A New Specification of Software Components for Embedded Systems

Takuya Azumi †, Masanari Yamamoto †, Yasuo Kominami ††,
Nobuhisa Takagi ‡, Hiroshi Oyama ‡‡, Hiroaki Takada †
†Graduate School of Information Science, Nagoya University
††TOPPERS Project Inc., ‡Kijineko Inc., ‡‡OKUMA Corporation

Abstract

In the last decade, the size and complexity of the software in embedded systems have increased. The present study attempts to decrease the complexity and difficulty of software development in embedded systems. We herein introduce a new component system that is suitable for embedded systems. It is possible to estimate the memory consumption of an entire application since the proposed system adopts a static configuration. In addition, this system takes into account to be used in several domains of embedded systems because several particle sizes of component are supported. Moreover, the concept of the component for a distributed application is presented.

1 Introduction

In recent years, three significant problems have become apparent with regard to software development in embedded systems. The size and complexity of the software in embedded systems are increasing rapidly. The required diversity of products, and their software, is increasing rapidly. The development time must be significantly decreased.

Over the last decade, the software component technology in versatile systems has been used to improve productivity, especially in the development of software for desktop applications and distributed information systems. Popular component systems include JavaBeans and ActiveX for desktop applications, and CORBA Component Model (CCM) [1, 2] and COM+ for distributed information systems. The productivity can be increased not only by reducing the coding time, but also by reducing the testing time. However, it is difficult to directly use these component technologies for embedded systems [4] because of the dynamic configuration of components. Recently, software component technologies for embedded systems have received increased attentions from researchers [10]. Component technologies for embedded systems, such as Koala component [8], PECT [9], and PBO [6], have been developed. Such component technologies, however, have not been widely used in the domain of embedded systems [3].

The TOPPERS [1] [7] Embedded Component System (TECS) has been investigated for three years. It is possible to estimate the memory consumption of an entire application because the TECS adapts a static configuration. The static configuration means that both the configuration of component behavior and interconnections between components are static. There are several benefits of using the static configuration. In addition, the TECS takes into account to be used in several domains of embedded systems because several particle sizes of component are supported and a diversity of component, such as an allocator or an RPC channel, is provided. Moreover, we also examine understandable interfaces and a distributed application for using a component on a different processor. Therefore, the TECS is suitable for embedded systems. The main purposes of the TECS are to decrease the complexity and difficulty of software development, increase productivity, reduce development duplication, and provide standard interfaces for increased reusability.

The TECS is described in Section 2 and Section 3. Section 4 focuses on a component of a distributed application. Evaluations and summaries of the present research are considered in Section 5 and Section 6.

2 The Specification of TECS

In this section, the specifications of the TECS are described in detail.

2.1 Features of the TECS

The embedded systems are usually considered to be resource constrained with respect to memory and must perform fast enough to fulfill their timing requirements. Typically, the greater the number of deadlines to be met, the

[1] TOPPERS (Toyohashi OPen Platform for Embedded Real-time Systems) Project, which is based on the technical development results obtained by applying ITRON, is aimed to developing base software for use in embedded systems.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

shorter the time between these deadlines. This is necessary in order to prevent the influence of using a component framework, such as increases in the memory consumption and processing time. Versatile component technologies, including JavaBeans and ActiveX for desktop application, and the CORBA Component Model (CCM) and COM+ for distributed information, are generally unsuitable for embedded systems. In the case of these component technologies, components are dynamically created and joined to other components in the execution time. This increases the overhead for creating, joining, and calling a component. In the case of the TOPPERS Embedded Component System (TECS), there is basically no need to reconfigure an application during the execution time. Consequently, components are statically created and joined to other components to develop an application. The static configuration is the most important feature in the TECS. In addition, the TECS takes into account to be used in several domains of embedded systems because several particle sizes of component are supported. A small particle size of component, such as a device driver component and so forth, is used to distinguish between the dependent and independent parts of hardware. A large particle size of component, such as a TCP/IP protocol stack and so forth, is used to enhance usability for an application developer (component user). A diversity of component, such as an allocator or an RPC channel, is provided to increase portability.

