동적 자원관리를 활용한 컴포넌트 서비스 레포지토리 설계 및 의존성 형상 알고리즘

최 상 균† · 송 영 재††

요 약

최근의 소프트웨어 개발 기술은 조립 가능한 컴포넌트를 모아 복잡한 소프트웨어 시스템을 만드는 것을 가능하게 하고 있다. 그러나 효율성, 신뢰성이 있는 동적 형상을 지원하는 컴포넌트 기반 시스템을 개발하기에는 어려움이 있다. 컴포넌트에 대한 명세화되지 않은 의존과 행위가 실패의 원인이 되고 있다. 따라서 컴포넌트 기반 소프트웨어 시스템은 컴포넌트 간의 의존성 및 컴포넌트 요구사항을 표현하는 명백한 유지가 있어야 한다. 본 논문은 컴포넌트 서비스에서 배치, 상태 보고 및 제어를 위한 컴포넌트 서비스 레포지토리를 설계하고, 기존에 연구된 컴포넌트 의존성 알고리즘의 커스터마이징을 통하여 동적 의존성을 지원하는 재형상 인터페이스 알고리즘을 제안하여 응용 프로그램의 다른 문맥을 재사용할 수 있도록 하였다.

A Design on Component Service Repository using Dynamic Resource Management and Algorithm on Configuration of Dependency

Sang-Kyoon Choi† · Young-Jae Song††

ABSTRACT

Recent developments in component technology enable the construction of complex software systems by assembling together off-the-shelf components. However, it is still difficult to develop efficient, reliable, and dynamically configurable component-based systems. Unspecified dependencies and behavior on components often has cause the source of the trouble. Therefore, component-based software systems must maintain explicit presentations of inter-component dependence and component requirements. In this paper, I describe the design on repository of component service to deployment, status reporting, and control in component service. Through the existing researches to customize component technology, I present the algorithm that supports dynamic dependency interface in order to reuse context of application.

키워드 : 컴포넌트 기반 형상관리(Component-Based Configuration Management), 의존관리(Dependency Management), 컴포넌트 Configurator(Component Configurator)

1. 서 론

컴포넌트(Component) 기술은 소프트웨어 시스템의 복잡화와 심각한 양적인 증가의 어려움에 대처할 수 있는 강력한 대안으로 대두되고 있다[1]. 최근에 개발되는 컴포넌트 시스템은 시각적 장치와 함께 프로그램적 인터페이스로 규격화된 소프트웨어 컴포넌트의 모음과 함께 조립함으로써 복잡한 시스템의 구축을 지원한다[2]. 컴포넌트는 다음 세대 소프트웨어 시스템의 포장, 배치, 전개의 구성단위가 될 것이다.

재사용 가능하고 조립가능한 컴포넌트를 이용하여 시스템을 개발하는 컴포넌트 기반 개발(CBD : Component-Based Development)에서 컴포넌트가 시스템에 어떤 영향을 미치는지에 관한 정보를 알 수 있도록 하는 것이 컴포넌트 의존 관리(Dependency Management)이다[3]. 의존 관리는 시스템 전반에 걸친 모든 컴포넌트에 관련된 정보를 메타 데이터(Meta Data)에 의한 의존 그래프라는 인터페이스를 통하여 가능한데, 메타 데이터는 새로운 컴포넌트 인터페이스로서 컴포넌트를 등록할 때 저장소에 저장된다. 그러나 컴포넌트 간의 의존을 관리하는 것에 대하여 개발자에게 충분하게 지원하지 못하고 있다. 컴포넌트가 여러 프로그래머에 의하여 만들어지고, 다른 방법론을 가진 그룹에서 작업하여 생성되기 때문이다. 컴포넌트 간의 의존 관리가 제대로 이행되지 않는다면, 강력하고 효과적인 시스템을 만들기

※ 이 논문은 2004학년도 김포대학의 연구비 지원에 의하여 연구되었음.
† 정 회 원 : 김포대학 컴퓨터계열 교수
†† 종신회원 : 경희대학교 컴퓨터공학과 교수
논문접수 : 2004년 3월 3일, 심사완료 : 2004년 4월 2일

어렵게 될 것이다.

현존하는 컴포넌트 기반 시스템은 신뢰성, 관리적인 측면, 아키텍처 구조 및 형상에 심각한 문제에 직면하고 있다[2]. 문제는 이 어려움이 의존성을 나타내는 것에 대한 단일화된 모델이 없고, 이러한 의존을 관리하는 체계가 없다는 것이다. 컴포넌트는 CPU, 메모리, 기타 장치등과 같은 하드웨어 자원 및 다른 컴포넌트, 서비스와 운영체제와 같은 소프트웨어에 의존해 있다. 이러한 의존관계를 적절한 해결 없이는 시스템의 효율성 및 신뢰성을 보장할 수 없다.

