Continuing progress in car navigation systems is rapidly increasing the cost and time needed to develop the associated software. To ease this problem, we have developed a car navigation software development framework called Victoria that substantially improves software productivity. The effectiveness of this framework has been confirmed and is described here together with an overview of Victoria.

The Victoria Approach

The cost and time required for software development, including the extension of specifications when a new model is created and the cost and time needed to make modifications, are expressed by the following equation.

\[ \text{Development cost and time} = \text{cost and time required to clarify specifications} + \frac{\text{the amount of modification work}}{\text{work done per unit time}} \]

The following approach has been devised to reduce software development cost and time.

**Reduction of Cost and Time Required to Clarify Specifications.** The process of making ambiguous specification requirements, including the specifications demanded by the customer and others, into rigorously defined specifications is a major and vitally important undertaking. A human-machine prototyping tool was developed and implemented to make it easy to determine the specifications of ill-defined user interfaces, which are responsible for the most specification changes in the entire system.

**Reduction of Modification Work.** The software was redesigned to localize modifications resulting from specification changes and to reduce the amount of modification work required. This involved separating the elements dependent on the human-machine interface specifications, the database-dependent elements and the platform-dependent elements. Additionally, an object-oriented language was used to develop the framework in order to localize the places requiring modification and improve extensibility.

**Improvement of Work Done per Unit Time.** The conventional development environment for embedded systems is referred to as a cross-development environment in which a host computer and the target hardware are connected to carry out the development work. The newly developed framework has a middleware layer that absorbs differences between the development environment on the target hardware and the development environment on a PC. This facilitates the sharing of higher level car-navigation software and enables the development work to be done on a PC.

An automatic software generator based on a human-machine builder was also developed to boost the development efficiency of the user interface element, which requires the largest number of modifications on each model.

**Object-Oriented Development**

C++ was adopted as the object-oriented language, and the software architecture was restructured to minimize the amount of software modifications needed to effect specification changes. This was done to reduce linkage between modules and to localize modifications.

**Development in a PC Environment**

A middleware layer is provided to absorb development environment differences between the target hardware and a PC, allowing higher level car-navigation software to be shared.

The higher level software does not issue a system call directly to the operating system (OS) but rather through the middleware. The middleware for the target hardware environment and that for the PC environment are interchangeable, facilitating software development on a PC.

Developing the software on a PC makes it possible to utilize the outstanding development environment available for PCs. Because the entire process from software development to system testing and improvement can be done on the same PC, the debugging cycle can be shortened.

Fig. 1 shows an example of the development environment on a PC. Operations can be executed via input from keyboard or mouse. A dia-
log box is provided to reproduce sensor data, such as the signals of the global positioning system (GPS), gyro and vehicle-speed pulses, and vehicle information and communication system (VICS) data. It is also possible to simulate driving operations and reception of VICS data. At the final stage of development, the middleware can be substituted for that of the target hardware environment, as shown in Fig. 2, and a cross-compiler can be developed in the target hardware environment. Microsoft’s Windows NT 4.0 has been adopted as the OS of the PC environment and Microsoft’s Windows CE as the OS of the target hardware environment.

Separation of Map Database-Dependent Part by Using a Map API

Map databases for car navigation systems have yet to be unified, and each company currently uses its own map format. Previously, we devel-
oped car-navigation software separately for each company’s format, as shown in Fig. 3. To avoid duplication of development work for each format, we separated the map database-dependent part as a map-access library and specified a map application program interface (API) as the programming interface for map access.

As shown in Fig. 4, the higher level software accesses maps in each format via the map API, making the software independent of the map format and allowing common usage.

**Software Development Using a Human-Machine Builder**

The elements dependent on the human-machine specifications were separated and a prototyping tool was adopted to support specification decisions and a human-machine builder was applied.

Fig. 3 Conventional access to map database

![Fig. 3 Conventional access to map database](image)

Fig. 4 Access to map database through map API

![Fig. 4 Access to map database through map API](image)
to the automatic software generator for the user interface element. The human-machine builder makes it easy to modify the areas of the software that have to be revised. It also minimizes duplicated effort because consistent support is provided from the upstream design process where the human-machine specifications are examined.

As shown in Fig. 5, the map screens of a car navigation system are created with a screen editor and the operational transitions and actions are described in state transition diagrams. Thanks to the use of an automatic code generator, the screen parts and actions thus created become the source code that can be run as it is on the PC environment and the target hardware environment. The entire cycle from confirmation of an action to its modification can be executed on the same PC. The screen editor is a tool that we have newly developed for creating car navigation system screens and can be used in the same way as an ordinary drawing tool. A commercially available tool is used to describe the state transition diagrams.

The new software support framework promises to increase the speed and flexibility with which Mitsubishi Electric can accommodate the rapidly increasing complexity and sophistication of car navigation systems.

Fig. 5 Development user-interface element using the human-machine builder