서비스 로봇용 결함 허용 미들웨어

Fault-Tolerant Middleware for Service Robots

백 범 현, 박 홍 성*
(Bum-Hyeon Baek and Hong-Seong Park)

Abstract : Recently, robot technology is actively going on progress to the field of various services such as home care, medical care, entertainment, and etc. Because these service robots are in use nearby person, they need to be operated safely even though hardware and software faults occur. This paper proposes a Fault-Tolerant middleware for a robot system, which has following two characteristics: supporting of heterogeneous network interface and processing of software components and network faults. The Fault-Tolerant middleware consists of a Service Layer(SL), a Network Adaptation Layer(NAL), a Network Interface Layer(NIL), a Operating System Abstraction Layer(OSAL), and a Fault-Tolerant Manager(FTM). Especially, the Fault-Tolerant Manager consists of 4 components: Monitor, Fault Detector, Fault Notifier, and Fault Recover to detect and recover the faults effectively. This paper implements and tests the proposed middleware. Some experiment results show that the proposed Fault-Tolerant middleware is working well.

Keywords : service robots, fault-tolerant, robot middleware, reusability, lightweight

I. 서론

근래의 로봇 기술은 산업 현장뿐 아니라, 가사 보조, 의료 보조, 엔터테인먼트 등의 다양한 서비스 분야로의 적용이 활발히 진행되고 있다. 이러한 서비스 분야에 적용되는 서비스 로봇은 인간에게 다양하고 편리한 서비스를 제공하기 위한 로봇으로써 특히 가정에서 많이 사용될 수 있는 로봇이다[1]. 가정에서 사용되는 서비스 로봇은 사람과 가까운 곳에서 동작하기 때문에 안전하게 동작을 수행하는 것이 중요하다. 왜냐하면 로봇에 오류가 생기면 동작이 불안하게 되어 사람에게 상해를 입힐 수 있기 때문이다. 예를 들어, 안내용 로봇이 있다고 할 때, 안내용 로봇이 하드웨어나 소프트웨어의 결함으로 사람과 충돌을 일으키면 사람이 다칠 가능성이 있다. 따라서 로봇에 결함이 발생하였을 때 결함으로부터 발생할 수 있는 비정상적인 동작을 사전에 방지하고, 결함이 발생하더라도 안전한 동작을 수행할 수 있는 결함 허용 로봇 시스템에 대한 연구가 필요하다. 여기서 결함 허용이란 어떠한 부분에 장애가 생겼을 때 예비 요소나 절차가 즉시 그 역할을 대체 수행함으로써 서비스의 중단이 없도록 하는 것을 말한다.

일반적으로 결함 허용 로봇 시스템은 소프트웨어 혹은 미들웨어에 의해 구현된다. 결함 허용성(fault tolerance)이란 하드웨어 혹은 소프트웨어 고장이 발생하더라도 설계상에 명시된 내용에 따라 수행할 수 있는 능력을 말하는 것으로써 이는 모든 고장에 완전하게 대처할 수는 없다 할지라도 최소한 설계 시에 명시된 고장에 대해서는 완벽하게 대응할 수 있는 능력을 말한다[2]. 서비스 로봇에 결함 허용 기능을 포함시킬 때는 서비스 로봇의 특징과 관련하여 많은 것들을 고려해야 한다. 서비스 로봇에는 청소용 로봇부터 안내용 로봇까지 매우 다양한 종류가 있고 여러 형태의 네트워크가 사용될 수 있기 때문에 결함 허용 미들웨어는 다음과 같은 특성을 가져야 한다.

- 이기종 네트워크 인터페이스 지원
- 컴포넌트 및 네트워크 결함 처리

다양한 네트워크 환경에 적응하기 위하여 이기종 네트워크 인터페이스를 제공하여 효율적인 통신 환경을 제공해야 한다. 이는 같은 결함에 대해 동시에 그리고 반복적으로 발생하는 경우를 막을 수 있다. 그리고 로봇 내부 시스템의 컴포넌트 및 네트워크 이상에 대해 감지할 수 있는 능력을 가지고 있어야 하며, 감지된 이상을 격리, 복구할 수 있어야 한다. 이러한 이유는 하드웨어 혹은 소프트웨어에서 오류(fault)가 발생되면, 이 오류는 다양한 에러들을 만들어 내고, 이에 따라 시스템은 많은 오류 감지 메시지를 전송하게 되어 시스템의 오류 제어 동작을 막게 되기 때문이다. 또한 네트워크 세션을 관리함으로써 네트워크의 비정상적인 종료에 대응할 수 있어야 한다.

로봇용 미들웨어에 대한 연구는 여러 연구자들이 진행하고 있다[3-5,9]. 하지만 이러한 미들웨어들은 결함허용을 지원하지 못하고 있다. 일반적인 미들웨어로 결함 허용을 지원하는 미들웨어로는 분산 시스템용으로 사용되는 FT-CORBA (Fault-Tolerant CORBA[6], ROAFTS(Real-time Object-oriented Adaptive Fault Tolerance Support)[7], Rainbow framework (Architecture-Based Self-Adaptation with Reusable Infrastructure)[8] 와 같은 종류가 있다. 이러한 시스템들은 결함 허용성을 지원하기 위해 중복(redundancy) 혹은 복제(replication) 메커니즘을 사용하고, TCP/IP를 기반으로 한 프로토콜을 사용하고 있다. 하지만 이 시스템들은 범용적인 분산 시스템에서는 많이 사용되고 있지만, 서비스 로봇에는 적용시키기 어려운 점이 존재한다. 이 미들웨어들은 TCP/IP 네트워크 만을 지원하기 때문에 로봇에서 사용되는 다양한 데이터를 처리하는 데에 효율성이 저하되고 같은 결함이 반복적으로 발생할 수 있으며, 네트워크에 이상이 발생할 때 다른 네트워크를 사용하는 것과 같은 즉각적인 대응이 어렵다. 또한 결함 허용성을 위

* 책임저자(Corresponding Author)
논문접수 : 2007. 11. 6., 채택확정 : 2008. 1. 30.
백범현, 박홍성 : 강원대학교 전자통신공학과
(bhbaek@control.kangwon.ac.kr/hspark@kangwon.ac.kr)
※ 본 연구는 강원대학교 BK21 사업 및 지식경제부의 지원에 의하여 연구되었음.

