

**Development of the Capability-Enhanced
PARAMICS Simulation Environment**

Lianyu Chu

California PATH, ATMS Center
Institute of Transportation Studies
University of California, Irvine
Irvine, CA 92697
Tel: 949-824-1876 Fax: 949-824-8385
Email: lchu@translab.its.uci.edu

Henry X. Liu

California PATH, ATMS Center
Institute of Transportation Studies
University of California, Irvine
Irvine, CA 92697
Tel: 949-824-2949 Fax: 949-824-8385
Email: hliu@translab.its.uci.edu

Will Recker

Institute of Transportation Studies
University of California, Irvine
Irvine, CA 92697
Tel: 949-824-5642 Fax: 949-824-8385
Email: wwrecker@uci.edu

Submitted to 2003 TRB Annual Meeting

ABSTRACT

This paper presents our practices on the development of the capability-enhanced PARAMICS simulation environment through API programming. Compared to the commercial PARAMICS model, the capability-enhanced PARAMICS simulation environment complements and enhances many important aspects of PARAMICS simulation, such as path-based routing, actuated signal control, ramp metering control, traffic information collection, database connection and performance measures. This capability-enhanced PARAMICS simulation can thus better model and evaluate ITS.

1 INTRODUCTION

Microscopic traffic simulation is a software tool to model the real-world traffic system, including the road, drivers, and vehicles, in fine details. In the micro-simulation process, the state of an individual vehicle is continuously or discretely calculated and predicted based on vehicle-vehicle interactions. The car-following, lane-changing and gap-acceptance models are the basic elements of a microscopic traffic simulator. Notable instances of micro-simulators include PARAMICS, CORSIM, VISSIM, AIMSUN2, TRANSIM, and MITSIM (1).

With the advancements of computer technology, micro-simulation has become an increasingly popular and effective tool for many applications, which are not amendable to study by other means. For the academic side, one application is to test new traffic models. For the practical side, most applications are the evaluation (and development) of Intelligent Transportation Systems (ITS) because they are difficult to be evaluated in the field operational tests and they need cost-benefit analyses before their implementation (2).

In general, ITS involves the introduction of a variety of advanced technologies, such as information technologies, to the transportation system in order to make the existing road network be efficiently used. Typical ITS applications include adaptive signal control, ramp metering, and dynamic route guidance system, etc. One of their common features is that they need the real-time traffic information, generally collected by detectors or probe vehicles. For a complex ITS strategy, such as Changeable Message Sign (CMS) routing, not only drivers' behaviors and routes but also traffic control facilities need to be controlled. Though current micro-simulators have enabled a lot of ITS features, it is still impossible to implement a complicated ITS strategy by the micro-simulator itself.

The direct way to enhance the capabilities of a micro-simulator is to work on the source codes. Another way is to code complementary or enhanced components through Application Programming Interface (API) programming and then interface them to the simulator. Since the source code is proprietary, API programming is a practical way for users. API programming needs to use provided API library of a micro-simulator, which includes a set of interface functions, through which users can access the core models of

the micro-simulator. Most current popular commercial micro-simulators, including PARAMICS, VISSIM, and AIMSUN 2, provide a series of their API functions for users.

This paper will describe our practices on developing a capability-enhanced PARAMICS simulation environment through API programming. It is organized as follows. First, we will give a brief overview of the micro-simulator, PARAMICS, used for the capability enhancements. We explain how API works in PARAMICS, in what aspects we enhance the capabilities of PARAMICS, and the framework of the capability-enhanced PARAMICS simulation environment. Then, we describe the basic enhancement modules in more details. The advanced modules, developed on top of basic modules, are further introduced. Finally, concluding remarks are presented.

2. FRAMEWORK OF CAPABILITY ENHANCEMENTS

2.1 Overview of PARAMICS

PARAMICS (PARAllel MICrosopic Simulation) is a suite of microscopic simulation tools used to model the movement and behavior of individual vehicles on urban and highway road networks (3). It offers very plausible detailed modeling for many components of the traffic system. Not only the characteristics of drivers, vehicles and the interactions between vehicles but also the network geometry can influence simulation results. PARAMICS is fit to ITS studies due to its high performance, scalability and the ability of modeling the emerging ITS infrastructures, such as loop detectors and VMS. In addition, PARAMICS provides users with API through which users can customize and extend many features of the underlying simulation model without having to deal with the underlying proprietary code. Though PARAMICS can model some simple ITS strategies, API programming is eventually required to implement more complicated ITS strategies.