본 논문은 CBD에서 각 컴포넌트간의 의존성을 지원하는 컴포넌트 Configurator에서 각 컴포넌트 간 의존 인터페이스 알고리즘을 개선하고, 컴포넌트 서비스 레포지토리(Repository)를 설계하여 각 컴포넌트가 로드되는 서비스 시간을 줄이는 방안을 보임으로써 컴포넌트 조립 시스템의 실행 속도를 빠르게 하도록 하였다. 또한 제안된 알고리즘을 기존의 알고리즘과 비교하여 알고리즘의 안정성 및 객관성이 유지되도록 하였다. 실험결과 레포지토리를 구축하여 서비스하는 경우 평균 3ms에서 4ms까지의 로드 시간의 차이를 보였다.

2. 기존 연구

2.1 컴포넌트 기반 시스템의 자동 형상

컴퓨터 하드웨어 시스템과 응용 프로그램 컴포넌트 사이의 상호작용을 구체화함으로서, 시스템 소프트웨어는 결함 허용, 안전성, 서비스 질, 최적화를 보다 더 잘 지원하기 위한 재구성의 필요를 인식할 수 있다[4]. 부가적으로, 이것은 시스템 안정성과 신뢰성을 손상시키지 않고 성능에 대한 최소한의 충격을 가지고 재배치를 실행할 수 있는 방법을 얻는다. 소프트웨어 아키텍처, 분산시스템의 동적 재배치와 서비스 명세화의 질에 관한 의존 관리를 자동 형상(Automatic Configuration)을 통하여 지원한다. 즉 최근의 컴포넌트 아키텍처에 적용될 수 있는 자동 형상에 관한 일반적 모델을 개발하였다. 그러나 [4]에서는 체계적인 재사용을 지원하는 동적 프레임을 이용하는 의존 관련 지원 부분은 미흡하였다.

2.2 분산 시스템의 동적 자원관리

소프트웨어는 계속적으로 발전되고 새로운 컴포넌트 버전이 자주 발표된다[2]. 어떻게 하면 사용자가 최신의 컴포넌트를 사용하고, 그것들이 조화롭게 운영되게 할 수 있을까 하는 것은, 새로운 컴포넌트가 사용 가능함에 따라 그러한 컴포넌트를 넣거나(Push) 뺄(Pull) 수 있는 광역 네트워크의 코드 분산에 대한 메카니즘과, 필요할 때 새로운 컴포넌트를 사용할 수 있는 안전한 동적인 재구성을 위한 메카니즘을 필요로 한다.

소프트웨어 시스템은 급속도로 발달하고 있다. 따라서 사용자들은 새로운 버전의 출시에 따라 새로운 소프트웨어를 설치한다. 일부 소프트웨어는 설치 마법사 인터페이스를 통하여 자동적으로 설치를 한다. 그러나 설치가 완료되지 않거나, 설치가 완료 되더라도 몇몇 명세화되지 않은 필수조건이 충족되지 않아서, 소프트웨어 패키지가 제대로 작동되지 않는 상황에 부딪히는 경우가 있다. 또 다른 경우에는 업데이트 전에는 작동하던 응용 프로그램이 새로운 버전의 시스템 컴포넌트나 새 도구를 설치하고 나면 작동을 멈춘다. 시스템은 어떠한 응용 프로그램이 어떠한 라이브러리를 사용하는지 구체화하는 명확한 메카니즘을 제공하지 않기 때문에, 응용 프로그램은 설치했던 모든 파일을 제거 할 수 있는지에 대해 알지 못한다. 그러한 문제를 해결하게 위하여 PC에 소프트웨어를 설치하고, 업데이트시키고, 제거하는 새로운 패러다임를 필요로 한다. 이를 위하여 [2]에서는 시스템과 응용 프로그램 소프트웨어가 네트워크 중심 컴포넌트 즉, 네트워크에 존재하는 공용 컴포넌트 레포지토리로부터 다운로드가 가능한 컴포넌트로 구성되도록 하고 있다. 컴포넌트 코드는 동적으로 적재될 수 있는 라이브러리에서 캡슐화되는데, 그러한 것은 동적 연결을 가능케 한다. 그러나 컴포넌트 서비스를 위한 레포지토리 구축이 이루어지지 않아, 안정적인 컴포넌트간 의존성을 지원하는 시스템 안정성 부분에서는 일부 부족한 면이 있다.

