The application of component dynamic scheduling technology in the automotive network control

Bin Sun[a]

Industrial Center,Nanjing Institute of Technology,211167,Nanjing,Jiangsu Province,China

Abstract. This paper introduces the component dynamic scheduling method. The method is used for automotive network control software design. In this paper, we introduce the research necessary for component-based software architecture. And we describe the rules of components dynamic scheduling based on graph theory and give a group of mathematical expression for the scheduling rules which can be used for matrix program. Based on these discussions, we introduce the method how to use this matrix algorithm in component dynamic scheduling for automotive network control software. At the end of this paper, based on practical engineering, we descript the effect of this algorithm.

Keywords: component scheduling; automotive; network control.

1 Introduction

With the development of automotive industry, the vehicle is using more and more microprocessors, and the microprocessor's internal structure has becoming increasingly complex, along with a corresponding increasing in software complexity. As a result, software development cycle is becoming longer day after day. It’s necessary to reduce the develop cycle by component technology. On the other hand, the vehicle control is developing from the original wiring harness control to bus control. CAN Bus, FlexRay Bus is widely using in the vehicle, which also needs a higher response rate for each processor on the bus. The Component technique can reduce the unnecessary software overhead; improve software efficiency to enhance processing speed. Third, unnecessary software of modern embedded electronic technology, we want to faster implementation of software system upgrade by component dynamic scheduling technology, without affecting the original system performance. In this situation, this paper presents a component-based software thread-level dynamic scheduling method. And the matrix algorithm also used in embedded software components scheduling for the program of this theory in this paper. This method can enhance not only the software process speed but also the system response speed to ensuring the stability of the system updates. The algorithm in this paper can be used not only in the field of automotive software design, but also can be used in other embedded systems.

[a] Corresponding author : binbin0011@163.com

2 Component-based software architecture

2.1 Automotive Electronics Components

In the computer field, the component is defined as a package which can be assembled to be an effective unit [1-3].

In the current research or applied embedded component model, they mainly customized for application-specific, many of which are improvement and cutting based on a common component model. Such as the Koala [4, 5], the PECOS (Pervasive Component Systems) [6], Rubus [7]. PECT (Prediction Enabled Component Technology) [8, 9], SaveCCM [10], AUTOSAR [11] and so on.

AUTOSAR is short for -“AUTomotive Open System Architecture”, which defines a set of supporting distributed, feature-driven software development methods and automotive electronic control unit software architecture standardization program for application to different vehicle and platform, to improve software reuse, reduce development costs.

2.2 Component of the thread and process

As shown in Fig.1, program composed by multiple components, and the program in memory organized in the form of process. A complete process consists of one or more threads: For multi-core processors, at the same time, a core can only run one physical thread. A component can encapsulate one or more threads, and is running as the form of thread at runtime. As the basic unit of scheduling, multiple threads in a component can only run on one processor core, in other words, if a program process together by the dual-core, we generally will regard as two components, multi-core is also. This parallelism and scheduling of thread-level component can plays a role for load balancing or facing special processor core. A user thread can be scheduled to run in a different processor core by OS, which is the basic OS feature for the use of multi-core processors by the OS scheduling policy decisions.

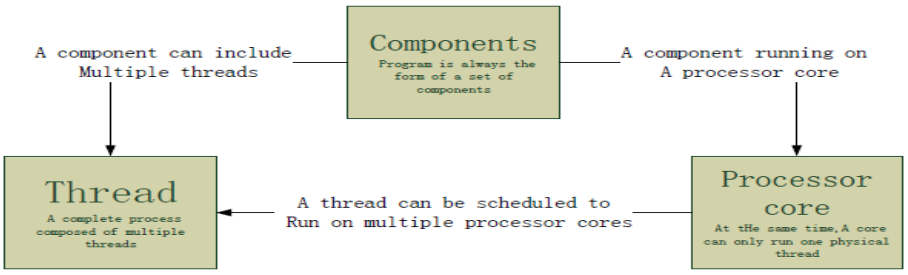

Figure 1. The Relationship between Components, Thread and the Processor core

3 Component scheduling

3.1 Schedulability of the component

CAPSULE[12] propose a thought:"We can packed with components, and the decision for parallel program run-time can be delayed until the thread runtime". The most important of this thought "run-time dynamic scheduling" is that we can use the ability to achieve the components dynamic changes. By replace components, we can change the software functionality. This way can meet the full use of limited resources in embedded software system, thereby reducing the energy consumption of electronic products.

