Optimization of Component Connections for an Embedded Component System

Takuya Azumi, Hiroaki Takada
Graduate School of Information Science
Nagoya University
Nagoya Japan
Email: {takuya.hiro}@ertl.jp

Hiroshi Oyama
FA Systems Division
OKUMA Corporation
Niwa-gun, Japan
Email: hi-ooyama@okuma.co.jp

Abstract—Software component techniques are widely used to enhance productivity and reduce the cost of software systems development. This paper proposes optimization of component connections for a component system that is suitable for embedded systems. This component system adopts a static model that statically instantiates and connects components. The attributes of the components and the interface code for connecting the components are statically generated by the generator. No instantiation overhead is introduced at runtime, and the runtime overhead of the interface code is minimized. A case study using a serial interface driver is presented to evaluate the execution overhead, the software code size, and the executable file size. The case study shows the effectiveness of optimization.

Keywords—component-based development; real-time operating system;

I. INTRODUCTION

Recently, three significant problems have appeared in software development for embedded systems. Firstly, the size and complexity of the software for embedded systems is increasing rapidly. Secondly, the required diversity of products and their software is increasing rapidly. Finally, the development time must be significantly decreased.

During the last decade, software component technology for 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 [1] and ActiveX [2] for desktop applications, and CORBA Component Model (CCM) [3] and COM+ [4] 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 [5] because of the dynamic configuration of components. Recently, researchers have focused on software component technologies for embedded systems [6]. Component technologies for embedded systems, such as PBO [7], Koala component [8], PECT [9], and SaveCCT [10], have been developed. However, such component technologies are not widely used in the domain of embedded systems [11].

TOPPERS 1 [12] Embedded Component System (TECS) [13], [14] has been investigated for several years. It is possible to estimate the memory consumption of an entire application, since TECS uses a static configuration. The static configuration implies 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 can be used in different embedded systems domains, since a variety of component granularities are supported, and a diversity of components, such as an allocator or an RPC channel, are provided. A small component, such as a component’s device driver, is used to distinguish the dependent and the independent hardware parts.

TECS adopts a static model that statically instantiates and connects components. The attributes of the components and interface code for connecting the components is statically generated by the generator. Furthermore, optimization of the interface code is proposed. Hence, no instantiation overhead is introduced at runtime, and the runtime overhead of the interface code is minimized.

The details of TECS are described in Section II. Section III presents a method for optimizing component connections, while Section IV summarizes the results of the current research.

II. TOPPERS EMBEDDED COMPONENT SYSTEM

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

A. Component Model

A cell is an instantiation of the component in TECS. Cells are properly connected 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 an interface that uses the services of other cells. In this environment, a cell communicates through

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 develop base software for use in embedded systems.
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, which is similar to the Class of an object-oriented language. A cell is an entity of a cell type.

Figure 1 shows an example of a serial port. Each rectangle represents a cell. The left cell is a SerialPort cell which is a target-independent part of the serial port, and the right cell is an SIOPort cell which is a target-dependent part of the serial port. Here, tSerialPort and tSIOPortST15c2550 represent the cell type name. Each triangle in the SerialPort or SIOPort cell depicts an entry port. The connection of the entry port in the cells describes a call port.

A call port can only connect to an entry port. Therefore, in order to connect several entry ports, a call port array is used. An entry port can connect to other call ports. However, in this case, it is impossible to identify which call ports are connected. Attribute and variable keywords are used to increase the number of different types of cells. For example, a serial communication cell has an attribute to define the send or receive buffer sizes. Although each attribute, such as the buffer size, can not be changed during execution time, each variable, such as the file name of a file cell, can be changed. Thus, it is assumed that the attributes are automatically mapped to ROM, and that the variables are automatically mapped to RAM. A singleton cell is a particular type of cell, of which there is only one in the system. The singleton cell is used to reduce the overhead because the cell can be optimized.

B. Development Flow

Figure 2 shows the development flow in TECS. In Section II-C, 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, attributes, and variables of a cell type. The build description is used to declare the cells and create the connections between cells in order to assemble an application. An interface generator generates several interface C- (.h or .c) and template codes for the comment source 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 types, as well as write the implementation code (Component Source) of the 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 connecting the cells.

C. Component Description

The description of a component in TECS can be divided into three parts: 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 the application developers.

