재사용 소프트웨어 컴포넌트의 Statemate 모듈화

김창진, 최진영
고려대학교 정형기법연구실
e-mail : {cjkim, choi}@formal.korea.ac.kr

Statemate-modulization of reusable software component

Chang Jin Kim, Jin-Young Choi
Formal Methods Lab., Korea University

요 약

Ada는 객체지향 특성과 소프트웨어 모듈화 및 일반화 메커니즘을 통해 프로그래밍 언어 차원에서 소프트웨어의 재사용성을 제공하고 있다. 그러나 특정 언어 자체를 이용하는 것만으로 소프트웨어의 재사용이 확보되지는 않으며 설계자가 컴포넌트간의 유기적인 관계를 파악하고 재사용을 통해 얻을 수 있는 장.단점을 판단할 수 있어야 한다. 본 논문은 재사용 가능한 소프트웨어 객체와 인터페이스를 식별하고 재사용 모듈의 설계에 필요한 요소들을 파악함과 동시에 소프트웨어 설계 시 적용할 수 있는 코딩 패턴을 제시한다. 또한 이들을 보다 효율적인 설계도구인 Statemate에서 재사용 모듈로 활용할 수 있도록 기존 generic chart의 한계를 고려한 확장 개념의 재사용 모듈을 작성한다.

1. 서 론

Ada는 신뢰성, 재사용성, 안전성, 이식성 등 소프트웨어 공학에서 거론되는 대부분의 이슈들과 연관된 언어적 속성을 포함한다 [1]. 본 논문은 그 중 Ada의 재사용성에 초점을 맞추고 있다.

Ada는 프로그래밍 언어 수준에서 신뢰성을 확보한 것으로 알려져 있는데 이는 Ada가 국제 표준으로서 가지는 규칙뿐만 아니라, 엄격한 자료형 검사, 컴파일 단계에서의 에러 검출 등과 같은 언어적 특성 때문이기도 하다. 또한 Ada는 소프트웨어를 모듈 단위로 관리하는데 적합한 언어이며 재사용 가능한 모듈은 소프트웨어의 생산성을 극대화시키는 방안이다.

본 논문의 이전 연구를 통해 전통적인 Ada 시스템의 구조적 또는 기능적 설계 방법론이 Statemate [2]의 설계 모델로 큰 장애 없이 변환 가능하다는 판단을 할 수 있었는데 이러한 가능성은 실제 개발현장에서 두 가지 측면의 중요성을 가진다.

첫째, Statemate와 같은 설계도구를 사용했을 경우 자연어 명세나 설계 레벨의 코딩보다 개발자의 의도를 보다 명확하게 표현할 수 있다. 이는 궁극적으로 개발 일정의 단축을 의미하는데 Statemate와 같은 그래픽이 강조된 언어를 사용했을 경우 시스템의 기능성과 동작에 대한 이해 또한 순조롭다.

두 번째는 신뢰성이다. Statemate의 경우 비록 간단한 시뮬레이션이나 'check model' 기능을 수행하더라도 설계단계에서의 'completeness' 및 설계 모듈간의 충돌 여부를 점검한다. 임무 / 안전 필수 시스템의 경우, 효과적인 툴 기반의 개발 프로세스는 편리성의 문제가 아니라 소프트웨어 품질과 직결되는 요건이다.

2 장에서는 지금까지 간략히 언급한 Ada 와 Statemate 의 속성들을 고려하여 재사용 가능한 Ada 컴포넌트 설계를 위한 기본 개념과 가이드라인을 기술한다. 3 장에서는 실제 설계 단계에서 재사용 컴포넌트들을 보다 효율적으로 사용하기 위한 도구 및 적용 방법을 제시하는데, 2 장에서 작성된 Ada 재사용 컴포넌트의 작성 개념을 Statemate 의 설계 모듈에 적용시키기 위한 재사용 모듈화 작업을 수행한다. 이를 위해 Statemate 의 generic chart 를 내포한 확장된 개념의 재사용 단위를 제시한다.

