이질적인 유비쿼터스 환경에서 지속적인 서비스 제공을 위한 최적의 어댑터 체인 생성 기법

김병오° 이경민 이동만
한국정보통신대학교
{cmossetup°, kmlee, dlee}@icu.ac.kr

An Optimal Adapter Chaining Scheme for Seamless Service in Heterogeneous Ubiquitous Environments

Byoungoh Kim° Kyungmin Lee Dongman Lee
Information and Communication University

요 약

유비쿼터스 환경에서 사용자는 장소의 이동에 상관없이 지속적인 서비스를 제공받아야 한다. 이동 중에 서비스 지속성을 보장하려면 대체 서비스를 검색하는 기술뿐만 아니라, 새로운 대체 서비스의 사용을 위한 서비스 인터페이스 간 불일치를 해결할 수 있는 기술이 필요하다. 인터페이스 불일치 문제를 해결하는 대표적인 방법은 인터페이스 어댑터를 사용하는 것이다. 하지만, 어댑터 개발의 부담이 크다는 단점이 있어, 이를 보완하기 위한 방법으로 어댑터의 반자동 생성이나 어댑터 체인 기법에 대한 연구들이 있다. 본 논문에서는 적응 손실 측면에서 최적의 어댑터 체인을 생성하는 기법을 제안한다. 제안한 기법은 기존의 어댑터 체인 기법에서 고려하지 못했던 단일 인터페이스 내의 메소드 의존성을 고려하여 적응 손실을 측정하는 방법과 이를 반영하여 어댑터 체인을 생성하는 어댑터 체인 생성 알고리즘으로 구성된다. 시뮬레이션을 통해 제안한 기법이 비교적 짧은 시간 내에 최적의 어댑터 체인을 생성할 수 있음을 보인다.

1. 서론

유비쿼터스 컴퓨팅 환경에서는 사용자가 한 공간에서 다른 공간으로 이동하더라도 (예를 들어, 사무실에서 집으로 혹은 우리 집에서 이웃집으로) 사용자가 원하는 서비스를 지속적으로 제공받을 수 있어야 한다. 각각의 지능 공간은 서로 독립적으로 구축되고 관리되기 때문에 모든 공간이 동일한 서비스를 제공할 수 없다. 따라서, 사용자가 기존에 사용하던 서비스와 동일한 서비스가 제공되지 않는 경우, 기존 서비스를 대체할 수 있는 새로운 서비스를 찾아 사용하여야 한다 [1,2]. 또한, 기존 서비스와 대체 서비스가 가진 인터페이스가 일치하지 않는 경우에도 지속적으로 서비스를 제공하기 위해서는 서비스 인터페이스의 불일치를 해결하여야 한다.

서비스 인터페이스의 불일치를 해결하는 대표적인 방법은 하나의 서비스 인터페이스를 다른 서비스 인터페이스로 변환해주는 어댑터를 이용하는 것이다 [3]. N개의 서비스 인터페이스 간의 변환을 위해서는 N(N-1)개의 어댑터가 필요하다. 따라서 서비스의 개수가 증가하면 어댑터 개발을 위한 비용이 급증하게 된다. 이러한 문제를 해결하기 위해 개발자의 어댑터 구현을 도와주는 여러 기법들이 제안되었다 [4,5,6]. 하지만 어댑터 개발을 완전 자동화하는 것은 매우 힘들거나 불가능하기 때문에 비용 절감 효과가 제한적이다.

또 다른 어댑터 개발 비용 절감 방법은 다수의 어댑터를 연결한 어댑터 체인을 만들어 사용하는 것이다 [7,8,9]. 어댑터 체인 기법을 이용하면 최소 N-1개의 어댑터를 이용하여 N개 서비스 인터페이스 간의 변환이 가능하게 되어 실제 개발이 필요한 어댑터의 수를 크게 줄일 수 있다. 어댑터 체인 기법에서는 두 개의 서비스 인터페이스 간의 변환이 가능한 다수의 어댑터 체인이 존재할 수 있다. 또한, 각 어댑터 체인의 적응손실이 서로 다를 수 있다[1]. 기존 기법에서는 어댑터 체인을 생성하는 과정에서 이러한 적응 손실의 차이를 고려하지 않기 때문에 최적의 어댑터 체인, 즉, 적응 손실이 가장 적은 어댑터 체인의 생성을 보장하지 못한다.

