컴포넌트 효율성 특성을 고려한 Wright기반의 커넥터 확장

(Extension of Wright-based Connector Considering Efficiency Characteristics of Component)

정 화 영 † 송 영 재 ††
(Hwa-Young Jeong) (Young-Jae Song)

요 약 소프트웨어 아키텍처기반의 컴포넌트 조립 및 합성기법에서 기존의 아키텍처 기반 조립기법인 ACME, Wright 등은 커넥터의 Role을 통하여 컴포넌트들 사이의 직접적인 연결구조에 의한 선요구 선처리로 운용된다. 그러나, 커넥터에서 운용되는 선요구 선처리 기법은 각 특성을 달리하는 컴포넌트들의 비동기적인 요구 발생시 효율성이 낮은 컴포넌트가 요청되어 처리가 할당되면 효율성이 높은 컴포넌트의 처리를 위해서는 단순히 순서를 기다려야 하므로 컴포넌트들의 요구처리에 대한 효율적인 처리 및 운용이 어렵다. 따라서, 조립 컴포넌트들의 운용성 향상을 위하여 커넥터에서 각 호출 컴포넌트들의 특성을 고려한 우선순위에 따라 요구에 대한 할당처리가 필요하다.

본 연구에서는 기존의 Wright명세를 기반으로 커넥터부분을 다중 연결구조에서 사용 가능하도록 확장하였으며, 컴포넌트로부터 요청된 서비스처리를 위하여 조립 컴포넌트들의 효율성 요소 중 CPU사용율, 빈 요구 처리시간, 메모리 사용율에 대한 가중치를 계산하여 우선순위를 산정하여 운용되도록 설계 및 구현하였다. 이러한 적용결과로 각 효율성 특성을 달리 갖는 샘플 EJB 컴포넌트 20개를 구현하여 시험 및 운용함으로서 기존의 선요구 선처리 기법 대비 481ms의 소요시간 차이를 보이고 있으나 효율성이 높은 컴포넌트의 요구부터 먼저 처리됨으로서 전체 시스템의 효과적인 운용이 가능하였다.

키워드 : 커넥터 운용기법, Wright, 소프트웨어 아키텍처 스타일, 컴포넌트 조립 및 운용

Abstract In the component assembly and composition technique of software architecture, It is operated that the existing composition techniques based on architecture, ACME, Wright etc., used in FIFO with the direct connection structure between components through connector's Role. But, when the non-synchronizing request of components that have different characteristics occurs, the FIFO techniques is applied to the connector is difficult to process and operate effectively because of the high performance component waiting the sequence order if the low performance component is allocated first. Thus, the allocated request process according to the priority considering the characteristics of each call components in connector is necessary to improve the operation of assembled component.

In this research, we extend the connector part that is available in multiplex connection structure based on existent Wright specification. For service process requested from component, the connector part is designed and implemented to operating with priority sequence through calculating the weight of CPU use rate, bean requesting process time and memory use rate among the efficiency elements of assembled components. To verify the efficiency if this designed connector, we implemented 20 samples EJB components that have different efficiency characteristics and applied these samples components to designed connector. The operating results with this designed connector show that the efficient operation of whole system is possible though the processing time takes 481ms more than the time of the existing FIFO techniques.

Key words : Connector process technique, Wright, Software Architecture Style, Component Composition

† 종신회원 : 예원대학교 정보경영학부 교수
jhymichael@empal.com
†† 종신회원 : 경희대학교 컴퓨터공학과 교수
yjsong@khu.ac.kr
논문접수 : 2003년 4월 8일
심사완료 : 2003년 9월 5일

1. 서 론

컴포넌트 기반 개발기법(CBD)은 소프트웨어 프로그래밍에서 하드웨어 개발 환경처럼 소프트웨어를 Plug-and-Play 방식으로 시스템을 구축하는 '합성을 통한 시스템 구축'으로의 전환을 목적으로 한 방법이다[1]. 따라서, 시스템은 독립모듈들, 이름, 소프트웨어 컴포넌트들이 잘 정의된 소프트웨어 아키텍처의 형식을 가지며[2], 컴포넌트 조립은 인터페이스 또는 커넥터가 컴포넌트사이의 교환을 나타냄으로서[3] 이루어진다. 아키텍처 기반 컴포넌트 조립기술은 오랜 기간에 사용된 Pipe-and-filter아키텍처[4], Aesop[5], C2[4], ACME[3], Wright[6] 등을 들 수 있다. Aesop, ACME, Wright의 경우는 컴포넌트들 사이의 직접적인 연결구조를 가지며 커넥터의 Role 구성에 따라 선요구 선처리 방식으로 요구 메시지를 처리한다. C2의 경우는 메시지 큐를 이용한 다중의 비동기적인 요구호출이 가능하지만 호출 컴포넌트의 특성을 고려하지 않은 선요구 선처리 방식으로 운용되며 직접적인 메소드 호출방식을 갖는 EJB의 경우는 수정이 필요하다. SOFA 컴포넌트 모델은 컴포넌트들의 동적처리를 위하여 채널형식으로 컴포넌트들의 요구처리를 하고 있으나 기본적으로 선요구 선처리 방식을 따른다. 즉, 커넥터에서 컴포넌트들 사이의 인터페이스는 컴포넌트들의 특성 및 조립구조에 의하여 대부분 선요구 선처리 방식을 택하고 있다. 그러나, 선요구 선처리 방식은 요구 호출하는 컴포넌트의 특성을 고려하지 않음으로서 효율성 낮은 컴포넌트가 먼저 요청되어 처리가 할당되면 효율성 높은 컴포넌트들이 요청되어도 서비스를 할당할 수 없다. 즉, 동시에 각 특성을 달리하는 많은 컴포넌트들의 요구 발생시 처리를 위해서는 단순히 순서를 기다려야 하므로 효율적인 처리 및 배치 운용이 어렵다. 특히, 직접적인 단순 연결구조를 갖는 Wright의 경우는 모든 컴포넌트의 수정 없이 적용 가능하지만 선요구된 요청에 대하여 처리가 종료되어야만 다음의 요구가 연결되고 핸들링 할 수 있다.

