濟州大學校 尖端技術硏究所 論文集, 第12卷, 2號, pp.133-140, 2001. 12
Cheju Nat'l Univ. Res. Inst. Adv. Tech. Jour., Vol.12, No.2, pp.133-140, 2001. 12

자료지향 클래스도와 협력도에 관하여

변 상 용*

On Data-Oriented Class-Diagram and Collaboration-Diagram

Sang-Yong Byun*

ABSTRACT

There is a trend of increasing interest for object-oriented software development. In the object-oriented requirement analysis phase, we construct use-case diagrams, class-diagrams and state-diagrams. However any of them can't provide information for data transfer between classes. In the data processing software, data flow between software components should be comprehended. And in the object-oriented design phase, we construct interaction-diagrams and detailed class-diagrams, but also any of them can't provide information for data transfer between classes or objects. Therefore this paper suggest class-diagrams and collaboration-diagrams with capability of data-transfer and data-return between software components.

Key Words : class-diagram, collaboration-diagram, object-oriented modeling

Ⅰ. 서 론

소프트웨어 분석 및 설계 방법에는 크게 행위지향, 자료지향, 객체지향적인 방법들이 있으며, 요즈음에는 객체지향 방법이 많이 사용되고 있다. 그 이유는 객체지향 파라다임에서 사용되는 모듈인 객체가 행위지향 파라다임에서의 모듈보다 덩치가 크기 때문에, 복잡한 대규모 소프트웨어를 개발할 때 유리하며, 객체지향 소프트웨어는 행위지향 소프트웨어에 비해 유지보수성이 뛰어나며, 재사용성도 우수하기 때문이다[1,2].

행위지향 접근방법에서는 소프트웨어가 갖게 될 행위들을 중심으로 문제 영역을 파악하며, 자료는 부차적인 것으로 생각한다. 반면에 자료지향 접근방법에서는 자료들을 중심으로 문제 영역을 파악하며, 행위는 부차적인 것으로 생각한다. 이들 접근 방법이 행위 또는 자료라는 편향된 시각에서 문제 영역에 접근한다면, 객체지향 접근방법은 자료와 행위에 대하여 동일한 비중을 두어 문제 영역에 접근한다[1,2].

객체지향 접근방법은 문제 영역에서의 객체들을 식별하고, 이들 객체들의 모임으로써 전체 시스템을 구성한다. 따라서 객체지향 접근 방법에서는 문제 영역에서 객체들을 식별하고, 객체들간의 관계를 파악하는 것이 가장 중요한 문제이다[3]. 이를 위해서 객체지향 분석 단계에서는 클래스 모형화(class modeling)를, 설계 단계에서는 상호작용도(interaction diagram)를 구축한다[4,5]. 상호작용도 중의 하나인 협력도(collaboration diagram)는 객체들간의 행위적 측면을 나타낸다. 순수 자료지향적인 클래스 모형화의 결과는 클래스도

* 제주대학교 통신 컴퓨터 공학부, 첨단기술연구소
Faculty of Telecommunication and Computer Eng., Res. Insti. Advanced Tech., Cheju Nat'l Univ.

(class diagram)로 나타내는데, 클래스간의 관계는 나타내지만, 자료적 측면을 보이지는 못한다. 협력도도 객체들간의 자료적 측면을 보이지는 못한다.

이 논문에서는 객체의 모태가 되는 클래스들간의 자료 전달 측면을 나타내기 위한 자료지향 클래스도를 제안하며, 객체들간의 자료의 전달과 반환을 나타내기 위해 자료지향 협력도를 제안한다. 클래스간의 관계를 나타내는 클래스도 보다는 소프트웨어 분석에서 더 강력한 도구가 되며, 객체간의 행위 측면을 나타내는 협력도 보다 소프트웨어 설계에서 더 강력한 도구가 된다는 것에 대하여 논한다.