본 논문은 적응 손실 측면에서 최적의 어댑터 체인을 생성하는 기법을 제안한다. 먼저, 한 인터페이스의

[1] 적응손실(adaptation loss)은 어댑터 또는 어댑터 체인에 의해 변환이 불가능한 메소드(기능)를 말함

메소드들 간의 의존성 정보와 어댑터를 통한 메소드들 간의 변환 정보를 이용하여 어댑터 체인의 적응 손실도(degree of adaptation loss)를 계산하는 방법을 제시한다. 또한 계산된 적응 손실도를 바탕으로 최적의 어댑터 체인을 생성하는 그래프 기반 알고리즘을 제시한다. 시뮬레이션을 통해 제안 알고리즘의 체인 생성 시간과 서비스의 개수 사이의 관계를 보이고, DFS와 최상의 어댑터 체인을 생성하는 Full-spanning DFS와 비교를 통해 제안 알고리즘의 성능을 평가한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구에 대해 논의한다. 3장에서 제안하는 기법에 대하여 자세히 기술하고 4장에서 실험 결과에 대해 논의한다. 마지막으로 5장에서 향후 계획과 함께 논문을 마무리한다.

2. 관련 연구

어댑터의 개발 부담을 감소시키는 방법 중 하나로 템플릿을 이용하여 어댑터를 반자동으로 생성하는 기법이 있다 [4,5]. 이 기법은 서비스 인터페이스 사이에 존재할 수 있는 불일치 패턴 별로 템플릿을 제공한다. 개발자가 템플릿에 입력한 정보를 바탕으로 어댑터 코드를 자동으로 생성한다.

이와 유사한 방법으로, 데이터 변환을 위한 데이터 컨버터들을 조립하여 어댑터를 자동으로 생성하는 방법이 있다 [6]. 이 기법에서 개발자는 인터페이스, 메소드, 데이터들 간의 호환성을 표로 나타내고, 표를 바탕으로 각각의 오퍼레이션을 위한 어댑터를 생성하고, 이를 다시 인터페이스 변환을 위한 어댑터로 통합한다.

하지만, 이러한 기법들은 어댑터를 생성하는 데 있어 개발자의 도움이 필요하기 때문에, 개발해야 할 어댑터의 수가 많아지면 어댑터 개발 비용 또한 증가한다. 그러므로, 어댑터 개발 필요를 최소화하는 방법들에 대한 연구가 필요하다.

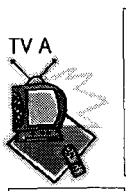
여러 개의 어댑터들을 연결하여 서비스 인터페이스 간 변환을 제공하는 어댑터 체인 기법은 개발이 필요한 어댑터의 수를 크게 줄일 수 있다 [7,8]. 하지만, 기존 연구는 인터페이스 내의 메소드의 지원 여부가 다른 메소드의 지원 여부에 영향을 끼칠 수 있음에도, 어댑터 체인의 적응 손실을 측정하는 과정에서 메소드 간의 관계를 고려하지 않는다[7].

3. 제안 기법

본 논문은 적응 손실 측면에서 최적의 어댑터 체인을 생성하기 위한 기법을 제안한다. 본 장에서는 먼저 어댑터 체인의 적응 손실을 측정하는 방법에 대해 제시하고 적응 손실을 반영한 어댑터 체인 생성 알고리즘을 기술한다.

3.1. 어댑터 적응 손실 평가

제안 기법에서는 인터페이스에 내의 전체 메소드 개수에 대한 어댑터에 의해 변환 불가능한 메소드 개수의 비율로 어댑터의 적응 손실을 나타낸다. 어댑터 체인의 적응 손실을 평가하기 위해 제안 기법에서는 동일 인터페이스 내 메소드들 간 의존성과 어댑터를 통한 메소드 변환 관계를 행렬로 표현하고, 이들 간의 이진 연산 작용을 이용한다.

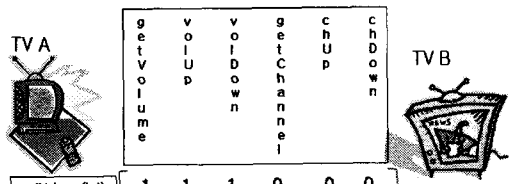


	setVolume	getVolume	setChannel	getChannel
setVolume(int)	1	1	0	0
getVolume	0	1	0	0
setChannel(int)	0	0	1	1
getChannel	0	0	0	1

그림 1. 메소드 의존성 행렬의 예

그림 1은 TV A 서비스 인터페이스의 메소드 의존성 행렬을 나타낸다. 의존성은 Identical, Primary, 그리고 Counter의 3가지 형태로 나타난다. Identical은 메소드 그 자체와의 의존성으로, 행렬 I로 표현할 수 있다. 그리고 Primary는 메소드를 사용하기 위해 요구되는 선행 메소드를 의미하며, powerOn/Off와 같은 함수들이 대표적인 예이다. Counter는 volUp, volDown과 같이 동일 기능에 작용하는 메소드들간의 관계를 의미한다.

