http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 9 (5): 849-863, 2010
ISSN 1812-5638

Management of the Reconfigurable Protocol Stack Based on SDL for Networked Control Systems

Chen Hui, Zhou Chunjie, Huang Xiongfeng, Qing Yuanqing and Shi Yufeng
Department of Science and Technology, Huazhong University of Science and Technology,
1037 Luoyu Road, Wuhan, Hubei-430074, China

Abstract: This study presented an integrated management framework for the implementation of reconfigurable protocol stack in Networked Control Systems (NCSs). Reconfiguration is a vital capability to cope with protocol heterogeneity and environmental variations and growing attention has been focused on run-time automated management of complex functional entities of communication protocol stacks. Around the discussions on protocol reconfiguration, there is still lacking of a systemic method for NCSs, especially ignoring the performance verification and evaluation for the stack code replacement. Therefore, in this study, a high level modeling language-Specification and Description Language (SDL) was used to formalize the protocol stack. During the reconfiguration procedure, the formal specification was evaluated through running on a set of SDL performance models. And optimization operations would update the inefficient protocol stack till performance constraints are satisfied. Then, a binary coding rule was proposed to construct a design space for exploring reconfiguration parts of the formal protocol specification and generating the protocol configuration file. Besides, the standard ELF (Executable and Linkable Format) file was employed to support the runtime code loading avoiding unexpected linking overheads. The experiment showed that the management framework was efficiency for promoting the reconfiguration capability in communications of NCS with critical real time guaranteed.

Key words: NCS, protocol stack, SDL, reconfiguration

INTRODUCTION

A networked control system is a distributed control architecture where sensors, actuators and controllers are interconnected through real time network. The communication protocol stack that directly affects the perceived network Quality of Service (QoS) plays a critical role in determining the system performance. However, the lacking of unified communication protocol standard leads to the problem of communication environment heterogeneity (Kolla *et al*., 2003). Besides, the availability of communication resources may change unexpectedly, due to changes in network user demands, or disturbances in the network environments such as the loss of a link (Xia *et al*., 2004). Consequently, the network QoS becomes unexpectedly changeable and may not be able to provide the required QoS level to some control tasks as needed. In particular, critical tasks are required to maintaining the QoS level in any case, such as maximum bound delay bound (Cheng *et al*., 2007); otherwise the result could be catastrophic.

To cope with this challenge, there has been an increasing attention on developing protocol stack with reconfiguration capability to provide the flexibility in such heterogeneous and changeable environment. Traditional monolithic protocol stacks are static in nature (An *et al*., 2006). It potentially hinders the networking of products from different manufactures or standards, let alone the adaptability to environment variations. The concept of dynamic configuration of protocol stack is introduced by Muhugusa *et al*. (1995), which is an environment that lets applications dynamically mix and match protocol functionality according to their requirements and network availability. Bridges *et al*. (2007) proposed separated micro-protocol modules to support a framework for constructing configurable protocol and services. Min *et al*. (2008) discussed the dynamic code reconfiguration mechanism in operating systems of wireless sensor network with focus on minimizing energy consumption. Niamanesh and Jalili (2009) proposed a DRAPS architecture supporting the peer synchronous reconfiguration that is concerning the consensus of

Corresponding Author: Chen Hui, Department of Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan, Hubei-430074, China Tel: +86 027 87558001 Fax: +86 027 87543230

protocol stack between two communication peers when reconfiguration happens on one side. However, few discussions have been given on the real time communication in NCSs. Moreover, most of reconfiguration approaches pre-compute the possible configurations, store them in a share space and at runtime do a simple table lookup to decide the next configuration. Due to the heterogeneous problem in NCSs, communication protocol configurations and control applications are too many to be pre-compute.

Furthermore, the functional correctness and the performance satisfaction are still lack studied in the existing dynamic reconfiguration architectures. Prior to the protocol implementation and system deployment, it is not possible to completely debug the logic holes and foresee the situation of services in the future. The SDL technique that provides unambiguous descriptions of the protocol functionality and the communication rules are needed in the protocol design phase as well as in the runtime reconfiguration phase. The SDL technique has been advanced for many years. The TAU SDL tool (Telelogic Inc., 2000) provided an inexpensive but reliable way of verifying protocols under development in advance of implementation and final agreement. Chan and Bochmann (2003) created an SDL framework that allowed users to reuse and to add SIP services to the core protocol. Yang *et al*. (2005) pointed out the lack of precise functional description in the traditional network simulators and presented a methodology of SDL-based network performance simulation. Fischer *et al*. (2005) presented an automated code generation that enabled the flexible connection of various communication protocols. Hannikainen *et al*. (2000) presented the automatic SDL-to-C code generation problem for an embedded target platform. Jabri *et al*. (2008) studied the formal validation of a new protocol for the load balancing approach using SDL pattern. It was a load balancing approach to improve QoS and to provide an adaptation between application and available physical resources of network. These researches have covered different aspect of SDL applications respectively in the development of protocol stacks. SDL could be served as the foundation for the design and management of protocol stack for NCSs. Nowadays, demands for an integrated management framework become more and more urgent, which comprises processes of evaluating, optimizing, determining and loading the new protocol stack configuration.

This study, our main contribution is to extend the formal technique SDL to automate and optimize the protocol stack reconfiguration procedure, including the aspects of performance self-evaluation, configuration self-optimization and automated code generation. Considering the real-time and dynamic characteristics, we firstly propose the reconfigurable protocol stack of NCS by given its SDL description models. Then, a set of environment and structure models are provided as the performance constraints for the running of formal protocol models. Such a pure-SDL based performance estimation approach is severed as the foundation to support the automated evaluation and optimization procedure. Besides, in order to optimize the code generation procedure, a binary design space is defined to coding and exploring the formal description space for the SDL parser. On receiving the protocol configuration file, formatted with the standard ELF file (Tool Interface Standards (TIS) Committee, 1995), a self-switching scheme is presented in the code composer to reduce the impact of code reconfiguration operations. The code reconfiguration operations, such as assembling, linking and swapping, are scheduled by utilizing the system idle time without interrupting tasks of reconfiguration independence. Finally, as a case study, a scheduling reconfiguration scenario for CAN system is taken to demonstrate the efficiency of our management framework for the proposed protocol stack.

SDL SPECIFICATION AND MANAGEMENT INFRASTRUCTURE

Reconfigurable protocol stack of NCS: In contemporary industry, networking system and environment are complicated, changeable and even unpredictable. Present research is focused on protocol stack reconfiguration for nodes of NCS. It is an environment that enables different applications could dynamically adjust the configuration of each layer, mixing or matching protocol function blocks in accordance with control demands and network availability. It adopts strict layered working mode (Fig. 1). According to the reconfiguration implementation emphasis, the protocol stack is structured into 3 layers: the Application (APP) layer with various objects, the Network Transmission Layer (NTL) with software reconfiguration and the Communication Link Layer (CLL) with hardware reconfiguration.

APP layer on the top: The APP layer is the interface between the protocol stack and users to collect the node information and interprets the user tasks. It provides high reliable data services for the applications on the user layer. Meanwhile, all the application data calling for the services of NTL are divided into two queues: the periodic and the non-periodic. Besides, if the number of nodes is changed, the node management block will reassign the node address and update this information for the scheduling scheme of NTL. If the system structure or

