© 2008 р. Vitaliy Mezhuyev, Eric Verhulst
Berdyansk State Pedagogical University, Berdyansk, Ukraine
Open License Society, Leuven, Belgium

OPENCOMRTOS VISUAL MODELLING ENVIRONMENT:
THE TOOL FOR DISTRIBUTED PARALLEL APPLICATIONS DEVELOPMENT

The paper describes the OpenComRTOS Visual Modelling Environment (VME) - the tool designed to help the software engineers the logic understanding, development, debugging, tracing, visualization and documentation of distributed parallel applications for the OpenComRTOS (network-centric Open Communication Real Time Operating System) and other RTOSes as well.

1. The VME main principles
VME was developed in view of next main principles.

Scalability: supports the development from small to very large and complex multi-processor applications.

Portability: i) an operating system independence; ii) a kernel independence; iii) supporting heterogeneous targets.

Extensibility: i) possibility of adding / removing functional modules (plug-ins) in an easy way; ii) possibility of changing and modifying the model of kernel services and objects in a

Introduction
There are some well known visual modelling environments for rapid applications development. They generated a target specific code from visual diagrams which have a fixed structure of system grammar (for example RTDS which uses SDL-RT syntax [5]). Recognizing the powerful of such systems lets notice that user cannot change the structure of concepts used for a visual modelling of a RTOS application.

The important feature of presented here Visual Modelling Environment (VME) is using the unique RTOS Application Metamodel (i.e. the model of a RTOS application model) on the base of Interacting Entities paradigm [1; 2]. The Metamodel allows for user define the set of own kernel objects and services needed for a visual application development. This makes the possible support of other RTOSes as well with using the plug-ins containing corresponding RTOS code generation tools.

VME supports software engineers in the following activities:

• visual definition of processing nodes topology;
• visual definition of a structure of a parallel application (distributed on processing nodes);
• visual definition of task code segments;
• building and running the application on target nodes;
• tracing a parallel application;
• editing Metamodel to support kernel object and services of other RTOSes;
• using additional plug-ins (code generation tools) for different RTOSes.

VME was developed with C++ in MS Visual Studio 2005 as the main IDE. As the library for GUI and Metamodel implementation the Qt 4.4 [3] was used. For efficient syntax analysis of C code the Meddle approach [4] was used.

The VME installation has specific for Win32 and other platforms OpenComRTOS include files, libraries and code generating utilities. This paper illustrates principles of VME using for development and execution of the OpenComRTOS applications on the Win32 host environment. As the compiler for OpenComRTOS applications on Win32 host environment the MinGW [6] is used.
Minimum Semantics: supporting the minimum application semantics of the initial system allowing extensions in a later stage of development.

2. RTOS Application Metamodel and Design Patterns

At different logical layers of the OpenComRTOS an application can be constructed from various entities (kernel objects) and interactions between them (kernel services). Kernel objects and kernel services also differ from one RTOS to another, but in general a RTOS application structure, properties and behaviour can be expressed in the Interacting Entities terms.

The following typification of a RTOS kernel objects and services is used in VME Metamodel:

- Kernel Object IS Entity.
- Kernel Service IS Interaction.

Also used the attributisation method:
- Kernel Entity HAS Attributes (2-N) as a Name, Type and Value corteges.

The definition (2-N) means that any Entity has at least 2 attributes: the Name and the processing Node, to which it belongs.

- Kernel Entity HAS Functions (1-N) which define its internal behaviour (e.g. EntryPoint as C-function for a Task entity).

As opposite to Functions the Interactions define an external behavious of entities.

- Kernel Interaction HAS Attributes (4), which specify a Name, a Subject (to be understood as initiator of the service), an Object (the entity that is engaged in the service) and the Direction of Interaction (i.e. to put or to get the service).

Such a simple Metamodel allows extending VME as to different sets of kernel entities and services of OpenComRTOS and also supports other RTOS entities and services (although this requires additional VME plug-ins with code generation tools).

The Metamodel reflects the RTOS application grammar i.e. the minimal system of concepts needed for unambiguous definition of a RTOS based application. That means that the internal functionality of a RTOS should be mostly hidden from the application developer (e.g. routing and scheduling can be considered as internal kernel mechanisms).

We consider the 2 possible ways of a RTOS visual application modelling:

1. If we define the Kernel as a separate entity we can consider Kernel Services as direct interactions between Application Tasks and the Kernel.

2. If we hide the Kernel entity i.e. consider it as a RTOS internal mechanism then all Kernel services become interactions between Application tasks by means of intermediate synchronisation entities (i.e. events, semaphores, FIFOs etc.).

To implement the principle of the minimum semantics we follow the second way i.e. the Kernel entity is hidden from a visual level of a RTOS application development. I.e. while in reality we have 5 different kinds of OpenComRTOS tasks: Kernel, Idle, Rx driver, Tx driver and Application task, for an application developer only the Application tasks need to be visible as he can consider the other tasks as the system level support.