한 프로토콜이 사용하는 메시지의 크기가 크기 때문에 오버헤드가 크다.

본 논문은 다양한 서비스 로봇 시스템에 사용 가능한 결함 허용 미들웨어를 제안한다. 제안하는 결함 허용 미들웨어는 다양한 로봇에 적용할 수 있도록 컴포넌트 기반으로 이기종 네트워크 인터페이스를 지원하며 경량형으로 설계한다. 이 미들웨어의 주요 컴포넌트인 결함 허용 관리자는 미들웨어의 컴포넌트, 응용 컴포넌트와 이기종 네트워크 인터페이스의 결함 발생 여부를 감지하고 대응을 하게 한다. 이를 위하여 결함 허용 관리자는 모니터(monitor), 결함 발견기(fault detector), 결함 통보기(fault notifier), 결함 복구기(fault recover)의 4개의 컴포넌트로 구성된다.

다음 장에서는 본 논문에서 제안하는 결함 허용 미들웨어 구조에 대해 간략하게 설명하고, 3장에서는 결함을 감지하고 처리하게 하는 결함 허용 관리자의 구조와 구성 요소들을 설명한다. 4장에서는 제안한 시스템들을 구현하여 그 성능을 분석하고, 마지막으로 5장에서는 결론을 맺는다.

II. 서비스 로봇을 위한 결함 허용 미들웨어

본 논문에서 제안한 결함 허용 미들웨어는 [9]를 기본으로 확장한 것이다. [9]는 모듈 단위로 이루어진 로봇에 적재되는 로봇용 미들웨어로써, 모듈은 로봇을 이루는 하드웨어 단위이고, 독립적으로 추가, 삭제, 확장, 교환이 가능하다. 로봇에서 모듈간 인터페이스는 하나 이상의 종류가 사용될 수 있기 때문에 각 모듈은 Ethernet, IEEE1394[10], USB[11], CAN[12], RS232 등의 이기종 네트워크 인터페이스를 포함하여 언제든지 다른 종류의 인터페이스들과 연결할 수 있다. 이러한 구성의 예가 그림 1에 나타나 있다.

본 논문에서 사용하는 결함 허용 미들웨어의 구조는 그림 2와 같고 상세 구조는 그림 3과 같다. 로봇 미들웨어는 다음과 같이 구성되어 있다: 응용에 미들웨어의 서비스를 제공하는 서비스 계층(Service Layer, SL), 종류가 다른 여러 네트워크를 수용하는 네트워크 적응 계층(Network Adaptation Layer, NAL), 네트워크 의존적인 기능을 담당하는 네트워크 인터페이스 계층(Network Interface Layer, NIL)의 세 계층과 운영체제에 독립적인 기능을 제공하기 위한 운영체제 추상화 계층(Operating System Abstraction Layer, OSAL), 미들웨어에 결함

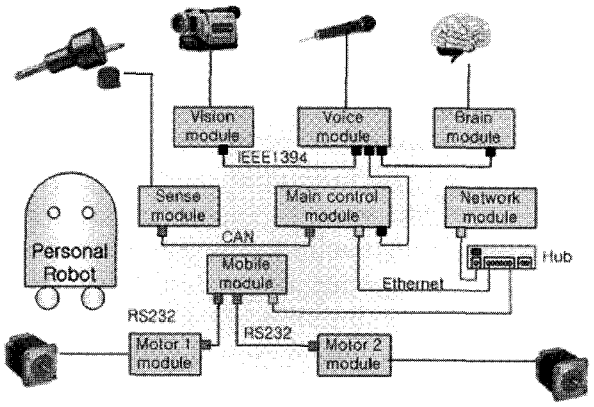


그림 1. 모듈 기반 로봇의 내부구조 예.
Fig. 1. Example internal structure of module-based robot.

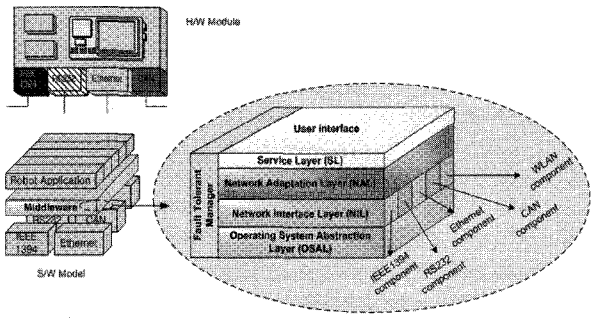


그림 2. 로봇 미들웨어 구조.
Fig. 2. Fault-tolerant middleware architecture for service robot.

