이동에이전트 기반 워크플로우 시스템의 설계

서 영 호[†]·유 정 준[††]·이 동 익[†††]

요 약

본 논문에서는 워크플로우 시스템이 갖추어야 할 다양한 구조적 요구 사항들 중에서 성능 및 확장성 이슈에 중점을 두어, 기존의 두 접근법들이 갖는 한계를 극복할 수 있는 새로운 구현전략을 제안한다. 새로운 접근법은 성능 및 확장성 측면에 중점을 둔 이동에이전트 기반 아키텍쳐를 마련한 후에, 이동에이전트가 갖는 본질적인 특성을 이용한 상위수준의 전략인 '위임 모델'을 도입하는 것이다. 여기서는, 이러한 접근법에서 고려해야 할 세 가지 주요 문제점들에 대한 해결방안을 수립하여 이들을 효과적으로 반영한 이동에이전트 기반 3-계층 워크플로우 시스템 아키텍쳐를 제안한다. 또한 제안된 아키텍쳐의 성능 및 확장성을 GSPN 시뮬레이션을 통해 기존의 두 접근법과 비교 분석함으로써, 제안한 방법이 워크플로우 시스템의 성능 및 확장성을 보장하기 위한 최적의 방법임을 보인다.

Design of a Mobile Agent-based Workflow System

Young-Ho Suh[†] · Jeong-Joon Yoo[††] · Dong-Ik Lee[†††]

ABSTRACT

This paper proposes a new design strategy that can overcome limitations of two existing approaches, focusing on performance and scalability issues among various architectural issues which must be considered in designing workflow systems. In the proposed approach, we suggest to introduce 'delegation model' which corresponds to the high-level strategy utilizing the fundamental characteristics of mobile agents, after establishing a mobile agent-based workflow system architecture focused on performance and scalability. We point out three major issues that should be considered in this approach and propose a 3-tier mobile agent-based workflow system architecture that effectively reflects these considerations. Also, we show that the new approach can provide better performance and scalability than existing ones - approaches based on the client-server paradigm and other approaches based on mobile agent paradigm represented by DartFlow system - by evaluating performance of the proposed architecture through the GSPN simulation.

1. 서 론

워크플로우 시스템이란, 하나 또는 그 이상의 워크플로우 엔진을 통해 워크플로우의 수행을 관리 및 감독할 뿐만 아니라, 워크플로우를 정의하고 생성할 수 있는 시스템을 말한다[1]. 이러한 워크플로우 시스템에 대한 개념은 1980년대 중반부터 사용되어왔으며, 오늘날 기업 내에서 조직구조와 조직내의 업무가 다양화, 복잡화되어 워크플로우 시스템에 대한 요구가 더욱 증가하고 있다. 즉, 기업 조직을 관리하는 관리자의 입장에서 보면 워크플로우 시스템을 도입함으로써 기업 내에 존재하는 개별적 비즈니스 프로세스들을 리엔지니어링하여 기업 업무흐름의 효율화를 꾀할 수 있고, 기업조직이나 개별업무 등의 변화에 따른 비즈니스 프로세스의 변경에 유연하게 대처할 수 있을 뿐만 아니라, 다양하고 복잡한 비즈니스 프로세스들의 수행을 통합하여 관리·감독할 수 있게 되어, 기업 내에 산재해있

† 정 회 원 : 한국전자통신연구원 인터넷서비스연구부 연구원
†† 준 회 원 : 광주과학기술원 대학원
††† 종신회원 : 광주과학기술원 정보통신학과 교수
논문접수 : 2000년 2월 15일, 심사완료 : 2000년 7월 10일

던 정보들이 상위 수준에서 체계적으로 통합 관리됨으로써 빠르고 정확한 의사결정을 가능하게 한다. 한편, 개별 단위업무를 수행하는 수행자의 입장에서 보면, 워크플로우 시스템이 수행자에게 업무흐름에 대한 투명성을 제공함으로써 단위업무 수행자는 단지 주어진 업무의 수행에만 집중할 수 있을 뿐만 아니라, 단위업무의 수행을 위해 필요한 응용프로그램들이 기반 IT시스템에 고정되어있지 않기 때문에 응용프로그램의 추가, 삭제 및 변경 등이 자유로워 유연한 업무 수행환경을 제공하게 된다.

워크플로우 시스템이 갖는 위와 같은 중요성이 인식되어 이미 많은 상용 시스템이 개발되었으나, 실제 기업 환경에 도입되어 성공적으로 사용되기 위해 갖추어야 할 다양한 요구 사항들을 충족시키지 못하고 있다[2]. 특히, 다음과 같은 구조적 이슈(architectural issue)들에 대한 고려가 취약하여 이로 인한 많은 한계점을 갖는다[3].

- 적응성(Adaptability to various enterprise structure)
  워크플로우 시스템은 적용되는 조직의 구조와 크기의 변화에 쉽게 순응할 수 있어야 한다.

- 성능(Performance)
  워크플로우 프로세스의 전체 혹은 많은 부분이 자동업무들(automatic tasks)로만 구성될 수도 있다. 이런 경우는 워크플로우 엔진이 프로세스 수행에 있어서 주된 지연의 원인이 된다. 워크플로우 시스템은 자동업무를 수행하는데 있어 성능을 향상시킬 수 있는 메커니즘을 제공해야 한다[4].

- 확장성(Scalability)
  워크플로우 시스템은 수시로 변동하는 많은 수의 비즈니스 프로세스를 처리할 수 있어야 한다. 즉, 수십 혹은 수천 개의 워크플로우 인스턴스들이 평균 성능의 큰 손실 없이 병행적으로 수행될 수 있어야 하며, 사용자나 워크플로우 인스턴스 수의 빈번하고 심한 변동에 순응할 수 있어야 한다.

- 신뢰성 및 가용성(reliability and availability)
  워크플로우 시스템은 신뢰성과 가용성이 있어야 한다. 즉, 워크플로우 시스템은 실패(failure)를 효과적으로 처리하여 전체 기업 업무를 마비시키는 일이 없어야 한다.

따라서 현재 워크플로우 시스템이 갖는 위와 같은 다양한 구조적인 이슈들을 효과적으로 해결하기 위한 연구가 활발히 진행되고 있으며, 그 접근법에 따라 크게 클라이언트-서버 패러다임에 기반한 아키텍처수준의 전략과 이동에이전트 패러다임에 기반한 상위수준의 전략으로 구분해 볼 수 있다.

클라이언트-서버 패러다임에 기반한 아키텍처 수준의 전략은 각각의 이슈들에 대해서 개별적으로 아키텍처 수준에서 보장하려는 방법으로, '분산된 워크플로우 엔진의 구현'[5]을 통한 성능 및 확장성 보장 전략이나 '트랜잭션 워크플로우 개념의 도입'[6]을 통한 신뢰성 제고 전략 등이 여기에 해당한다. 이 접근법은 개별적인 구조적 이슈에 대해 아키텍처 수준에서의 근본적인 해결책을 제공할 수 있다는 장점이 있지만, 다양한 구조적 이슈들을 갖는 워크플로우 시스템의 특성상 전체 아키텍처가 매우 복잡해질 수 있을 뿐만 아니라, 각각의 이슈들에 대해 상충되는 부분이 생기면 트레이드오프가 발생하게 되어 개별 전략들이 훼손되거나 변질되어 원래의 의도를 충족시키지 못하게 될 수도 있다.

한편, 워크플로우 시스템의 기반구조에 이동에이전트 패러다임을 도입하여, 보다 체계적이고 종합적인 해결방안을 모색하려는 시도가 있다. 이러한 시도는 본질적으로 통합성과 유연성이 우수한 이동에이전트 시스템을 기반으로 워크플로우 시스템을 구현함으로써 워크플로우 시스템이 갖추어야 할 통합성과 유연성 그리고 적응성을 수월하게 보장할 수 있을 뿐만 아니라, 이동에이전트가 갖는 본질적인 특성들을 워크플로우 시스템 수행 수준에서 향유할 수 있게 되어 이들을 이용한 상위 수준의 구현전략이 가능하다는 발상에 근거한다. 실제로 이동에이전트가 갖는 자율성, 이동성은 전체 시스템의 제어 및 부하의 자연스런 분산을 통한 '성능 및 확장성'의 향상을 가져올 수 있고, 이동에이전트의 비동기적인 수행특성과 능동성은 네트워크 오류의 발생 가능성을 줄일 수 있을 뿐만 아니라 오류 발생시 대처능력을 향상시켜 '신뢰성'의 향상을 가져올 수 있다[7]. 이 접근법은 클라이언트-서버 패러다임에 기반한 아키텍처수준의 전략에 비해 보다 간결하고 체계적인 시스템 구현을 가능하게 하는 장점이 있지만, 개별적인 이슈들에 대해 충분한 해결책을 제공하는데 한계가 있다. 왜냐하면 이동에이전트 패러다임이 갖는 잠재적 장점들이 실제 시스템에 발현되는 것은 이동에이전트 시스템 구조를 포함한 전체 시스템 구조에 좌

우될 뿐만 아니라, '분산엔진 구조'나 '트랜잭션 워크플로우' 등과 같은 아키텍쳐수준의 고려 없이 이동에이전트가 갖는 본질적인 특성들만을 이용한 상위수준의 전략 만으론 적응성, 성능 및 확장성, 그리고 신뢰성 등의 개별적 구조적 이슈들을 충분히 만족시킬 수는 없기 때문이다.

따라서, 본 논문에서는 위의 두 가지 접근법이 서로 다른 수준의 전략이면서 또한 각각의 장단점들이 상호 보완적임을 감안하여, 이들을 중첩한 형태를 갖는 새로운 접근법으로 '높은 우선순위를 갖는 구조적 이슈들에 중점을 둔 이동에이전트 패러다임 기반 워크플로우 시스템 아키텍쳐에서의 이동에이전트의 특성을 이용한 상위수준의 구현전략'을 제안하고, 성능 및 확장성을 높은 우선순위를 갖는 이슈로 하여 제안한 접근법의 장점을 정당화하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 워크플로우 시스템의 성능 및 확장성 이슈와 관련된 연구로 두 가지의 접근법에 대해 알아보고, 각각의 장단점을 지적한다. 3장에서는 2장에서 제기된 기존의 접근법들이 갖는 한계를 극복할 수 있는 새로운 방법을 제안하고, 새로운 접근법에서 고려되어야 할 세 가지 주요문제에 대해 해결책들을 수립하여, 이들이 효과적으로 반영된 이동에이전트 기반 워크플로우 시스템 아키텍쳐를 설계한다. 4장에서는 3장에서 제안한 해결책들 중 정량분석을 통한 정당화가 필요한 '위임모델'을 GSPN 시뮬레이션을 통해 성능평가 한다. 5장에서는 GSPN 시뮬레이션을 통해 새로운 접근법과 기존의 두 가지 접근법의 성능 및 확장성을 비교하여 그 결과를 제시한다. 끝으로, 6장에서 결론을 맺는다

2. 관련 연구

이 장에서는 워크플로우 시스템의 성능 및 확장성에 관한 구조적 이슈에 대해 기술하고 이를 해결하기 위해 지금까지 진행되었던 관련 연구를 클라이언트-서버 패러다임에 기반한 아키텍쳐수준의 접근법과 이동에이전트 패러다임을 통한 상위수준의 접근법으로 나누어 고찰한다.