For discussion of the components scheduling, we need a support system. One is an support for operation system, the other is support for multi-core communication support. Complete support system for component dynamic scheduling process as shown in Fig.2:

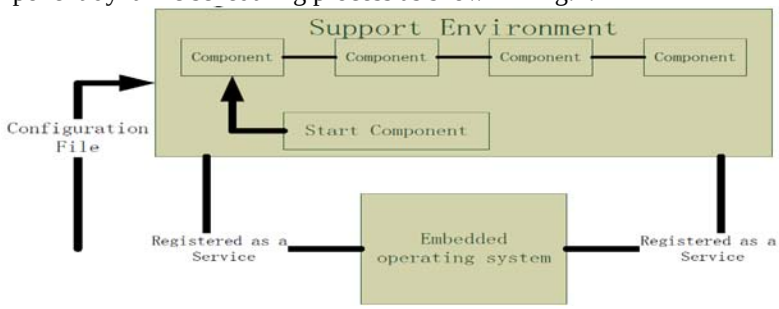

Figure 2. Components Support with The Operation System

In Fig.2, the relationships between "start component" and other components, we can be graphically as shown in Fig.3:

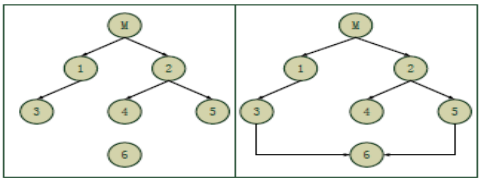

Figure 3. Schematic Drawing of the Dynamic Components

In Fig.3, M is the "start components", Point 1 to 6 is expressed as different components, this model can be extended to the thread-level components which is the smallest particle size in system scheduling, and these thread-level components also presence calling relationship between each other. For Fig.3(a) , $M \to 1 \to 3$ is a dynamic calling relationship, $M \to 2 \to (4,5)$ is another group of dynamic calling relationship. For Fig.3(b), $M \to 1 \to 3 \to 6$ and $M \to 2 \to (4,5 \to 6)$ are groups of dynamic calling relationships. This dynamic calling relationship can be compiled by the build system to collate and summarize the relationship as a component scheduler configuration files to be called by operating system.

The component scheduling principles are:

Conclusion 1: For a component, its premise of unloaded is that its connection number with other components is less than 2.

Conclusion 2: For a component, when it's unloaded, the associated component which should be unloaded is that the component is only called by this component.

Combined with knowledge of graph theory, we will make each component as a "vertex", and we can call the relationship with each other as "edge". These above conclusions can be expressed as:

Conclusion 3: Considering the components calling relationship as a directed graph (digraph), for each vertex(Vi), when it's unloaded, the vertex(Vj)which related to this vertex(Vi)'s out-degree (ODvi), and the vertex(Vj)'s degree(Dvj<2), should be unloaded.

Conclusion 4: Considering the component relationship as a directed graph, for each vertex, the premise which can be unloaded is that the vertex degree (Dv) is less than 2.

These conclusions can be used for the judge of whom system components can be unloaded. The unload operation can release the appropriate resources. This can not only reduce system resource usage, but also can be used to update the system.

The actual connectivity figure for complex software architecture is much more complex than Fig.3, and the calling relationship is relatively complex. There is large number of complex calling relationship, As a result, we need a further discussion for the operating system level to dynamically load and unload components.

3.2 Components weighted scheduling

Typically, a fixed state priority scheduling methods, that is, a given relationship for each function calling is used in both automotive electronics and in other embedded applications. The components scheduling in this article discussed is that change such a fixed priority to a dynamic priority based on the operation system and support environment, which can making the system, can adjust with the external environment to more effective use of system resources.

A weighted graph is shown in Fig.4:

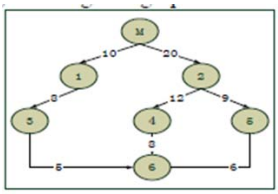

Figure 4. Component Weight Path Diagram

Definition 1: Weight—It means the call number between each node within a certain period of time (200ms in this article).

Because the operation system has a thread management mechanism such as time-slicing management or preemptive management, the calling number can reflect the frequency of the component used by the system.

Therefore, we can further handle as following: We can found in the Fig.4, $M \to 1$ weight is $w_{M1} = 10$, and $M \to 2$ weight is $w_{M2} = 20$, so $w_{M1} < w_{M2}$ .Therefore, although the component 1 may be loaded, and the default has higher priority, but the use number of components 2 is more. Therefore, the priority of the component 2 will be improved.

As a result, we can summarize the following components of scheduling rules:

Rule 1: In Component Weight Path Diagram, the component weight indicates that the number of calling by operation system in per unit time. The higher the weight, the higher its priority, correspondingly.

