Journal of the Society of Korea Industrial and Systems Engineering
Vol, 31, No. 3, pp.43－51, September 2008.

# 유비쿼터스 환경에서 커뮤니티 컴퓨팅 지원을 위한 코디네이터 개발

남진규* · 김현우* · 신동민** · 박재일*** · 허 선**†

*한양대학교 산업공학과
**한양대학교 정보경영공학과
**아주대학교 산업정보시스템공학부

# A Formal Model of Coordination for Supporting Community Computing in a Ubiquitous Environment

Jingyu Nam* · Hyunwoo Kim* · Dongmin Shin** · Jaeil Park*** · Sun Hur**†

*Department of Industrial Engineering, Hanyang University
**Department of Information and Industrial Engineering, Hanyang University
**Division of Industrial and Information Systems, Ajou University

Recent advances in mobile computing technologies and platform-independent information systems have enabled to realize a ubiquitous environment. Community computing has been developed as a useful tool for realizing collaborative services in a ubiquitous environment. In this paper, we present a formal model of a ubiquitous space that takes community concept into consideration and propose two management frameworks that prevent conflicts among communities. To demonstrate the validity of the proposed frameworks, an example for coordinating two communities is provided.

Keywords : Community Computing, Finite State Automata, Supervisory Control Theory

## 1. 서 론

정보 기술 및 하드웨어 지원 시스템의 발전으로 컴퓨팅 기기의 성능은 급속하게 향상되는 동시에 기기의 크기는 일반 사용자가 용이하게 휴대할 수 있을 정도로 소형화됨으로써 다양한 사용자와 다양한 기기들로 구성된 유비쿼터스 컴퓨팅 환경이 점차 실현되고 있다. 유비쿼터스 환경에서는 사용자가 원하는 서비스가 하나의 기기에 의해 제공되는 것이 아니라 다양한 기기들에 존재하는 서비스들의 조합으로 제공되며, 이러한 환경은 기존의 분산 시스템이나 모바일 컴퓨팅 환경보다 더 복잡한 특징을 지니고 있다. 특히, 주변에 존재하는 여러 기기를 이용하여 사용자에게 다양한 서비스가 제공 될 수 있게 하기 위해 사용자간 및 기기간의 협업을 통한 복합 서비스의 제공 및 서비스 제공을 통한 상호 작용에 대한 관심이 증가하고 있다.

커뮤니티 컴퓨팅은 이러한 유비쿼터스 환경을 실현하기 위해 제안된 패러다임으로, 커뮤니티는 시스템 설계자 또는 개발자에 의해 생성(creation)된 후, 커뮤니티 구성원을 조직하여(organization) 이벤트에 의해 활성화 되

논문접수일 : 2008년 05월 22일 논문수정일 : 2008년 07월 22일 게재확정일 2008년 07월 29일
† 교신저자 hursun@hanyang.ac.kr
※ 본 논문은 인하대학교 정석물류통상연구원의 지원에 의하여 연구되었음(INHA-JRI-2007).

고(execution) 커뮤니티의 목적이 성취되면 비활성화(deactivation) 되거나 소멸(extermination)되는 특징을 지닌다. 유비쿼터스 컴퓨팅 공간에는 여러 개의 커뮤니티가 동시에 형성이 될 수 있고 구성원 및 지원 기기 역시 여러 커뮤니티에 동시에 관여할 수 있으므로, 커뮤니티에 관여하는 참가자 및 지원 기기들의 상호 작용에 대한 제어가 필요하다.

다양한 분야에서 커뮤니티 컴퓨팅과 관련된 사용자간 지원 시스템, 자원의 효율적 공유 및 할당, 사용자간의 사회적 연계(social networking)에 대한 모델 수립과 운영 시스템의 프로토타입 개발 및 운영에 관한 연구가 진행중이나, 많은 부분의 연구가 기술적(technical) 접근과 인지적(social cognitive) 접근에 초점을 두고 있다. 그러나 본 연구에서는 커뮤니티 컴퓨팅을 이용하여 효과적이고 근본적인 유비쿼터스 환경을 실현하기 위해 대상 시스템에 대한 정형적 이론(formal theory)을 기반으로 접근한다. 특히, 정형적 방법론은 구축된 모델의 이해도를 높이는 동시에 시스템의 모의 운영(simulation)을 통해 보다 더 체계적인 분석을 가능하게 하며, 유비쿼터스 환경을 실현하는데 보다 깊이 있는 고찰을 할 수 있다.

커뮤니티는 목적(goal)과 구성원의 역할(role), 구성원이 역할을 수행하기 위한 액션(action), 그리고 구성원을 선택하고 구성원의 액션을 제어하는 규칙(policy)으로 이루어진다. 유비쿼터스 공간에 존재하던 다양한 개체(entity)는 커뮤니티가 조직되는 단계에서 역할-구성원 바인딩(role-participant binding)을 통해 커뮤니티의 구성원이 되는데, 각 개체는 여러 커뮤니티의 구성원이 될 수 있으므로 다양한 역할을 부여 받을 수 있다. 그러나 부여받은 역할은 각 커뮤니티의 목적을 달성하기 위한 것이므로, 한 개체가 실행하는 서로 다른 커뮤니티의 역할들이 서로 충돌을 발생 시킬 수 있는데, 예를 들어 프린터가 "사용자가 검색한 내용을 요약하여 인쇄하는 것"을 목적으로 하는 커뮤니티에서 'print a document'라는 역할로 인해 '인쇄'라는 액션을 수행하는 중에 "자동으로 보안 업데이트"를 목적으로 하는 다른 커뮤니티 요청으로 'reboot' 하는 액션을 수행한다면 프린터는 인쇄중이던 내용을 메모리에서 삭제하게 되므로 다시 프린터가 켜지더라도 사용자가 원하는 요약본의 인쇄를 완료할 수 없다.

커뮤니티의 목적은 한 명의 사용자를 위한 것일 수도 있고 다수의 사용자를 위한 것일 수 있다. 이 경우 커뮤니티의 사용자는 선호도(user preference)를 가질 수 있는데, 여러 사용자를 위해 생성된 커뮤니티의 경우 사용자들마다 선호도가 서로 다를 수 있으므로, 서로 다른 선호도는 커뮤니티의 구성원이 수행하는 액션간의 충돌을 발생시킬 수 있다. 그러므로 커뮤니티의 상위에서 각 구성원의 액션을 제어할 수 있는 관리 프레임이 필요하며 이것은 커뮤니티 운영이 지속됨에 따라 점진적으로 진화되어 커뮤니티의 생성 소멸의 사이클에 따라 보다 효율적으로 커뮤니티 컴퓨팅 시스템을 지원할 수 있어야 한다.

