ABSTRACT

When developing embedded systems, the heterogeneity and compositionality of functions and architecture of the systems must be described during design stage. UML, describing static structures and dynamic behaviors of a system from multi viewpoints, can naturally specify such characteristics. This paper proposes an extension method of UML for developing embedded systems, and it is extended on the aspects of real-time, concurrency and distribution properties. It realizes the usage of UML in the domain of network communication protocol. Finally the extended UML is successfully applied to the practice of developing FF HSE communication protocol stack software.

INTRODUCTION

As Grant says (2002) Embedded system now has been developed to multi-function integrated system involving computing, audio and video signature processing, wired and wireless communication and other domains. Modeling such systems need a set of necessary notations for expressing the logical heterogeneous characteristics of systems such as continuous time, finite-state-machine, dataflow, discrete event, reactive, etc. Moreover the functions of embedded system are realized by combination of multi software and hardware subsystems, and this physical heterogeneous integration must also be presented. Therefore, models of embedded systems must be able to describe the heterogeneous and compositional characteristics of functions and architecture of the system.

Unified Modeling Language (UML) provides a general-purpose notation for software system description and documentation. It has several static and dynamic diagrams that describe one system from different viewpoints. UML has become defector standard for expressing OO software and arrested engineers' more and more attentions in the embedded system domain. However, as current version of UML specification lacks expression and verification of real-time property, and UML itself is not a formal language, it needs necessary extensions for real-time and formalization when used to design embedded systems.

Many academe researchers and tool developers have proposed a lot of real-time solutions and formal extensions of UML. Academe usually focuses on UML formalization, checking models on the basis of UML formal semantics. For examples, Ivan (2003) transformed UML models to Z language, Jiang (2002) provided formal semantics for statechart, and Subash (2002) used Temporal Logic for formal extension of real-time semantics. While tool developers generally focus on notations of real-time properties and automatic technology, such as UML-RT of Rational adding capsule, port, connector and protocol for modeling structure and behavior of complex real-time systems, Rhapsody of I-Logic providing strong functions for statechart implementation, TAU suite of TeleLogic combining UML and SDL, and Software Real-Time Studio of Artisan proposing new diagrams to describing embedded real-time properties.

As mentioned above, so many UML extension methods are there, but there still exist a lot of shortages. For examples, the academic researches are so theoretic that little are really applied in the engineering practices, however, UML tools still have comparatively weak capabilities of model checking and implementation.

So this paper combines researches of academe and tool developer, and proposes an extension method for modeling embedded real-time systems from viewpoint of engineering. It first defines some notations in sequence diagram for describing system temporal properties for real-time extension, defines two stereotypes for describing system tasks and the communication mechanism between tasks and give precise semantics of association relationships for concurrency extension, and specifies some constraints in sequence diagram for distribution extension. These extensions achieve usage of UML in the special domain of communication protocol.

On the basis of extensions above, the paper also refines a set of rules for mapping UML models to program codes for guiding manu-programming, which realizes the integration of design and implementation tightly.

The extended UML is successfully applied to the practice of developing Fieldbus Foundation High Speed
Ethernet communication protocol stack software, which reduces development cycle and improves quality of the software when compared to the common development method.

**UML EXTENSIONS**

The network communication protocol stack software in this paper is referred to embedded real-time application program developed above real-time operating system (RTOS). According to the characteristics of the network communication, UML should be extended on aspects of real-time and concurrency and distribution.

**Real-Time Extension**

Network communication has tow typical applications: industrial control system and Internet, and both applications need real-time requirements. Therefore, when modeling network communication through UML, it is on the aspect of real-time properties that the UML must be extended first. This paper achieves real-time extension in sequence diagram and statechart diagram.

Among the existing methods of real-time extension, some abstract timeliness characteristics needed for describing embedded system, such as start time, transmission time, execution time, delay and so on. Whatever timeliness characteristics, when expressing or analyzing performance, may all be converted to the expression of absolute time and relative time (or temporal interval). Current version of UML specification lacks expression of these timely concepts.

UML sequence diagram is used to describe message sequences among system objects, and the object lifeline is relative to timeliness. So the temporal requirements are described in sequence diagram. Then we should extend the sequence diagram for expressing the two above timely concepts.

