Software Modelling and Automatic Code Generation Based on Reactive State Diagram

M.C. Qu
Department of Aerospace Engineering
School of Computer Science and Technology
Harbin Institute of Technology
China

L.J. Meng, X.H. Wu
School of Computer Science and Technology
Harbin Institute of Technology
China

N.G. Cui
Department of Aerospace Engineering
Harbin Institute of Technology
China

***Abstract*—Based on the UML state diagram it can model the reactive system which is event-driven exactly and elaborately. Building a system model from the top level using the UML state diagram model has many advantages, such as reusability, maintainability, interactively, etc. Although there are some mainstream commercial modelling tools supporting code generation from diagram model, most of the generated code frameworks are OO (object-oriented) and could not satisfy the resource constraint in the real-time embedded system [1]. In the embedded system software, there are lots of reusable public service codes, we design and realize a real-time framework using the reusable code. By implementing an operating system encapsulated layer based on the real-time framework, we achieve the objective of developing application software independent of the target platform, and making the generated code be capable of cross-platform capability. Based on the real-time framework and the characteristic of state diagram, we designs and implements an algorithm for generating code automatically and a real-time framework of state machine [2], which are all verified by the engineering project.**

Keywords- state diagram; software modelling; generating code automatically

I. INTRODUCTION

In 1997 OMG(Object Management Group) released the UML, there are many software modelling tools such as Microsoft's Visio, IBM's Rational Rose and Rational Rhapsody, etc.

Now days existing many kinds of diagram modelling software of UML, which have large function, complex interfaces and are specific to the OO development process almost all of them satisfy the whole standard of UML. Our development of embedded software is more specific to procedure-oriented [3]. The tools existing could not satisfy our requirements, so, developing a diagram modelling software which is fit for resource constrain embedded software has practical significance.

This paper designs and implements a diagram modelling tool, which bases on the standard of UML state diagram, applying the plug-ins technology of Eclipse and GEF. The tool realizes some functions such as modelling, item management, document management, project wizard, exporting mages, etc. It uses the patterns of MVC and the given algorithm exporting the XML models for generating code automatically. In this article, firstly describing the relevant technologies, secondly designing the modelling tool, then providing the key part, lastly giving the test and conclusion.

II. STATE MACHINE REAL-TIME FRAMEWORK

The RTF (real time framework) is regarded as a middleware, existing between user application and target platform [4]. Figure. 1. shows the overall hierarchical structure. The RTF shield the difference of different target platform, providing the unified OO interfaces. It is not need to consider the difference of the platform when analysing user and designing the application, but concentrate on the design and implement of the software's function. The application's objects and events will be mapped to the relevant part of the RTF, which will be used to access resource and call relevant operation [5].

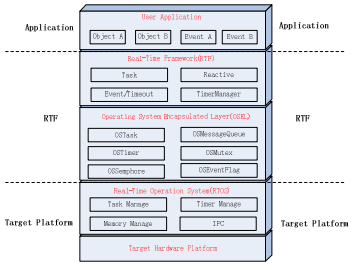

FIGURE I. .HIERARCHY OF RTF.

RTF uses Operation System Encapsulated Layer (OSEL) to call on the resources and services of the target system. Many other cross-platform software used this method too. In Figure. 2. OSEL includes two layers.

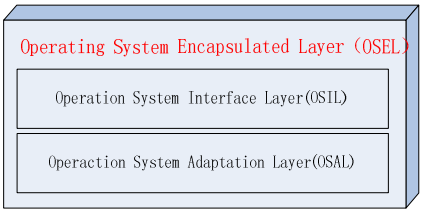

FIGURE II. TWO-TIER STRUCTURE OF OSEL.

(1) Operation System Interface Layer (OSIL): OSIL provides the unified interfaces for the upper layer of RTF and the RTOS application. Adopting these interfaces, it is not need to consider the differences of the operation system. For example, adopting function OSTaskCreate() to create a new task, adopting function OSTaskDestory() to destroy an active task. The function's implementation is related to the operation system.

(2) Operation System Adaptation Layer (OSAL): the interfaces the OSIL provides is implemented by the OSAL. Designing different adaptation for the specific operation system. Such as in the VxWorks, the implementation of the function OSTaskCreate() calls for the VxWorks' system call taskSpawn(). So the differences of the system's basement are shielded by the adaptation layer.

The layer upper OSEL is the key part of the RTF, including task management module, event management module, timeout management module, etc. RTF is designed and implemented by the object-oriented approach, so these modules are classes, cooperate to achieve functional [6].

To meet the needs of developing different kinds of real-time software, RTF provides two methods for Allocating memory, dynamic allocation and static allocation. The C language function malloc() and free() are used for dynamic allocation and RTF uses static memory pool to manage the static memory allocation.

III. BASED ON STATE DIAGRAM TO GENERATE CODE

Almost all the real-time software are responsive, that is event-driven. So the state diagram is the best choice for describing the dynamic characteristic of the instance. All the users who used the state diagram modelling inherits the RTF frame of responsive class Reactive. Task class receive the event, then call on the class Reactive. The event call on method dispatchEvent(void* me, Event* ev) to handle the event. Figure. 3. Shows a simple state modelling of object Car.

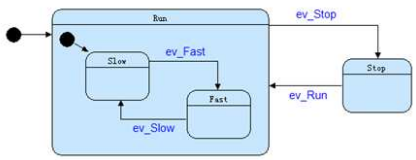

FIGURE III. SIMPLE STATE MODELING OF OBJECT CAR.

