특집

유비쿼터스 모바일 자율 커뮤니티 네트워킹 기술

조위덕, 강경란, 이정태 (아주대학교)

요 약

본 논문은 커뮤니티라는 메타포(metaphor)를 이용한 유비쿼터스 컴퓨팅 환경의 구축에 관한 문제를 고찰한다. 이를 구현하기 위한 방법으로 먼저 커뮤니티 기반 컴퓨팅의 포괄적인 개념과 기술 이슈를 정립하였고, 커뮤니티의 효과적인 상호작용을 지원하기 위해 통합 프레임워크를 이용, 상황인지기반 커뮤니티 관리 컴포넌트들을 설계하고자 했다. 본 논문의 결과로서 커뮤니티의 클래스와 라이프 모델, 커뮤니티 기반 컴퓨팅을 위한 시스템 아키텍처를 도출하였다. 이로써 복합적인 유비쿼터스 서비스를 자율적으로 제공하기 위한 유비쿼터스 환경 구축기반을 마련하였다.

1. 개요

컴퓨팅 기술 및 네트워크 기술의 발달로 유비쿼터스 컴퓨팅 환경 구축에 대한 관심이 높아지고 있다. 1990대부터 전 세계적으로 연구가 활발하게 진행되어 오고 있다. 기존의 연구들은 주로 유비쿼터스 컴퓨팅 환경 내의 개별 장치에 컴퓨팅/네트워킹 기능과 지능적인 판단 기능들을 강화하여 개별 상황에 어떻게 효율적이고 지능적으로 대응할 것인가에 주력하고 있다. 그 결과로, 개별적인 센서, 서비스, 네트워크 기술 등 세부적인 기술들이 계속해서 발전해 오고 있다.

이러한 세부 기술들을 통합하여 유비쿼터스 컴퓨팅 환경을 구축하는 데 있어서는 단순하게 장치들이나 서비스들을 나열하고 이들 각각이 필요에 따라 협력 대상을 검색하여 일시적으로 협력하도록 하는 경우가 일반적이다. 즉, 어떤 주어진 상황에 대응해서 처리하는 장치나 서비스들이 다수 존재하고, 이러한 상황이 일시적으로 그치는 것이 아니라 반복될 수 있다는 고려가 배제되어 있다. 유비쿼터스 컴퓨팅 환경에서 발생할 수 있는 상황에 대한 연구가 필요하고, 또한 각 상황에서의 장치 혹은 서비스들이 협력하는 관계에 대한 포괄적인 개념을 정립하여 그 개념 하에서 전체 유비쿼터스 컴퓨팅 환경을 구축하기 위한 노력이 필요하다.

University of Texas, Arlington의 PICO project나 Arizona State University의 RCSM

project에서는 이와 유사한 고려 하에 연구를 진행하고 있다. RCSM은 일상 생활에서의 group application을 유비쿼터스 컴퓨팅 환경에서 상황 인지를 기반으로 지원하기 위한 middleware를 개발하는데 주력하고 있다. PICO project도 역시 middleware 개발에 주력하고 있다는 점에서 RCSM과 접근 방법이 비슷하나, RCSM에 비해서 비교적 광범위한 개념의 커뮤니티를 고려하고 있다. 공통의 목표를 달성하기 위해 협력하는 'delegent'들의 모임을 커뮤니티라고 칭하며, mobile computing 개념을 도입하여 'delegent'이 여러 다양한 장치들을 이동해 다니면서 필요한 task를 완수하게 된다.

본 연구에서는 RCSM이나 PICOproject에서와 같이 middleware 차원에서의 커뮤니티를 지원하기 위한 방법론을 개발하는 것에 그치지 않고, 유비쿼터스 컴퓨팅 환경을 구축하는데 있어 개별 장치나 서비스들 간의 collaboration model의 metaphor로서 '커뮤니티' 모델을 활용한다. 개별 장치나 서비스들이 독자적인 기능을 가지고 개발되지만, 보다 복잡하고 다양한 서비스를 제공하기 위해 'Community'를 형성하게 된다. 그리고, 유비쿼터스 컴퓨팅 환경 내의 장치나 네트워크의 자원들도 이러한 커뮤니티를 지원하기 위해 동적으로 allocate되고 재구성될 수 있다. 이 '커뮤니티'는 기존의 PICO나 RCSM에서 제시하는 커뮤니티나 그룹보다 더 광범위한 개념이다. 우선, 커뮤니티 모델을 사용자 관점에서의 서비스들 간의 collaboration을 위한 'service-level community'와 이 서비스 community가 실제 유비쿼터스 컴퓨팅 환경 내의 resource들을 효율적이고 형평성이 있게 또한 QoS 요구 사항에 만족하도록 할당하고 관리하는 'infrastructure-level community'로 구별한다. 커뮤니티는 일시적이고 일회적인 것일 수도 있지만, 일반적으로 life model을 가지고 진화하며 발전하는 것을 가정한다. 유비쿼터스 컴퓨팅 환경 내의 서비스나 장치들은 개발 단계에서 이미 이러한 'Community'라는 metaphor를 기반으로 하여 개발될 수도 있으며, 그렇지 못한 경우에도 본 연구에서의 유비쿼터스 컴퓨팅 환경 통합 플랫폼에서 제공하는 messaging architecture에 의해 쉽게 community의 member로서 참여할 수 있도록 한다.