2.3 컴포넌트 Configurator 클래스

컴포넌트간 동적 의존성의 명확한 표현은 실행시간 각각의 컴포넌트에 부착된 특별 객체를 통해서 이루어진다. 이러한 대상을 컴포넌트 Configurator라고 불린다. 이는 특정 컴포넌트를 위한 실행시간의 의존성을 구체화하고, 다른 컴포넌트로부터 오는 이벤트들을 다루기 위한 정책을 실행하는 책임을 진다[2].

실행시간 의존성의 구체화는 하나의 컴포넌트 Configurator 객체를 각각의 컴포넌트에 할당시킴으로써 완수 된다. Pseudo C++에 있는 컴포넌트 Configurator의 단순화된 선언이 뒤 따르도록 하였다[2, 5].

[5]에서 제안한 ComponentConfigurator abstract class 알고리즘에서 클래스 생성자는 매개 변수로서 컴포넌트 실행에 대한 포인터를 받는다. 이것은 나중에 implementation() 메소드를 통하여 얻어질 수 있다. 그러나 알고리즘은 C++로 작성된 것으로 현재의 동적 의존성을 지원하기에는 부족한 면이 있어 이를 개선하고 조정하는 커스터마이징(Customizing) 작업이 필요하다.

2.4 컴포넌트 간 동적 의존성

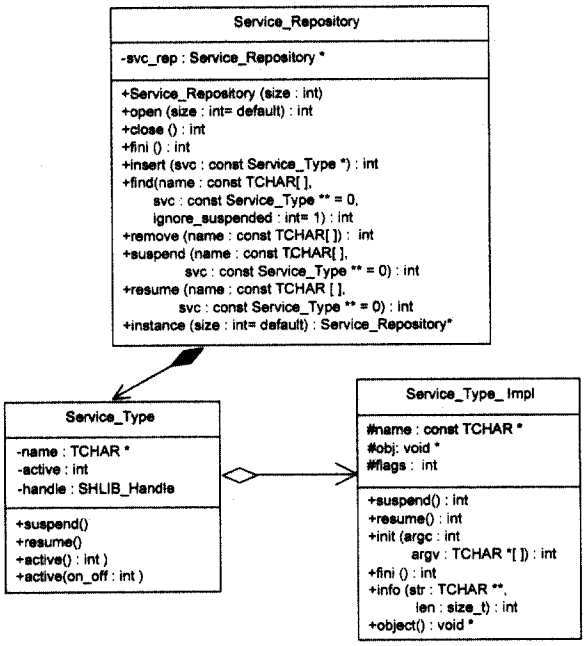
특정한 컴포넌트와 다른 시스템 사이의 의존성과 응용 프로그램 컴포넌트를 저장하는 일에 책임을 지는 컴포넌트 Configurator에 의해 각각의 컴포넌트는 관리된다[2, 4]. 구현되는 방법에 따라 의존되는 컴포넌트 Configurator는 하나의 번지 공간, 다른 번지 공간과 프로세스, 또는 분산 시스템에 있는 다른 하드웨어에서 실행되는 컴포넌트를 참조할 수 있다.

컴포넌트 Configurator는 또한 독립된 컴포넌트 사이에 이벤트들을 분산시키는 책임을 진다. 일반적 이벤트는 클라이언트의 파괴, 내부적 재배치, 또는 연결된 컴포넌트 구현의 교체이다. 이러한 이벤트는 모든 의존하는 컴포넌트에 영향을 미치는 것이다. 컴포넌트 Configurator는 이러한 형상과 연결된 이벤트를 다루기 위하여 프로그래머가 코드를 삽입하는 곳이다. 컴포넌트 개발자들은 특정 컴포넌트의 특성을 아는 전문화된 컴포넌트 Configurator 버전을 프로그램 할 수 있다. 따라서 이러한 전문화된 Configurator는 응용 프로그램 명세화적 방식으로 컴포넌트 의존성을 다루기 위해 개별화된 정책으로 구현할 수 있다.

