IE Interfaces
Vol. 23, No. 1, pp. 24-34, March 2010.

IP 기반 융합서비스를 위한 서비스 충돌 감지 및 해결에 대한 연구

오요셉 · 신동민†

한양대학교 산업경영공학과

A Mechanism for Conflict Detection and Resolution for Service Interaction : Toward IP-based Network Services

Joseph Oh · Dongmin Shin

Department of Industrial and Management Engineering, Hanyang University

In the telecommunication system which is based on the existing PSTN(public switched telephone network), feature interaction has been an important research issue in order to provide seamless services to users. Recently, rapid proliferation of IP-based network and the various types of IP media supply services, the feature interaction from the perspective of application services has become a significant aspect. This paper presents conflict detection and resolution algorithms for designing and operating a variety of services that are provided through IP-based network. The algorithms use explicit service interactions to detect conflicts between a new service and registered services. They then apply various rules to reduce search space in resolving conflicts. The algorithms are applied to a wide range of realistic service provision scenarios to validate that it can detect conflicts between services and resolve in accordance with different rule sets. By applying the algorithms to various scenarios, it is observed that the proposed algorithms can be effectively used in operating an IP-based services network.

Keyword: conflict detection, conflict resolution, IP-based network, rule-based resolution, and service interaction

1. 서 론

최근 네트워크 기반 기술 및 정보응용 기술의 발달로 IP기반 통신망에서 융합 서비스를 제공하는 다양한 기기들이 보편화 되고 있다. 이에 따라 이들 기기들을 통해 제공되는 융합 서비스도 다양한 사용자의 요구에 부합하기 위하여 복잡하고 다양해지고 있다.

IP 기반 통신망을 사용하는 대표적인 기기인 IPTV는 최근 그 시장이 급속히 성장하고 있다. 이는 단순히 기기 보급이 확대된 것을 의미하는 것뿐만 아니라 컨텐츠나 부가 서비스를 제공하는 시장도 함께 성장함을 의미한다. 이처럼 IP 기반 통신망을 통해 서비스를 제공하는 시장이 급속히 성장해가는 시점에서 원활한 서비스 제공을 위한 관련 기술의 연구개발이 매우 중요하다. 본 연구에서는 IP 기반 통신망을 대표하는 기기로서 IPTV를 선정하여 이를 토대로 충돌상황을 고려하였고, 연구를 진행하였다.

이러한 IPTV서비스와 관련된 서비스 제공자의 증가와 융합적 형태를 띠는 서비스의 다양화는 서비스간 충돌(conflict)의 원인

†연락저자 : 신동민 교수, 426-791 경기도 안산시 상록구 사3동 1271 한양대학교 산업경영공학과, Fax : 031-409-2423, E-mail : dmshin@hanyang.ac.kr
투고일(2009년 11월 02일), 심사일(1차 : 2009년 11월 11일), 게재확정일(2010년 01월 17일).

이 되고 있다. 이러한 서비스 간 충돌은 다양한 형태로 발생할 수 있으며, 예상하지 못한 서비스 간의 상호작용(interaction)으로 인하여 서비스 사용자와 제공자의 의도에 부합하지 않는 상황들이 발생할 수 있다.

본 연구에서는 IP 기반 통신망 환경에서 발생할 수 있는 서비스간 충돌을 감지하고 해결할 수 있는 알고리즘을 제시한다. 이를 통하여 IP 기반 통신망 환경에서 새로운 융합 서비스의 개발 및 등록에 필요한 지침을 제공하고, 융합 서비스 시스템 운영을 효과적으로 지원할 수 있는 운영체계를 구축할 수 있다.

본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 본 연구와 관련된 기존연구들을 고찰하고, 제 3장에서는 본 연구의 대상이 되는 IP 기반 서비스 시스템에 대하여 설명한다. 제 4장에서는 서비스의 구성요소들을 제시하고, 제 5장에서는 본 연구에서 해결하고자 하는 서비스 충돌의 유형을 구분한다. 제 6장과 제 7장에서는 각각 서비스 설계시점과 운영시점에서 서비스 충돌을 해결하는 방법론과 알고리즘을 제시한다. 제 8장에서는 실제 시나리오를 바탕으로 서비스 충돌 감지 및 해결 알고리즘을 검증하고 제 9장에서는 결론과 향후 연구방향을 제시한다.

2. 관련 연구

2.1 오토마타 이론(Automata theory)

오토마타 이론은 이산사건 시스템을 유한한 상태의 정형적(formal)모형으로 표현하는 도구로 사용되어 왔다(Christos G. Cassandras, 2007). 이는 대상이 되는 시스템을 특정 상태(state)에서 외부 혹은 내부에서 발생하는 이벤트(event)에 따라 상태가 전이하는 객체로 표현하는 방식을 사용한다. 이로써 시스템 상태의 변화와 변화를 야기시키는 이벤트간의 관계를 명확하게 나타낼 수 있다는 장점으로 인해 다양한 분야에 널리 이용되고 있다. 특히 개념적인 시스템의 모델링이 용이할 뿐 아니라 수학적 전개와 분석을 가능하게 할 수 있어 통신 서비스 분야에 효과적으로 적용될 수 있다.

일반적인 이산사건 시스템은 상태 집합(state set), 이벤트 집합(event set), 상태 전이함수(state transition function), 초기상태(initial state), 수락상태 집합(final state set)로 구성된 automaton(G)로 표현될 수 있다. 식 (1)은 G를 다섯 개(5-tuples)의 구성요소로 표현하고, 각 요소에 대한 설명을 나타낸 것이다.

$$G=(\Sigma,\ Q,\ \delta,\ q_0,\ Q_m) \tag{1}$$

여기에서
Σ : G가 인식할 수 있는 이벤트들의 집합
Q : G가 가지는 상태들의 집합
δ : $\Sigma\times Q\rightarrow Q$: G의 상태전이함수
q_0 : G의 초기상태, $q_0\in Q$
Q_m : G의 수락상태 집합, $Q_m \subseteq Q$
를 의미한다.

예를 들어 $\delta(q,\ e)=q'$는 상태 q에 있는 시스템이 이벤트 e의 발생으로 인하여 새로운 상태 q'으로 상태전이가 발생했음을 의미한다. 본 연구에서는 IP기반 통신망 서비스라는 이산사건시스템을 오토마타 이론을 사용하여 정형적으로 표현한다. 이는 개별적인 서비스가 사용자의 요구와 통신망에서의 특정 이벤트가 진행됨에 따라 서비스의 진행상황을 파악하고 표현할 수 있도록 하여 서비스 간의 상호작용을 감지하는데 사용될 수 있다.