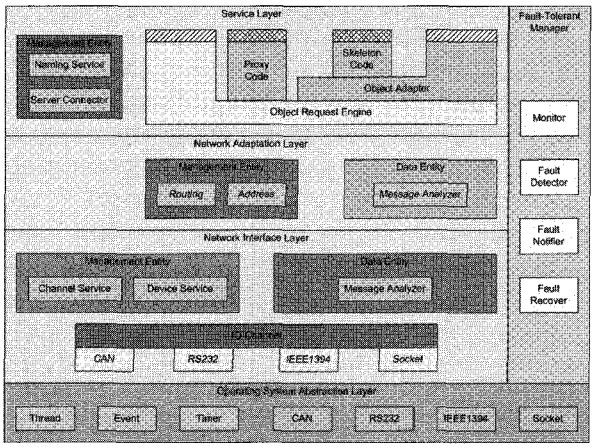


그림 3. 로봇 미들웨어 상세 구조.
Fig. 3. The detailed architecture of the fault-tolerant middleware.

허용성을 제공하기 위한 결함 허용 관리자(Fault-Tolerant Manager, FTM)로 구성되어 있다.

서비스 계층은 미들웨어의 서비스를 사용하는 어플리케이션에 관한 요소로 이루어져 있다. 주요 기능은 네트워크 적응 계층을 이용하여 어플리케이션/응용 컴포넌트가 네트워크의 종류에 상관없이 원격 모듈에 존재하는 어플리케이션/응용 컴포넌트에 접근하기 위한 메커니즘을 제공하는 것이다. 다시 말하자면, 원격 모듈에 존재하는 변수를 쉽게 읽거나 쓰기 또는 원격 객체를 호출하는 등의 서비스를 서비스 계층에서 제공하며, 이와 관련된 어플리케이션/응용 컴포넌트 관리, 트랜잭션 관리 등도 병행하여 실행한다.

네트워크 적응 계층은 네트워크 인터페이스 계층에 추가된 여러 종류의 네트워크 컴포넌트들을 통합하는 계층으로서, 이기종 네트워크간 메시지 라우팅, 이기종 네트워크간 모듈 어드레싱, 네이밍 서비스 등의 기능을 수행한다.

네트워크 인터페이스 계층은 여러 종류의 네트워크 컴포넌트가 존재하는 계층으로서 네트워크 종류에 따라 하드웨어, 소프트웨어에 의존적인 부분을 수행하는 컴포넌트로 구성되어 있다. 이러한 네트워크 인터페이스 계층은 미들웨어에서 새로운 네트워크 인터페이스가 추가되거나 삭제될 때 일일이 미들웨어를 수정하는 일이 없도록 하며, 각 네트워크의 의존적인 부분을 담당 및 관리할 수 있는 구조를 갖춘다.

운영체제 추상화 계층은 운영체제에 대한 독립성을 제공하는 계층으로서 운영체제가 제공하는 스레드, 타이머, 이벤

트 등과 같은 시스템 종속적인 기능을 추상화시킴으로써 동일한 API(Application Programming Interface)를 사용하여 운영체제의 API를 사용할 수 있도록 한다.

결함 허용 관리자는 로봇 어플리케이션과 미들웨어를 이루는 컴포넌트들을 관리하고, 모듈간 네트워크 및 통신을 감시한다. 결함 허용 관리자는 컴포넌트의 초기화(initialize), 적재(load), 구성(configuration), 종료(finalize) 등을 관리하고, 내부적으로 정의한 이벤트를 사용하여 내부 컴포넌트간 통신이 가능하게 함으로써 런타임에 각 컴포넌트 상태를 실시간으로 감시할 수 있다. 또한 예외처리 기능을 제공하여, 미들웨어와 어플리케이션에 발생할 수 있는 예외에 대처하며, 네트워크 연결 상태를 감시함으로써 데이터 송수신에서 발생할 수 있는 전송 오류를 사전에 방지하고, 지속적인 연결을 유지할 수 있도록 한다. 또한 네트워크의 고장을 빠르게 감지하게 된다.

위와 같은 결함 허용 미들웨어 구조에서는 사용자에게 필요한 네트워크를 그 종류에 상관없이 미들웨어에 쉽게 추가할 수 있고, 응용 개발자는 네트워크 상의 원격 또는 로컬에 존재하는 변수나 객체에 접근할 때 네트워크의 종류에 무관하게 미들웨어를 통해 손쉽게 응용 프로그램을 설계 및 구현할 수 있다. 또한 어플리케이션 등의 소프트웨어 컴포넌트 혹은 하드웨어 모듈의 동작의 이상 여부에 관계없이 안전하게 동작되도록 설계되어 있기 때문에 다양한 서비스 로봇의 응용 컨텐츠를 구현할 수 있다.

III. 결함 허용 관리자(fault-tolerant manager) 구조

결함 허용 관리자는 미들웨어와 어플리케이션에 결함 허용성을 제공하기 위한 컴포넌트로써 미들웨어 내에서 발생할 수 있는 결함으로부터 미들웨어 내에 존재하는 여러 객체들과 어플리케이션 객체들을 관리한다. 결함 허용 서비스는 소프트웨어 만에 의해 제공되는 경우도 있지만, 하드웨어에 내장된 형태 또는 몇 가지가 조합된 형태로 제공될 수 있다. 본 논문에서는 하드웨어 측면에서의 결함 허용은 고려하지 않고 소프트웨어 측면의 결함 허용만 고려한다.

결함 허용은 결함 감지가 중요한데, 소프트웨어에서 결함 감지는 하나의 트랜잭션 내에서 미리 정해진 시점에 중요한 데이터를 검사할 수 있도록 함으로써 이루어진다[13]. 이와 같은 방식으로 결함 허용 관리자는 트랜잭션을 수행하는 개체를 모니터링 하고, 결함과 에러와 같은 이상이 발생하면 시스템 동작에 독립적으로 복구시키는 역할을 수행한다.