2. 분할 및 재사용 정책

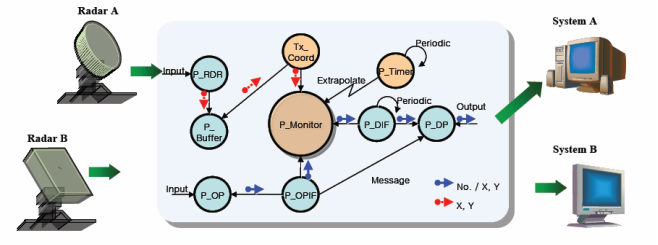
2.1 시스템 분할

객체지향, 구조적 설계, 기능적 분해 등 설계방법론에 관계없이 재사용 컴포넌트를 식별하기 위해서는 시스템을 서브시스템과 그 구성 컴포넌트로 분해할 수 있어야 한다. 물론 개략설계 단계에서 식별된 컴포넌트가 상세설계 단계에서는 보다 세부적인 컴포넌트로 재분해될 수도 있다. 본 논문에서 서브시스템이나 CSCI (Computer Software Configuration Item) 혹은 CSC (Computer Software Component) 수준의 재사용을 논하지는 않는다. 아직은 그 수준의 컴포넌트 규모가 너무 크고 실질적인 재사용 단위가 되기 어렵기 때문이다. 본 논문의 논지는 가능한 한 설계의 조기 단계에서 CSC 이하의 컴포넌트 즉, CSU (Computer Software Unit) 또는 그 구성 객체와 객체간 인터페이스를 재사용 가능하도록 설계하는 방법에 있다.

비록 소규모의 재사용 컴포넌트라고 하더라도 이는 프로젝트 수준의 설계 정책과 관련된 문제이다.

그림 1에서와 같이 레이더 A 및 시스템 A와 연동

중인 기존 ATCS (Air Traffic Control System) [3] 시스템이 기존과 다른 형태의 레이더 B 및 시스템 B에 대해 추가적인 연동을 해야 하는 경우를 생각해보자.



(그림 1) 레이더 및 외부시스템과의 ATCS 연동 개념도

이 때 기존 시스템에 충격과 변경요구를 최소화하면서 신규요구를 충족하기 위한 방안을 강구해야 하는데 재사용을 고려하지 않은 설계인 경우 완전히 다른 시스템을 다시 구상해야 하는 문제도 발생할 수 있다. 다음 장에서는 재사용 객체의 설계를 위한 기본 정책에 대해 기술한다.

2.2 객체 및 인터페이스의 재사용

본 논문에 인용된 ATCS는 레이더의 입력을 처리하여 항공기의 위치정보를 포함하는 'plot'이라는 특수한 데이터객체를 생성한다. 연동하는 레이더 별로 ATCS로 보내는 자료의 형태가 다를 수 있는데 이 경우 각 레이더 형태 별로 서로 다른 자료형을 가지는 plot 객체를 별도로 생성할 수도 있다. 그와 달리 아래 코드와 같이 모든 레이더 자료 처리에 필요한 데이터를 하나의 객체에 포함시킴으로써 두 가지 레이더 타입에 모두 적용 가능한 데이터 객체를 만들 수도 있는데 이는 가장 단순한 형태의 재사용 객체로 볼 수 있다.

```
type Plot_For_Both_Radars is
  record
    X_Position : Coordinate_Type;
    Y_Position : Coordinate_Type;
    -- other fields
    DATA_1 : Common_Data_Type;
    DATA_2 : Common_Data_Type;
    DATA_3 : A_Spefific_Type;
    --type A specific data
    DATA_4 : B_Spefific_Type;
    -- type B specific data
  end record;
```

(예제 1) 통합 자료형 객체

그러나 이러한 자료구조가 자주 사용될 경우 필요 이상의 공간 낭비뿐 아니라 필드 개수가 과다하거나 데이터의 변경 요구가 잦아질 경우 매우 비효율적인 설계작업을 유발한다. 또한 문서화에 각별한 노력을 기울이지 않은 한 변경 추적이 매우 어렵다.