본 논문의 구성은 다음과 같다. II장에서 커뮤니티 모델을 기반으로 한 유비쿼터스 컴퓨팅 환경에 대한 개념을 설명하고, III장에서는 커뮤니티 모델을 실현하기 위해 유비쿼터스 컴퓨팅 환경 개발 시에 고려해야 할 사항들을 기술한다. 그리고, IV장에서는 community model을 기반으로 개발되는 유비쿼터스 컴퓨팅 환경 내의 component들의 architecture를 개괄적으로 기술하고 구현한 예제 application을 설명한다. V장에서는, 본 연구와 유사한 연구를 수행하고 있는 PICO project와 RCSM project의 커뮤니티 모델을 소개한다. VI장에서 앞으로의 연구 진행 방향을 제시하는 것으로 본 논문을 마무리한다.

II. Community-based Computing

1. 'Community'의 정의

'Community'란 공통의 목표를 가지고 협력하는 유비쿼터스 컴퓨팅 환경 내의 장치들과 서비스들의 집합과 이들 간의 협력 관계를

상징한다. 즉, 하나의 커뮤니티는 〈goal, members, functions of the members〉를 구성 요소로 갖는다. 커뮤니티가 또 다른커뮤니티의 member가 될 수 있으며, 하나의 장치나 service가 여러 커뮤니티에 동시에 속할 수 있다. 즉, 커뮤니티들은 그 목표에 따라 서로 포함 관계에 있을 수도 있으며, 공유하는 장치가 서비스가 있을 수 있다.

'Community-based computing' 이란 이러한 커뮤니티라는 metaphor를 기반으로 service나 장치를 개발하고, 이들을 통합하여 새로운 서비스를 창출하는 것을 지칭한다. 또한, 동적으로 발생하는 새로운 서비스에 대한 요구에 대해, 새로운 개발을 시작하기보다 이미 유비쿼터스 컴퓨팅 환경에 구축된 서비스나 장치들을 커뮤니티로 묶어서 필요한 서비스를 구축할 수 있도록 한다.

2. Community의 goal

유비쿼터스 컴퓨팅 환경 내에 있는 사용자들이 쾌적하고 안전하며 편리한 생활을 할 수 있도록 하는 것이 전체 유비쿼터스 컴퓨팅 환경을 개발하는 개발자들의 목표이다. 그러므로, 유비쿼터스 컴퓨팅 환경을 구성하는 커뮤니티의 목표 역시 그에 부합하는 것들이 될 것이다. 궁극적으로 전체 커뮤니티들이 이상적인 목표를 달성하고자 할 것이며, 다양한 상황에 대해 이상적인 목표에 근접하기 위한 상황으로의 전이를 위한 task들이 커뮤니티에 의해 수행될 것이다. 즉, 보다 광범위한 목표를 가진 커뮤니티가 보다 특화된 목표를 가진 커뮤니티를 포함할 수 있다.

커뮤니티의 goal이 직접적으로 사용자의 요구 사항을 만족시키기 위한 것일 수도 있으며, 전체적으로 커뮤니티들이 각각의 중요성과 긴급성에 따라 필요한 자원들을 확보하고 상호 작용하도록 하는 것일 수 있다.

3. Community의 class

커뮤니티는 크게 두 가지 종류로 나누어 생각할 수 있다. 첫째, 'service-level community' 이다. 이 커뮤니티는 사용자의 요구 사항을 만족시키기 위한 서비스를 제공하는 것을 목적으로 하며, community의 구성원은 서비스를 수행하기 위한 애플리케이션들이 포함될 것이다. 하나의유비쿼터스 컴퓨팅 환경 내에는 동시에 여러 개의 service-level community가 run하고 있을 수 있다. 사용자의 입장에서는 service-level community만이 유비쿼터스 컴퓨팅 환경에 존재하는 커뮤니티라고 간주할 수 있다.

