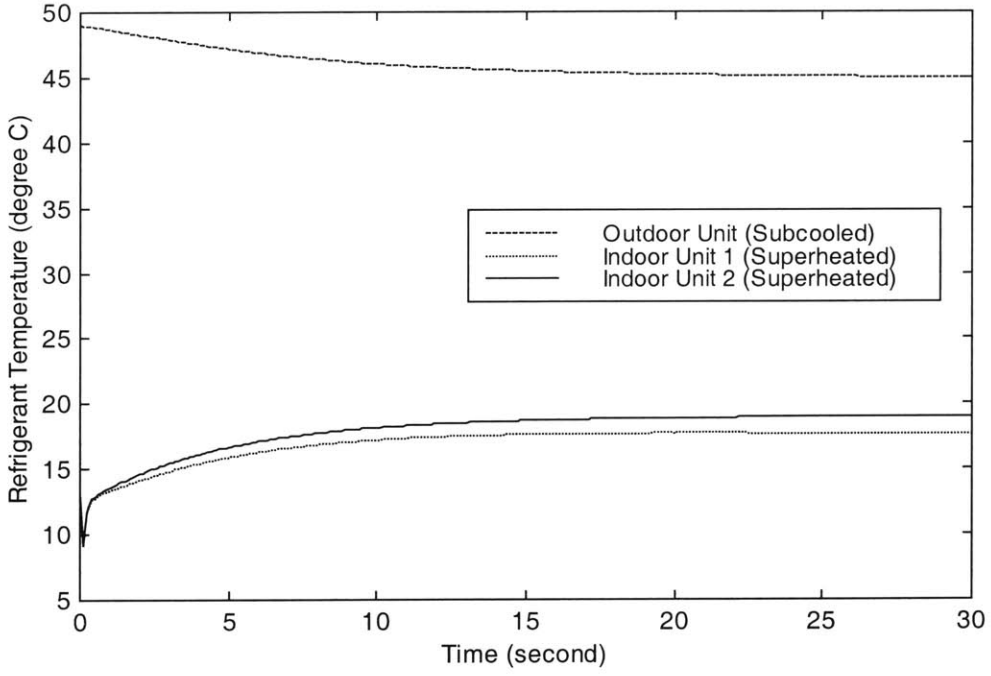


Figure 7.4.6 Plot of refrigerant temperature in superheated or sub-cooled region T_s.

VIII Conclusions and Directions for Future Work

This thesis presents a method for simultaneously running a collection of dynamic simulators coupled by algebraic boundary conditions. Termed as "Co-Simulation", this new simulation environment facilitates companies in a supply chain or alliance to communicate by exchanging subsystem simulators. Subsystem simulators may dynamically be coupled with each other through boundary conditions given by algebraic constraints. Dynamic interactions are simulated without use of proprietary information of the subsystem models. All computations are performed based on input-output numerical data of encapsulated subsystem simulators coded by independent groups.

Combining such subsystem simulators may incur non-causal relationship and numerical instability, which is identified as the major difficulty of such object-oriented simulations. Interacting subsystems with a causal conflict are described as a high-index, Differential-Algebraic Equation. The Boundary Condition Coordinator is used to resolve these causal conflicts and a systematic solution method is developed using Discrete-Time Sliding Mode control. First, stability and convergence conditions as well as error bounds are analyzed by using control theory. Second, the algorithm is modified such that the subsystem simulator does not have to disclose its internal model and state variables for solving the overall DAE. Third, a multi-rate algorithm is developed for improving efficiency, accuracy, and convergence characteristics.

Numerical examples verify the major theoretical results and illustrate features of the proposed method. A software environment for Co-Simulation is developed. This software environment, written in Java, defines the protocol for subsystem the simulators. Subsystem simulators and BCCs run as independent processes in the Co-Simulation environment. Class templates containing all necessary functions for the different types of subsystems are defined. A real world problem consisting of a multiunit air-conditioning/heat pump system is simulated using the Co-Simulation software environment.

However, the basic assumption of the coupled simulation considered in this thesis work is that the algebraic constraints considered result solely from causal conflicts are considered. Under this assumption, the solution of the DAE system always exists. Hence it is not necessary to verify the existence of the solution before the DAE system is solved. Also because of this

assumption, the index of the DAE will be limited to three. Additionally these energy transmission-based modeling tools, such as Bondgraphs, can be used to find possible input-output pairs, which may be inverted and thus form causal conflicts. For such input-output pairs, the derivatives of the outputs can be prepared for possible causal conflicts. If the Co-Simulation software environment is used to simulate coupled system with arbitrary algebraic constraints, any combination of input and output can form an input-output pair of high relative order. The preparation for inverting all arbitrary input-output pairs is not feasible when the system is large. Moreover, the asymptotic stability of the zero dynamics cannot be assumed. Therefore, some analytical capabilities will be required for the Co-Simulation software environment if other types of algebraic constraints are to be considered.

For some subsystems, one component of the output object, $y^{(r-1)}(z)$, needs extensive computation to obtain. The computation of this function may critically affect the computation time of the multi-rate BCC since this function is evaluated many times during one time step. The computation of this function may be replaced by a nonlinear map to enable more efficient computation of the BCC.

The extension of this feedback control-inspired approach for resolving causal conflicts to Finite Element Analysis may be interesting. The essence of this approach is to invert only the relevant input-output pair by guided iteration in order to resolve the algebraic constraints, rather than iteration for the entire system. Application of this approach to distributed parameter systems may substantially reduce the computational effort since a number of algebraic constraints in the spatially discretized system may be large.