Every node in the state machine has the only one numerical symbol. System gives every users an enum to restore the message. The state machine is stored as a trees tat. A program of preorder traversal will get all the nodes' message, then store in the enum. For example the object car:

```
Type def enum Car_States_Enum {
   Car_rootState = 0,
   Car_Run = 1,
   Car_Slow = 2,
   Car_Fast = 3,
   Car_Stop = 4
}Car_States_Enum;
```

Every node in the state machine has a state handler indicates that the reactive movement when the event happens. The function has two parameters, the first is a pointer points to the object, the other is a pointer points to the event.

```
Type def UINT32 (*stateHandler) (void* me, const Event* ev);
```

Every function bases on the event ID to choose the related branch, the event including user event and predefine event. The predefine event includes entry, exit, init, null. According to the action, reaction, transition to choose the module, there are if-else and guard condition in some switch branches.

For distributing the event of the object, there is an UINT32 member activeState in the object struct to state the object's state id which is changed all the time. The activeState is the bottom active id. Every state has a function to tell whether the state is active. The proof is to tell the activeState whether is its or not.

```
Type def struct Car{
   . . .
UINT32 activeState;          /* current active state id */
}Car;
Boolean Run_isActive(Car* me){    /* Run state active */
  if(me->activeState == Car_Run)
    return TRUE;
  if(Slow_isActive())
        return TRUE;
  if(Fast_isActive())
        return TRUE;
  return FALSE;
}
```

The function dispathEvent() handles the appointed event, an object may has many dispathEvent(), and a container state has one dispathEvent(). The object's state is regarded as an rootState, the function is rootState_dispathEvent(). The whole object's event distributing is accomplished by disoathEvent(). The process of dispathEvent is shown in Figure. 4.:

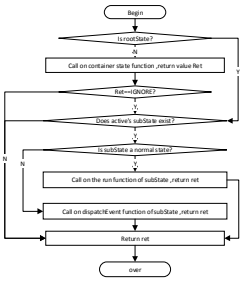

FIGURE IV. PROCESS OF DISPATCHEVENT().

When rootState_dispatchEvent return the value TRAN presents the state will transfer. If the target state is a container state or there is a null transfer, the state will be transfer again. The object's function taskEvent() get the rootState_dispatchEvent()'s return value, using init and null event to drive the state machine until the object turn in a stable state.

When the object's state has changed, all the state will be executed on the path. For example in Figure. 5. , the movement sequence is: firstly, executing state B_exit, secondly, executing A_exit, thirdly, executing tran(), then, executing C_entry(), at last, executing D_entry().

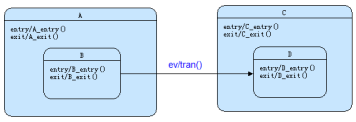

FIGURE V. A SIMPLE STATE TRANSITION.

Because all the state machine's state node is stored as a treestat, the compute is begin from the source state S transfer to target state T, and the action sequence is: firstly, call on the exitActions to save all the state, then, call on the entryActions to store the entry state. The key part of the process is find the first ancestral state node P of the source state node S and target state node T. Along the path from P to S, putting all the pointer of exit movement to the array exitActions. Then along the path form T to P, putting all the pointer of entry movement to the array entryActions. At last, calling on the function in array exitActions is sequence to transfer, and then, calling on the function in array entryActions is revered order. The whole process is shown in Figure. 6. :

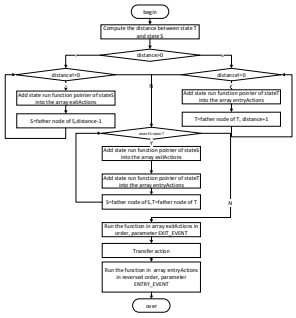

FIGURE VI. THE CALCULATION AND IMPLEMENTATION PROCESS OF THE ACTION SEQUENCE

IV. CONCLUSIONS

Based on RTF to realize the software transportation and reusing. Through packaging the difference of the real time system, user could not consider the difference of the target platform, but to concrete on the design of the function and action. RTF provides many date structure and code for embedded application. This paper implements the RTF and the state diagram algorithm for generating code automatically. Verifying them in the engineering project application.

ACKNOWLEDGEMENTS

Supported by the National Science Foundation of China under Grant No.61402131; the China postdoctoral science foundation under Grant No.2014M551245; the Heilongjiang postdoctoral science foundation under Grant No.LBH-Z13105.

REFERENCE

[1] Malinowski A, Yu H. Comparison of embedded system design for industrial applications[J]. Industrial Informatics, IEEE Transactions on, 7(2), pp. 244-254, 2011.

[2] Cagkan Erbas. A framework for system-level modeling and simulation of embedded systems architectures[J]. EURASIP Journal on Embedded Systems, (1), pp. 2-2, 2007.

[3] Kopetz H. The complexity challenge in embedded system design[C]//Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. IEEE, pp. 3-12, 2008.

[4] Sebastien Gerard, Jean-Philippe Babau,Joel Champeau. Model Driven Engineering for Distributed Real-Time Embedded Systems M]. Wiley-IEEE, pp. 12-15, 2010.

[5] Abdoulaye Gamatié, Éric Piel. A Model-Driven Design Framework for Massively Parallel Embedded Systems[J]. ACM Transactions on Embedded Computing Systems (TECS), 10(4), pp. 300-302, 2011.

[6] Ali Fouad, Keith Phalp, John Mathenge Kanyaru, Sheridan Jeary. Embedding requirements within Model-Driven Architecture[J]. Software quality journal, (19), pp. 411-415, 2011.