ISBN 978-89-6211-387-7

Common Component Architecture의 분석

(Survey of Common Component Architecture)

구 기 범 (voxel@kisti.re.kr)

한 국 과 학 기 술 정 보 연 구 원

Korea Institute of Science & Technology Information

제목 차례

표 차례

그림 차례

1. 개요

계산과학자들이 슈퍼컴퓨터에서 실행하는 시뮬레이션 코드는 오랜 시간에 걸쳐서 점진적으로 개발돼왔기 때문에 상당수가 포트란과 MPI/OpenMP를 사용하고 있다. 하지만 최신 IT 기술을 도입하는 계산과학자가 증가하면서 C/C++, JAVA, Perl, Python 등 다양한 프로그래밍 언어로 각자의 관심분야에 적합한 시뮬레이션 코드를 구현하는 사례가 증가하고 있다. 그 결과, scientific computing 분야에서는 FORTRAN부터 C/C++, JAVA, Perl, Python에 이르기까지 아주 다양한 프로그래밍 언어로 구현된 코드가 존재한다.

한편, 고성능 컴퓨팅 분야에서는 다분야 역학 문제(예: 유체역학+구조해석)를 위한 어플리케이션을 개발하는 움직임이 점점 많아지고 있다. 이 때 문제는 서로 다른 분야의 문제를 해결하는 어플리케이션이 각각 다른 프로그래밍 언어로 개발돼있기 때문에 단일 어플리케이션으로 통합하는 작업을 어렵게 하는 경우가 존재한다는 점이다. 비즈니스 컴퓨팅 분야에서는 이와 유사한 문제를 해결하기 위해 CORBA(Component Object Request Broker Architecture), COM(Component Object Model) 등 컴포넌트 기반 소프트웨어 개발 방법론을 사용해왔다. 하지만 이 방법론들은 고성능 컴퓨팅 분야에서 필요로 하는 성능을 충분히 제공하지 못하기 때문에 해당 모델을 그대로 가져와서 적용하는 데에는 무리가 있다.

CCA(Common Component Architecture)는 고성능 컴퓨팅 환경에서 컴포넌트 기반 소프트웨어 개발을 실현하기 위해 제안한 아키텍처다. 이 모델은 ORNL(Oak Ridge National Laboratory), SNL(Sandia National Laboratory), LLNL(Lawrence Livermore National Laboratory) 등 미국의 주요 국립 연구소가 공동으로 개발했으며, 현재도 개선작업을 진행하고 있다. CCA는 서로 다른 프로그래밍 언어로 구현된 HPC 컴포넌트를 연계하는 방법론을 제공하며, 이 때 컴포넌트의 성능저하를 최소로 줄이고, 병렬 프로그램으로 구현된 컴포넌트의 특징을 그대로 살려줌으로써 HPC 환경에 적합하도록 설계되었다. 현재 미국의 주요 대학교와 연구소에서는 CCA와 호환이 가능한 프레임워크와 각종 툴을 개발하고 있는데, 그 중에서도 Utah의 SCI 그룹이 개발한 S

CIRun2, Legion, XCAT 등이 CCA 사양을 구현하고 있으며, LLNL의 Babel과 같이 서로 다른 프로그래밍 언어로 구현된 컴포넌트 사이의 연계도구 또한 공개되어 있다.

이 보고서에서는 “A Component Architecture for High-Performance Scientific Computing”이라는 논문을 요약해서 CCA에 대해 알기 쉽게 설명하고자 작성한다.

2. 개요

이른바 '슈퍼컴퓨터'를 이용해서 대형 문제를 해결하는 고성능 컴퓨팅(high performance scientific computing) 분야에서는 해를 거듭할수록 단일 어플리케이션이 사용하는 컴퓨팅 자원의 규모가 기하학적으로 커지고 있다. 매해 6월과 11월, 세계에서 가장 높은 성능을 갖춘 컴퓨터의 순위를 정리해서 보여주는 Top500의 성능 변화를 살펴보면 이 현상을 더 분명히 확인할 수 있는데, 최근 Top500의 수위를 차지하는 컴퓨터의 성능은 이제 PFLOPS(PetaFLOPS) 수준에 도달해 있고, 2020년이 되기 전에 EFLOPS(ExaFLOPS)의 성능을 갖춘 시스템이 등장할 것으로 예상하고 있다. 특히 Cell, GPU, Larrabee 등 연산 속도가 수백 GFLOPS를 상회하는 프로세서가 등장하기 시작하면서 최고 수준의 성능을 갖춘 슈퍼컴퓨터의 성능 향상 속도는 더 빨라질 것으로 예상하고 있다.