또한, 메소드 변환 행렬은 어댑터가 변환 가능한 메소드들과 변환에 필요한 메소드들의 관계를 표현한다. 그림 2의 변환 행렬은 TV A의 setVolume이란 메소드가 TV B의 getVolume, volUp, 그리고 volDown 메소드를 필요로 함을 표현하고 있다.



	getVolume	volUp	volDown	getChannel	chUp	chDown
setVolume(int)	1	1	1	0	0	0
getVolume	1	0	0	0	0	0
getChannel	0	0	0	1	0	0
setChannel(int)	0	0	0	1	1	1

그림 2. 메소드 변환 행렬

본 기법에서는 어댑터 체인의 적응 손실을 평가하기 위해 어댑터의 메소드 변환 행렬들의 곱하여 어댑터 체인의 입력 인터페이스와 출력 인터페이스 간의 관계를 표현하는 메소드 변환 행렬을 이용하는 방법을 제안한다. 또한, 어댑터에 의해 제공 가능한 메소드들을 빠르게 체크할 수 있도록 제공 가능한 메소드들을 벡터 형태로 제공하고, 제공된 벡터와 어댑터 체인 행렬을 비교하여 제공되는 메소드들을 빠르게 판단할 수 있는 방법을 제안한다.

$$C_{AB}\begin{bmatrix}1&0&1\\1&1&0\end{bmatrix} \times C_{BC}\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\end{bmatrix} = C_{AC}\begin{bmatrix}1&1&0&0\\1&0&1&0\end{bmatrix}$$

그림 3. 변환 행렬 도출 예

그림 3은 어댑터 체인의 변환 행렬의 생성 예이다. 행렬 C_{AB}는 인터페이스 A에서 인터페이스 B로의 서비스 적응을 이루어주는 어댑터의 메소드 변환 여부를 이진값으로 표현한 2X3 행렬이다. C_{AB}는 인터페이스 A의 메소드 m_1과 m_2를 제공하기 위한 인터페이스 B의 메소드 m_3, m_4, m_5들의 필요 여부를 표현한다. 어댑터 체인의 적응 손실은 이러한 메소드 간 의존성 행렬과 메소드 변환 행렬의 이진 연산을 통해 나타날 수 있다. 예를 들면, 그림 3의 C_{AB}와 C_{BC}의 이진 연산은 어댑터 체인 A-B-C에 대한 변환 행렬 C_{AC}로 보여진다. C_{AC}의 (1,1)의 1이라는 값은 (1 AND 1) OR (0 AND 0) OR (1 AND 0)의 연산을 통해 얻을 수 있다.

3.2. 어댑터 체인 생성 알고리즘

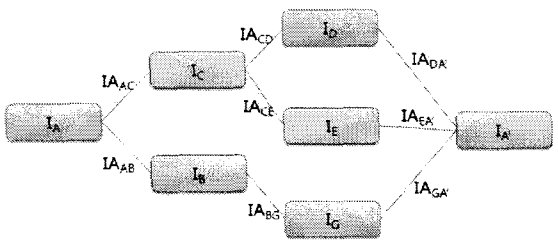


그림 4. 서비스 인터페이스 어댑터 연결 그래프

그림 4는 서비스 인터페이스 A와 A' 사이에 존재하는 서비스 인터페이스와 어댑터를 통해 구성된 그래프이다. 그래프에서 서비스 인터페이스는 점(vertex)에 해당되고 인터페이스 어댑터가 선(edge)에 해당한다. 그리고 선에 부가되는 비용은 적응 손실로 측정이 가능하다.

```
I_A:  Interface of service A
IC_A-TO-A': Adapter chain between interface A and A'
G = (I, IA), w(u, v) >=0 for each interface adapter (u,v) ∈ IA
S: Min-priority queue of interfaces, keyed by their lossiness

DIJKSTRA-SEARCH(R, I_SOURCE, I_TARGET)
1 S ← Ø
2 Q ← I[G]
3 while Q ≠ Ø
4   do u ← EXTRACT-MIN(I)
5     if u = I_TARGET
6         out ← IC_SOURCE-TO-TARGET
7     S ← S ∪ {u}
8     for each interface v ∈ Adj[u]
9       do w ← CALCULATE-LOSSINESS(I_SOURCE, u, v)
10            RELAX(u, v, w)

CALCULATE-LOSSINESS(I_SOURCE, u, v)
1  m_u ← CONVERSION-MATRIX(I_SOURCE, u)
2  m_v ← CONVERSION-MATRIX(u, v)
3  m_uv ← MULTIPLY(m_u, m_v)
4  out ← EVALUATE(m_uv)
```