2.1 워크플로우 시스템의 성능 및 확장성 이슈

워크플로우 시스템은 성능, 확장성 혹은 신뢰성과 같은 개념보다는 주로 라우팅(routing), 공유(sharing) 그리고 협력(coorperation)에 관한 이슈들에 중점을 두었던 사무 자동화(office automation), 영상 처리(image processing) 혹은 컴퓨터 기반 협동 작업(CSCW) 시스템등에 그 기원을 두고 있다. 이런 연유로 기존의 상용 워크플로우 시스템들에도 이러한 특성이 그대로 계승되어 성능 및 확장성에 대한 고려가 매우 취약하다[8]. 그러나 기업 규모가 방대해지면서 조직구조나 조직내의 업무가 다양화 전문화됨을 고려할 때, 워크플로우 시스템이 성공적으로 거대기업 환경에 적용될 수 있기 위해서는, 수 백에서 수 천, 심지어 수 만에까지 이르는 사용자나 프로세스들을 동시에 수용할 수 있어야 한다.

2.2 클라이언트-서버 패러다임에 기반한 아키텍쳐 수준의 접근법

기존의 워크플로우 시스템들은 한 개의 워크플로우 서버를 가진 중앙 집중형 아키텍쳐를 채택하고 있다. 이와 같은 중앙 집중형 아키텍쳐를 기반으로 한 시스템에서는 워크플로우 서버에 발생할 수 밖에 없는 병목현상이 시스템의 성능 및 확장성을 저해하는 원인이 된다. 따라서, 이 접근법은 단일 워크플로우 서버로 이루어진 중앙 집중형 아키텍쳐가 갖는 한계를 아키텍쳐의 개선으로 극복하려는 것으로, 다수의 서버를 가진 중앙 집중형 아키텍쳐를 통한 접근과 서버를 전혀 갖지않는 완전한 분산 아키텍쳐를 통한 접근에 관한 연구가 있다[9-13].

2.2.1 다수의 서버를 가진 중앙 집중형 아키텍쳐

중앙 집중형 아키텍쳐를 유지하면서 두 개 이상의 워크플로우 서버가 서로 협력하여 작동하게 함으로써 단일한 워크플로우 서버가 갖는 확장성의 문제를 해결하고자 하는 접근법이다. FlowMark[9]와 COSA[10] 등의 최근 버전의 아키텍쳐에서 이 방법을 제안했지만, 워크플로우 서버끼리 어떻게 협동하는 지에 대한 문제를 남겨두고 있다. Alonso et al.[11]은 클러스터를 도입하여 워크플로우 서버간의 협동 문제를 해결할 것을 제안하였다. 한 개의 클러스터는 공통의 데이터베이스를 이용하여 같은 워크플로우들을 수행하는 다수의 워크플로우 서버들로 구성된다. 이에 따라 부하가 이들 서버들 간에 분산될 수 있으나 공통의 데이터베이스가 여전히 잠재적 병목점으로 존재하게 되고, 더구나 특정한 기업 환경에서 어떻게 클러스터를 구성할 것인지

에 대한 해결책도 제시하지 못했다. 결국 구현상의 차이는 있지만 이러한 접근법에서는 부하가 증가하면 부가적인 워크플로우 서버 인스턴스를 추가로 생성시키는 방법으로 확장성을 제공하려는 것이다. 그러나 이 방법은 위에서 지적한 바와 같이 다수의 서버들간의 협동 메커니즘이 제공되어야 하고 이 과정에서 오버헤드가 발생하는 단점이 있다.

2.2.2 완전한 분산 아키텍처 아키텍쳐

어떠한 집중된 워크플로우 서버나 집중된 데이터베이스도 없는 완전히 분산된 아키텍쳐를 통해 중앙 집중형 아키텍쳐가 갖는 확장성의 문제를 본질적으로 해결하려는 접근법이다. IBM의 Exotica 프로젝트[12]에서는 시스템 내의 모든 노드가 자율적으로 동작하는 완전 분산 아키텍쳐를 제안하였다. 워크플로우 타입에 대한 스키마 정보는 모든 노드에 중복되어 있고, 인스턴스 정보는 각각의 노드에서 로컬하게 관리되면서 노드들 끼리 영구 큐(persistent queue)들을 통해 통신하는 아키텍쳐이다. 그리고 Miller et al.[13]은 CORBA 서비스를 통신 하부구조로 하는 완전 분산 아키텍쳐를 제안하기도 하였다. 이 방법에서는 구현상의 차이는 있지만 워크플로우 스키마 정보를 모든 노드에 중복시키고 실제 인스턴스는 각 노드에서 자율적으로 수행되면서, 각 노드들끼리 인스턴스 수행 정보를 메시징 시스템이나 CORBA등의 통신하부구조를 통해 교환하면서 프로세스를 수행하는 완전 분산 아키텍쳐를 구성함으로써 확장성의 문제를 해결하였다. 그러나 이 방법은 모든 프로세스 인스턴스들이 자율적인 노드에서 독립적으로 수행되므로, 프로세스 수행과정에 대한 집중된 조망(centralized view)이 존재하지 않기 때문에 워크플로우의 수행을 모니터링 할 수 없다는 심각한 단점이 있다[3].

2.3 이동에이전트 패러다임의 도입을 통한 상위수준의 접근법

이동에이전트가 갖는 본질적인 특성을 이용하면 워크플로우 프로세스를 워크플로우 서버와 비동기적으로 수행할 수 있어서, 시스템 내의 제어 및 부하가 워크플로우 서버에 집중되지 않고 자연스럽게 전체 시스템에 분산될 수 있다. 따라서, 이 접근법은 중앙 집중형 아키텍쳐가 갖는 한계점을 이동에이전트 패러다임이 성능 및 확장성 측면에서 갖는 잠재적 장점들을 통해 극복하려는 것으로 DartFlow 시스템[14]이 그 대표적인 예이다.

2.3.1 이동에이전트의 특성

이동에이전트란 네트워크 내에서 이동하면서 사용자나 다른 개체를 대신하여 임무를 자율적으로 수행할 수 있는 컴퓨터 프로그램을 말하며 기능을 자체포함하고 식별 가능해야 한다. 이러한 이동에이전트에 대한 정의는 다음과 같은 이동에이전트의 본질적인 특성에 기반하고 있다[15] :

- 위임성(delegation)
  사용자나 다른 프로그램들은 과업들을 에이전트에게 위임하고 그들을 대신해서 수행할 권한을 부여할 있다.
- 자율성(autonomy)
  에이전트는 소유자의 목적(goals), 기호(preferences) 그리고 정책(policies)에 대한 기술에 근거하여 스스로 결정을 할 수 있다.
- 사회성(social ability)
  에이전트는 동배(peers), 환경(environment), 소유자(owners)와 통신을 통한 상호작용을 할 수 있다.
- 이동성(mobility)
  에이전트는 부여된 임무를 점진적으로 수행하기 위해서 이형의 컴퓨터 네트워크 상을 옮겨 다닐 수 있다.
- 유연성(flexibility)
  에이전트는 고정된 역할을 갖지 않고, 클라이언트, 서버, 관찰자 등의 역할을 필요에 따라 수행할 수 있다.

2.3.2 DartFlow 시스템[14]

DartFlow에서는 이동에이전트 패러다임을 도입함으로써 병행성(concurrency), 가용성(availability), 성능(performance), 확장성(scalability), 고장허용(fault-tolerance) 등의 트랜잭션 성질들을 대부분 처리하면서도, 프로세스의 동적변경이나 적응성 등의 다른 요구 사항들도 보장할 수 있다고 주장한다. 실제로 확장성 측면에서 보면, 프로세스 에이전트가 조직 서버(organization server)로부터 해당 워크플로우 프로세스의 업무 리스트(task list)를 넘겨받아 그 프로세스 인스턴스를

자율적으로 수행하는 방법으로 확장성을 제공하고 있다. 그러나 각각의 단위업무가 끝날 때마다 프로세스 에이전트는 작업리스트 서버(worklist server)에게 결과를 알리면 작업리스트 서버가 이에 따라 사용자 인터페이스의 정보를 갱신하게 되어 있어, 많은 프로세스 에이전트가 동시에 수행되게 되면 작업리스트 서버에 병목현상이 발생하게 된다. 또한, 하나의 이동에이전트가 하나의 프로세스의 수행을 전부 책임져야 하므로, 많은 수의 업무로 구성된 복잡한 프로세스인 경우에는 제어 및 이동에 따른 오버헤드로 인해 성능을 크게 떨어뜨린다. 그리고 이동에이전트 기반의 시스템에 있어서 성능 및 확장성과 밀접한 관련을 가지는 이동에이전트의 위치정보 관리 메커니즘에 대한 고려가 없다. 즉, 이동에이전트 패러다임이 성능 및 확장성의 측면에서 갖는 잠재적인 장점들이 극대화되어 시스템에 발현되려면, 이동에이전트 시스템 자체의 아키텍쳐는 물론이고 에이전트의 수행 구조를 적절히 최적화 시켜야 함에도 불구하고 이러한 고려가 되지 못했다.

3. 이동에이전트 기반 워크플로우 시스템 아키텍쳐

이 장에서는 기존의 두 접근법이 갖는 한계를 상호보완적으로 극복하여, 워크플로우 시스템이 갖는 다양한 구조적 이슈들을 효과적으로 해결할 수 있는 새로운 전략을 제안한다. 새로운 접근법은 높은 우선순위를 갖는 주요한 이슈들에 중점을 두어 이동에이전트 기반 워크플로우 시스템 아키텍쳐를 구성한 후에, 높은 우선순위를 갖는 이슈들을 포함한 다양한 구조적 이슈들에 대해 이동에이전트가 갖는 본질적인 특성들을 이용한 상위수준에서의 종합적인 구현전략을 수립하는 방법이다. 본 논문에서는 다수의 서버를 가진 중앙 집중형 아키텍쳐를 통한 아키텍쳐 수준의 전략과 이동에이전트 패러다임을 통한 상위수준의 전략을 중첩한 다수의 서버를 가진 이동에이전트 기반 중앙 집중형 아키텍쳐 상에서의 상위수준 구현 전략을 제안한다. 다수의 서버를 가진 이동에이전트 기반 중앙 집중형 아키텍쳐 상에서의 상위수준 구현전략이 갖는 장점을 극대화 시키기 위해 반드시 고려되어야 할 주요한 사항들은 크게 다음의 세 가지가 있다.

다수의 서버를 가진 중앙 집중형 아키텍쳐가 갖는 단점인 다수의 서버들 간의 협동에 필요한 오버헤드를 이동에이전트 패러다임의 도입을 통해 어떻게 최소화 할 것인가?