한편, 계산과학자들은 '하드웨어'의 규모가 커지는 것만큼은 계속 인지하고 있기 때문에 전보다 더 큰 문제에 도전하고자 하는 욕구가 아주 강한 편이다. 하지만 대부분의 계산과학자는 지금도 MPI/OpenMP와 배치 스케줄러를 이용한 전통적인 프로그래밍 패러다임을 따르기 때문에 대규모 컴퓨팅 자원의 효율적인 사용에는 취약점을 드러내고 있다. 그리고 같은 physics를 이용해서 단순히 mesh의 해상도만 높이는 것만으로는 의미 있는 연구를 수행하는 데에 한계가 있기 때문에 다분야 해석문제를 같이 풀거나 전과는 다른 physics를 도입해서 규모가 다른 문제를 해결하려고 하는 경향이 증가하는 추세다. 문제는 다분야 해석문제나 새로운 physics를 적용한 대형 문제를 풀기 위한 어플리케이션을 개발할 때에는 그에 적절한 개발 방법론 – 서로 다른 프로그래밍 언어로 된 모듈의 통합 등 – 을 적용해야 하는데, 대부분의 계산과학자가 이 문제에 대해서는 깊게 고민하지 않는다는 점이다.

Common Component Architecture(이하 CCA)는 고성능 컴퓨팅 분야에서 대규모 어플리케이션 개발을 용이하게 하기 위해 제안한 컴포넌트 모델이다. 비즈니스 컴퓨팅 분야에서는 이미 오래 전부터 CORBA(Component Object Request Broker Architecture), COM(Componen

t Object Model) 등 객체지향방법론을 한 단계 더 발전시킨 '컴포넌트 기반 방법론'이 사용되었지만 HPC 분야에서는 컴포넌트 기반 소프트웨어 개발 방법론의 도입이 거의 없었다. CCA의 가장 큰 특징은 CORBA나 COM과 같은 비즈니스 컴퓨팅 분야에서는 '분산 환경에서의 컴포넌트 사이의 효과적인 연결'에 초점을 맞추는데 비해 CCA는 '컴포넌트 사이의 연결'뿐만 아니라 '고성능'을 중요하게 생각한다는 점이다. 하지만 초기 진입장벽 등을 고려해서 CCA는 의도적으로 CORBA와 같은 일반적인 컴포넌트 모델과 유사한 형태로 디자인됐다. 그러나 HPC의 특성이 필요한 부분(추후 설명)에 대해서는 구현을 달리함으로써 HPC 어플리케이션의 특성을 충분히 살릴 수 있도록 했다.

가. CCA의 디자인 철학

CCA는 다음의 특징을 갖도록 설계되었다.

- Component : CCA에서의 컴포넌트는 SPMD나 MPMD 모델로 구현된 프로그램을 연계하는 데에 초점을 맞추고 있다. 이를 위해서는 메시지 전달(message passing), 컴포넌트를 연계하기 위한 런-타임 시스템, 멀티 쓰레드, 대용량 데이터의 효과적인 전송방법론 등 HPC 어플리케이션 개발에 필요한 모든 방법론을 망라해야 한다.
- Heterogeneity : CCA에서는 단일 어플리케이션을 구성하는 컴포넌트가 서로 다른 언어로 작성되어 있으면서, 동시에 각 컴포넌트가 서로 다른 아키텍처에서 실행되는 것을 허용해야 한다.
- 로컬 / 리모트 컴포넌트 : CCA에서는 컴포넌트가 단일 어드레스 공간을 갖고 있는지의 여부에 따라서 로컬 컴포넌트와 리모트 컴포넌트로 구분한다. 로컬 컴포넌트 사이의 정보 교환은 실질적으로는 함수의 호출 이상의 오버헤드를 갖지 않도록 하며, 리모트 컴포넌트 사이의 정보 교환은 LAN이나 WAN 등 연결 거리나 형태에 구애받지 않고 최신 네트워크 기술을 모두 활용하면서 최적의 성능을 구현하도록 권장하고 있다.
- 통합 : CCA 환경 내에서의 컴포넌트의 통합은 가능한 쉬워야 하며,

컴포넌트 형태로 구현되지 않은 어플리케이션이라고 해도 이를 컴포넌트로 바꾸는 데에 필요한 오버헤드는 가능한 적어야 한다. CCA에서는 일단 컴포넌트가 만들어지면 CCA 호환이 되는 서로 다른 프레임워크에서 같이 활용할 수 있어야 한다.