2.2 How API works in PARAMICS

PARAMICS provides users with an API library that include a set of interface functions, which can be used to access its core models. Basically, PARAMICS provides two groups of interface functions, callback functions and control functions. The callback functions are used for providing information about the attributes of vehicles and their environment. There are two types of control functions, override and overload functions. Override functions are used to replace an internal function of standard simulation loop, such as car-following models. Overload functions are used to add additional functions to the PARAMICS simulation loop.

As shown in Figure 1, the simulation process is like this: after the start of simulation, some basic elements of the simulation, such as the speed and position of vehicles, traffic signals, etc., are updated at every time step. If an API module is involved in the simulation, it may work at every time step, or be triggered at a specific simulation time or by a specific event. In general, an API module gets necessary information from the simulation world through callback function and then affects the simulation through control functions.

2.3 Aspects of PARAMICS need to be complemented and enhanced

Each micro-simulator has its own features to simulate the real-world traffic. Specifically for PARAMICS, many aspects of PARAMICS can be complemented and enhanced through API programming. Our current efforts are limited to the following basic aspects of the software.

2.3.1 Routing

PARAMICS is a link-based simulator. Vehicles being simulated do not carry their whole routes but decide their route based on the routing table stored at each node along its route. These routing tables are pre-calculated based on the currently used traffic assignment method. A path-based routing mechanism is required for the applications of traveler information related ITS strategies, such as dynamic route guidance or CMS routing.

2.3.2 Real-time traffic information collection

The common feature of ITS is that ITS needs the real-time traffic information, generally collected by detectors or probe vehicles, for decision-making. PARAMICS can model loop detectors, the most frequently used sensors in the real world. However, aggregated loop data, which are generally provided by freeway systems and can be used for real-time traffic control, cannot be obtained directly from PARAMICS simulation. Also, the concept of probe vehicles needs PARAMICS to track a certain percentage of probe vehicles and extract travel time or travel speed information from them. This cannot be done without the involvement of API programming.

2.3.3 Signal control

PARAMICS can basically model the fixed-time signal control. Besides, PARAMICS also provide a plan/phase language (i.e. a kind of script language) to simulate some simple actuated signal control logics. However, in the field, the widely used actuated signal controller uses the complex NEMA logic or type-170 logic. Our experiences found this script language is difficult to be used to model these complex control logics and to replicate these logics to multiple signalized intersections.

2.3.4 Ramp metering control

PARAMICS can model fixed-time ramp metering with multiple timing plans. However, a ramp-metering controller, developed in PARAMICS API, is required for the support of development of adaptive ramp metering algorithms, which have more complicated control logics. The ramp-metering controller should provide interface functions that can be used for querying the old metering rate and setting a new metering rate based on the adaptive ramp metering algorithms. When the adaptive ramp-metering algorithm is not activated, the fixed-time metering will be the default control.

2.3.5 Database connection

PARAMICS does not have the capability to connect with a database. The advantage of the use of database is that database can become the medium where API modules can exchange data with outside programs or applications.

2.3.6 Performance measure

An important aspect of the evaluation studies is to use some overall performance measures to evaluate how the implementation of an ITS strategy benefit the whole traffic system, including the freeway and arterial part of the network. PARAMICS has strong abilities on the collection of statistics data. Except some general performance data, PARAMICS can output link-based, trip-based, intersection-based, and detector-based data. With the increase of the size of the network, the number of links, trips, intersections, and detectors increases drastically in PARAMICS. The current difficulties are

- Large amount of data are required to be processed after simulation runs in order to obtain the expected performance measures.
- Some performance measures cannot be extracted from output measurement data
- PARAMICS has a restriction on the number of output files to be opened during simulation under WINDOWS version.