For absolute time, since RTOS provides function to obtain absolute time, namely systematic time or global time, it need only define a notation of global time to denote the absolute time. We define the notation "@" with form of "@ absolute time", for example, "@18s" means at the absolute time 18s. The absolute time is annotated with a line linked to the object lifeline.

Relative time constraints are main temporal constraints of embedded systems. This paper uses timer concept for describing these temporal constraints and defines notations of timer creation, expiration, deletion and restart. These notations are annotated on object lifeline. In addition, as almost most embedded systems have state machines, different results of timer may lead to different state transformation of the object, we also add object state notations in Sequence diagram.

RTOS provides independent timer management, that is to say, once the timer is created, it is the operating system that supervises whether the timer expires, while users only need to offer functions for timer expiration.

So it is specified in this paper that in statechart diagram, temporal requirements (restrictions) are converted to signal-based events. When proper time arrives, the system will send a timeout event to object's state machine, and the state machine handles the timeout event as other normal events.

In order to satisfy conformance between sequence diagram and statechart diagram, the different states of objects are annotated on the object lifeline in sequence diagram, and the message sequence below state denotes what message can be transferred under this state. In addition, as almost most embedded systems have state machines, different results of timer may lead to state transformation. Therefore, it is defined notations of creation, expiration, deletion and restart of timers in this paper. This part will be studied deeper in future research.

**Concurrency Extension**

The main contribution of this paper is concurrency extension of UML. Network communication protocol is real-time system. A property of real-time system is that multi control threads (tasks) execute simultaneously. That is the concurrency property of real-time system. Although in UML, an active class is corresponding to a thread, class is not sufficient to present the independency of tasks as modules. Input of class is realized through invoking entering function of class operation, while communication between tasks should be data exchanging instead of invoking functions. So two stereotypes are defined for concurrency extension of UML in this paper: <<Task>> and <<Gate>>.

As basic components, <<Task>>, which may contain state machine, is mapped directly to task (thread) of program when implemented. The tasks of embedded real-time system are usually preemptive, thus they have priorities. Gate is the only communication interface between basic modules in program. A module may own at most one gate, and the communication between tasks can only through gates, which conforms the component methodology. The attributes of <<Task>> are defined as Table 1.

If necessary, <<Task>> can also be defined other timeliness related attributes such as start time, execute time, deadline etc., therefore schedule analysis can be conducted during model stage. However, the purpose of this paper is to model FF HSE protocol stack, which need not analyze these timeliness properties, therefore these attributes are not defined.

<<Gate>> is implemented to communication mechanism such as message queue, pipe, buffer, semaphore provided by RTOS depending on the value of ComMech. Since RTOS provides management of these communication mechanisms, users only need to use APIs.
instead of considering problems of resource sharing and criticalness. Gate is specified only to answer for receiving input data and does not know content of the receiving data, which handles and simplifies the communication relationship between tasks on the whole. Content of data is represented and analyzed concretely in sequence diagram and statechart diagram. The attributes of <<Gate>> are defined as Table 1 and the structures are shown in figure 1. Output of task is not separately defined in this paper, and users only need to invoke the APIs for outputting data. In class diagram, tasks are stereotyped by <<Task>> and as basic modules compose the whole software system.

Additionally, association relationships between tasks are given precise semantics in this paper, which reflect the relationships between tasks, for example, the executions of tasks are ordinal or concurrently, the communication of tasks is synchronous or asynchronous, a task creates or deletes another task. If there are data transmissions of tasks, there will exist arrows on the association relationships. The arrows are the directions of data transmission, and bi-directional arrow denotes that the two tasks inter-exchange data with each other. The precise semantics of association relationship are defined as follows:

- **Sequence**: Tasks execute in sequence. As (T1 Sequence T2) shows that T1 triggers T2 and terminates before T2 starting, T1 may deliver data to T2. If there is no data transmission, it is presented by the notation of "<Creation->" in class diagram, if there is, it is presented by the notation of "<Creation->", arrow points to the triggered task.