특정 레이더에만 해당하는 고유 데이터 필드가 지나치게 많은 경우나 그러한 필드가 다른 레이더에 대해 상호 배타적인 경우는 다음과 같은 선택적 옵션을 가지는 공유객체를 고려해볼 수 있다

```
type Radar_Type is (A, B);

type Plot_For_Radar(Radar: Radar_Type) is
  record
    X_Position : Coordinate_Type;
    Y_Position : Coordinate_Type;
    -- other fields
    DATA_1 : Common_Data_Type;
    DATA_2 : Common_Data_Type;
    case Radar is
      when A =>
        DATA_3 : A_Spefific_Type;
      when B =>
        DATA_4 : B_Spefific_Type;
    end case;
  end record;
```

(예제 2) 선택적 자료형 객체

그러나 위 두 가지 모두 정도의 차이가 있을 뿐, 변경요구 발생 시 그 객체 전체가 영향을 받게 되며, 특히 객체가 글로벌로 선언되었을 경우 변경 작업이 미치는 영향이 클 뿐 아니라 작업 자체도 복잡하다.

다음의 예와 같이 모든 레이더 환경에 공통적인 데이터 필드만으로 구성된 객체의 유형을 생각해보자.

```
type Basic_Plot is
  record
    X_Position : Coordinate_Type;
    Y_Position : Coordinate_Type;
    -- other fields
    DATA_2 : Common_Data_Type;
    DATA_3 : Common_Data_Type;
  end record;
```

(예제 3) 공통 자료형 객체

공통객체가 선언되었다면 각 레이더의 고유 데이터 필드만을 정의한 부분 역시 필요한데 아래 코드에 선언된 자료형 Fields_For_A는 레이더 A 해당 자료를 처리하는 컴포넌트에 포함되어야 하고 Fields_For_B 의 경우 레이더 B의 자료를 처리하는 컴포넌트에 포함되어야 한다.

```
type Fields_For_A is
  record
    DATA_3 : A_Spefific_Type;
  end record;

type Fields_For_B is
  record
    DATA_4 : B_Spefific_Type;
  end record;
```

(예제 4) 고유 자료형 객체

위와 같은 plot 객체를 사용하는 어떤 서브프로그램을 작성하는 경우, 만약 plot 의 공통객체만을 처리하는 것이라면 그 서브프로그램은 완전한 재사용성을 확보할 수 있다. 그렇지 않을 경우는 공통부분과 고유부분을 모두 처리하도록 작성해야 하는데 이 때는 재사용성을 위해 다음과 같은 방법을 사용할 수 있다.

작성하고자 하는 서브프로그램에서 plot 의 고유 필드를 처리하는 부분을 별도의 섹션으로 분리하고 다음과 같은 일반화(generic) 프로그램을 선언한다.

```
generic
  type Other_Fields is private;
```

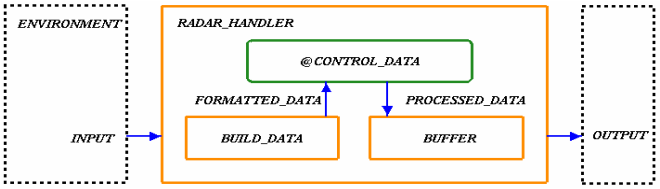
```
with procedure Process
  (These_Other_Fields : Other_Fields);

procedure Process_Generic
   (This_Plot : Basic_Plot;
     And_These_Fields : Other_Fields);
```

(예제 5) 일반화(generic) 서브프로그램의 명세

(예제 5)에서 선언된 프로시저 Process_Generic의 구현부(body)는 먼저 Basic_Plot 타입으로 선언된 입력 파라미터 This_Plot의 모든 필드를 처리한다. 그리고 Other_Fields 타입으로 선언된 고유한 필드들에 대한 처리는 generic formal procedure로 선언된 Process를 호출하여 수행하도록 한다. Generic formal procedure Process와 함께 Other_Fields가 generic formal parameter로 선언되었으므로 어떤 환경에 대해서도 이 generic 컴포넌트는 재사용 가능하다.

