A Visual Modeling Environment for Embedded Component Systems

Takuya Azumi †, Shimpei Yamada ††, Hiroshi Oyama ‡, Yukikazu Nakamoto ††, and Hiroaki Takada †
†Graduate School of Information Science, Nagoya University
††Graduate School of Applied Informatics, University of Hyogo
‡OKUMA Corporation

Abstract

This paper proposes a new visual modeling environment for embedded component systems that improves the productivity of application developers. This embedded component system decreases the complexity and the difficulty of software development for embedded systems. Furthermore, it is possible to estimate the memory consumption of an entire application, since the proposed system uses a static configuration. This environment builds the development components for the application. As well, the visual modeling environment can automatically generate a build description that is part of the component description language. Finally, this environment can be used to generate C-language interface code, either .h or .c, from the component descriptions.

1 Introduction

During the last decade, three important issues in the development of software for embedded systems have been recognized: the increased size and complexity of the required systems; the increased diversity in the end products and software required; and the decreased development time.

Concurrently, software component technology for versatile systems, such as JavaBeans [7], CORBA Component Model (CCM) [2, 3], and COM+, has been developed to increase productivity. Productivity can be increased by not only reducing coding time, but by also reducing testing time. However, since these component systems are primarily used for desktop applications or distributed information systems, they cannot be easily used for embedded systems [8]. Recently, software component technologies for embedded systems have received increased attention from researchers [13]. Component technologies for embedded systems, such as Koala component [11], PECT [12], and PBO [9], have been developed. Such component technologies, however, have not been widely used in the domain of embedded systems [4].

For the past three years, TOPPERS 1 [10] Embedded Component System (TECS) [1] has been investigated. TECS allows an estimation of the memory consumption of an entire application because TECS adapts a static configuration. The static configuration means that both the configuration of the component behavior and the interconnections between components are static. There are several benefits of using the static configuration. Furthermore, TECS has an easy to use interface and can be used on different processors. Therefore, TECS is suitable for embedded systems. The main purposes of TECS are to decrease the complexity and difficulty of software development, increase productivity, reduce development duplication, and provide standard interfaces leading to increased reusability.

To improve the productivity and usability of applications, this paper proposes a new visual modeling environment for the embedded component systems that allows the components to be individually built for the appropriate application. As well, the visual modeling environment can automatically generate a build description that is part of the component description language. Finally, this environment can be used to generate C interface code, either .h or .c, for the component descriptions.

TECS is described in Sections 2 and 3. The visual modeling environment and interface generator are explained in Section 4. Section 5 presents the conclusions that can be drawn based on this paper.

2 TOPPERS Embedded Component System

In this section, the specifications of the TOPPERS Embedded Component System (TECS) are described in detail.

---

1TOPPERS (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.
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 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 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 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 Composite Cell Type

The TECS provides several levels of composition for application developers. A composite cell type includes two or more cells. Figure 2 is one of the simplest examples of
a composite cell type. The tCompositeCellType, shown in Figure 2, includes two cells.

Figure 3 shows a composite cell type of a serial driver. This cell has two semaphore cells, for sending or receiving, and a cell to control the serial driver. The developers are able to deal with composite cells, such as general cells. All of the benefits of compositionality imply significant reductions in complexity.

2.4 Development flow

![Diagram of Development Flow]

Figure 4. Development Flow

Figure 4 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 C-language interface codes, either .h or .c, from the signature, cell type, and build descriptions. The interface generator and interface code are described in Section 4.2. 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.

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.

```c
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.

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

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

```c
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 cell type 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;  
};
```

The composite cell type description of Figure 2 is described as below. 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.

```
composite tCompositeCelltype{  
    cell tCelltype20 cell20{  
    };
    cell tCelltype10 cell110{  
        cCall12 = cell20.eEnt21;  
    };
    eEnt = cell110.eEnt;  
    cCall = cell110.cCall;  
};
```

### 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, `cSerialPort` (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;  
};
```

### 4 Development Environment For Application Developer

Figure 5 shows the application development process, which is divided into two stages. The first stage involves generating the build description and requires the use of a visual modeling environment. The signature and cell type descriptions, which are determined by the component developers, are the input data for the visual modeling environment. The second stage involves generating the interface codes using an interface generator, which are described in Section 4.2.
4.1 Visual Modeling Environment

Figure 6 shows the visual modeling environment based on the Eclipse [5] Integrated Development Environment. This modeling environment supports application developers in building the components. As well, in order to increase productivity, this modeling environment automatically generates the build description after modeling. The modeling environment is divided into four parts: a project manager, a visual editor, a property editor, and an outline view.