3. 컴포넌트 서비스 레포지토리 설계

컴포넌트 서비스의 경우 Configurator 프레임워크를 사용하여 관리되는 모든 서비스들에 대한 중앙 레포지토리를 제공하고, 응용 프로그램에 이미 설정된 모든 서비스를 배치하고, 상태 보고 및 제어를 하기 위한 메소드를 제공할 필요가 있다. 이러한 기능을 수행하도록 하는 것이 컴포넌트 서비스 레포지토리이다[6]. 이는 컴포넌트 서비스 프레임워크가 객체에 의해 제공되는 서비스에 대한 접근 기능을 효과적으로 활용하도록 하기 위해서는, 응용 프로그램 서비스는 반드시 서비스 정보를 잘 알려진 레포지토리에 저장해야 하며, 이들 서비스 객체들을 개별적으로 또는 묶음 형태로 접근하고 제어할 수 있어야만 한다. 다음은 컴포넌트가 서비스될 때 중앙 레포지토리에서 지원할 수 있도록 하기 위하여 설계한 컴포넌트 서비스 레포지토리의 클래스를 나타내고 있다(그림 1).

컴포넌트 서비스 레포지토리 클래스는 응용 프로그램에 설정되는 모든 서비스 구현을 추적하고, 활성화 또는 일시 중지와 같은 서비스 상태를 개별적으로 관리한다. 컴포넌트 서비스 Configurator 프레임워크가 서비스를 추가, 관리, 제거하기 위한 수단을 제공하며, 초기화되었을 때와는 반대의 순서로 차례차례 모든 서비스들을 중단시킬 수 있는 편리한 명령을 제공하며, 각각의 서비스가 자신의 명칭을 사용하여 검색할 수 있도록 하는 기능이 있다. 이러한 컴포넌트 서비스 레포지토리 클래스의 핵심 메소드 설명을 다음과 같이 요약한다<표 1>.



(그림 1) 컴포넌트 서비스 레포지토리 클래스

<표 1> 컴포넌트 서비스 레포지토리 클래스 메소드

메소드 명	메 소 드 설 명
Service_Repository() open()	저장소를 초기화하고 자원들을 할당
Service_Repository() close()	저장소와 저장소 안의 동적 할당된 자원들을 모두 해제
insert()	저장소의 새로운 서비스를 추가
find()	저장소 내 특정 서비스에 대한 진입점을 위치
remove()	저장소 내에 등록되어 있는 서비스를 제거
suspend()	저장소 내의 특정 서비스의 실행을 일시 정지
resume()	저장소 내에서 일시 정지된 서비스의 실행을 재개
instance()	단일 객체 포인터를 반환하는 정적 메소드

4. 컴포넌트 의존성 알고리즘

2.3절에서 논의했던 ComponentConfigurator abstract class 알고리즘의 기본적인 기능을 커스터마이징 하여 응용 프로그램의 다른 문맥을 재사용할 수 있도록 하였다.

4.1 응용 프로그램 명세

동적 의존 정보는 컴포넌트의 재형상을 가능하게 한다. 이 구조가 자체적으로 안전한 재형상을 보장하지 못한다 할지라도 안전한 재형상을 구현하는 프로그래머에게 더 쉽

고, 단순하게 가치 있는 프레임워크를 제공할 것으로 기대된다.

다음은 *eventFromHookedComponent* 메소드가 JVM의 *ComponentConfigurator*로 부터 *REPLACED* 이벤트를 가지고 대체할 수 있는 것을 나타내고 있다.

```
int WebBrowserConfigurator :: eventFromHookedComponent
(ComponentConfigurator *cc, Event e)
{
  if (cc == JVMConfigurator)
   {
    if (e == REPLACED)
      try {
       FrozenObjs fo = currentJVM->freezeAllObjs () ;
       currentJVM = JVMConfigurator->implementation () ;
       currentJVM->meltObjects (fo) ;
      }
      catch (Exception exp)
      throw ReconfigurationFailed(exp) ;
   }
  else ···
}
```

(그림 2) eventFromHookedComponent 메소드의 커스터마이징

4.2 로킹 Configurator

[5]에서 구현했던 C++ 컴포넌트 Configurator의 대부분의 기본 버전은 *SimpleConfigurator*라 불린다. 이는 다중 쓰레드에 의해 컴포넌트 Configurator가 현재 접근하는 어느 것도 지원하지 않는다. 지원하는 것은 *LockingConfigurator*이라 불리는 서브 클래스로 구현되었다.

