TINET+TECS: Component-based TCP/IP Protocol Stack for Embedded Systems

Takuro Yamamoto, Takuma Hara, Takuya Ishikawa, Hiroshi Oyama, Hiroaki Takada
Graduate School of Engineering Science, Osaka University
OKUMA Corporation

Takuya Azumi
Graduate School of Engineering Science, Osaka University

***Abstract*—Embedded systems are applied to Internet of Things (IoT), and the high productivity of embedded network software is required.**

TINET is a TCP/IP protocol stack for embedded systems. TINET is a compact TCP/IP protocol stack, however consists of many complex source codes. Therefore, it is difficult to maintain, extend, and analysis the software.

To improve the scalability and configurability, this paper has proposed a component-based TCP/IP protocol stack for embedded systems: TINET componentized with TOPPERS embedded component system (TECS). The component-based TINET provides software developers high productivity such as change of network buffer size and add/remove TCP (or UDP) function. We evaluate the component-based TINET compared with the original TINET, and confirm that the overheads of execution time and memory consumption are low and that the configurability is improved.

In this paper, we also demonstrates a use case of the dynamic component connection method of TECS with a network application.

I. Introduction

Internet of Things (IoT) is an essential keyword for the next era. Such as wearable devices, smart devices, and smart homes, various things are connected to the Internet, enriching our lives. Embedded systems are the elements constituting IoT, e.g., sensing data and controlling actuators. Therefore, embedded network software is required for the high productivity and quality.

TINET (Tomakomai InterNETworking) [1] is a compact TCP/IP protocol stack for embedded systems. TINET supports the ability such as minimum copy frequency and elimination of dynamic memory control. TINET needs small memory for its TCP/IP protocol stack, therefore it is suitable for embedded systems. However, there are several issues that TINET consists of many complex source codes. It may take a lot of time for software developers to maintain, extend, and analysis the software because TINET is composed of many files and defines many macros.

Component-based development is an approach to improve software productivity. Component-based development is a design technique that can be applied to reusable software development [2], such as TECS [3], AUTOSAR [4], and SaveCCM [5]. Component-based systems are flexible to software extension and specification changes. In addition, individual Component diagrams enable the visualization of an entire system.

In this paper, component-based TCP/IP protocol stack, i.e., componentized TINET, is proposed to improve the configurability and scalability of software. TECS (TOPPERS Embedded Component System) [3] is utilized to componentize TINET, because TECS is a component system suitable for embedded systems. TECS supports static configuration, that component behaviors and interconnections are defined statically, thus TECS can optimize the overhead of componentization.

Contributions: This paper provides the following contributions.

1) **Improve configurability**
   TINET+TECS is a component-based system, therefore the software is flexible to change the configuration such as resize network buffer, add/remove TCP (or UDP) functions, and support both IPv4 and IPv6. In addition, a component diagram provides visualization of TINET, a complicated system.
2) **Dynamic connection method**
3) **Support legacy codes**
   TECS supports the adapter to call TECS functions from C codes, thus TINET+TECS can use an existing application.

Organization: The remainder of this paper is organized as follows. Section II introduces the basic technologies, i.e., TINET and TECS. Section III describes the design and implementation of the proposed framework. Section IV evaluates the proposed framework. Related work is discussed in Section V. Conclusions and suggestions for future work are presented in Section VI.

II. Basic Technology

A. TINET

TINET is a compact TCP/IP protocol stack for embedded systems based on the ITRON TCP/IP API Specification [6], developed by TOPPERS (Toyohashi OPen Platform for Embedded Real-time Systems) Project [7]. To take restrictions for

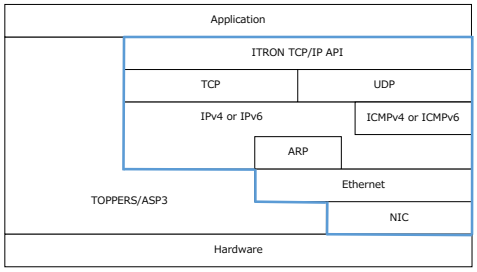

Fig. 1. Hierarchy Diagram of TINET and TOPPERS/ASP3

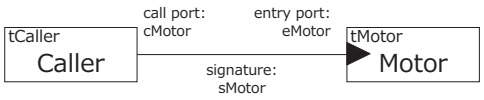

Fig. 2. Component Diagram

embedded systems such as memory capacity, size, and power consumption, TINET supports following functions:

1) Minimum copy frequency
2) Elimination of dynamic memory control
3) Asynchronous interface
4) Error detailed per API

TINET runs as middleware on TOPPERS/ASP3 [8], a realtime kernel. TINET has been released as open source.