- 고성능 : 컴포넌트 사이의 정보 교환이나 컴포넌트가 CCA 프레임워크에 통합될 때에는 병렬 데이터 전송 등 리소스가 제공하는 최고의 성능을 이끌어내기 위해 가능한 모든 방법을 동원한다.
- 개방형 구조와 단순함 : 컴포넌트를 사용하는 가장 큰 이유이기도 한데, 별다른 어려움 없이 기존의 코드(컴포넌트 또는 어플리케이션)를 도입하거나 재사용할 수 있어야 한다.

나. CCA의 개념

CCA 포럼(http://www.cca-forum.org)에서는 앞에서 설명한 조건을 충족시키는 컴포넌트 모델의 표준안을 만들고 있다. 여기에는 컴포넌트 사이의 정보 교환을 위해 필요한 구성요소와 정보 교환 방법론 등을 모두 포함하고 있다. 우선 CCA의 구성요소를 간단하게 살펴보면 다음과 같다.

- Component : 컴포넌트는 어플리케이션을 구성하는 단위 요소다. 컴포넌트의 내부 구조는 개발자나 이용자가 전혀 몰라도 상관없으며, 잘 정의된 일련의 인터페이스를 이용해서 컴포넌트를 사용할 수 있다.
- Port : 포트는 컴포넌트 사이의 정보 교환을 위해 정의된 추상적인 인터페이스를 통칭한다. 간단하게 생각하면 C++의 클래스에서 말하는 member function이 개념적으로 유사하다. 컴포넌트는 다른 컴포넌트가 자신의 기능을 사용할 수 있도록 포트를 외부에 제공할 수도 있고(provides port라고 부른다), 다른 컴포넌트가 제공하는 포트를 사용할 수도 있다(uses port라고 부른다).
- Framework : CCA와 호환이 되는 모든 컴포넌트는 프레임워크 내에서 통합되고, 그 안에서 단일 어플리케이션으로 실행된다. 프레임워크는 각 컴포넌트의 uses port와 provides 포트를 연결해줌으로

써 컴포넌트의 연계를 책임진다.

다. CCA 표준의 개요

CCA에서는 SIDL(Scientific Interface Definition Language)라는 언어를 이용해서 컴포넌트 사이의 인터페이스를 구현한다. SIDL은 Babel이라는 language interoperability tool이 사용하는데, Babel 덕분에 서로 다른 프로그래밍 언어로 구현된 컴포넌트 사이의 연계가 가능하게 되는 것이다.

CCA 스펙의 가장 핵심은 Services라는 인터페이스로, 컴포넌트와 프레임워크가 통신하는 수단이 된다. 컴포넌트는 Services 인터페이스를 이용해서 자신이 제공하는/사용하는 포트를 프레임워크에 알려준다. 그 반대로 프레임워크가 제공하는 서비스를 확인하는 데에도 이 인터페이스를 사용한다. Services는 컴포넌트와 관련이 있는 포트를 선언하기 위해 다음의 method를 제공한다.

- addProvidesPort
- removeProvidesPort
- registerUsesPort
- unregisterUsesPort

프레임워크는 위의 함수를 통해서 CCA 프레임워크 기반 어플리케이션을 구성하는 컴포넌트 사이의 연결 관계를 정리한다. 그리고 사용자가 포트를 사용할 때에는 다음과 같은 method를 이용해서 port handle을 관리할 수 있다.

- getPort
- getPortNonblocking
- releasePort

특정 컴포넌트가 CCA와 호환이 되도록 하려면 그 컴포넌트는 CCA의 Component라는 인터페이스를 구현해야 한다. 이 인터페이스가 바로 일반적인 어플리케이션 코드가 CCA 컴포넌트로 바뀌도록 하는 최소한의 장치다. CCA 컴포넌트는 고유의 ComponentID가 있어서 프레임워크 내에서 특정 컴포넌트를 식별하는 데에 사용하고 있다. 컴포넌트와 마찬가지로 CCA 포트 역시 타입과 이름을 갖고 있다. 이 때 타입은 포트의 SIDL 인터페이스 이름으로 설정되고, 이름은 컴포넌트 내에서는 중복되는 일이 없어야 한다.