II. 기존 객체지향 모형화에서의 부족점

2.1 객체지향 분석에서의 부족점

객체지향 소프트웨어 개발은 문제 영역에서의 객체를 식별하고, 식별된 객체들의 집합으로 소프트웨어 시스템을 개발한다. 객체지향 분석은 객체지향 파라다임을 위한 반 정형적 명세 기법이며, 도형적 표기가 주요 부분이며, 결과는 통합 모형화 언어(UML : Unified Modeling Language)[6]를 사용하여 나타낸다. 객체지향 분석 단계는 사용사례도(use case diagram)와 각본(scenario)의 구축, 클래스 모형화와 동적 모형화 단계로 구성된다.

사용사례도와 각본을 구축하는 단계에서는 결과 제품이 어떻게 하여 여러 결과들을 계산하는 가만 나타내어, 장차 클래스를 식별하고 클래스들간의 관계를 식별하기 위한 정보를 제공한다. 클래스 모형화 단계에서는 클래스와 그 클래스가 가지는 속성을 결정하고, 클래스간의 상호관계를 결정하고, 결과를 개체-관계도(entity-relationship)와 상당히 유사한 UML 클래스도로 나타내는 것이 일반적이다. 이 단계는 순수 자료지향적이다. 한편 동적 모형화 단계에서는 각 클래스와 하위 클래스의 행위들을 결정하며, 결과를 UML 상태도(state diagram)로 나타내는 것이 일반적이다. 이 단계는 순수 행위지향적이다.

상태도는 한 클래스에 대해서만 관심을 가지고, 클래스간의 관계에 관심을 가지지는 않는다. 따라서 객체지향 분석 단계동안에 클래스들간의 관계를 나타낼 수 있는 것은 클래스도 뿐이다. 클래스도는 클래스가 가지는 속성은 나타내지만, 클래스가 가지는 행위는 나타내지 않는다. 클래스에 대한 UML 표기법은 Fig. 1과 같다.

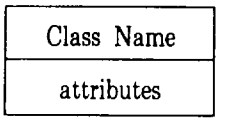

Fig. 1. UML notation for Class.

두 클래스간의 기본적인 클래스도는 Fig. 2와 같다.

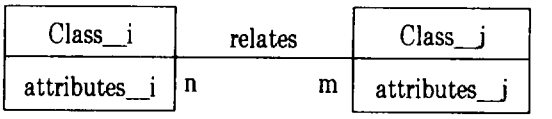

Fig. 2. basic class-diagram between two classes.

관계는 일반적으로 'Class_i'가 'Class_j'에 대해 가지는 관계이다. 'n'은 'Class_i'의 실체인 n 개의 객체가 'Class_j'의 실체인 객체 하나와 관계를 가진다는 것이고, 'm'은 'Class_j'의 실체인 m 개의 객체가 'Class_i'의 실체인 객체 하나와 관계를 가진다는 것을 의미한다.

Fig. 3은 응용 영역에 대한 클래스도 예를 나타내고 있다. 예제 클래스도에서 보이는 바와 같이 클래스 자체가 가지는 속성과 클래스간의 관계는 나타내지만 클래스간에 어떠한 자료들이 오고 가는 지는 명확하게 나타내지 못하고 있다. 단지 자료 전달에 대한 추측만이 가능하다. 예를 들어 'PointScore' 클래스에서 'StudentGroup' 클래스로 PointScore 자료들이 전달될 것이라는 것을 추측을 할 수 있지만, 전부 또는 일부의 자료가 전달되는 지는 문맥을 살펴보아야 한다. 한편 'Controller' 클래스에서 'StudentGroup' 클래스로 어떤 자료가 전달될 지는 추측하기 어렵다.

2.2 객체지향 설계 단계에서의 부족점

객체지향 설계 단계에서는 객체지향 분석 단계에서 구축한 각 각본에 대한 상호작용도(interaction diagram)