Service-level community가 필요한 기능을 수행할 수 있도록 하는 infrastructure를 제공하는 'infrastructure-level community' 가 있다. 'Infrastructure-level community'의 goal은 사용자의 요구에 직접 대응되지 않고, 유비쿼터스 컴퓨팅 환경 내의 device의 컴퓨팅 자원이나 network resource를 형평성이 있게 그리고 application의 서비스 품질 요구에 맞게 할당할 수 있도록 하는 것을 그 goal로 한다. 즉, service level community의 서비스 애플리케이션이 동작하는 장치나 network level에서의 packet delivery를 담당하는 라우터들이 커뮤니티의 member가 될 것이다.

4. Community의 life model

1) Community information creation

유비쿼터스 컴퓨팅 환경 내에서 완수해야 할 goal이 새롭게 만들어지면, 이를 도달하기 위해 필요한 service와 장치들이 선택될 것이고 어떤 식으로 협력해야 할 것인지 결정될 것이다.

이러한 정보는 커뮤니티의 goal이 미치는 영향의 범위에 따라 global storage에 위치할 수도 있고, local storage에 위치할 수도 있다. 그리고, 단순히 archive되는 것으로 그치는 것이 아니라 커뮤니티의 goal을 원하는 사용자 혹은 개체들이 있을 수 있으므로 커뮤니티의 scope에 따라 announce될 수 있다.

2) Community organization

커뮤니티에 대한 정보가 만들어지고 나면, 커뮤니티의 성격에 따라 커뮤니티를 trigger할 event가 발생하지 않았다고 하더라도 미리 해당하는 member들을 찾아서 커뮤니티에 참여하도록 요청할 수 있다. 커뮤니티가 추구하는 목표의 긴급한 정도나 중요한 정도에 따라 그리고 그 지속 기간에 따라 판단할 수 있을 것이다. 그리고, 커뮤니티의 goal이 외부 요인에 의해서 trigger되는 것이 아니라 커뮤니티 내의 member에 의해서 판단되어야 하는 경우에도 event가 발생하기 전에 필요한 member들이 확보되어야 할 것이다.

어떤 커뮤니티의 경우에는 trigger하는 event가 발생하는 시점에 커뮤니티에 속할 member들을 찾아서 구성하게 될 것이다. 이러한 커뮤니티는 자체 member들의 판단에 의한 것이 아니라 기존의 커뮤니티 혹은 유비쿼터스 컴퓨팅 환경 내의 타 서비스에 의한 요청을 처리하기 위한 경우가 해당할 것이다.

3) Community activation

해당 커뮤니티의 goal을 달성해야 할 triggering event가 발생하면 커뮤니티 내의 member들이 activate되어서 goal 달성을 위해 필요한 procedure들을 처리하게 된다. 어떤 entity가 같은 커뮤니티에 속하는가를 판단하는 것은 커뮤니티를 monitoring하는 entity가 담당한다. 이 monitoring하는 entity를 결정하는 것은 유비쿼터스 컴퓨팅 환경을 운영하는 strategy에 따라 결정될 것이다.

커뮤니티 내의 member들은 일대일로 data를 교환하거나 전체 member들에게 필요한 data를 distribute하고, 혹은 자신이 필요한 data를 register해서 이를 획득하는 식의 interaction이 발생할 것이다.

4) Community hibernation

커뮤니티의 goal이 달성되고 나면, 커뮤니티에 따라서는 다시 triggering event를 기다리면서 dormant 상태에 들어간다. 이 때, 중요한 것은 community 내의 member들이 goal의 달성 상태를 인지할 수 있어야 한다는 것이다. Community를 monitoring하는 entity가 전체 member에게 announce할 수도 있으며, goal의 final step을 담당하는 member가 전체 member들에게 announce할 수도 있을 것이고, community의 member들이 자신의 상태 정보를 공통의 저장소에 저장해 두고, 각 member들이 이 상태 정보를 refer하여 goal이 달성된 것을 확인하는 식의 방법이 가

능할 것이다.

Hibernation 상태에 들어간 커뮤니티는 다시 외부적인 triggering event에 의해서 그리고 내부 member의 triggering event detection에 의해 activation될 것이다.

