C-012

A System Generator for a Distributed Real-Time Operating System with Distributed Shared Memory

Doan Truong Thi† Yuji Tamura† Myungryun Yoo† Takanori Yokoyama†

1. Introduction

Real-time operating systems (RTOS) are widely used in embedded systems. In a RTOS, software modules need to exchange its input and output values with each other. This is often done through global variables. Distributed shared memory (DSM) provides location-transparent shared variables, therefore distributed software modules can exchange their input and output values through shared variables on DSM.

Based on OSEK OS[1], we have already developed a distributed real-time operating system (DRTOS) with a real-time DSM service to fulfill that requirement[2]. In this paper, we present a system generator (SG) for the DRTOS. We extend the OIL—OSEK Implementation Language[3], a language for a standardized configuration information of OSEK OS—to declare DSM configuration. We also extend the system generator (SG) to support generating DSM configuration data referring to the extended OIL file.

2. Distributed Real-Time Operating System with Distributed Shared Memory

The structure of the DRTOS with DSM which we have developed is illustrates in figure 1. The DRTOS consists of the OSEK OS original functions, a timer synchronization module, a remote system call module, a distributed shared memory module and configuration data.

The DRTOS has task location determination feature, that task would get determined which node it belongs to when a target task's system call got issued. If that task belongs to current node, DRTOS executes OSEK OS's original system call. If that task belongs to other node, DRTOS executes remote system call.

The remote system call is another feature of DRTOS. The DRTOS sends a request message to a node, on which the target task of the remote system call resides. When a remote system call's request message is received from other node, DRTOS executes the requested OSEK OS's system call and sends back return value as well as parameters to the request node.

The distributed shared memory module manages the copies of shared variables and maintains the consistency between all nodes in the system.

Figure 2 illustrates example distrbuted control software with DSM, which shows how DSM maintains the consistency of shared variables between nodes. *Task11* on *Node1*, *Task21* on *Node2*, *Task31* on *Node3* and *Task41* on *Node4* respectively executes *Software Module X*, *Software Module Y*, *Software Module Z* and *Software Module W*. The DSM service on the DRTOS copies the shared variables x and y to the nodes and maintains the consistency of DSM.

Although it may be expected to maintain a sequential consistency of DSM, it is hard to implement the sequential consistency while keeping the expected performance. Using FlexRay communication features, we have developed a DRTOS which supports two types of DSM consistency, sequential-equivalent consistency and partial-sequential consistency.

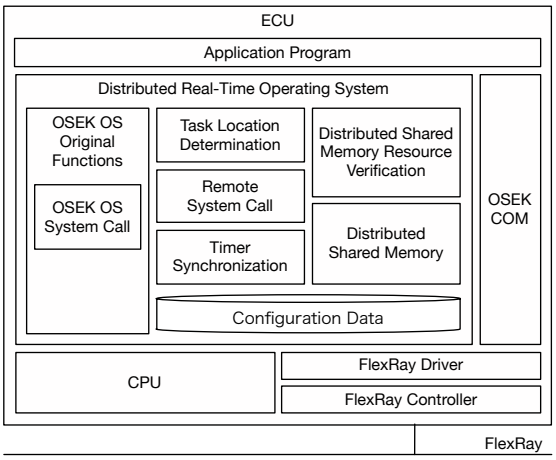

Figure 1: Structure of Distributed Real-Time Operating System

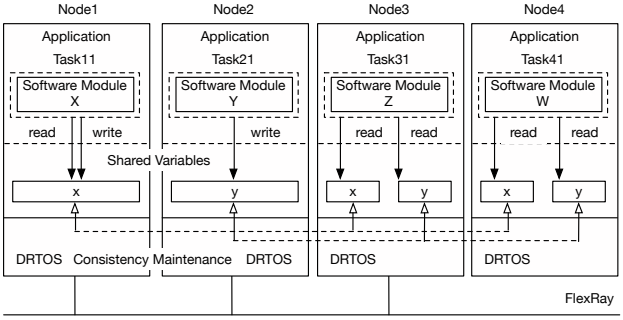

Figure 2: Example of Distributed Control Software

3. OIL Specification

Configuration information of DSM has to be defined in an OIL file so that SG can generate source code of configuration for DRTOS with DSM. Since the information needs to be accessible from all CPU objects, we define them in a separate section. We call it "distributed shared memory section". This section is defined after the implementation section and before the application section. If there are more than one DSM object, each one is defined one after another.