*LockingConfigurator*는 두개의 잠금(Lock)을 사용한다. 첫 번째는 의존에 접근하여 읽기(Read)와 쓰기(Write)를 통제하고, 두 번째는 컴포넌트 구현이다. 이 모두 다중 동시 읽기를 허용하거나(쓰기는 금지), 단일 사용자의 쓰기를 허용한다.

listHook() 메소드는 읽기 모드에서 첫 번째 획득한 의존 잠금에 의하여 구현된다. 그러면, 슈퍼 클래스로 동등한 메소드를 불러들이고, 잠금을 풀어준다.

```
inline DependencyList *listHooks ()
    {
      dependenciesLock_.acquire_read() ;
      DependencyList *ret = SimpleConfigurator :: listHooks() ;
      dependenciesLock_.release () ;
      return ret ;
    }
```

(그림 3) Listhook의 처리 메소드

한편, *registerClient()* 메소드는 쓰기 모드에서 잠금을 획득한다.

```
inline int registerClient(ComponentConfigurator * client, const
char *hookNameInClient)
   {
     dependenciesLock_.acquire_write() ;
     int ret = SimpleConfigurator :: registerClient(client,
     hookNameInClient) ;
     dependenciesLock_.release () ;
     return ret ;
   }
```

(그림 4) Registerclient의 잠금 획득 메소드

이와 같은 방법으로 프로그래머는 컴포넌트의 구현에서 잠금을 걸 수 있고, 잠긴 동안 어떠한 변경도 없이 안전하게 작업할 수 있다.

4.3 동적 재형상 인터페이스 알고리즘

동적 재형상 인터페이스 알고리즘은 동적 재배치 자원관리를 위한 컴포넌트의 운영을 위한 알고리즘이다. 기능은 *abstraction, namely, categories, implementation, hooks, configurable* 컴포넌트를 실행할 수 있도록 설계하였다.

```
interface DynamicConfigurator
{
 typedef sequence<string> stringList ;
 typedef sequence<octet> implCode ;

 stringList list_categories () ;
 stringList list_implementations (in string categoryName) ;
 stringList list_loaded_implementations ()
 stringList list_domain_components () ;
 stringList list_hooks (in string componentName) ;
 string get_impl_info (in string implName) ;
 string get_comp_info (in string componentName) ;
 string get_hooked_comp (in string componentName,
                         in string hookName) ;
 string get_latest_version (in string categoryName) ;

 long load_implementation (in string categoryName,
                           in string impName,
                           in string params
                           in Configuration :: Factory factory,
                           out Configuration ::
                           ComponentConfigurator cc) ;

 void hook_implementation (in string loadedImpName,
                           in string componentName,
                           in string ;
 void suspend_implementation (in string loadedImpName) ;
 void resume_implementation (in string loadedImpName) ;
 void remove_implementation (in string loadedImpName) ;
```

```
  void configure_implementation (in string loadedImpName,
                                 in string message) ;

  void upload_implementation (in string categoryName,
                              in string impName,
                              in implCode binCode) ;
  void download_implementation (in string categoryName,
                                inout string impName,
                                out implCode binCode) ;
  void delete_implementation (in string categoryName,
                              in string impName) ;
} ;
```

(그림 5) The ReconfigurationCongurator Interface

알고리즘에서 처음 9개 오퍼레이션(list_categories, list_implementations, list_loaded_implementations, list_domain_components, list_hooks, get_impl_info, get_comp_info, get_hooked_comp, get_latest_version)은 도메인의 동적 구조를 검사하는 기능이 있고, 다른 추상에 관한 정보를 검색한다. *category*는 컴포넌트의 형태를 나타내고, 각 *category*는 서로 다른 *implementation*을 포함한다. 일단 시스템 실행시간으로 로드된 *implementation* 오퍼레이션은 *loaded implementation*이 된다. 마침내 컴포넌트는 컴포넌트가 A가 컴포넌트 B에 의존된다면, 이 의존성은 A에 있는 *hook*로 B가 참가되는 모양의 형태를 띠는 컴포넌트 간 의존성을 표현하는데 사용되는 *hook*을 갖는다. *load_implementation* 오퍼레이션은 컴포넌트 서비스 레포지토리로부터 implementation을 동적으로 로드하고 시작한다. *hook_implementation* 오퍼레이션은 컴포넌트에 *hook*를 첨가한다. 다음의 4가지 메소드(suspend_implementation, resume_implementation, remove_implementation, configure_implementation)는 *loaded implementation*을 작동한다. 즉 메인 쓰레드를 중지하여 재시작하고, 프로세스로부터 제거하며, 컴포넌트 명세 재형상 메시지를 보낸다. *upload_implementation* 오퍼레이션은 외부 엔티티를 *implementation*으로 보내고 컴포넌트 서비스 레포지토리에 저장한다. 반대로 말하면, *download_implementation* 오퍼레이션은 원격 엔티티를 컴포넌트 서비스 레포지토리로부터 *implementation*을 검색한다. 마지막으로 *delete_implementation* 오퍼레이션은 컴포넌트 서비스 레포지토리에 저장된 *implementation*을 삭제한다.