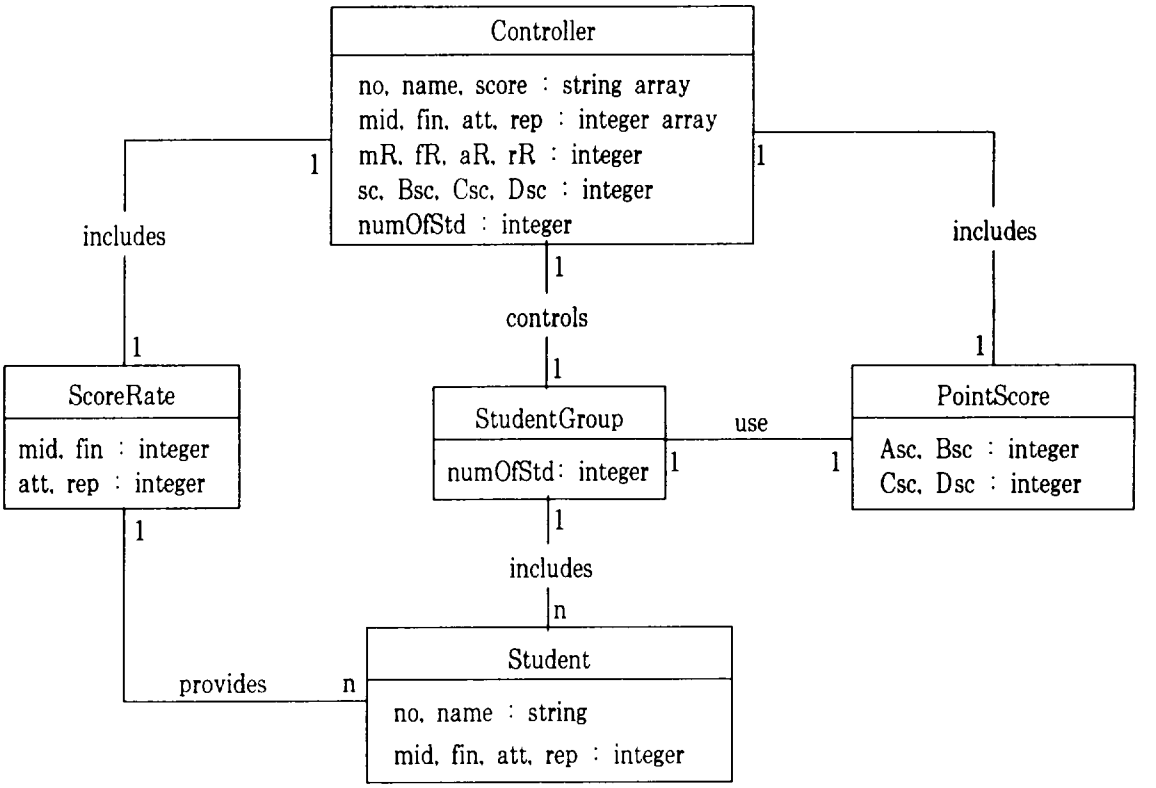

Fig. 3. example of class diagram.

를 구축하고, 상세한 클래스도(detailed class diagram)을 구축하며, 객체들의 고객들의 측면에서 제품을 설계한다.

상호작용도에는 순차도(sequential diagram)과 협력도(collaboration diagram)가 있으나, 협력도가 객체들간의 관계를 나타내는 데에 사용된다. Fig. 4는 두 객체간의 기본적인 협력도를 보이고 있다.

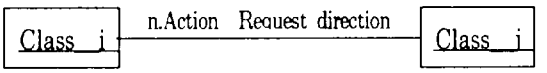

Fig. 4. basic collaboration-diagram between two objects.

'n'은 다른 객체에게 행위를 요청하는 순서를 나타내며, 'Action_Request'는 한 객체가 다른 객체에게 요청하는 행위를 나타내며, 'direction'은 객체들 사이에 행위를 요청하는 방향을 나타낸다. Fig. 5는 응용영역에 대한 협력도 예를 보이고 있다.

이 협력도에서 'Action_Request'가 적절하게 기술되면, 객체간에 전달되는 자료들을 예측하는 것이 가능하지만, 한계를 가질 수 밖에 없다. 예를 들어 'setRateData'이라면 'Controller' 객체에서 'ScoreRate' 객체로 'rateData'가 전달되는 것을 짐작할 수 있지만, 'deterPoint'인 경우에는 'Controller' 객체가 'StudentGroup' 객체에게 어떠한 자료를 전달하는지 알기 어렵다.