본 연구에서는 여러 다양한 커뮤니티의 목적들이 생성되고 성취될 때 목적들 간에 발생할 수 있는 충돌 상황을 예방하는 코디네이터와 커뮤니티 내에서 사용자의 선호도에 따라 구성원의 액션들 간에 발생할 수 있는 충돌현상을 예방하는 코디네이터를 제안한다. 그래서 본 연구는 제안된 코디네이터가 충돌 방지에서 중요한 논-블록킹(non-blocking) 특성을 가진다는 것을 수리적으로 증명한다. 따라서 본 코디네이터의 적용을 통해 다양한 커뮤니티의 목적과 커뮤니티에 관여하는 구성원들의 액션들 간에 충돌을 예방할 수 있는 시스템의 구축이 가능하다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련연구에 대한 소개를 하고, 제 3장에서는 사용자의 요구에 따라 동적으로 커뮤니티를 형성하는 유비쿼터스 컴퓨팅 공간을 정형적 기법으로 표현하고, 유한상태기계(Finite State Automata, FSA)를 이용하여 커뮤니티의 목적과 구성원이 수행하는 액션을 정의하고, 각각을 관리하는 메커니즘을 제안한다. 제 4장에서 커뮤니티 컴퓨팅 예를 제시하여 본 논문에서 제안된 코디네이터의 역할을 보이며, 제 5장에서 결론과 추후 연구 과제를 제시한다.

## 2. 관련 연구

유비쿼터스 컴퓨팅 환경에서 다양한 서비스들이 복잡한 형태의 협업을 통해 제공되고 있다[11]. 이러한 협업적(collaborative)이고 사용자의 요구에 적합한 적응적(adaptive)인 서비스들을 보다 자연스럽고 직관적으로 개발할 수 있도록 하는 커뮤니티 컴퓨팅 모델이 제시되었다[1, 12]. 커뮤니티 컴퓨팅은 다수의 협력하는 개체들이 커뮤니티를 형성하여 하나의 서비스를 제공하는 모델로 이를 구체적으로 실체화하는 두 가지 방법론이 진행되고 있다. 하나는 model-driven architecture에 근거한 커뮤니티 시스템 개발 방법론[7, 8]이고 다른 하나는 기존의 서비스를 커뮤니티 구성원으로 참여 시키는 방안이다[9].

이러한 방법론과 더불어 유비쿼터스 환경에서 다양한 기기들의 코디네이션에 관한 연구가 활발히 진행 중이며, 특히 기기간의 충돌은 사용자 중심으로 운영되는 시스템에 중요한 영향을 미치는 요소로 많은 연구가 진행 중이다. 현재 충돌 감지에 관한 해결책들은 대부분 정적인 컴파일 환경에만 집중되어 있다[10]. 그러나 점

차 유비쿼터스 환경으로 진화함에 따라 운영환경에서의 동적인 충돌 감지와 해결방안에 대한 필요성이 증가하고 있다.

유한상태기계는 이산 사건 시스템을 제어하고 분석하기 위한 정성적인 모형을 개발하는데 널리 사용되고 있다[4, 6]. 특히, 유한상태기계의 다양한 오퍼레이션들은 협업적인 코디네이션과 제어에 관한 많은 연구들의 기저가 되어왔다[3]. 특히, 충돌 감지, 충돌 해법과 예방은 생산 시스템, 컴퓨터 기반 협업 작업 그리고 통신과 같은 분야에서 활발히 연구가 진행되고 있다[2, 5, 13].

커뮤니티 컴퓨팅과 관련한 기존연구에서는 대부분 실제 구현에 있어 필요한 커뮤니티 기술언어(description language)나 온톨로지 기반의 메타서비스 구현 등에 치중되어 있는 반면 커뮤니티 컴퓨팅 환경에서 발생할 수 있는 충돌현상에 대한 연구는 미비한 편이다. 본 연구에서는 유한상태기계와 제어이론(supervisory control theory)을 이용하여 커뮤니티들 사이에 발생할 수 있는 충돌현상을 예방하기 위한 방법론을 제안하고, 실제 커뮤니티 컴퓨팅에서 어떻게 적용되는지를 설명한다.

## 3. 커뮤니티 컴퓨팅 기반 유비쿼터스 환경

### 3.1 커뮤니티 기반의 유비쿼터스 환경

커뮤니티를 기반으로 한 유비쿼터스 공간에서 사용자의 요구에 적합한 서비스는 하나의 기기를 통해 제공될 뿐 아니라 여러 다양한 기기들의 협업을 통해 제공된다. 즉, 유비쿼터스 컴퓨팅 공간은 사용자가 원하는 서비스를 관련되는 각종 이벤트를 통해 감지하고 그에 적합한 서비스를 제공할 수 있는 협업 커뮤니티를 동적으로 구성한다. 본 절에서는 사용자의 요구에 의해 발생되는 이벤트에 따라 동적으로 커뮤니티를 생성하는 유비쿼터스 컴퓨팅 공간을 정의한다.

커뮤니티는 유비쿼터스 컴퓨팅 공간에 의해 구성되는데, 먼저 감지된 이벤트를 이용하여 사용자가 원하는 서비스에 적합한 목적과 그 목적 달성에 필요한 역할들을 생성한다. 이렇게 생성된 역할들은 유비쿼터스 공간에 존재하는 적합한 개체에 바인딩되며, 그 이후 각 역할을 부여 받은 개체들은 한 커뮤니티의 구성원이 되고 그에 적합한 액션을 수행하게 된다. 이러한 커뮤니티 기반의 유비쿼터스 환경을 다음과 같이 정의한다.