2.2 Components

A *cell* is a component in the TECS. *Cells* are properly joined in order to develop an appropriate application. A *cell* has *entry port* and *call port* interfaces. The *entry port* is an interface to provide services (functions) to other *cells*. The service of the *entry port* is called the *entry function*. The *call port* is a interface to use the services of other cells. A *cell* communicates in this environment through these interfaces. The *entry port* and the *call port* have *signatures* (sets of services). A *signature* is the definition of interfaces in a *cell*. Interface abstraction using a *signature* provides control of the dependencies of each cell. The *cell type* is the definition of a *cell*, such as the *Class* of an object-oriented language. A *cell* is an entity of a *cell type*.

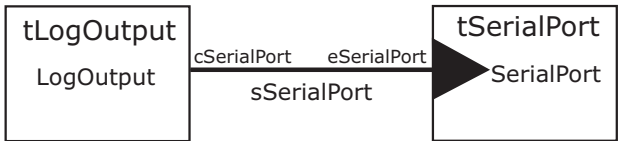

Figure 1. Example of Cells

Figure 1 shows an example for generating a log to a serial port. Each rectangle represents a *cell*. The left *cell* is a LogOutput *cell*, and the right *cell* is a SerialPort *cell*. Here, tLogOutput and tSerialPort represent the *cell type* name. The triangle in the SerialPort *cell* depicts an *entry port*. The connection of the *entry port* in the LogOutput *cell* describes a *call port*.

A *call port* can only connect an *entry port*. Therefore, in order to join several *entry ports*, *call port array* is used. A *entry port* can connect *call ports*. However, in this case, it is impossible to identify which *call ports* are connected. *Attribute* and *variable* keywords are to increase a variety of *cells*. For example, a *cell* of serial communication has an *attribute* to control the baud rate. A *singleton cell* is a particular *cell*, only one of which exists in a system. The *singleton cell* is used to reduce the overhead because the cell can be optimized.

2.3 Development flow

Figure 2 shows the development flow in TECS. In the next section, the *signature*, *cell type*, and build description are explained in detail. The *signature* description is used to define a set of function heads of a *cell type*. The *cell type* description is used to define the *entry ports*, *call ports*, and *attributes* of a *cell type*. The build description is used to declare *cells* and connect between *cells* for creating an application. An *interface generator* generates several interface codes (.h or .c) of the C language from the *signature*, *cell type*, and build descriptions. Developers in this framework are divided into two parts: a component developer and an application developer. The role of the component developer is to define the *signatures* and *cell type* and to write implementation codes (Component Source) of *cells*. Generally, a component is provided by the source code. On the other hand, the role of the application developer is to develop an appropriate application by joining *cells*.

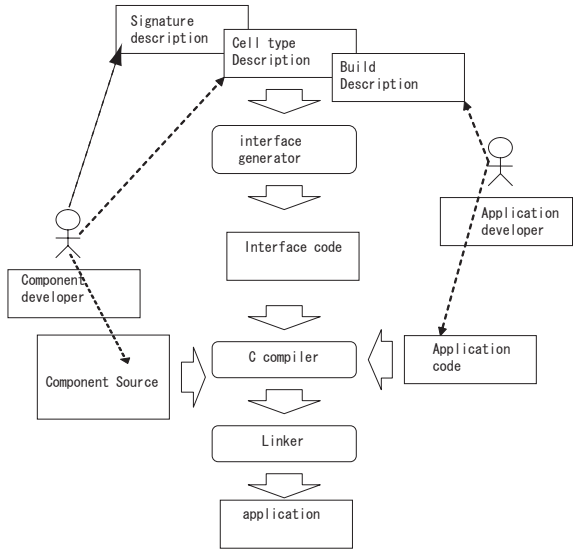

Figure 2. Development Flow

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

3 Component Description

The description of a component in the TECS can be divided into three descriptions: a *signature* description, a *cell type* description, and a *build* description. The *signature* and the *cell type* descriptions are described by component developers. The *build* description is written by application developers.