이를 보완하기 위하여, 'Action_Request'에 매개변수와 반환 자료를 가지는 경우도 있다[7]. 예를 들어 '15. deterPoint →'인 경우에 '15. point := deterPoint(pointScore) →'와 같이 나타낸다. 하지만 모든 행위 요청들을 이와 같이 나타낸다면 협력도가 복잡해지며, 공간제약으로 인해 모든 자료를 다 나타내지 못하는 문제가 있다. 특히 넘겨 받아야 될 자료가 둘 이상인 경우에는 또 다른 표현 방법을 모색해

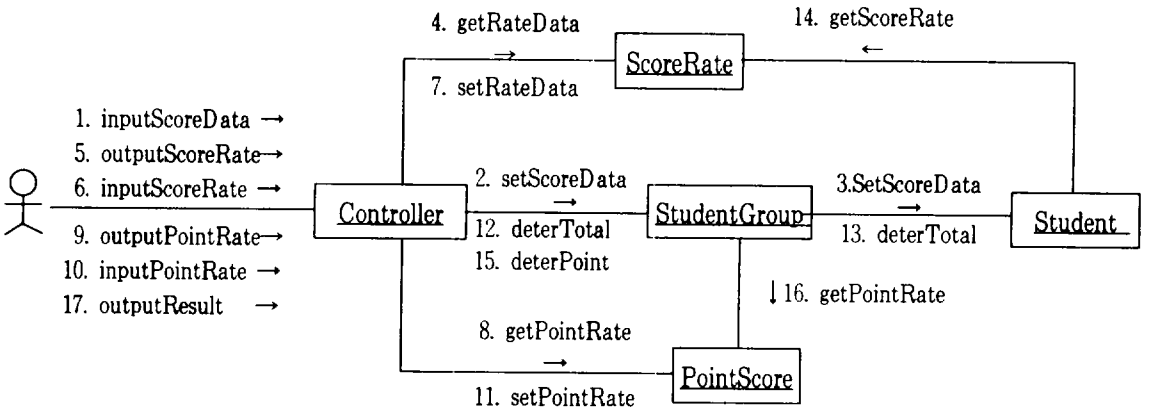

Fig. 5. example of collaboration-diagram.

야 하는 문제점이 있다.

상세화 클래스도는 객체지향 분석 단계에서의 클래스도에 각 클래스가 가지는 행위들을 추가한 것으로, 클래스들간의 관계는 나타낼 수 있어도 클래스들간에 전달되는 자료들은 나타내지 못한다.

III. 클래스도와 협력도의 개선 방안

소프트웨어를 개발하는데 있어서 자료와 행위 측면을 모두 살펴 보아야 한다. 하지만 앞 절에서 살펴본 바와 같이 기존의 객체지향 분석 및 설계 단계에서는 객체들간의 행위 요청에 대해서는 협력도를 통해서 나타낼 수 있었지만, 클래스간이나 객체들간에 전달되는 자료는 직접적으로 나타낼 수 없었다. 우리가 제어 중심의 소프트웨어를 개발할 때에는 클래스들간에 전달되는 자료가 별로 없기 때문에 별 문제가 없을 수 있으나, 자료 중심의 소프트웨어를 개발할 때에는 기존의 모형화 방법으로는 자료의 이동을 파악할 수 없기 때문에 부족한 점이 있다.

3.1 클래스도의 개선 방안

앞 장에서 살펴 보았던 클래스도는 기본적으로 자료 영역 분석을 위한 개체-관계도에 기반을 둔다. 개체-관계도는 데이터베이스 구축 이론에서 등장하는 기법으로서, 개체들간의 관계를 파악하는 데에 중점을 두며, 개체들간의 행위나 자료의 전달 등에 대해서는 관심을 가지지 않는다[8]. 이러한 개체-관계도를 기반으로 하여 클래스도를 구축하면, 클래스들간의 자료 전달을 나타내지 못하는 것이 당연하다.

개발해야 할 소프트웨어가 자료 처리를 중심으로 한다면, 자료들의 이동 경로를 파악하는 것은 필수적이다. 그렇지 않고는 개발될 소프트웨어의 전체적인 측면을 제대로 파악할 수 없기 때문에 소프트웨어를 개발 또는 유지보수하는데 있어서 어려움을 겪게 된다.