3 Statemate의 재사용 모듈 설계

실제 운영 중인 ATCS 사례를 볼 때, 그 특성상 다수의 서로 다른 기종의 레이더와 다양한 인터페이스를 구축한 경우가 많다. 본 연구에서는 (그림 1)에서와 같이 ATCS가 2개의 레이더로부터 자료를 수신한다고 가정 하고 서로 다른 레이더에 대해 인터페이스 모듈을 재사용하기 위해 (그림 2)와 같이 activity chart GENERIC_HANDLER를 generic chart로 작성한다.

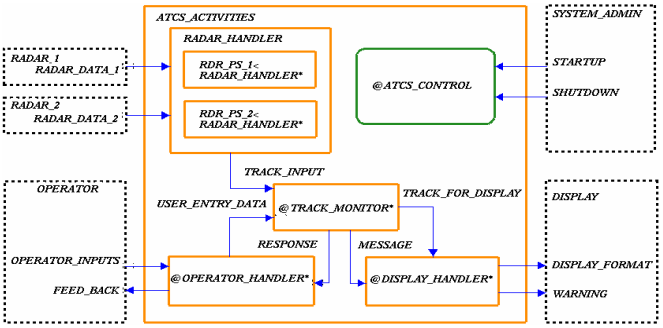


(그림 2) Generic chart RADAR_HANDLE

Statemate의 generic chart의 인스턴스는 기본적으로 generic chart의 formal parameter와 동일한 데이터 타입을 사용해야 한다. 이러한 기본 규칙을 고려할 때 (그림 2)에서 선언된 INPUT의 가장 단순한 형태는 역시 (예제 1)과 같이 모든 레이더 타입의 데이터를 포함하는 통합 자료형일 것이다.

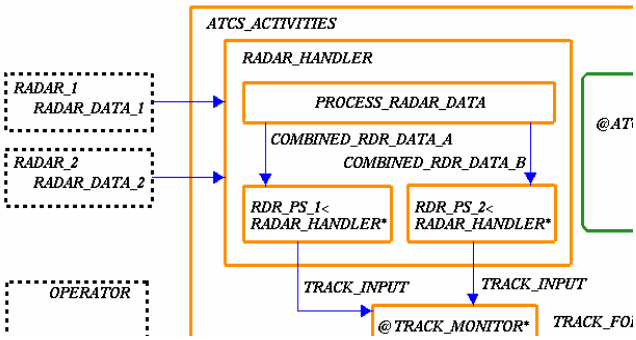
그러나 이 경우 공간적 낭비의 문제는 무시하더라도 Statemate에서 사용하기 곤란한 근본적 문제가 있다. 즉, 레이더 타입에 따라 실제로는 actual parameter가 formal parameter의 일부 필드를 사용하지 않게 되고 Statemate는 이를 오류로 처리하게 된다. 모든 레이더가 동일한 자료형을 데이터를 송신한다면 위에서 정의한 generic chart 자체가 완전한 재사용 모듈로 활용될 수 있으나 아래 (그림 3) 에서 DATA_1과 DATA_2가 서로 다른 데이터 필드를 가진다고 가정하면 actual parameter와 formal parameter의 자료형 불일치에 따라 (그림 2)의 generic chart는 재사용할 수 없다.

(그림 3)의 activity RADAR_HANDLER 내부에 사용된 generic 인스턴스들이 재사용 모듈로서 의미를 가지는 경우는 입력 데이터 DATA_1과 DATA_2의 자료형이 동일함과 동시에 generic chart의 formal parameter와 자료형이 일치할 때뿐이다. 뿐만 아니라 그렇지 않을 경우 위 차트는 오류를 포함한 모델로 처리되어 정상적 작동 자체가 불가능하다.