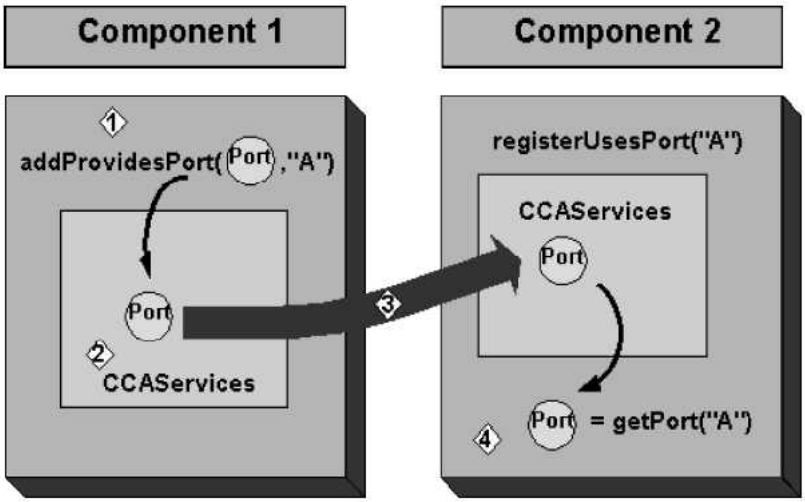
CCA에서는 특수한 목적을 갖고 있는 포트가 다음과 같이 준비되어 있다.

- GoPort
- BasicParameterPort
- ParameterPortFactory

CCA 프레임워크 역시 자체 서비스를 위해 포트를 컴포넌트에 제공한다. CCA 컴포넌트가 제공하는 포트와의 차이점은 프레임워크가 제공하는 포트는 언제나 사용 가능하지만 컴포넌트가 제공하는 포트는 포트 사이의 연결이 맺어진 후에만 사용할 수 있다는 사실이다. 모든 CCA 프레임워크는 다음의 포트를 제공해야 한다.

- ConnectionEventService
- BuilderService
- AbstractFramework
- ComponentRepository

라. Component의 라이프사이클



그림은 서로 다른 컴포넌트의 포트가 연결되고, 사용되는 과정을 보여주고 있다. Services는 프레임워크가 소유하지만 각각의 컴포넌트가 해당 인터페이스를 모두 사용하기 때문에 일단 그림에는 Services를 포함시켰다.

먼저 컴포넌트 1이 addProvidesPort를 이용해서 Services에 어떤 포트를 제공하는지 알려주고, 컴포넌트 2는 registerUsesPort를 이용해서 어떤 포트를 사용하는지 알려준다(결국 CCA 프레임워크에 등록되는 것으로 보면 된다). 그 다음 각 컴포넌트의 포트 등록 결과에 따라서 프레임워크가 uses 포트와 provides 포트를 연결하고, 컴포넌트 1의 provides 포트 정보를 컴포넌트 2의 Services에 복사해준다. 컴포넌트 2가 컴포넌트 1의 포트를 사용할 때는 getPort를 호출함으로써 해당 포트에 대한 핸들을 가져온 후 사용하면 된다.

이 그림에 나타내지 않은 것이 releasePort 호출인데, 이는 해당 포트를 모두 사용했음을 의미한다.

3. Language interoperability

Scientific computing 분야에서 서로 다른 역할을 담당하는 컴포넌트의 결합을 어렵게 하는 문제는 바로 프로그래밍 언어의 다양성에 있다. 특이하게도 계산과학자들은 지난 수십 년 간 FORTRAN으로 작성된 코드를 주로 사용해왔으며, 아직도 그 사용비율이 전체 프로그래밍 언어의 50%를 상회하는 것이 현실이다(반면 비즈니스 컴퓨팅 분야에서 FORTRAN은 오래 전에 사라졌다). 반면 새로 개발하는 소프트웨어는 C/C++이나 JAVA, Perl, Python을 이용하는 사례가 늘고 있기 때문에 여러 종류의 컴포넌트를 연계할 때에는 서로 다른 프로그래밍 언어로 구현된 컴포넌트를 연계하는 방안을 제시해야 한다.