1) Signature Description: The signature description is used to define a set of function heads. Figure 3 represents the signature descriptions to define the serial port interface. A signature name, such as sSerialPort, follows a signature keyword to define the signature. The initial part of the signature name (“s”) represents the signature. A set of function heads is enumerated in the body of this keyword.

A detailed explanation of the interface description is given below.
Figure 3. Signature Description of Serial Port.

```
1 //to use dependent part
2 signature sSIOPort {
3   void open(void);
4   void close(void);
5   bool_t putChar([in] char_t c);
6   int_t getChar(void);
7   void enableCBR([in] uint_t cbrtn);
8   void disableCBR([in] uint_t cbrtn);
9 }
10 //call back
11 signature siSIOCBR {
12   void readySend(void);
13   void readyReceive(void);
14 }
15 //to use serial port for application
16 signature sSerialPort {
17   ER open(void);
18   ER close(void);
19   ER_UINT read([out, size_is(len)] char_t *buf, [in] uint_t len);
20   ER_UINT write([in, size_is(len)] char_t *buf, [in] uint_t len);
21 }
```

Figure 4. Cell Type Description of Independent Part.

```
1 //cell type definition of independent part
2 celltype tSerialPortST {
3   //entry port for application
4   entry sSerialPort eSerialPort;
5   //entry port for dependent part
6   entry siSIOCBR eiSIOCBR;
7   //call port to use dependent part
8   call sSIOPort cSIOPort;
9   attr{
10     //receive buffer size
11     uint_t rcv_bufsz = 128;
12     //send buffer size
13     uint_t snd_bufsz = 128;
14   }
15   var{
16     //receive buffer
17     [size_is(rcv_bufsz)]
18     char_t *rcv_buffer;
19     //send buffer
20     [size_is(snd_bufsz)]
21     char_t *snd_buffer;
22 }
```

Figure 5. Cell Type Description of Dependent Part.

2) Cell Type Description: The cell type description is used to define the entry ports, call ports, attributes, and variables of a cell type. Figure 4 and 5 describe the cell type description used to define components for the serial port. A cell type can have entry ports, call ports, attributes, and variables. A cell type name, such as tSerialPort, follows a celltype keyword to define cell type. The initial part of a cell type name (“t”) represents the cell type. To declare an entry port, an entry keyword is used. Two words follow the entry keyword: a signature name, such as sSIOPort, and an entry port name, such as eSIOPort. The initial part of the entry port name (“e”) represents an entry port. Likewise, to declare a call port, a call keyword is used. The initial part of the call port name (“c”) represents a call port.

The attr and var keywords that are used to increase the number of different cells are attached to the cell type and are initialized when each cell is created. For example, a serial port cell has an attribute to define the buffer size. The set of attributes or variables is enumerated in the body of these keywords. These keywords can be omitted when a cell type does not have an attribute and/or a variable.

3) Build Description: The build description is used to declare cells and to connect between cells for creating an application. Figure 6 shows a build description to instantiate and connect the cells. To declare a cell, the cell keyword is used. Two words follow the cell keyword: a cell type name, such as tSerialPort, and an cell name, such as SerialPort (Lines 2-7 in Figure 6). In this case, eSIOPort (entry port name) of SIOPort (cell name) connected to cSIOPort (call port name) of SerialPort (cell name). The signatures of the call port and the entry port must be the same in order to connect the cells.
III. COMPONENT CONNECTION OPTIMIZATION

A. Component Connection Structure

Figure 7 shows the component connection structure. In this case, the structure represents the SerialPort cell (left-hand side) that calls the open function of the SIOPort cell (right-hand side). The numbers in Figure 7 describe the order of the calling procedure.

The developers only implement the entry port function. The other parts, entry port descriptor, entry port skeleton function table, and entry port skeleton, are generated by the interface generator.

The words in Figure 7 are explained below.

- **Cell initialized block**
  
  A cell initialized block (cell INIB) is a data structure that has the call ports and the attributes of each cell. The call port stores the pointer of entry port descriptors. The data of the Cell initialize block are mapped to ROM.

  Figure 8 represents the structure of a cell INIB for tSerialPort. The structure is defined based on the cell type description for each cell type. The structure has attributes (Lines 3-4). In case of a call port, the structure stores the pointer that connects with the pointer of the entry port descriptors (Line 5).

