Simulation is an important aspect in product development.

ABSTRACT:

Off-line simulation is important prior to the actual development to investigating its feasibility and success. Real-time simulation is important (e.g., in traction control) as it provides a realistic testing procedure and a means for fine-tuning the different control strategies. An Interactive real-time vehicle simulator has been developed on an IBM PS/2 486 PC for low cost and portability as described in [1]. This paper describes an object-oriented approach used to logically model the physical system of an automobile. This paper concentrates on the software issues in a real-time design. Although this design is specific to the vehicle simulator used for traction control, it could easily be adapted for any other type of simulation as well. Previous simulations [2], [3] had been done where the user is the same person that designed the simulation, therefore not paying too much attention on the software maintenance and user interface requirements. Good design strategies are necessary for maintenance and development of the simulator during its entire life cycle, increasing its capabilities and making it more versatile. As the system is expanded, there will be the need for more processing power to meet certain hard deadlines, requiring the models to be moved to a distributed computing environment, discussed in [8], further underlying the need for an object-oriented design.

I. INTRODUCTION

During the entire software life cycle, logical modeling of the system plays an important role in designing a distributed computing system. For a distributed system, the software components of the system reside on separate processors executing code sequentially, ultimately fulfilling the entire task. Therefore, the entire task needs to be partitioned into subtasks so that the software for each of these subtasks could be allocated onto separate processing units. In the case of our vehicle simulator, the software was designed for a distributed processing environment, although at present it runs on an IBM PS/2 with a single processor.

Originally, for logical modeling, the functional decomposition technique was used. In functional decomposition, we map the problem into various functions and sub-functions that need to be executed to solve the problem. This function mapping is done by analyzing the problem closely. Since intuition plays a major role, there is no direct map between the problem and the functions. The functions should be such that there is none or very little cohesion between them so that any changes in one function only minimally affects the other functions. Therefore, the breakdown of these functions is difficult and prone to much change. To overcome the defects in the functional technique, the dataflow diagram was introduced. In the dataflow approach, the software is decomposed into modules according to its functionality at different levels in the system. The problem is mapped into dataflows and modules. As discussed in [8], the traditional dataflow-oriented design approach, structured analysis and structured design [5], emphasizes strongly on functional decomposition, whereas the decisions on interprocess communication and synchronization are done only in the implementation phase. Therefore, the same weakness still exist. The dataflow approach has a very weak emphasis on the data structure [7]. The software system which is represented by dataflow diagrams may sometimes require redesigning to obtain the required performance. The communication-oriented design approach, Petri-Nets [6], deals primarily with the communication and synchronization aspects of the software system overcoming those shortcomings inherent in the previous approaches. The Petri Net approach is used especially in problems where the computation and data structures are relatively simple. It is good in a communication intensive application, but not in a computationally intensive application like vehicle modeling. Recently, the object-oriented paradigm has been getting much attention especially in large scale application software for enhancing program comprehension for easier development and maintenance during its life cycle. The object-oriented approach directly maps the problem into the model. Instead of an indirect mapping from the problem into the function or dataflow,
The basis of the object-oriented approach is to identify the relationships among objects in the problem domain. The model is made up of entirely objects and messages which are defined in the real world e.g., automobile. It is especially useful in a problem that consists of many different functions or responsibilities. The main merits of the object-oriented analysis is that its main emphasis is on the problem, it provides more stability for changes in the requirement specifications, and it provides a smooth transition from the analysis phase to the design phase. An object is an entity that encapsulates some local data and operations (functions) called methods. An object is an instance of a class. A class represents all objects with some similar characteristics. The local data within the object can be encapsulated so that the data is hidden from other objects. Objects communicate by sending and receiving messages. Messages are the only means of manipulating another object's encapsulated data. An object can also inherit characteristics from its base classes. An example, consider the graphical interface which uses figures of various shapes like boxes, triangles, labels, windows, etc. Figure 1, shows the class hierarchy for such a group of figures, where the arrows indicate the inheritance relationships of the classes. Inheritance greatly simplifies the design and implementation as new classes can be derived from one or more of its base classes. These new classes inherit all the methods of its base classes, by default, unless the class provides new methods that override or supplement them. The features and applications of object-oriented design is documented in [4], [7]. Emphasis is placed on the characteristics of the software e.g., components of the software that can be parallelized, components that need corporation through communication and synchronization, partitioning of the software models for allocation onto different processing elements for further expansion, etc. Generally, object-oriented programming requires a substantial amount of memory. Therefore in the past, object-oriented models were avoided in real-time applications because of the perception that such run-time systems perform dynamic memory management which requires a long unpredictable amount of time [9]. However, memory is now no longer a limiting factor even in PC-based applications. The application problem and the procedure used to derive its objects are described in sections II and III respectively.