3.1 Signature Description

The *signature* description is used to define a set of function heads. A *signature* name, such as sSerialPort, follows a *signature* keyword to define *signature*. The initial of *signature* name ("s") represents *signature*. A set of function heads are enumerated in the body of this keyword.

```
signature sSerialPort {
/* open serial port */
   ER   opn_port( void );
/* close serial port */
   ER   cls_port( void );
/* write */
   ER_UINT wri_dat(
        [in , size_is(len)] char *buf,
        [in] UINT len);
/* read */
   ER_UINT rea_dat(
        [out, size_is(len)] char *buf,
        [in] UINT len);
/* control serial port  */
   ER   ctl_por( [in] UINT ioctl);
/* reference serial port */
   ER   ref_por(
        [out] T_SERIAL_RPOR *pk_rpor);
};
```

An interface description (prototype declaration) of the C language is ambiguous, as shown below. The ER represents the error code of the return value.

```
ER do_function(char * buf, int size,
               int *result);
```

An interface (*signature*) description in the TECS is very understandable, as shown below.

```
ER do_function(
 [in, size_is(size)] const char * buf,
 [in]                int size,
 [inout]             int *result);
```

The detail of understandable interface description is described as follows.

- Input or Output: The *in*, *out*, and *inout* keywords are used to distinguish whether a parameter is an input or an output. These keywords are understandable when a parameter is a pointer. In this case, the *result* parameter is used as input and output because the previous result has an effect on the subsequent result. It is important to use these keywords with respect to memory allocation in a distribute framework.
- Pointer: A pointer indicates an array or a value in the TECS. In this case, the *buf* parameter represents an array.
- Array Size: It is necessary to describe the size of an array by using *size_is* keyword in the TECS.

3.2 Cell Type Description

The *cell type* description is used to define the *entry ports*, *call ports*, and *attributes* of a *cell type*. A *cell type* can have *entry ports*, *call ports*, and *attributes*. A *cell type* name, such as tLogoutput, follows a *celltype* keyword to define *cell type*. The initial of *cell type* name ("t") represents *cell type*. To declare *entry port*, an *entry* keyword is used. Two words follow an *entry* keyword: a *signature* name, such as sLogOutput, and an *entry port* name, such as eLogOutput. The initial of *entry port* name ("e") represents an *entry port*. Likewise, to declare a *call port*, a *call* keyword is used. The initial of *call port* name ("c") represents a *call port*. To declare *attribute* of *cell type*, an *attribute* keyword is used. A set of *attribute* keywords are enumerated in the body of this keyword. This keyword can be omitted when a *cell type* does not have an *attribute*.

```
celltype tLogOutput {
  entry sLogOutput  eLogOutput;
  call  sSerialPort cSerialPort;
};
 celltype tSerialPort {
  entry sSerialPort   eSerialPort;
  attribute{
    ...
  };
};
```

3.3 Build Description

The build description is used to declare *cells* and to connect between *cells* for creating an application. To declare *cell*, a *cell* keyword is used. Two words follow a *cell* keyword: a *cell type* name, such as tSerialPort, and an *cell* name, such as SerialPort. In this case, eSerialPort (*entry port* name) of SerialPort (*cell* name) joined cSerialPort (*call port* name) of LogOutput (*cell* name). The *signatures* of *call port* and *entry type* must be the same in order to join cells.

```
cell tSerialPort  SerialPort {
};
cell tLogOutput LogOutput {
 cSerialPort = SerialPort.eSerialPort;
};
```

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

4 Distributed Component

4.1 Composite Cell Type

The TECS provides several levels of composition for application developers. A *composite cell type* includes two or more *cells*. Figure 3 is one of the simplest examples of a *composite cell type*. The tCompositeCellType, shown in Figure 3, includes two *cells*.