The project manager, shown in Figure 6 (A), is used to manage the signatures, cell types, and build the data. The signature and cell type data are provided by the component developers. The cell type data is used to create the cell in the visual editor. The build data is used to hold the cells and connections between the cells that the application developer has created using the visual editor.

Figure 6 (B) shows the visual editor, which is based on the Graphical Editing Framework (GEF) [6]. It is divided into two parts: a palette part and an editor part. The palette part is located on the left hand side of the visual editor and consists of three items: Select, Marquee, and Join. Select is used to select an editor symbol in the editor part. Marquee is used to select a set of symbols in the editor part. Join is used to enter the join mode for connecting a call port and an entry port. The editor part is found on the right hand side of the visual editor and is used to build components using the visual symbols. Each rectangle in the editor shows a cell. The square in each cell represents the call port. The triangle in each cell represents the entry port. To move the symbols, position the pointer inside the bounding box and drag. To scale the symbols, drag the side or corner handle. Connectors represent the abstraction of the interactions between the call port of a cell and the entry port of another cell. In order for the cells to be joined, the signatures of the call port and the entry port must be the same. To minimize errors on the part of the developers, the editor provides some functions that increase usability. When a call port is clicked, the color of the entry ports that have the same signature as the call port is changed form black to blue. The visual editor also has the following functions: undo, redo, zoom in, and zoom out. Once the component has been designed, the editor automatically generates the build description.

The property editor shown in Figure 6 (C) is used to show and edit the normal data, which includes the cell type name, the cell name, and attributes, as well as the drawing data, which includes the co-ordinates, width, and height, of the selected cell.

Figure 6 (D) shows the outline view, which is divided into two parts: a text-based outline part and a navigator part. The text-based outline part located in the upper section of the outline view is used to select a cell that is to be displayed using the visual editor. The navigator part located in the lower section of the outline view is used to display a thumbnail view of the visual editor for easy navigation.
The outline viewer is useful when there are many cells in the visual editor.

4.2 Interface Generator

An interface code for joining cells is necessary to build an appropriate application. An interface generator generates several C-language interface codes, either .h or .c, from the signature, cell type, and build descriptions.

- **global_tecsgen.h**
  This file includes the definitions of data types, structures, and constants in an overall application.

- **sSigname_tecsgen.h**
  This file is generated for each signature and includes the definitions of the signature (function table). The sSigname represents each signature name, such as a sSerialPort_tecsgen.h.

- **tCelltype_tecsgen.h**
  This file is generated for each cell type and includes the definitions of type for a control block of each cell type. The tCelltype represents each cell type name, such as a tLogOutput_tecsgen.h. A descriptor for entry ports and a Macro for entry functions. The control block is a structure to manage a cell. The descriptor for entry ports is abstract entry ports which call ports of other cell use.

- **tCelltype_tecsgen.c**
  This file is generated for each cell type and includes the definitions of control block of each cell type and the skeleton functions of entry ports. The skeleton functions are used to call the entry of entry port.

- **tCelltype_template.c**
  This file is generated for each cell type, includes template codes of entry functions, and is based on an implementation code for a component developer.

5 Conclusion

This paper describes a visual modeling environment for application developers and an interface generator for TOP-PERS Embedded Component System (TECS). The visual modeling environment is divided into four parts: a project manager, a visual editor, a property editor, and an outline view. To increase productivity and ease of use for application developers, each part allows the user to easily construct the component. The visual editor is used to build the components for the development of an appropriate application. As well, the visual modeling environment can automatically generate the build description. The interface generator generates C-language interface codes, either .h or .c, from the signature, cell type, and build descriptions. Thus, this modeling environment and interface generator can speed-up the development process.

Acknowledgment

This work was partially funded by the Exploratory Software Project of Information technology Promotion Agency (IPA). The authors would like to thank the TOPPERS component working group for their helpful comments and suggestions.

References