1) Overview: Fig. 1 shows the hierarchy diagram of TINET and TOPPERS/ASP3.

B. TECS

TECS is a component system suitable for embedded systems. TECS can increase productivity and reduce development costs due to improved reusability of software components. TECS also provides component diagrams, which help developers visualize the overall structure of a system.

In TECS, component deployment and composition are performed statically. Consequently, connecting components does not incur significant overhead and memory requirements can be reduced. TECS can be implemented in C, and demonstrates various feature such as source level portability and fine-grained components.

1) Component Model: Fig.2 shows a component diagram. A *cell*, which is an instance of a component in TECS, consists of *entry* ports, *call* ports, attributes and internal variables. An *entry* port is an interface that provides functions to other *cell*s, and a *call* port is an interface that enables the use of other *cell*'s functions. A *cell* has one or more *entry* ports and *call* ports. *Cell* functions are implemented in C.

The type of *entry*/*call* port is defined by a *signature*, which is a set of functions. A *signature* is the interface definition of a *cell*. The *cell*'s *call* port can be connected to the *entry* port of another *cell* by the same *signature*. Here, *celltype* defines one or more *call*/*entry* ports, attributes, and internal variables of a *cell*.

```
1  signature sMotor {
2     int32_t getCounts( void );
3     ER resetCounts( void );
4     ER setPower( [in]int power );
5     ER stop( [in] bool_t brake );
6     ER rotate( [in] int degrees,
7              [in] uint32_t speed_abs,
8               [in] bool_t blocking );
9     void initializePort( [in]int32_t type
          );
10 };
```

Fig. 3. Signature Description

```
1  celltype tCaller {
2     call sMotor cMotor;
3  };
4  celltype tMotor {
5     entry sMotor eMotor;
6     attr {
7        int32_t port;
8     };
9     var {
10       int32_t currentSpeed = 0;
11    };
12 };
```

Fig. 4. Celltype Description

2) Component Description: In TECS, components are described by *signature*, *celltype*, and build written in component description language (CDL). These components are described as follows.

Signature Description

The *signature* defines a *cell* interface. The *signature* name follows the keyword *signature* and takes the prefix "s" e.g., sMotor (Fig.3). In TECS, to clarify the function of an interface, specifiers such as [in] and [out] are used, which represent input and output, respectively.

Celltype Description

The *celltype* defines *entry* ports, *call* ports, attributes, and variables. A *celltype* name with the prefix "t" follows the keyword *celltype*, e.g., tCaller (Fig.4). To define *entry* ports, a *signature*, e.g., sMotor, and an *entry* port name, e.g., eMotor, follow the keyword *entry*. *Call* ports are defined similarly. Attributes and variables follow the keywords *attr* and *var*, respectively.

Build Description

The build description is used to instantiate and connect *cell*s. Fig.5 shows an example of a build description. A *celltype* name and *cell* name, e.g., tMotor and Motor, respectively, follow the keyword *cell*. To compose *cell*s, a *call* port, *cell*'s name, and an *entry* port are described in that order. In Fig.5, *entry* port eMotor in *cell* Motor is connected to *call* port cMotor in *cell* Caller. *C_EXP* calls macros defined in C files.

```
cell tMotor Motor {
   port = C_EXP("PORT_A");
};
cell tCaller Caller {
   cMotor = Motor.eMotor;
};
```

Fig. 5. Build Description

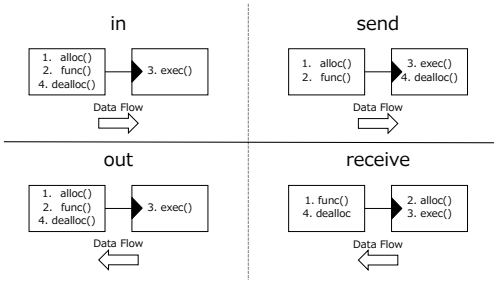

Fig. 6. Differences between in/out and send/receive

III. Design and Implementation

A. send, receive specifier

TECS supports send and receive specifiers, which are interface specifiers [9]. TINET+TECS uses send and receive specifiers instead of in and out to reduce the number of copies.