2.2 피쳐 상호작용(Feature Interaction)

전통적으로 피쳐(feature)라는 용어는 하나의 소프트웨어 애플리케인션에서의 기본적인 기능의 단위를 의미한다. 최근에는 하나의 통신망에서 다양한 융합서비스를 제공할 때 발생하는 피쳐간 상호작용(feature interaction)에 대한 연구의 필요성이 증대됨에 따라 다양한 관점에서의 연구가 진행되고 있다(Calder, Kolberg *et al*., 2003). 웹 서비스(web services) 분야의 피쳐간 상호작용에 관한 연구들이 이러한 전형적인 예가 될 수 있다(Weiss, Oreshkin *et al*., 2005; Weiss, Esfandiari *et al*., 2007).

본 연구의 대상이 되는 IP 기반 통신망에서의 융합 서비스 제공을 위한 피쳐간 상호작용과 밀접한 연구로는 Reiff 등의 연구를 들 수 있다(Reiff 2000; Reiff 2002; Calder, Kolberg *et al*., 2003). 이 연구는 서비스가 운영되는 시간 동안(run time)에 발생하는 복수 피쳐간 상호작용에 대한 충돌감지와 해결을 다루었다. 이때 메시지(message)와 트레이스(trace)라는 개념을 사용하여 피쳐가 진행되는 과정을 표현하였고, 규약(rule)을 통하여 충돌이 해결되는 방법론을 사용하였다. 그러나 이 연구에서 제시한 방법론은 그 대상 시스템이 기존의 전화망 시스템으로 국한되어있어 IP 기반 통신망에서의 다양한 융합 서비스의 제공을 위한 동적인 상황에는 적용하기 어렵다. 따라서 본 연구에서는 기존 방법론을 확장하여 IP 기반 통신망에 적용할 수 있도록 수정 및 보완하였으며, 시스템의 설계시점과 운영시점의 알고리즘으로 체계화하였다.

3. IP 기반 서비스 시스템 구조

본 연구의 대상이 되는 IP 기반 서비스 시스템은 <그림 1>과 같은 구조이며 구성요소에는 터미널 디바이스(terminal device), 미디어 서버(media server), 게이트웨이(gateway), 어플리케이션 서버(application server)가 있다.

IP 기반 서비스 시스템에서 서비스 사용자는 터미널 디바이스를 통하여 자신이 원하는 서비스를 제공받게 된다. 이때 사용자의 터미널 디바이스는 IP네트워크에 접속할 수 있는 모든

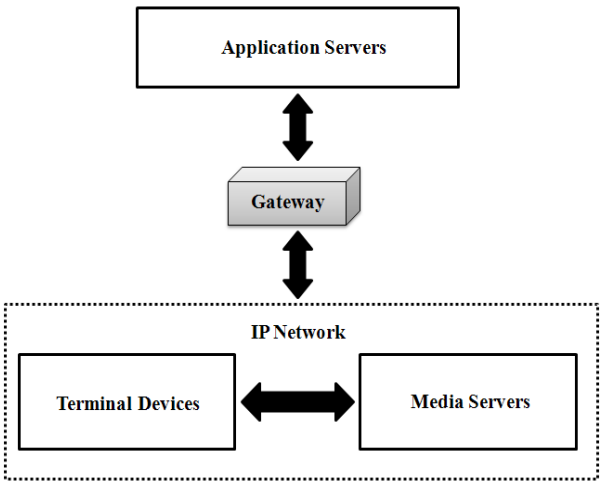

그림 1. IP 기반 서비스 시스템 구성도

기기를 의미하며 구체적으로 PDA, VoIP, IPTV 등이 있다. 또한 미디어 서버는 서비스 사용자에게 제공하기 위한 미디어 컨텐츠의 저장 서버이며, 미디어 컨텐츠의 구체적인 예로는 영화나 방송 VoD 등이 있다. 미디어 서버는 터미널 디바이스에 IP 네트워크를 통하여 미디어 컨텐츠를 제공한다. 이때 IP 기반 서비스 시스템에서는 사용자에게 미디어 컨텐츠 외에 부가적인 서비스를 제공할 수 있다. 부가적인 서비스는 어플리케이션 서버에 저장되어 있으며 이러한 서비스의 구체적인 예로 영화 컨텐츠 제공시 어학 공부를 위한 영어자막 서비스, 해당 컨텐츠에 대하여 가격을 할인해주는 서비스 등이 있다. IP 기반 서비스 시스템은 부가적인 서비스가 많아질수록 시스템 내에서의 서비스간 충돌 위험성이 증가하기 때문에 이를 제어하기 위하여 게이트웨이라는 요소를 필요로 한다. 게이트웨이는 등록된 부가 서비스들을 관리하고 IP네트워크를 모니터링 하여 서비스 간 충돌이 발생하지 않도록 제어한다.

본 연구에서는 게이트웨이에서 이루어지는 서비스간 충돌 해결 방법을 IP 기반 서비스 시스템의 설계시점(design time)과 운영시점(run time)으로 나누어 제시한다. IP 기반 서비스 시스템의 설계시점은 새로운 서비스가 기존의 시스템에 등록되는 시점이고 운영시점은 실제로 서비스가 진행되는 시점을 의미한다.

4. 서비스 구성요소

4.1 리소스

리소스(resource)는 서비스가 수행될 때 필요한 물리적, 논리적 개체를 의미한다. 서비스 수행 시 특정 리소스가 필요하다면 이를 점유해야만 서비스가 계속 진행될 수 있으며, 점유하지 못한다면 서비스가 대기상태에 빠지거나 충돌이 발생할 수 있다. 리소스는 여러 가지 형태로 분류될 수 있으며 이에 대한 예로는 서버(server) 리소스, 기기(device) 리소스, 네트워크(network) 리소스, 데이터베이스(database) 리소스 등이 있다. 리소스는 형태에 따라서 점유될 수 있는 횟수, 동시 점유 가능 여부 등의 특징을 가지므로 이러한 형태와 특징에 따라 리소스 분류(resource classification) 체계를 구성할 수 있다. 그리고 리소스 분류는 서비스 충돌을 감지할 수 있는 한 방법으로 사용될 수 있다. 예를 들어 동시 점유가 불가능한 특징을 가진 리소스에 대하여 두 가지 서비스가 동시에 점유하려 한다면 서비스 충돌이 발생하게 된다. 또한 한 번 밖에 점유될 수 없는 리소스에 대하여 한 서비스의 점유 후 다른 서비스가 점유하려는 경우에도 서비스 충돌이 발생하게 된다.