Fig. 6은 객체지향 소프트웨어 분석 단계에서 사용될 것으로서, 본 논문에서 제안하는 두 클래스간의 자료전송 관계를 나타내는 기본적인 자료지향 클래스도를 보이고 있다.

Class_i	sendData direction	Class_j
attributes_i	n m	attributes_j

Fig. 6. proposed basic data-oriented class-diagram.

'sendData'는 세부 자료항목들을 가질 수 있는 집합적 자료항목이며, 'direction'은 자료가 어느 방향으로 전달되는 지를 나타낸다. 나머지는 기존 클래스도에서 사용되는 것과 같다. 'n'과 m' 모두 1인 경우에는 생략할 수 있으며, 클래스들간의 상속 관계를 나타내는 것도 기존의 클래스도에서와 동일하게 나타내도록 한다.

제안된 클래스도에서는 기존 클래스도의 '관계'를 '전달_자료'와 '방향'으로 대체하였다. 이것은 기존의 클래스들간의 관계를 자료를 주고 받는 관계로 대체함을 의미한다. 클래스들간의 대부분의 관계는 자료의 전송과 수신, 그리고 메시지의 송수신을 통하여 나타낼 수 있다. 따라서 제안된 자료지향 클래스도는 기존의 클래스도를 대체하거나, 경우에 따라서는 클래스도 이외의 부가 모형화 기법으로 사용될 수 있다.

'Class_i'가 'Class_j'에게 자료를 전달하고, 그에 따라 'Class_j'가 'Class_i'에 자료를 반환할 경우에는 Fig. 7과 같이 ':'를 사용하여 'sendData'와 'returnData'를 구분하여 나타낼 수 있다.

3.2 객체지향 설계에서의 개선 방안

객체지향 설계에서도 객체간의 자료 관계를 나타내기 위해, 항상 협력도에 전달 자료와 반환 자료를 나타내도록 개선한다. Fig. 8은 객체지향 소프트웨어 설계 단계에서 사용될 것으로서, 본 논문에서 제안하는 두 객체간의 자료전송 관계를 나타내는 기본적인 협력도를 보이고 있다.

'argumentList'는 세부 자료항목들을 가질 수 있는 집합적 자료항목이며, 'direction'은 행위를 어느 방향으로 요청하는 지를 나타낸다. 'argumentList'의 세부 자료항목들은 ','로 구분하며, 나머지는 기존 협력도에서 사용되는 것과 같다.

'Class_i'가 'Class_j'에게 행위를 요청하면서 자료를 전달하고, 그에 따라 'Class_j'가 요청된 행위를 수행한 후에 'Class_i'에 자료를 반환할 경우에는 Fig. 9와 같이 ':'를 사용하여 'sendList'와 'returnList'를 구분하여 나타낼 수 있다.

한 가지 부가적인 것은 행위를 요청할 때에는 전달될 자료목록과 반환될 자료목록을 함께 기재하는 것이 바람직하며, 가능하다면 세부 자료항목들을 기재하는 것이 좋다. 하지만 공간적 제한 때문에 전달될 자료목록과 반환될 자료목록을 모두 기재하기 어려울 때에는, 전달될 자료항목 수와 반환될 자료항목 수를 대신 기재할 수도 있다. 이것은 Fig. 7에서도 동일한데, 이미 객체지향 분석의 클래스도에서 자료의 전달과 반환을 파악했기 때문에 직접 세부항목들을 나열하지 않아도 이해할 수 있다. 물론 전달될 자료목록과 반환될 자료목록을 자료사전을 사용하여, 간략하게 기재할 수도 있다. 전달되는 자료목록도 반환되는 자료목록도 없는 경우에는 () 속을 비워두며, 전달되는 자료목록은 있고 반환되는 자료목록은 없는 경우에는 ':' 이하가 생략된다. 그리고 전달되는 자료목록은 없고, 반환될 자료목록만 있는 경우에는 ':반환_목록'과 같이 나타낸다.