The Kernel task implements the interactions between the tasks and manages the kernel entities, but at the application level these kernel actions are invisible. Also Rx (receive) and Tx (transmit) tasks are defined from the nodes topology and are not directly used by the OpenComRTOS application developer.

In the OpenComRTOS an interaction is implemented by Packets interchange (as a request for services or data transaction). While the Packet remains the base entity of the L0 logical layer of OpenComRTOS it is not directly used as an entity in VME. During OpenComRTOS application development user specifies interactions between application tasks and assumes that at internal level interactions are implemented by packets interchange for services calls or data transfer.

Thus in general selection of concrete L0, L1 or L2 logical layers entities for OpenComRTOS application development is relative (e.g. as we made the Packet entity invisible). The same principle concerns any entities of different logical layers of other RTOSes. So selection of entities is caused by level of abstraction we choose for a RTOS application development.

As such an OpenComRTOS application is constructed from next entities: task, port, event, semaphore, resource, FIFO, Packet Pool, and Memory Pool by means of specifying interactions between them. This actually defines a logical concurrent system.

Other RTOSes have their own specific set of entities and interactions, but remains as invariant
3. Expression of a RTOS Metamodel

The next figure shows horizontal and vertical levels of RTOS Model and Metamodel concepts relationships. Horizontal level is defined by Is-a, vertical Instance-of relations.

Each kernel object and service has defined in the Metamodel type, specifying the general properties and behavior.

For a RTOS Application Metamodel expression the XML format is used. XML format has no difference between conceptual and procedural kinds of knowledge (i.e. between entities and interactions), all kind of XML objects are just attributed nodes.

To distinguish between such kinds of objects in XML the Entity and Interaction keywords are used. This allows properly parse semantics of a RTOS Metamodel and configuration files.

In VME the Kernel Entities are shown as boxes with some inputs and outputs (interfaces). Allowed by Metamodel Interactions are shown as links between such entities.

An XML Metamodel file defines semantics of kernel entities and interactions by means of containing attributes XML nodes, i.e.:

```xml
<Interaction Name = "L1_PutPacketToPort_W" Subject = "Task" Object = "Port" Service="Put"/>
```

Means for the application programmer that Task and Port can take a part in L1_PutPacketToPort_W interaction (i.e. a Task sends packet to a Port entity).

4. VME design

From the RTOS Application Metamodel the VME takes information needed to parse concrete RTOS Application Model (see Figure 2).

The RTOS Application Model is stored in the XML configuration files defining the System parameters, Nodes and Links between Nodes, attributes of Application Tasks and Synchronization Entities.

RTOS configuration file is managed by GUI components: i.e. by Topology, Application diagrams and Flowchart. Structure of GUI is also depending on Metamodel. The output of GUI - the RTOS Configuration file at the same time is the input for specific for target the Code Generation Tools. Generated code is put on defined as project option compiler and after linking with RTOS libraries an application is launched on target proce-
ssor or host environment. Behaviour of RTOS application can be further traced by the special agent – Event Tracer.

5. VME workspace structure and function

VME can be defined as a graphical layer between services of a RTOS kernel and a RTOS application developer. VME GUI graphically represents general for parallel application features such as tasks, semaphores, events and other RTOS objects. Specific for kernel features can be easily added by the Metamodel changing. All these objects are structured in tree-graphs by different criteria (i.e. by the node, the class belonging to).

VME GUI is implemented as interrelated visual (topology, application, code diagrams) and textual (code segments editor) levels. From the visual diagrams an application repository is build up and a source code (in C) is generated as far as possible. Visual level is used to describe the application sketch, while the segments of the tasks are defined by the text editor.

The VME workspace is composed from several views, which offer an efficient way to handle its components, including kernel objects and services. Components are displayed in a hierarchical tree graphs associated with visual diagrams.

- The **Nodes View** shows the distribution of the kernel entities on the processors, grouping the objects accordingly to the node they belong. Entities that do not have direct association with a specific node (e.g. global parameters) do not appear in the Node View, being shown only in the Entities View.

- The **Entities View** groups all the kernel objects accordingly to the classes they belong to. It shows the hierarchical structure of Application entities by tree graph, gives possibility of objects composition and decomposition.

- The **File View** allows the edition of the text files belonging to the project. Both input files (used to configure and generate the application) and output files (generated when the application is build) are shown. The files are organized by a tree control where each item represents a file.

General structure of the VME workspace is shown on the Fig. 3. It has all inherent to such environments futures, can be easily understand from the figure.

The general algorithm of the visual definition of a parallel application is:

1. In the Topology diagram the topology of a system is visually defined (e.g. using the mouse to link with an arrowed line from a node A to a node B defines a directed link).

2. In the application diagram the logical structure of a program is visually defined by means
of the objects provided by the kernel (i.e. tasks and synchronization entities).

3. In the code diagram the code segment of each Task created at the application diagram is defined. A highlighting text editor is used for editing code segments.