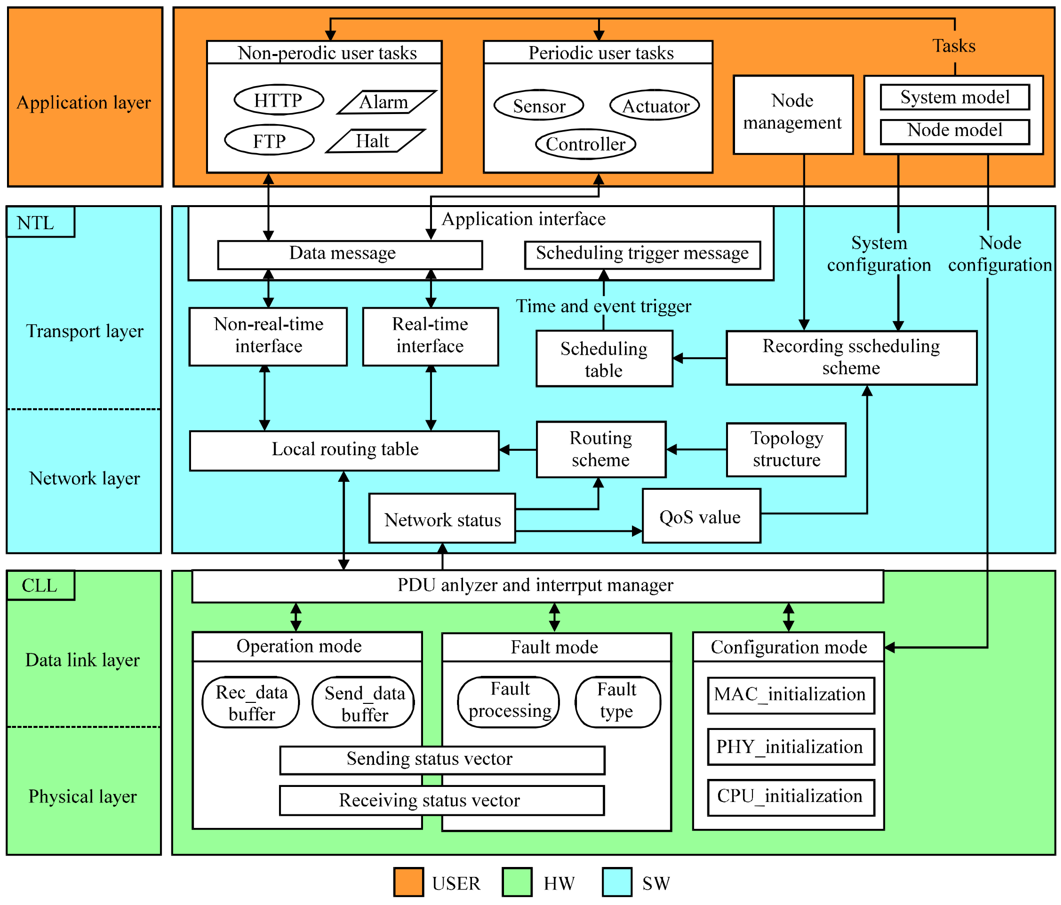


Fig. 1: Architecture for the reconfigurable protocol stack of NCS

application object is modified, the system model and the node model will be constructed to re-initialize platform resources, which are required by the configuration mode of CLL.

NTL on the middle: The NTL is the key of software reconfiguration composing of three parts: the transport channels, the scheduling scheme and the routing scheme. The transport channels are implemented for periodic, sudden and non-real time data in NCS, therein, the non-real time part deals with the seamless integration to general file transfer and monitoring services (such as FTP or FTTP) and the real time part concentrated on time critical networking and controlling services to coordinate with the real time scheduling scheme. The scheduling scheme is implemented to control the data flow between APP and NTL through the scheduling table and the transmission trigger signals. Based on the non-periodic transmission declarations or measured QoS level, the scheduling scheme could proactively reconfigure the scheduling parameters, such as bandwidth per task, the polled sequence for slave nodes and the length of a time slice. On the other hand, after the transmission is triggered, the routing scheme could provide end to end data link service with QoS guaranteed for the scheduling scheme. It means the reconfiguration of routing paths among nodes (formed as a routing table) when an equipment join in To our best knowledge or leave. Therefore, all periodic and non-periodic tasks from APP layer could be scheduled with the assumption that the transmission delay for each routing path is bounded into a certain range.

CLL on the bottom: The CLL comprises the Physical Layer (PHY) and the Data Link Layer (DLL). In common, the functionalities of PHY and DLL are integrated in a special-purpose chip, such as the MCP2510 for CAN and the DM9000 for Ethernet, named commuication controller. Thus, different from the NTL, the CLL is the key of hardware reconfiguration that is concerning the appropriate parameter configuration on the hardware chip. The CLL could provide a reliable point to point physical

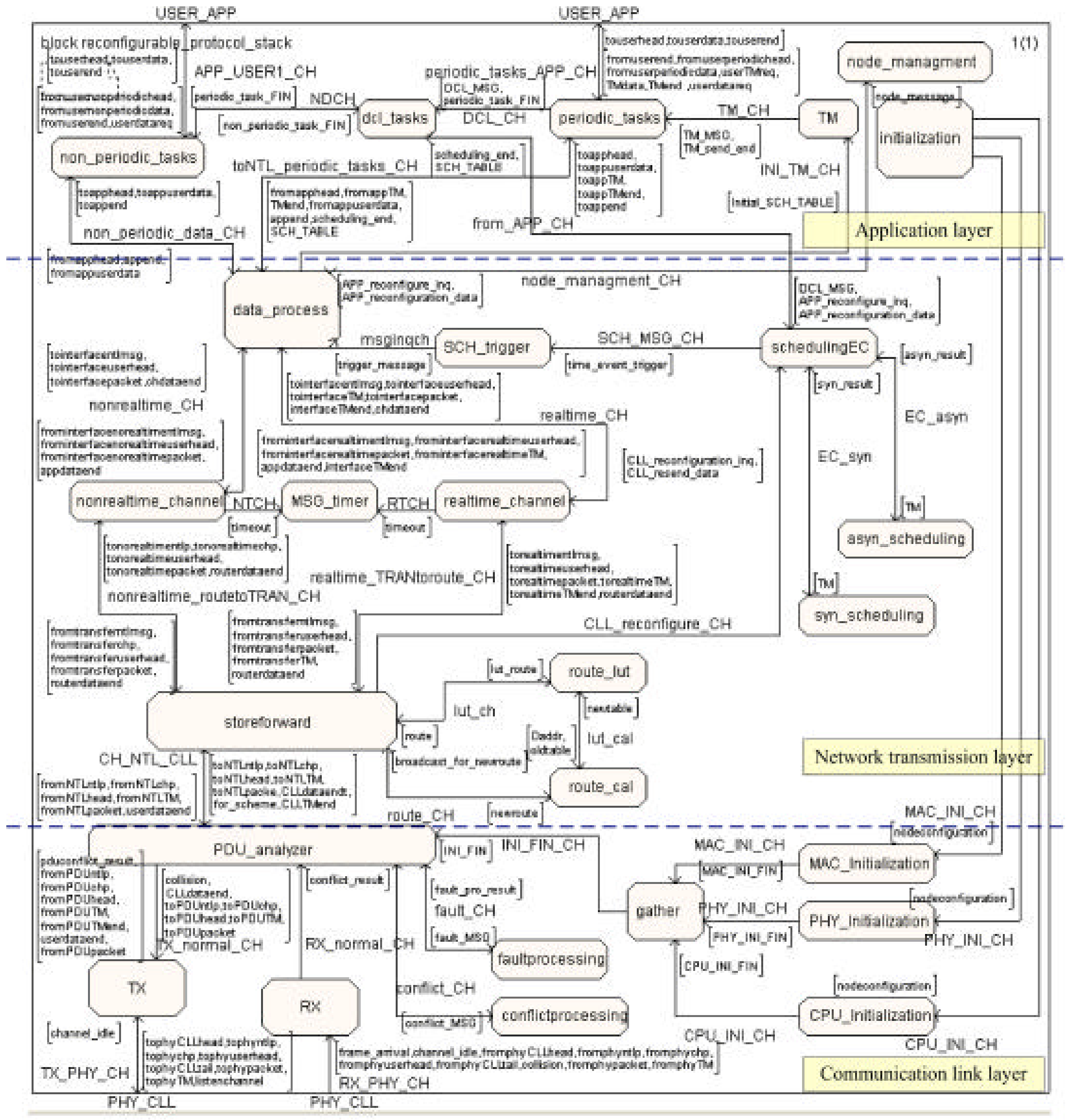


Fig. 2: SDL based description for the reconfigurable protocol stack of NCS

link for the routing scheme of NTL. Operation mode and fault mode are two basic operation mode of CLL. Once conflict or link loss happening, the sending or receiving vector of communication controller will record it. Then, based on the type of error, the PDU Analyzer starts the Interrupt Manager to switch the working mode of MAC controller, from the normal operation to the fault mode. After executing operations like rollback, step out or reset, the CLL will be back to the normal mode.