미들웨어와 어플리케이션에서 트랜잭션을 수행하는 개체는 태스크나 객체가 대부분이다. 여기서 태스크란 어플리케이션에서 사용하는 기본 단위로써 어플리케이션이 될 수도 있고, 또는 계속되는 프로그램의 호출이 될 수도 있다. 한 어플리케이션이 여러 개의 다른 유틸리티 어플리케이션이나 프로그램들에게 요구를 할 수 있기 때문에, 태스크는 주로 스레드 단위로 동작한다. 객체는 미들웨어와 어플리케이션에서 생성되는 클래스의 인스턴스로써 하나의 객체는 단일 서비스를 제공하거나 다른 객체와 연동하여 서비스를 제공하기도 한다. 컴포넌트는 특정 서비스를 제공할 수 있는 단위로써, 단일 객체 또는 여러 객체가 복합(composite)된 형태이다.

따라서 시스템 동작에 기반이 되는 객체 및 태스크의 생존

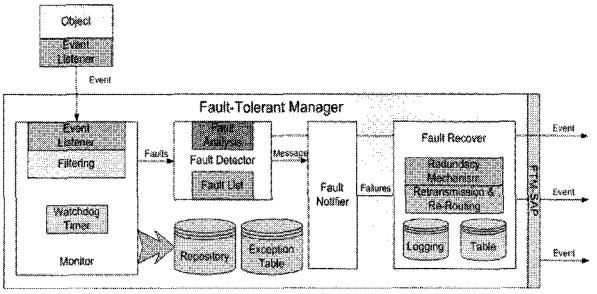


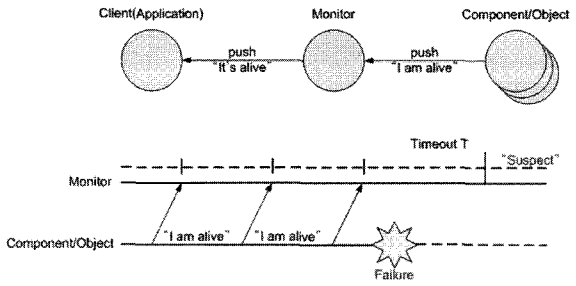
그림 4. 결함 허용 관리자 구조.

Fig. 4. A Structure of fault-tolerant manager.

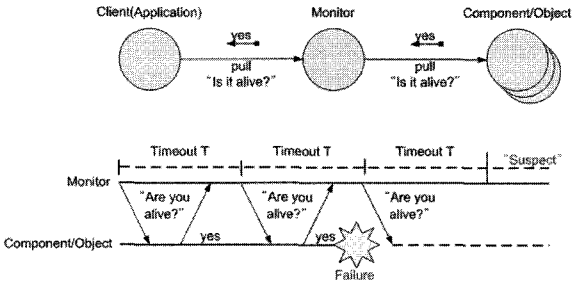
주기 및 상태를 점검하는 것은 결함 허용성을 제공하기 위한 기본적인 사항이 된다. 이러한 객체와 태스크들은 서비스 로봇에서 네트워크를 통해 다양한 서비스들과 상호 연동하고, 필요한 데이터를 송수신하기 때문에 네트워크에 대한 연결 관리가 필요하다. 그리고 네트워크 연결을 관리함으로써 네트워크 연결이 갑자기 종료되었을 때 대체 경로를 빠른 시간 내에 생성하고, 재전송까지의 시간을 단축시켜야 한다. 또한 모듈 단위로 이루어진 로봇 내에서 모듈 간의 의존성을 고려하여 대체 모듈 검색 및 사용을 원활히 해야 한다. 결함 허용 관리자는 이러한 요구사항에 결부하여 로봇 시스템을 결함으로부터 안전할 수 있도록 보장한다.

결함 허용 관리자는 시스템이 결함 허용성을 요구하는 소프트웨어에 사용될 수 있도록 재사용 가능한 컴포넌트 기반으로 설계되었다. 결함 허용 관리자는 복합 컴포넌트로 되어 있으며 모니터(monitor), 결함 발견기(fault detector), 결함 통보기(fault notifier), 결함 복구기(fault recover)의 4개의 컴포넌트로 구성된다. 그림 4는 결함 허용 관리자의 구조를 보여준다.

이들 중 결함 발견에 관한 사전 작업을 모니터에서 수행한다. 모니터는 미들웨어 내의 모든 객체와 어플리케이션의 서비스 객체, 네트워크 세션을 담당하는 객체들을 주기적으로 관찰함으로써 각 객체들의 생존 주기와 상태를 파악하는 개체이다. 각 객체들의 생존 주기 및 상태 파악은 다음의 두 가지 모델을 사용하여 가능하다: 첫 번째가 푸쉬(push) 모델이고, 두 번째가 풀(pull) 모델[14]. 푸쉬 모델은 객체가 능동적으로 자신의 상태를 알려주는 형태로써, 미들웨어에서 동작하는 능동적인(active) 객체(로봇 전체 시스템에 반드시 필요한 객체)나 어플리케이션 객체가 푸쉬 모델에 적용되는 것들이다. 능동적인 객체들은 모니터에 T 시간마다 주기적으로 "I am alive" 메시지를 전송한다. 모니터는 워치독(watchdog) 타이머를 사용하여 정해진 시간 내에 메시지가 도착하면 플래그를 설정하여 객체가 정상적으로 동작하고 있다고 판단한다. 만약 T 시간 내에 메시지가 전달되지 않는 다면 객체가 소멸된 것으로 간주하고, 객체 복구 요청을 수행한다. 여기서 시간 T는 모니터가 메시지를 수신해야 하는 시간을 말하며, 메시지가 T 시간 내에 도착하지 않으면 타이머는 만기되고 오류가 발생하였다고 가정하게 된다. 이러한 푸쉬 모델이 그림 5(a)에 설명되어 있다. 풀 모델은 객체가 수동적으로 자신의 상태를 알려주는 형태로써, 상태 정보 전송 요청을 수신하면 자신의 상태를 알려주는 형태이다. 미들웨어에서 동작하는 수동적인 객체가 풀 모델에 적용되는데 수동적인



(a) 푸쉬 모델



(b) 풀 모델

그림 5. 푸쉬/풀 모델.