$\mathbb{S} = \langle \Sigma, G, E, R, M, A, \theta, \chi, \lambda, \beta, \sigma, \eta \rangle$

$\Sigma$ : 이벤트의 집합
$G$ : 목적 집합
$E$ : 구성원 선별을 위한 조건 집합
$R$ : 역할 집합
$M$ : 구성원 집합
$A$ : 액션 집합
$\theta : \Sigma \rightarrow G$ : 목적 생성 함수(goal generating function)
$\chi : G \rightarrow E$ : 조건 생성 함수(constraints generating function)
$\lambda : G \rightarrow R$ : 역할 생성 함수(role generating function)
$\beta : C \rightarrow M$ : 구성원 선별 함수(member generating function)
$\sigma : R \rightarrow R \times M$ : 역할-구성원 바인딩(role-member binding)
$\eta : R \times M \rightarrow A$ : 액션 생성 함수(action generating function)

위의 커뮤니티 기반의 유비쿼터스 컴퓨팅 공간에서 다음과 같이 개별 커뮤니티가 구성된다.

$$\mathbb{C} = \langle G, E, M, A \rangle$$

컴퓨팅 공간은 사용자가 직접적으로 서비스를 요구하거나 사용자의 요구를 간접적으로 인지하여, 이를 바탕으로 목적을 생성하고, 커뮤니티에 참여할 수 있는 제약조건을 구성한다. 그 다음, 제약조건을 바탕으로 구성원을 커뮤니티에 참여 시키고 액션을 생성하고 제어하는 구조로 이루어진다.

### 3.2 커뮤니티 운영 지원 프레임

커뮤니티는 사용자가 요구하는 서비스를 제공하기 위해 구성되며, 각 구성원들이 역할에 따라 액션을 수행하여 목적을 달성하게 된다. 또한, 유비쿼터스 공간에서는 동시에 서로 다른 목적을 가진 여러 커뮤니티가 발생 할 수 있으며, 공간내의 각 개체들은 여러 커뮤니티의 구성원으로 동시에 참여할 수 있다. 이때 각 커뮤니티의 목적들이 서로 상반되거나, 여러 커뮤니티에 참여한 구성원이 수행하는 역할들이 서로 충돌할 수 있다. 본 절에서는 동시에 형성된 커뮤니티들의 목적이 서로 상반되는 것을 예방하는 관리 프레임과 각 개체가 수행하는 역할들이 서로 충돌하는 것을 예방하는 관리 프레임을 설명한다.

#### 3.2.1 커뮤니티의 목적

커뮤니티의 목적은 사용자의 요구에 따라 생성된다. 그러므로 처음 사용자의 요구가 발생했을 때에는 아직 목적이 달성되지 않았고, 적절한 커뮤니티 구성원들이

선택되어 각자의 역할을 수행할 경우에 목적이 달성된다. 사용자의 요구를 받아 들여 목적을 달성하는데 까지 과정을 유한 상태 기계(FSA)를 이용하여 정의하면 다음과 같다.

정의 1 : 임의의 커뮤니티의 목적 $g_j$를 다음과 같이 정의한다.

$g_j = \langle Q_j^g, \Sigma_j^g, \delta_j^g, q_{j,0}^g, Q_{j,m}^g \rangle$,

$Q_j^g = \{0, 1, 2\}$ : 목적의 상태 집합,

$\Sigma_j^g = \Sigma_{c,j}^g \cup \Sigma_{uc,j}^g$ : 입력 인자(input alphabet)의 집합

$\Sigma_{c,j}^g = \{\text{achieve}_j, \text{defer}_j\}$ : 제어 가능한(controllable) input 집합

$\Sigma_{uc,j}^g = \{\text{request}\}$ : 제어 불가능한(uncontrollable) input 집합

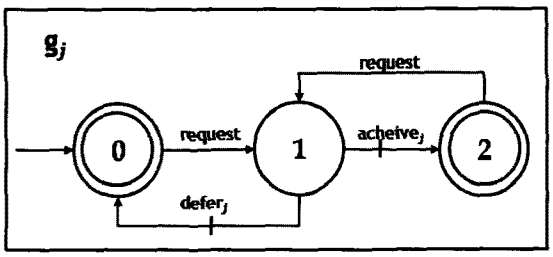
request : 입력 이벤트 통보(input event notification)

$\delta_j^g : Q_j^g \times \Sigma_j^g \rightarrow Q_j^g$ : 상태 전이 함수(state transition function)

$q_{j,0}^g = 0$ : 초기 상태(initial state)

$Q_{m,j}^g = \{0, 2\}$ : 수락 상태 집합(set of accepting states)

$g_j$는 목적이 사용자의 요구에 의해 발생하여 달성할 때까지의 상태 변화를 표현한 것이다. $g_j$는 목적이 커뮤니티에 의해 달성되는 과정에서 유한한 상태 수를 가진다고 가정하였다. 목적 $g_j$는 사용자의 요구(request)에 의해 생성될 때 상태 1로 전이하고, 다른 커뮤니티의 목적과 상반되지 않는다면 성취되어($\text{achieve}_j$) 상태 2로 전이하고 그렇지 않다면 보류되어($\text{defer}_j$) 상태 0으로 전이한다. 다음 <그림 1>은 $\delta_j^g$의 상태 전이 다이어그램을 나타낸 것이다.




<그림 1> $g_j$의 상태 전이 다이어그램

$g_j$는 두 가지 종류의 입력 인자(input alphabet)로 나누어져 있다. 제어 가능한(controllable) 입력(input)은 유비쿼터스 공간에서 각 커뮤니티의 목적을 관리하는 프레임에서 처리하는 결정 변수이며, 제어할 수 없는(uncontrollable) 입력(input)은 사용자에 의해 발생하는 요청을 나타낸다.

유비쿼터스 컴퓨팅 공간에서 여러 커뮤니티가 동시에 존재할 수 있으므로, 각 커뮤니티의 목적들을 하나의 프레임으로 표현할 필요가 있다.

정의 2 : 유비쿼터스 컴퓨팅 공간에 동시에 존재하는 커뮤니티의 목적들을 개별적인 목적 $g_j$의 parallel composition으로 다음과 같이 정의한다.

$\text{GOAL} = \| \{g_j \mid j = 1, \cdots, l\}$,

$\Sigma^g = \bigcup_{j=1}^{l} \Sigma_j^g = \Sigma_c^g \cup \Sigma_{uc}^g$, where

$\Sigma_c^g = \bigcup_{i=1}^{l} \Sigma_{c,j}^g$, $\Sigma_{uc}^g = \bigcup_{i=1}^{l} \Sigma_{uc,j}^g$이고, $l$은 유비쿼터스 컴퓨팅 공간에 존재하는 커뮤니티의 수이다.

### 3.2.2 구성원의 액션

각 구성원은 참여한 커뮤니티에 따라 수행해야 할 역할을 부여 받는다. 그러므로 그 역할에 따라 실행하는 액션 또한 서로 다르다. 액션은 커뮤니티의 요청에 의해 구성원이 실행하는데, 이것을 $g_j$와 유사한 유한 상태 기계로 표현할 수 있다.