Formalized specification with SDL: As shown in Fig. 2, the SDL description for the reconfigurable protocol stack of NCS is specified, which is structured with distributed processes of each layer. It could be grouped with layers of APP, NTL and CTL. The layer-to-layer interface is detailed through the interaction messages between two processes. It is the minimum functional set for the design of real time protocol stacks of NCS, unlike the style of OSI reference model. For instance, when the standard TCP protocol is used as the implementation of process non-real time channel, it is required to provide the signals interfacing with process storeforward, MSG_timer and data_process. An overview of our SDL working model is described as followed.

Firstly, the initialization of whole protocol stack is triggered before the beginning of communication or recalled after node resetting. A process Initialization on the top APP is implemented as a user interface to convey the configuration parameters to three processes on the bottom CLL: CPU_Iniialization, PHY_Initialization and MAC_Initialization. The ready signal from the process gather is used to activate the process PDU_analyzer, which is to manage the normal mode and the fault mode of CLL. Mode switching could be control by two signals: one is fault_MSG due to link errors; the other is conflict_MSG due to the simultaneous sending request. In the fault mode, the process fault processing would prevent the next following messages entering the sending buffer, then, check the error type and renew the link status. The process conflict processing would assign a period of backoff time to the low priority message before its resending. After fault correcting or waiting a backoff time, the CLL will be back to normal mode: the message sending process TX or receiving process RX.

After that, when the CLL is in the normal sending and receiving processes, it could provide a reliable peer-to-peer link service for the processes of routing protocol in NTL, which is consists of three processes: storeforward, route_lut and route_cal. Considering dynamic non-periodic traffics, the routing table may not complete. The destination address should be checked first when the message entering the routing block. If the process route_lut reports that the destination does not exist, the process route_cal is activated to calculate the shortest (lowest cost) routing path. Then, the route_lut could be continued and generated an address valid signal to the process storeforward, forwarding messages to CLL directly if the destination is in the same network segment, or conveying messages to next hop node according to the table generated by the process route_cal.

Besides, scheduling scheme is another important function of NTL. The process MSG_timer is the calculation of transmission time, which is shared by two channel processes, Real Time (RT) channel and Non-Real Time (NRT) channel. Once the time has expired, the outdate message will be discarded and inform the application layer that there is no time left for the retransmission. The key to control message transmission is the scheduling scheme that maintains the time table for each sending message with QoS variations and changes of application objects. It relied on the management of the micro duration that distributed in the process syn_scheduling and asyn_scheudling, where the former is the sorting algorithm for periodic traffic, while the latter is the arrangement of the polling sequence for non-periodic declaration. The process scheduling_EC configures the time length of syn_scheduling and asyn_scheduling in each macro cycle, which is defined as the updating period for the scheduling table. The scheduling results update the scheduling table and control the message flow from APP to NTL through the process SCH_trigger, which implements the hybrid time-triggered and event-triggered mechanism.

Finally, the processes of APP layer are working on the conflict-avoidance end-to-end transmission services provided by the transfer control in NTL. The process Node management works independently to provide the basic address information of scheduling and routing scheme in NTL. In the communication NCS, there exists three deterministic data types. The process periodic_tasks is the message queuing service to deal with the periodic field sampling data; while the process non_periodic_tasks is the queuing of the emergency data and the monitoring configuration data. The complement of sending process in all nodes (i.e., queue in the periodic_tasks are empty) would generate a periodic_task_FIN signal to start the sending and receiving process in the process non_periodic_tasks. On the data receiving way, all the received data are checked in the process dcl_tasks with the criteria defined in the user layer. If an error occurs, the re-transmission service will be issued before deadline. If a non-periodic declaration occurs, the information of occurrence intervals, source and important level will be formed as the DCL_MSG signal and then, be sent to NTL and process non_periodic_tasks. During the asynchronous phase determined by the process TM, a non-periodic message will be sent out once receiving its token, repeating till all the declared message are received, called the polling way. Then, the non-periodic queue space is released, a non-periodic_task_FIN signal is sent to the process periodic_tasks and a new macro begins.

Dynamic reconfiguration management framework: Based on the proposed reconfigurable protocol stack, the integrated management framework is the key to bridge the gap between the SDL formal space and the executable code space. The framework offers flexibility to build a protocol stack of dynamically loaded code components that provide system performance guaranteed transmission services for the heterogeneous and changeable environment. As shown in Fig. 3, the SDL based dynamic management framework implements code reconfiguration on the target protocol stack from four phases: performance evaluation, description optimization, SDL parser and code composer. Reconfiguration of the protocol stack (in nodes of NCS) is triggered by monitoring the environment or system structure changes. The network status monitor will invoke the

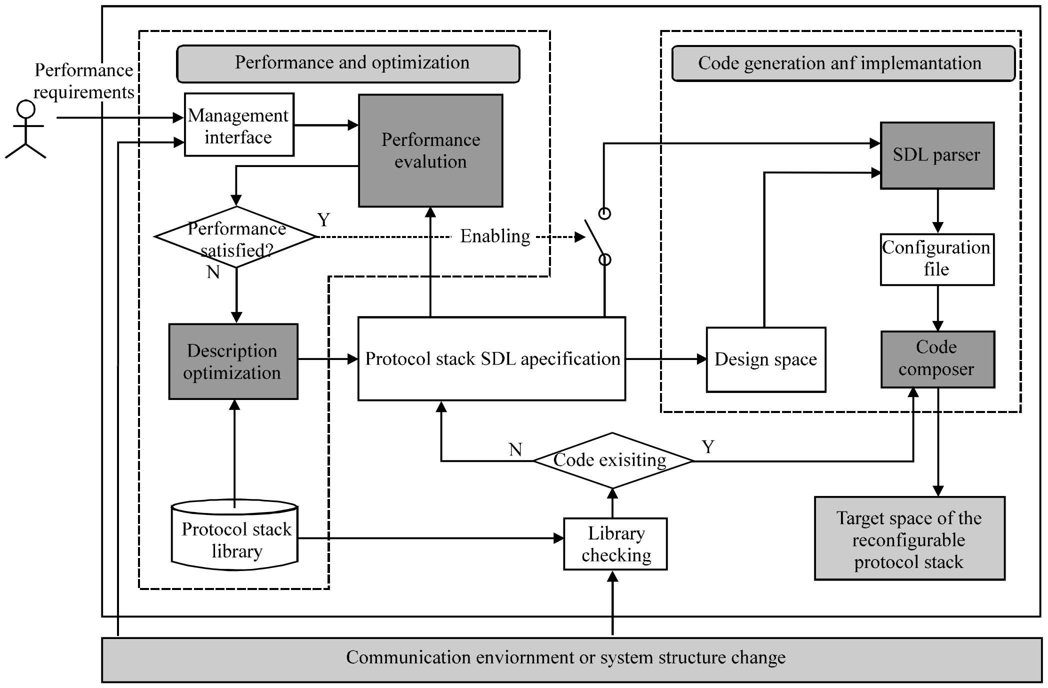


Fig. 3: Dynamic reconfiguration management framework

reconfiguration process for the new code to adapt with the new communication environment. If the required code version has already existed, the reconfiguration process is executed in a static way that the new version is directly downloaded into the object physical address. Otherwise, the dynamic reconfiguration process is triggered, which is divided into two phase: the first is the performance estimation and optimization phase; the second is the code generation and implementation phase.

During the first phase, the performance evaluation component gives estimation results on basis of the formalized architecture and environment models. These models represent the system constraints that affect the execution of protocol stack SDL specification, such as computation ability, resource availability and transmission delay. Comparing with the system requirements, the performance evaluation results could determine the object SDL models whether need to be optimized or not. If performance constraints are not satisfied, the description optimization component will fetch a new SDL process instance from the protocol stack library to replace the old one and enhance the related protocol functionality. The procedures of evaluation and optimization will be executed in a cyclic manner till the protocol stack specification meets system requirements.