Fig. 5. A push/pull model.

```
/* The Event base class */
class FTM_API Event : public FTMBasic::BaseObject
{
public:

    /* Constructor */
    Event();

    /* Destructor */
    virtual ~Event();

    /* Get the Publisher source which sent the event */
    Publisher* getSource() const;

    /* Tells if this Event is exactly of type EVENT */
    template <typename EVENT>
    bool is_type() { return (typeid(*this) == typeid(EVENT)); }

private:

    friend class Publisher;

    void setPublisher(Publisher* p);

    Publisher* publisher_;
};
```

그림 6. 이벤트 클래스.

Fig. 6. Event class.

객체들은 모니터로부터 "Are you alive?" 메시지를 수신하여 그에 대한 응답을 전송한다. 이 때 푸쉬 모델과 마찬가지로 시간은 T시간 이내로 전송해야 하고, 만약 모니터로부터 전송된 메시지의 응답이 도착하지 않으면 객체가 소멸되었다고 간주한다. 그림 5는 푸쉬/풀 모델의 동작 원리를 보여준다.

본 논문은 이벤트를 그림 6과 같이 정의하고, 결함 허용 관리자는 이러한 이벤트를 발행/구독(publish/subscribe) 방법을 사용하여 이벤트를 전송한다.

이벤트 타입은 발행자(publisher)에 의해 특정 타입을 설정하고 구독자(subscriber)는 수신하고자 하는 이벤트 타입을 이벤트 청취기(event listener)에게 등록을 하면 된다. 발행자가 이벤트를 전송하면 이벤트 청취기는 모든 이벤트를 수신한

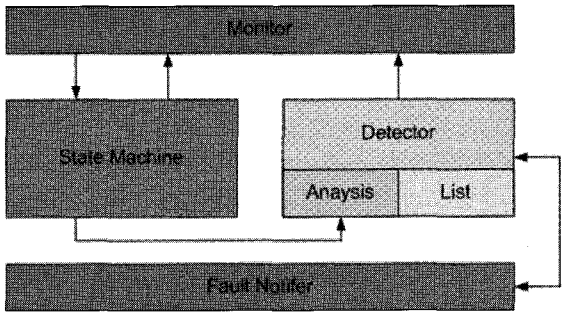


그림 7. 결함 발견기 구조.

Fig. 7. A structure of fault detector.

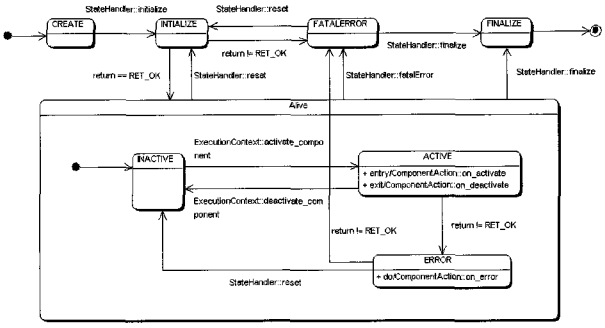


그림 8. 상태 머신.

Fig. 8. State machine.

후 등록된 이벤트 타입만 받아들이게 되고, 나머지 이벤트는 무시한다. 일반적으로 객체들은 자신의 상태를 전송하는 발행자이고, 모니터는 객체들의 상태를 수신하는 구독자와 동시에 이벤트를 전송하는 발행자이다. 모니터는 각 객체들로부터 수신한 객체의 상태를 저장소에 저장하고, 주기적으로 갱신한다. 또한 워치독 타이머를 사용하여 정해진 시간 내에 상태를 알려주는 메시지가 수신되지 않았을 때 이벤트를 발생시켜 결함 발견기로 결함 처리를 위임한다.

결함 발견기는 갑작스런 객체 소멸, 네트워크 종료, 객체 및 네트워크의 비정상적인 상태로의 전환과 같이 설계 시 명시된 결함 목록을 사용하여 잠재적인 결함을 발견하고, 기록하는 컴포넌트이다. 결함 발견기의 세부 구조는 그림 7과 같다.

결함 발견기는 상태 머신과 발견기 모듈로 나누어진다. 모니터로부터 객체의 상태를 수신한 후 상태 머신(state machine)에 사전에 정의되어 있는 상태와 비교한 후 객체의 상태가 에러(error), 치명적 오류(fatal error)일 경우 사전에 정의된 결함 목록을 토대로 비교/분석하여 발견기(detector)로 결과를 넘긴다. 발견기는 결함으로 판단된 객체의 정보를 결함 통보기로 전송한다. 각 개체는 데이터 수신을 확인하기 위해 확인 응답(confirm)을 수행한다.

그림 8의 상태 머신은 각 객체의 상태를 가리키는 매크로들을 포함하고 있으며, 각 상태에서 수행해야 할 메소드(method)들로 이루어져 있다. 그림 8은 상태 머신이 가지는 객체의 상태와 각 객체의 생존 주기(lifecycle)을 보여준다.

그림 8에서, 각 객체의 생존 주기는 객체 생성(create)을 시작으로 초기화(initialize)를 수행한다. 초기화 수행 후 객체는 활성화 상태(active)로 들어가고, 해당 오퍼레이션을 수행한다.