5. 실험결과 및 비교분석

5.1 실험 환경

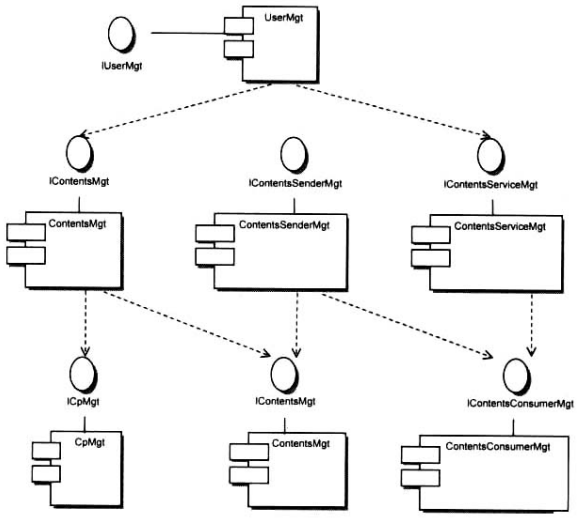
설계한 컴포넌트 서비스 레포지토리의 실험을 위하여 만든 실험 시스템은, 이미 만들어진 컨텐츠를 수집하여 고객의 요구에 맞는 컨텐츠를 제공하는 컨텐츠 관리 시스템이다. 이는 가치 있는 정보를 컨텐츠 제공자(Contents Provider)로부터 제공 받아 이를 필요로 하는 임의의 컨텐츠 사용자(Contents Consumer)에게 다양한 분류 및 검색 서비스를 통하여 컨텐츠를 제공하는 것을 주요 기능으로 한다. 컨텐츠 브로커(Contents Broker)는 컨텐츠 제공자와 컨텐츠 사용자를 연결하여 주는 중개자 역할을 한다.

컨텐츠 관리 시스템에서 사용된 컴포넌트는 총 7개이고,

〈표 2〉 예제 시스템 컴포넌트 종류 및 설명

번 호	컴포넌트 명	컴 포 넌 트 설 명	컴포넌트 크기 (kbyte)	오 퍼 레 이 션
1	UserMgt	시스템을 사용할 수 있는 사람	32	getId, getPasswd
2	ContentsMgt	컨텐츠를 관리하는 사람	31	
3	ContentsSenderMgt	컨텐츠를 제작하여 제공하는 주체	38	queryList, insert, delete
4	ContestsServiceMgt	컨텐츠 사용자가 신청한 서비스 제공방식	30	queryServiceType
5	CpMgt	컨텐츠 제공자가 수행하는 업무	40	queryContentsProviderList, insertContentsProvider, deleteContentsProvider
6	ContentsMgt	제공되는 각종 컨텐츠	44	queryList, insert, delete, update
7	ContentsConsumerMgt	컨텐츠를 사용하는 사람	48	checkDupId, insertConsumer, updateConsumer, queryConsumerList, queryContentsProviderList, queryUserContentsList, insertContents, deleteContents, queryContentsList

각각의 컴포넌트 설명은 <표 2>와 같고, 컴포넌트 다이어그램은 (그림 6)과 같다. 또한 실험을 하기 위한 시스템 OS는 Windows 2000 Professional이고, 팬티엄 Ⅲ 프로세서에서 실험이 이루어졌다.

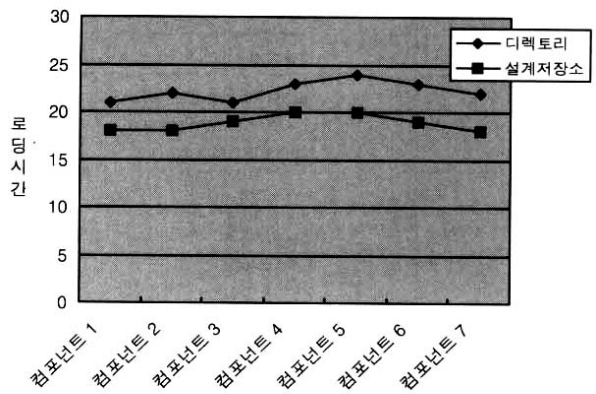


(그림 6) 예제 시스템 컴포넌트 다이어그램

실험은 먼저, 각 컴포넌트가 로드되는 시간을 측정하기 위하여, 일반 디렉토리 상에 놓고 시간을 측정하였고, 두 번째로 서비스 레포지토리에 넣고 시간을 측정하였다. 또한 커스터미이징한 알고리즘을 클래스로 만들어 적용하여 인터페이스 되는 시간을 측정하였다.