4.2 메시지

메시지(message)는 서비스의 상태를 변화시키는 서비스 수행의 최소단위이며, 구체적인 예로써 *기기리소스 점유요청 메시지, 기기리소스 점유해제 메시지* 등이 있다. 본 연구에서 메시지는 단순히 서비스의 상태를 변화시키는 이벤트(event)보다 좁은 의미로 사용되었다. 즉 시스템 내에서 발생할 수 있는 다양한 이벤트 중에서 서비스의 진행과 직접적으로 관련되어 있는 이벤트만을 표현한 것이다.

하나의 서비스는 이를 구성하는 일련의 메시지들을 하나씩 처리하면서 진행되며 최종의 목적에 도달하는 과정을 따르게 된다. 일련의 메시지로 구성되는 서비스의 상태 변화는 <그림 2>와 같이 표현할 수 있다.

<그림 2>에서 m_1, m_2, m_3, m_4는 서비스의 메시지를 의미하고, q_1, q_2, q_3, q_4는 서비스의 상태를 의미한다. 또한 <그림 2>에 표현된 서비스는 *로그인 영화컨텐츠 전송 서비스*로 상태 q_1(초기상태)에서 시작하여 상태 q_1, q_4(수락상태)에서 종료

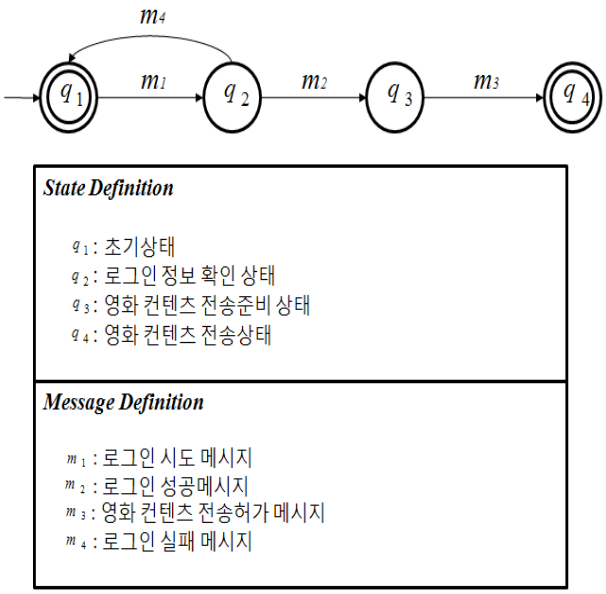

그림 2. 서비스 상의 메시지 처리 과정

된다. 서비스의 각 상태에서는 메시지를 통하여 다른 상태로 전이가 일어나게 되며 상태 q_2인 *로그인정보 확인상태*에서 로그인이 확인된다면 메시지 m_2로 인하여 상태 q_3로 전이하게 되고, 로그인 정보가 확인되지 않는다면 메시지 m_4로 인하여 상태 q_1으로 전이하게 된다.

4.3 트레이스

트레이스(trace)란 메시지가 순차적으로 결합한 형태로 서비스 과정을 표현하는 단위라고 할 수 있다. 서비스 과정은 수많은 트레이스 형태로 표현될 수 있으며, 이 중에서 서비스의 초기상태부터 최초의 수락상태까지 이르는 메시지의 연결 과정이 표현된 트레이스를 서비스 트레이스라 한다. <그림 2>와 같이 표현된 서비스에서 트레이스는 m_1, m_1m_2, $m_1m_2m_3$ 등이 될 수 있으며, 이 중에서 서비스 트레이스는 m_1m_4, $m_1m_2m_3$가 된다.

5. 서비스 충돌 유형

5.1 서비스 상호배타 충돌

서비스 상호배타 충돌은 두 서비스가 동시에 양립할 수 없는(mutually exclusive) 관계를 갖는 경우에 발생하는 충돌을 의미한다. 이러한 서비스 상호배타 충돌이 발생하는 경우, IP 기반 서비스 시스템에서 게이트웨이는 상호배타 관계에 있는 서비스들 중에서 단 하나의 서비스만이 진행되도록 제어하여 충돌을 해결한다.

서비스 상호배타 충돌의 예로는 다음과 같은 두 가지 정액과금 서비스의 경우를 들 수 있다. 만약 한 사용자가 IPTV시청 정액 서비스에 대하여 한 달 1만원인 것과 한 달 2만원인 것에 동시에 가입되어 있는 경우 상호배타적인 두 가지 정액과금 서비스 중 단 하나만으로 서비스가 진행되어야 할 것이다. 이러한 두 가지의 양립할 수 없는 정액과금 서비스가 동시에 진행되고자 하는 상황을 상호배타 충돌로 분류할 수 있다.

5.2 서비스 진행순서 충돌

IP 기반 서비스 시스템에서 게이트웨이는 서비스가 진행될 때, 해당 서비스의 메시지 처리 여부를 결정하는 역할을 한다. 다시 말해서 서비스는 메시지 처리에 앞서 게이트웨이로부터 메시지 처리여부의 허가를 받아야 한다. 만약 서비스가 메시지를 처리하려할 때, 게이트웨이가 메시지 처리를 허가해주지 않는다면 서비스는 해당 메시지부터 서비스가 진행되지 않는다. 게이트웨이는 메시지 처리 허가를 하나씩 결정하기 때문에 복수개의 서비스가 동시에 진행될 경우에는 메시지 허가 순서로 인해 서비스 간 충돌이 발생할 수 있다. 이렇게 복수개의 서비스들이 진행될 때, 서로 간에 영향을 미침으로써 특정 순서로 서비스가 진행될 경우 발생하는 충돌을 서비스 진행순서 충돌이라 한다.

서비스 진행순서 충돌의 예로 영화시청 서비스와 TV시청 서비스를 들 수 있다. 영화시청과 TV시청 서비스 두 가지 모두 IPTV라는 기기 리소스를 점유해야 서비스가 진행될 수 있다. TV시청 서비스가 영화시청 서비스보다 먼저 진행된다는 가정이 있는 경우, TV시청 서비스가 기기 리소스 점유를 해제한 후에 영화시청 서비스가 기기 리소스 점유 요청을 해야 한다. 만약 TV시청 서비스가 기기 리소스 점유를 해제하지 않은 상태에서 영화시청 서비스가 기기 리소스 점유를 요청한다면 충돌이 발생하게 되며 이를 서비스들 간 진행순서 충돌로 분류할 수 있다.

6. 설계시점 서비스 충돌해결 방법

설계시점은 서비스 생성자에 의하여 새로운 서비스가 IP 기반 서비스 시스템의 게이트웨이에 등록되는 시점이다. 이때 게이트웨이는 기존에 등록되어 있는 서비스들과 새롭게 등록되려는 서비스 간에 충돌이 발생하지 않도록 설계시점에 감지되는 충돌을 해결한다.