정의 3 : 커뮤니티의 목적 달성을 위해 구성원이 수행하는 액션을 다음과 같이 정의한다.

$a_j = \langle Q_j^a, \Sigma_j^a, \delta_j^a, q_{j,0}^a, Q_{j,m}^a \rangle$,

$Q_j^a = \{0, 1, 2\}$ : 액션 상태 집합

$\Sigma_j^a = \Sigma_{c,j}^a \cup \Sigma_{uc,j}^a$ : 입력 인자 집합

$\Sigma_{c,j}^a = \{\text{invoke}_j, \text{revoke}_j\}$ : 제어 가능한 입력 인자 집합

$\Sigma_{uc,j}^a = \{\text{request}\}$ : 제어 불가능한 입력 인자 집합

request : 입력 이벤트 통보

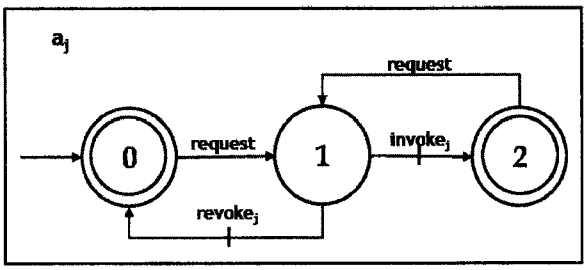
$\delta_j^a : Q_j^a \times \Sigma_j^a \rightarrow Q_j^a$ : 상태 전이 함수

$q_{j,0}^a = 0$ : 초기 상태

$Q_{m,j}^a = \{0, 2\}$ : 수락 상태 집합

$a_j$는 액션을 기술한 제 3장의 $a_j$와는 달리 액션의 상태 수를 유한하다고 보고 상태 변화를 표현한 것이다. 즉, 액션 $a_j$는 구성원이 액션을 실행할 때, 액션이 유한한 상태 수를 가진다는 것을 나타낸다. 구성원의 액션

이 요청(request)을 받으면 액션은 상태 1로 전이하고 실행 여부를 결정한다. 이때 다른 액션과의 충돌이 발생하지 않는다면 액션은 실행(invoke$_j$) 될 것이고, 그렇지 않다면 취소(revoke$_j$) 될 것이다. 다음 <그림 2>는 $\delta_j^a$의 상태 전이 다이어그램을 나타낸 것이다.




<그림 2> $a_j$의 상태 전이 다이어그램

$a_j$ 역시 두 가지 종류의 입력 인자(input alphabet)으로 나누어져 있다. 제어 가능한(controllable) 입력(input)은 유비쿼터스 컴퓨팅 공간의 각 구성원이 처리하는 결정 변수이며, 제어할 수 없는(uncontrollable) 입력(input)은 사용자에 의해 발생하는 요청을 나타낸다.

유비쿼터스 컴퓨팅 공간에서 여러 구성원이 동시에 여러 커뮤니티에 참여하여 다양한 액션을 수행할 수 있으므로 각 구성원들의 액션들 역시 하나의 프레임으로 표현할 필요가 있다.

정의 4 : 유비쿼터스 컴퓨팅 공간에 동시에 존재하는 구성원들의 액션들을 개별적인 액션 $a_j$의 parallel composition으로 다음과 같이 정의한다.

ACTION $= \| \{a_j \mid j = 1, \cdots, m\}$,

$\Sigma^a = \bigcup_{j=1}^{m} \Sigma_j^a = \Sigma_c^a \cup \Sigma_{uc}^a$, where

$\Sigma_c^a = \bigcup_{i=1}^{m} \Sigma_{c,j}^a$, $\Sigma_{uc}^a = \bigcup_{i=1}^{m} \Sigma_{uc,j}^a$이고, $m$은 유비쿼터스 컴퓨팅 공간에 존재하는 액션의 수이다.

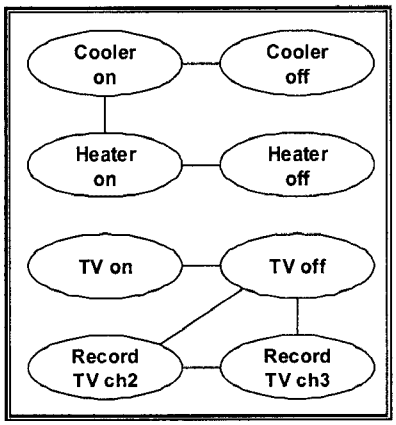
### 3.2.3 커뮤니티 관리 프레임

커뮤니티의 목적과 구성원의 액션은 각각 성취되고 실행하기 전에 상반되거나 충돌이 발생하지 않는지를 점검해야 한다. 본 연구에서는 상호 배타적인 관계(mutually exclusive relationship)를 형성하는 목적 혹은 액션들의 충돌을 예방하는 메커니즘을 제안한다. 목적과 액션의 상호 배타적인 관계는 각각 다음의 목적 제약과 액션 제약에 의해 표현된다.

정의 5 : 목적들간의 제약 관계와 액션들 간의 제약 관계는 각각 그래프 $GC = (G, \Gamma)$와 $AC = (A, \Theta)$로 정의된다.

단, $G$는 유비쿼터스 공간에 동시에 존재하는 커뮤니티들의 목적 집합을 나타내고, $\Gamma$는 두 목적간의 제약관계를 나타낸다. $A$는 유비쿼터스 공간에 존재하는 구성원들의 액션 집합을 나타내고, $\Theta$는 두 액션간의 제약 관계를 나타낸다.

<그림 3>은 동시에 실행될 수 없는 액션들의 제약을 나타낸 액션 제약 그래프의 예를 나타낸 것이다.




<그림 3> 액션 제약 그래프의 예

이러한 목적과 액션간의 제약은 다음과 같이 정의할 수 있다.

정의 6 : 서로 다른 두 목적의 제약과 액션의 제약은 다음과 같이 정의한다.