2.4 Framework

The above capability enhancements are focused on the basic aspects of the micro-simulator, PARAMICS. Each of these basic modules refers to an important aspect of simulation. These functionality enhancements can be classified into the following four categories:

1. Basic control modules, including signal, ramp metering and routing
2. Traffic information collection
3. Database connection
4. Performance measures

Figure 2 shows the framework of the capability-enhanced PARAMICS simulation environment. Any an API module in the enhanced PARAMICS environment exchanges dynamic data with the core PARAMICS model and other advanced API modules through the Dynamic Linking Library (DLL) mechanism.

Besides these basic modules, the capability-enhanced PARAMICS environment also includes advanced modules. Examples of advanced modules include ramp metering algorithms, adaptive signal control, and integrated control, which include several ITS components. These advanced modules are developed on top of basic enhancement modules. This hierarchical API development approach, demonstrated in Figure 3, can thus re-use the codes developed in the basic modules.

3. BASIC MODULES

3.1 Basic control modules

3.1.1 Full-actuated Signal Control

This plug-in module implements the eight-phase, dual-ring, concurrent controller logic. The data input to this API is the signal timing plan, the geometry and detector information of each intersection. Interface functions have been provided by this API for external modules to acquire and change the default timing plan. This API provided a couple of interface functions for external API modules to acquire the current signal timing plan and set a new timing plan to a specific signal. An advanced signal control algorithm API can be further developed based on them. The prototypes of these interface functions are shown below (4).

Signal* signal_get_parameters(char *nodeName);

Function: Querying the current signal timing plan of a specific actuated signal

Return Value: The current timing plan of an actuated signal.

Parameters: **nodeName** is the name of the signal node.

Signal is the structure of actuated signal data, whose definition is:

```
type Signal
{
    // intersection name and location
    char *node;
    char *controllerLocation;

    // signal parameters
    int movements [8];
    float maximumGreen [8];
    float minimumGreen [8];
    float extension [8];
    float storedRed [8];
    int recallPhase [2];

    // current phase information
    int currentPhase;
    int expiredTime;
}
```

Void signal_set_parameters(Signal *sig);

Function: Setting a new timing plan to a specific signal.

Return Value: None

Parameters: **sig** stores the new timing plan.

3.1.2 Ramp Metering Control

This plug-in module is designed to model pre-timed ramp metering control on either one-car-per-green basis or n-cars-per-green basis (with $n > 1$). It also supports multiple timing plans, HOV bypass, and the use of ramp detectors for metering control. The data input of this API is a time-of-day ramp control plan and the detector information of each meter. In addition, this API provided a couple of interface functions for external API modules to acquire the current metering rate and set a new metering rate to a specific ramp meter. An advanced ramp-metering algorithm API can be further developed based on these interface functions. The prototypes of them are shown below.

RAMP *ramp_get_parameters (char *rampnode)

Function: Querying the current metering plan of a specific ramp meter.

Return Value: The current metering control plan of an on-ramp signal.

Parameters: **rampnode** is the name of an on-ramp signal node.

RAMP is the structure of ramp control data, whose definition is:

```

type Ramp
{
// on-ramp signal node name and its location
char *rampNode;
char *controllerLocation;
// ramp control types and parameters
int ControlType;
int meteringCycle;
};

```

Where **controlType** is the status (or type) of the ramp metering control, which can be 0 (if RAMP_CLOSURE), 1 (if RAMP_ON with single-entry metering), 2 (if RAMP_ON with platoon metering) and 9 (if RAMP_OFF).

void ramp_set_parameters (RAMP *ramp, Bool staus)

Function: Setting a new metering rate to a specific ramp meter.

Return Value: None

Parameters: **ramp** stores the new metering control data of a specific on-ramp; **status** is a Boolean value. **status** = TRUE means to set a new metering rate based on an external algorithm; **status** = FALSE means to restore the default time-of-day timing plans.