따라서, 본 연구는 다중의 비동기적인 컴포넌트 요구처리를 효율성 특성에 따른 가중치를 이용하여 효율적으로 할당할 수 있는 커넥터를 설계 및 구현하였다. 즉, 커넥터에서 서비스를 요청 받은 컴포넌트들의 효율적인 처리 할당을 위하여 효율성 특성에 의한 요구처리 우선순위를 두었다. 우선순위 설정대상은 국제표준(ISO/IEC 9126)에서 규정하고 있는 컴포넌트 명세 중 측정이 용이한 효율성요소에서 CPU사용율, 빈요구처리시간, 메모리 사용율을 이용하였다. 이러한 효율성요소값에 따라 가중치를 설정하였으며, 가중치들의 합에 의해 효율성이 높은 순으로 우선순위를 산출하였다. 또한, 본 제안기법에서 적용되는 컴포넌트는 서버측 컴포넌트 모델인 EJB를 기반으로 하였다. 이를 모든 컴포넌트들에서 적용 가능하도록 간단한 조립구조를 갖는 Wright 명세기반으로 설계하였으며, 커넥터 부분에서 다중의 컴포넌트 조립 및 운용이 가능하도록 요구처리를 위한 메시지 큐의 적용과 처리 우선순위 결정 알고리즘을 추가하여 확장하였다.

이러한 적용결과는 각 특성을 달리 갖는 예제 EJB 컴포넌트 20개를 구현하여 시험 및 운용함으로서 기존의 선요구 선처리기법에 비하여 481ms의 소요시간 차이를 보였으나 효율성이 높은 컴포넌트의 요구부터 먼저 처리됨으로서 전체 시스템의 요구 메시지운용에서 효율성에 따른 조립 컴포넌트의 자동처리 배치 및 운용이 가능함을 보였다.

2. 관련연구

2.1 소프트웨어 아키텍처 기반 컴포넌트 조립 및 운용기법

아키텍처 모델은 컴포넌트들간의 상호작용에 대한 그래프의 형태로 나타내며, 노드의 형태를 갖는 컴포넌트와 선 형태의 커넥터를 이용하여 컴포넌트들 사이의 상호작용에 대한 경로를 표현한다[7]. 아키텍처 기반의 컴포넌트 조립기법에서 ACME는 아키텍처 명세 교환을 지원하며, C2는 메시지 기반 스타일을 이용하여 사용자 인터페이스를 지원하고[8], Wright, Aesop는 컴포넌트, 커넥터, 연결규칙을 갖는 아키텍처를 지원한다[9].

아키텍처기반 컴포넌트 운용에서 C2는 메시지 큐를 이용한 FIFOPort를 통하여 컴포넌트들의 요구처리를 수행하고 있다[5,10]. 또한, 시멘틱 모델기반의 Wright는 그림 1, 2와 같이 직접적인 연결구조(Link)를 가지며

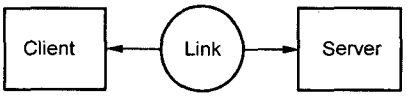

그림 1 Wright의 구성 예

```
Style Client-Server
  Component Client
    Port p  request → reply → p □ §
    Computation  internalCompute → p.request → p.reply → Computation □ §

  Component Server
    Port p  request → reply → p ▯ §
    Computation  p.request → internalCompute → p.reply → Computation ▯ §

  Connector Link
    Role c  request → reply → c □ §
    Role s  request → reply → s ▯ §
    Glue  c.request → s.request → Glue ▯ s.reply → c.reply → Glue ▯ §

  Constraints
    ∃! s ∈ Component, ∀ c ∈ Component : TypeServer(s) ∧ TypeClient(c) ⇒ connected(c,s)
EndStyle
```

그림 2 Wright의 구성 예에 대한 명세

커넥터에서 Role과 Glue에 의하여 컴포넌트들의 요구처리를 수행하고 있다.

그림 2에서 ⊓은 내부 요소의 사용, □은 외부요소의 사용을 나타낸다. 또한, $\overline{\text{request}}$는 컴포넌트의 요구 메시지 생성, reply는 응답, $\overline{\text{reply}}$는 응답 메시지 생성, §은 성공적인 처리 종료를 나타낸다.