- 목적 제약(goal constraints)

  $GC_i(j, k) = \langle X_i^g, \Sigma_i^g, \eta_i^g, x_{i,0}^g, X_{i,m}^g \rangle$,

  $X_i^g = \{0, 1, 2\}$ : 목적 $g_i$와 $g_k$의 목적 제약 상태 집합

  $\Sigma_i^g = \Sigma_j^g \cup \Sigma_k^g$ : 목적 집합

  $\eta_i^g : X_i^g \times \Sigma_i^g \rightarrow X_i^g$ : 상태 전이 함수

  $x_{i,0}^g = 0$ : 초기상태

  $X_{m,i}^g = \{1, 2\}$ : 수락 상태 집합

- 액션 제약(action constraints)

  $AC_i(j, k) = \langle X_i^a, \Sigma_i^a, \eta_i^a, x_{i,0}^a, X_{i,m}^a \rangle$,

  $X_i^a = \{0, 1, 2\}$ : 액션 $a_i$와 $a_k$의 액션 제약 상태 집합

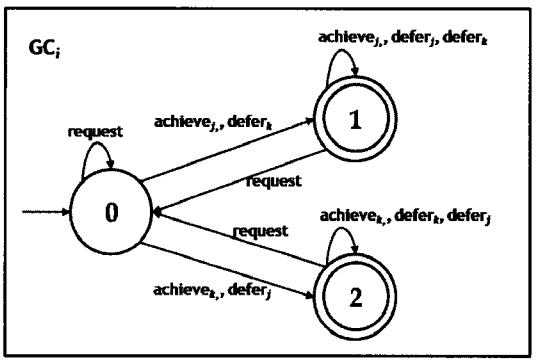
  $\Sigma_i^a = \Sigma_j^a \cup \Sigma_k^a$ : 액션 집합

$\eta_i^a : X_i^a \times \Sigma_i^a \rightarrow X_i^a$ : 상태 전이 함수

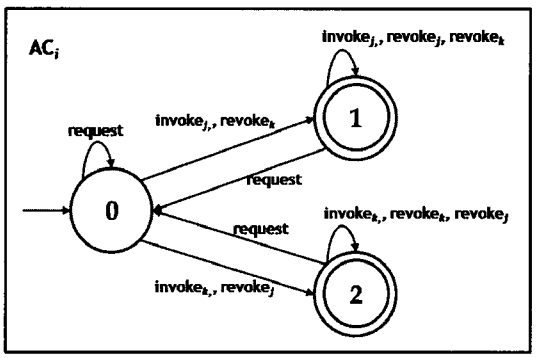
$x_{i,0}^a = 0$ : 초기상태

$X_{m,i}^a = \{1, 2\}$ : 수락 상태 집합

목적(액션) 제약은 상호 충돌적인 관계를 가지는 목적(액션)이 동시에 추구되지 않도록 조정하는 코디네이터의 작용에 이용된다. 다음 <그림 4>와 <그림 5>는 목적 제약과 액션 제약의 상태 전이 다이어그램을 나타낸 것이다.




<그림 4> 목적 제약의 상태 전이 다이어그램 : Self-loop $\left[\sum^g - \sum_i^g\right]$




<그림 5> 액션 제약의 상태 전이 다이어그램 : Self-loop $\left[\sum^g - \sum_i^g\right]$

위의 <그림 4>와 <그림 5>에서 알 수 있듯이 목적(액션) j와 k가 사용자의 요구에 의해 생성되고 두 목적(액션)이 서로 충돌이 발생하는 경우 $GC_i(AC_i)$에 의해 두 목적(액션)들이 동시에 성취(실행)될 수 없음을 알 수 있다. 목적 $g_i$(액션 $a_j$)가 먼저 달성 되었다면 $GC_i(AC_i)$는 achieve$_j$(invoke$_j$)를 받아서 state 1로 전이 되고 state 1에서는 achieve$_k$(invoke$_k$)는 발생할 수 없게 된다. $GC_i(AC_i)$가 state 2로 전이되는 경우에는 반대로 achieve$_j$(invoke$_j$)를 생성할 수 없게 된다. 이러한 각각의 목적(액션) 제약은 목적(액션)들이 서로 상반되거나 충돌이 발생하는 것을 예방하는 역할을 하게 된다. <그림 4>와 <그림 5>에서 Self-loop $\left[\sum^g - \sum_i^g(\sum^a - \sum_i^a)\right]$는 $GC_i(AC_i)$의 모든 상태에서 $\sum^g - \sum_i^g(\sum^a - \sum_i^a)$에 해당하는 모든 alphabet은 자기 자신의 상태로 전이한다는 것을 의미한다.

### 3.2.4 커뮤니티 관리 프레임의 논-블로킹

본 절에서는 주어진 목적과 액션 제약을 고려한 모델이 각 커뮤니티의 목적과 구성원의 액션을 조정하는 동시에 논-블로킹 특성을 갖는 관리 프레임임을 보인다.

정리 1 : $GC_i(j, k), i = 1, \cdots, |\Gamma|$는 GOAL의 modular supervisor들이다.

증명) 제어 이론에서 supervisor는 대상 시스템이 특정 상태에 있을 때 시스템에서 발생하게 될 제어 가능 입력 인자(controllable input)들 만을 제어할 수 있는 automaton이다. 이를 바탕으로 $GC_i$에 대해 다음과 같은 함수를 정의한다.

$RC_i : X \rightarrow 2^{\Sigma}$

$RC_i(x)$ : 어떤 상태 $x$에서 $\eta_i$에 정의되지 않은 입력 인자(input alphabet)의 집합

$RC_i(x)$은 $R_i(j, k)$가 어떤 상태에 머무르게 됨으로써 발생할 수 없는 alphabet들의 집합을 나타낸다.

$RC_i(0) = \phi$

$RC_i(1) = \{\text{Process}_k\} \subseteq \Sigma_c$

$RC_i(2) = \{\text{Process}_j\} \subseteq \Sigma_c$

이를 통해 $GC_i$는 단지 제어 가능 입력 인자들만을 제어할 수 있음을 알 수 있다. 그러므로 각각의 $GC_i$는 목적에 대한 modular supervisor이다. ■

각각의 목적 제약들은 각 커뮤니티의 목적들이 상반되게 충돌되는 것을 예방하기 위해 전체 컴퓨팅 공간에 있는 커뮤니티의 목적이 달성되는 것을 제어하는 일을 하게 된다. 적어도 한 목적 제약에 의해 제한(disable)된 제어 가능한 입력 인자는 발생할 수 없게 된다. 유비쿼터스 컴퓨팅 공간에서 여러 목적 제약들에 의해 제어되는 목적들의 행동양식(controlled behavior)은 제약이론에 의해 하나의 유한 상태 기계로 표현될 수 있으며, 이러한 유한 상태 기계는 다음과 같은 형식언어(language) 체계에 속하게 된다.

$$L_m(W) = L_m(\text{GOAL}) \cap \left( \bigcap_{i=1}^{n} L_m(g_i) \mid n = |\Gamma| \right)$$

$$L(W) = L(\text{GOAL}) \cap \left( \bigcap_{i=1}^{n} L(g_i) \mid n = |\Gamma| \right)$$

유한 상태 기계 $W$가 논-블로킹 특성을 가진다는 것을 보이기 위해 다음의 보조정리 1을 먼저 증명한다.