성능 및 확장성 측면에서 이동에이전트 패러다임이 갖는 잠재적 장점들이 다수의 서버를 가진 이동에이전트 기반 중앙 집중형 아키텍쳐 상에서 극대화되어 발현되기 위해서, 어떤 사항들이 이동에이전트 시스템 아키텍쳐 수준에서 고려되어야 하는가?

위의 두 가지 고려 사항들을 반영하여 설계된 전체 아키텍쳐 상에서의 성능 및 확장성을 향상시키기 위한 상위수준 전략은 무엇인가?

이 장에서는 이러한 세 가지 고려 사항들을 구체적인 문제점들로 명확히 하고, 각각에 대한 해결 전략을 수립하여 이들을 효과적으로 반영한 이동에이전트 기반 워크플로우 시스템 아키텍쳐를 제안한다.

3.1 설계 고려 사항 및 구현 전략

3.1.1 이동에이전트를 기반으로 한 다수의 서버 간의 협동

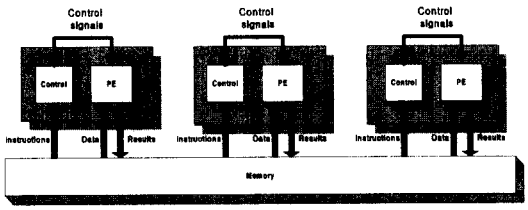
(1) 문제점

클라이언트-서버 패러다임에 기반한 다수의 서버를 가진 중앙 집중형 아키텍쳐가 갖는 단점으로 지적된 '다수의 서버들 간의 협동에 필요한 오버헤드'를 이동에이전트 패러다임을 통해 어떻게 최소화 할 것인가에 대한 문제이다.

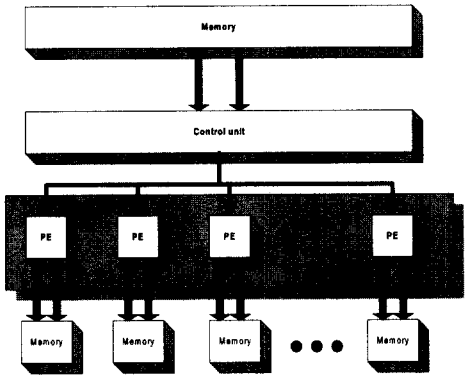
(2) 착안점

기존의 클라이언트-서버 패러다임에 기반한 다수의 서버 구조는 (그림 1)과 같은 MIMD병렬 구조와 유사하다. 즉, 워크플로우 서버가 갖는 세 가지 기능 중 프로세스 인스턴스의 생성 및 수행 관리를 담당하는 모듈을 제어부(control unit), 프로세스 인스턴스의 수행을 담당하는 모듈을 처리부(processing element)라 하고 또한 프로세스 인스턴스의 생성, 수행, 그리고 수행관리를 위해 필요한 모든 정보, 즉 워크플로우 스키마 정보나 인스턴스 정보 등이 저장되어있는 데이터베이스를 메모리부(memory system)라고 가정하면, 다수의 워크플로우 서버들은 공통의 데이터베이스를 가진 강결합 혹은 각자의 지역 데이터베이스를 둔 약결합을 통하여 동일한 메모리를 공유하면서 각자의 제어부와 처리부를 가지고 독립적으로 프로세스 인스턴스를 생성 및 수행, 관리하면서 전체적으로 마치 하나의 워크플로

우 서버처럼 동작한다. 따라서, 결과적으로 이러한 구조에서는 MIMD 병렬 구조에서의 프로세스 간 동기화 및 협동에 관한 문제들이나 공용 메모리에서의 병목현상이 비율적으로 확대되어 나타나며, 이를 다수의 워크플로우 서버들 간의 협동에 필요한 오버헤드로 지적한 바 있다.



(그림 1) MIMD 병렬 구조

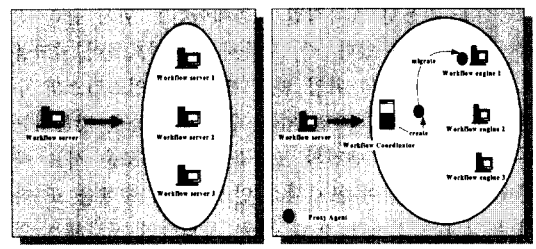


(그림 2) 프로세서 배열 구조를 갖는 SIMD 병렬 구조

그러나 이동에이전트 패러다임을 도입하여 다수의 서버를 구성할 경우에는 이동에이전트가 갖는 본질적인 특성인 '이동성', '사회성', '위임성', 그리고 '자율성' 등을 이용하여 (그림 2)와 같은 SIMD 병렬 구조와 유사한 2-계층 분산 워크플로우 서버 아키텍처를 고려할 수 있다. 즉, 프로세스 인스턴스의 생성 및 수행 관리를 담당하는 제어부가 별도로 분리되어 상위계층에 존재하게 되고, 이 제어부는 프로세스 인스턴스의 생성을 위해 필요한 워크플로우 스키마 정보 만을 관리하는 별도의 데이터베이스를 갖는다. 이와 같은 구조에서는 제어부가 이동에이전트를 생성하여 프로세스 인스턴스의 수행을 위임하게 되면, 생성된 이동에이전트가 하위계층인 처리부로 이동하여 인스턴스 수행정보를 로컬하게 관리하면서 자율적으로 프로세스 인스턴스를 수행하게 되고, 이러한 수행과정이 제어부에 의해서 원격으로 관리됨으로써 전체적으로 마치 하나의 워크플로우 서버처럼 동작한다. 따라서, 이러한 구조에서는 어떠한 공통 데이터베이스도 존재하지 않으므로 공통 데이터베이스에서의 병목현상이 발생하지 않게 되고, 워크플로우 서버간의 협동 또한 필요하지 않게 된다.

(3) 해결 방안

(그림 3)의 왼쪽과 같이 기존의 방법은 하나의 워크플로우 서버가 단순히 수평적으로 중복 확장되어 동일한 다수의 서버가 서로 협력하여 기능상 하나의 워크플로우 서버처럼 동작하는 것이었다. 앞의 착안점에 기반하여 (그림 3)의 오른쪽과 같이 이동에이전트를 이용한 분산 워크플로우 서버 아키텍처를 제안한다. 워크플로우 서버가 갖는 기능이 분리되어 상위계층에 해당하는 워크플로우 조정자가 워크플로우 프로세스의 생성 및 수행관리 기능을, 하위계층에 해당하는 워크플로우 엔진이 프록시 에이전트와 동적으로 결합하여 워크플로우 프로세스의 수행 기능을 하게 되고, 두 계층의 기능이 이동에이전트가 갖는 특성들을 이용해 동적으로 유연하게 결합되어 하나의 워크플로우 서버처럼 동작한다. 즉, '일회성'을 갖는 워크플로우 프로세스 생성의 기능은 이동에이전트의 '이동성'을 통해, 그리고 '간헐성(intermittence)'을 갖는 워크플로우 프로세스 수행 관리의 기능은 '사회성'을 통하여 하위계층과 결합된다. 따라서 프록시 에이전트가 수행을 위해 워크플로우 조정자로부터 워크플로우 엔진으로 이동하는데 소요되는 오버헤드와 프로세스 인스턴스의 수행을 관리하기 위해 워크플로우 조정자와 프록시 에이전트 사이에 필요한 원격 상호작용을 제외하고는 워크플로우 프로세스의 수행을 위한 다수의 워크플로우 엔진들 간의 협력은 필요없다. 설령 동적 부하 균형화(dynamic load-balancing)을 위해 프록시 에이전트가 다른 워크플로우 엔진으로 이동을 하더라도, 그때까지의 모든



(그림 3) 이동에이전트 기반 2-계층 분산 워크플로우 서버

수행정보를 가지고 이동하므로 이동에 소요되는 오버헤드 외에 엔진끼리의 협력은 필요하지 않다. 또한 프록시 에이전트에 의한 자율적 프로세스 수행으로부터 야기되는 프로세스 수행에 대한 집중된 조망(centralized view)의 부재 문제도 계층적으로 상위에 있는 워크플로우 조정자를 통해서 해결할 수 있다.

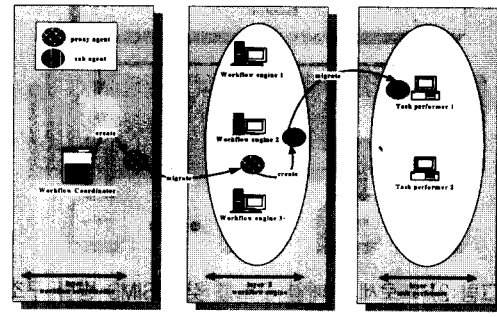
3.1.2 이동에이전트 시스템 아키텍쳐 수준의 고려

(1) 문제점

이동에이전트들이 갖는 '자율적인 이동성'을 보장하기 위해서는 이동에이전트들의 현재 위치를 관리하는 메커니즘을 반드시 제공해야 한다. 즉, 자율적으로 시스템 내의 여러 노드들을 이동하는 이동에이전트들이 서로 통신을 하거나 이동에이전트의 소유자가 자신이 생성한 이동에이전트를 원격으로 관리할 수 있게 하기 위해서는, 각 이동에이전트의 유일한 이름을 통해 그 에이전트의 현재 위치를 찾을 수 있어야 한다. 위치 독립적인 이동에이전트 위치 관리를 위해서는 이동에이전트의 이름과 현재 그 이동에이전트의 위치를 맵핑시켜 주는 네임 서비스(name service)가 필요하다. 그러나 워크플로우 시스템과 같이 많은 이동에이전트가 병행적으로 수행되어야 하는 경우에는 네임 서비스가 심각한 병목현상을 초래할 수 있기 때문에, 성능 및 확장성 측면에서 이동에이전트 패러다임이 갖는 잠재적 장점들이 극대화되어 발현되기 위해서는 효과적인 이동에이전트 위치관리 전략이 이동에이전트 시스템 수준에서 고려되어야 한다.

(2) 착안점

제안한 이동에이전트 기반 2-계층 분산 워크플로우 서버 아키텍쳐에서 프록시 에이전트들의 위치 관리 메커니즘이 요구되는데, (그림 4)와 같은 계층적 위임에 의한 프로세스 인스턴스 수행 구조를 도입하게 되면 계층성을 활용한 효과적인 이동에이전트 위치 관리 전략이 고려될 수 있다. 즉, 프록시 에이전트들이 워크플로우 조정자로부터 위임 받은 프로세스 인스턴스의 수행을 위해 자신이 직접 업무 수행 노드들을 방문하는 대신 워크플로우 엔진의 도움을 받아 새로운 서브 에이전트를 생성하여 프로세스 인스턴스의 수행을 또 다시 위임하는 것이다. 이럴 경우, 독립적인 프로세스 인스턴스의 수행을 담당하는 프록시 에이전트들 간의 원격 상호작용은 필요하지 않으므로, 단지 워크플로우 조정자가 자신이 생성한 프록시 에이전트들을 원격으로 관리하기 위해서 프록시 에이전트들의 위치 관리 메커니즘이 요구되므로, 워크플로우 조정자 내부에서 직접 프록시 에이전트의 위치를 관리하는 것이 가장 효과적이다. 마찬가지로 워크플로우 엔진 계층에서 프록시 에이전트에 의해서 생성되어 업무 수행 노드계층에 존재하는 여러 노드들을 자율적으로 이동해 다니면서 실제로 프로세스 인스턴스를 수행하는 서브 에이전트들의 경우도 이들 간의 원격 상호작용은 필요하지 않으므로, 단지 프록시 에이전트가 자신이 생성한 서브 에이전트들을 원격으로 관리하기 위한 서브 에이전트들의 위치 관리 메커니즘이 요구된다. 따라서, 프록시 에이전트 내부에서 직접 서브 에이전트들의 위치를 관리하는 것이 가장 효과적이다. 실제로 제안한 이동에이전트 기반 2-계층 분산 워크플로우 서버 아키텍쳐는 이미 암시적으로 이러한 계층적 위임에 의한 프로세스 인스턴스의 수행구조를 가정하고 있다.