II. APPLICATION - Problem domain

The Interactive real-time vehicle simulator, fully described in [1], provides a real time hardware-in-the-loop vehicle simulation for traction assist. Traction control equipment is already included in some vehicles and will become a standard feature in the future. Its usefulness is clearly seen when driving, accelerating, or negotiating a curve, on a slippery surface. Generally, the problem of wheel lock during braking is already included in the antilock brake system (ABS). When a wheel slip is detected, the different combinations of the Engine and Brake management techniques are applied to control slip so as to obtain the optimum performance in stability, comfort, maneuverability and safety. Brake management alone is insufficient due to the immense amount of heat generated during traction control. Therefore reducing the drive torque during the application of brakes will definitely have a lower impact on the brakes. Engine torque reduction by way of retarding spark, cutting off some injectors, and closing the throttle, together with the brake control strategy, has been shown to more effectively control slip.

An analysis of the latest and future traction control technology trends is discussed in [10]. In our simulation, since some of the components are of a high degree of complexity (e.g. Electronic Engine Control Unit (ECU)) and even proprietary from other manufactures (e.g. ABS), it becomes necessary to have these hardware in the loop. In order to develop and improve the traction control strategies, we must be able to vary different parameters on-line and see its influence on different vehicle parameters. Generally, traction assist controls the longitudinal slip when the vehicle starts or accelerates on a smooth surface, this simulation will also help to investigate the effect of the lateral slip when a vehicle negotiates a turn (the yaw rate). The simulation also helps in the use of repeatable testing conditions as the surface of ice or snow changes drastically with temperature, and some potentially dangerous driving modes could be tested, which help in reducing the costly winter testing effort as now the initial investigation could be done with the simulator before use in the test vehicle. The simulator should contain the model of the whole vehicle, but the complexity is sufficient to reflect the effect of traction control on the vehicle. Subsequently, this simulator will need to be improved to have a more complete vehicle model that represents all effects of the real vehicle which will make the simulator more generic, suitable for other simulations as well.

III. OBJECT MODEL

Figure 2, shows the overall logical model of the simulator for the closed loop control system. By logical modeling we mean the ability to map the real world problem into a modular system suitable for expansion, model improvement, and the ability to port the system to a distributed environment with minimal changes. As discussed in section II, the simulator will need to be regularly updated to encompass more complex models for future development purposes. Therefore, the object-oriented design approach seems to be the most natural analysis required for such a project that needs constant enhancements, as the new design should be able to use most of the previous design and code. This analysis
focuses on the design itself, with some rules to a good design process. Coad and Yourdon [7] describe the object-oriented analysis in five major activities - Finding class and objects, identifying structures, identifying subjects, defining attributes, and defining services. Yau [8] incorporates the distributed computing environment into the analysis.

The first step is to distribute the functions of the vehicle into separate physical models. A modular approach was used so that a distributed processing system could be adapted. The automatic transmission vehicle was divided into the following basic models:

- Driver (user interface)
- Engine
- Torque Converter
- Driveline
- Brake
- Tire (separate for left & right)
- Vehicle

The driver model represents the users input/output interface to the overall system. The inputs required by the simulator are the primary throttle position, idle engine speed (rpm), type of road surface like inclination and frictional coefficient, respectively. The driver controlled throttle could be varied during the run, but the sequence of the variation in the properties of the road have to be initially preset. The outputs are the different vehicle parameters needed to verify the effectiveness of traction control e.g. driven wheel speeds, vehicle speed, engine, shaft and brake torques, etc., which could be viewed graphically. The engine model represents the features of the variation of throttle, spark and injectors on the torque produced. The engine torque feeds a torque converter model which smoothes the torque transitions during gear shifts. It provides the maximum torque multiplication, fed to the driveline model, which progressively decreases with increase in engine speed. The driveline model represents the mechanical linkage of the gear box and half shafts to the wheels. The brake model represents the effect of variation of brake torque due to the fluid pressure in the brake lines, including the activation delays in braking. The tire model represents the lateral and longitudinal forces on the wheel including the effects due to road surface conditions. The vehicle model incorporates the drag forces due to wind resistance and when the vehicle moves up an incline.

In order to map these different models into objects, we use an iterative process which adds one component class at a time and by combining the approaches proposed in [7] and [8] as described next. The identification of suitable objects is a key factor in the design approach. But there is no fixed method of doing that. The objects are created from the physical models so that is depicts and represents the properties of the actual system.

The second step is the logical modeling, in which we identify the corresponding objects and then derive the classes for them. The Objects are identified by analyzing the requirement specifications of each of the physical models. Classes are built for each of the models described in the first step. This is most convenient because then we can easily update each of these classes for different types of vehicles, since all vehicles will consist of these basic classes.

The third step is to identify the messages between each of the objects, thereby being able to define how the objects interact with each other. This is needed in defining whether the data/methods need to be private or public.