The above analysis is only one level. We can extend it to other scheduling relationship in Fig.4, thereby, adjusting the priority of related components.

That means, the calling number of component 6 should be 19 times. This frequency was even higher than the component 1, and then the priority of component 6 should be higher than the component 1.

Rule 2: In Component Weight Path Diagram, the number of the component calling time should be the sum of all the weight values. And the system should adjust the component priority based on the value of the sum, correspondingly.

Because the system is running in a dynamic external environment, the weight of the system needs to be dynamic update. And the weight maintenance needs the support environment to help, also. Because calculate instruction for total cost is simple, only the use statistics of components and update the component configuration file (using the assembler instruction set), the time required about a few less than dozens instruction cycles, so, compared to its role, the component dynamic scheduling is more favorable.

3.3 Component scheduling matrix algorithm

We can express the relationship between n components to be an adjacency matrix:

$$Dv_{ij} = \begin{pmatrix} a_{12} & & \\ \vdots & \ddots & \\ a_{n1} & \cdots & a_{n-1n-1} \end{pmatrix}, a_{ij} = \begin{cases} w_{ij}, if\ v_i \text{ connect to } \mathrm{v_j} \\ 0, \text{if } \mathrm{v_i} \text{ disconnect to } \mathrm{v_j} \end{cases} \quad (1)$$

For equation (1), the conclusion 3 and 4 can be expressed as:

Theorem 1: The Component (t) can be unloading depends on: $\sum_{i=0}^{n-1} Dv_{it} + Dv_{ti} < 2$

Theorem 2: When Component (t) is unloaded, if $\begin{cases} Dv_{it} \neq 0 \\ \sum_{m=0}^{n-1} Dv_{mi} + Dv_{mi} < 2 \end{cases}^{i \in (0,n)}$, the corresponding node of V_{it} should be unloaded simultaneously.

When we consider the weight value, suppose P_i as the priority of the component (i). For the Equation (1), we can describe Rule 1 as follows:

Rule 3: For the operation system schedule, max (Dv_{ij}) have min(P_i).

Rule 4: For complex scheduling, $P_i = \sum_{j=0}^{n-1} w_{ij}$.

4 Scheduling application software components

The above theory is a base for this practice, the practical effect of the application is as follows:

4.1 System hardware architecture

System in this article uses MC9S12XEP100 chip[13] as a processor. This chip is a heterogeneous multi-core architecture. And the chip has a processor core S12 and an XGATE co-processor core, where S12 is 16-bit CPU processor and support the expansion bus technology. XGATE is a 16-bit RISC co-processor and support a higher operation frequency, with the same interrupt and other functions as S12 processor. Scheduling software components.

As description above, the component dynamic scheduling has two purposes: one is updating the original components, another is adjusting operation efficiency in according with component frequency.

Component architecture M-AUTOSAR used in the system is an improved multi-core architecture based on AUTOSAR architecture. The operation system is a modified uC/OS-II operation system for dual-core, we using the component support developed by ourselves.

In this system, components 1 to 9 is list in following Table.1:

Table 1. Lists of The System Level Components

Task Name	Function	Priority
KeyOperation()	Keyboard Service	1
CAN0Interrupt()	High-speed CAN bus interrupt service	2
CAN4Interrupt()	Low-speed CAN bus interrupt service	3
TeenOperation()	Time related services	4
LightControl()	Lighting Control Service	5
CANTest()	Bus State Machine Services	6
K_Operation()	KWP2000 bus service	7
F_Operation()	FlexRay bus service	8
SelfTest()	Self-diagnosis services	9

Each component contains more than one thread. Following analyze the component “low-speed CAN bus interrupt service” (defined as the component O) as example, this component contains the Interrupt reception (thread A), Interrupt transmission (thread B), Clear the receive buffer (thread C), Clear transmit buffer (thread D), Data processing (thread E), Data transmission (thread F), Data receiver (thread G), a total of 7 threads. Its structure is shown in Fig.5.

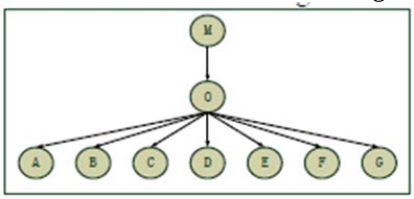

Figure 5. CAN0 Interrupt component model

Base on analysis of previous section, and we can conclude: A~G thread can be unloaded, and if component O is unloaded, we should unload the components of A~G in order to avoid program error.