3.1.3 Path-based routing

The path-based routing plug-in module establishes the mechanism for vehicles to follow a given path, which is an essential requirement for the simulation of driver responses to the information supply and the resulting route choice. Only those vehicles that need to follow specific paths will be guided by this API and other vehicles will select route based on the internal routing method of PARAMICS. The interface function used to set a path to a vehicle to follow is as follows:

void uci_vehicle_route_set (void *Vp, VROUTE route)

Parameters: Vp: the pointer of a vehicle, equivalent to vehicle ID
route is the initial address of the whole path
VROUTE is a link list storing whole path the vehicle should follow, which is defined as:

```
type VROUTE
{
// link name along the route
char *linkName;
VROUTE *next;
};
```

3.2 Traffic Information Collection

3.2.1 Loop detector data

In the real world, loop detectors are placed on freeways and arterials for collecting aggregated data at a certain time interval (typically, 30 seconds) for the purposes of traffic analysis and traffic control. These aggregated loop data are stored either in the database or shared memory. Other traffic operation components can get access to these data through data communication networks and use them for generating real-time control strategies.

The loop data aggregator API works as the traffic data collection and provision server in the enhanced PARAMICS environment. It emulates the real-world data collection from inductive loop detectors and broadcasts the latest aggregated loop data to the dynamic memory during simulation. Other API modules can obtain these data in real time through the interface function provided by this API. In addition, this API can report the aggregated loop data to text files or the MYSQL database as performance measures for data analysis and performance comparison.

The interface function of this API can be used for querying the aggregated loop data at a detector station at a certain time interval. The aggregated loop data includes grouped volume, average occupancy and average speed, as well as lane-based volume, average occupancy and average speed.

LOOPAGG loop_agg (char *detectorName)

Return Value: The aggregated detector data of a loop detector

Parameters: **detectorName**: loop detector name

LOOPAGG is a structure that has the following definition:

```
type LOOPAGG
{
int    detectorIndex;
float  AggregationTime;
int    lane;
int    g_vol;
float  g_occ;
float  g_spd;
int    *vol;
float  *occ;
float  *spd;
};
```

where

detectorIndex is the network-wide index for the detector;

Aggregationtime is the time of the latest aggregation, determined by the loop data collection interval;

g_vol is the total traffic counts passing all lanes of a detector station;

g_occ is the average occupancy of all lanes at a detector station;

g_spd is the average speed of all vehicles passing a detector station;

lane is the total number of lanes at the detector station;

**vol*, **occ*, **spd* are pointers for recording values of volume, occupancy and average speed at each lane of a detector station.

3.2.2 Point-to-point travel time data collection

Similarly, we develop a probe vehicle API to simulate point-to-point travel time data collection through GPS equipped vehicles. The sample rate, which is equivalent to the percentage of GPS equipped vehicles, can be specified through control interface. Point-to-point travel time data can be output to the dynamic memory and text files or the MYSQL database.

3.3 Database connection

MYSQL database is currently regarded as the most popular and highly efficient Open Source SQL (Sequential Query Language) database in the world. We developed the interface functions, including a set of simple C routines programmed in MYSQL API, in order to connect PARAMICS with MYSQL database.

MYSQL database can thus be used for storing intermediate simulation data or final simulation results if large amounts of data are generated. In addition, API modules of PARAMICS can also get access the database during the simulation to obtain outside data.

3.4 Performance measures

To be more efficient to evaluation studies, we developed a MOE API for computing, gathering and reporting a number of user-preferred overall performance measures, each of which corresponds to a specific aspect of interest.

3.4.1 Overall performance

MOE #1 system efficiency measure: average system travel time (ASTT) of the whole simulation period. ASTT is calculated as the weighted mean of the average travel times of all OD pairs

$$ASTT = \frac{\sum_{\forall i,j} (T_{i,j} \cdot N_{i,j})}{\sum_{\forall i,j} N_{i,j}} \quad (1)$$

where $N_{i,j}$ is the total number of vehicles that actually traveled from origin i to destination j ; $T_{i,j}$ is the average OD travel time from origin i to destination j ;

MOE #2 system reliability measure: average standard deviation of OD travel times (Stdev_TT) of the entire simulation period. Stdev_TT is calculated as the weighted standard deviation of the average travel times of all OD pairs for the whole study period:

$$Stdev_TT = \frac{\sum_{\forall i,j} (Std(T_{i,j}) \cdot N_{i,j})}{\sum_{\forall i,j} N_{i,j}} \quad (7)$$

where $Std(T_{i,j})$ is the standard deviation of the average OD travel time from origin i to destination j .