- **Synchro**: Communication between tasks is synchronized. As (T1 Synchro T2) shows that T1 must wait for response from T2 before continuing. Synchro association must have ability of bi-directional communication. Synchronized communication between task may mono- or bi-directional. mono-directional relationship (T1 Synchro T2) depicts that only T1 waits for T2 not T2 for T1, and is presented by the notation of "<−Synchro −>", bi-directional relationship (T1 Synchro2 T2) depicts that the two tasks are both synchronized, and is presented by the notation of "<−Synchro−>".

- **Asynchro**: Communication between tasks is asynchronous. As (T1 Asynchro T2) shows that T1 can execute other actions without waiting for response from T2. It is specified that all data transmissions except synchronized ones are all Asynchro association relationships. Asynchronized communication between task may mono- or bi-directional. mono-directional relationship (T1 Asynchro T2) is presented by the notation of "<−Asynchro→>", bi-directional relationship (T1 Asynchro T2) is presented by the notation of "<−Asynchro−>".

- **Creation**: A task creates another task. As (T1 Creation T2) shows that T1 creates T2. T1 may deliver data to T2. If there is no data transmission, it is presented by the notation of "<−Creation−>", if there is, it is presented by the notation of "<−Creation−>", arrow points to the created task.

- **Deletion**: A task deletes another task that has been created before. As (T1 deletion T2) shows that T1 deletes T2 and is presented by the notation of "<Deletion−>".

Associations with data transmission relationships are realized on gate of communication interfaces. If there is no relationship between tasks, there is no communication nor creation/deletion relationship between them. In addition, creation relationship reflects creation of tasks dynamically, and through the stackSize attribute of <<Task>>, it can calculate the number of tasks that can be created dynamically simultaneously according to the memory resource.

**Distribution Extension**

In network communication protocol, nodes must be installed at different position of the network, and act different roles during communication. This is the distribution property of the system. In this paper it is specified that objects in sequence diagram demonstrate the layered architecture of communication protocol including peer entries (such as user layer, application layer and transport layer) and different roles (such as Client/Server, Publisher/Subscriber). Messages present the data transmission from top to bottom and bottom to top inside nodes and over networks.

**RULES FOR MAPPING UML MODELS**

According to the extensions in this paper, some rules are refined for mapping UML models to program codes for guiding manu-programming. The rules are:

- <<Task>> is mapped to the task of real-time system. All systems have a main task as interface between RTOS and user program, in which other tasks are created. The priority and size of stack are determined by the two attributes.

- <<Gate>> is mapped to communication mechanism such as message queue, pipe, buffer and semaphore depending on the attribute of ComMech, and the size of it is determined by ComSize. Gate must be created at the same time as the task owning it in the main task. Sending and receiving data are realized through invoking APIs.

- The associations between tasks reflect relationships of execution orders, communication, and creation/deletion relationships. When a task creates
another task, the stack will be assigned at the same
time, so will the communication mechanism if
necessary. The same problems are encountered when
a task deletes another task.

**FF HSE COMMUNICATION PROTOCOL**

FF HSE specification (2000) adopts the previous H1
specification referred to the ISO OSI 7-layer model. From
physical layer to transport layer there are well-known
standard Ethernet (IEEE 802.3) and TCP/IP protocols,
while application layer is specified by FF, and above it
there exists an additional layer called user layer designed
for control capability in field devices. Session layer and
presentation layer are not used in Fieldbus. Except user
layer and physical layer, the middle layers are called
"protocol stack" as a whole in the FF specifications.

According to the functions accomplished in physical
devices, HSE communication system is composed of three
components: Communication Entity (CE), System
Management Kernel (SMK) and Fieldbus Application
Process (AP). Communication Entity produces services
messages and provides message transmission; SMK
integrates devices on an HSE network into a coherent
communication system; Application Process has the actual
control capability and is presented as Function Blocks.

Figure 2 depicts the layered architecture of FF HSE
protocol stack. Bolder frame denotes Communication
Entity in which the shadowed part denotes HSE particular
properties and non-shadowed part denotes standard
TCP/IP protocol; dashed frame denotes SMK and AP that
are users of CE. This figure shows the relationships
among CE, SMK and APs.