보조정리 1 : $L(W) \cap L_m(\text{GOAL}) = L_m(W)$

증명) 보조정리는 다음의 두 가지 경우로 나누어 증명한다.

Case 1) $L(W) \cap L_m(\text{GOAL}) \supseteq L_m(W)$

Case 2) $L(W) \cap L_m(\text{GOAL}) \subseteq L_m(W)$

1)의 경우는 $L_m(W) \subseteq L(W)$

$\Rightarrow L_m(W) \cap L_m(\text{GOAL}) \subseteq L(W) \cap L_m(\text{GOAL})$

$\Rightarrow L_m(W) \subseteq L(W) \cap L_m(\text{GOAL})$.

2)의 경우는 임의의 문자열(string) $s \in L(W) \cap L_m(\text{GOAL})$가 항상 $s \in L_m(W)$임을 보이면 된다. 임의의 문자열 $s$가 $L(W) \cap L_m(\text{GOAL})$에 포함된다고 하고, $s_j$를 $L_m(\text{GOAL}_j)$에 포함되는 $s$의 부분 문자열(substring)이라고 하자. $s_k$도 마찬가지로 $L_m(\text{GOAL}_k)$에 포함되는 $s$의 부분 문자열(substring) 이라고 하자. 이때 $\text{GOAL}_j$와 $\text{GOAL}_k$는 $GC_i$에 의해 제약을 받는다. $s_j \in L_m(\text{GOAL}_j)$이므로 $\delta_j(0, s_j) = 0$ or $2 \in Q_{m,j}$, $j = 1, \cdots, l$임을 알 수 있다. $s_k$의 경우도 마찬가지이다. 그러나 $GC_i$에 의해 $\delta_k(0, s_k) = 0$이면 $\delta_j(0, s_j) = 0$ or 2이고, $\delta_j(0, s_j) = 0$이면 $\delta_k(0, s_k) = 0$ or 2이다. *achieve*(*s*), *defer*(*s*)를 문자열(string) $s$ 중 achieve 및 defer 입력(input)의 출현 횟수라고 하자. $\text{GOAL}_j$와 $\text{GOAL}_k$의 상태 전이 함수에서 $\text{request}(s_j) = \text{achieve}_j(s_j) + \text{defer}_j(s_j)$이고 $\text{request}(s_k) = \text{achieve}_k(s_k) + \text{defer}_k(s_k)$이며, $\text{request}(s) = \text{achieve}_j(s) + \text{defer}_j(s) + \text{achiver}_k(s) + \text{defer}_k(s)$임을 알 수 있다. 이를 $\eta_i^g$의 정의에 적용시키면, $\eta_i^g(0, s) = 1$ or $2 \in X_{m,i}^g$, $i = 1, \cdots, |\Gamma|$임을 알 수 있다. 그러므로 $s \in L_m(g_i)$임을 알 수 있다. 결국 문자열 $s$는 $L_m(\text{GOAL})$에 포함되면서 동시에 $\left( \bigcap_{i=1}^{n} L_m(g_i) \mid n = |\Gamma| \right)$에 포함된다. 그러므로 $s \in L_m(W)$임을 알 수 있다. ■

정리 2 : 목적 제약을 나타내는 유한 상태 기계 $W$는 논-블로킹 특성을 갖는다.

증명) 어떤 문자열 $s \in L(W)$에 대해 $su \in L_m(W)$이게 하는 문자열 $u \in \Sigma^*$가 항상 존재함을 보이면 된다. 문자열 $s \in L(W)$에 대해 다음이 성립함을 $\delta_j$, $j = 1, \cdots, l$를 통해 알 수 있다.

1) $\text{request}(s_j) = \text{achieve}_j(s_j) + \text{defer}_j(s_j)$ 또는

2) $\text{request}(s_j) = \text{achieve}_j(s_j) + \text{defer}_j(s_j) + 1$

1)의 경우 $\delta_j^g(0, s_j) = 0$ or 2이므로 $s_j \in L_m(\text{GOAL}_j) \Rightarrow s \in L_m(\text{GOAL})$이다. 그러므로 $s \in L(W) \cap L_m(\text{GOAL})$이며 보조정리 1에 의해 $L(W) \cap L_m(\text{GOAL}) = L_m(W)$이므로 $s \in L_m(W)$임을 알 수 있다. 2)의 경우 $\delta_j(0, s_j) = 1$이고 이때 $s_j \text{achieve}_j \in L_m(\text{GOAL}_j)$ 또는 $s_j \bullet \text{defer}_j \in L_m(\text{GOAL}_j)$이고, $s\text{achieve}_j \in L_m(\text{GOAL})$, $s \bullet \text{defer}_j \in L_m(\text{GOAL})$임을 알 수 있다. 또한 $s\text{achieve}_j \in L(W)$, $s \bullet \text{defer}_j \in L(W)$임을 알 수 있다. 이를 문자열로 확장하면 $s_j u_j \in L_m(\text{GOAL}_j)$이게 하는 마지막 alphabet이 항상 제어 가능 인자인 문자열 $u_j$가 존재하며 $su \in L_m(\text{GOAL})$이게 하는 문자열 $u$가 존재함을 알 수 있다. 또한 $su \in L(W)$이므로, 결국 $su \in L(W) \cap L_m(\text{GOAL})$이다. 보조정리 1에서 $L(W) \cap L_m(\text{GOAL}) = L_m(W)$이므로 $su \in L_m(W)$이며, 문자열 $u$가 항상 존재함을 알 수 있다. 그러므로 유한 상태기계 $W$는 논-블로킹이다. ■

액션 제약의 경우도 위와 유사한 과정을 통해 증명될 수 있다.

## 4. 유비쿼터스 환경에서의 커뮤니티 예제

본 절에서는 커뮤니티의 목적들이 서로 상반되지 않도록 제안된 커뮤니티 운영지원 프레임을 이용하는 커뮤니티 컴퓨팅 예제를 설명한다. 본 예제에서는 다음과 같이 두 종류의 커뮤니티가 형성된다.