3.4.2 Freeway Performance

MOE #3 freeway efficiency measure

- (1) average mainline travel speed of the entire simulation period (AMTS)
- (2) average mainline travel speed during the congestion period (peak_AMTS). The congestion period is defined as the congestion period of the baseline scenario.

MOE #4 on-ramp efficiency measure

- (1) total on-ramp delay (TOD)
- (2) time percentage of the on-ramp queue spillback to the local streets (POQS)

3.4.3 Arterial Performance

MOE #5 arterial efficiency measure

- (1) average travel time from the upstream end to the downstream end of an arterial (ATT)
- (2) the standard deviation of ATT (std_ATT)

4. ADVANCED MODULES

4.1 How to develop advanced modules

An advanced module, such as a signal coordination strategy, adaptive ramp metering algorithm, or an integrated control strategy, generally involves the control of one or

several control components, including signals, ramps, VMS, and routing vehicles, based on traffic information. As introduced in Section 2.4, the capability-enhanced PARAMICS environment uses a hierarchical approach to develop advanced modules. In other words, advanced modules are developed based on several basic modules, which can be accessed by interface functions provided by basic modules.

As an example, Figure 4 describes how an advanced ramp-metering algorithm is developed. The advanced API is built on top of two basic plug-in modules, i.e., ramp metering controller and loop data aggregator. The entrance ramp signals in the simulation network are controlled by the ramp metering API, through which metering rates can be queried and set by other API modules. The loop data aggregator API emulates the real-world loop data collection, typically with a thirty-second interval, and broadcast the latest loop data to the dynamic memory. The advanced API can access the dynamic memory and obtains the required loop data through interface functions provided by the loop data aggregation API. Then the metering rate for the next control interval is calculated based on the advanced ramp-metering algorithm. The new metering rate is sent back to the ramp controller API for implementation (5).

4.2 Current Developed Advanced Modules

The advanced modules we have developed include two signal control algorithms based on the actuated signal control system and several ramp-metering algorithms. The two signal control algorithms are actuated signal coordination and adaptive signal control based on real-time delay estimation technology. The ramp-metering algorithms include the traditional demand-capacity and occupancy control strategies, ALINEA, a local feedback ramp metering control proposed by Papageorgiou (6) and three coordinated ramp metering algorithms, including ZONE (7), a metering algorithm of Minnesota, BOTTLENECK (8) of Washington, and SWARM (9) of California.

5. CONCLUDING REMARKS

This paper presents our practices on the development of a capability-enhanced PARAMICS simulation environment through integrating some plug-in modules implemented in PARAMICS API. These plug-in modules complement and enhance the functionalities of the commercial PARAMICS model. As a result, the capability-enhanced PARAMICS simulation can better model and evaluate ITS.

Our experiences show that API can be used to access the core models of a micro-simulator and potentially, researchers can use commercial micro-simulators as a shell for testing their own models and algorithms. Since other commercial micro-simulators, such as VISSIM and AIMSUN 2, also provide users with their own API functions, users can replicate our methods to enhance their capabilities.

Our current capability enhancements of PARAMICS only cover some aspects of the micro-simulator, more efforts are expected in order to make the enhanced PARAMICS better fit to our ITS-related studies.