아키텍처 기반에서 컴포넌트의 조립 및 운용의 중요한 요소는 커넥터가 된다. 커넥터에서는 순차적으로 연결된 컴포넌트들의 서비스 이름을 비교하며, 컴포넌트들의 요구처리는 분산된 시스템에서 접속된 컴포넌트들의 확실한 명세 하에서 실행된다[11]. 이와 같은 방법으로, 커넥터는 컴포넌트들의 기능적인 요구사항들로부터 인터페이스 요구사항들을 분리한다. 그 결과, 커넥터는 컴포넌트들 사이의 상호작용 프로토콜을 포함하여 매우 다양한 목적을 위하여 사용된다.

2.2 선요구 선처리에 따른 기존의 커넥터 설계 및 운용 기법

커넥터는 통신규약과 메커니즘에 따른 실행순서와 컴포넌트들의 상호작용을 포함한다. Wright는 직접적인 연결구조를 정의하고 있으며 컴포넌트들의 다중연결을 위하여 그림 3과 같은 구조로 연구된 바 있다[12].

이는, 단일 커넥터에 3개의 컴포넌트를 연결한 구조로서 컴포넌트의 요구 발생시 채널과 같은 역할을 커넥터에서 수행함으로서 요청된 서비스 처리를 수행하였다. 그림 4는 이에 대한 Wright 명세를 나타내며, control.changeOK 부분은 그림 3의 연결구조에서 Primary 컴포넌트에서 Secondary 컴포넌트로의 서비스 처리전환을 나타낸다. 즉, Wright는 직접연결에 따른 링크 인터페이스를 통하여 컴포넌트들의 조립 및 운용을 수행함으로서 EJB와 같은 메소드 호출방식의 컴포넌트에서도 수정 없이 적용가능 하지만 다중연결 구조를 지원하지 않음으로서 채널구조 형식에 따른 커넥터의 확장이 연구되었으나, 채널에 따른 커넥터 사이의 컴포넌트들이 일대일 대응으로 이루어짐으로서 조립 컴포넌트들의 다중 요구처리가 어렵다. 따라서, 조립 및 운용기법은 적용하려는 컴포넌트의 수정 없이 운용되어야 하며, 커넥터에서 다중 컴포넌트들의 요구처리를 위한 다중연결구조의 지원이 요구된다.

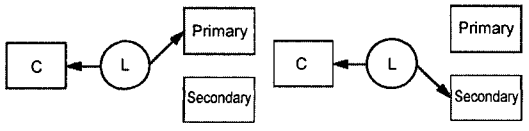

그림 3 다중연결을 위한 Wright 기반의 커넥터 연결구조

```
Connector FTLink
  Role c = request → reply → c ⊓ §
  Role s = (request → reply → s ⊓ control.changeOk → s)□ §
  Glue = c.request → s.request → Glue
       □ s.reply → c.reply → Glue
       □ §
       □ control.changeOk → Glue
```

그림 4 다중연결을 위한 Wright명세 기반의 커넥터 확장

메시지 기반의 C2에서 커넥터는 요구 컴포넌트의 파라미터를 통하여 컴포넌트를 식별하고 비동기적으로 요구 요청이 생성시 선요구 선처리 방식에 따라 처리하고 있다[4]. 그러나, 조립 및 운용에서 전체 조립 구성중 컴포넌트 처리시간, 프로세서와 메모리의 사용량이 많은 컴포넌트가 먼저 요청되었을 경우 처리시간, 프로세서, 사용량 등이 작은 컴포넌트가 요청되어도 처리할당을 위해서는 단순히 순서를 기다려야만 하였다. 이는, 요구순서에 따라 효율성이 낮은 컴포넌트가 처리를 점유함으로서 효율성 높은 컴포넌트들의 요구가 있을지라도 요구에 대한 처리를 할당받을 수 없다. 이러한 결과는 컴포넌트 단위의 처리를 기반으로 한 전체시스템 운용 측면에서 비효율성을 야기시킨다.

따라서, 컴포넌트들의 수정 없이 적용 가능한 다중연결구조를 지원하는 커넥터의 설계 및 운용이 필요하며 전체 시스템에 대하여 하부 조립 컴포넌트들의 특성을 고려한 처리기법이 요구된다.

3. 효율성 특성을 이용한 컴포넌트 요구처리 기법

3.1 컴포넌트 효율성 특성에 의한 우선 순위

본 연구는 컴포넌트의 수정 없이 적용 가능하도록 단순 연결구조를 갖는 Wright 명세기반으로 다중연결구조를 지원하도록 커넥터를 설계하였으며 커넥터에서 컴포넌트로부터 요청 받은 요구처리에 대한 효율적 운용을 위하여 ISO/IEC 9126에서 규정하고 있는 컴포넌트 명세 중 효율성을 고려한 우선순위를 산정하였다. 컴포넌트 명세값들은 개발업체 또는 개발자에 의해 컴포넌트 명세로 제공되어지며, 사용자에 의해 주/객관적으로 평가된다. 적용된 효율성 특성요소는 CPU사용율, 빈 요구처리시간, 메모리 사용율을 선정하였다. 커넥터를 중심으로 컴포넌트사이의 통신으로 인한 전체 시스템운용에서, CPU사용률은 조립 컴포넌트의 처리에 대한 CPU의 점유율을 나타낸다. 또한, 빈 요구처리시간은 컴포넌트의 처리시간을 나타내며 전체 시스템의 운용시간에 영향을 미친다. 또한, 메모리 사용율은 컴포넌트가 처리되