(그림 4) 계층적 위임에 의한 프로세스 인스턴스 수행구조

(3) 해결 방안

앞의 착안점에서 논의한 바와 같이 이동에이전트 기반 2-계층 분산 워크플로우 서버 아키텍쳐에 계층적 위임에 의한 프로세스 인스턴스 수행 구조를 도입함으로써 별도의 네임 서버를 두지 않고도 워크플로우 프로세스의 독립적 수행 특성과 계층성을 이용하여 효과적인 이동에이전트 위치관리 전략을 마련할 수 있다. 그러나, 단지 이와 같은 이동에이전트 위치관리 전략만으로는 워크플로우 시스템의 성능 및 확장성 측면에서 이동에이전트 패러다임이 가지는 잠재적 장점을 극대화시킬 수는 없다. 왜냐하면 에이전트가 이동하기 위해서는 오버헤드(migration overhead)가 소요되기 때문이다. 특히 마샬링과 언마샬링의 과정에 소요되는

시간은 에이전트 이동 시간의 약 90%를 차지할 정도로 주된 지연시간이면서 에이전트 크기에 거의 비례하여 증가한다[16]. 그러므로 하나의 워크플로우 프로세스를 성능 및 확장성의 측면에서 전략적으로 여러 개의 서브 프로세스로 분할하여, 각각의 서브 프로세스의 수행을 위임하기 위해 서브 에이전트들을 적절히 생성하고 그들을 스케쥴링하기 위해서는 분할 정책이 정해져야 한다. 그러나 워크플로우 프로세스는 그 타입에 따라 내부 구조가 다양하므로, 분할 정책은 구체적인 것이 아닌 모든 워크플로우 타입에 적용될 수 있는 추상적인 모델이 될 수 밖에 없다. 따라서 우리는 이러한 추상적인 프로세스 분할 정책을 '위임 모델'이라 하고 이를 계층적 위임에 의한 프로세스 인스턴스 수행 구조와 결합하였다. 즉, 모든 워크플로우 프로세스는 워크플로우 조정자에서 프로세스 인스턴스 생성시에 위임모델에 의해 분할되고, 프록시 에이전트는 이 분할 정보를 갖고 워크플로우 엔진으로 이동하여, 워크플로우 엔진이 제공하는 스케쥴러 모듈의 도움을 받아 적절히 서브 에이전트들을 생성하고 이들을 스케쥴링하여, 전체 프로세스 인스턴스를 수행하게 된다. 본 논문에서는 성능 및 확장성 측면에서 갖는 잠재적 장점을 극대화 시키기 위한 이동에이전트 시스템 아키텍처 수준의 전략으로, '위임 모델'에 기반한 계층적 위임에 의한 프로세스 수행 구조하에서 워크플로우 조정자와 프록시 에이전트에 의한 계층적인 에이전트 위치 관리 전략을 제안한다.

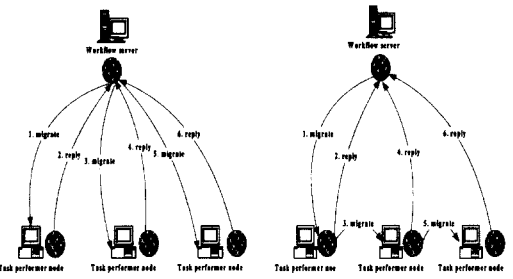
3.1.3 상위 수준 전략

(1) 문제점

'위임모델'의 도입으로 인한 이익을 극대화하려면, 시스템의 성능 및 확장성 측면에서 최적의 위임 모델을 찾아야 한다. 그러나, 워크플로우 시스템 내에는 많은 워크플로우 프로세스들이 존재하고 그 타입에 따라 프로세스의 구조가 다양하므로 최적의 위임 모델을 찾는 것은 어렵다. 그러므로 최적에 가까운, 소위 '비교적 좋은 위임 모델'을 찾아내어 위임 모델을 통한 성능 및 확장성의 향상을 얻어내야 한다.

(2) 착안점

비교적 좋은 위임 모델을 얻기 위해서는 비교할 수 있는 기준 모델과 함께 판단 근거(decision criteria)가 마련되어야 할 것이다. 따라서 하나의 프로세스 인스턴스의 수행을 하나의 이동에이전트에게 전부 위임하여 그 이동에이전트가 프로세스 내의 모든 업무들을 자율적으로 수행하는 모델을 최대 위임모델(그림 5 오른쪽)이라 하고, 반대로 워크플로우 프로세스를 각각의 단위업무로 분할하여 모든 단위업무에 대해 하나의 서브 에이전트를 생성하여 그 수행을 위임하는 모델을 최소 위임모델((그림 5) 왼쪽)이라고 할 때, 두 가지 기본(trivial) 위임 모델을 기준 모델로 삼고 이들이 성능 및 확장성 측면에서 갖는 특성들을 판단 근거로 삼아, 이들 기준 모델들 보다 성능 및 확장성 측면에서 비교적 좋은 위임 모델을 고려한다. 실제로는 워크플로우 프로세스의 구조에 의존하겠지만, 일반적으로



(그림 5) 최소 위임모델과 최대 위임모델

- 최대 위임 모델은 확장성 측면(워크플로우 엔진과의 비동기적 수행특성)에서 최적이지만, 성능 측면(이동 오버헤드)에서는 최악에 가깝다.
- 최소 위임 모델은 이와 반대로 성능 측면에서 최적에 가깝지만, 확장성 측면에서는 최악의 모델이다.

최적의 위임 모델은 이 두 모델의 적절한 혼합형태가 될 것이다. 즉, 최대 위임 모델에 기반하여 최소 위임 모델을 혼합하는 모델과 그 반대의 경우가 그것이다. 그럴 경우 전자는 확장성을, 후자는 성능을 더 강조하는 모델임은 자명하다. 지금까지의 아키텍처 수준의 전략에서 확장성의 측면을 주로 고려하였기 때문에, 상위수준 전략인 위임 모델은 최소 위임 모델에 기반하여 최대 위임 모델을 혼합함으로써 성능 측면에 더 중점을 둔 비교적 좋은 모델을 고려한다. 최소 위임모델에 기반으로 함으로써 서브 에이전트 크기의 효율성으로 인한 성능의 장점을 향유하되, 확장성 측면에서 임계경로(critical path)에서는 일부 최대 위임모델을 도입하여 비동기적인 수행 특성을 부여함으로써, 최소 위임 모델이 갖는 비동기적 수행 특성의 결핍을 보완

할 수 있는 위임 모델을 제안하고자 한다.

(3) 해결 방안

최소 위임 모델에 기반하여, 다음과 같은 확장성 측면(워크플로우 엔진과의 비동기적 수행특성)을 고려하였다.

- 선택경로(OR-split path)는 최소 위임 모델 입장에서 순차경로(sequential path)와 같다. 그러나, 최대 위임 모델에서는 에이전트의 중복성 측면에서 임계경로(critical path)이다.
- 병행경로(AND-split path)는 최소 위임 모델에서 임계경로이지만, 최대 위임 모델에서는 최적경로(optimal path)이다. 그러나, 그 분기점 및 병합점은 최대 위임 모델에서 에이전트의 제어 오버헤드 측면에서 임계점(critical point)이다.
- 병행경로는 그 안에 재귀적으로 내포된(nested) 병행/선택 경로를 가질 수 있다.

이러한 고려 사항을 기반으로 한 새로운 위임 모델은 다음과 같다.

- 새로운 위임 모델

만약 주어진 워크플로우 프로세스 내에 어떠한 병행 경로도 존재하지 않는다면 제안하는 모델은 최소 위임 모델과 같다. 그렇지 않다면, 다음과 같다.

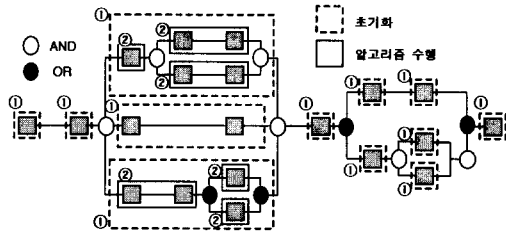
◦ 초기화

어떤 병행경로에도 내포되지 않은 모든 병행 경로들을 각각 하나의 서브 프로세스로 정의하고, 나머지 부분은 모든 단위 업무들을 각각 하나의 서브 프로세스로 정의한다.

◦ 알고리즘

- 병행/선택 경로를 포함하는 서브 프로세스들에 대해, 어떤 병행경로에도 내포되지 않은 모든 병행/선택 경로들을 각각 하나의 서브 프로세스로 정의하고, 나머지 부분은 분기/병합 점을 경계로 하는 서브 프로세스들로 분할된다.
- 더 이상 병행/선택 경로를 포함하는 서브 프로세스가 존재하지 않을 때 까지 위와 같은 분할을 반복한다.

워크플로우 프로세스가 (그림 6)과 같이 주어졌을 때, 새로운 위임 모델은 (그림 6)에 나타난 사각형들로 워크플로우 프로세스를 분할 한다. 먼저 초기화 과정에서 12개의 서브 프로세스(점선 사각형)로 분할된다. 이들 중 병행/선택 경로를 포함하는 2개의 서브 프로세스에 대해 (그림 6)과 같이, 알고리즘에 따라 각각 3개의 서브 프로세스(실선 사각형)로 분할되어, 결과적으로 16개의 서브 프로세스로 분할된다. 새로운 위임 모델은 앞의 확장성 측면의 고려사항을 반영하여 순차 및 선택 경로에서는 최소 위임 모델을 적용하여 성능의 장점을 취하면서도, 병행경로 및 병행경로 내에 내포된 병행/선택 경로에서는 최대 위임 모델을 적용하여 비동기적 수행특성을 부여함으로써 최소 위임 모델이 가지는 확장성의 문제를 완화하였다. 이러한 위임 모델은 병행 경로 및 병행경로 내에 내포된 병행/선택 경로에서 최대 위임 모델을 적용함으로써 이동 오버헤드가 증가하여 성능 저하 요인이 발생한다. 그러나, 제어 오버헤드 측면에서 임계점에 해당하는 분기/병합점이 제외될 뿐만 아니라, 원래 최소 위임 모델이 갖지 못했던 병행 경로 내의 업무들의 병행 수행이 가능해짐을 고려할 때, 제안한 모델은 두 가지 기본 모델인 최소/최대 위임 모델에 비해 더 우수한 성능 및 확장성을 제공할 수 있다. 여기에 대한 정량 분석은 4장에서 보이도록 하겠다.