6.1 설계시점 서비스 충돌해결

서비스간 충돌이 발생하기 위해서는 복수개의 서비스들이 한 사용자에 대하여 동시에 진행 중이어야 한다. 만약 한 사용자가 두 가지 서비스에 가입한 경우, 특정 서비스가 종료된 후에 다른 서비스가 진행된다면 두 서비스는 서로 충돌이 발생하지 않을 것이다. 그러나 한 사용자가 가입한 두 서비스가 동시에 진행 중인 경우라면 서비스 간에 주고받는 영향으로 인하여 충돌이 발생할 위험을 내포하게 된다. 설계시점의 서비스 충돌해결은 서비스가 서로 간에 영향을 미치는 시점부터 충돌이 발생하지 않도록 서비스 진행 순서를 게이트웨이가 미리 정의된 순서대로 서비스 처리를 제어하는 방식이다.

(1) 서비스 상호작용

서비스 충돌은 서비스가 동시에 진행 중이어야 한다는 전제가 필요하며, 본 논문에서는 서비스의 동시 진행이 발생하는 시점을 서비스 상호작용이라 정의한다. 이는 다시 말해서 한

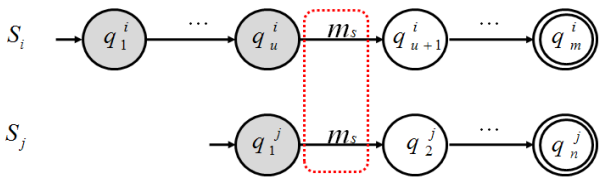

그림 3. 서비스 상호작용 예시

서비스의 진행 중에 특정 메시지로 인하여 다른 서비스가 시작되는 상황을 의미하며 <그림 3>에 표현되어 있다.

임의의 서비스 S_i는 자신의 서비스를 수행할 때, 서비스 진행과정을 구성하는 메시지를 하나씩 처리하게 된다. 이때 메시지 처리 과정에서 발생하는 특정 메시지 m_s가 다른 서비스 S_j의 첫 메시지가 되는 경우 서비스 상호작용이라 하며, 특정 메시지 m_s를 상호작용 메시지라 한다.

(2) 서비스 트레이스 솔루션 공간

서비스 상호작용이 발생한 서비스들은 상호작용 메시지 이후에 서비스를 진행함에 있어서 서로 간에 영향을 미치기 때문에 서비스 충돌의 위험을 내포하고 있다. 이러한 경우 게이트웨이는 서비스들 간의 동시진행에 있어서 충돌이 발생하지 않도록 메시지들의 처리 순서를 결정할 수 있다. 서비스 상호작용 이후로 메시지가 진행되는 방식은 두 서비스의 상호작용 메시지 이후 메시지들의 중첩 상호배치(overlapping interleaving)로 인해 나타난다. 만약 서비스 상호작용이 발생한 두 서비스에서 한 서비스의 상호작용 메시지 이후의 메시지 개수가 n개이고, 다른 서비스의 상호작용 메시지 이후의 메시지 개수가 m개일 경우, 중첩 상호배치로 인해 나타나는 트레이스의 개수는 식 (2)와 같다.

$$\frac{(m+n)!}{m!n!} \tag{2}$$

예를 들어 <그림 3>에서 서비스 상호작용이 발생한 서비스 (S_i)와 (S_j)의 경우, (S_i)은 상호작용 메시지 이후에 $m-(u+1)$개의 메시지가 있고, (S_j)은 상호작용 메시지 이후에 $n-2$개의 메시지가 있다. 따라서 중첩 상호배치로 인한 트레이스 개수는 식 (3)과 같다.

$$\frac{(m-u+n-3)!}{(m-u-1)!(n-2)!} \tag{3}$$

이는 (m_s)메시지 이후로 서비스가 진행될 수 있는 모든 경우의 수의 개수이며 이러한 경우들의 집합을 서비스 트레이스 솔루션 공간이라 한다. 서비스 트레이스 솔루션 공간은 서비스가 진행될 수 있는 모든 경우를 포함한 집합이기 때문에 서비스 충돌을 발생시키는 트레이스 또한 포함하고 있다.

(3) 충돌해결 서비스 트레이스 공간

서비스 트레이스 솔루션 공간은 서비스 상호작용으로 인해 새롭게 생성될 수 있는 모든 서비스 트레이스의 집합이며 충돌상황을 발생시킬 수 있는 트레이스 역시 포함하고 있다. 따라서 서비스 트레이스 솔루션 공간에 포함되어 있는 서비스 트레이스 원소 중에서 충돌상황을 유발시키는 서비스 트레이스를 제거할 필요가 있으며 이것이 제거된 집합을 충돌해결 서비스 트레이스 공간이라 한다.

이 과정에서 각 서비스의 최종적인 목적을 달성하기 위한 요구조건(requirement)을 반영할 수 있는 룰의 적용이 필요하다. 이러한 룰은 서비스 트레이스 솔루션 공간의 범위 내에서 서비스 충돌해결 서비스 트레이스 공간을 추출하는데 이용된다.

(4) 충돌해결 서비스 트레이스 탐색 룰

본 연구에서는 두 가지 종류의 룰을 수립하여 적용한다. 첫 번째의 룰은 서비스 트레이스를 구성하는 메시지간의 관계를 고려하는 룰(message dependent rule: MDR)이며, 이는 주어진 서비스가 최종 목적으로 하는 수락상태에 도달할 수 있도록 한다. MDR의 예로 "메시지 m_a 처리를 위해서는 반드시 메시지 m_b가 처리되어야만 한다", "메시지 m_a 이후에 메시지 m_b는 처리될 수 없다" 또는 "메시지 m_a와 메시지 m_b는 동일한 트레이스 내에서 처리될 수 없다" 등이 있을 수 있다.

두 번째로 적용되는 룰은 1차적으로 서비스 트레이스 솔루션 공간에 MDR을 적용하여 탐색된 충돌해결 서비스 트레이스 공간에 2차적으로 적용되는 룰이다. 이는 서비스의 보다 효율적인 제공을 도모할 수 있도록 고안된 것이며, 이러한 룰은 서비스 트레이스 내 메시지간의 관계에는 무관한 룰(message independent rule : MIR)이다. MDR이 서비스 충돌이 해결된 서비스 트레이스를 추출하기 위해 적용되는 반면, MIR은 서비스 제공자와 사용자의 만족도를 향상시키기 위한 목적으로 적용된다. MIR의 예로는 "서비스 트레이스 t^A보다 서비스 트레이스 t^B가 우선적으로 실행될 우선권을 부여한다" 또는 "가능한 많은 개수의 서비스와 관련된 메시지를 포함하고 있는 트레이스 우선권을 부여한다" 등이 있을 수 있다.