는 과정에서 점유하는 메모리 자원을 나타낸다. CPU 사용률과 메모리 사용율은 조립 컴포넌트에 의한 시스템의 하드웨어적 자원을 의미하며, 빈 요구처리시간은 소프트웨어적인 처리기간을 의미한다. 본 연구에서는 적용된 컴포넌트들의 각 특성값 측정을 위하여 Weblogic에서 컴포넌트의 기능 테스트를 이용하였다. 그러나, 특정 변수에 따라 컴포넌트의 각 효율성 수치가 달라질 수 있는 경우는 고려하지 않았으며, 임의의 샘플변수에 따라 일반적인 컴포넌트 기능테스트에 의한 수치만을 반영하였다. 선정된 효율성 특성은 그 수치에 비례하여 설정범위를 지정하였고, 가중치 산정을 위한 가중치 결정테이블을 구성하였다.

컴포넌트들의 효율성 특성값은 가중치 결정 테이블에 의하여 해당 가중치로 변환하였다. 따라서, 각 가중치의 합은 요구 컴포넌트들의 효율성 특성값에 대한 상대적인 수치를 나타냄으로서 이에 대한 수치가 작은 컴포넌트들부터 먼저 처리 및 수행하도록 하였다. 그러나, 요구 컴포넌트들에 대한 가중치의 합이 같을 경우는 선요구 선처리방식으로 처리하였다. 표 1은 적용된 컴포넌트의 효율성 수치(R_{11}~R_{n3})를 기준으로 가중치 결정 테이블에 의한 가중치(W_{11}~W_{n3})를 나타내며, 이를 위하여 표 2에서와 같이 각 효율성 수치의 범위에 따라 가중치를 설정하도록 하였다.

가중치 부여방법에서 실제적인 효율성 수치인 R_{nm}에 대한 범위는 상대적 계산에 의하여 산출된다. 즉, 각 효율성 항목들에 대하여 R_{nm}수치들의 전체분포를 기준으로 25%미만은 가중치 1을, 75%이상이면 가중치 4를 적용하였다. 예를들어, CPU사용율 속성의 경우, R_{11}, R_{21}, ..., R_{n1}에 대한 상대범위 25%는 (최대수(R_{11}, R_{21}, ..., R_{n1})-최소수(R_{11}, R_{21}, ..., R_{n1})) *0.25+최소수(R_{11}, R_{21}, ..., R_{n1})로 산출한다. 즉, 측정된 효율성 수치(R_{nm})들에 대하여 영역(최대수-최소수)의 25%를 산출하며, 상대범위 산출을 위하여 최소수를 더하였다. 빈 요구 처리시간과 메모리 사용율 또한 같은 방법을 적용하였다. 우선순위 결정은 가중치 결정 테이블에 따라 산정된 가중치의 합이 해당 컴포넌트의 우선 순위 평가치(S_i)가 된다. 즉, 본 연구에서는 컴포넌트의 3가지 효율성 모두 고려하였으므로 컴포넌트들의 최종 처리순서를 결정하는 Si는 각 효율성 요소들의 가중치들을 모두 합산하여 적용하였다.

표 1 컴포넌트 효율성과 가중치

컴포넌트 효율성	요구된 컴포넌트						
	1		2		.....	n	
CPU 사용율	R_{11}	W_{11}	R_{21}	W_{21}	.....	R_{n1}	W_{n1}
빈 요구 처리 시간	R_{12}	W_{12}	R_{22}	W_{21}	.....	R_{n2}	W_{n2}
메모리 사용율	R_{13}	W_{12}	R_{23}	W_{23}	.....	R_{n3}	W_{n3}

R_{nm} : 해당 컴포넌트의 효율성 수치

W_{nm} : 가중치, (m:1≤m≤3)

표 2 컴포넌트 가중치 결정 테이블

컴포넌트 효율성	범위(%)	가중치
CPU 사용율	$0 < R_{nm} \leq 25$	1
	$26 < R_{nm} \leq 50$	2
	$51 < R_{nm} \leq 75$	3
	$76 < R_{nm} \leq 100$	4
빈 요구 처리 시간	〃	〃
메모리 사용율	〃	〃

$$S_i = \sum_{j=1}^{m} W_{ij},\ (1 \leq i \leq n)$$

산정된 우선 순위 평가치(S_i)에 따라 요구된 컴포넌트의 처리 순서를 재배치한다. 즉, 컴포넌트 효율성 특성인 CPU 사용율, 빈 요구 처리 시간, 메모리 사용율은 수치가 낮을수록 작은 용량에서 빠른 처리를 줄 수 있으므로 우선 순위 평가치가 낮은 요구부터 처리를 할당할 수 있다.