(그림 3) ATCS Top Level Activity Chart

(그림 4)와 같이 재정의된 RADAR_HANDLER는 generic 인스턴스인 RDR_PS_1과 RDR_PS_2 외에 PROCESS_RADAR_DATA라는 새로운 activity chart를 포함하고 있다. 이 차트의 역할은 외부 환경으로부터 들어오는 다양한 형식의 데이터를 generic chart가 처리할 수 있도록 formal parameter 형식에 맞도록 변환시키는 것이다.



(그림 4) Generic 인스턴스를 포함한 재사용 모듈

(그림 4)에서 activity RDR_PS_1와 RDR_PS_2는 동일한 generic chart의 인스턴스들이므로 이들에 대한 입력 COMBINED_RDR_DATA_A, COMBINED_RDR_DATA_B 는 동일한 formal parameter에 바인딩 되어야 한다. 이 때 formal parameter로서 RADAR_DATA_1과 RADAR_ DATA_2의 모든 데이터 필드를 포함하는 하나의 통합자료형 데이터 객체를 사용할 수 있다. 그러나 위에서 설명한 바와 같이 generic 인스턴스들이 formal parameter와 다른 타입의 데이터를 직접 처리할 수는 없으므로 이들 인스턴스들과 RADAR_1, RADAR_2는 직접 연결되어서는 안된다. 또한 현재의 Statemate 기능상으로는 선택적 필드를 가지는 레코드를 Data Item으로 선언할 수 없으므로 (예제 2)와 같은 선택적 자료형을 formal parameter로 선언할 수 없다. 따라서 선택적 자료형이 가지는 효과를 얻기 위해 새로운 activity가 요구되는데 (그림 4)에서는PROCESS_RADAR_DATA가 그 역할을 수행한다.

PROCESS_RADAR_DATA는 입력 데이터의 유형에 상관없이 generic chart의 formal input parameter와 동일

한 유형의 출력 데이터를 생성해야 한다.

이 과정에서 (그림 4)의 각 모듈 별로 입력 또는 출력하는 데이터의 유형을 정리하면 다음과 같다.

먼저 각 레이더에서 RADAR_HANDLER로 보내는 데이터는 기본적으로 X, Y 및 DATA_1과 DATA_2를 BASIC_PLOT 필드에 포함하고 있다. 그리고 각 데이터의SPECIFIC_DATA에서 DATA_FOR_RDR_A 타입은 DATA_3, DATA_FOR_RDR_B타입은 DATA_4를 각각의 필드로 포함하고 있다.

< RADAR_1 to RADAR_HANDLER>

Field Name	Field Data Type
BASIC_PLOT	COMMON_DATA_TYPE
SPECIFIC_DATA	DATA_FOR_RDR_A

< RADAR_2 to RADAR_HANDLER>

Field Name	Field Data Type
BASIC_PLOT	COMMON_DATA_TYPE
SPECIFIC_DATA	DATA_FOR_RDR_B

다음으로 각 레이더로부터 서로 다른 데이터 유형을 입력 받은 PROCESS_RADAR_DATA는 generic 인스턴스인 RDR_PS_1 또는 RDR_PS_2에 formal parameter와 동일한 형식의 데이터를 생성해서 내보내는데 그 형태는 다음과 같다.

<PROCESS_RADAR_DATA to RDR_PS_1 / RDR_PS_2>

Field Name	Field Data Type
RADAR_TYPE	RADAR_ID_TYPE
BASIC_PLOT	COMMON_DATA_TYPE
SPEFIFIC_DATA	RDR_SPECIFIC_DATA

<SPECIFIC_DATA : RDR_SPECIFIC_DATA>

Field Name	Field Data Type
SPECIFIC_DATA_KEY	Integer min=1 max=4
SPEFIFIC_DATA_VALUE	Integer min=1 max=100

이 때 SPECIFIC_DATA는 각 레이더의 고유 데이터 유형을 처리하기 위해 SPECIFIC_DATA 필드를 가지는데 RADAR_1에서 데이터를 입력 받은 경우 SPECIFIC_DATA의 SPECIFIC_DATA_KEY 필드에 '3'을 세팅하고 SPECIFIC_ DATA_VALUE에 해당 정수값을 저장함으로써 generic chart내부적으로는 BUFFER에 저장되는 레코드 PROCESSED_DATA의 DATA_3 필드를 업데이트하게 된다.