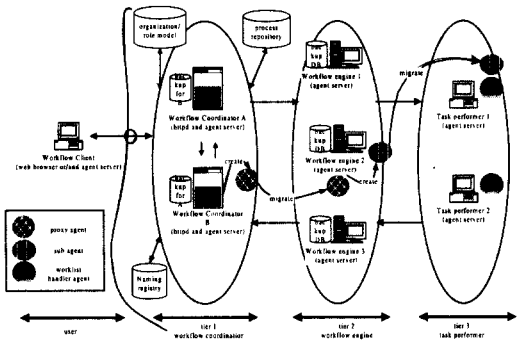


(그림 6) 새로운 위임모델

3.2 전체 아키텍처

앞 절에서 제안한 이동에이전트 기반 2-계층 분산 워크플로우 서버 아키텍쳐에 업무 수행 계층이 추가되어 전체 아키텍쳐는 (그림 7)과 같은 3-계층 아키텍쳐를 형태를 갖는다. 이와 같은 3계층 아키텍쳐가 갖는 결점으로 계층1의 워크플로우 조정자가 single source of failure를 제공한다는 문제가 발생한다. 즉, 워크플로우 조정자에 고장이 발생하면 전체 시스템이 마비된다. 따라서, (그림 7)에 나타난 것처럼 두 개 이상의 워크플로우 조정자를 두어 상호 예비저장(backup)을

통한 가용성(availiability)제고 노력이 요구된다.



(그림 7) 이동에이전트 기반 3-계층 워크플로우 시스템 아키텍쳐

3.2.1 아키텍쳐의 구성 요소 및 기능

본 아키텍쳐는 (그림 7)에서 보는 것처럼 다음과 같은 구성 요소로 이루어져 있으며 각각의 기능은 다음과 같다.

- 프로세스 저장소(Process repository)
  - 모든 종류의 워크플로우 스키마 정보가 프로세스 템플릿 형태로 저장되어있다. 이 프로세스 템플릿은 프록시 에이전트를 생성할 수 있는 중간형태 자료(immediate data)에 해당한다.
  - 모든 프로세스 템플릿들은 이미 시스템에서 정의된 위임모델에 의해 서브 프로세스들의 병렬/직렬 연결로 분할 되어있다.

- 워크플로우 조정자(workflow coordinator)
  - 워크플로우 시스템의 외부, 즉 워크플로우 클라이언트, 모니터링 툴, 관리·감독 툴 등과 워크플로우 시스템 내부와의 인터페이스를 제공한다.
  - 클라이언트로부터의 요청을 받아 프록시 에이전트를 생성하여 워크플로우 엔진으로 보냄으로써 프로세스 인스턴스를 생성한다.
  - 프록시 에이전트들과의 통신이나 프록시 에이전트들을 관리·감독할 수 있는 원시함수(primitives)를 제공하여 계층2에 대한 통제 기능을 마련해 준다.

- 워크플로우 엔진(workflow engine)
  - 많은 프록시 에이전트가 동시에 수행될 수 있는 환경을 제공한다.
  - 스케쥴링 모듈이나 회복모듈 등의 프로세스 인스턴스의 수행에 필요한 핵심 기능을 제공하여 프록시 에이전트들이 프로세스 인스턴스를 수행하도록 지원한다.
  - 프록시 에이전트들의 요청에 따라 서브 에이전트들을 생성하여 해당하는 업무 수행자(task performer)에게 보냄으로써, 태스크 인스턴스를 생성한다.
  - 서브 에이전트들과의 통신이나 서브 에이전트들을 관리·감독할 수 있는 원시함수를 제공하여 계층1에 대한 통제 기능을 마련해 준다.

- 과업 수행 노드
  - 과업 수행자가 맡은 과업을 수행할 수 있도록 여러 툴들과 인터페이스를 제공한다.
  - 서브 에이전트들과 작업리스트 처리 에이전트(worklist handler agent)가 서로 상호작용을 통해 태스크 인스턴스를 수행할 수 있는 환경을 제공한다.

- 프록시 에이전트(Proxy agent)
  - 프로세스 인스턴스를 나타내며 수행시 정보(run-time data)를 캡슐화해서 가지고 있다.
  - 주어진 프로세스 분할 정보에 따라 워크플로우 엔진의 스케쥴러 모듈에 요청하여 서브 에이전트를 생성하고 스케쥴링한다.
  - 그들이 생성한 서브 에이전트들의 위치 정보를 관리한다.

- 서브 에이전트(sub - agent)
  - 서브 프로세스 인스턴스를 나타내며 수행시 정보](run-time data)를 캡슐화해서 가지고 있다.
  - 서브 프로세스 내에 존재하는 모든 업무들을 갖고 자율적으로 이동하며 각각의 업무에 대해 작업리스트 처리 에이전트의 중재에 의한 업무 수행자와의 상호작용을 통해 태스크 인스턴스를 수행함으로써 전체 서브 프로세스 인스턴스를 수행한다.

- 작업리스트 처리 에이전트(worklist handler - agent)
  - 서브 에이전트들과 과업 수행자간의 상호 작용을 중재한다.

3.2.2 워크플로우 프로세스 수행시나리오

본 아키텍쳐에서의 프로세스 수행시나리오는 다음과 같다.

● 프로세스 인스턴스 생성 과정
- 사용자가 워크플로우 클라이언트에서 제공하는 사용자 인터페이스를 통해 워크플로우 조정자에 로그온한 후, 희망하는 워크플로우 프로세스의 수행을 요청한다.
- 사용자로부터 얻은 정보를 바탕으로, 워크플로우 조정자는 프록시 에이전트를 생성하기 위해 필요한 여러 가지 정보를 준비하기 위해서 여러 데이터베이스에 접근한다.
- 워크플로우 조정자는 프록시 에이전트를 생성하고 가장 부하가 적은 워크플로우 엔진을 찾기 위해 부하 관리자를 열람한다. 프록시 에이전트를 보낼 목적지 워크플로우 엔진이 정해지면, 이 목적지를 프록시 에이전트 위치 관리 레지스트리에 등록하고 프록시 에이전트를 보낸다.
- 프록시 에이전트가 도착한 워크플로우 엔진에서는 프록시 에이전트를 재활성화시킨다.

● 프로세스 인스턴스 수행 과정
- 프록시 에이전트는 각각의 서브 프로세스에 대해 서브 에이전트의 생성을 엔진에게 요청한다.
- 워크플로우 엔진은 요청받은 서브 에이전트를 생성시키고 정해진 과업 수행노드로 보낸다.
- 서브 에이전트는 부여 받은 모든 과업들을 하나씩 이동하면서 점진적으로 모두 수행하고 소멸한다.
- 서브 에이전트는 각각의 과업이 끝나고 다음 과업 수행 노드로 이동하기 전에, 해당 과업이 완료되었음과 함께 다음 목적지의 위치 정보를 프록시 에이전트에게 알린다.

● 프로세스 인스턴스 종료 과정
- 수행해야 할 모든 서브 프로세스들을 모두 수행하면, 프록시 에이전트는 프로세스 인스턴스가 종료되었음을 워크플로우 조정자에게 알리고 소멸한다.
- 워크플로우 조정자는 종료된 프록시 에이전트에 대한 정보를 프록시 에이전트 위치 관리 레지스트리와 부하 관리자로부터 제거하고 해당 사용자에게 워크플로우 프로세스가 종료되었음을 알린다.

3.2.3 아키텍처의 성능 및 확장성

여기서는 본 아키텍처가 앞 절에서 제안한 세 가지 구현 전략들을 반영함에 따라 갖게 된 특성들을 정리해보고, 이러한 특성들로부터 얻을 수 있는 성능 및 확장성 측면에서의 장점들을 정성적으로 살펴본다.

(1) 3계층 구조

본 아키텍처가 갖는 3계층 구조는 워크플로우 프로세스 수행 구조가 갖는 계층 구조와 일치 한다. 즉, 워크플로우 조정자 계층이 워크플로우 프로세스 수준을, 워크플로우 엔진 계층이 프로세스 인스턴스 수준을 그리고 업무 수행 계층이 태스크 인스턴스 수준을 전담하게 된다. 이처럼 아키텍처가 갖는 계층구조가 워크플로우 프로세스 수행의 계층 구조와 일치됨으로써 얻을 수 있는 장점은 전체 시스템 내의 제어와 부하가 3계층에 골고루 분산될 수 있다는 것이다. 다시 말해, 워크플로우 조정자는 프로세스 인스턴스의 생성 및 관리·감독 기능을, 워크플로우 엔진은 프로세스 수행의 기능을, 그리고 업무 수행 노드는 태스크 인스턴스 수행의 기능을 전담하게 되므로, 워크플로우 프로세스 수행시 제어와 부하가 상위 계층으로부터 하위 계층으로 이동에이전트에 의해 위임되어 전달되기 때문이다.

(2) 계층적 위임에 의한 프로세스 인스턴스 수행 구조

본 아키텍처가 갖는 3-계층 구조의 특성으로 인한 장점을 극대화함과 동시에 효과적인 이동에이전트 위치관리 전략을 제공하기 위해, 프로세스 인스턴스의 수행을 위한 위임에 계층성을 부여하였다. 즉, 계층적인 위임에 의한 프로세스 인스턴스의 수행은 본 아키텍처의 3계층 특성 및 워크플로우 프로세스 수행 구조의 계층성과 조화를 이루어 전체 시스템의 제어 및 부하를 3계층에 골고루 분산시켜주는 효과적인 전략이며, 지적한 대로 효과적인 이동에이전트 위치관리 전략을 가능하게 한다. 또한, 계층2에서 서브 에이전트들을 생성할 때 워크플로우 엔진이 직접 생성하지 않고, 워크플로우 엔진의 도움을 받아 프록시 에이전트가 자율적으로 서브 에이전트의 생성 및 스케쥴링을 담당하게 되는데, 이러한 프록시 에이전트 기반의 계층적 위임 및 스케쥴링으로 인하여 모듈성이 증가된다. 하나의 워크플로우 엔진에 많은 수의 프록시 에이전트들이 동시에 수행할 수 있기 때문에 이러한 모듈성은 프로세스 관리·감독 차원에서 매우 중요하며, 특히 이동에이전트인 프록시 에이전트의 이동성을 이용한 동적 부하 균형화(dynamic load-balancing)을 가능하게 하는 전제 조건이 되므로 성능 및 확장성 측면에서도 중요한 장점이 된다.

4. 위임모델의 정량 분석

이번 장에서는 위임모델이 이동에이전트 기반 워크플로우 시스템의 성능 및 확장성에 미치는 영향을 확인하고 3장에서 비교적 좋은 위임 모델로서 제안한 새로운 위임 모델이 성능 및 확장성 측면에서 두 개의 기본 위임모델보다 우수함을 정량적으로 보이기 위해, 각각의 위임 모델에 대해 GSPN(Generalized Stochastic Petri Net)[17] 시뮬레이션을 통한 성능평가를 수행하고 그 결과를 제시한다. 단, 세 가지 위임모델에 대한 정량분석에 무관한 프록시 에이전트를 통한 계층적 위임이나, 서브에이전트 위치관리는 고려되지 않았다.