객체가 활성화 상태 동안 결함으로 인해 에러상태로 들어가면 해당 콜백 함수를 호출하여 결함 복구를 수행할 수 있게 한다. 만일 초기화가 실패하여 활성화되지 못할 경우 치명적 오류 상태로 들어가고, 종료함으로써 객체 생존 주기를 재시작한다. 객체는 처음 생성되고, 종료(finalize)될 때까지의 상태를 관리하며 초기화/활성/에러의 상태를 포함하고, 각 상태로 전이할 수 있는 메소드를 포함한다.

그림 7의 발견기는 상태 머신으로부터 객체 상태와 메소드를 사용하여 설계 시 명시된 결함 목록과 비교하여 결함 발생의 유무를 판단하며, 발견된 결함을 분석하는 작업을 수행한다. 발견기에서 수행하는 이 같은 과정은 기록된 결함을 분석하고, 무엇이 혹은 어디에서 결함이 발생하였는지 결정한다. 이 과정은 결함 격리(fault isolation) 혹은 결함 복구를 위해 필요하다. 발견기는 발생된 결함을 판단 및 분석 후 결함 복구를 위해 결함에 관한 정보를 결함 통보기에 전달하여 결함 복구기가 결함을 복구하게 하거나, 네트워크로 전달하여 결함 복구기가 결함을 복구하게 하거나, 네트워크로 전달하여 다른 모듈이 결함을 복구하거나 결함이 발생한 서비스를 대체할 수 있는 대체 서비스를 제공할 수 있게 한다.

그림 4와 그림 7의 결함 통보기는 발생된 결함 정보를 결함 복구기로 전송하는 연결기(connector)로써, 결함 발견기와 결함 복구기를 연결하는 컴포넌트이다. 결함 통보기는 컴포넌트 모델에서 사용되는 포트(port)[15]와 비슷한 개념으로써 로컬 결함 복구기 사이 혹은 로컬 결함 복구기와 원격 결함 복구기를 연결할 수 있게 한다.

그림 4의 결함 복구기는 발생한 결함의 종류에 따라 결함 복구 메커니즘을 사용하여 결함을 복구하거나 결함이 있는 개체의 동작을 정지시키는 개체이다. 결함 복구기는 다양한 결함 메커니즘을 사용할 수 있도록 인터페이스를 제공하며 제공되는 인터페이스를 사용하여 결함 복구 메커니즘을 확장 및 추가할 수 있다. 결함의 종류가 다양하기 때문에 이에 상응하는 결함 복구 메커니즘도 다양하다. 이렇게 다양한 결함 복구 메커니즘을 런타임 시에 모두 사용하는 것은 메모리에 부하를 줄 수 있으므로, 결함 복구기는 이러한 결함 복구 메커니즘들을 라이브러리 형태로 포함하여 발생된 결함의 종류에 따라 호출할 수 있게 한다.

IV. 구현 및 성능 측정

1. 구현 환경

본 장에서 제안된 시스템의 성능을 측정하고, 서비스 로봇에 적용한지 살펴본다.

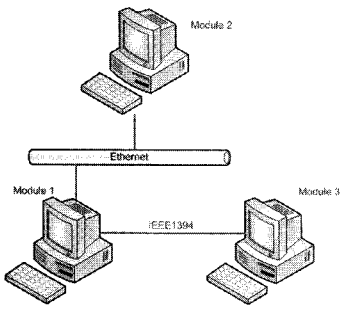


그림 9. 하드웨어 구성.

Fig. 9. Hardware configuration.

테스트 모듈은 실제 로봇이 없는 관계로 3개의 데스크탑을 이용하여 가상의 로봇 모듈들 설정하였다. 데스크탑의 CPU는 Pentium IV 3.0 GHz이고 메모리 용량은 1GHz이며, 네트워크 인터페이스로는 Ethernet과 IEEE1394가 장착되어 있다. 그림 9는 테스트 모듈들의 구성을 보여주고 있다. 제안된 시스템 및 로봇 어플리케이션은 C++ 언어를 이용하여 구현하였으며 OS는 구현상의 편의를 위해 Windows XP를 사용하였다.

2. 성능 측정

우선 모듈 내에 존재하는 미들웨어의 객체와 어플리케이션 객체의 상태 모니터링을 위해 객체와 모니터가 논리적으로 연결되어 이벤트를 전송하기까지의 시간을 측정하였다. 이벤트를 전송하여 수신하는 방법은 두 가지 형태로 나뉘는데, 첫 번째는 구독자가 자신이 관심 있는 이벤트의 필터링(Filtering) 없이 이벤트를 수신하는 방법이고, 두 번째는 구독자가 자신이 관심 있는 이벤트만을 수신하는 방법이다.

그림 10은 두 가지 방법을 사용하여 이벤트 생성을 위해 발행자와 구독자가 연결을 맺고, 이벤트를 전송하기까지 소모되는 전체 시간을 보여준다. 이 결과는 실제로 필터링을 사용하더라도 동작에 영향을 거의 영향을 받지 않는다는 것을 보여 주고 있다. 즉, 결함이 발생하여 이벤트를 전송하는 데까지 소요되는 시간이 필터링을 사용하는 경우와 그렇지 않은 경우가 거의 차이가 나지 않기 때문에 필터링을 사용하는 경우가 실제로 추후에 일어나는 처리를 생각하면 적절하다는 것을 보여 준다.