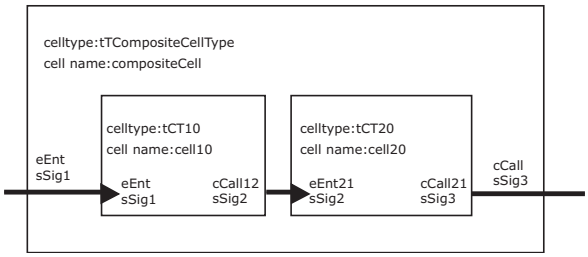

Figure 3. Composite Cell Type

```
composite tCompositeCelltype{
   cell tCelltype20 cell20{
   };
   cell tCelltype10 cell10{
     cCall12 = cell20.eEnt21;
   };
   eEnt = cell10.eEnt;
   cCall  = cell10.cCall21;
};
```

The *composite cell type* description of Figure 3 is described as above. A *composite cell type* name, such as tCompositeCelltype, follows *composite* keyword to define *composite cell type*. The eEnt and cCall are automatically decided as a *signature* and an *entry port*/a *call port* by checking the connected port. All of the benefits of compositionality imply significant reductions in complexity.

4.2 RPC Channel Cell

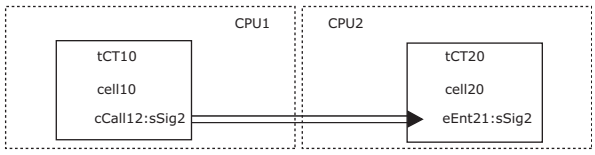

Figure 4. Connection of a Distributed Cell

Figure 4 represents an example of the communication between *cells* on different CPUs. In this case, cell10 (*cell* name) of tCT10 (*cell type* name) on CPU1 uses cell20 (*cell* name) of tCT20 (*cell type* name) on CPU2 through an RPC channel. The dual line between the cells shown in Figure 4 depicts an RPC channel. The RPC channel is one of the *cell types* in the TECS. An RPC channel *cell type* is shown in Figure 5. The tRPCChannel is a composite *cell type*. The tRPCChannel can be divided into four parts: tRPCMain, tMarshall, tUnmarshall, and tChannel *cell types*. The tRPCMain cell type controls the RPC channel. The tMarshall cell type is used to convert a standard date type for an RPC. The tUnmarshall cell type is used to reverse a standard data type. The tChannel cell type is used to transfer data that is converted by tMarshall.

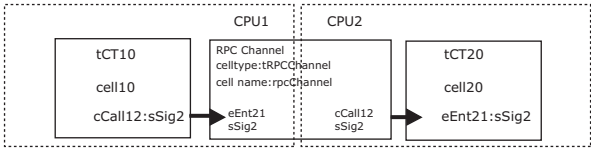

Figure 5. RPC Channel Cell Type

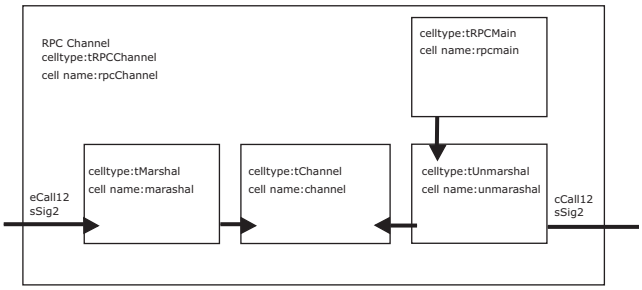

Figure 6. Composite Cell Type of an RPC Channel Cell Type

5 Evaluation

The requirements of component technologies for embedded systems can be divided into two different aspects [5]: technical requirements and development requirements. The technical requirements include analyzability, modeling and computation, openness, portability, and resource constraint. The development requirements include reusability, maintainability, introducibility and understandability. The detail of each technical requirements is described as follows.

- **Analyzability:** A Software component technique should be easy to analyze with respect to the non-functional properties such as the timing correctness and the memory consumption in embedded systems. The timing correctness and the memory consumption are at least as important as the functional correctness in real-time systems. It is possible to estimate the memory consumption of a *cell* because of the static configuration in the TECS. Therefore, application programmers are able to calculate the memory consumption of an entire application.