Figure 3, shows the object communication diagram of the vehicle simulator. The direction of the arrow indicates the flow of information.

The fourth step is to determine the data structure of the local variables and the algorithm for each method. For each class the relevant data structure is selected and methods that manipulate this data are defined. As an example, the class engine has local variables throttle, manifold pressure, engine rpm and torque of type real, and spark, injectors as type integer respectively. There are methods for calculating the mass air flow through the throttle, engine torque as a function of the delayed air flow into the cylinders, spark, engine speed, injectors respectively, etc.

The final step is the partitioning of the objects into subtasks to be allocated to individual processing units. The subtasks are determined so as to minimize the communication between them and maximize parallelism, thereby these tasks can be distributed on to individual processors for efficient execution. This speeds up the entire processes resulting in a faster execution time. In a mixed-mode type simulation [3] where some of the very complex hardware control units can be suitably modeled in software, these tasks could then be conveniently inserted into or extracted from the closed loop whenever a non-real time or real time mode of execution is required, as shown in Figure 4. If there is a sufficient number of processors to process each object, then we do not require any partitioning. But when there are fewer processors than objects, then we require some accumulation of objects within each processor. For example, in the engine object, the torque converter object is combined and the output from that combination, which is the output torque, flows into the driveline.

The models were initially developed in MatrixX [11], for ease of simulation and validation, and subsequently programmed using the object-oriented language C++ and Borland's Application Framework. Objectwindows [12] was used in developing the application in a Microsoft windows environment. It has an object-oriented class library that encapsulates the behavior that windows applications commonly perform, reducing development
time. The windows in our application are derived from the basic window frames provided in their class library. The resource workshop editor was used to create buttons, dialog boxes and pull down menus. The buttons enabled the user to select a particular operation, e.g. start simulation, plot, view files and documentation, etc. The dialog boxes allow convenient user input. The pull-down menus within each window facilitates the selection of different options in each category as depicted by its title. Figures 5a-d show the snapshots of the screen at different points during the simulation. Figure 5a shows the initial simulation control panel. Button "Sim Models" invokes the window shown in Figure 5b, which allows the user to pick objects from the pull-down menus under each category, such as a particular engine type, driveline, tires with different sizes and standard threads, etc., from the available objects, depending on the vehicle being simulated. Button "Build Sim" then dynamically links these selected objects. Button "Activate Sim" allows the user to either enter new initial conditions or retrieve previous settings through the dialog boxes, as shown in Figure 5c. Once the user is satisfied, he starts the simulation. On completion of the run, the user activates the button "Plot Data", which brings up the screen shown in Figure 5d (on ice with wide-open-throttle), for plotting the selected parameters, selected from the pull-down menu "Y-axis", either individually or overlaid with the same parameter from a previous file.

IV. CONCLUSION

The object-oriented design approach was used, as the software structure reflects the structure of the application problem, since everything can be represented as objects and messages. It promotes a smooth transition from system analysis all the way to the actual code. The inherent parallelism in the software system can be modeled naturally. The data abstraction and encapsulation reduces the interdependencies between the software components and consequently facilitates modification. The inheritance mechanism also allows reuse of the code enhancing maintainability. The advantage of the object-oriented approach is not only its closeness to the real world situation but also the ability to run the code allocated to each object on different processors like transputers or digital signal processors to speed up overall program execution. This enables constant model development with increasing complexity during the life cycle of the system. Comparing other implementations, [2] represents a mainframe type vehicle simulator which has some good models but lacks the real-time aspect, while [3] is a good distributed processing real-time simulator but does not have a user friendly interface and also lacks a realistic engine model, implemented specially for ABS/TA (Traction Assist) development only. Our simulator is user friendly, user interactive, and facilitates continuing development.

ACKNOWLEDGMENTS:
We sincerely thank those in the Electrical and Electronics group at Ford Motor Co. for all the support and encouragement given to us, especially Mark Hoffman in his role as the new traction assist group supervisor. We also thank Prof. Vaclav Rajlich, from the department of computer science, for introducing to us the concept of the object-oriented paradigm.

REFERENCES:
class POINT
variables row, column.
methods show(), hide()
move_to(x, y)
setcolor(color)

class BOX
variables height, width
methods:
show()
hide()
fillcolor(color)

class TRIANGLE
variables: height, base
methods:
show()
hide()
fillcolor(color)

class LABEL
variable: text
methods:
show()
hide()

class SCROLLBAR
methods:
getposition(x,y)
setposition(x,y)
moveup(), movedown()
pagup(), pagedown()
show()
hide()

class WINDOW
methods:
show()
hide()

Figure 1: example of a class hierarchy

Figure 2: Overall vehicle model

Figure 3: Object communication diagram
Torque Converter

Figure 4: Mixed-mode simulation

Figure 5a

Figure 5b

Figure 5c

Figure 5d