3.2 우선순위를 고려한 Wright 명세기반의 커넥터 설계

본 연구에서 적용된 EJB 컴포넌트 모델은 다중 쓰레드를 갖지 않고 단일 쓰레드에서 실행되며, 설계된 커넥터에서 컴포넌트들의 메소드 정보를 가지는 메소드 리스트를 기반으로 컴포넌트들 사이의 인터페이스를 수행함으로서 컴포넌트들의 동기/비동기적 호출의 경우를 모두 허용하고 있다.

효율성 특성을 고려한 우선순위 알고리즘에 따라 Wright명세기반에서 그림 5와 같이 커넥터를 설계 및 구현하였다. 즉, GetComponentEfficient에서는 우선순위 알고리즘에 따라 컴포넌트 속성 테이블로부터 요구 컴포넌트들의 효율성에 대한 수치와 가중치결정테이블에 의하여 각 요소에 대한 가중치를 산정하였다. Calc-Weight에서 가중치들의 합계를 계산하였으며, Sort.ng

```
Style Client-Server
  Component Component1
    Port p = request → reply → p □ §
    Computation = internalCompute → p.request → p.reply → Computation □ §

  Component Component2
    Port p = request → reply → p [] §
    Computation = p.request → internalCompute → p.reply → Computation [] §

  Connector Link
    Role c = request → reply → c □ §
    Role s = request → reply → s [] §
    Glue = c.request → s.request → Glue []
          {GetComponentEfficient → CalcWeight → Sorting } → s.reply → c.reply → Glue [] §

  Constraints
    ∃! s ∈ Component, ∀ c ∈ Component : TypeServer(s) ∧ TypeClient(c) ⇒ connected(c,s)
EndStyle
```

그림 5 효율성의 가중치에 따른 우선순위를 고려한 커넥터 설계

에서는 가중치의 합을 기준으로 수치가 낮은 컴포넌트 요구부터 정렬함으로서 우선순위 처리순서를 정하였다.

본 시스템의 구현을 위한 클래스 다이어그램은 그림 6과 같이 나타낸다. 클라이언트는 조립된 전체 시스템에서의 요청 및 결과를 나타내며 초기화를 통하여 구현된 EJB 컴포넌트들을 등록시킨다. 또한, 각 컴포넌트의 호출에서 최종결과까지의 빈 처리 응답시간을 검색하기 위하여 StartTime과 EndTime을 두었다. 커넥터부분에서는 각 컴포넌트의 명세에 따른 속성 테이블을 기반으로 한 처리 우선 순위를 두었으며, 우선 순위 평가치를 정렬하기 위하여 버블정렬 알고리즘을 사용하였다. CompoEffi()는 컴포넌트 가중치 결정 테이블에 따른 가중치를 산정 하였으며, CalcWeight()에서는 전체 가중치의 합을 구함으로서 우선 순위 평가치를 산정하고 정렬을 호출하여 오름차순에 의한 호출순서를 결정한다. 메시지 리스트에서는 요청된 컴포넌트의 메시지를 요구큐에 저장하며, 이를 커넥터에 제공한다.

그림 7은 시퀀스 다이어그램을 나타낸다. 클라이언트에서 초기화시 조립 컴포넌트들을 Lookup을 통하여 찾아 등록하였다. 메시지등록에서는 서버 컴포넌트들이 수행하여야 할 메소드들을 지정하는 리모트 메소드들의 호출형식을 커넥터로 보내어 컴포넌트 운영 전에 메시지 큐에 미리 저장하는 역할을 한다.

Request는 조립된 컴포넌트에 따라 실제적으로 클라이언트에서 서버 컴포넌트들의 처리를 요구하는 호출부분이다. 커넥터에서는 클라이언트로부터 호출된 서버 컴포넌트들의 처리요구들을 받아 Get메시지를 통하여 메시지 리스트에서 해당 컴포넌트가 수행하여야하는 메소드들의 호출형식을 찾으며, 요청된 메시지를 통하여 이를 커넥터에 반환한다. 우선순위 산정과 처리 컴포넌트 할당부분에서는 수행할 각 메소드 호출형식을 가지는