in is a specifier for input arguments. A callee side uses the memory of arguments with in while executing the callee function. When the processing returns to the caller side, the caller can reuse and deallocate the memory.

send is also a specifier for transferring data to a callee from a caller such as in. The difference between in and send is whether to deallocate the data memory in caller or callee, shown as Figure 6. In case of in specifier, both allocating and deallocating the data memory are performed in the caller. On the other hand, in case of send, the caller allocates the data memory and the callee deallocates it.

out is a specifier for output arguments. A callee writes data in the memory allocated by a caller, and the caller receives the data.

receive is also a specifier for a caller receiving data from a callee such as out. The difference between out and receive is whether to allocate the data memory in caller or callee, shown as Figure 6. While, in case of out, a callee writes data in the memory allocated by a caller, in case of receive, the callee allocates the data memory. Deallocating the memory is performed in the caller in both cases.

B. TECS Adapter

TECS supports *Adapter* functionality which enables to call a function in TECS from C codes. An adapter connects a TECS component, links a C function to a TECS function shown as Fig. 8. Software developers can utilize an existing application for TECS owing to the adapter.

```
signature sNicDriver {
  void start([send(sNetworkAlloc),size_is(
      size)]int8_t *outputp,
           [in]int32_t size,
           [in]uint8_t align);
  void read([receive(sNetworkAlloc),
      size_is(*size)]int8_t **inputp,
          [out]int32_t *size,
          [in]uint8_t align);
   /* Omit: other functions */
};
```

Fig. 7. An example of send and receive

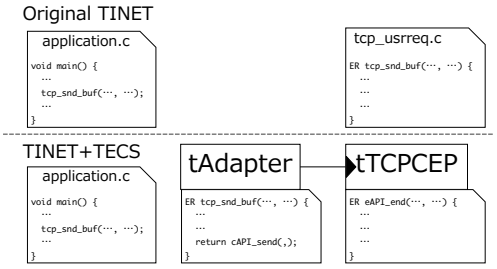

Fig. 8. TECS Adapter

IV. Evaluation and Demonstration

V. Related Work

VI. Conclusion

Acknowledgment

References

[1] TOPPERS, "TINET," https://www.toppers.jp/en/tinet.html.

[2] X. Cai, M. R. Lyu, K.-F. Wong, and R. Ko, "Component-based software engineering: technologies, development frameworks, and quality assurance schemes," in *Processings of Seventh Asia-Pacific Software Engineering Conference (APSEC 2000)*, 2000, pp. 372–379.

[3] T. Azumi, M. Yamamoto, Y. Kominami, N. Takagi, H. Oyama, and H. Takada, "A New Specification of Software Components for Embedded Systems," in *Proceedings of the 10th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, 2007, pp. 46–50.

[4] "AUTOSAR," http://www.autosar.org/.

[5] M. kerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli, "The SAVE Approach to Component-based Development of Vehicular Systems," *Journal of Systems and Software*, vol. 80, no. 5, pp. 655–667, 2007.

[6] "ITRON TCP/IP API Specification (Ver. 1.00.01, only Japanese version is available)," http://www.ertl.jp/ITRON/SPEC/tcpip-e.html.

[7] "TOPPERS Project," http://www.toppers.jp/en/index.html.

[8] TOPPERS, "TOPPERS/ASP3 kernel," https://www.toppers.jp/asp3-kernel.html.

[9] T. Azumi, H. Oyama, and H. Takada, "Memory Allocator for Efficient Task Communications by Using RPC Channels in an Embedded Component System," in *Proceedings of the Ninth IASTED International Conference on Software Engineering and Applications*, 2008, pp. 204–209.

TABLE I
EVALUATION BOARD ENVIRONMENT

Board	GR-PEACH
CPU	Cortex-A9 RZ/A1H 400MHz
Flash ROM	8 MB
RAM	10 MB
LAN Controller	LAN8710A