표 1은 모듈간 평균 응답시간을 보여준다. 여기서 응답시간이란 원격 모듈로 이벤트를 전송하고 전송에 대한 응답을 받기까지의 시간을 말한다. 네트워크 인터페이스는 Ethernet,

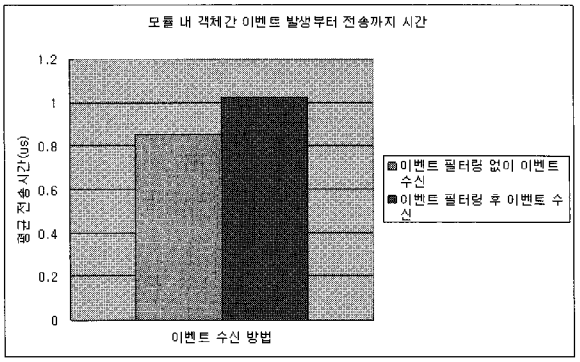


그림 10. 모듈 내 객체간 이벤트 전송 시간.

Fig. 10. Transmission time for event delivery between objects in module.

표 1. 모듈간 응답시간 (단위, ms).

Table 1. Response time between modules (unit, ms).

모듈	사용된 네트워크	평균 응답시간
Module1-Module2	Ethernet	0.345
Module1-Module3	IEEE1394	0.420
Module2-Module3	Ethernet, IEEE1394	0.759

IEEE1394를 사용하였으며 응답시간은 10,000번 반복하여 측정하였다. 네트워크 별 최악(worst) 응답시간은 IEEE1394의 경우 110ms가 Ethernet의 경우 20ms가 소모되었다. 최소 응답시간은 IEEE1394의 경우 0.259ms가 소모되었고, Ethernet의 경우 0.41ms가 소모되었다. 평균 응답시간은 최악 응답시간을 제외한 결과 IEEE1394의 경우 0.345ms가 소모되었고, Ethernet의 경우 0.420의 시간이 소모되었다. 각 네트워크의 분산은 IEEE1394의 경우 5.716이고, Ethernet의 경우 4.06이다. 평균적으로 1ms 이내에 사용되는 연성(soft) 실시간에 사용될 수 있다는 것을 보여 주고 있다. 이 결과에서 IEEE 1394가 좀더 시간이 많이 소요되는 이유는 아직도 드라이버가 최적화되지 않았기 때문으로 생각된다. Ethernet과 IEEE 1394를 모두 통과한 경우의 평균 응답시간은 0.759ms로 1ms 이내의 연성 실시간에 충분히 사용할 수 있다는 것을 알 수 있다.

런타임시 결함을 발생시켜 결함에 대한 대처 능력을 측정하였다. 발생시킨 결함의 종류는 객체의 오류와 네트워크 케이블의 제거이고, 객체 오류의 경우 100개의 객체를 생성하고, 특정 객체가 결함으로 인해 갑자기 소멸되었을 때의 복구되는 시간을 측정하였고, 네트워크의 경우 케이블을 분리하여 네트워크 연결이 갑자기 종료되었을 때 재연결 후 다시 정상적인 메시지 전송까지의 시간을 측정하였다. 이 경우들의 결함 발견부터 복구까지의 시간은 표 2와 같다. 여기서 객체의 복구는 해당 객체의 재생성이며, 네트워크 복구는 다른 네트워크로 연결하여 통신하는 것을 말한다. 그리고 이런 형태의 복구가 되지 않으면 해당 모듈의 정지 혹은 해당 데이터의 송수신 금지를 통하여 객체의 안전한 동작을 하게 하는 형태의 복구를 수행한다.

객체 소멸의 발견부터 복구까지의 시간은 윈도우 플랫폼에서 MFC에서 제공하는 OnTimer를 사용할 경우 55ms 보다 빠른 동작이 되지 않으므로 상태 전송의 주기를 100ms로 정하여 측정하였다. 결함 발견부터 복구까지의 시간은 1000번 반복하여 측정한 결과 최대 약 12ms, 최소 약 10ms의 시간이 걸렸다. 평균 복구시간은 약 11.1ms가 되고, 분산은 0.703이 되었다. 미들웨어가 스레드를 사용하고 여러 가지 서비스를 수행하기 때문에 약간의 차이가 발생했다. 네트워크 종료 발견부터 복구까지의 시간은 상태 전송 주기를 3초로 설정하여 측정하였다. 네트워크 연결 종료 후부터 재연결 및 데이터 전송까지의 과정은 네트워크 연결 종료를 확인하고, 경로 재설정(rerouting)을 수행한 후 서비스 발견(discovery)를 수행하여 서비스 호출 후 응답을 받을 때까지이다. 네트워크 종료는 네트워크 케이블을 인위적으로 분리한 후부터 다시 접속하기까지의 시간이며, 이 시간은 약 10초이다. 1000번 반복하여 측정한 결과 복구시간은 네트워크 분리 후 접속까지의 10초를 제외하고 최대 약 13ms, 최소 약 10.5ms의 시간이 걸리고, 평균 약 12ms의 시간이 소모되며 분산은 0.518이 되는 것을 확인하였다.

표 2. 결함 발견부터 복구까지 평균 소모 시간 (단위, ms).
Table 2. A time from fault detection to fault recover (unit, ms).

내용	사용된 네트워크	시간
갑작스런 객체 소멸	-	11.1
갑작스런 네트워크 종료	IEEE1394 Ethernet	12

표 2는 제안하는 결함허용 미들웨어 구조가 객체 소멸 혹은 네트워크 케이블의 절단과 같은 갑작스런 종료에 의한 결함 발생 시에도 최대 13ms 이내에 복구할 수 있다는 것을 보여 주고 있다.

V. 결론

본 논문에서는 서비스 로봇에서 결함 허용 서비스를 지원하기 위한 구조를 제안하고 구현하였다. 제안된 시스템은 결함 발견을 위한 다양한 객체 및 내부 시스템을 감시하기 위한 모니터(monitor), 모니터에 수집된 정보를 바탕으로 결함을 판단하는 결함 발견기(fault detector), 발견된 결함을 복구하기 위해 복구 요청을 수행하는 결함 통보기(fault notifier), 발생된 결함의 종류를 분석하여 결함을 시스템 독립적으로 복구시키는 결함 복구기(fault recover)의 4가지 컴포넌트로 구성되었다.