5) Community transformation

해당 커뮤니티가 추구하는 목표는 변경되지 않는다 하더라도, 유비쿼터스 컴퓨팅 환경의 구성 요소나 configuration의 변경에 따라 커뮤니티에 참여하는 member의 요구되는 기능이나 종류가 변경될 수 있으며, member의 기능 변화에 따라 그 절차가 변경될 수 있다. 근본적인 goal이 변경하는 것이 아니고 community가 변형되어 가는 과정이다.

6) Community evolution

유비쿼터스 컴퓨팅 환경의 중요한 변화, 즉, 물리적인 환경의 변화나 구성 요소의 이동에 따라 추구하는 세부적인 goal이 변경될 수있다. 그런데, 그 goal이 전혀 다른 방향으로 변경되는 것이 아니라 보다 intelligent하고 adaptive하게 변경되는 것이다. 이 경우, 새 커뮤니티를 만드는 것이 아니라 기존의 커뮤니티를 기반으로 하여 진보된 형태의 커뮤니티를 구성하도록 한다.

5. Community의 발전 방향

초기의 커뮤니티는 유비쿼터스 컴퓨팅 환경 개발자에 의해 static하게 구성될 것이다. 목표로 하는 service에 대한 계획이 마련되어 있고, 해당 environment 내에 어떤 장치 혹은 서비스들이 있는지 알려진 상태이므로 이러한 정보를 기반으로 필요한 커뮤니티의 목표와member에 대한 spec, procedure 등에 대해 정의할 수 있다.

이러한 static configuration에 의해 커뮤니티가 동작하는 것은 단기적으로 커뮤니티를 정착시키는 시기에 적용되는 것이며, 그 다음 단계로는 manual하게 정의되는 goal 내에서 dynamic하게 goal을 달성하기 위한 community가 정의되고 필요에 따라 activate되고 deactivate될 수 있도록 발전되어야 한다. 이러한 dynamic community들은 static community로부터 시작하여 context-awareness를 위한 요소 기술들이 안정화되고 정착되면서 더불어 가능해질 것이다.

궁극적으로는, 유비쿼터스 컴퓨팅 환경 내의 autonomic computing 역량이 더욱 강화되면서 goal이 동적으로 정의되고, goal로부터 필요한 커뮤니티와 member들이 추론되어 정의될 수 있는 단계로까지 발전하게 될 것이다.

Ⅲ. 'Community-based computing' 실현을 위한 technical issue

1. Community에 대한 information management

1) Community 구성을 위한 goal 및 triggering event specification

커뮤니티는 '공통의 목표'를 갖고 협력한다고 했는데, 여기서 '공통의 목표'는 도달해야 하는 상태에 대한 specification이며 동시에 언제 이러한 공통의 목표를 달성하기 위해 협력해야 하는가 대한 specification도 함

께 갖고 있어야 한다. 즉, triggering event에 대한 specification도 갖고 있어야 한다. 그런데, 유비쿼터스 컴퓨팅 환경에서 발생할 수 있는 구체적인 event에 대한 description이라기보다, ontology 차원에서의 abstract specification이 될 수 있을 것이다. 그래서, 전체 커뮤니티를 coordinate하는 entity가 개별적인 상황에서 abstract specification과의 match를 파악할 수 있는 식의 specification이 되어야 할 것이다. Triggering event는 커뮤니티 자체가 만들어지기 위해 필요한 event가 아니고, 커뮤니티 내에 속한 member들의 action을 initiate하는 event이다.

2) Goal을 수행하기 위한 community의 member requirement specification

Goal의 specification이 만들어지고 나면, 해당 goal을 담당하기 위해 community에 참여해야 할 member들에 대한 specification이 정의되어야 한다. 유비쿼터스 컴퓨팅 환경 내에 새로운 장치나 서비스들이 추가되고 이동하고 할 것이고, 해당 goal을 이루기 위한 활동이 요구되는 환경이 특정한 곳으로 제한되지 않을 것이므로, 해당 환경에서 적절한 것을 찾아 대응시켜서 커뮤니티의 member로서 참여시킬 수 있도록 준비되어야 할 것이다. 즉, 구체적인 instance를 찾아서 대응시킬 수 있도록 member가 갖추어야 할 기능 요구 사항 및 resource 요구 사항 등을 기술하는 식의 abstract한 specification이 되어야 한다.

이 specification은 한 번 만들어진 이후로 상황의 지속적인 변화에 따른 update가 필요할 수 있다. 이러한 specification 자체에 대한 update를 어떤 기준 하에 할 것인지는 유비쿼터스 컴퓨팅 환경을 운영하는 strategy에 따라 달라질 수 있을 것이다.