6.2 설계시점 서비스 충돌해결 알고리즘

<그림 4>는 설계시점에서의 서비스 충돌해결 알고리즘 적용과정을 나타낸 것이다. 이는 크게 세 가지 단계로 구분할 수

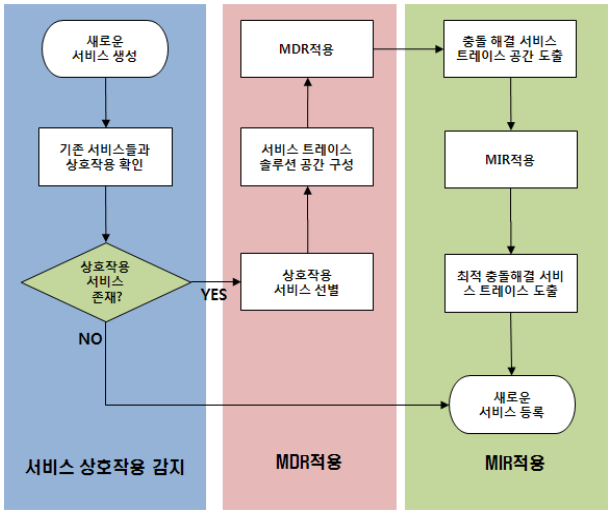

그림 4. 설계시점 서비스 충돌해결 알고리즘 적용

있다 : 1) 서비스 상호작용 감지 단계; 2) MDR 적용 단계; 3) MIR 적용 단계. 설계시점 알고리즘은 <부록>에 있는 <표 1>의 수도코드에 더 자세히 표현되어 있다.

(1) 서비스 상호작용 감지 단계

서비스 생성자에 의해 새롭게 생성된 서비스는 기존에 등록되어 있던 서비스와의 상호작용 여부를 판단 받는다. 이때 상호작용 여부는 기존에 등록되어 있던 모든 서비스에 대하여 판단된다.

만약 새롭게 생성된 서비스가 기존에 등록되어 있던 특정 서비스와 상호작용이 있다고 감지된다면 MDR적용 단계로 넘어가게 되고, 상호작용이 없다면 새로운 서비스를 IP기반 서비스 시스템의 게이트웨이에 바로 등록한다. 이는 새로운 서비스가 기존 서비스 집합의 원소로 추가되는 것을 의미한다.

(2) MDR 적용 단계

MDR 적용 단계에서는 먼저 기존에 등록되어 있는 서비스들 중에서 새로운 서비스와 상호작용이 있는 서비스들을 선별한다. 이 선별된 서비스들은 새로운 서비스와 상호작용으로 인해 서비스 충돌에 대한 위험을 내포하고 있으므로 이를 해결하기 위한 서비스 트레이스 솔루션 공간을 구축한다. 그리고 구축된 서비스 트레이스 솔루션 공간에 기존에 등록되어 있는 MDR을 적용하여 MDR에 위배되는 트레이스들을 제거한다.

(3) MIR 적용 단계

MDR 적용 단계에서 MDR을 적용함으로써 도출되는 트레이스들을 충돌해결 서비스 트레이스라 한다. MIR 적용 단계에서는 이 충돌해결 서비스 트레이스 공간에 MIR을 적용하여 최적 충돌해결 서비스 트레이스를 도출한다. 이는 서비스간 충돌이 해결된 충돌해결 서비스 트레이스 공간의 원소들 중에서 더 효율적인 트레이스를 도출하는 과정이다. 그리고 마지막으로 새로 등록되려는 서비스를 최적 충돌해결 서비스 트레이스와 함께 게이트웨이에 등록한다.

7. 운영시점 서비스 충돌해결 방법

운영시점은 서비스 사용자가 자신이 가입한 서비스를 운영하여 실제로 서비스 제공이 이루어지는 시점을 의미한다. 이때 IP 기반 서비스 시스템의 게이트웨이는 사용자가 가입한 서비스들의 운영에 있어서 충돌이 발생하지 않도록 메시지 하나하나의 처리에 대한 허용 여부를 결정하는 역할을 한다. 이때의 메시지 처리 허가의 판단기준은 설계시점에서 도출된 충돌해결 서비스 트레이스이다. 설계시점에서의 충돌해결은 운영시점에서 발생할 잠재적 충돌에 대한 위험을 제거하는 것이었다면, 운영시점에서의 충돌해결은 실제로 충돌이 발생한 후의 대처방안이다.

7.1 운영시점 서비스 충돌해결

운영시점에서 서비스 충돌해결 방법은 서비스 롤백이다. 이는 충돌 시점으로부터 가장 가까운 서비스 상호작용 시점으로 서비스를 되돌려 다른 방안으로 서비스가 진행될 수 있도록 하는 방법이다. 서비스 상호작용이 발생한 이후의 충돌 없이 진행될 수 있는 트레이스의 집합을 충돌해결 서비스 트레이스 공간이라 하며 이는 롤백 후 다른 방안으로 선택될 수 있는 트레이스의 대안 집합이라 할 수 있다. 만약 이러한 대안이 존재하지 않는다면 서비스 충돌로 인해 더 이상의 서비스 진행이 불가능함을 사용자에게 통보하고 서비스를 종료하여야 할 것이다.

서비스 롤백을 위하여 게이트웨이는 메시지 처리에 대한 허가를 판단함과 동시에 메시지가 상호작용 메시지인가에 대한 여부를 판단하여야한다. 처리 여부를 판단하려는 메시지가 상호작용 메시지인 경우, 게이트웨이는 롤백시점을 저장한다. 이후로 서비스를 진행함에 있어서 충돌이 발생한다면 가장 최근의 상호작용이 있었던 시점으로 되돌아가게 되는데 이는 전형적인 후입선출(last in first out)의 방식이다.

7.2 운영시점 서비스 충돌해결 알고리즘

<그림 5>는 운영시점에서의 서비스 충돌해결 알고리즘 적용과정을 나타낸 것이다. 이는 크게 세 가지 단계로 구분할 수 있다 : 1) 서비스 진행 단계; 2) 충돌해결 서비스 트레이스 갱신 단계; 3)대안 검색 및 롤백 단계. 운영시점 알고리즘은 <부록>에 있는 <표 1>의 수도코드에 더 자세히 표현되어 있다.