(1) 출퇴근 시간에 신호 위반으로 인한 교통 정체가 발생하므로 이를 단속하기 위해 신호 위반 단속 커뮤니티가 형성된다. 커뮤니티는 신호 감시 카메라와 구역마다 배치된 교통경찰에 의해 운영된다.

(2) 출근시간에 교통사고로 인한 응급환자가 발생하여, 응급환자 이송 커뮤니티가 형성된다. 이송 커뮤니티는 구조대원과 구급차에 의해 운영된다.

위 예에서 신호 위반 단속 커뮤니티의 목적은 신호 위반에 의해 발생하는 교통정체를 줄이는 것이고, 응급환자 이송 커뮤니티는 환자를 빨리 병원으로 이송하는 것이 목적이다. 응급환자를 빨리 이송해야 하는 응급환

자 이송 커뮤니티는 불가피하게 신호위반을 행해야 하는 경우가 발생할 수 있으므로 신호 위반 단속 커뮤니티에 의해 환자 이송에 지연 될 수 있다. 그러나 신호 위반 단속 커뮤니티의 경우에는 환자 이송 커뮤니티 때문에 교통정체를 줄이지 못하고 오히려 증가 시키게 된다. 그러므로 두 커뮤니티의 목적은 서로 상반되는 관계에 있다.

그러면 본 연구에서 제안한 목적 제약을 이용하여 두 커뮤니티의 목적이 서로 충돌하지 않도록 예방하는 방법을 설명하기 위해, 먼저 신호위반 단속 커뮤니티와 응급환자 이송 커뮤니티의 목적을 다음과 같이 정의하자.

$g_{TM} = \langle Q^g_{TM}, \Sigma^g_{TM}, \delta^g_{TM}, q^g_{TM,0}, Q^g_{TM,m} \rangle$ : 신호 위반단속

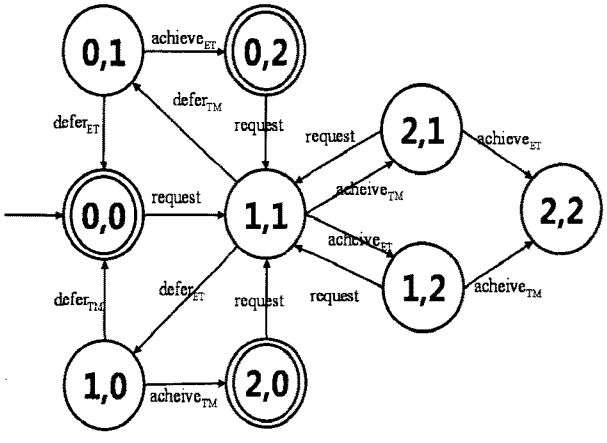
$g_{ET} = \langle Q^g_{ET}, \Sigma^g_{ET}, \delta^g_{ET}, q^g_{ET,0}, Q^g_{ET,m} \rangle$ : 응급환자 이송

두 커뮤니티 $g_{TM}$와 $g_{ET}$는 parallel composition을 이용하여 다음과 같이 하나의 유한상태기계로 나타낼 수 있다.

$$g_{TM} \| g_{ET} = \left\langle \begin{array}{l} Q^g_{TM} \times Q^g_{ET}, \Sigma^g_{TM} \cup \Sigma^g_{ET}, \delta^g_{TM,ET}, \\ (q^g_{TM,0}, q^g_{ET,0}), Q^g_{TM,m} \times Q^g_{ET,m} \end{array} \right\rangle$$

단, $\delta^g_{TM,ET}\left((q^g_{TM}, q^g_{ET}), e\right) = \left(\delta^g_{TM}(q^g_{TM}, e), \delta^g_{ET}(q^g_{ET}, e)\right)$는 커뮤니티 $g_{TM}(g_{ET})$의 상태 $q^g_{TM}(q^g_{ET})$에서 입력 인자 $e$가 인식할 수 있으면 상태전이함수 $\delta^g_{TM}(\delta^g_{ET})$에 의해 상태 $\delta^g_{TM}(q^g_{TM}, e)(\delta^g_{ET}(q^g_{ET}, e))$로 변하게 되고, $e$를 인식할 수 없다면 $\delta^g_{TM}(q^g_{TM}, e) = q^g_{TM}(\delta^g_{ET}(q^g_{ET}, e) = q^g_{ET})$가 된다.

다음 <그림 6>은 유한상태기계 $g_{TM} \| g_{ET}$의 상태전이함수 $\delta^g_{TM,ET}$를 나타낸 것이다.




<그림 6> $g_{TM} \| g_{ET}$의 상태전이 다이어그램

<그림 6>에서 상태 (2, 2)는 상반되는 관계에 있는 두 목적이 성취되는 상태이다. 이것은 두 커뮤니티의 목적이 성취되게 되면 신호 위반 단속 커뮤니티는 교통정체를 줄이지 못하고, 응급환자 이송 커뮤니티는 환자 이송이 지연될 수 있음을 의미한다.

이러한 유한상태기계 $g_{TM} \| g_{ET}$는 제어이론과 목적 제약을 이용하여 두 목적이 서로 충돌하지 않도록 할 수 있다. 우선 목적 제약 $GC(TM, ET)$을 다음과 같이 정의한다.

$$GC(TM, ET) = \langle X^g, \Sigma^g, \eta^g, x^g_0, X^g_m \rangle$$

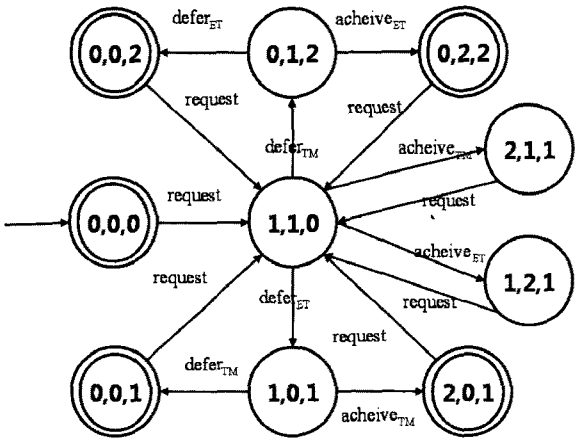
제어이론을 이용하여 유한상태기계 $g_{TM} \| g_{ET}$에 목적 제약 $GC(TM, ET)$를 적용하면 다음과 같다.