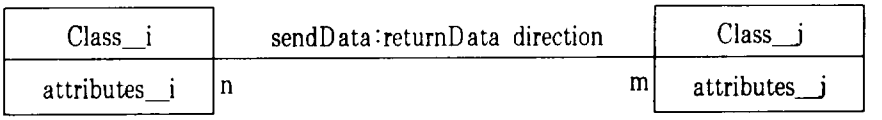

Fig. 7. proposed data-oriented class-diagram reflected data transfer.

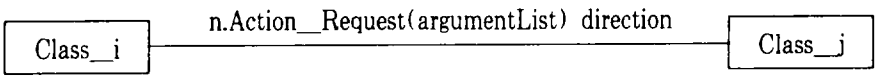

Fig. 8. proposed basic data-oriented collaboration-diagram.

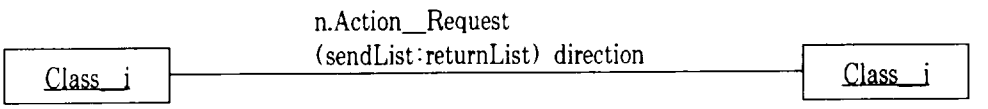

Fig. 9. proposed data-oriented collaboration-diagram reflected data transfer between two objects

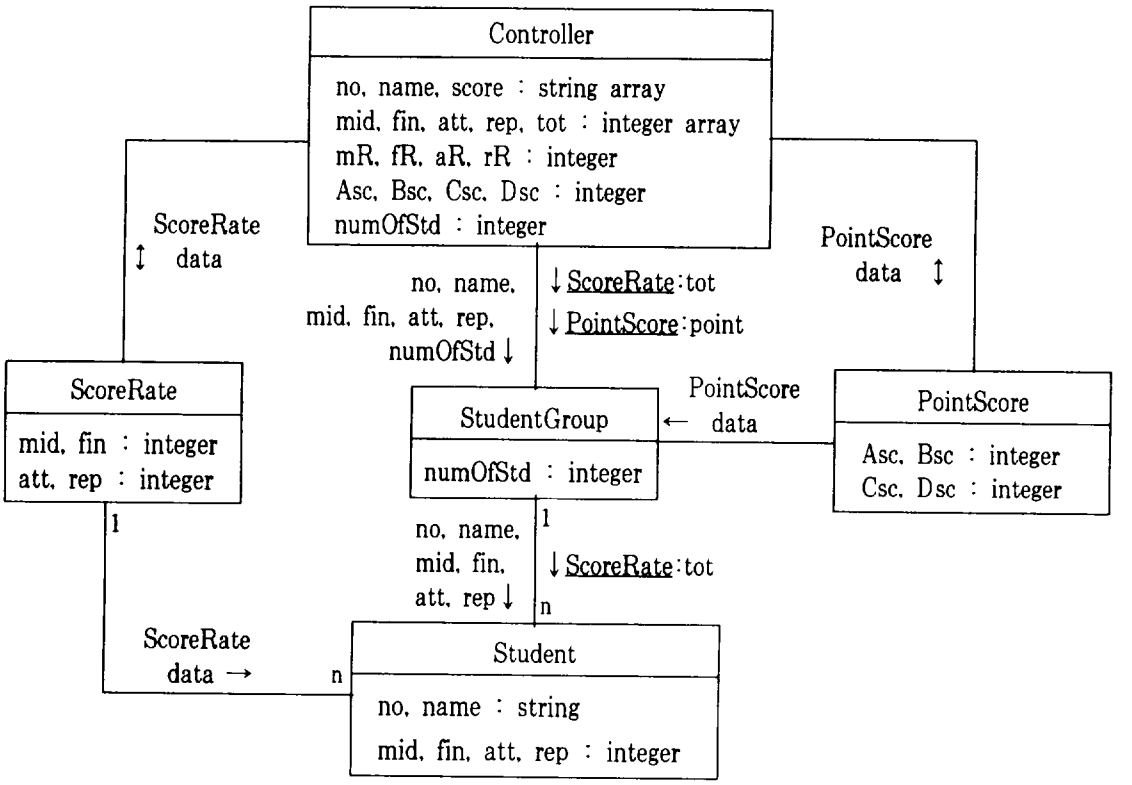

Fig. 10. example of proposed class-diagram.