4.1 성능 평가 모델 정의

하나의 워크플로우 서버와 20개의 업무 수행 노드로 구성된 중앙 집중형 이동에이전트 기반 워크플로우 시스템을 고려한다. 시스템 형상(system configuration)은 오직 (그림 6)과 같은 워크플로우 프로세스 하나만을 위임모델에 기반하여 수행할 수 있도록 고정되어 있다. 따라서 업무 수행 노드들의 기능 이나 수행시간을 제외하고는, 세 개의 위임 모델에 대해서 서로 다른 세 개의 시스템 형상이 존재하게 되고 워크플로우 서버의 기능도 각기 다르다. 세 가지 위임 모델에 대한 시뮬레이션 모델의 정의는 다음과 같다.

4.1.1 공통 사항(가정)

시스템 내의 노드끼리는 논리적인 채널을 통해 연결되고, 여러 개의 에이전트나 메시지가 하나의 채널을 통해 동시에 이동할 수 있다. 단, 워크플로우 서버와 업무 수행노드 간의 채널은 양방향으로 채널이 각각 하나씩 존재하며, 업무 수행노드 간의 채널은 단방향으로 하나만 존재한다. 또한 모든 업무는 자동 업무이며 업무 수행시간은 결정적이고 하나의 업무를 수행하기 위해 필요한 에이전트의 크기는 수행할 업무의 종류나 수행시간에 관계없이 동일하다. 만약 하나의 업무를 자율적으로 수행하기 위해 필요한 에이전트의 크기가 M Kbytes라면 N개의 업무를 자율적으로 수행하기 위해 필요한 에이전트의 크기는 0.5M(N+1) Kbytes이다. 채널을 통하는 모든 메시지의 크기는 0.1M Kbytes로 같다. 논리적 채널을 통해 에이전트가 이동하는데 소요되는 시간은 마샬링과 언마샬링 시간만으로 이루어지며, 이 시간은 에이전트의 크기와 정비례한다. 서브에이전트들의 위치 정보 관리는 별도의 네임 서버를 두지 않고 워크플로우 엔진에서 한다.

4.1.2 최소 위임 모델 기반 워크플로우 시스템 모델

한 개의 워크플로우 서버와 20개의 업무 수행 노드 간의 논리적 채널이 star topology로 연결되어 있고, 업무 수행노드 간의 채널은 존재하지 않는다. 워크플로우 서버는 프로세스 인스턴스가 생성되면 정의된 모든 업무가 수행 완료될 때까지, 각각의 업무에 대해 서브 에이전트를 생성하여 그 수행을 위임하고, 서브 에이전트로부터 수행완료에 대한 통지를 받게 되면 다음 업무의 수행을 위해 서브 에이전트를 생성하는 과정을 반복한다.

4.1.3 최대 위임 모델 기반 워크플로우 시스템 모델

한 개의 워크플로우 서버와 20개의 업무 수행 노드 간의 논리적 채널이 star topology로 연결되어 있고, 업무 수행노드 간에는 (그림 6)에서 나타난 업무간의 연결과 동일하게 왼쪽에서 오른쪽으로 채널이 연결되어 있다. 워크플로우 서버는 프로세스 인스턴스가 생성되면 하나의 서브 에이전트를 생성하여 전체 프로세스 인스턴스를 수행을 위임하고, 서브 에이전트는 프로세스 내에 정의된 모든 업무들을 자율적으로 수행하며, 각각의 업무가 수행 완료 될 때마다 워크플로우 서버에게 통보한다. 이때, 서브 에이전트는 병행 분기에서 클론(clone)을 생성하여 분기하고, 병행병합에서는 클론 들간의 그룹 통신(group communication)을 통해 수행 정보를 취합하여 하나의 에이전트만 살아 남아서 수행을 진행한다.

4.1.4 제안된 위임 모델 기반 워크플로우 시스템 모델

한 개의 워크플로우 서버와 20개의 업무 수행 노드 간의 논리적 채널이 star topology로 연결되어 있고, 업무 수행노드 간에는 (그림 6)에서 나타난 서브 프로세스간의 연결과 동일하게 외쪽에서 오른쪽으로 채널이 연결되어 있다. 워크플로우 서버는 프로세스 인스턴스가 생성되면 워크플로우 프로세스 정의된 모든 업무가 수행 완료될 때까지, 각각의 서브프로세스 대해 서브 에이전트를 생성하여 그 수행을 위임하고, 서브 에이전트로부터 수행완료에 대한 통지를 받게 되면 다음 서브프로세스 수행을 위해 서브 에이전트를 생성하는 과정을 반복한다. 이때, 서브 에이전트는 서브 프로

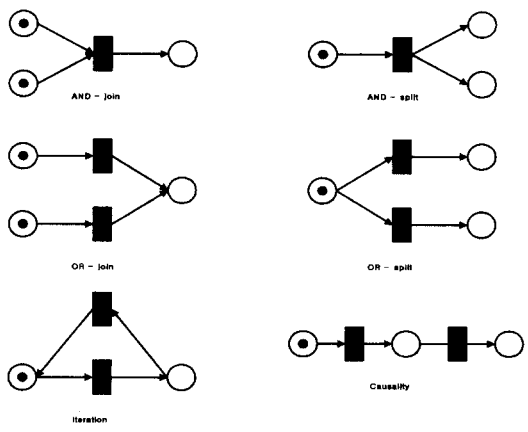
세스 내에 정의된 모든 업무들을 자율적으로 차례로 방문하여 수행하며, 각각의 업무가 수행완료 될 때마다 워크플로우 서버에게 통보한다.

4.2 시뮬레이션 모델

여기에서는 앞 절에서 정의된 성능 평가 모델에 대한 시뮬레이션을 통해 성능평가를 수행하기 위해서, 세 개의 성능 평가 모델 각각에 대한 GSPN 모델을 얻고자 한다.

4.2.1 워크플로우 프로세스 수행의 GSPN 모델

워크플로우 프로세스의 수행은 페트리 넷을 통해 모델링될 수 있을 뿐만 아니라, 몇 가지 장점을 가진다[18]. (그림 8)은 WfMC에서 제시한 6가지 workflow primitive들이 어떻게 각각 페트리 넷 모델로 맵핑되는지를 보여준다. 그림에서 보듯이, 업무들은 시간 트랜지션(timed transitions)으로 맵핑되고 업무들 간 인과관계(causal relation)들은 플레이스(place)들로써 모델링된다. 그리고 프로세스 인스턴스의 수행 상태는 전체 넷에 분포된 토큰의 분포로써 모델링된다. 다만 동시에 수행중인 여러 프로세스 인스턴스를 모델링하고자 할 때는 각각의 프로세스 인스턴스에 해당하는 토큰들을 구분해내야 하므로 칼라(color)를 가진 상위 수준 페트리넷 모델이 필요하게 된다. 즉, GSPN으로는 동시에 수행 중인 다수의 프로세스 인스턴스의 수행 상태를 제대로 모델링 할 수 가 없다. 그러나, 우리는 여기서 다수의 프로세스 인스턴스의 동시 수행에 따른 프로세스 평균 인스턴스 수행시간 만을 얻고자 하므로 개별적인 프로세스 인스턴스들의 수행상태를 굳이 구분할 필요가 없다. 따라서 위와 같은 방법으로 하나 이상의 프로세스 인스턴스의 수행을 GSPN을 가지고 효과적으로 모델링 할 수 있다.



(그림 8) Workflow primitives

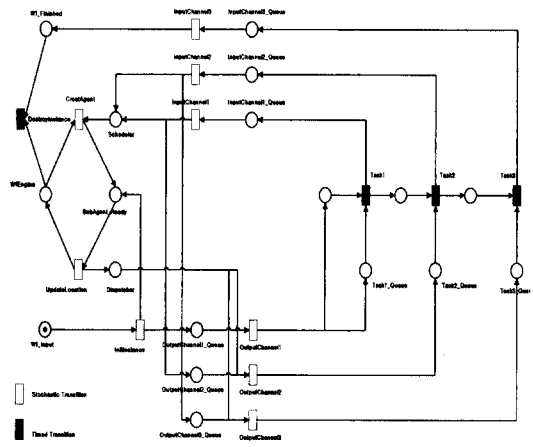
4.2.2 워크플로우 시스템의 GSPN 모델

이제 지금까지 논의된 사항들을 기반으로, 4.1절에서 정의된 성능 평가 모델들 각각에 대한 GSPN 모델을 정의하도록 하자.

- 공통 사항들
  - 업무 수행 노드
  - 서브 에이전트들을 위한 대기 큐를 모델링한 플레이스 하나를 입력으로 갖는 시간 트랜지션(업무 수행시간)으로 모델링 된다.
  - 논리적 채널
  - 서브 에이전트들이나 메시지를 위한 하나의 대기 큐를 모델링한 플레이스 하나를 입력으로 갖는 확률 트랜지션(stochastic transition)(마샬링, 언마샬링 오버헤드)으로 모델링 된다.

- 최소 위임 모델 기반 워크플로우 시스템

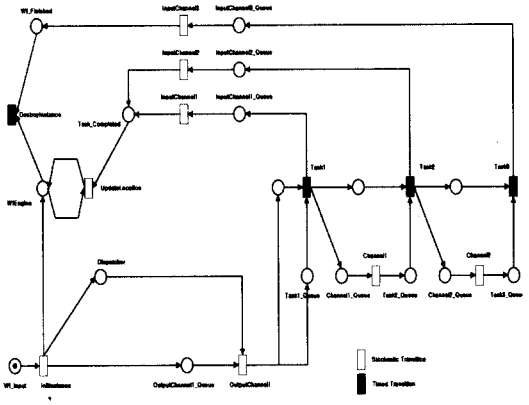
  세 개의 업무를 순차적으로 수행하는 최소 위임 모델 기반 워크플로우 시스템의 GSPN 모델이 (그림 9)에 제시되어 있다. 그림의 왼쪽부터 차례로 워크플로우 서버, 워크플로우 서버와 세 개의 업무 수행 노드들을 연결하는 채널들, 그리고 업무 수행노드가 각



(그림 9) 최소 위임모델 기반 워크플로우 시스템의 GSPN 모델

각 모델링 되어있고, 이 세 요소가 arc에 의해 위임 모델에 따른 프로세스 수행 절차대로 고정되어 있다.

- 최대 위임 모델 기반 워크플로우 시스템
  세 개의 업무를 순차적으로 수행하는 최대 위임 모델 기반 워크플로우 시스템의 GSPN 모델이 (그림 10)에 제시되어 있다.



