A Task Generation Method for the Development of Embedded Software

Zhigang Gao, Zhaohui Wu, and Hong Li

College of Computer Science, Zhejiang University,
Hangzhou, Zhejiang, P.R. China, 310027
{gaozhigang, wzh, lihong}@zju.edu.cn

Abstract. The development of embedded software has the tendency towards higher levels of abstraction. This development paradigm shift makes the synthesis of embedded software a key issue in the development of embedded software. In this paper, we present a new method of generating real-time tasks, which uses Component Sequence Diagram to organize tasks and assign their priorities. Moreover, we use simulated annealing algorithm to explore design space and iterate the process of task generation until an optimization implementation is obtained. Experimental evaluation shows this method can yield correct implementation and has better time performance.

1 Introduction

Synthesis of embedded software refers to the process from function specification to code implementation on a specific platform. It is a key problem in the design of embedded system. Non-functional requirements are implemented in task generation phase when synthesizing embedded software. Current research works often ignore or deal with non-functional properties roughly [3, 5]. In this paper, we focus our attention on the problem of task generation on single processor and propose a novel task generation method. Gu et al. [6] proposed a method of synthesizing real-time implementation from component-based software models. Our work is an extension of the work of Gu et al., which combines simulated annealing (SA) algorithm with Component Sequence Diagram (CSD). We use SA to explore design space and CSD to organize tasks, to assign their priorities, and to prune the search space of simulated annealing algorithm. The process of task generation is iterated until an optimization implementation is obtained.

The rest of this paper is organized as follows. Section 2 presents the computational model. Section 3 describes the process of task generation. Section 4 gives the experimental results and the conclusions of this paper.

2 Computational Model

The software model before task generation is called the structural model, and the software model after task generation is called the runtime model.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994, pp. 918 – 921, 2006.

Structural Model. We use the structural model proposed in [1]. A transaction is defined as a series of related components' actions triggered by an event. An example of structural model is shown in Fig. 1.

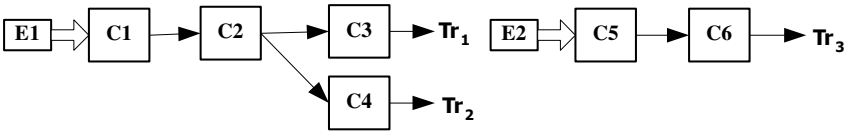

Fig. 1. Structural model consisting of three transactions

Runtime Model. The runtime model is based on task sequences. It is a sequence of related tasks communicating with each other through events.

3 The Implementation of Task Generation

The problem of assigning priorities to tasks for obtaining a schedulable system is an NP-hard problem [7]. Similarly, the problem of task generation is an NP-hard problem, too. It usually resorts to heuristic algorithms. Unfortunately, heuristic algorithms would result in high computation overhead because of large search space. Here we use a diagram, CSD, to describe assignment sequence of components. It has no scale and only denotes the constraints of time sequence (CTS). Since the assignment of components on CSD denotes a kind of execution orders of components according to the constraints of time sequence, we can organize components into tasks and assign their priorities according to their places on CSD. The process of task generation is shown in Fig. 2. It includes two phases: initialization and task generation. Task generation is performed iteratively until an acceptable Critical Scaling Factor (CSF) [2] is obtained or no schedulable task set can be found.

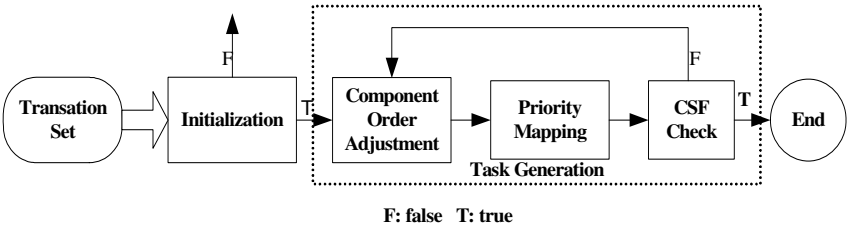

Fig. 2. The process of task generation

3.1 Initialization

The purpose of initialization is to abstract the constraints of time sequence between components and to obtain a feasible component arrangement scheme that accords with their CTS. It includes three steps: First, generate the transaction set. Only the transactions within the least common multiple (LCM) of all periods of transactions in the structural model need to be generated. We introduce the notion of virtual

transaction. A transaction with the period less than LCM is divided into multiple virtual transactions. Second, abstract the constraints in transactions and virtual transactions. Third, assign all the components of transactions and virtual transactions obtained in the first step on CSD, while satisfying the constraints obtained in the second step.