SWIG(Simplified Wrapper and Interface Generator)은 Perl, Python, Ruby, Tcl로 만들어진 코드 내에서 C/C++로 구현된 코드를 호출할 수 있도록 하지만 그 반대의 경우는 지원하지 않는다는 단점이 있다. 이 문제를 해결하기 위해서 LLNL에서는 Babel이라는 툴을 개발했다. Babel은 IDL(interface definition language) 기반 툴로, 서로 다른 프로그래밍 언어로 구현된 컴포넌트를 연계할 수 있도록 해준다. Babel에서는 SIDL(Scientific Interface Definition Language)을 통해서 각 컴포넌트가 주고받는 데이터 형식을 정의한다.

Babel은 CORBA나 COM과 마찬가지로 OOP를 지원하지 않는 프로그래밍 언어라고 해도 OOP을 지원하도록 관련 개념(object identity, inheritance, polymorphism, encapsulation 등)을 다수 구현하고 있다. Babel의 inheritance 모델은 JAVA와 유사하고, 그에 따라서 SIDL 클래스는 단 하나의 클래스와 상속관계를 유지한다.

당연한 얘기지만 Babel과 SIDL이 CORBA나 COM과 같은 다른 IDL 기반 도구와 다른 점은 바로 scientific computing 어플리케이션을 위한 특징(데이터 타입, 동적 관리가 가능한 다차원 배열 등)을 별도로 갖추고 있다는 점이다.

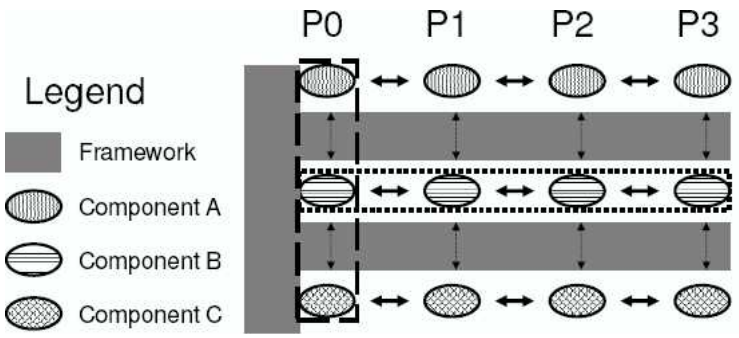
4. Parallel Components

CCA에서는 병렬 컴퓨팅 환경을 지원하기 위해 다양한 컴포넌트의 통합을 가능하게 한다. 현존하는 병렬 코드를 CCA 환경으로 통합하는 가장 쉬운 방법은 그 코드의 병렬 프로그래밍 모델을 그대로 사용하는 것이다. CCA에서는 이 부분을 가장 중요한 이슈 중 하나로 생각했고, 이 문제를 해결하기 위해 몇 가지 방법을 제안하고 있다.

지금까지 개발된, CCA 규격과 호환이 되는 프레임워크는 여러 가지가 있는데, 현재 기술로는 각 프레임워크가 HPC 병렬 환경(클러스터와 같은)이나 WAN 기반의 분산 환경을 동시에 지원하지는 못하고 있다. 따라서 이와 같은 복잡한 환경에서 실행되는 어플리케이션을 개발하려면 `BuilderService`와 `AbstractFramework`와 같은 서비스를 이용해서 서로 다른 프레임워크에서 실행되는 컴포넌트를 연계하는 방법을 사용해야 한다.

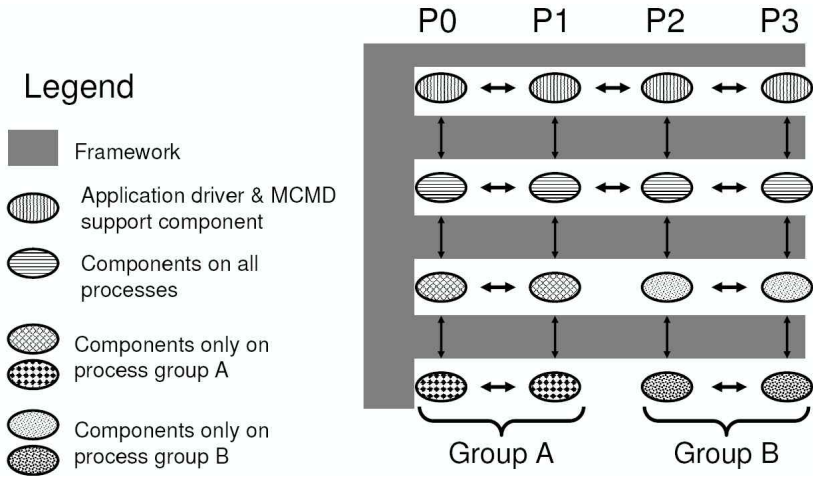
Ccaffeine은 HPC 병렬 컴퓨팅 환경에 최적화된 CCA 프레임워크로 SPMD 데이터 모델과 MPMD 데이터 모델을 모두 지원한다. CCA 규격에서는 이것을 SCMD/MCMD (두 번째 글자인 C는 Component를 의미한다)라고 부르는데, 실제 두 개념 사이의 차이는 없다.