References

Andersson, M., 1990, *An Object-Oriented Language for Model Representation*, Licenciate thesis TFRT-3208, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Asada, H. H., Gu, B., and Gordon, B.W., 2000, "A unified approach to modeling and realization of constraint robot motion using singularly perturbed sliding manifolds," *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 736 –743.

Asada, H. H., He, X.-D., Gu, B., Tarraf, D. C., 2001, Simulation and Control of Building Energy Systems, Report No. 1, Progress Report to the Daikin Air-Conditioning R&D Laboratory, Ltd., Osaka, Japan.

Baugarte, J., 1972, "Stabilization of Constraints and Integrals of Motion in Dynamical Systems," *Comp. Math. Appl. Mech. Eng.* Vol.1, pp. 1-16.

Brenan, K., Campbell, S., and Petzold, L., 1989, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, Amsterdam, North-Holland.

Campbell, S.L., 1995, "High-Index Differential Algebraic Equations," *Mechanics of Structures and Machines*, Vol. 23, No. 2, pp. 199 - 222.

Cellier, F.E. and Elmqvist H., 1993, "Automated Formula Manipulation Supports Object-Oriented Continuous-System Modeling," *IEEE Control Systems*, Vol. 13, No. 2.

Delonga, D. M., 1989, *A Control System Design Technique for Nonlinear Discrete Time Systems*, Ph. D. Thesis, Massachusetts Institute of Technology.

Drakunov, S. V. and Utkin, V. I., 1992, "Sliding Mode Control in Dynamic Systems," *International Journal of Control*, Vol. 55, No. 4, pp. 1029 – 1037.

Gordon, B. W. and Liu S., 1998, A Singular Perturbation Approach for Modeling Differential-Algebraic Systems. *ASME Journal of Dynamic Systems, Measurement, and Control*, December 1998.

Gordon, B. W., 1999, *State Space Modeling of Differential-Algebraic Systems using Singularly Perturbed Sliding Manifolds*, Ph.D. Thesis, Massachusetts Institute of Technology.

Haier, E., and Wanner, G., 1991, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems,* Springer-Verlag, New York.

He, X.-D., 1996, *Dynamic Modeling and Multivariable Control of Vapor Compression Cycles in Air Conditioning Systems*, Ph.D. Thesis, Massachusetts Institute of Technology.

Hogan, H., 1987, "Modularity and Causality in Physical System Modelling," *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol. 109, pp. 384-391.

Karnopp, D., Margolis, D. and Rosenberg, R., 1990, *System Dynamics: A Unified Approach*, Wiley-Interscience, New York, NY.

Khalil, H. K., 1996, *Nonlinear Systems*, Second Edition, Prentice Hall, Upper Saddle River, NJ, USA.

Kokotovic, P., Khalil, H. K., and O' Reilly, J., 1986, *Singular Perturbation Methods in Control: Analysis and Design*, Academic Press.

Mattsson, S. E., and Soderlind, G., 1993, "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives," *SIAM Journal of Computing*, Vol. 14, No. 3, pp. 667-692.

Mattsson, S.E., Elmqvist, H., and Otter, M. 1998, "Physical system modeling with Modelica," *Control Engineering Practice*, Vol. 6, pp. 501-510.

Otter, M., and Cellier, F. E., 1996, "Software for Modeling and Simulating Control Systems," *The Control Handbook*, Levine, W. S., Ed., CRC Press, Inc., pp. 415-428.

Petzold, L. R., 1983, "A Differential/Algebraic System Solver," *Scientific Computing*, Stepleman, R. S. et al., Eds., Amsterdam, North Holland, pp. 65-68.

Senin, N., Wallace, D. R., and Borland, N., 2000, "Object-based Simulation in a Service Marketplace," *to be appeared in ASME Journal of Mechanical Design.*

Sinha, R., Paredis, C. J. J., Liang, V.- C., and Khosla, P. K., 2001, "Modeling and Simulation Methods for Design of Engineering Systems," *Journal of Computing and Information Science in Engineering*, Transactions of ASME, Vol. 1, pp. 84 – 91.

Slotine, J. - J. E. and W. Li, 1991, *Applied Nonlinear Control*, Prentice Hall.

Sun Micro Systems, 2001, *The Source for Java ™ Technology*, http://java.sun.com.

Utkin, V., Guldner, J., and Shi, J., 1999, *Sliding Mode Control in Electromechanical Systems*, Taylor & Francis Inc.

Vidyasagar, M., 1993, *Nonlinear Systems Analysis*, Second Edition, Prentice-Hall International, Englewood Cliffs, NJ, USA.

Wallace D. R., Abrahamson, S., Senin, N., and Sferro, P., 2000, "Integrated Design in a Service Marketplace," *Computer-aided Design*, Vol. 32, no 2, pp. 97-107.

Yu, X., 1994, "Digital Variable Structure Control with Pseudo-Sliding Modes," *Variable Structure and Lyapunov Control*, Zinober, A. S. I., Eds., Springer- Verlag, pp. 134-159.

Zeid, A. A. and Overholt J. L., 1995, "Singularly Perturbed Bond Graph Models for Simulation of Multibody Systems," *ASME Journal Dynamic Systems, Measurement and Control*, Vol. 117, pp. 401-410.