As for the following second phase, the SDL specification that meets system constraints is converted to the design space by a rule of binary coding. The converted binary design space provides a support for the automated code generation in the SDL parser. The SDL parser reads the C code files from the SDL Specification and the protocol variation information from the design space and generates a set of individual configuration files (defined as the ELF format) for each declared reconfiguration target space. The code generation is automated with the help of the Telelogic TAU SDL suite. Then, these configuration files are packeted and physically transported over the network. Considering the code implantation on the target space, the Composer implements the self-switching scheme to manage the runtime swapping the old stack code with the new one by maintaining two link-list data structures and a memory ID counter. On receiving messages of configuration file, the composer stores each file segment in the flash address where actual reconfiguration takes place and invokes the new protocol functionalities during a set of specific idle time slices, which are scheduled with the guarantee that the normal operation mode would not be interrupted.

PERFORMANCE AND OPTIMIZATION OF SDL SPECIFICATION

Performance evaluation: The performance evaluation is to identify satisfactory configurations of the system. In our implementation, the self-evaluation mechanism is based on modeling the architecture and environment constraints for the running of protocol objects with the same SDL method. The runtime performance evaluation is implemented with interactions among three type models: the application model, the architecture model and the environment model. Our evaluation objects, the application models, are instances of the SDL protocol stack specification, representing a set of protocol functionalities of a single node. Architecture models are constructed as the running environment to represent the extra implementation-dependent information on the resource requirements of the SDL processes governing the protocol, as well as information on the underlying machinery. To stimulate the application model and the architecture model, the environment model is to generate different types of traffic load on each node and give the performance testing report simultaneously.

Architecture model: The architecture model can be divided into two types of sub-models reflecting platform-dependent constraints: the node constraints-processor model and the network constraints-connection model.

Processor model: It mainly contains three parameters-processing speeds, buffers of input and output and OS scheduling strategies. The processing speed of CPU is of obvious influence on the execution time for each protocol entity, which may be used to vary the computing power on different running platforms. The buffers are, of course, key elements in the context of asynchronous communication. The buffers of input and output are assumed to be of type no-wait-send, which means calculating the behavior delays of messages adding/removing in queues and assigning enough space without taking into account memory consumptions. Here, the OS scheduling strategies are defined as the type of deterministic and shared, i.e., communication activities of reconfigurable protocol stack are processed under a deterministic conflict-free scheduling principle and reconfiguration activities shared the idle bandwidths with communication tasks.

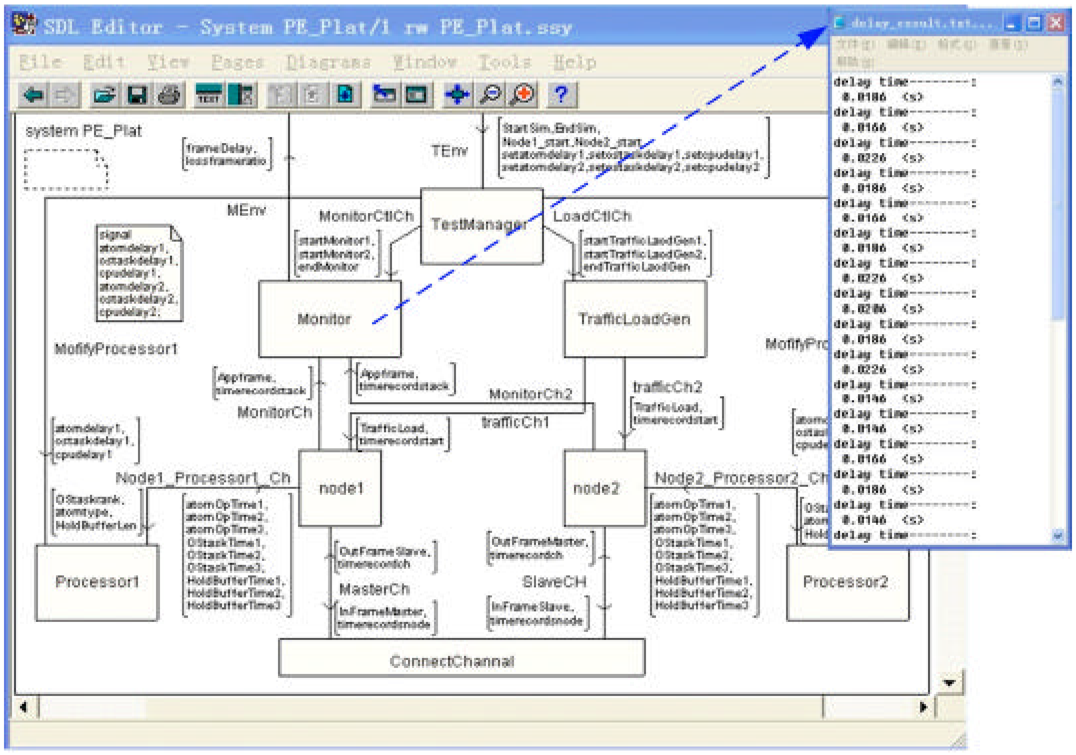
Connection model: It mainly contains three parameters-degree of parallelism, system structure, channel delay and packet loss rates. Parallelism in the execution of the protocol functions is to represent the capability of serializing actions by running all protocol stacks (i.e., application models) on a number of nodes. The system structure determines the execution sequence of the message exchanging activities among nodes. The channel delay and the packet loss rates are the reflection of point-to-point events that impediments the data flow in the protocol stack.

Environment model: The environment model is consisted of three sub-models for controlling the evaluation course: the traffic load generator, the performance monitor and the test manager. According to different types of application objects, the traffic load can be varying on each node. Flexibility and robustness can be evaluated through testing with different configuration on the load generator. Within a specific architecture, the performance of the protocol stack is evaluated under different assumed generation rates of periodic and non-periodic messages. During the course of performance evaluation, the performance monitor gives runtime prediction reports and the test manager is to support varying the testing conditions or stimulus for different evaluation objects.

Evaluation methodology: To evaluate the network performance, we should first construct the corresponding simulation platform with SDL tools. We have defined block types to specify the basic behavior and data of the network elements such as nodes and links. The modification of particular specifications (e.g., parameters configuration) of each network element can be easily accomplished by inheriting and redefining the basic block type when the simulation scenarios are built. Note that in the final simulation platform, we should create a specific program component to snap and process the desired simulation data from the SDL process Monitor. As shown in Fig. 4, a point-to-point evaluation scenario is deployed, where the node block is a complete protocol stack model and other design-independence blocks represent system constraints or variations. The evaluation result is write into a log file with the self-defined evaluation object, for example delay result.txt.

Description optimization: The description optimization is the process of reconstructing a new SDL description when the performance was not satisfied. The formalized SDL specification is described by a set of extended state machines to control the data flow and a number of precise procedures associated with protocol behaviors. Here, our description optimization is focused on procedures, which could be optimized by composing different process instances in sequence and concurrent manner.

- Sequential composition is deployed in a static way that supports minor optimization in the SDL process. Such a composition contains choice elements for the enumeration of protocol possible execution procedures, which the choice condition is set by the external performance prediction signals. When the protocol performance is detected to be inefficiency, a guard behavior is used for describing preconditions that prefix the execution of new processes and a