[그림 4-1]은 CCA에서 사용하는 SCMD 모델을 보여주고 있다. 이 그림에서 각각의 컴포넌트는 MPI task로 생각할 수 있으며, P0, P2, ... 는 각 컴포넌트를 구성하는 프로세스를 의미한다.



[그림 4-1] CCA에서의 SCMD 모델

[그림 4-1]에서 동일한 컴포넌트를 구성하는 프로세스 사이의 통신은 기존의 전통적인 메시지 패싱 방법(MPI, PVM 등)을 사용하면 된다.



[그림 4-2] CCA에서의 MCMD 모델

[그림 4-2]는 CCA에서의 MCMD 모델을 보여준다. MCMD 모델의 가장 전형적인 예는 전지구 기상모델에서 대기, 해양, 지표 기상 모델을 서로 다른 컴포넌트로 실행하는 경우다. MCMD 어플리케이션의 경우 데이터 교환, 프로세스 그룹의 관리 등 복잡한 문제들이 발생하기 때문에 대부분 CCA의 `BuilderService`를 이용해서 MCMD 어플리케이션을 개발하도록 유도하고 있다.

5. CCA기반 개발 도구

[표 5-1]은 2005년 6월 당시 사용 가능했던 CCA 툴을 요약해서 보여주고 있다.

Category	Name	Version	Capabilities
CCA Spec	cca-spec-babel	0.7.5	
Language Interoperability	Babel	0.10.0	Supports C, C++, FORTRAN 77, Fortran 90/95, Java, Python
	Chasm	1.1.0	Used by Babel for Fortran 90 array support
Frameworks	Ccaffeine	0.5.6	Local direct-connect and parallel computing (SCMD and MCMD paradigms)
	XCAT-Java	3.0.6	Local direct-connect and distributed computing compatible with Grid- and web-services approaches
	XCAT-C++	1.0	Uses the Proteus Multi-protocol library and distributed computing compatible with Grid- and web-services approaches
	Legion-CCA	1.0	Distributed computing compatible with the Legion environment
	SCIRun2	1.92	Local direct-connect, MPI/threaded parallel, and distributed computing
Graphical Interface	ccafe-gui	0.5.5	.Visual programming. interface for assembly of component-based applications in the Ccaffeine framework
Complete CCA Environment	cca-tools	0.5.7	Integrated software distribution including the CCA specification, Babel, Chasm, Ccaffeine, and ccafe-gui. It is the recommended way for most users to obtain a complete CCA environment.

[표 5-1] CCA 규격과 호환 가능한 소프트웨어의 목록

6. 결론

이 보고서는 Common Component Architecture에 대한 일반적인 사항에 대해서 설명했다. 우리나라에 비해 계산과학자들의 IT 기술 도입이 상당히 빠른 미국에서는 대규모 HPC 어플리케이션을 개발할 때 발생하는 문제에 대해 Component 기반 방법론을 도입하는 등 다양한 기법을 통해서 해결하려는 움직임이 활발한 편이다. 특히 계산과학자가 동시에 실행하는 쓰레드(또는 프로세스)의 수가 10만개 수준을 넘어서고 있는 요즘, 전통적인 메시지 패싱 라이브러리만으로는 안정적인 어플리케이션 실행환경을 확보할 수 없기 때문에 새로운 미들웨어를 개발하려는 움직임이 많이 관측되고 있다. 하지만 국내의 경우 여러 가지 사정상 유사한 주제에 대한 연구활동이 부진한 것에 대해서는 많이 아쉬움이 남는다.

비록 CCA를 구현한 소프트웨어는 그다지 많이 사용되고 있지는 않으나, 대규모 시뮬레이션과 data visualization의 연계에 대한 고민의 시발점이 될 수 있을 것으로 기대한다.