커뮤니티에 따라서는 member들 각각이 해당 커뮤니티의 triggering event를 detect하는 경우 바로 community를 trigger해서 커뮤니티가 필요한 task를 수행하도록 하는 것을 허용할 수 있다. 이 경우에는, 커뮤니티의 goal과 member에 대한 specification을 community 내의 member들이 안전하게 access할 수 있도록 준비되어 있어야 한다. 이러한 정보는, member에 해당하는 service instance 혹은 resource instance가 커뮤니티의 member로서 등록되는 과정에서 주어질 수 있다. 자세한 명세가 주어지기보다 URI와 같은 형식으로 information의 location 정보가 주어지게 될 것이다.

3) Goal을 완수하기 위한 procedural steps

커뮤니티 내의 member들에 대한 요구 사항이 준비되면, 해당 member들이 goal을 완수하기 위해 어떤 식으로 협력을 해야 하는가에 대한 specification이 필요하다. 이 specification은 커뮤니티가 처음 만들어질 때는 커뮤니티를 정의하는 방법과 마찬가지로 만들어질 것이다. 즉, 커뮤니티 자체가 개발자 혹은 사용자에 의해 manual하게 정의될 때는 그 procedure 자체도 manual하게 정의될 것이고, dynamic하게 정의될 때는 이후 만들어지는 발전된 방법론에 의해 그 procedure 역시 dynamic하게 정의될 수 있을 것이다.

Procedure는 한 번 정의된 이후로 고정되는 것이 아니고 상황에 따라 변형해서 운영

하는 것이 가능해야 한다. 유비쿼터스 컴퓨팅 환경이 dynamic하게 변화할 수 있으므로 이에 적절하게 대응할 수 있어야 하기 때문이다. 그러나, 이러한 specification의 변형에 대한 판단은 manual하게 이루어질 수 있을 것이고, 학습에 의해 어떻게 조치하는 것이 적당할 지의 판단에 의해 dynamic하게 이루어질 수 있다. 그런데, 이렇게 상황에 따라 변형된 경우를 기본 틀에 바로 적용하는 것이 바람직한 지 아니면, 다른 평가 절차를 통해 갱신하는 것이 좋을지는 유비쿼터스 컴퓨팅 환경을 운영하는 strategy에 따라 달라질 것이다.

4) Community monitoring

커뮤니티의 member들이 activate되어서 자신에게 할당된 역할 및 지정된 procedure에 따라 동작을 할 것으로 기대하지만, 경우에 따라서는 malfunction하는 member도 있을 것이고, 예상치 못한 중간 결과가 발생하여 예외 상황에 처하게 되는 경우도 있을 것이다. 이런 경우, parallel하게 여러 entity가 community의 member로서 run하고 있는데 이들을 monitoring하여 상황에 따라 적절하게 coordination을 담당할수 있는 special entity가 필요하다.

Monitoring 기능에는 community에서 추구하는 목표가 달성되어서 더 이상의 action이 필요가 없게 되었을 때 해당 community의 member들에게 공지하는 기능도 함께 포함된다. Goal의 최종 상태에 해당하는 역할을 담당하는 member가 나머지 member들에게 goal의 종료를 announce하는 방법도 가능하다. 그러나, 커뮤니티 자체의 활동이 아니라 다른 환경의 변화에 따라서 goal이 종료될 수도 있는 것이므로 이에 대한 판단을 담당하는 entity가 필요하다.

커뮤니티의 규모에 따라서는 member들이 상호 작동 상태를 monitoring할 수도 있겠지만, 그 규모가 커지고 참여하는 member의 수가 커지게 되면 이러한 분산된 monitoring의 정확도가 떨어질 수 있다. 물론 하나의 entity가 담당한다면 overhead가 집중되는 부담이 있다. 이러한 monitoring entity를 분산시켜서 위치시키고 이들 간의 협력을 추구해서 overhead의 부담을 줄이고 정확도를 높이는 효과를 얻을 수 있을 것이다.

5) Conflict resolution strategy와 mechanism

동시에 run하는 커뮤니티가 추구하는 goal의 conflict가 발생할 수 있다. 이 경우, service-level community들 간의 우선순위를 결정할 수 있는 방법론이나 strategy가 마련되어 있어서 이러한 판단을 근거로 우선순위가 있는 service level community가 그 goal을 달성하도록 할 수 있어야 할 것이다.

동시에 여러 service-level community가 하나의 infrastructure-level community로의 요청이 발생하는 경우에도 conflict라고 간주할 수 있다. 이러한 경우에 있어서도, service level 커뮤니티 사이에 우선순위를 판단할 수 있는 근거가 있다면 resource level community가 우선순위 높은 service level community의 요청을 먼저 처리하도록 할 수 있다. 경우에 따라서는, alternative가 가능하다면 동일한 커뮤니티를 중복해서 만들어서 run할 수 있다.