- **Cell control block**

  A cell Control Block (cell CB) is a data structure that stores the pointer of the cell INIB and the variables of each cell. The data of the cell CB are mapped to RAM. The arrays shown in Figure 10 are allocated by the interface generator. Figure 11 represents the initialization of cell CB for SerialPort. The structure
stores the pointer of the buffers allocated in Figure 11 (Lines 5-7) and the pointer of the cell INIB (Line 9).

- **IDX**
  IDX is an identifier to obtain the cell CB. There are two types of IDX: a subscript to the array-managed cell CB, and the pointer to the cell CB. Figure 7 shows an example where IDX is a pointer.

- **Entry port descriptor**
  An entry port descriptor (Figure 7(A)) is a structure that has entry ports and the cell information, such as a pointer to the entry port skeleton function table, IDX, and a subscript to the entry array.

- **Entry port skeleton function table**
  An entry port skeleton function table (Figure 7(B)) is a function table that has a set of function pointers to the entry port skeleton.

- **Entry port skeleton**
  An entry port skeleton (Figure 7(C)) is a function for calling an entry port function that has the caller cell information, such as IDX and the subscript to the entry array.

- **Entry port function**
  An entry port function (Figure 7(D)) is a function for providing the functionality of the cells. An entry port function is based on template code generated by the interface generator.

### B. Connection Optimization

The call port and entry port of the cells can be optimized when certain conditions are met.

1) **Optimization of a call port:** There are four types of optimization for call ports that reduce the execution time.

   1. Omission of the entry port skeleton function
   The entry port function is directly called without passing through the entry port skeleton function if and only if the number of parameters of the entry port functions connected to a call port is constant. The number of parameters depends on whether a cell type is singleton or not, and whether an entry port is an entry port array or not.

   2. Omission of function table
   The entry port function is directly called without referring to the entry port skeleton function table if and only if each call port of all the cells is connected to the cells that are instantiated by a cell type.

   3. Omission of the entry port descriptor
   The cell CB of a calling cell is mapped to the cell INIB of the caller’s cell if and only if the call port optimization 2 and the entry port connected to a call port are not the entry port array.

   4. Single cell
   The call port of a caller cell directly includes the cell CB and cell INIB of the caller cell if and only if the call port optimization 3 and all the call ports of all the cells that are instantiated by a cell type are connected to a cell.

2) **Optimizations of entry port:** There three types of optimization for entry ports that reduce memory consumption.

   1. Deletion of the entry port skeleton function
   An entry port skeleton function can be deleted if and only if there are no references to an entry port skeleton function when the call port optimization 1 is applied to the call port.

   2. Deletion function table
   An entry port skeleton function table can be deleted if and only if there are no references to a function table when the call port optimization 2 is applied to the call port.

   3. Deletion of the entry port descriptor
   An entry port descriptor can be deleted if and only if there are no references to an entry port descriptor when the call port optimization 3 is applied to the call port.

### C. Improving Call Port Optimization

The optimization of the call port can be improved when the composition of the cells are known using Figure 12 as an example. In this example, the optimization of the call port of cells (A1 or A2) instantiated by the same cell type tA is described. The numbers in Figure 12 represent the call port optimization number from Section III-B1.

In case (A), it is not possible to adapt to the omission of the function table optimization because the connected cell types of cell A1 and A2 are different (tC or tD). Furthermore, the number of parameters in the entry functions is different because tB is not a singleton cell type and tC is a singleton cell type. Therefore, this case cannot be optimized.

In case (B), it is also not possible to adapt to the omission of the function table optimization. It is possible to adapt the omission of the entry port skeleton function optimization.
because the cell type connected to the A1 or A2 cell is not a singleton nor an entry array. In cases (C-F), it is possible to adapt to the omission of the entry port skeleton function operation.

In case (C), it is possible to adapt to the omission of the entry port skeleton function optimization because the cell type connected to A1 or A2 cell is the same cell type (tE). It is not possible to adapt to the omission of the entry port descriptor optimization, because the entry ports connected to A1 and A2 cell are entry port arrays.

In case (D), it is possible to adapt to the omission of the entry port skeleton function optimization because the cell type connected to A1 or A2 cell is the same cell type (tB). As well, it is possible to adapt to the omission of the entry port descriptor optimization, because the entry ports connected to A1 and A2 cells are not an entry port array. It is not possible to adapt the single cell optimization because the cells connected to A1 and A2 cells are different (B1 or B2).