**DEVELOPMENT OF FF HSE PROTOCOL STACK
SOFTWARE**

The main development task is to program the
protocol stack software. Actually, the protocol stack
software is referred to software developed on application
layer and uses standard socket APIs provided by RTOS
for transporting user data over Ethernet.

**Composite Modules Of Stack Software**

According to FF HSE specification, the protocol
stack software is divided into three modules that are
mapped to tasks when implemented. The modules are:
FDA Agent, System Management (SM) and HSE network
Management Agent (HMA). FDA Agent is composed of
two sublayers: Fieldbus Message Specification (FMS)
sublayer and Fieldbus Device Access (FDA) sublayer.
Functions of The modules are shown in Table 2.
Application Process is not a part of the protocol stack, but
for purpose of convenience, it is presented as a user of the
stack when system is designed. The modules and their
association relationships are presented in class diagram
and shown as figure 3.

Other models of the protocol stack in sequence
diagram and statechart diagram are not shown in this
paper for the reason of space.

**CONCLUSION**

This paper proposes an extension method of UML for
usage in the domain of network communication protocol.
And the method has been applied in development of FF
HSE protocol stack software and achieved good results of
shortening lifecycle, reducing cost and improving quality
of the software. The stack software has already been
adapted in FF HSE I/O devices and HSE/H1 network
gateway and passed the consistency test from Fieldbus
Foundation.

**REFERENCES**

Grant, M., 2002, “UML for Embedded System
Specification and Design: Motivation and Overview”,
Proceeding of the 2002 Design, Automation and Test in
Europe Conference and Exhibition.

Ivan P., “Modeling and Analyzing Software Behavior
in UML”, Doctoral thesis, 2003.11, Department of
Computer Science Åbo Akademi University, Finland

Machine”, 2002 Journal of Software, Vol.13, No.12,
2244-2249

UML”, ICFEM 2002, LNCS 2495, 573-577

Apvrille, L., et al, “A New UML Profile for Real-
Time System Formal Design and Validation”, UML 2001,
LNCS 2185, pp. 287-301

Fieldbus Foundation™ High Speed Ethernet Program
Final Specifications, FS 1.1, 2000

Fieldbus Foundation™ Technical Overview, FD-403
Rev 5.0
Table 1: Definition of stereotypes of <<task>> and <<gate>>

<table>
<thead>
<tr>
<th>S*</th>
<th>Base Class</th>
<th>Tags</th>
<th>C*</th>
<th>Type of Tags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>Class</td>
<td>Entry</td>
<td>None</td>
<td>String</td>
<td>Entry function of the task</td>
</tr>
<tr>
<td></td>
<td>StackSize</td>
<td>None</td>
<td>Integer</td>
<td>Size of task's stack, and the unit is byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority</td>
<td>None</td>
<td>Integer</td>
<td>Priority of task, the smaller, the higher</td>
<td></td>
</tr>
<tr>
<td>gate</td>
<td>Class</td>
<td>ComMech</td>
<td>None</td>
<td>Enum:{Queue, Pipe, Buffer}</td>
<td>Communication mechanism</td>
</tr>
<tr>
<td></td>
<td>ComSize</td>
<td>None</td>
<td>Integer</td>
<td>Size of Communication mechanism</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Description of components of stack software

<table>
<thead>
<tr>
<th>Module Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM</td>
<td>The activity that integrates devices on an HSE network into a coherent communication system and maintains the systemic information of devices.</td>
</tr>
<tr>
<td>FDA Agent</td>
<td>FMS describes the communication services, message formats, and protocol behavior needed to build messages for the User Application.</td>
</tr>
<tr>
<td>FDA</td>
<td>FDA transmits all user data including SM and FMS service primitives. All primitives are converted to unified FDA messages.</td>
</tr>
<tr>
<td>HMA</td>
<td>HMA delivers FDA messages among devices and realizes interfaces to TCP/IP by using socket APIs.</td>
</tr>
</tbody>
</table>

![Layered architecture of FF HSE protocol](image-url)

**Fig. 1** Structures of stereotypes

**Fig. 2** Layered architecture of FF HSE protocol

**Fig. 3** Components and their association relationships of the stack software