2. Application 개발과 resource allocation 을 위한 framework

유비쿼터스 컴퓨팅 환경에서 사용될 서비스 애플리케이션 및 device 구동을 위한 software 등이 커뮤니티를 인식하여 동작할 수 있도록 설계되고 개발되어야 할 것이다. 이를 위해서 application framework이 제시되고 이를 기반으로 하여 개발되는 service application들은 다양한 커뮤니티에 참여하며 타 application service 등과 협력할 수 있게 될 것이다.

Infrastructure-level의 community와 service level의 community 간의 합리적인 mapping을 지원하기 위해서는 resource를 manage하는 framework이 함께 개발되어야 한다.

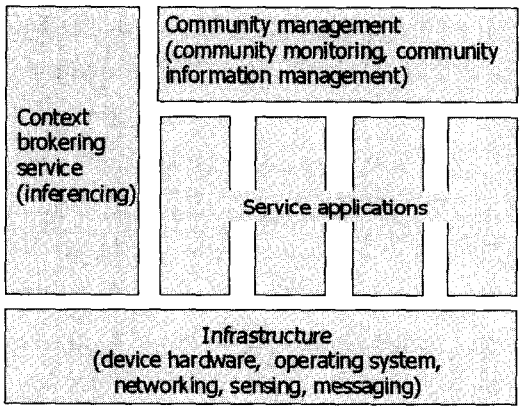
3. Interaction infrastructure

동일한 커뮤니티 내에 속한 entity들 간에 data를 교환하기 위한 infrastructure가 제공되어야 하고, 또한 이 infrastructure는 서로 다른 커뮤니티 내에 속한 entity들 간에도 data를 교환하기 위한 기능을 함께 담당해야 할 것이다. 그리고, 이 infrastructure를 사용하여 커뮤니티 monitor와 community member들 간의 interaction도 함께 이루어지는 것이 전체 유비쿼터스 컴퓨팅 환경을 유지하고 운영하는데 있어서의 부담을 최소화할 수 있을 것이다.

본 연구에서는, 다양한 entity들을 효율적으로 수용할 수 있는 interaction infrastructure로 message 기반의 interaction infrastructure를 활용하고자 한다. 이message 기반의 방법론은 일대일 상호작용뿐만 아니라 다수의 수신자와 송신자를 지원하는 pub/sub 방식의 상호 작용도 지원하고, 상호 작용에 필요한 메시지의 형식이나 종류를 infrastructure의 관점에서 동적으로 추가하고 삭제할 수 기능을 제공한다.

IV. Community-based ubiquitous computing environment

1. Community-based system architecture



〈그림 1〉 *Basic system architecture for community-based ubiquitous computing environment*

그림 1은 community라는 collaboration metaphor를 가지고 유비쿼터스 컴퓨팅 환경을 구축할 때 구성 요소들을 분류하고 그들 간의 관계를 보이고 있다. 가장 바탕에는 전체 유비쿼터스 컴퓨팅 환경을 물리적으로 구성하고 연동하는 기능을 제공하는 infrastructure가 위치한다. 이 infrastructure는 주로 컴퓨팅 혹은 네트워킹 자원을 관리하고 이를 할당하는 기능을 담당하는 것뿐만 아니라 커뮤니티를

구성하는 애플리케이션과 기타 구성 요소들 간의 상호작용을 지원하는 messaging 기능을 포함한다. 그리고, 이 infrastructure의 구성 요소들이 앞서 언급한 infrastructure level의 community의 manage 대상이 된다.

Community computing infrastructure 상에서 다양한 서비스 애플리케이션들이 동작한다. 이들은 독자적으로 동작하기도 하고 community management 기능을 담당하는 entity들에 의해 community에 참여할 수 있다. 이러한 서비스들은 context brokering service에 의해 주어진 상황을 인식하고 대응하게 된다. 이 context brokering service는 다양한 상황 정보를 수집하기 위해 infrastructure에서 sensor들을 활용할 것이고, 이 정보들을 기반으로 context를 추론하는 inference 기능들을 포함할 것이다. 그런데, 이 때 context를 인식하고 파악하는 서비스도 커뮤니티라는 metaphor를 기반으로 동작할 수 있다. 즉, context 기반 정보를 수집하는 sensor들 사이의 협력이 필요할 것이며, context inference도 단순히 하나의 inference engine에 의해서 완료되는 것이 아니라 종합된 판단을 위해서는 다양한 inference engine들의 협력이 필요하기 때문이다.