(1) 서비스 진행

서비스 사용자가 요청한 서비스는 요청된 메시지를 시작으로 진행되는 충돌해결 서비스 트레이스를 따라 진행된다. 서비스의 진행은 게이트웨이에서 메시지 처리에 대하여 하나하나 허가를 결정하는 과정도 포함한다. 게이트웨이에서 판단할

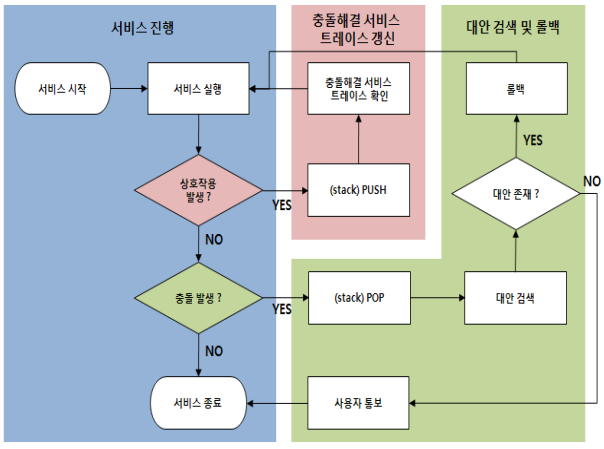

그림 5. 운영시점 서비스 충돌해결 알고리즘 적용

대상이 되는 메시지가 상호작용 메시지라면 충돌해결 서비스 트레이스 갱신 단계로 넘어가게 되고, 충돌을 감지하는 경우에는 대안 검색 및 롤백 단계로 넘어가게 된다. 이때 운영시점의 충돌 감지는 리소스 분류(resource classification)와 충돌해결 서비스 트레이스를 통하여 이루어진다.

(2) 충돌해결 서비스 트레이스 갱신

서비스 진행 단계에서 서비스 상호작용이 감지된다면 현재의 서비스 상태(state)를 저장하고, 현재의 충돌해결 서비스 트레이스에서 서비스 상호작용까지 고려한 새로운 충돌해결 서비스 트레이스를 도출하는 과정이다.

먼저 진행 중이던 충돌해결 서비스 트레이스 중 아직 처리되지 않은 메시지들로 구성된 트레이스와 서비스 상호작용을 야기한 서비스의 현 시점 이후 트레이스를 대상으로 새로운 서비스 솔루션 공간을 구성한다. 여기에 다시 MDR과 MIR을 적용하여 새로운 충돌해결 서비스 트레이스를 도출하여 서비스를 진행한다.

(3) 대안 검색 및 롤백

서비스 진행 단계에서 서비스 충돌이 발생한다면 현재 상태에서 직전 서비스 상호작용 상태로 롤백하기 위해 롤백 가능 여부를 확인한다. 롤백이 불가능한 경우는 이전에 단 한 차례도 서비스 상호작용이 없어서 롤백할 수 있는 상태가 존재하지 않는 경우 등이 있을 수 있다. 만약 롤백이 가능하다면 가장 최근의 서비스 상호작용이 있었던 상태로 롤백하여 충돌해결 서비스 트레이스 공간에 실행 가능한 대체 후보가 존재하는지 확인한다. 만약 실행 가능한 후보가 존재한다면 이를 새로운 충돌해결 서비스 트레이스로 선정하여 서비스를 진행한다. 만약 실행 가능한 대안 트레이스가 존재하지 않는다면 사용자에게 더 이상 서비스가 진행될 수 없음을 통보한다.

8. 시나리오 적용

본 연구에서 개발한 서비스 충돌해결 알고리즘을 검증하기 위하여 서비스 설계시점과 운영시점에 대하여 시뮬레이션을 실시하였다.

본 시나리오는 기존에 등록되어 있는 서비스(S_1)와 새로 등록하려는 서비스(S_2)를 가정한다. 첫 번째 서비스(S_1)는 서버 접속 영화시청 서비스로 서비스 사용자가 영화 컨텐츠가 존재하는 서버에 직접 접속하여 영화를 시청하는 서비스이고, 두 번째 서비스(S_2)는 실시간 다운로드 영화시청 서비스로 서비스 사용자가 영화 컨텐츠를 자신의 기기에 다운로드함과 동시에 실시간으로 영화를 시청하는 것이다. <그림 6>은 서비스 S_1과 S_2를 표현한 것이다.

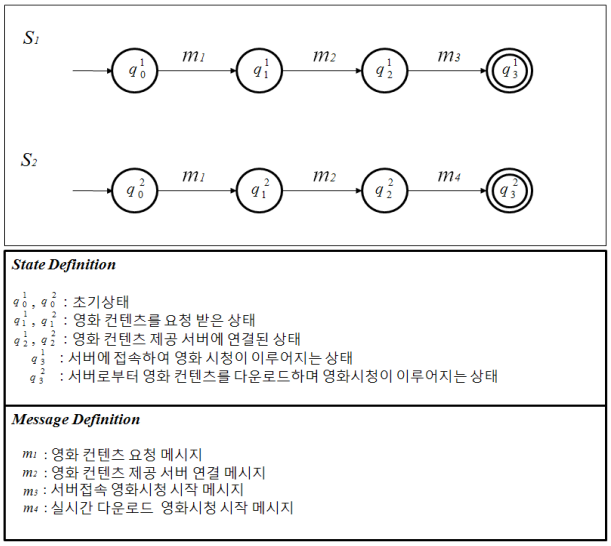

그림 6. 시나리오 적용 서비스

8.1 설계시점 시나리오 적용

IP기반 서비스 시스템에 서비스 S_1이 이미 등록되어 있고 설계시점에서 서비스 S_2가 등록되려는 상황이다. 이때 서비스 S_1과 S_2간에 서비스 상호작용이 메시지 m_1과 m_2에서 발생한다. 따라서 상호작용메시지 이후의 서비스 트레이스의 집합은 식 (4)와 같다.

$$\{m_3, m_4, m_3m_4, \ m_4m_3\} \tag{4}$$

식 (4)에서 메시지 m_3과 m_4는 각각의 서비스대로 서비스 트레이스가 진행되는 것을 의미하고, m_3m_4와 m_4m_3는 두 서비스가 동시에 진행되는 것을 의미한다. 메시지 m_3와 m_4는 모두 영화를 시청하게 하는 메시지로 TV라는 기기 리소스를 점유하여야만 서비스가 진행될 수 있기 때문에 두 메시지는 서로 상호배타적 메시지라 할 수 있다. 따라서 두 서비스가 동시에 진행되는 서비스 트레이스인 m_3m_4와 m_4m_3는 충돌 상황을 발생시키는 것이다. 결과적으로 이 두 가지 트레이스를 제거한 충돌해결 서비스 트레이스 공간은 식 (5)와 같다.

$$\{m_3, \ m_4\} \tag{5}$$

따라서 서비스가 충돌 없이 진행되기 위해서는 식 (5)의 한 원소로 서비스가 진행되어야 한다. 식 (5)에서 더 이상의 서비스 충돌이 없다는 가정 하에 서비스 S_2를 IP 기반 서비스 시스템의 게이트웨이에 등록할 수 있다.