5.2 실험 결과

컴포넌트 로딩시간을 측정하기 위하여 7개의 컴포넌트가 로딩되어 시스템에 적재하는 시간을 총 5회 측정하여 평균값으로 계산하고, 이를 비교한 결과가 (그림 7)에 나타나 있다. 실험결과 평균 3ms에서 4ms까지의 로드시간의 차이



(그림 7) 예제 시스템 컴포넌트 로딩 시간

를 보이고 있다. 일반 디렉토리 대비 레포지토리와의 차이가 크지는 않지만, 레포지토리에 저장된 컴포넌트의 수가 많아 로드되는 컴포넌트 수가 많으면 시스템에 컴포넌트의 로드시간이 문제가 될 수 있음을 보여준다.

5.3 알고리즘 비교분석

기존 연구[5]와 제안된 알고리즘과 비교분석을 하였다. 이를 통하여 제안된 알고리즘의 안정성 및 객관성을 검증하고자 한다. 비교 항목은 자바 가상 머신(Java Virtual Machine) 지원여부, 메소드의 수, 변수의 수, 인터페이스 표현 방식, 알고리즘의 구성항목, 컴포넌트의 컴포넌트 Configurator에 대한 접촉 횟수로 구분하여 비교하였다.

〈표 3〉 기존 알고리즘과의 비교

항목 \ 알고리즘	기존 연구	제안 알고리즘
JVM 지원	미 흡	지 원
메 소 드	2	4
인터페이스	unspecified	specified
구 성 항 목	함수형 데이터	클래스 상세정보
접 촉 횟 수	9	9

제안된 알고리즘은 메소드나 접촉횟수는 기존 연구와 비슷하지만 JVM을 지원하고 구성항목을 클래스로 표현하기 때문에 재사용성이 높게 평가 되었다. 더욱이 JVM의 지원이 용이하여 JVM이 설치된 클라이언트에서의 사용이 용이해지는 특징을 가지고 있다.

6. 결 론

특정한 컴포넌트와 다른 시스템 사이의 의존성과 응용 프로그램 컴포넌트를 저장하는 일에 관여하는 컴포넌트 Configurator에 의해 각각의 컴포넌트는 관리된다. 컴포넌트 Configurator는 또한 독립된 컴포넌트 사이에 이벤트들을 분산시키는 책임을 진다. 일반적 이벤트의 예는 클라이언트의 파괴, 내부적 재배치, 또는 연결된 컴포넌트 구현의 교체이다. 그러한 이벤트는 모든 의존하는 컴포넌트에 영향을 미치는 것이다. 그러나 이러한 컴포넌트 의존성을 지원하는 컴포넌트 서비스의 경우 Configurator 프레임워크를 사용하여 관리되는 모든 서비스들에 대한 중앙 레포지토리를 제공하고, 응용 프로그램에 이미 설정된 모든 서비스를 배치하고, 상태 보고 및 제어를 하기 위한 메소드를 제공할

필요가 있다. 이러한 기능을 수행하도록 하는 것이 컴포넌트 서비스 레포지토리이다. 이는 컴포넌트 서비스 프레임워크가 객체에 의해 제공되는 서비스에 대한 접근 기능을 효과적으로 활용하도록 하기 위해서는, 응용 프로그램 서비스는 반드시 서비스 정보를 잘 알려진 레포지토리에 저장해야 하며, 이들 서비스 객체들을 개별적으로 또는 묶음 형태로 접근하고 제어할 수 있어야 한다. 본 논문에서는 컴포넌트 서비스 레포지토리의 클래스를 설계하여 실험하였고, 실험결과 평균 3ms에서 4ms까지의 로드시간의 차이를 보였다. 또한 제안한 알고리즘을 비교하여 분석한 결과 제안된 알고리즘은 JVM을 지원하고 구성항목을 클래스로 표현하기 때문에 재사용성이 높게 평가 되었다. 더욱이 JVM의 지원이 용이하여 JVM이 설치된 클라이언트에서의 사용이 용이해지는 특징을 가지고 있다. 향후 연구 방향은 컴포넌트를 지원하는 형상관리 도구의 형상관리자를 설계하여 구현할 때, 컴포넌트 의존성을 지원하는 컴포넌트 Configurator의 구축하여 종합적인 형상관리자의 구현까지 이어지도록 하여야 할 것이다.

참 고 문 헌

[1] Magnus Larsson, Ivica Crnkovic, "Configuration Management for Component-based Systems," In Software Configuration Management(SCM-10), 23th ICSE Toronto, Canada, May, 2001.