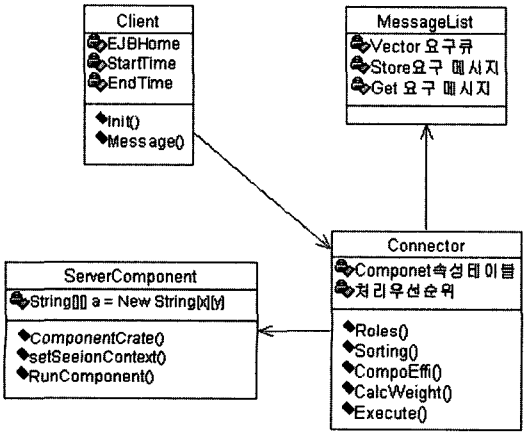

그림 6 클래스 다이어그램

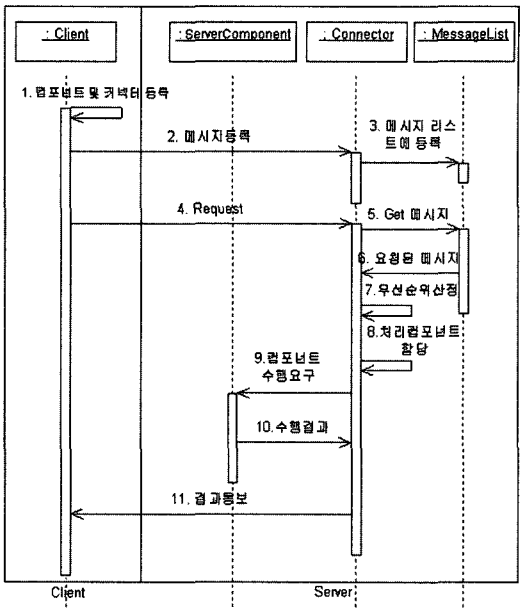

그림 7 시퀀스 다이어그램

컴포넌트들의 효율성특성을 고려하여 요청된 서비스 처리 우선순위를 산정한다. 컴포넌트 수행요구에서는 우선순위에 의하여 할당된 처리순서에 따라 메소드 호출형식을 통하여 실제적으로 해당 컴포넌트들의 메소드를 호출한다.

4. 시험 및 결과분석

본 시스템의 운영 및 시험을 위하여 그림 8과 같은 방법으로 효율성의 각 특성을 다르게 갖는 예제 EJB 컴포넌트 20개를 구현하였다.

이는, 무상태 세션 빈에서 Cloudscape를 통한 데이타베이스 내용을 확인하는 간단한 구조를 갖는다. 또한, 각 컴포넌트의 효율성 특성 차이를 위하여 컴포넌트 내부의 프로세스 중 정적 배열의 사용으로 메모리 사용율을 다르게 하였고, 중첩 반복문 사용으로 인하여 처리시간을 지연시킴으로서 빈 요구 처리시간을 다르게 하였으며, CPU 사용율은 빈 요구 처리시간의 지연과 메모

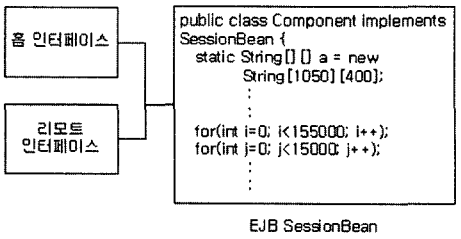

그림 8 예제 EJB 컴포넌트 구조

표 3 예제 EJB 컴포넌트 효율성 특성

	EJB 컴포넌트																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CPU 사용율(%)	64.6	84.2	100	100	100	100	100	100	100	68.8	58.2	62.2	68.7	65.6	76.5	69.7	100	93.7	89.8	92.3
빈 요구 처리 시간(ms)	701	691	2754	2804	4256	5568	6590	2864	2884	701	629	688	702	699	714	702	2855	2649	2331	2611
메모리 사용율(%)	6.36	6.47	7.01	7.11	7.11	7.28	7.01	7.01	7.00	6.80	6.01	6.28	6.80	6.39	6.98	6.91	7.01	7.01	6.99	6.99

리 사용율의 차이에 비례하여 나타남으로서 각 효율성이 다른 컴포넌트를 적용할 수 있었다.

이에 따라 예제 컴포넌트들의 효율성 특성들을 측정한 결과는 표 3과 같이 나타내었다. 또한, 각 효율성 수치에 따라 전체분포에 의한 가중치 결정 테이블은 표 4와 같이 설정된다. 가중치결정에 필요한 범위산정은 측정된 각 효율성 수치분포에 따라 상대적 적용범위에 의해 상대분포영역이 산출된다.

가중치 결정 테이블을 기준으로 각 효율성에 대한 가중치와 우선순위는 표 5와 같다. 본 연구의 적용결과로는 Wright에 의하여 효율성 특성을 고려하지 않는 기존의 선요구 선처리 방식과 우선순위를 고려한 제안된 커넥터 처리방식을 각각 구현하여 그 결과를 비교하였다. 따라서, 기존의 선요구 선처리 방식으로 조립되어 운용된 결과는 각 컴포넌트의 특성과는 상관없이 요청된 순서대로 처리됨을 나타낸다. 또한, 컴포넌트 초기화 및 등록을 포함한 20개의 컴포넌트 수행에 대한 총 처리시간은 44526ms가 소요된다. 반면, 본 제안 알고리즘에 의해 효율성 우선순위에 따른 수행결과는 총 수행시간 45007ms로서 우선순위 알고리즘으로 인하여 기존의 선요구 선처리 방식에 비하여 481ms가 더 소요된다.

그러나, 각 효율성 특성을 고려하여 요청된 컴포넌트들의 서비스에 대한 처리순서를 정함으로서, CPU와 메모리의 부하가 작고 빠른 처리를 수행할 수 있는 컴포넌트들이 우선적으로 처리되었다.

전체적인 처리결과에서 각 특성들에 대한 비교결과 그림 9는 CPU사용율을 나타내며, 제안기법이 CPU의 부하가 작은 컴포넌트부터 우선적으로 처리됨을 보인다.

그림 10은 컴포넌트의 실제적인 서비스 처리시간을 나타낸다. 이는 제안기법이 처리시간이 짧은 컴포넌트들부터 우선적으로 처리함으로서 시스템 운영초기에 많은 컴포넌트들이 처리되고있음을 나타낸다.