8.2 운영시점 시나리오 적용

서비스 사용자가 서비스 S_1과 S_2에 가입한 상태이고 서비스 S_2에 더 우선순위를 주어 서비스를 진행한다고 가정한다. 메

시지 m_1과 m_2에서 서비스 S_1과 S_2의 서비스 상호작용이 발생하였기 때문에 이때의 상태를 저장하고 이후의 메시지는 충돌해결 서비스 트레이스 공간의 원소 중의 하나로 서비스를 진행한다. 서비스 사용자는 서비스 S_2에 더 우선순위를 주었다고 가정하였기 때문에 서비스는 메시지 m_4로 진행하게 된다. 이때 운영시점의 충돌은 사용자 기기의 다운로드 메모리 용량이 초과된 상황을 가정한다. 사용자 기기의 다운로드 메모리 용량이 초과되었으므로 더 이상 다운로드를 하며 실시간으로 영화를 볼 수 없는 운영시점 서비스 충돌상황이 되었다. 따라서 운영시점의 충돌을 해결하기 위하여 가장 최근의 서비스 상호작용 시점으로 돌아가 충돌해결 서비스 트레이스 공간에서 다른 대안 트레이스를 검색한다. 이 경우 우선순위는 트레이스 m_4보다 낮지만 m_3라는 다른 트레이스가 존재한다. 이는 서버에 직접 접속하여 영화를 시청하는 서비스이다. 따라서 게이트웨이는 충돌상황을 해결하기 위하여 현재 상태를 롤백하고 사용자에게 다른 트레이스로 서비스를 제공한다.

9. 결 론

본 연구는 IP 기반 통신망에서의 서비스 간 충돌감지 및 해결에 대한 방안을 제시하고 있으며, 이에 따른 연구의 결과물로 서비스간 충돌해결 알고리즘을 제시하였다. 서비스간 충돌해결 알고리즘은 시스템의 설계시점과 운영시점 시점으로 나뉘어 적용된다.

설계시점의 서비스간 충돌해결 알고리즘은 서비스 상호작용을 감지하고, 이에 대한 룰을 생성하는 역할을 하였다. 운영시점의 서비스간 충돌해결 알고리즘은 서비스를 서비스 트레이스 대로 진행할 수 없는 상황이 발생했을 때 해결방안을 제시하는 역할을 하였다.

본 연구는 주로 기존 전화망을 기반으로 다루어졌던 서비스 상호작용과 관련된 서비스 충돌해결 연구를 최근 급속히 성장하고 있는 IP 기반 통신망에 적용하였다는 점에서 큰 의미를 갖는다. 또한 연구의 결과로 제안하는 서비스간 충돌해결 알고리즘은 각각의 서비스 제공자와 서비스 가입자들의 프로파일 데이터베이스 정보를 관리하고, 시스템 내 메시지의 흐름을 제어할 수 있는 서버가 존재하는 통신망 시스템에 적용이 가능하다. 특히 향후 서비스 수가 급속히 증가할 것으로 예측되는 IP 기반 통신망 시스템이 이러한 방식에 의해 구축됨을 고려하면 본 연구가 제시하는 충돌해결방법이 새로운 서비스의 추가와 운영에 중요한 역할을 할 것으로 기대된다.

본 연구는 IP 기반 통신망 시스템의 서비스간 충돌문제를 해결하여 시스템의 안정성을 확보할 수 있다. 이는 국내 IP 기반 통신망 시장과 이와 관련된 서비스 산업을 더욱 성장시키는데 긍정적인 영향을 미칠 것이다. 또한 본 연구가 IP 기반 통신망의 서비스간 충돌을 해결하는 기술의 구체적인 적용 사례로써 향후 연구의 전개방향을 주도하는데 기여할 것으로 기대된다.

참고문헌

Calder, M., M. Kolberg, *et al*. (2003), Hybrid solutions to the feature interaction problem, *Feature Interactions in Telecommunications and Software Systems VII*.

Calder, M., M. Kolberg, *et al*. (2003), Feature interaction：a critical review and considered forecast, *Computer Networks,* 41(1), 115-141.

Christos G. Cassandras, S. L. (2007), *Introduction to Discrete Event Systems*, Springer.

Reiff, S. (2002), Runtime resolution of feature interactions in evolving telecommunications systems, University of Glasgow, UK.

Reiff, S. (2000), Identifying resolution choices for an online feature manager, *Feature Interactions in Telecommunications and Software Systems VI*.

Weiss, M., B. Esfandiari, *et al*. (2007), Towards a classification of web service feature interactions, *Computer Networks* 51(2), 359-381.

Weiss, M., A. Oreshkin, *et al*. (2005), Invocation Order Matters：Functional Feature Interactions of Web Services, *Engineering Service Compositions (WESC 2005)*, 69-76.

<부 록>

<표 1>과 <표 2>는 각각 <그림 4>의 설계시점 서비스 충돌해결 알고리즘과 <그림 5>의 운영시점 서비스 충돌해결 알고리즘을 수도코드로 나타낸 것이다. 운영시점의 수도코드 <표 2>는 <그림 5>의 알고리즘과는 구성이 다르게 되어 있다. <그림 5>의 경우 서비스 시작부터 종료 상황까지를 한 알고리즘으로 표현하였지만, 운영시점의 수도코드는 서비스 진행 중 한 메시지에 대한 처리를 게이트웨이에서 어떻게 처리할 지에 대한 알고리즘이다. 다시 말해 운영시점의 수도코드는 운영시점 알고리즘인 <그림 5>의 재귀적인 한 부분으로 볼 수 있다.