[2] Fabio Kon, Tomonori Yamane, Christopher Hess, Roy Campbell, and M. Dennis Mickunas, "Dynamic Resource Management and Automatic Configuration of Distributed Component Systems," Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001), San Antonio, Texas, January, 2001.

[3] 송영재, 김귀정, 변정우, 서영준, 최한용, 한정수, "객체지향 모델링과 CBD 중심 소프트웨어 공학", 이한출판사, p.130, 2003.

[4] Fabio Kon and Roy H. Campbell, "Supporting Automatic Configuration of Component-Based Distributed Systems," In proceeding of the 5th USENIX Conference on Object-Oriented Technologies and Systems(COOTS'99), San Diego, CA. pp.175-187, May, 1999.

[5] Fabio Kon, Roy Campbell, "Dependence Management in Component-Based Distributed Systems," IEEE Concurrency, 8(1), pp.26-36, January-March, 2000.

[6] Douglas Schmidt, Frank Buschmann, Hans Rohnert, Michael Stal, "Pattern-Oriented Software Architecture Vol. 2 : Patterns for Concurrent," Chapter 2, John Wiley & Sons, 2000.

[7] Schmidt Douglas C, Huston Stephen D, "C++ Network Programming Vol.2 : Systematic Reuse with ACE and Framework," Addison Wesley, 2003.

[8] Dilma M. Silva and Fabio Kon and Roy Campbell, "Dynamic Configuration of a Directory Service Using the Component Configurator Framework," Proceedings of the Project SIDAM Workshop, September, 2000.

[9] Magnus Larsson, Ivica Crnkovic, "New Challenges for Configuration Management," In System Configuration Management, SCM-9, 1999.

[10] Magnus Larsson, "Applying Configuration Management Techniques to Component-Based Systems," MRTC, 2000.

[11] Francisco Assis Rosa and Antonio Rito Silva, "Component Configurer : A Design Pattern for Component-Based Configuration," 1997.

[12] Stefan Rock, Alexander Gierak, "Evaluating and Extending the Component Configurator Pattern," Hasso-Plattner-Institute for Software Systems Engineering, Germany, 2001.

[13] Eddy Truyen, Bo Norregaard Jorgensen, Wouter Joosen, "Customization of Component-based Object Request Brokers Through Dynamic Reconfiguration," in Proceedings of TOOLS Europe'2000, IEEE, pp.181-194, June, 2000.

[14] Eiichi Sunagawa, Kouji Kozaki, Yoshinobu Kitamura, Riichiro Mizoguchi, "Management of dependency between two or more ontologies in an environment for distributed development," Proceedings of the International Workshop on Semantic Web Foundations and Application Technologies, March 12th, Japan, 2003.

[15] Carl Lebsack, Austin Mroczek, and Carl Mueller, "Controlling Configuration Items in Component Based Software Development," In Software Configuration Management(SCM-10), 23th ICSE Toronto, Canada, May 2001.

[16] R. Balter, L. Bellissard, F. Boyer, M. Riveill, and J.Y. Vion-Dury. Architecturing and Conguring Distributed Applications with Olan, In Proc. IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing(Middleware'98), The Lake District, UK, September, 1998.

최 상 균

e-mail : skchoi@kimpo.ac.kr

1986년 한남대학교 전자계산공학과 공학사
1993년 서강대학교 정보처리학과 이학석사
2000년 경희대학교 전자계산공학과 박사 과정 수료
1986년~1997년 한국생산기술연구원 선임 연구원
1998년~현재 김포대학 컴퓨터계열 조교수
관심분야 : CBSE, 소프트웨어 재공학, 소프트웨어 품질

송 영 재

e-mail : yjsong@khu.ac.kr

1969년 인하대학교 전자공학과 공학사
1976년 일본 Keio 대학교 전산학과 공학석사
1980년 명지대학교 전산학과, 공학박사
1982년~1983년 미국 Maryland대학교 객원교수
1986년~1988년 대한전자공학회 전자계산연구회전문위원장
1984년~1989년 전국 전산소장 협의회 부회장
1990년~1991년 일본 Keio 대학교 객원교수
1984년~1989년 경희대학교 전자계산소장
1993년~1995년 경희대학교 교무처장
1996년~1998년 경희대학교 공과대학장
1999년~2000년 경희대학교 기획조정실장
2001년~2002년 경희대학교 산업정보대학원장
1976년~현재 경희대학교 교수
관심분야 : 소프트웨어공학, CBSE, CASE 도구, S/W 재사용