그림 11은 메모리 사용율을 나타내며, 본 제안기법이 메모리의 점유가 낮은 컴포넌트들부터 먼저 처리됨을

표 4 가중치 결정 테이블

컴포넌트 효율성	범위		가중치
	적용범위(%)	상대분포영역	
CPU 사용율	0~25	0~68.65	1
	26~50	68.66~79.1	2
	51~75	79.11~89.55	3
	76~100	89.56~100	4
빈 요구 처리 시간	0~25	0~2119.25	1
	26~50	2119.26~3609.5	2
	51~75	3609.51~5099.75	3
	76~100	5099.76~6590	4
메모리 사용율	0~25	0~6.3275	1
	26~50	6.3276~6.645	2
	51~75	6.646~6.9625	3
	76~100	6.9626~7.28	4

표 5 EJB 컴포넌트 효율성 특성에 따른 가중치 및 우선순위

	EJB 컴포넌트																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CPU 사용율	1	3	4	4	4	4	4	4	4	2	1	1	2	1	2	2	4	4	4	4
빈 요구 처리 시간	1	1	2	2	3	4	4	2	2	1	1	1	1	1	1	1	2	2	2	2
메모리 사용율	2	2	4	4	4	4	4	4	4	3	1	1	3	2	4	3	4	4	4	4
우선순위 평가치	4	6	10	10	11	12	12	10	10	6	3	3	6	4	7	6	10	10	10	10
우선순위	3	5	10	11	18	19	20	12	13	6	1	2	7	4	9	8	14	15	16	17

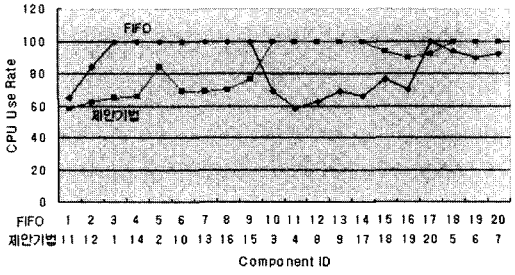

그림 9 CPU사용율 부분

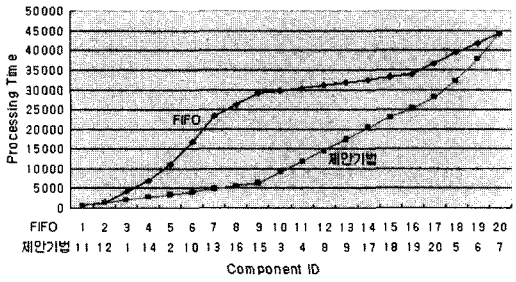

그림 10 빈 요구 처리시간 부분

표 6 단위시간당 컴포넌트 처리 개수

	단위시간									
	5000	10000	15000	20000	25000	30000	35000	40000	45000	50000
선요구 선처리	3	1	1	1	1	3	6	2	2	0
제안 기법	7	3	2	1	2	2	1	1	0	1

알 수 있다.

단위 시간당 컴포넌트의 처리 개수는 표 6과 같이 나타나며, 효율성 특성에 따른 우선순위에 의하여 초기의 처리시간에 많은 컴포넌트들이 처리됨을 알 수 있다.

또한, 컴포넌트들의 처리대기시간은 그림 12와 같다. 이에 따라 컴포넌트 처리에 대한 평균대기시간은 기존의 선요구 선처리방식이 23042.4ms이며, 제안기법은 12525.35ms로서 제안기법이 컴포넌트들의 처리를 위한 대기시간에서 더 짧은 것으로 나타났다.

결과적으로 본 제안기법이 우선순위 알고리즘 처리에 의한 요구처리 소요시간이 더 필요하지만 전체 시스템의 운용측면에서는 초기에 보다 많은 컴포넌트의 요구처리를 수행할 수 있었으며, 처리를 기다리는 각 컴포넌트들도 처리 대기 시간이 짧았다. 또한, CPU 및 메모리의 부하가 작은 컴포넌트들의 처리를 우선함으로서 초기에 빠른 처리 및 응답이 가능하였다.