(그림 10) 최대 위임모델 기반 워크플로우 시스템의 GSPN

- 제안된 위임 모델 기반 워크플로우 시스템
  이 모델의 경우는 (그림 6)에서 점선으로 된 사각형, 즉 서브 프로세스 단위는 최소 위임 모델 기반으로 모델링하고, 각각의 서브 프로세스를 구성하는 업무 단위는 최대 위임 모델을 기반으로 위의 두 가지 GSPN모델을 이용해서 쉽게 모델링 할 수 있다. 세 개의 업무를 순차적으로 수행하는 워크플로우 시스템의 GSPN 모델은 최소 위임 모델 기반 시스템(그림 9)과 동일하다.

4.3 GSPN 모델들의 시뮬레이션

여기서는 앞 절에서 얻어진 세 개의 위임 모델에 대한 GSPN 모델들을 GreatSPN[19]으로 각각 시뮬레이션 하였다.

4.3.1 시뮬레이션 파라미터

- 프로세스 인스턴스 갯수
  본 시뮬레이션의 목적이 동시에 수행하는 프로세스 인스턴스의 수에 대해 세 개의 모델이 성능 및 확장성을 비교하는 것이므로, 프로세스 인스턴스의 개수는 가장 주요한 시뮬레이션 파라미터이다.

세 가지 모델이 성능 및 확장성 측면에서 갖는 특성을 공정하게 비교하기위해서는 각 모델의 충실성(fidelity)에 달려있다. 그러므로, 적어도 시뮬레이션 결과가 의미를 갖게 하기위해서, 모델 자체에 포함되어야 할 다음 사항들을 파라미터화 함으로써 최소한의 공정성을 확보할 수 있다.

- 워크플로우 서버의 처리 시간
  워크플로우 서버의 처리시간은 워크플로우 서버가 서브 에이전트를 생성하여 파견하거나, 서브에이전트의 위치 정보를 관리하는 데 소요되는 시간으로, 세 모델의 성능 및 확장성 측면에 영향을 미친다. 특히, 최소 위임 모델의 경우는 이 파라미터에 가장 민감하다.

- 채널에서의 소요 시간
  이것은 이동에이전트나 메시지가 채널을 통해 전송되는데 소요되는 시간으로, 최대 위임 모델의 경우는 이 파라미터에 가장 민감하다.

- 업무수행 시간
  만약 채널이나 워크플로우 서버의 서비스 비율(service rate)이 부하 의존적(load-dependent)이면, (그림 9)와 (그림 10)에서 알 수 있듯이, 업무 수행 시간은 최대 위임모델의 경우에는 업무 수행 노드를 연결하는 채널의 입력 플레이스들, 그리고 최소 위임모델의 경우엔 워크플로우 서버에 존재하는 스케쥴러에 대한 토큰의 도착 간격(inter-arrival time)을 결정하므로, 각각의 모델에서의 성능 및 확장성 비교에 영향을 미친다.

4.3.2 시뮬레이션에서 취한 가정

그러나, 위의 네 가지 파라미터를 모두 파라미터화해서 성능평가를 하는 것은 너무 많은 시간을 필요로 할 뿐만 아니라, 업무 수행시간의 경우에는 부하 의존적인 서비스 비율에 있어서의 충실성을 확보하기 어렵기 때문에, 다음과 같은 단순화 가정을 하였다.

- 시스템 내의 모든 서비스는 부하 독립적(load-independent)이다.
- 따라서 업무 수행시간은 각 모델에 대한 성능 및 확장성 비교에 있어서 아무런 영향을 주지않게 된다.

4.3.3 시뮬레이션 결과

우리는 앞에서 살펴본 세 가지 시뮬레이션 파라미터에 대해서 <표 1>, <표 2>, <표 3>에 나타난 것과 같이 세 개의 위임모델들을 다음과 같이 시뮬레이션을 하였다. 워크플로우 클라이언트로 부터의 프로세스 인스턴스의 수행에 대한 요청은 $\lambda = 1$(단위 시간)인 포아송 프로세스(Poisson process)로 가정하였고, 모든 업무 수행시간은 단위 시간으로 고정하였다.

- 워크플로우 서버가 주요 성능지연 요인인 경우 <표 1> (그림 11) 참조
- 채널이 주요 성능지연 요인인 경우 <표 2> (그림 12) 참조
- 주요 성능지연 요인이 없는 경우 <표 3> (그림 13) 참조

〈표 1〉 워크플로우 서버가 주요 성능지연 요인인 경우

	Transition	rate	semantics
최소 위임모델	CreatAgent	0.1	single-server
	UpdateLocation	0.1	single-server
	InitInstance	0.1	single-server
	InputChannel	10	infinite-server
	OutputChannel	1	infinite-server
최대 위임모델	UpdateLocation	0.1	single-server
	InitInstance	0.1	single-server
	InputChannel	10	infinite-server
	OutputChannel	0.095238	infinite-server
새로운 모델	위와 동일		

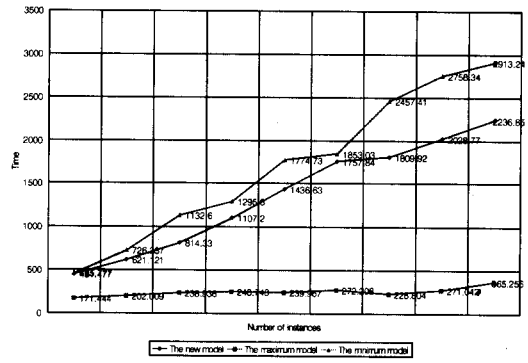
〈표 2〉 채널이 주요 성능지연 요인인 경우

	Transition	rate	semantics
최소 위임모델	CreatAgent	1	single-server
	UpdateLocation	1	single-server
	InitInstance	1	single-server
	InputChannel	1	infinite-server
	OutputChannel	0.1	infinite-server
최대 위임모델	UpdateLocation	1	single-server
	InitInstance	1	single-server
	InputChannel	1	infinite-server
	OutputChannel	0.0095238	infinite-server
새로운 모델	위와 동일		

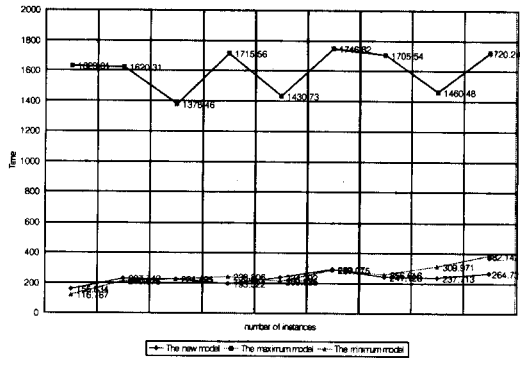
시뮬레이션 결과를 보면 예상한 대로 세 가지 경우 모두, 최대 위임 모델은 확장성에서 그리고 최소 위임 모델과 최소 위임모델을 기반으로 한 새로운 위임모델은 성능에서 비교 우위를 가짐을 알 수 있다. 또한 새로운 위임모델은 최소 위임모델이 확장성

〈표 3〉 주요 성능지연 요인이 없는 경우

	Transition	rate	semantics
최소 위임모델	CreatAgent	1	single-server
	UpdateLocation	1	single-server
	InitInstance	1	single-server
	InputChannel	10	infinite-server
	OutputChannel	1	infinite-server
최대 위임모델	UpdateLocation	1	single-server
	InitInstance	1	single-server
	InputChannel	10	infinite-server
	OutputChannel	0.095238	infinite-server
새로운 모델	위와 동일		



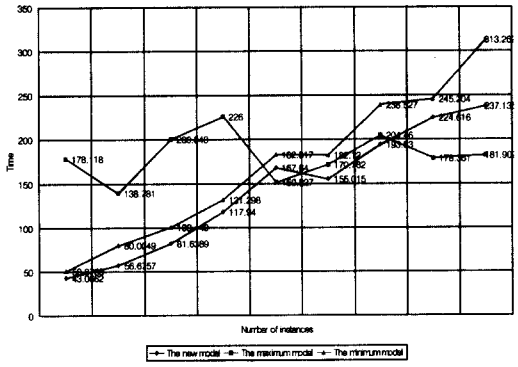
(그림 11) 워크프로우 서버가 주요 성능지연 요인인 경우



(그림 12) 채널이 주요 성능지연 요인인 경우

측면에서 갖는 한계를 개선하고 있음을 보인다. 그러나, 이미 지적한 바와 같이 워크플로우 서버가 주요 성능지연 요인인 경우는 최소 위임모델과 최소 위임모델에 기반한 새로운 위임모델이 민감한 영향을 받기 때문에, 최대 위임모델이 두드러진 절대우위를 보이게 됨을 알 수 있다. 반대로, 채널이 주요 성능지연 요인인 경우는 최대 위임모델이 민감한 영향을 받아

서, 최소 위임모델과 새로운 위임모델이 월등한 절대 우위를 보이고 있다. 따라서, 주요 성능지연 요인이 없는 경우를 기준으로 세 가지의 위임모델이 성능 및 확장성 측면에서 갖는 특성을 비교하면, 새로운 모델은 최소 위임모델이 갖는 성능 측면에서의 장점을 거의 그대로 흡수하면서도, 확장성 측면에서도 최대 위임모델과 최소 위임모델의 중간 정도의 기울기로 단조증가하고 있음을 알 수 있다.



(그림 13) 주요 성능지연 요인이 없는 경우

5. 성능 평가

이번 장에서는 새로운 접근법에 의한 시너지 효과를 정량적으로 확인하기 위하여, 새로운 접근법에 기반하여 워크플로우 시스템의 성능 및 확장성 보장을 위해 지금까지 고려한 세 가지 구현 전략들을 반영한 (그림 7)과 같은 아키텍쳐 상에 새로운 위임모델을 도입한 경우를, 4장에서와 동일한 방법으로 GSPN 시뮬레이션을 통하여 기존의 두 가지 접근법에 기반한 아키텍쳐들과 비교해 보고자 한다. 이때, 성능 평가 모델 자체는 다르지만 시뮬레이션 과정의 제반 사항은 모두 4장의 위임 모델의 정량분석과 동일하므로, 여기서는 성능 평가 모델만 정의하고 바로 시뮬레이션 결과를 제시하겠다.

5.1 성능 평가 모델 정의

세 가지 접근법에 대한 성능평가 모델은 4장에서 정의 되었던 세 가지 위임 모델에 대한 성능평가 모델들로부터 각각 유도될 수 있다. 여기서도 4장에서와 마찬가지로 서브 에이전트들의 위치 관리를 위한 별도의 네임 서버를 두지않고, 워크플로우 서버에서 관리하는 것으로 가정한다. 그리고 4장에서 제시한 세 가지 파라미터에 대해서는 <표 3>과 같이 주요 성능 지연 요인이 없는 경우를 가정하였다.

- 클라이언트-서버 패러다임 기반의 구조적 접근법
  이 모델은 최소 위임 모델의 성능평가 모델을 변형함으로써 얻을 수 있다. 즉, 최소 위임 모델에서는 하나의 서버로 구성된 중앙 집중형 구조를 가정했으나, 여기서는 3개의 분산된 워크플로우 서버로 구성된 중앙 집중형 구조이다. 그리고, 최소 위임 모델에서는 업무의 수행이 서브에이전트에 의해 자율적으로 업무 수행노드에서 로컬 상호작용을 통해 수행되었지만, 여기서는 워크플로우 서버와 업무 수행 노드간의 세 번의 원격 상호작용을 통해 수행되는 것으로 가정하였다.