이러한 서비스들을 총괄적으로 커뮤니티 metaphor를 기반으로 효과적으로 동작하도록 총괄하는 community management entity가 존재한다. 커뮤니티가 혹은 커뮤니티 내의 member들이 정상적으로 동작하는 지를 monitoring하고 community의 정보를 관리하는 기능을 담당한다. Community life model을 실제 realize하는 역할을 담당하는 entity이다. 이 entity는 독립된 개체의 형태로 존재할 수도 있을 것이고, 필요하다면 서비스 애플리케이션이나 infrastructure에 embedding될 수 있다. 그리고, 그림 1에서는 community management와 infrastructure 사이에 직접적인 관계가 표현되지 못하고 있지만, infrastructure level의 커뮤니티를 운용하기 위해 community management function은 infrastructure에 속하는 entity들에 대해서도 관리 기능을 적용한다.

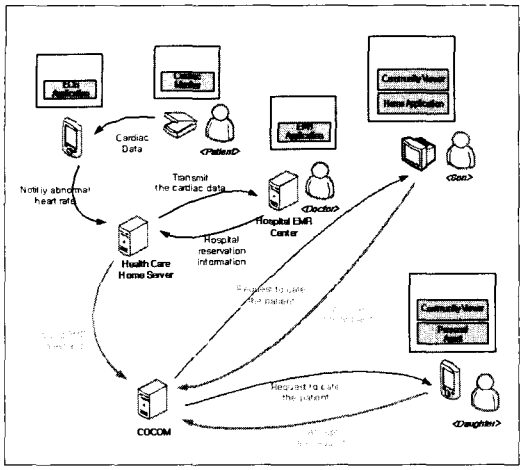
2. Example Application

커뮤니티의 개념을 적용한 예제 시나리오를 구현해 보았다. 시나리오는 health-care service community가 생성되어 응급한 상황이 발생 하였을 때 어떤 식으로 환자를 돌보는 가를 보여주고 있다.

평소 심장 질환을 앓고 있는 환자는 몸에 cardiac monitoring device를 부착하고 있다. 환자의 심장 박동수가 비정상적으로 변하였을 때, cardiac 이 수집한 데이터는 그의 주치의에게 자동으로 전달되고 주치의는 데이터를 분석하여 필요할 경우 환자를 위한 병실을 예약하고 환자의 가족들에게 연락을 취하게 된다.

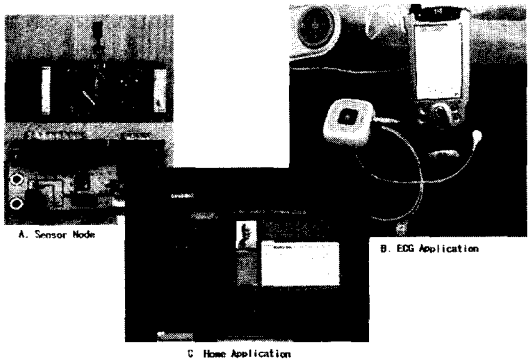
그림 2는 health-care service 커뮤니티의 구성 예를 보여 준다. health-care service community의 멤버는 cardiac monitoring device와 그것으로부터 데이터를 받아 보여주는 PDA 상의 application 그리고, health-care home server, hospital emergency center, 주치의를 위한 원격 분석 application, 가족의 구성원을 나타내는 personal agent로 구성되어 있다.

그림 3은 이 시나리오에서 사용된 device들을 나타낸다. 해당 device들을 자체 구축



〈그림 2〉 *Proactive Health-Care Community*

한 test-bed에 설치하였고 커뮤니티의 개념을 적용하여 시나리오를 구현하였다.