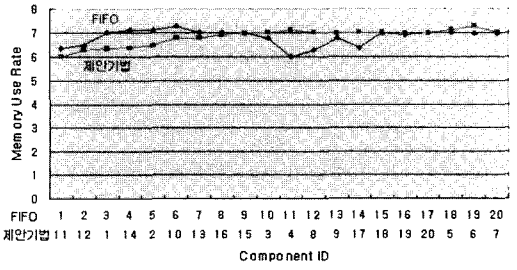

그림 11 메모리 사용율 부분

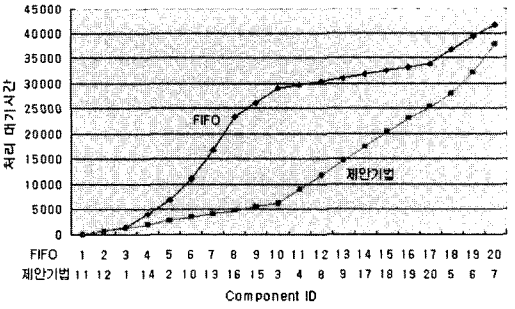

그림 12 컴포넌트들의 처리 대기시간

5. 결 론

본 연구에서는 아키텍처 기반의 컴포넌트 조립운용시 컴포넌트들 사이의 상호작용을 효율적으로 운용하기 위하여 커넥터에서 컴포넌트의 요구처리에 대한 우선 순위를 두어 처리하는 기법을 제안하였다. 이는, 컴포넌트의 효율성 특성 중 프로세스처리에 많은 영향을 주는 CPU 사용율, 빈 요구처리, 메모리 사용율을 기반으로 가중치 평가 테이블에 의한 가중치를 설정하였으며, 전체 가중치의 합계가 낮은 컴포넌트를 우선적으로 처리함으로서 시스템 운용상의 효율성을 주었다. 또한, 예제 EJB 컴포넌트 20개를 구현하여 기존의 기법과 비교함으로서 우선순위 처리에 따른 481ms 만큼의 시간차이를 보였으나, 조립 컴포넌트의 처리를 기반으로 하는 전체 시스템 운용에 있어서 효율성이 높은 컴포넌트들의 요구처리를 우선함으로서 초기의 단위시간내에 많은 컴포넌트들의 처리결과를 나타낼 수 있었다.

그러나, 본 제안기법은 우선 순위 알고리즘의 수행시간에 따른 추가적인 시간이 소요되었으며, 효율성 특성에서도 CPU 사용율, 빈 요구처리 시간, 메모리 사용율만을 이용하였다. 따라서, 향후연구과제로서 컴포넌트 조립특성에 맞는 다양한 기능성 및 효율성 특성에 따른 우선 순위 알고리즘이 필요하고, 대단위 조립 시스템에서도 효율적으로 운용될 수 있는 커넥터의 설계가 요구된다.

참 고 문 헌

[1] F. Brosard, D. Bryan, W. Kozaczynski, E. S. Liongorari, J. Q. Ning, A. Olafsson, and J. W. Wetterstrand, "Toward Software Plug-and-Play," *in Proc. of the 1997 Symposium on Software Reusability*, 1997.

[2] Ioannis Georgiadis, "Self-Organising Distributed Component Software Architecture," University of London, Ph.D Thesis, 2002. 1.

[3] David Garlan, Robert T. Monroe, David Wile, "Acme: Architectural Description of Component-Based Systems," Cambridge University Press, 2000.

[4] Taylor, R. N., Medvidovic, N., Anderson, K. M., Whitehead, E. J., Jr., Robbins, J. E., Nies, K, A., Oreizy, P. and Dubrow, D. L., "A Component-and Message-Based Architectural Style for GUI Software," IEEE Transactions on Software Engineering, Vol.22. No.6., June, 1996.

[5] Robert T. Monroe, David Garlan, "Style Based Reuse for Software Architectures," Proceedings of the 1996 International Conference on Software Reuse, April 1996.

[6] R. Allen and D. Garlan, "Formalizing Architectural Connection," Proceedings of 16th Int l Conference on Software Engineering, Sorrento, Italy, May 1994.

[7] Bradley Schmerl, David Garlan, "Exploiting Architectural Design Knowledge to Support Self-repairing Systems," Fourteenth International Conference on Software Engineering and Knowledge Engineering, July 15-19, 2002.

[8] David Garlan, "Software Architecture," Encyclopedia of Software Engineering, 2001.

[9] David Garlan, Zhenyu Wang, "A Case Study in Software Architecture Interchange," Proceedings of Coordination'99, 1999.

[10] D. C.Luckham and J. Vera, "An Event-Based Architecture Definition Language," IEEE Transactions on Software Engineering, Vol. 21, No. 9, Sept. 1995, pp. 717-734.

[11] David Garlan, Bradley Schmerl, "Model-based Adaptation for Self-Healing Systems," ACM SIGSOFT Workshop on Self-Healing Systems (WOSS'02), November 18-19, 2002.

[12] Robert J. Allen, Remi Douence, and David Garlan, "Specifying and Analyzing Dynamic Software Architectures," Proc. of the 1998 Conference on Fundamental Approaches to Software Engineering, March 1998.

정 화 영

1991년~1994년 경희대학교 전자계산공학과(공학석사). 2000년~2001년 경희대학교 전자계산공학과(박사수료). 2000년~현재 예원대학교 정보경영학부 조교수 관심분야는 소프트웨어공학, OOP/S, S/W 재사용, 컴포넌트기반 개발 방법론

송 영 재

1969년 인하대학교 전기공학과(공학사) 1976년 일본Keio University 전산학과(공학석사). 1979년 명지대학교 대학원 졸업(공학박사). 1971년~1973년 일본 Toyo Seiko 연구원. 1982년~1983년 미국 Univ. of Maryland 전산학과 연구교수. 1984년~1989년 경희대학교 전자계산소장. 1976년~현재 경희대학교 컴퓨터공학과 교수. 1993년~1995년 경희대학교 교무처장. 1996년~1998년 경희대학교 공과대학장 1998년~2001년 경희대학교 기획조정실장. 2002년~2003년 경희대학교 산업정보대학원 원장. 관심분야는 소프트웨어공학, OOP/S, CASE 도구, S/W 개발도구론, S/W 재사용, 컴포넌트기반 개발 방법론