그림 5. 어댑터 체인 생성 알고리즘

어댑터 체인을 생성하기 위한 알고리즘은 그림 5와 같은 유사 코드를 가진다. 먼저, 인터페이스에 존재하는 메소드 상관 관계 와 어댑터 사이의 메소드 상관 관계를 이용하여 어댑터의 적응 손실 및 성능 손실을 실행 시간에 측정한다. 다음으로 측정된 값들을 이용하여 최소의 비용으로 목적 인터페이스까지 도달할 수 있는 어댑터 체인을 찾게 되며, 이러한 과정은 그래프에 추가 가능한 edge들의 비용이 가장 적은 어댑터들을 반복하여 선택 연결하여 최적의 어댑터 체인을 생성하는 Dijkstra 알고리즘의 변형이다.

4. 평가

시뮬레이션을 통해 제안한 어댑터 체인 기법의 성능 평가를 위하여 비교대상으로써 DFS를 이용한 두 가지 brute force 알고리즘을 구현하였다. 첫 번째 DFS 알고리즘은 어댑터 체인의 적응 손실이 1보다 작고, 가장 빠른 시간 내에 발견되는 어댑터 체인을 검색한다. 그리하여 최상의 조건에서는 최소의 검색 시간을 갖게 된다. 두 번째 DFS 알고리즘은 두 서비스 인터페이스 간의 변환을 제공해 주는 모든 어댑터 체인 중 적응 손실이 가장 적고 가장 짧은 어댑터 체인을 제공하는 Full-spanning DFS이다. 이러한 두 알고리즘과의

비교를 위해 시뮬레이션에서는 동일한 서비스 개수에 대하여 서로 다른 10개의 그래프를 통해 실험을 했다. 어댑터의 개수는 2(N-1)개로 서비스의 개수 N에 비례하여 임의로 생성하였다. 또한, 각각의 그래프마다 임의의 서비스 50쌍에 대한 어댑터 체인을 생성하여 보고 필요한 수치들의 평균값을 구하였다.

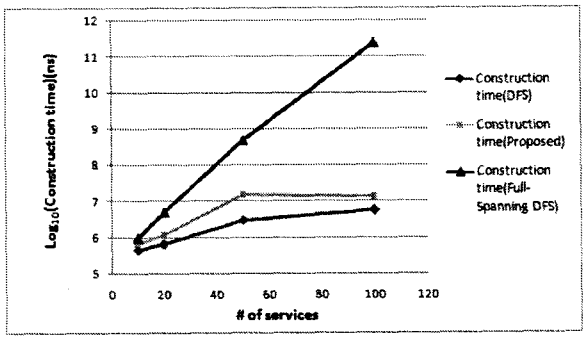


그림 6. 서비스 개수와 어댑터 체인 생성 시간의 관계

그림 6은 서비스 개수에 따른 어댑터 체인 생성 시간을 나노초 단위로 나타낸 그래프이다. 생성시간의 Log 값으로 그려진 그래프로 DFS와 제안 알고리즘의 어댑터 체인 생성 시간이 100밀리초 미만임을 알 수 있다. 또한, Full-spanning DFS의 경우 서비스 개수의 증가에 따른 시간의 상승폭이 매우 크고, 서비스 개수가 100개일 경우 100초 이상의 시간이 걸림을 알 수 있다.

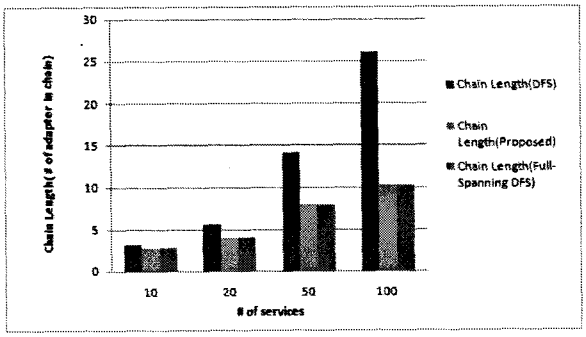


그림 7. 서비스 개수와 생성된 어댑터 체인 길이의 관계

그림 7은 각 알고리즘을 통해 생성된 어댑터 체인의 길이를 표현하고 있는 그래프이다. 서비스의 개수가 증가함에 따라 체인의 길이가 증가하고 있음을 알 수 있다. 또한, DFS의 상승 폭이 제안 기법과 Full-spanning DFS의 상승 폭에 비해 월등히 큼을 알 수 있다. 또한, Full-spanning DFS가 항상 최적의 어댑터 체인을 생성한다는 점을 감안했을 때, 제안 기법 또한 항상 최적의 어댑터 체인을 생성한다고 할 수 있다.