4.2 Component update feature

Online component update feature requires the system finish receive for the corresponding program or components at first, and then pause and unload the component, and reload the corresponding components at last.

During verify, the system remains running, we use of low-speed CAN bus to write the program[14]. We modify some components in the system. The update of program is using CAN bus program burner developed by ourselves, as shown in Fig.6:

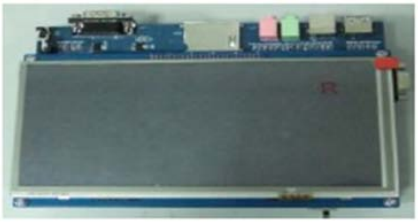

Figure 6. CAN Bus Program Burner

In the updating process, the processing flow chart on the motherboard is shown in Fig.7.

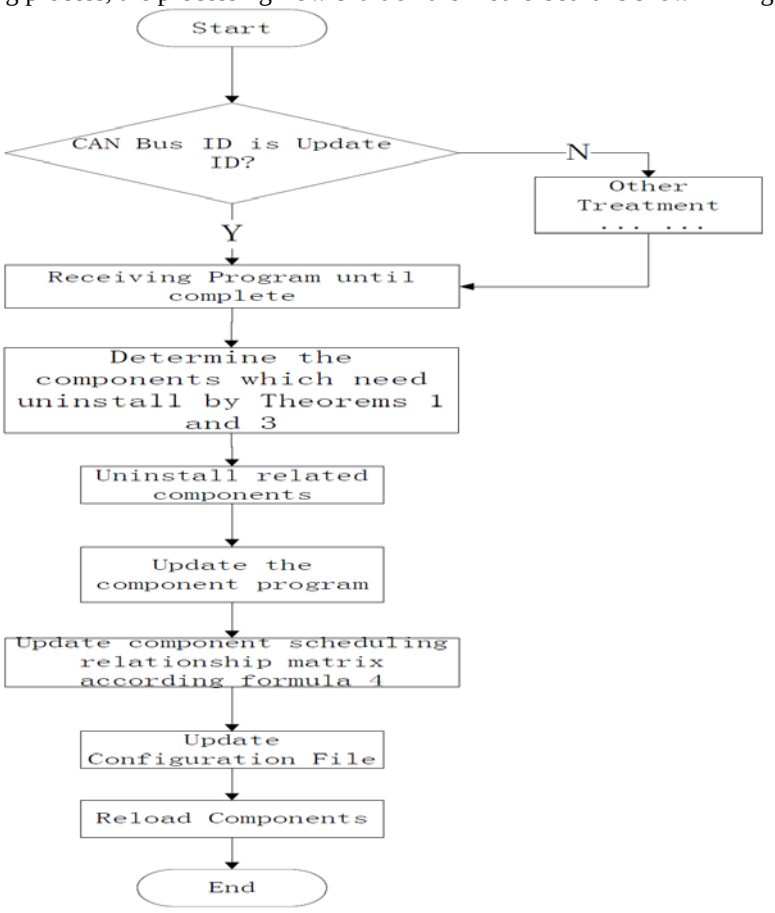

Figure 7. Update Flowing Chart

When we update through CAN bus, the burner is shown in Fig.8 as follows:

Figure 8. Program updating by CAN bus program burner

4.3 Component optimization

The second component scheduling functionality is reducing or increasing the priority of the component based on the frequency of the component. The system dynamic updates the component priority to improve the running speed. The process is as shown in Fig.9:

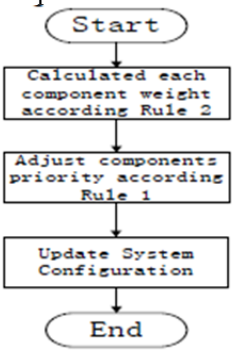

Figure 9. Flowchart of Optimization Components

In the actual system application, we make a statistical focus on a number of modules in the entire software system to schedule. Fig.10 is the statistical situation of K_Operation module occupies machine cycle. At the beginning, KWP2000 bus is not connected to the system. And we connect the KWP2000 bus at 10th seconds. At 15th seconds, we disconnected KWP2000 bus again. We can found in Fig.10, when KWP2000 bus is not connected, the module occupies gradual decline in the number of machine cycles. And when connected to the bus, the module is called more frequently, with greatly increasing of the number of cycles. Then, re-off KWP2000 bus, the number of cycles gradually is reduced again.