SDL Editor - System PE_Plat/1 rw PE_Plat.ssy
File Edit View Pages Diagrams Window Tools Help
system PE_Plat
frameDelay, loadframeratio
TEnv
StartSim,EndSim, Node1_start,Node2_start, setatomdelay1,setostaskdelay1,setcpudelay1, setatomdelay2,setostaskdelay2,setcpudelay2
MEnv
MonitorCtlCh
TestManager
LoadCtlCh
signal atomdelay1, ostaskdelay1, cpudelay1, atomdelay2, ostaskdelay2, cpudelay2;
startMonitor1, startMonitor2, endMonitor
startTrafficLaodGen1, startTrafficLaodGen2, endTrafficLaodGen
Monitor
TrafficLoadGen
MofifyProcessor1
Appframe, timerecordstack
MonitorCh
MonitorCh2
trafficCh2
trafficCh1
TrafficLoad, timerecordstart
atomdelay1, ostaskdelay1, cpudelay1
Node1_Processor1_Ch
node1
node2
Node2_Processor2_Ch
OStaskrank, atomtype, HoldBufferLen
atomOpTime1, atomOpTime2, atomOpTime3, OStaskTime1, OStaskTime2, OStaskTime3, HoldBufferTime1, HoldBufferTime2, HoldBufferTime3
OutFrameSlave, timerecordch
OutFrameMaster, timerecordch
MasterCh
SlaveCH
InFrameMaster, timerecordsnode
InFrameSlave, timerecordsnode
Processor1
Processor2
ConnectChannal
delay_result.txt...

```
delay time---------:
 0.0186  (s)
delay time---------:
 0.0166  (s)
delay time---------:
 0.0226  (s)
delay time---------:
 0.0186  (s)
delay time---------:
 0.0166  (s)
delay time---------:
 0.0186  (s)
delay time---------:
 0.0226  (s)
delay time---------:
 0.0206  (s)
delay time---------:
 0.0186  (s)
delay time---------:
 0.0226  (s)
delay time---------:
 0.0146  (s)
delay time---------:
 0.0146  (s)
delay time---------:
 0.0166  (s)
delay time---------:
 0.0186  (s)
delay time---------:
 0.0146  (s)
delay time---------:
```

Fig. 4: A SDL-based performance evaluation scenario (2 nodes, point to point performance)

disabling behavior is used for reestablishing the output signals

- Concurrent composition suits a plug-and-play approach where the SDL processes for replacement are designed off-line and deployed dynamically. Concurrent means there is a replaceable process running and verified off system and based on this analysis and the specified performance aspects (performance requirements, resource requirements of the specification, available resources, etc.), the most appropriate implementation process is selected for the different parts of the SDL specification

Therefore, no matter sequential or concurrent composition, process is the basic element for protocol optimization. The management framework only need the information of which process has been modified in the optimization, instead of what extent of modification has been made on it.

CODE GENERATION AND IMPLEMENTATION OF SDL SPECIFICATION

SDL parser: In our management framework, the SDL parser is responsible for governing the code generation course and generating the protocol configuration file. Determining the new configuration is a searching problem in the SDL formal space. The exploration of the formal space is a challenging problem since it must be performed within stringent time bounds and resource constraints. Moreover, to compile the whole protocol stack as a single file for the code reconfiguration in the target space is normally considered as a high cost way and is not suitable for being transferred through network. Thus, exploring the SDL protocol stack specification and extracting the functional inefficiency parts are big challenges for optimizing the dynamic code reconfiguration process. In this section, the design space exploration and generation of configuration file are discussed.

Design space exploration: Design space exploration involves: coding the formal file system and finding the files that are need to be reconfigured. As shown in Fig. 5, all the processes of the whole protocol stack file system can be further organized into a tree structure with three degree depth. The root is the whole protocol stack system and the leaf represents the process. Models on each degree of the tree structure are assigned by a set of 3 bit binary codes except for the root node and based on this

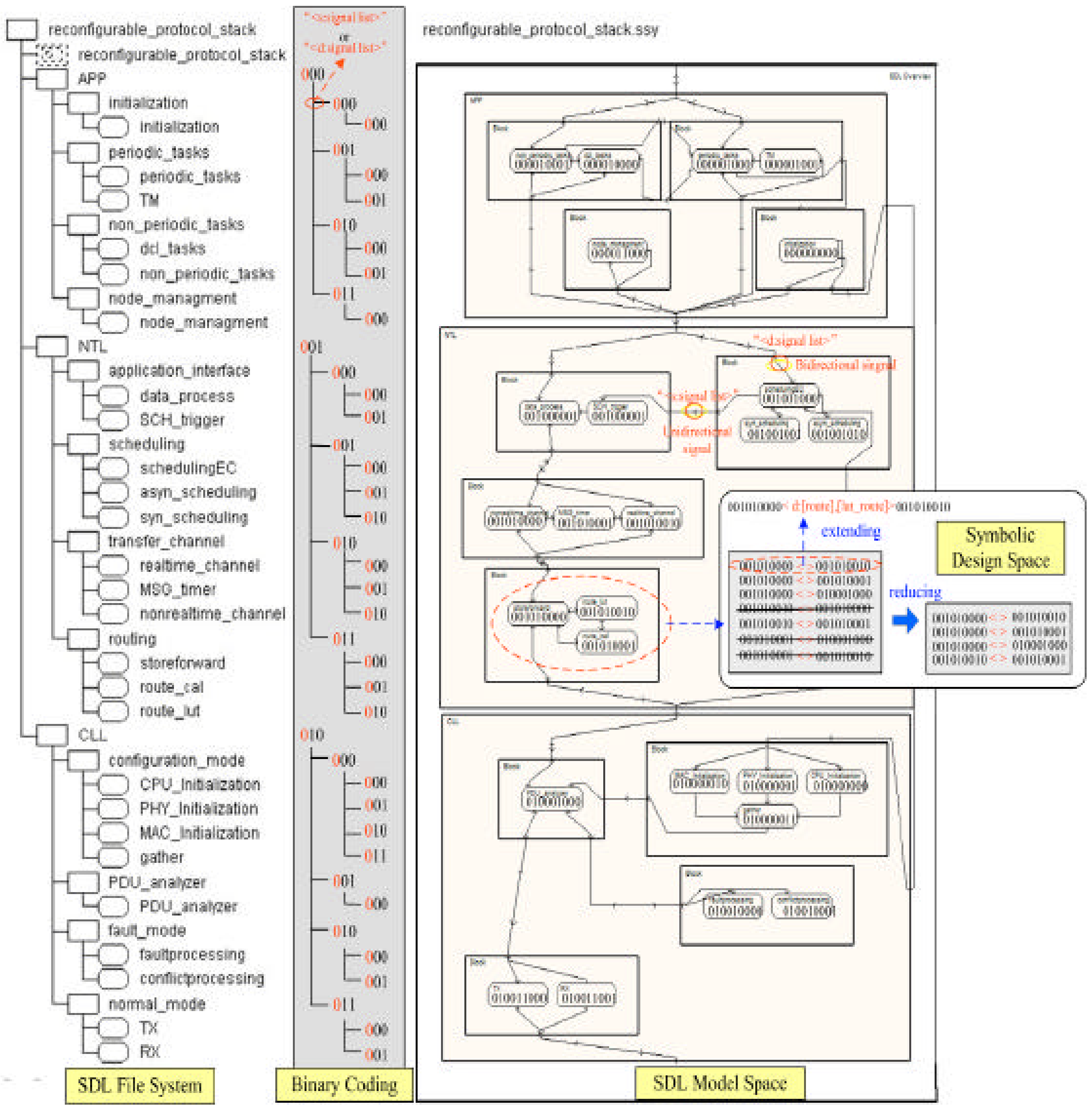


Fig. 5: Mapping the SDL space with the symbolic design space

rule, a unique combination code can be easily obtained for leaf node-SDL process model, such as 000000000 for the Initialization process. For each code, the 3*n bit (n = 1,2,3) are used to denote whether the corresponding models are modified during description optimization. If some modification happens, these special bits will be set to 1. For construing a reduced symbolic design space, the coding process is conducted by the rule that the former process model only searches its latter relationship on ascending order. The relationship between two SDL processes is denoted by the symbol < >, which can be extended to <s[d]: singal list> depended on the signal direction and content. We can quickly locate the change parts of the file system by comparing new design space with the old one. Alternatively, the configuration parts can also be located by the special bits on the pre-order code, following, by partially comparison on their post-order code, other related files can be found out.

Generation of the configuration file: The protocol configuration file is generated from the reconfiguration parts of formal specifications and designed for alleviating the burden of transmitting reconfiguration tasks on network and loading code into the target space. It is composed of three elements: interaction, message and structure. Comparing with the design space that deals with the reconfiguration problem in pure logical aspects of platform-independent, the configuration file is focus on providing the platform-dependent information for implementing code linking and executing, called physical reconfiguration. Such a platform-dependent description file could be formatted as the standard HEX or ELF, where the former is normally widely applied for the applications on OS-free platforms while the latter is the most prevalent exchanging file format for UNIX based platforms.