Figure 3 represents an example of an OIL file for a DRTROS with DSM. A DSM object consists of 3 attributes: *CDATATYPE*, *INITIALVALUE* and *CONSISTENCY*. *CDATATYPE* is the data type of the shared variable. *INITIALVALUE* is the initial value of the shared variable. *CONSISTENCY* is the consistency type of DSM, which is either *SEQUENTIAL* (DSM must be maintained a sequential-equivalent consistency), *PARTIAL* (DSM can be maintained with a partial-sequential consistency), or *NONE* (no need to maintain the consistency).

†Tokyo City University

In each *TASK* object, in order to tell the object which memory to access and its access authorization, we add a new attribute called *SHAREDRESOURCE*. This attribute has the following information: name of DSM which it refers to and parameter *ACCESS* that shows the access type to that DSM, which is either *READWRITE*, *READ* or *WRITE*.

With all above added configuration information, the new OIL specification is defined.

```
...
DSM sharedData0 {
    CDATATYPE = "long";
    INITIALVALUE = 0x0000;
    CONSISTENCY = EQUIVALENT;
};
CPU cpu0 {
    ...
    SHAREDRESOURCE = sharedData0 {
        ACCESS = READWRITE;
    }
    ...
};
CPU cpu1 { ... }
...
```

Figure 3: Example of OIL for a Distributed Real-Time Operating System with Distributed Shared Memory

4. System Generator

The System Generator that we develope in this research is based on TOPPERS/OSEK OS's System Generator[5]. It can generate all nodes' standard configuration data as well as DSM data at once.

The system generation process is devided into two phases—extraction phase and parsing phase.

4.1. Extraction Phase

The configuration information for a DRTOS with DSM is defined in an OIL file with new sepcification presented in section 3. All processes that are executed in this phase are explained below:

- The SG reads the implementation section from input OIL file and copies these definitions into each node's application OIL file (**app.oil**)
- The SG reads the DSM section from input OIL file and copies these definitions into each node's DSM OIL file (**dsm.oil**)
- The SG reads the application section from input OIL file, extracts each CPU's definition and appends it into the appropriate node's application OIL file (**app.oil**)

4.2. Parsing Phase

The whole process of the parsing phase is shown in figure 4. First of all, the SG parses each node's **app.oil** file which is generated at extraction phase and generate each node's configuration header file (**kernel_id.h**) as well as implementation file (**kernel_cfg.c**). This is done using the original TOPPERS/OSEK OS SG's original functions. The SG then parses each node's **dsm.oil** file and generate a DSM definition's header file (**distmem.h**) and an implementation file (**distmem.c**) for each node.

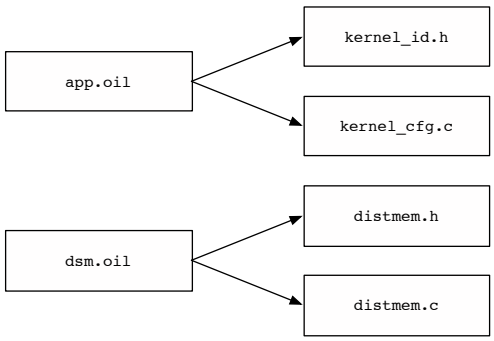

Figure 4: System Generator's parsing phase for each node

5. Conclusion

We have extended the OIL specification to adopt the definition of DSM objects. We designed and implemented an SG which is able to parse that new OIL specification and generate configuration data for DRTOS with DSM. With the extended SG, the generation process of configuration data for DRTOS with DSM could finally be done completely automatically.

Acknowledgments

We would like to thank the members of the TOPPERS Project for the development of TOPPERS/OSEK Kernel. This work was supported in part by JSPS KAKENHI Grant Number 24500046 and 15K00084.

References

[1] OSEK/VDX: *Operating System Version 2.2.3*, February 17, 2005.

[2] Takahiro Chiba, Myungryun Yoo, Takanori Yokoyama: *A Distributed Real-Time Operating System with Distributed Shared Memory for Embedded Control Systems*, IEEE 11th International Conference on Dependable, Autonomic and Secure Computing (DASC) 2013, Chengdu, China, pp.248-255, December 21-22, 2013, doi:10.1109/DASC.2013.71.

[3] OSEK/VDX: *System Generation, OIL: OSEK Implementation Language Version 2.5*, July 1, 2004.

[4] TOPPERS Project, http://www.toppers.jp/en/

[5] WITZ Co. Ltd: *TOPPERS/OSEK OS SG Instruction Manual Ver. 3.00*, May 30, 2006 (In Japanese).