REFERENCES

1. Jayakrishnan, R., Oh, J., and Sahraoui A. (2001) *Calibration and Path Dynamics Issues in Microscopic simulation For Advanced Traffic Management and Information Systems*, 80th Transportation Research Board Annual Meeting, Washington D.C.
2. Algers, S., Bernauer, E., Boero, M., Breheret, L., di Taranto, C., Dougherty, M., Fox, K., and J. Gabard (1998) *Smartest Project deliverable D3*. University of Leeds.
3. Smith M., S. Druitt, G. Cameron and D. MacArthur, *Paramics final Report*, Technical Report EPCC-PARAMICS-FINAL, University of Edinburgh, July, 1994.
4. Liu, X., Chu, L., and Recker, W. (2001) *PARAMICS API Design Document for Actuated Signal, Signal Coordination and Ramp Control*, California PATH Working Paper, UCB-ITS-PWP-2001-11, University of California at Berkeley, 2001.
5. Chu, L., Liu X., Recker, W., Zhang, H.M. (2002) *Performance Evaluating of Adaptive Ramp Metering Algorithms*, accepted by ASCE Journal of Transportation Engineering.
6. Papageorgiou, M., Hadj Salem, H., and Blosseville, J. M. (1991). "ALINEA: A Local Feedback Control Law for On-Ramp Metering." *Transportation Research Record*, 1320, 58-64.
7. Lau R. (1997). *Ramp Metering by Zone – The Minnesota Algorithm*, Minnesota Department of Transportation.
8. Jacobsen, L., Henry, K., and Mahyar, O. (1989). "Real-Time Metering Algorithm for Centralized Control." *Transportation Research Record*, 1232, 17-26.
9. Paesani, G., Kerr, J., Perovich, P., and Khosravi, E. (1997). "System Wide Adaptive Ramp Metering In Southern California." *Conference Proceeding of the 7th Annual Meeting, ITS America*.