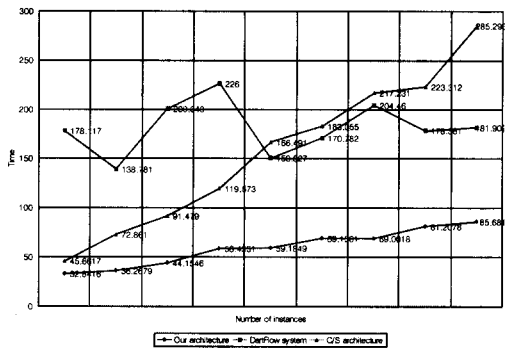
- DartFlow 시스템
  이 모델은 최대 위임 모델의 성능 평가 모델과 동일하다.

- 새로운 접근법
  이 모델은 새로운 위임 모델에 대한 성능 평가 모델로부터 유도될 수 있다. 즉, 하나의 워크플로우 조정자와 세 개의 워크플로우 엔진으로 구성된 워크플로우 서버가 프록시 에이전트에 의해서 동적으로 연결되고, 이들 프록시 에이전트를 기반으로 계층적인 위임이 이루어진다는 것만 다르고 나머지는 같다.

5.2 성능 평가 결과

(그림 14)에 시뮬레이션 결과를 제시하였다. DartFlow 시스템은 (그림 13)에서의 최대 위임모델과 동일하다. 따라서, (그림 13)이 상위 수준의 구현 전략인 새로운 위임모델만을 고려한 경우에 새로운 접근법과 DartFlow 시스템과의 성능 및 확장성의 비교를 보여주었다면, (그림 14)에는 이동에이전트 패러다임에 기반한 아키텍쳐 수준의 구현 전략까지도 포함한 경우에 새로운 접근법과 DartFlow 시스템과의 성능 및 확장성이 비교되고 있다. 그러므로, (그림 13)과 (그림 14)를 비교하여 보면, 새로운 접근법이 갖는 시너지 효과를 쉽게 확인할 수 있다. 즉, 새로운 접근법에서는 확장성 측면을 주로 고려한 아키텍쳐 수준의 구현 전략인 '이동에이전트 기반 2-계층 분산 워크플로우 서버'

와 '계층적 위임에 따른 프로세스 수행 구조로 인한 효과적인 에이전트 위치 관리 전략'을 통해, (그림 13)에서 새로운 위임모델이 보여준 확장성이 (그림 14)에 나타난 바와 같이 괄목할 정도로 향상되어, 결과적으로 성능 및 확장성 모두 다른 접근법들 보다 월등한 절대우위를 나타내고 있다. 한편, 클라이언트-서버 서버에 기반한 접근법의 경우는 최소 위임모델이 여러 개의 서브 업무로 세분된 경우에 해당하므로 (그림 14)에 나타난 바와 같이 최소 위임모델과 유사한 그래프를 나타낸다.



(그림 14) 세 가지 접근법의 성능 및 확장성 비교

6. 결 론

본 논문에서는 워크플로우 시스템이 갖추어야 할 다양한 요구 사항들을 충족시키기 위한 기존의 접근법을 크게 '클라이언트-서버 패러다임 기반의 구조적 접근법'과 '이동에이전트 패러다임의 특성을 이용한 접근법' 두 가지로 구분하고, 이들이 갖는 한계를 지적하였다. 따라서 이 두 접근법들이 서로 다른 수준에 존재하면서 상호 보완적임을 감안하여, 이들이 갖는 한계를 극복할 수 있는 새로운 구현 전략을 제안하고 GSPN 시뮬레이션을 통해 성능 및 확장성 측면에서 그 효용성을 보였다. 즉, 이동에이전트 패러다임에 기반한 3-계층 아키텍처를 제안하고, 제안된 아키텍처 상에서의 워크플로우 수행 시나리오에 워크플로우 수행 수준의 전략인 '위임 모델'을 도입함으로써 성능 및 확장성 측면에서 최적의 효과를 얻을 수 있음을 정량적으로 살펴보았다. 한편, 본 논문에서 제안한 이동에이전트 기반 3-계층 워크플로우 시스템은 성능평가 결과에 나타난 것처럼 기존의 두 가지 접근법들 보다 우수한 성능 및 확장성을 가지면서도 이동에이전트 패러다임을 도입하였기 때문에, 워크플로우 시스템이 갖는 다양한 추가적인 요구 사항들 즉, 신뢰성, 적응성 등에 대해서도 DartFlow 시스템과 같은 방법으로 만족시켜줄 수 있다. 다시 말해서, 제안한 접근법은 주요한 요구 사항들에 대해서는 시너지 효과를 통한 최적의 해법을 제공하면서도, 그 외의 다양한 요구 사항들에 대해서도 이동에이전트 패러다임을 통한 체계적이고 간결한 해법을 제공할 수 있는 장점을 갖는다.

본 논문에서 비교적 좋은 '위임모델'로서 제안한 워크플로우 프로세스 분할 정책은 단지 이동에이전트 시스템 수준에서 고려된 것이다. 따라서, 워크플로우 시스템 수준에서 고려되어야 할 여러 요소들 즉, 성능 및 확장성, 신뢰성, 동적 변경, 동시성 제어 등은 고려되지 않았다. 즉, 이동에이전트를 이용한 워크플로우 관리엔진에서의 이동에이전트 스케쥴링 차원에서 고려되어야 할 여러 고려 사항들은 본 논문에서 다루지 않았고, 현재 이에 대한 연구를 수행하고 있다.

참 고 문 헌

[1] WfMC, 'Workflow Management Coalition Terminology and Glossary', WfMC Specification, pp.8, 1996.

[2] G. Alonso, D. Agrawal, A. el Abbadi, C. Mohan, 'Functionality and Limitations of Current Workflow Management Systems', Electron. Lett., pp.439-444 (1996).

[3] Petra Heinl, Hans Schuster, 'Towards a Highly Scaleable Architecture for Workflow Management Systems', Proc.7th Int. Workshop on Database and Expert Systems Applications, pp.439-444, 1996.

[4] Gustavo Alonso, Hans-Jorg Schek, 'Database Technology in Workflow Environments', Applications, 1996.

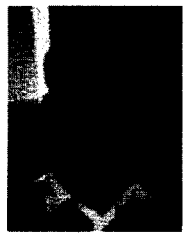
[5] Barbara, D., Mehrota, S., and Rusinkiewicz, M., 'INCAS : A Computation Model for Dynamic Workflows in Autonomous Distributed Environments', Technical report, Matsushita Information Technology Laboratory, 1994.

[6] M. Rusinkiewicz and A. Sheth, 'Specification and Execution of Transactional Workflow', Modern Database Systems : The Object, Interoperability and Beyond, W. Kim(Ed.), Addison-Wesley, 1994.

[7] 유정준, 서영호, 송상범, 이동익, '에이전트 기반 워크플로우 시스템 구조 및 이동에이전트 요구사항', 한국정보처리 학회 99 추계학술발표논문집, 1999.
[8] M. Kamath, G. Alonso, G. Gunthor, and C. Mohan, 'Providing High Availability in Very Large Workflow Management Systems', Proc. EDBT '96, 1996.
[9] 'FlowMark-Managing Your Workflow, Version 2.1', IBM, 1995.
[10] 'COSA Reference Guide', Software-Ley GmbH, 1994.
[11] G. Alonso, D. Agrawal, A. el Abbadi, C. Mohan, R. Gunthor, M. Kamath, 'Failure Handling in Large Scale Workflow Management Systems', IBM Research Report, 1994.
[12] G. Alonso, D. Agrawal, A. el Abbadi, C. Mohan, R. Gunthor, M. Kamath, 'Exotica/FMQM : A Persistent Message-Based Architecture for Distributed Workflow Management', Proc. IEIP Working Conf, on Information Systems for Decentralized Organizations, 1995.
[13] Miller, J. A., Sheth, A. P., Kochut, K. J., Wang, X, 'CORBA-Based Run-Time Architectures for Workflow Management Systems', Journal of Database Management, Special Issues on Multidatabases, Vol.7, 1996.
[14] Ting Cai, Peter A. Gloor, Saurab Nog, 'DartFlow : A Workflow Management System on the Wep using Transportable Agents', DartMouth College, Technical Report PCS-TR96-283, 1996.
[15] Luiz A.G.Oliveira, Paulo C.Oliveira, Eleri Cardozo, 'An Agent-Based Approach for Quality of Service Negotiation and Management in Distributed Multimedia Systems', Proc. First Int. Workshop, MA '97, 1997.
[16] Manfred Dalmeijer, Eric Rietjens, Dieter Hammer, Ad Aerts, Michiel Soede, 'A Reliable Mobile Agents Architecture', Proc. of the Int. Symposium on Object-Oriented Real-Time Distributed Computing(ISORC'98), 1998.
[17] Ajmone Marsan M., Balbo G., Chiola G., Conte G., 'Generalized stochastic Petri nets revisited : Random switches and priorities', Proc. Int.Workshop on Petri Nets and Performance Models, pp.44-53 1987.
[18] W. M. P. van der Aalst, 'Three good reasons for using a Petri-net-based Workflow Management System', Proc. Int. Working Conference on Information and Process Integration in Enterprises(IPIC '96), 1996.
[19] G. Chiola, 'Simulation Framework for Timed and Stochasic Petri Nets', Proc. International Journal in Computer Simulation, 1991.

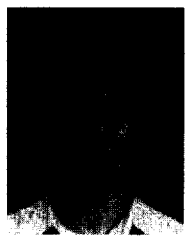
서 영 호

e-mail : yhsuh@etri.re.kr
1998년 전남대학교 컴퓨터공학과 졸업(학사)
2000년 광주과학기술원 정보통신공학과(공학석사)
2000년 현재 한국전자통신연구원 인터넷서비스연구부 연구원
관심분야 : 분산처리, 이동에이전트, 성능평가 등



유 정 준

e-mail : jjyoo@geguri.kjist.ac.kr
1995년 인하대학교 전자계산공학과 졸업(학사)
1997년 인하대학교 전자계산공학과 (공학석사)
1997년 현재 광주과학기술원 박사과정
관심분야 : 분산처리, 이동에이전트, 성능평가, 워크플로우 등



이 동 익

e-mail : dilee@kjist.ac.kr
1985년 영남대학교 전기공학과 졸업(학사)
1989년 일본Osaka 대학 전자공학과(공학석사)
1993년 일본 Osaka 대학 전자공학과(공학박사)
1993년~1994년 University of Illinois 객원 연구원
1990년~1995년 일본 Osaka대학 문부교관 조수
1995년 현재 광주과학기술원 정보통신공학과 조교수, 부교수
관심분야 : 비동기 시스템 CAD/설계, 자율 분산 시스템, 프로토콜 공학, 페트리넷 이론, 정형 기법, 병행시스템 설계 및 분석 등