The mapping principles to interpret physical configuration file with logical design space can be described as: interactions are represented through composing and extracting lines with the same binary code, messages are the signal lists between the two SDL process codes of each line and structures information can be allocate to specific corresponding SDL descriptions by the unique binary codes. In this way, the reconfiguration information denoted in the design space will be mapped into the modification of addressing or composition information in the configuration file, such as the segment construction of ELF (content of segment .data, .text and elf head).

In order to get instances of configuration file, the related C codes that denoting the reconfiguration parts of protocol stack have to be generated. Supporting by the Telelogic TAU SDL suite, there are two modes for automated generating code, namely light integration and tight integration. Here, tight integration that supports independent execution on the SDL process level is the better choose, for reasons that an operating system is used for dynamic interpreting and loading ELF files and the processes distributed in the formal space are expected to work in the multiple-thread concurrent execution manner.

Code composer: Once the protocol configuration files arriving at the target space, the code composer is the governor for embedding the new code. The code composer can acted in a static or dynamic reconfiguration manner. In this section, our emphasis is on the dynamic reconfiguration to complete the working flow of the whole SDL based management framework. The key to the composer implementation is to make a self-switching scheme for the scheduling of code loading operations, such as assembling, linking and swapping.

Self-switching scheme: Communications between nodes of NCS are governed by a specific network scheduling scheme in our reconfigurable protocol stack. Besides, it is common that there exists an amount of other application tasks which are executed in a parallel mode by the scheduler of embedded operation system (OS, or kernel) on the node platform, called task scheduling. Code loading operations on the target space is involving both network scheduling and task scheduling process. Either receiving a packet of configuration file, or sending a message of file resending request due to the packet loss, both these communication activities are dealt with as non-periodic tasks in the proposed protocol stack. And the linking and swapping operations of code loading is taken as application tasks being inserted into the scheduling queue of OS kernel and pended for its time slice. Generally, in our implementation, in order to guarantee that the communication activities of the reconfiguration independence functionalities are not being interfered, the communication tasks of reconfigurable protocol stack are taken as a whole scheduling object for the OS scheduler with the highest priority. In this way, the tasks related to the new code reconfiguration can only be scheduled in the idle time slice of protocol stack. Once the performance of protocol stack is detected to be inefficiency, the protocol stack will allocate time resources for code loading operations and switch itself to the new communication state through retrieving the time resources with executions of new code loading. In our previous work, the online Genetic Algorithm (GA) based reconfigurable scheduling for NCS are discussed. Here, all the code loading operations are considered as controller tasks that are scheduled between the message sending and receiving tasks in this GA-based methodology (Chen *et al*., 2009).

Code loading operations:

- Assembling operation: the configuration file has to be segmented into different packets if the target space and the model space are not in the same machine, due to the limited packet size. In this case, the target space should assemble these file segments according to the information of ELF head file before linking to the actual physical space
- Linking operation: the new declared data and functions are two main objects of linking operations. They are included in the segments of .data and .text of ELF file separately. When a new configuration file, i.e. ELF file, is generated. The address and size information of the actual target space is empty to the related .data and .text segment. On parsing the .rela segments of ELF, the actual available address in the target space can be extracted to renew .data and .text segment. According to the ELF standard, each .rela is corresponding with a specific segment, such as the rela.text to .text segment

- Swapping operation: existing operation systems have provided API functions for loading the ELF file to complete the swapping operations, which is to replace the inefficiency process with the new code. For example, the SOS operating system (Han *et al*., 2005) provides swapping operation functions for registering the new code module and releasing the unused memory space, such as loader_handler(), handler_loader_is_on_node() and handle_fetcher_done()

CASE STUDY

Here, we give a practical scenario to illustrate the application of our proposed management framework for the reconfigurable protocols stack in NCS. Our system test-bed consists of six ARM-Linux slave devices and a Windows XP master station. Such a hero-type hardware test bed overcomes the limited computing capability on the embedded system. Thus, complicated procedures, such as the interfacing with users, the design space exploration, the SDL specification evaluation, optimization and parser, could be implemented in the master node. Our experiment result is OS constrained, with the help of code locating and loading function provided by ELF file and ARM-Linux system. The slave system (ARM-Linux) is performed as a network status reporter and responser to commands or configuration files from the master.

Specifically, the practical experiment test-bed is mapped into the SDL performance evaluation architecture (Fig. 6). We assume following running conditions: a CAN-based reconfigurable protocol stack employed an FTT-CAN scheme (Pedreiras and Almeida, 2002) to guarantee continued real-time operation under dynamic communication requirements. An extended 29-bit CAN frame format is adopted and the baud rate of the CAN network is 500 Kb sec^{-1}. The number of online slave nodes is 5 and the sixth slave node will enter the channel as a burst periodic traffic to the scheduling scheme. The data for maintaining the address table in node management (a SDL process) is characterized with the non-periodic non-real time traffic. The initial configuration for the synchronous scheduling algorithm (in process syn_scheudling of SDL specification) is Rate-Monotonic (RM) and an alternate Earliest Deadline First (EDF) algorithm is stored in the protocol library. Therefore, the performance of one process reconfiguration under the proposed management framework could be tested.

In Fig.7, the process syn_scheudling is implemented in the protocol stack of master node to sort the periodic tasks (generated in slaves) for the synchronous phase. After the performance evaluation process starts, this

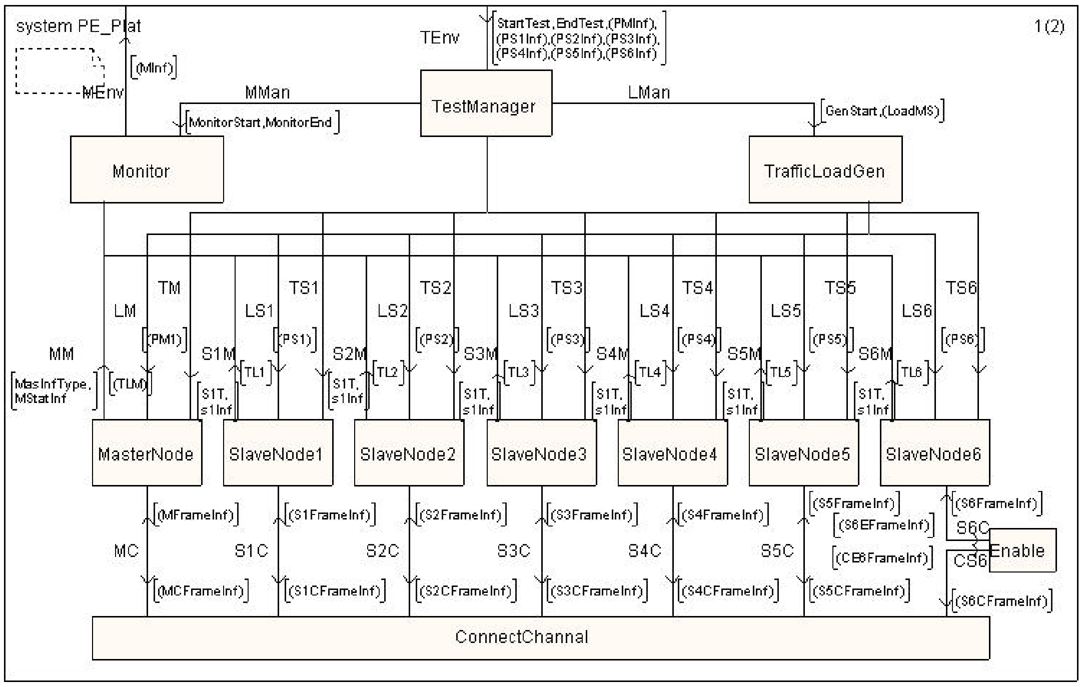
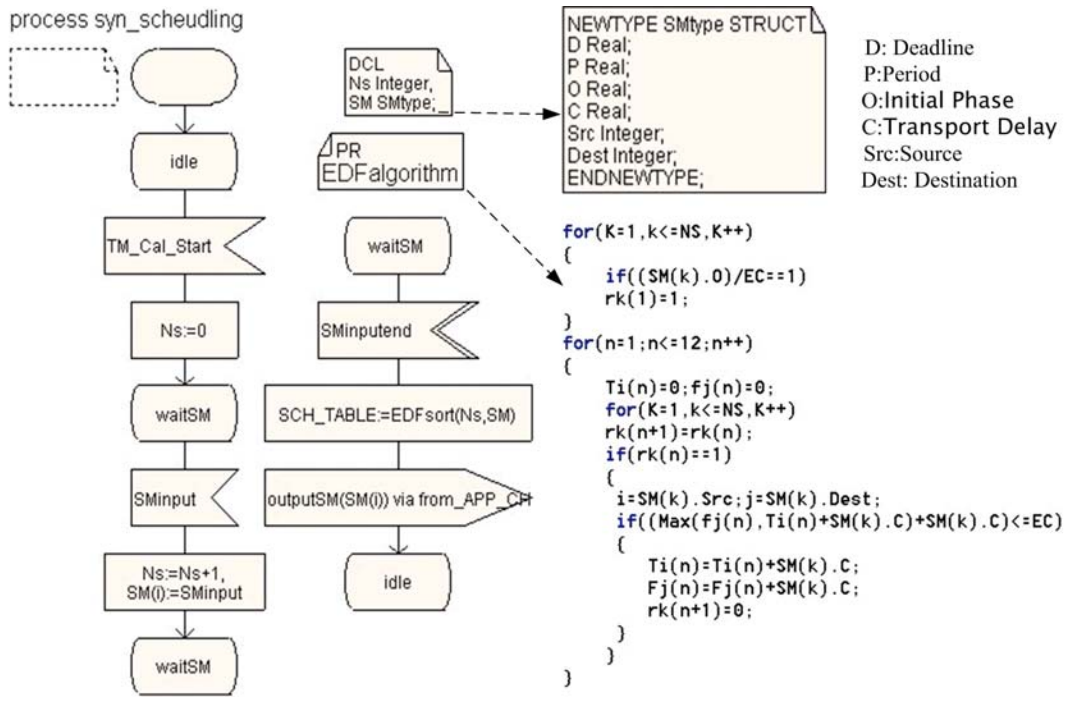