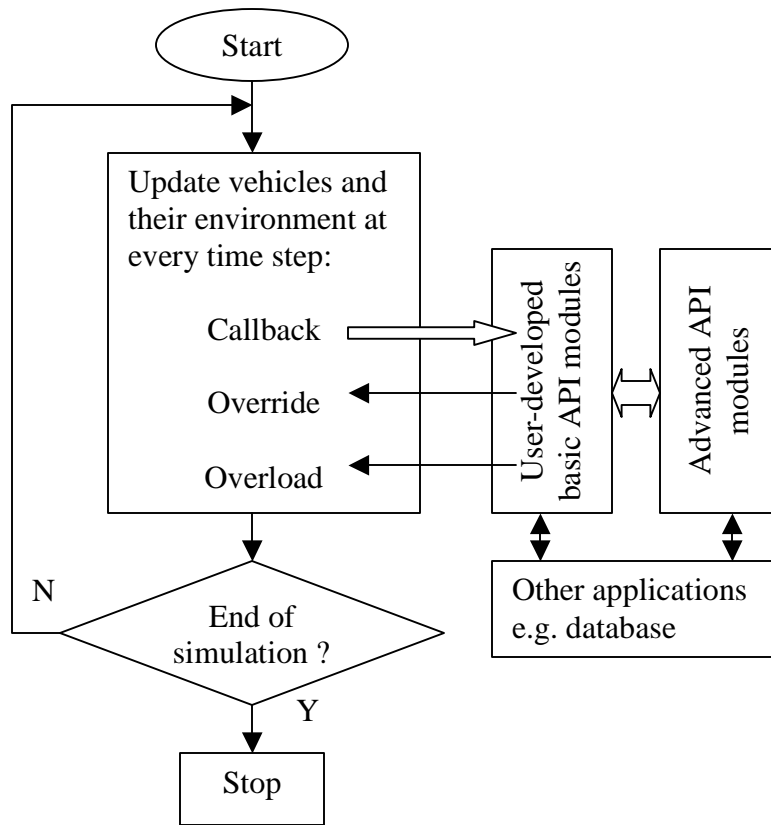


Figure 1 The PARAMICS simulation process with API modules

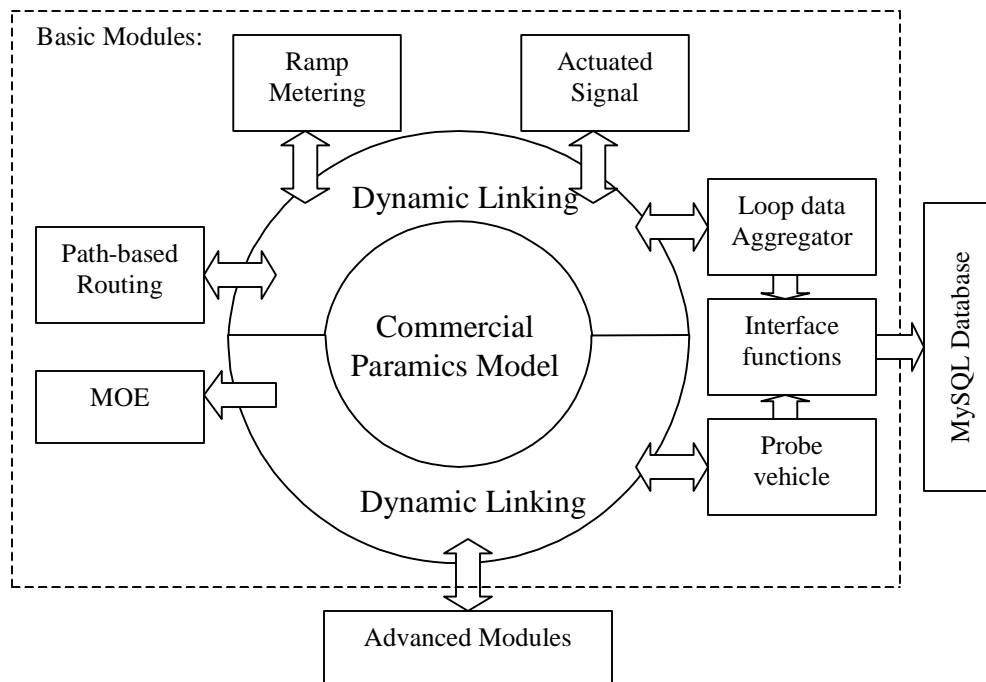


Figure 2 Framework of the capability-enhanced PARAMICS simulation

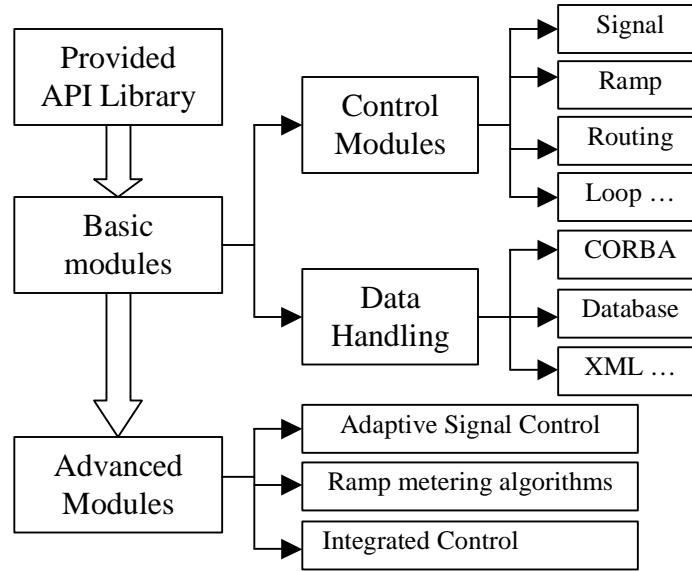


Figure 3. The hierarchical API development approach

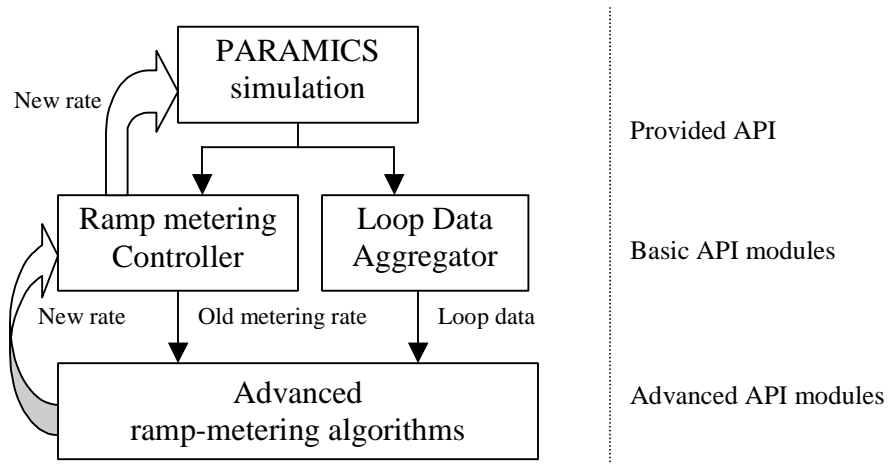


Figure 4. The hierarchical approach to develop advanced ramp metering module