In case (E), it is possible to adapt to all the optimizations because the cell connected to A1 or A2 cell is the same cell (B1).

In case (F), it is also possible to adapt to all the optimizations because the cell installed by the tA cell type is only A1.

As mentioned in Section III-B2, optimization for the entry ports is adapted if and only if there are no references to the information when the call port optimization implements the call port.

**D. Experimental Results**

This subsection presents a set of experimental results obtained by comparing a non-component program with a component program in order to demonstrate the function call overhead and the effectiveness of optimization. Table I depicts the experimental platforms.

Three different measurements of the function calls are performed: non-component program, component program without optimization, and component program with optimization. The serial driver of TOPPERS/ASP which is a real-time operating system is used as a non-component program. In the non-component case, the glue code between the dependent and non dependent parts is hand-coded.

All the optimizations described in Section III-B1 are adapted to the call port. The measurement range of execution is from the point where an application calls the send function of the serial driver to the point where the procedure returns to the caller application.

Table II shows the calling overhead results. Function call overhead is evaluated in two cases where the number of characters is one or five. The numbers in Table II are the average execution times for 10,000 iterations. Component programs with optimization were approximately 2 μs faster for the one-character case and 5 μs faster for the five-character case compared with the component program without optimization. Furthermore, the overhead of a component program with optimization is less than 1 μs compared to the non-component program. These results imply that the execution time overhead for the component program with optimizations is relatively small, which indicates that the appropriate interface code has been generated by the interface generator.

Next, the line numbers of the source code written by a developer are compared. The values in Table III represent the independent part, the dependent part, the glue code (glue), the component description (CD), and the total number of lines of code. Comments and empty lines are not measured.
Table III

<table>
<thead>
<tr>
<th>Section</th>
<th>text</th>
<th>data</th>
<th>bss</th>
<th>rodata</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>456B</td>
<td>185B</td>
<td>0</td>
<td>125B</td>
<td>766B</td>
</tr>
<tr>
<td>Non component</td>
<td>549B</td>
<td>278B</td>
<td>0</td>
<td>119B</td>
<td>946B</td>
</tr>
</tbody>
</table>

Table IV

<table>
<thead>
<tr>
<th>Section</th>
<th>text</th>
<th>data</th>
<th>bss</th>
<th>rodata</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non component</td>
<td>3,632B</td>
<td>0B</td>
<td>568B</td>
<td>100B</td>
<td>4,300B</td>
</tr>
<tr>
<td>Component without optimization</td>
<td>3,632B</td>
<td>100B</td>
<td>512B</td>
<td>190B</td>
<td>4,436B</td>
</tr>
<tr>
<td>Component with optimization</td>
<td>3,120B</td>
<td>100B</td>
<td>512B</td>
<td>88B</td>
<td>3,820B</td>
</tr>
</tbody>
</table>

Table III shows that the total code size of the component program is smaller than that of the non-component program. This is because the source code lines generated by the interface generator are more than those of the component description. Furthermore, in the TECS case, developers only modify a few lines of the build description to change other serial ports. On the other hand, in the non-component case, developers have to implement glue codes for each serial controller. Therefore, by using TECS, it is easy to change system configurations.

Finally, Table IV shows the size of each section. The text section includes the program operation, the data section includes the variables of the cell, the bss section contains the send or receive buffer, and the rodata section contains the attributes of the cell. It is possible to reduce RAM consumption because the attributes of each cell are mapped to ROM (rodata). Additionally, the text and rodata sections are reduced because of the deletion of the entry port descriptor and the entry port skeleton function if the proposed optimization technique is used. Furthermore, the glue code is simplified in components that are optimized. Therefore, the total file size of the component program with optimization is smaller than the non-component program.

IV. Conclusions

The present paper described TOPPERS Embedded Component System (TECS). It is possible to minimize the execution time overhead and the memory consumption of an entire application because TECS adopts a static configuration. A static configuration means that both the configuration of the component behavior and the interconnections between components are static. Additionally, an optimization of the interface code was proposed. No instantiation overhead was introduced at runtime, and runtime overhead of the interface code was minimized. A serial interface driver was used as a case study to evaluate the execution time overhead, the software code size, and executable file size. The case study demonstrated the effectiveness of the proposed optimization.

REFERENCES