Fig. 6: Performance testing scenario for CAN system



```
DCL
Ns Integer,
SM SMtype;
```

```
NEWTYPE SMtype STRUCT
D Real;
P Real;
O Real;
C Real;
Src Integer;
Dest Integer;
ENDNEWTYPE;
```

```
for(K=1,k<=NS,K++)
{
    if((SM(k).O)/EC==1)
    rk(1)=1;
}
for(n=1;n<=12;n++)
{
    Ti(n)=0;fj(n)=0;
    for(K=1,k<=NS,K++)
    rk(n+1)=rk(n);
    if(rk(n)==1)
    {
     i=SM(k).Src;j=SM(k).Dest;
     if((Max(fj(n),Ti(n)+SM(k).C)+SM(k).C)<=EC)
     {
        Ti(n)=Ti(n)+SM(k).C;
        Fj(n)=Fj(n)+SM(k).C;
        rk(n+1)=0;
     }
    }
}
```

Fig. 7: SDL description of the process syn_scheudling (in master node)

Table 1: Periodic tasks of slave nodes

Task No.	(D, P, O, C, Src, Dest)
A21	(2,2,0.5,0.3,2,1)
A22	(2,2,1,0.3,2,1)
A23	(2,2,1.5,0.3,2,1)
A31	(3,3,0.5,0.3,3,1)
A32	(3,3,1,0.3,3,1)
A4	(4,4,1.5,0.2,4,1)
A5	(5,5,0.6,0.2,5,1)
A6	(6,6,2,0.2,6,1)
A7	(6,6,3,0.1,7,1)

process will gather periodic transmission requirements from slaves at runtime. Each requirement is a six-tuple structure data: SMx = (D, P, O, C, Src, Dest), representing the basic information that Deadline, Period, Initial Position, Processing Delay, Source Address and Destination Address. Inputting periodic task requirements, the scheduling algorithm EDF can generate a scheduling queue to complete the synchronous part of TM. The EDF algorithm is implemented by a call for C function, which can be of RM algorithm. During the performance evaluation, periodic traffics of slaves are governed by the receiving TM from master node.

In our scenario, the master node is numbered as A1 and the following Ai (I = 2, 3, ..., 7) is to denote the slaves. Thus the periodic tasks in each slave can be assumed as in Table 1, where the deadline D of each task is equal to its sampling period P, a multiple of the Element Cycle (EC). Figure 8 and 9 snapshot the scheduling result due to RM algorithm and EDF algorithm separately. The red dot denotes the interval of a message issued, the white dot indicates this message is delayed in the EC and the black block represents the Processing Time of task. In Fig. 8, the message A21, A22, A23, are scheduled in its EC before their deadlines by the RM algorithm, A31 is also scheduled after one EC delay. However, due to fixed priority mechanism of RM algorithm, the low priority tasks A32, A4, A5 are unable to be scheduled in its EC before their deadlines. Comparing with RM, EDF algorithm would adjust the priority of tasks that have the earlier deadlines and could be completed in the idle time belonged to high priority tasks. Figure 9 shows that the messages can all be scheduled by EDF before their deadlines, even when task A7 (of the 6th slave node) was inserted at the beginning of 3rd EC (according to the Initial Position of A7). In all, the evaluation results in our management framework shows that the implementation of EDF scheduling on CAN allows higher channel utilization factors than the original RM scheduling on FTT-CAN.

Mapping with the optimization on the process syn_scheduling, its related symbol in the design space is formatted as 001001110. The 3rd bit 1 indicates that a single process reconfiguration happened and there is no need to compile its related processes, since blocks for protocol layers are stable (the 6th and 9th bit are remain 0). To generate the C file, the Target Expert provided by

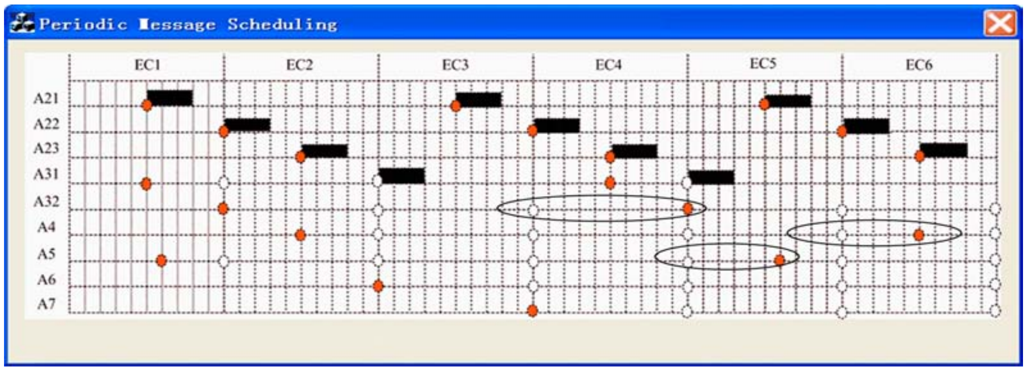


Fig. 8: Schedule using the RMS approach

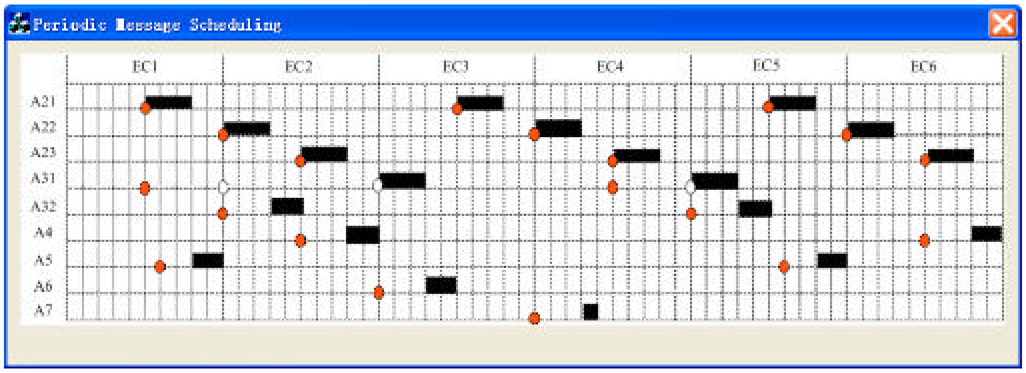


Fig. 9: Schedule using the EDF approach

the Telelogic TAU suite is defined as #ifdef <ARM_THUMB> construct in the ml_mcf.h file. Then, the code is compiled to be the configuration file with the standard arm-elf-gcc cross-compiler, distributed as the non-periodic and non-real time tasks on network and dynamic executed on slave processors ARM S3C2440. Thus, reconfiguration execution time is measured between the Start signal of enabling the Target Expert (indicating the start of the reconfiguration execution) and the Stop signal of loading mechanism (indicating the end of the reconfiguration execution). It can be calculated following the Eq. 1, which consists of three parts: the code generation time, the code distribution time and the code loading time. They are determined by various issues such as size of source file (.pr file), efficiency of compiler, congestion status of network and processor capability.