표 1. 서비스 충돌해결 설계시점 수도코드

A service S is a set of traces. A trace T is a sequence of messages.
Input : A new service $S_{new} = \{T^1, T^2, T^3, \cdots, T^m\}$, an arbitrary trace of service $T = m_1 \bullet m_2 \bullet m_3 \bullet \ldots \bullet m_N$
Output : A new resolution policy set RP_{new}, where $RP_{new} := \{(m, R_{new}) \mid$ interaction message m, new resolution $R_{new}\}$

Algorithm *design_time_conflict_resolution*(**in**(S_{new}), **out**(RP_{new}))
1 : $RP_{new} := \{\}$;
2 : **for** each $T_{new} \in S_{new}$ **do**
3 : **for** each $S_{old} \in \Theta$ **do** *//A set of already registered service* Θ
4 : **for** each $T_{old} \in S_{old}$ **do**
5 : **for** each m that is found from first message to last message in T_{new} **do**
6 : **if** m = message of T_{old} **then**
7 : $piece\_T_{new}$ = A sequence of messages after m in T_{new} ;
8 : $piece\_T_{old}$ = A sequence of messages after m in T_{old} ;
9 : $RP_{new} = \{ruleApplication(m, piece\_T_{old}, piece\_T_{new})\} \cup RP_{new}$;
10 : **end if**
11 : **end for**
12 : **for** each m that was found from first message to last message in T_{old} **do**
13 : **if** m = message of T_{new} **then**
14 : $piece\_T_{old}$ = A sequence of messages after m in T_{old} ;
15 : $piece\_T_{new}$ = A sequence of messages after m in T_{new} ;
16 : $RP_{new} = \{ruleApplication(m, piece\_T_{new}, piece\_T_{old})\} \cup RP_{new}$;
17 : **end if**
18 : **end for**
19 : **end for**
20 : **end for**
21 : **end for**
22 : **return** RP_{new} ;

Function *ruleApplication*(m, $piece\_T_x$, $piece\_T_y$)
23 : $SS := \{t \mid t$ is interleaving with $piece\_T_x$ and $piece\_T_y\}$; *//solution space SS*
24 : $RS := \{\}$; *//resolution space RS*
25 : **for** each $t \in SS$ **do**
26 : **if** t does not violate *MDR* **then** $RS = \{t\} \cup RS$; *//MDR is a set of message dependent rules*
27 : **end for**
28 : **for** each $t \in RS$ **do**
29 : **if** t meets *MIR* **then** $R = \{t\} \cup R$; *//MIR is a set of message independent rules*
30 : **end for**
31 : $R_{new} = R$;
32 : **return** (m, R_{new}) ;

표 2. 서비스 충돌해결 운영시점 수도코드

Input : An input message m_{input}, are solution R for a subscriber
Output : A next process policy of service PP_{next}, where PP_{next} := ($P_{boolean}$, $M_{boolean}$, R) | $P_{boolean}$ determines whether a service is subsequently processed or stopped, $M_{boolean}$ indicates message process approval or not, a resolution R for a subscriber)

Algorithm *run_time_conflict_resolution*(**in**(m_{input}, R), **out**(PP_{next}))
1 : $P_{boolean}$:= *true*; *//if $P_{boolean}$ is true then a service is subsequently processed*
2 : $M_{boolean}$:= *true*; *//if $M_{boolean}$ is true then message process approved*
3 : **if** m_{input} is the first message of a trace && all services are stopped **then** *//initialize when service begins*
4 : S_{sub} = {a subscriber's services};
5 : S_{m_input} = {services that the first message is m_{input}};
6 : $S_{sub,m\_input} = S_{sub} \cap S_{m\_input}$
7 : **if** the number of services in S_{sub,m_input} = 1 **then**
8 : $R = S_{sub,m\_input}$
9 : **else if** the number of services in S_{sub,m_input} > 1 **then** *//m_{input} is an interaction message*
10 : R=resolution that the first message is m_{input}
11 : **end if**
12 : **end if**
13 : **for** each $RT \in R$ **do** *//if a resolution is different with current process then remove the resolution*
14 : **if** $m_{input} \neq$ first message in RT **then** $R = R - \{RT\}$
15 : **end for**
16 : **if** $R = \emptyset$ **then**
17 : $M_{boolean}$ = *false* ;
18 : **else**
19 : **if** *conflictDetection*(m_{input}) = *true* **then**
20 : $state_{pop}$ = pop at stack ;
21 : **if** *RS* presence at $state_{pop}$ **then**
22 : rollback from current state to $state_{pop}$;
23 : **else**
24 : $P_{boolean}$ = *false* ;
25 : **end if**
26 : **else**
27 : **if** m_{input} is an interaction message **then** *//if m_{input} is an interaction message then update R*
28 : **for** each $RT \in R$ **do**
29 : current state pushes to stack ;
30 : R_{m_input} = it is are solution that the first message is m_{input} *//there solution is subscribed by subscriber*
31 : **for** each $RT_{m\_input} \in R_{m\_input}$ **do**
32 : $piece\_T_x$ = A sequence of messages after m_{input} in RT_{m_input} ;
33 : $piece\_T_y$ = A sequence of messages after m_{input} in RT ;
34 : SS = {t | t *is interleaving with* $piece\_T_x$ *and* $piece\_T_y$} ;
35 : $RS = \{\}$;
36 : **for** each $t \in SS$ **do**
37 : **if** t does not violate *MDR* **then** $RS = \{t\} \cup RS$;
38 : **end for**
39 : **for** each $t \in RS$ **do**
40 : **if** t meets *MIR* **then** $R = \{m_{input} \bullet t\} \cup R$;
41 : **end for**
42 : **end for**
43 : $R = R - \{RT\}$;
44 : **end for**
45 : **else**
46 : **for** each $RT \in R$ **do** *//m_{input} is processed*
47 : **if** $RT = m_{input} \bullet RT_{remain}$ **then** $RT = RT_{remain}$
48 : **end for**
49 : **end if**
50 : **end if**
51 : **end if**
52 : **return** ($P_{boolean}$, $M_{boolean}$, R) ;

Function *conflictDetection*(m_{input})
53 : $Conflict_{boolean}$:= *true*;
54 : *Res* := a set of all resources that m_{input} occupies ;
55 : *ResClass* := a set of all resource classes ;
56 : **for** each $r \in Res$ **do**
57 : **for** each $C \in ResClass$ **do**
58 : **if** r is included in C **then**
59 : **if** r is available for C **then**
60 : $Conflict_{boolean}$ = *false* ;
61 : **end if**
62 : **end if**
63 : **end for**
63 : **end for**
64 : **return** $Conflict_{Boolean}$

오 요 셉
한양대학교 정보경영공학과 학사
현재 : 한양대학교 산업경영공학과 석사과정
관심분야 : 서비스 상호작용, Distributed Computing, Device Coordination

신 동 민
한양대학교 산업공학과 학사
포항공과대학교 산업공학과 석사
펜실베니아 주립대 산업공학과 박사
현재 : 한양대학교 산업경영공학과 교수
관심분야 : Discrete Event Systems, Human-Machine Interaction, Information Systems