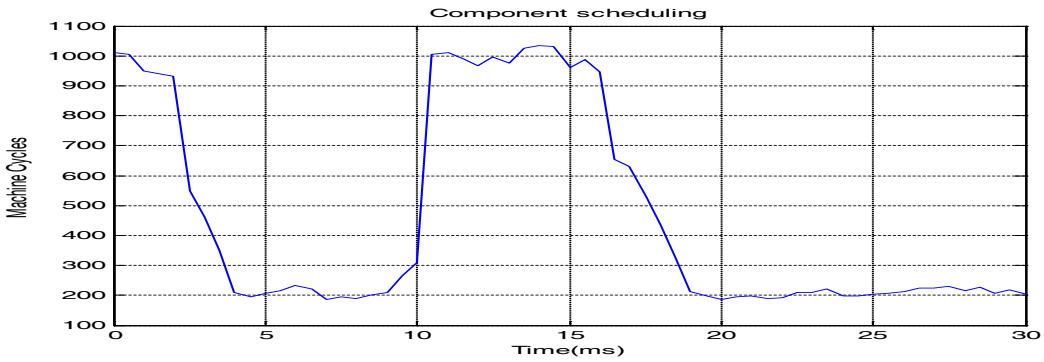

Figure 10. The Number of Machine Cycle for K_Operation by Component Scheduling

5 Summary

This article describes a component-based software architecture analysis and matrix-based components dynamic scheduling algorithm. We summed up the necessary condition which software components you can unloaded. And we descript the matrix algorithm method for components dynamic scheduling. The paper in-depth researches the component dynamic scheduling technology in the automotive electronics applications, and gives a project application. These researches have certain reference value for both automotive electronics, and other component-based embedded software design applications. This work was support by the research fund of introducted talent of Nanjing Institute of Technology(Grant No.272341726101440).

References

1. C. Szyperski, Component Software: Beyond Object-Oriented Programming. USA, New York: Addison-Wesley, ACM Press (1998).
2. Yang Fu-qing, Mei Hong and Li Ke-qing, Software Reuse and Software Component Technology. Chinese Journal of Electronics, 27(2)(1999),68-75
3. M.D.Mcllroy. Mass-produced Software Components. Proceedings of 1968 NATO Conference on Software Engineering.(1968)138-150
4. Van OMMERING, R., et al. The Koala component model for consumer electronics software.Computer. 33(3) (2000), 78-85.
5. Fang Hong-zheng, Zhao Gui-geng and Liu Ke-jun, Embedded Component Model Research. Microcomputer Applications. 26(5) (2005),521-524
6. Jawawi,D.N.A., S, Deris and R.Mamat. Enhancements of PECOS Embedded Real-Time Component Model for Autonomous Mobile Robot Application. Proceedings of 2006 IEEE International Conference on Computer Systems and Applications. Sharjah, United arab emirates: Institute of Electrical and Electronics Engineers Computer Society. (2006),882-889.
7. Norstrom, C., Gustafsson. M., Sandstrom. K, Maki-Turja. J and Bankestad. N.-E. Experiences from introducing state-of-the-art real-time techniques in the automotive industry. Proceedings of Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems. Washinton. DC, U.S.: Institute of Electrical and Electronics Engineers Inc. (2001),111-118.
8. Wallnau, K. C. and J. Ivers. Snapshot of CCL: A Language for Predictable Assembly. USA: CMU, CMU/SEI-2003-TN-025,(2003).
9. Wallnau and K.C.A. Technology for Predictable Assembly from Certifiable Components. USA: CMU, CMU/SEI-2003-TR-009,(2003).
10. Hansson. Hans, Âkerholm.Mikael, Crnkovic. Ivica and Torngren. Martin. SaveCCM-a component model for safety-critical real-time systems. Proceedings of 30th Euromicro Conference, Rennes, France: IEEE Computer Society (2004)627-635.
11. Technical Overview 2.0.1, AUTOSAR GbR, (2006). http://www.autosar.org/download/R2.0/AUTOSAR_TechnicalOverview.pdf
12. Pierre Palatin,Yves Lhuillier and O.Temam. CAPSULE: Hardware-Assisted Parallel Execution of Component-Based Programs. Proceedings of 39th Annual IEEE/ACM International Symposium on Micro-architecture, MICRO-39, Orlando. FL, U.S.: Inst of Elec (2006), 247-258.
13. MC9S12XEP100 Refernce Manual Covers MC9S12XE Family, Freescale Semiconductor. Inc, Arizona (2008).
14. Rekik and Rojdi. Application of a CAN BUS transport for DDS middleware, 2nd International Conference on the Applications of Digital Information and Web Technologies, ICADIWT 2009, London, U.K.: IEEE Computer Society(2009), 766-771.