Generic 인스턴스를 거쳐 TRACK_MONITOR로 최종 전달되는 TRACK_INPUT은 다음과 같이 TRACK_ID필드와 RADAR_1 및 RADAR_2의 모든 데이터를 통합한 DATA 필드로 구성된다.

<RDR_PS_1 / RDR_PS_2 to TRACK_MONITOR>

Field Name	Field Data Type
TRACK_ID	TRACK_NUMBER_TYPE
DATA	COMBINED_DATA_TYPE

<DATA : COMBINED_DATA_TYPE>

Field Name	Field Data Type
X	Integer min=0 max=1000
Y	Integer min=0 max=1000
DATA_1	Integer min=0 max=1000
DATA_2	Integer min=0 max=1000
DATA_3	Integer min=0 max=100
DATA_4	Integer min=0 max=9

이상과 같은 일련의 과정을 통해 사용된 데이터 유형을 볼 때, 외부환경에서 들어오는 고유한 데이터는 generic chart를 포함한 재사용 모듈을 거치면서 (예제 3) 및 (예제 4)와 같이 분리된 공통자료와 고유자료의 형태를 거쳐 최종적으로 (예제 1)과 같은 통합자료형 데이터를 형성하게 된다. 이 때 재사용 모듈 내부에 사용된 PROCESS_RADAR_DATA는 (예제 2)에서와 같이 조건 별로 데이터를 구분하는 역할과 (예제 5)에서와 같이 공통 및 고유데이터를 처리하여 원하는 출력 형태로 내보내는 서브프로그램의 역할을 수행해야 한다.

4. 결 론

본 논문을 통해 Ada 프로그램의 설계에서 고려해야 할 객체와 그 인터페이스의 재사용 개념을 소개하고 설계 언어의 코딩 패턴 및 Statemate와 같은 설계도구에서 사용 가능한 재사용 모듈의 작성 방법을 알아보았다.

재사용 가능한 객체는 다수의 프로그램에서 최소한의 조정작업만을 거쳐 사용 가능해야 하며 이 때 자원의 낭비, 액세스 빈도, 향후의 환경 변화에 대해도 가장 효과적으로 대응할 수 있어야 한다 [4].

소프트웨어 컴포넌트를 재사용 하는 것은 생산작업의 효율성뿐 아니라 검증된 컴포넌트의 사용을 통해 신뢰성과 안전성을 보장을 받기 위해서이기도 하다. 그런 면에서 Statemate와 같이 설계의 검증이 가능한 도구에서의 설계모듈 재사용은 효율성과 안전성을 향상시키는 효과적인 방법이다.

단, Statemate의 generic chart가 좋은 재사용 메커니즘이기는 하나 formal parameter에서 자료형 자체를 매개변수화(type parameterization)할 수 없는 한계가 있다. 이를 극복하기 위해서는 본 논문에서 제시한 바와 같이 보다 확장된 재사용 모듈의 설정이 필요하다.

참고문헌

[1] Michael A. Smith, "Object-oriented Software in Ada95 Second Edition", Mcgraw-Hill, 2001

[2] David Harel, "Modeling Reactive Systems with State charts: The Statemate Approach", I-Logix, 1999

[3] Kjell Nielsen, "Designing Large Real-Time Systems with Ada", Hughes Aircraft Company, 1987

[4] Colin Atkinson, "Object-Oriented Reuse, Concurrency and Distribution", Addison -Wesley, 1991