- **Modeling and Computation:** The component modeling should be based on a standard modeling, such as UML. A GUI editor is provided to build components for the developers in the TECS.

Proceedings of the 10th IEEE International Symposium on
Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)
0-7695-2765-5/07 $20.00 © 2007 IEEE

- **Openness:** Components should be source codes. Generally, a component is provided in the TECS. It is useful to provide the source code of a component for optimizing the code during the compilation time. Moreover, application developers can be used to the source code, find functional errors, and enable support for white box testing. This approach leads to better reliability and safety of components.
- **Portability:** The components should be platform-independent to the highest possible degree. The TECS has a high portability because a *cell* is hardware-independent, real-time OS-independent, and communication-independent.
- **Resource Constraint:** The embedded systems considered here are usually resource-constrained, with respect to the CPU and memory. Therefore, in order to decrease the overhead, the TECS adapts a static configuration and the singleton *cell* is used in the TECS.
- **Reusability:** The components should be easy to be reused for other applications. The standard signatures, such as the input/output signature, are provided in the TECS. The component developers, having made effective use of these signatures to implement a component, the component becomes the high reusable.
- **Maintainability:** Components should be easy to change and maintain. As mentioned above, standard signatures are used. Therefore, it is possible to change and extend components without influencing other joined components.
- **Introducibility:** An appropriate support for developers to migrate into new technologies should be considered for the component technologies. An application developer can utilize the TECS without any difficulties because standard signatures are provided. Moreover, it is possible to covert legacy codes into *cells*.
- **Understandability:** The system should be easy to understand and should simplify evaluation by the application developer. It is possible to estimate the memory consumption of each *cell*. The particle size of a *cell* can be changed by using a composite *cell type*. Therefore, the application developer can evaluate an application on both the component level and the system level.

6 Conclusion

The present paper described the TOPPERS Embedded Component System (TECS). It is possible to estimate the memory consumption of an entire application because the TECS adopts a static configuration. A static configuration means that both the configuration of the component behavior and the interconnections between components are static. the TECS takes into account to be used in several domains of embedded systems because several particle sizes of component are supported and a diversity of component, such as an allocator or an RPC channel, is provided. Therefore, the TECS is suitable for embedded systems. In addition, the TECS takes into account distribute components. In the future, we will consider an access control mechanism between *cells* for protecting the resources of each *cell*.

Acknowledgment

The authors would like to thank the TOPPERS component working group for their helpful comments and suggestions.

References

[1] OMG, CORBA Component Model 4.0. http://www.omg.org/technology/documents/formal/components.htm.
[2] OMG, CORBA. http://www.omg.org/corba/.
[3] I. Crnkovic. Component-based software engineering for embedded systems. In *Proceedings 27th International Conference on Software Engineering (ICSE2005)*, pages 712–713, Missouri, USA, 15-21 May 2005.
[4] S. Lin, J. Wu, and Z. Hu. A contract-based component model for embedded systems. In *Proceedings. Fourth International Conference on Quality Software, 2004. QSIC 2004*, pages 232–239, Braunschweig, Germany, 8-9 September 2004.
[5] A. Möller, M. Åkerholm, J. Fredriksson, and M. Nolin. Software component technologies for real-time systems. In *WiP Proceedings in Real-Time Systems Symposium (RTSS),*, pages 49–51, Canun, Mexico, 3-5 December 2003.
[6] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software usingport-based objects. *Software Engineering*, 23(12):759–776, December 1997.
[7] TOPPERS Project. http://www.toppers.jp/en/index.html.
[8] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, March 2000.
[9] K. C. Wallnau. Volume iii: A component technology for predictable assembly from certifiable components. In *Technical report, Software Engineering Institute, Carnegie Mellon University*, Pittsburg, USA, 2003.
[10] I.-L. Yen, J. Goluguri, F. Bastani, and L. Khan. A component-based approach for embedded software development. In *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2002. (ISORC 2002)*, pages 402–410, Washington, DC, USA, 29 April-1 May 2002.