본 논문에서는 제안된 결함 허용 시스템을 구현하였으며, 모듈간 네트워크 인터페이스로는 IEEE1394, Ethernet을 사용하였다. 구현된 시스템은 프로시저 기반으로 구현된 [9]와 달리 컴포넌트 기반으로 구현하였으며, 결함 처리와 관련된 모든 개체를 컴포넌트화하였다. 따라서 결함을 처리하고 관리함에 있어 동일한 추상화된 방법을 사용할 수 있었다. 또한 컴포넌트 모니터링을 추가함으로써 단위 기능을 수행하는 컴포넌트 관리를 융통성 있게 하였고, 동적으로 컴포넌트를 추가/삭제함으로써 로봇이 동작하는 환경에 효율적으로 적응하게 하였다.

본 논문에서는 이종 네트워크 환경에서, 미들웨어를 이용한 응답시간을 측정하고 분석함으로써 제안된 시스템의 동작을 검증하였고, 로봇 미들웨어에 적재하여 제한된 자원을 가지는 로봇에 사용될 수 있음을 검증했다. 또한 결함 발생 시 결함 발생부터 복구까지 걸리는 시간을 측정하여 최대 13 ms 이내에 복구함으로써 제안된 결함허용 미들웨어가 로봇 시스템에 적용 가능함을 보여 주었다.

응용 개발자는 종류가 다른 여러 네트워크로 구성된 미들웨어를 사용함으로써 네트워크에 대한 고려 없이 응용들을 쉽게 개발할 수 있고, 응용에 부수적인 예외처리나 결함 처리의 기능을 포함시키지 않더라도 미들웨어에서 제공하는 결함 허용 서비스를 사용함으로써 응용의 에러나 결함에 대처할 수 있다. 제안된 시스템은 서비스 로봇과 같은 분산 시스템에 인터페이스를 제공함으로써 확장성과 재사용성을 증가시키고, 결함 허용성을 제공함으로써 로봇이 안정적으로 동작하는데 크게 기여할 것이라고 생각된다.

참고문헌

[1] 이승익, 서범수, 김중배, "지능형 서비스 로봇을 위한 유비쿼터스 환경과 제어구조," 전자통신동향분석 제 22 권 제 2 호, pp. 20-30, April, 2007.

[2] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Journal of Dependable and Secure Computing IEEE Transaction*, vol. 1, no. 1, pp. 11-33, Jan. 2004.

[3] S. C. Ahn, J. H. Kim, K. W. Lim, H. D. Ko, Y.-M. Kwon, and H.-G. Kim, "UPnP approach for robot middleware," *Proceeding of the 2005 IEEE International Conference on Robotics and Automation (ICRA2005)*, pp. 1959-1963, April 2005.

[4] Hans Utz, Stefan Sablatnog, Stefan Enderle, and Gerhard Kraetzschmar, "Miro-middleware for mobile robot applications," *IEEE Transaction on Robotics and Automation*, vol. 18, no. 4, August 2002.

[5] S. S. Hong, J. S. Lee, H. S. Eom, and G. I. Jeon, "The Robot Software Communication Architecture(RSCA): embedded middleware for networked service robots," *Proceeding of the IEEE International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2006.

[6] FT-CORBA specification. http://www.omg.org.

[7] K. H (Kane) Kim, "ROAFTS: a middleware architecture for real-time object-oriented adaptive fault tolerance support," *Proc. IEEE CS 1998 High-Assurance Systems Engineering (HASE) Symp.*, WashingTon, D.C., pp. 50-57, Nov 1998.

[8] D. Garlan, S.-W. Cheng, A.-C. Hyang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46-54, October 2004.

[9] G. Yun, H.-Y. Kim, and H. S. Park, "Middleware structure for module-based personal robot," *Journal of Control, Automation and Systems*, vol. 10, no. 5, pp. 464-474, May 2004.

[10] IEEE standard for a High Performance Serial Bus "IEEE std 1394-1995, IEEE 1394 std 1394a-2000".

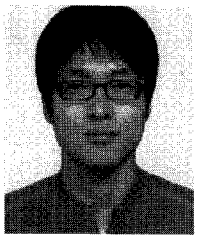
[11] Universal Serial Bus Specification revision 1.1: September 23. 1998.

[12] CAN specification Part A and Part B.

[13] http://www.terms.co.kr/fault-tolerant.html.

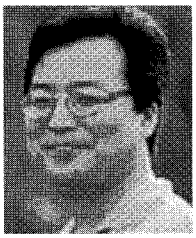
[14] Felber, P. Defago, X. Guerraoui, R, Oser, P, "Failure detector as first class objects," *Distributed Objects and Applications, 1999. Proceedings of the International Symposium*, pp. 132-141, 5-6 Sept. 1999.

[15] Lightweight Fault Tolerance for Distributed Real-Time Systems Request for Proposals. http://www.omg.org.



백 범 현

1980년 7월 13일생. 2006년 강원대학교 전자통신학과 졸업. 2006년~현재 동 대학원 전자통신공학과 석사과정 재학중. 관심분야는 분산 시스템, 유비쿼터스 컴퓨팅 등.



박 흥 성

1961년 3월 16일생. 1982년 서울대학교 제어계측공학과(공학사). 1986년 동 대학원(공학석사). 1992년 동 대학원(공학박사). 1992년~현재 강원대학교 전자통신공학과 교수. 관심분야는 로봇 SW 플랫폼, 실시간 네트워크 등.