$$T_{\text{reconfiguartion}} = T_{\text{generation}} + T_{\text{distribution}} + T_{\text{loading}} \quad (1)$$

The system clock accuracy in the measurement was 10 m sec. The measured time results are summarized in Table 2. In our testing scenario, a single network segment consists of 7 nodes, the execution time of code generation and loading process is notable 85% of the total execution time for reconfiguration. These two processes are executed offline (i.e., computed in an independent computing unit) without having impact on communication activities of the old protocol stack during reconfiguration execution period. The distribution the new code is scheduled as the non-periodic non-real time traffic with assigning a fixed time slice in each macro scheduling cycle. In this way, the impact on the real-time traffic could be controlled on a low level. Thus, though the transmission delay is very small, the average value 1370 msec for the distribution time is caused by the postponed sending in each scheduling cycle. Similarly, when communicating with a heterogeneous protocol, i.g., Ethernet, all processes of CLL layer in related nodes have to be replaced with Ethernet. Because the size of SDL

Table 2: Time cost for the reconfiguration of one process (unit: millisecond)

	Time cost			
Measurement	Generation	Distribution	Loading	Total
Min/Max value	2760/3280	760/790	1360/1380	4880/5450
Average value	3114	775	1370	5259

Table 3: Time cost for the reconfiguration of one layer (unit: millisecond)

	Time cost			
Measurement	Generation	Distribution	Loading	Total
Min/Max value	21960/25280	6220/6490	9920/11040	39100/48070
Average value	23104	6375	11000	44269

description file for CLL layer is related to a number of processes, the time cost of reconfiguration is nearly 8 times than the one process reconfiguration (Table 3).

CONCLUSIONS AND FUTURE WORK

The dramatic growth of networked control system confronts designers with serious difficulties of distributed infrastructure complexity and communication environment heterogeneity. It has motivated the call for reconfigurable protocol stack that can adapt to the environment changes. In this study, an integrated management framework was presented for the implementation of reconfigurable protocol stack in NCSs. Benefiting from SDL, modification and refinement on the protocol stack design could be carried out without worrying the incompatibility between the initial design and the final implementation. Within the management framework, a binary symbol space was designed for exploring the SDL specification files and the pure SDL performance evaluation approach was implemented. They both performed as a seamless way when integrating with the SDL specification and served as the foundation for the optimized code execution on the target space. A SDL process reconfiguration experiment showed that the management framework can work well to meet the new tasks transmission requirements added in.

Large scale and complex network is another big challenge for the protocol stack design in NCS. In the future, taking the methodology proposed here, we will go further study on the optimizing the SDL implementation through intelligent algorithms, for instance, applying the genetic algorithm to speed up the rerouting procedure when the network scale covering hundred nodes. Additionally, a rich library of SDL performance models needs to be implemented for more network testing scenario, especially the mesh topology model that can simulated the dynamic convergence and nondeterministic behaviors in complex network.

ACKNOWLEDGMENT

This study was supported by the National Natural Science Foundation of China (No. 60674081 and No. 60834002) and the International Science and Technology Cooperation Project (No. 2008DFA12150).

REFERENCES

An, L., H.K. Pung and L. Zhou, 2006. Design and implementation of a dynamic protocol framework. Comput. Commun., 29: 1309-1315.

Bridges, P.G., G.T. Wong, M. Hiltunen, R.D. Schlichting and M.J. Barrick, 2007. A configurable and extensible transport protocol. IEEE/ACM Trans. Networking, 15: 1254-1265.

Chan, K. and G.V. Bochmann, 2003. Modeling IETF Session Initiation Protocol and its Services in SDL. In: SDL 2003-Systrm Design, Reed, R. (Eds.). LNCS. 2078, Springer-Verlag, Berlin Heidelberg, ISBN: 978-3-540-40539-9, pp: 159.

Chen, H., C. Zhou and W. Zhu, 2009. A hybrid neural-genetic approach for reconfigurable scheduling of networked control system. Proceedings of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation, June 12-14, Association for Computing Machinery, Shanghai, China, pp: 33-38.

Cheng, L., L. Kong, C. Ma and X. Zhu, 2007. Stability and maximum delay bound of networked control systems with multi-step delay. Inform. Technol. J., 6: 780-783.

Fischer, J., T. Neumann and A. Olsen, 2005. SDL code generation for open systems. Proceedings of the 12th International SDL Forum, SDL 2005: Model Driven, June 20-23, Springer Verlag, Grimstad, Norway, pp: 313-322.

Han, C., R. Kumar, R. Shea, E. Kohler and M. Srivastava, 2005. SOS: A dynamic operating system for sensor networks. Proceedings of the 3rd International Conference on Mobile System, Applications and Services, June 5-7, Washington, USA., pp: 163-176.

Hannikainen, M., A. Takko, J. Knuutila, T.D. Hamalainen and J. Saarinen, 2000. SDL-to-C conversion for implementing embedded WLAN protocols. Proceedings of the 26th Annual Conference of the IEEE Electronics Society IECON 2000, Oct. 22-28, IEEE Computer Society, Nagoya, Japan, pp: 2455-2460.

Jabri, I., A. Soudani, N. Krommenacker, T. Divoux and S. Nasri, 2008. QoS protocol specification for IEEE 802.11 WLAN. Inform. Technol. J., 7: 549-559.

Kolla, S., D. Border and E. Mayer, 2003. Fieldbus networks for control system implementations. Proceeding of the Electrical Insulation Conference and Electrical Manufacturing and Coil Winding Technology Conference, Sept. 23-25, Bowling Green State Univ., OH, USA., pp: 493-498.

Min, H., J. Heo, Y. Cho, K. Lee, J. Son and B. Song, 2008. A module management scheme for dynamic reconfiguration. Proceedings of the International Conference on Computational Science and Its Applications, June 30-July 3, Springer Verlag, Perugia, Italy, pp: 820-828.

Muhugusa, M., G. Di Marzo, C. Tschudin, E. Solana and J. Harms, 1995. Implementation and interpretation of protocols in the ComScript environment. Proceedings of the 1995 IEEE International Conference on Communications, June 18-22, IEEE, Seattle, WA, USA., pp: 379-384.

Niamanesh, M. and R. Jalili, 2009. DRAPS: A framework for dynamic reconfigurable protocol stacks. J. Inform. Sci. Eng., 25: 827-841.

Pedreiras, P. and L. Almeida, 2002. EDF message scheduling on controller area network. Comput. Control Eng. J., 13: 163-170.

Telelogic Inc., 2000. Telelogic Tau 4.0. Telelogic AB, Malmo, Sweden.

Tool Interface Standards (TIS) Committee, 1995. Executable and linking forma. (ELF) Specification, Version 1.2. http://en.wikipedia.org/wiki/Executable_and_Linkable_Forma.

Xia, F., Z. Wang and Y.X. Sun, 2004. Integrated Computation, Communication and Control: Towards Next Revolution in Information Technology. In: Intelligent Information Technology, Das, G. and V.P. Gulati (Eds.). LNCS. 3356, Springer-Verlag, Berlin Heidelberg, ISBN: 978-3-540-24126-3, pp: 117-125.

Yang, Y., L. Yang and L. Xiaokang, 2005. SDL-based network performance simulation. Proc. SPIE Int. Soc. Opt. Eng., 6022: 60222O.1-60222O.8.