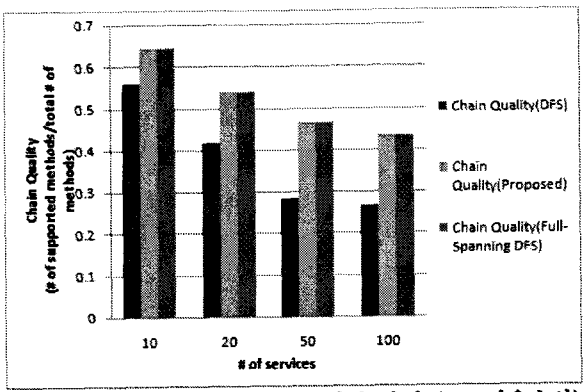


그림 8. 서비스 개수와 적응 손실의 관계 (1 - 적응손실)

그림 8은 서비스 개수와 적응 손실의 관계를 보여준다. 그림을 통하여, 서비스의 개수가 증가할수록 적응 손실이 증가하여 생성된 어댑터 체인의 품질이 떨어짐을 알 수 있다. 또한, DFS를 통해 생성된 어댑터 체인의 품질이 제안 기법과 Full-spanning DFS에 비해 낮음을 발견할 수 있다. 어댑터 체인의 길이와 마찬가지로, 어댑터 체인의 품질에 대해서도 Full-spanning DFS가 최적의 어댑터 체인을 생성하기 때문에, 제안 기법 역시 최적의 어댑터 체인을 생성한다는 사실을 유추가 가능하다.

따라서, 본 논문에서 제안된 기법은 생성 가능한 모든 어댑터 체인들을 비교하여 선택하는 것이 아니라, 어댑터 생성 과정에서 적응 손실을 고려함으로써 짧은 시간 내에 최적의 어댑터 체인을 생성할 수 있음을 알 수 있다.

5. 결론

본 논문에서는 유비쿼터스 환경에서 사용자가 대체 서비스를 사용하기 위해 필요한 인터페이스간 호환성을 제공해 주는 어댑터 체인 생성 기법을 제안하였다. 제안된 어댑터 체인 생성 기법은 행렬을 이용한 어댑터 체인 평가 기법을 이용하여 짧은 시간 내에 최적의 어댑터 체인을 생성 가능함을 알 수 있었다.

향후에는 어댑터 적응 손실 평가에 있어 사용자의 서비스 사용 기록을 반영할 수 있는 방안을 연구할 필요가 있다. 또한, 어댑터 체인의 병렬 통합을 통하여, 실시간으로 서비스 조합을 형성할 수 있는 방안에 대해서 연구할 필요가 있다.

Acknowledgments

본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스 컴퓨팅 및 네트워크 원천기반 기술개발 사업의 지원에 의한 것임

참고문헌

[1] A. Ranganathan, S. Chetan, and R. Campbell. "Mobile Polymorphic Applications in Ubiquitous Computing Environments," in Proc. of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous'04), 2004

[2] J. P. Sousa and D. Garlan. "Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments," in Proc. of the 3rd Working IEEE/IFIP Conference on Software Architecture, 2002

[3] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Reading, Mass. : Addison-Wesley, 1995.

[4] B. Benatallah, F. Casati, D. Grigori, H. R. Motahari Nezhad, and F. Toumani. "Developing Adapters for Web Service Integration," Proceedings of the 17th international conference on advanced information systems engineering, LNCS3520, pp.415-429, 2005

[5] H. R. Motahari Nezhad, A. Martens, F. Curbera, F. Casati. "Semi-Automated Adaptation of Service Interactions", in 16th International World Wide Web Conference, 2007

[6] J. M. Zaha, M. Geisenberger, and M. Groth. "Compatibility Test and Adapter Generation for Interfaces of Software Components", in 1st International Conference on Distributed Computing & Internet Technology, 2004

[7] S. R. Ponnekanti and A. Fox. "Application-service interoperation without standardized service interfaces," in Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003), pp. 30-37, 23-26 March 2003,

[8] P. Kaminski, M. Litoiu, H. Müller. "A design technique for evolving web services," in Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, 2006