3.2 Task Generation

The purpose of task generation is to find a task set with acceptable CSF. It includes three steps: First, adjust component order to search for all feasible assignment schemes of components in the search space of SA. Second, merge the components that belong to the same transaction and are adjacent into a task and assign their priorities. Third, check Whether CSF is acceptable. The worst-case local response time of a task can be obtained using the formula presented in [4]. A task sequence's worst-case local response time (WCRT) is the sum of the worst-case local response time of all its tasks. After all task sequence's WCRT is computed, we can obtain the CSF of a system according to the deadlines of all task sequences.

The complete process of task generation is shown in **Algorithm TG.**

Algorithm TG (TS)

```
  /* TS is the set of transactions. TE is the temperature
of SA. E and E' are the energy of components assignment on
CSD. CSF_exp is the expected CSF. T_min is the threshold of
temperature of SA */
  TE = 10*max{D_Tri / ∑C_ik | Tri ∈ RM} ;
  Initialize components assignment on CSD;
  E = 0; CSF = CSF_old = 0;
  Priority mapping;
  Calculate CSF of the task set;
  E' = -CSF;
  do {
      Select two components C_i and C_j randomly on CSD;
      if (C_i and C_j are exchangeable)
           Exchange their positions on CSD;
      else
            continue;
      Priority mapping;
      Calculate CSF of the task set;
      if (E < E')
           E' = E;
      else {
              if ( e^((E-E')/TE) < 0.5 )
                    E = E';
              else
                    Return to the components assignment
              before exchanging;
                }
          if (CSF > CSF_old ) TE = TE*0.95;
  } while (CSF < CSF_exp and T > T_min)
  if (T <= T_min) return false;
  return true;
```

4 Experiments and Conclusions

In order to evaluate the effect of our algorithm, we compared our method, named SA+CSD, with the method only using SA. The results are shown in Fig. 3.

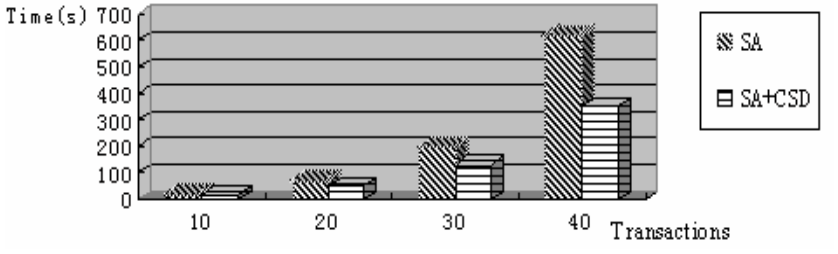

Fig. 3. The result of experiments

In this paper, we present a new method of task generation. It uses CSD to generate tasks and assign their priorities. By using of CTS, the speed of task generation is improved greatly. Our future work will focus on the influence of other resource constraints, such as memory, energy, on task generation.

References

1. Wang, S., Shin, K.G.: An Architecture for Embedded Software Integration Using Reusable Components. CASES. (2000) 110-118
2. Vestal, S.: Fixed-Priority Sensitivity Analysis for Linear Compute Time Models. IEEE Trans. Software Eng., Vol. 20. (1994) 308-317
3. Kodase, S., Wang, S., Shin, K.G.: Transforming Structural Model to Runtime Model of Embedded Software with Real-Time Constraints. DATE. (2003) 20170-20175
4. Tindell, K., Clark, J.: Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Microprocess & Microprogram, Vol. 40. (1994) 117-134
5. Gomaa, H.: Designing Concurrent Distributed, and Real-Time Applications with UML. Addison–Wesley. (2000)
6. Gu, Z., Shin, K.G.: Synthesis of Real-Time Implementations from Component-Based Software Models. Proc. IEEE Real-Time Systems Symposium (RTSS 2005). Miami, FL
7. Burns, A.: Scheduling Hard Real-Time Systems: A Review. Software Engineering Journal, Vol. 6, No. 3. (1991) 116-128