Ⅳ. 결과 고찰

객체지향 분석 과정에서 Fig. 3과 같이 나타날 수 있는 클래스도를 제안된 클래스도로 변환하여 나타내면 Fig. 10과 같다. 클래스간에 전달되는 자료항목 또는 자료항목 집단을 전달 방향과 같이 나타내었기 때문에, 클래스간에 전달되는 자료들을 쉽게 파악할 수 있다. 클래스간의 관계를 파악하는 것 보다는 자료의 전달 정보를 파악하는 것이 더 쉽기 때문에 개체관계도와 유사한 클래스도 보다는 제안된 클래스도가 더 쉽게 도출될 수 있으며, 직접적인 정보를 제공하는 우수성이 있다.

예를 들어 'StudentGroup' 클래스와 'Student' 클래스의 관계는 Fig. 3에서 나타난 것에 의하면 'StudentGroup' 클래스 하나가 n 개의 'Student' 실체들을 '포함하다'는 관계를 가지게 되지만, 여기에서는 'StudentGroup' 클래스가 'Student' 클래스에게 'no, name, mid, fin, att, rep'를 제공하는 것과 'ScoreRate' 객체를 넘겨주고 'tot'를 반환받는 것으로 나타나기 때문에 기존의 클래스도 보다 제안된 클래스도가 더욱 확실한 관계를 나타내며, 이해를 용이하게 한다.

한편 Fig. 11에는 객체지향 설계 과정에서 Fig. 5와 같이 나타낼 수 있는 협력도를 제안된 협력도로 변환하여 나타내었다. 공간적인 제약 때문에 자료항목들을 다 나타낼 수 없기 때문에, 전달되고 반환되는 자료항목들의 수를 대신 나타내었다. 이와 같이 나타내면, 객체들간에 행위를 요청하면서 자료들이 전달되고 반환되는 관계를 쉽게 파악할 수 있다. 객체간의 단순한 행위 요청보다는 전달 자료 및 반환 자료를 추가함으로써 객체간의 관계를 파악하는데 더 유용하다. 특히 기존의 협력도는 반환 자료가 둘 이상일 때에는 표현하기가 어려웠지만, 제안된 협력도에서는

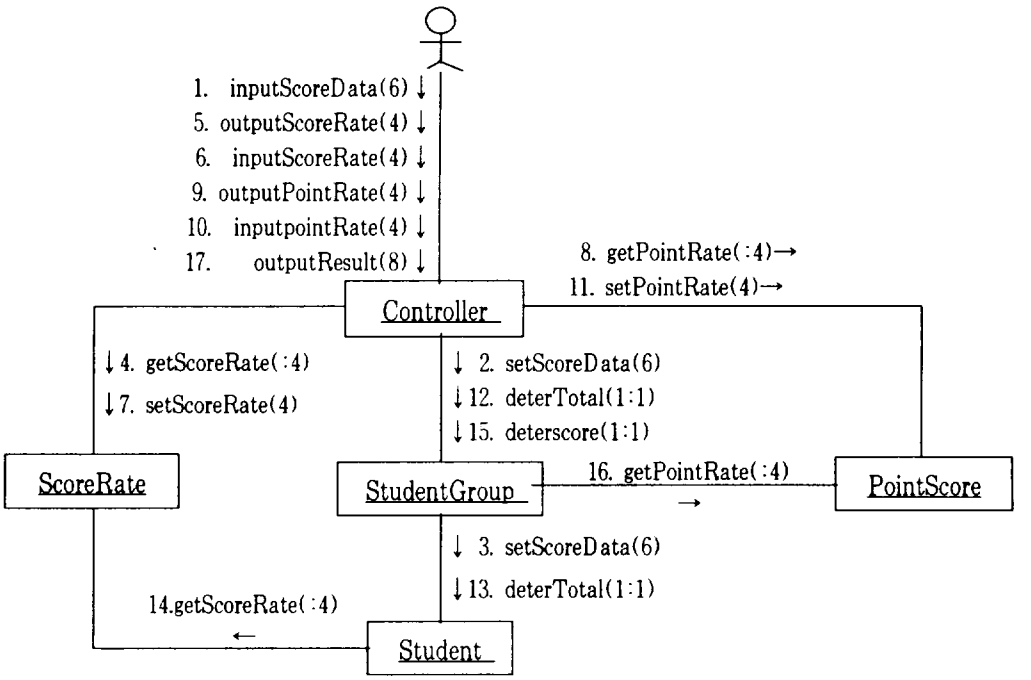

Fig. 11. example of proposed collaboration-diagram.