$$W = \left\langle \begin{array}{l} Q^g_{TM} \times Q^g_{ET} \times X^g, \Sigma^g, \delta^q_{TM,ET} \times \eta^g, \\ (q^g_{TM,0}, q^g_{ET,0}, x^g_0), Q^g_{TM,m} \times Q^g_{ET,m} \times X^g_m \end{array} \right\rangle$$

$$\text{단, } \begin{array}{l} \delta^g_{TM,ET} \times \eta^g\left(\left(q^g_{TM}, q^g_{ET}, x^g\right), e\right) = \\ \left(\delta^g_{TM,ET}\left(\left(q^g_{TM}, q^g_{ET}\right), e\right), \eta^g\left(x^g, e\right)\right) \end{array} \quad (1)$$

위 식 (1)은 $\delta^g_{TM,ET}\left(\left(q^g_{TM}, q^g_{ET}\right), e\right)$과 $\eta^g(x^g, e)$가 둘 다 동시에 존재해야만 성립한다.

<그림 7>은 유한상태기계 $W$의 상태전이함수를 표현한 것이다.




<그림 7> W의 상태전이 다이어그램

유한상태기계 $W$의 상태는 27개의 조합이 있을 수 있으나, $\delta^g_{TM,ET} \times \eta^g$의 정의에 의해 <그림 7>과 같이 10개의 상태만이 존재한다. $W$의 상태정의$\left(q^g_{TM}, q^g_{ET}, x^g\right)$에 의해 $q^g_{TM}, q^g_{ET}$는 $g_{TM} \| g_{ET}$의 상태를 의미한다. 그러므로

<그림 7>에서 알 수 있듯이 본 연구에서 제안한 목적 제약 $GC(TM, ET)$가 제어이론에 의해 커뮤니티의 목적 $g_{TM} \| g_{ET}$에 적용이 되면, $g_{TM} \| g_{ET}$는 상태 $(2,2)$로 전이할 수 없으므로 두 커뮤니티간의 목적은 서로 충돌하지 않음을 알 수 있다.

## 5. 결 론

본 연구에서는 유비쿼터스가 지향하고 있는 사용자 중심(human-centered)의 서비스를 제공하고, 다양한 서비스를 통해 시스템 내에 속한 각 구성원들에게 양질의 환경 이용을 제공하기 위해서는 각 구성원들이 지향하는 목적 및 커뮤니티 내에서의 액션들에 대한 효과적인 코디네이터의 필요성을 만족시키기 위해 정형적 접근 방법을 이용한 목적 제약과 액션 제약이 제시되었다.

제시된 코디네이터는 그 표현방식에 있어 이해도를 높일 수 있는 정형적(formal) 접근 방법이 이용되었으며, 이를 통해 코디네이터가 가져야 하는 필수적인 논-블로킹 특성을 확보할 수 있도록 고안되었다. 본 논문에서는 코디네이터의 수리적 증명과 함께 커뮤니티 예제를 통해 실제 코디네이터가 어떻게 작동하는지에 대해 설명하였다.

본 논문에서 제시된 코디네이터의 모델은 상호 배타적인 관계만을 기술하였으나, 그 구조상 다양한 관계들을 포함시킬 수 있는 기반을 제공할 수 있다. 그러나 이때 코디네이터의 논-블로킹 특성을 규명하기 위한 추가적인 연구가 필요할 것으로 판단된다. 또한 진화형 커뮤니티 환경 구축을 위해서는 시간을 고려한(temporal) 모델의 개발 및 검증이 요구된다. 이러한 연구는 본 연구를 통해 제시된 목적 및 액션 모델과 코디네이터 모델에 시간의 인자들을 추가함으로써 확장될 것으로 기대된다.

## 참고문헌


[1] 강경란, et al.; "커뮤니티 컴퓨팅 : 협업 기반 환경 자동 적응의 컴퓨팅 모델", 정보과학회지, 24(12) : 84-91, 2006.

[2] Brandin, B. A.; "The Real-time Supervisory Control of An Experimental Manufacturing Cell," *IEEE Trans. On Robotics and Automation*, 12(1) : 1-14, 1996.

[3] Cassandras, C. G. and Lafortune, S.; *Introduction to Discrete Event Systems*, Springer, New York, 1999.

[4] Castillo, I. and Smith, J. S.; "Formal Modeling Methodologies for Control of Manufacturing Cells : survey and comparison," *Journal of Manufacturing Systems*, 21(1) : 40-57, 2002.

[5] Feng, L. and Chen, L.; "Achieving Online Coordination in real-time collaborative assembly modeling : a supervisory control approach," *Journal of Computing and Information Science in Engineering*, 6(3) : 252-262, 2006.

[6] Hopcroft, J. E., Motwani, R., and Ullman, J. D.; *Introduction to Automata Theory, Languages, and Computataion*, 2nd Ed. Addison-Wesley, New York, 2001.

[7] Jung, Y., Lee, J., and Kim, M.; "Multi-agent based community computing system development with the model driven architecture," in Proceedings of the fifth international joint conference on Autonomous agents and multiagent, Hakodate, Japan, 1329-1331, 2006.

[8] Jung, Y., Lee, J., and Kim, M.; "Community based Ubiquitous System Development in Multi-agent Environment," in CAiSE'06 Workshop Ubiquitous Mobile Information and Collaboration Systems(UMICS 2006), Luxembourg, Grand-Ducy of Luxembourg, 2006.

[9] Kim, H., et al.; "Community Manager : A Dynamic Collaboration Solution on Heterogeneous Environment," in 2006 ACS/IEEE International Conference on Pervasive Services, Lyon, France, 2006.

[10] Lupu, E. C. and Sloman, M. S.; "Conflicts in Policy-Based Distributed Systems Management," *IEEE Transactions on Software Engineering-Special Issue on Inconsistency Management*, 25(6) : 852-869, 1999.

[11] Sivashanmugam, K., et al.; "Framework for Semantic Web Process Composition," *International Journal of Electronic Commerce*, 9(2) : 71-106, 2004.

[12] Shim, Y.-J., et al.; "A Ubiquitous System Architecture Based on the Community Computing Model," in IC-HIT, Jeju, Korea, 2006.

[13] Wong, K. C. and Wonham, W. M.; "Modular Control and Coordination of Discrete-event Processes," *Discrete Event Dynamic Systems : Theory and Applications*, 8 (3) : 247-297, 1998.