5.1. Topology diagram

The Topology diagram is a space for creating and editing the processing network topology. Following Metamodel a network is composed from nodes and links.

Between the nodes multiple links are possible; they can be either uni- or bi-directional kind.

Thus the base graphical entities of the Topology diagram are:

1. **Node** is a processing device in a network containing CPU and with local memory and periphery. **Node** is graphically shown as the sphere with a node name as e.g. (see Figure 4).

2. **Link** is a physical or virtual point to point communication path between nodes. **Link** is graphically shown as a line with one or two ended arrows (to reflect unidirectional or bidirectional types of links) and connecting two nodes.

Fig. 3. The workspace of VME

Fig. 4. Using Topology Diagram
Modifications of the Nodes Topology on the diagram are automatically synchronized with the content of the Nodes and the Links sections of OpenComRTOS Configuration files.

For Win32 host environment:
- Each OpenComRTOS Node runs as a Windows process.
- OpenComRTOS Tasks run as Windows threads.
- Interaction between OpenComRTOS Nodes is implemented by means of TCP/IP socket communication.
- By default the Host attribute of a Node has the localhost value, but OpenComRTOS Node can be run on any host in the network by specification its IP address.

5.2. Application diagram

The general principle of an application development in VME is defining interactions between Tasks through dedicated Synchronisation Entities (i.e. Ports, Events, FIFOs etc.). In general, the Application diagram is a graph showing the logical structure of the RTOS supported application. The nodes of the graph represent the kernel entities used to construct the concurrent application. The links between the application entities represent their interactions.

Thus the base graphical entities of the Application diagram are:

1. **Kernel entities** (Task, Port, Event, FIFO, Packet Pool, Memory Pool, Resource, Semaphore) which are graphically represented by the nodes of the graph (e.g. $T_1$).

2. **Kernel interactions**. The edges of the graph represent calls of the tasks to the kernel services. The type of service is indicated by a service name placed over the edge (e.g. $\text{_1\_SignalSemaphore\_W}$).

5.3. Tasks code segments editor

The Application Diagram describes a RTOS application from a high level view as a set of Interacting Entities. This representation can be seen as a prototype that needs to be refined by application engineer to become a complete application. The functionality of the task segments between two interactions are defined by the using the Text Editor. Some parts of the source code are generated automatically from the graphical representation in the Application diagram:

1. For each Task entity of the Application an Entry Point is created in the source code.

2. For each Interaction between entities a corresponding kernel function call is created in the Task source code.
6. OpenComRTOS Event Tracer

The OpenComRTOS Event Tracer (see Figure 5) is an easy to use tool for real-time tracing OpenComRTOS tasks that are executed on a given number of processing nodes. These nodes can be linked using some form of a communication network and can be heterogeneous.

Event Tracer based on a tracing function provided by the OpenComRTOS kernel for each node. The tool displays in a table textual and graphical format the execution history of the application on a node:

- textual: event sorted listing (showing event number, timestamp, type of event, service description, a time of task execution);
- graphical: scrollable timelines showing state transitions between application, kernel, idle tasks, ISR and drivers with hubs using.

Event Tracer consists of three components – Event Table, Tasks Execution Diagram and Tasks Info Table. All components are synchronized and serve for representation of the same trace data taken from the trace file.

Trace files are generated per each processing node. Event Table shows the list of all loaded from trace file events, using special icons for visual designation. Tasks Info Table represents the generalized information on each task execution.

Tasks Execution Diagram and Sequence Chart visually displays the state transitions of tasks and also shows tasks service calls through hubs (e.g. in Figure 5 is shown the process of synchronization of Task1 with Sema1).

Conclusions

1. OpenComRTOS Visual Modelling Environment decreasing the time of distributed parallel applications development, minimizing errors at the coding process and so increasing productivity of software engineers work.

2. Defining the Metamodel of a RTOS as a System of Interacting Entities allows implementation of the Meta Modelling Environment i.e. the tool allowing modelling different sets of kernel objects and services of a RTOS.

3. Selection of concrete set of RTOS entities and interactions for an application development depends on the chosen level of abstraction. This level defines the patterns of RTOS application modelling. E.g. for simple user an internal functionality of a RTOS can be mostly hidden.

4. Exploring the different modelling patterns allows choosing optimal way of a RTOS applications development. To result of this research will be devoted our next paper.

REFERENCES

3. www.trolltech.com

В. Межуєв, Е. Верхалст Візуальне середовище моделювання OpenComRTOS: інструмент для розробки розподілених паралельних додатків

В. Межуєв, Е. Верхалст Визуальна среда моделирования OpenComRTOS: инструмент для разработки распределенных параллельных приложений

В статье описывается Визуальная Среда Моделирования VME – инструмент, разработанный для помощи программистам в понимании логики, написании, отладке, отслеживании событий, визуализации и документации разделенных параллельных приложений операционной системы реального времени OpenComRTOS, а также других RTOS.