쉽게 나타낼 수 있었다.

예를 들어 'StudentGroup' 객체와 'Student' 객체의 관계는 Fig. 5에서 나타난 것에 의하면 'StudentGroup' 객체가 'Student'에게 'setScoreData'와 'deterTotal' 행위를 요청하는 관계를 가지게 되지만, 여기에서는 'StudentGroup' 객체가 'Student' 객체에게 6 개의 자료를 보내면서 'setScoreData'를 요청하는데, Fig. 10을 참조하면 이것은 'no, name, mid, fin, att, rep'를 제공하면서 'setScoreData'를 요청한다는 것을 알 수 있다. 또한 1 개의 자료를 보내면서 'deterTotal' 행위를 요청하고 1 개의 자료를 반환받는데, Fig. 10을 참조하면 이것은 'ScoreRate' 객체를 넘겨주고 'deterTotal' 행위를 요청하고 'tot'를 반환받는 것을 알 수 있다. 따라서 기존의 협력도 보다 제안된 협력도가 객체들간의 관계를 더욱 확실하게 알 수 있으며, 이해를 용이하게 한다.

V. 결 론

본 논문은 자료처리를 중심으로 하는 소프트웨어에 대한 객체지향 분석 단계에서 나타나는 모형 중에는 클래스간의 자료전달 정보를 파악할 수 없는 것과 객체지향 설계 단계에서 나타나는 모형 중에는 객체간의 자료전달 정보를 제대로 파악하지 못하는 것을 개선하고자 하였다.

객체지향 분석 단계에서는 기존의 클래스도를 개선하여 클래스간의 자료전달 정보를 파악할 수 있는 새로운 클래스도를 제안하였으며, 객체지향 설계 단계에서는 기존의 협력도를 개선하여 객체간에 행위를 요청하면서 전달되고 반환되는 자료들의 관계를 파악할 수 있도록 새로운 협력도를 제안하였다.

제안된 클래스도와 협력도를 실제 응용에 적용하여, 소프트웨어 부품 사이에서의 자료흐름을 파악하는데 도움이 되는 것을 보였다. 제안된 클래스도는 기존 클래스도에 비해 클래스간의 보다 직접적인 관계를 보이며, 클래스간의 관계를 파악하는데 보다 용이하였다. 그리고 제안된 협력도는 두 객체간의 관계를 보다 직접적으로 나타내며, 특히 여러 자료들을 반환하는 경우에 기존의 협력도 보다 잘 표현할 수 있었다.

기존의 클래스도와 협력도에 대해서 컴퓨터 지원 도구가 존재하는 것 처럼, 제안된 클래스도와 협력도에 대한 컴퓨터 지원 도구를 개발하면 아주 유용하게 이용될 수 있다.

참고문헌

1) G. Radin. 1996. *Object Technology in Perspective.* IBM Systems Journal 35 (NO.2). pp.124-26

2) E.H.Khan. M. Al-A'All. and M.R.Girgis. 1995. *Object-Oriented Programming for Structured Procedural Programming.* IEEE Computer 28. pp.48-57

3) B. Meyer. 1997. *Object-Oriented Software Construction.* Second Edition. Prentice Hall

4) G.Booch. 1994. *Object-Oriented Analysis and Design with Application.* Second Edition. Benjamin/Cummings.

5) J.Martin and J.J.Odell. 1992. *Object-Oriented Analysis and Design.* Prentice Hall

6) R.C.Lee and W.M Tepfenhart. 1997. *UML: A Practical Guide to Object-Oriented Development.* Prentice Hall

7) C.Larman. 2001. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design.* Second Edition Prentice Hall

8) A.Silberschatz. H.F.Korth. 2001. *Database System Concepts.* McGraw Hill