〈그림 3〉 *Sample Community Entities*

V. 기존 연구와의 비교

University of Texas, Arlington의 PICO project나 Arizona State University의 RCSM project에서는 본 연구에서 고려하는 'Community' 혹은 'group'에 대한 연구를 진행하고 있다. PICO project에서는 여러 개의 service나 application이 'Community'를 구성해서 공통의 goal을 달성하기 위해 협력하는 model을 제시하고 있다. 이 community의 building block으로 camileun과 delegent를 제시한다. Camileun은 computing device들이 나타내며 delegent가 run할 수 있는 장치들이다. Delegent는 community goal을 달성하기 위해 수행해야 하는 service들로서 여러 camileun을 이동해 다니면서 동작하는 것을 가정한다. 그리고, Community를 구성하는 범주를 service level과 network resource까지 다양하게 고려하고 있으나, 실제 연구에서 주력하는 것은 service level에서의 커뮤니티이고, community-based computing 환경을 위한 middleware 개발에 주력한다. 본 연구와 PICO project의 가장 큰 차이점은, service application들의 이동성에 대한 관점에 있다. 본 연구에서는 각 device들이 서비스 애플리케이션을 갖고 있고, 이 서비스 애플리케이션이 integration platform에서 제공하는 messaging infrastructure를 사용하여 상호 작용하는 것을 기본 모델로 한다. 즉, 서비스 애플리케이션이 navigation하는 것이 아니고, 각 서비스 애플리케이션의 자발적이고 독립적인 interaction에 의해 커뮤니티의 goal을 달성하기 위한 task들을 수행하게 되는 것이다.

RCSM project에서는 실제 일상생활에서 social 그룹으로 활동하는 것을 model하기 위한 방법론으로 'group'이라는 개념을 도입하였다. 기존의 그룹 커뮤티케이션 모델을 활용하여 유비쿼터스 컴퓨팅 환경에서 사용자의 context에 따라 device들이 동적으로 그룹을 구성하고 운영하는가 하는 방안을 개발하고자 한다. RCSM은 일반 사회에서의 'Community' 혹은 'group'이라고 칭할 수 있

는 collaboration을 유비쿼터스 컴퓨팅 환경에서 구현하기 위한 것으로 서비스 애플리케이션에 의존적이다. 본 연구와 RCSM의 가장 큰 차이점은, 'Community'의 범주에 대한 관점이다. 본 연구에서 고려하는 커뮤니티는 일상 생활에서의 '사회' 라는 metaphor를 유비쿼터스 컴퓨팅 환경의 다양한 서비스에 mapping시키는 것으로 RCSM의 'group'이라는 개념이 적용되는 사례보다 더 광범위한 경우에 적용하고자 한다.

VI. 결론

본 연구를 통해 복합적인 유비쿼터스 환경을 구축하기 위한 방안으로 커뮤니티 기반의 컴퓨팅 환경을 제시하였다. 구체적으로는 커뮤니티라는 메타포가 갖는 포괄적인 개념들을 언급하면서 구현을 위한 요구사항과 기술 이슈들을 정립하였고, 커뮤니티의 클래스, 라이프 모델과 시스템 아키텍처를 도출하였다.

이로써 기존의 유비쿼터스 컴퓨팅 환경의 연구가 개별상황에 따른 대응이나 단순한 형태의 디바이스와 서비스들의 협력에 그침으로써, 보다 복합적이고 자율적인 서비스를 개발하는 데 지니고 있던 제약을 극복할 수 있는 기반이 마련되었다고 하겠다.

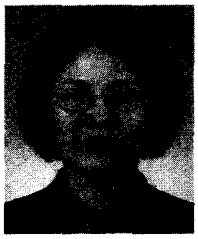
향후 본 연구에서 제기된 기술 이슈들의 이론적 체계를 마련하고 단계적으로 구현함으로써 보다 상세한 논문으로 발전될 것으로 기대한다.

참고 문헌

[1] Mohan Kumar, et al., "PICO: A Middleware Framework for Pervasive Computing," IEEE Pervasive Computing, July/September 2003.

[2] Bin Wang, John Bodily, and Sandeep K. S. Gupta, "Supporting Persistent Social Groups in Ubiquitous Computing Environments Using Context-Aware Ephemeral Group Service", Proc. 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom), Orlando, FL, March 2004, pp. 287-296

저자소개



강 경 란

1992년 2월 서울대학교 공학사
994년 2월 한국과학기술원 공학 석사
1999년 2월 한국과학기술원 공학 박사
주관심 분야 Network, Community Computing

이 정 태

1988년 4월 – 현재 아주대학교 정보통신대학 부교수
1983년 9월 – 1988년 3월 아주대학교 정보통신대학 전임강사
1981년 3월 – 1983년 9월 울산대학교 전산학과 전임강사
주관심 분야 컴포넌트 베이스 시스템, 미들웨어, 객체지향 응용 프레임워

조 위 덕

2004년 4월 – 2005년 현재 재단법인 유비쿼터스컴퓨팅사업단 사업단장
1991년 11월 – 2004년 3월 전자부품연구원 (2003 시스템연구본부 본부장)
1995년 2월 – 1995년 10월 영국 TTP/Cambridge GSM Division 공동개발연구원
주관심 분야 유비쿼터스 컴퓨팅