소프트웨어 설계 모듈의 재사용을 위한 Statemate 일반화 차트의 확장

김창진° 최진영
고려대학교 컴퓨터학과 컴퓨터이론 및 정형기법 연구실
{cjkim°, choi}@formal.korea.ac.kr

Extension of Statemate Generic Chart for the Reuse of Software Design Module

Changjin Kim° Jin-Young Choi
Computer Theory & Formal Method Lab, Department of Computer Science, Korea University

요 약

Statemate는 정형이론 및 Statechart를 기반으로 하는 설계도구로서 시스템의 기능적, 구조적 분해를 통해 계층적으로 모듈화된 설계 단위를 작성할 수 있도록 한다. 또한 동일한 기능을 수행하는 설계모듈을 반복적으로 재사용하기 위한 일반화 차트 메커니즘을 지원하는데 본 논문에서는 Statemate가 지원하는 Ada 언어의 재사용 컴포넌트 개념을 적용하여 일반화 차트가 처리할 수 있는 형식매개변수의 자료형 제약을 완화시킨 확장된 개념의 재사용 설계모듈을 제시한다.

1. 서 론

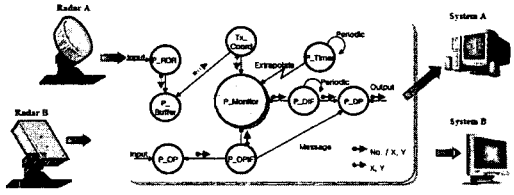
Ada는 전통적으로 존재했던 엄격한 자료형과 재사용성 간의 비양립성 문제를 해결한 언어이며[1], 그러한 특성을 반영한 구체적인 도구로서 Ada의 일반화 단위(Generic Unit)를 들 수 있다. 즉, 알고리즘은 동일하지만 처리하는 자료의 유형이 서로 다른 프로그램에 대해 형식매개변수(Formal Parameter)의 자료형을 일반화자료형(Generic Type)으로 선언함으로써 실제매개변수(Actual Parameter)의 자료형과 무관하게 동일한 알고리즘을 수행할 수 있도록 한 것이다.

본 연구에서는 실질적인 소프트웨어 컴포넌트의 재사용을 위해 일반화 단위 뿐 아니라 개념적으로 가능한 다양한 Ada 재사용 컴포넌트의 개념을 Statemate[2]의 설계 모듈에 적용하는 접근방법을 제시한다. 단, Statemate가 제공하는 재사용 메커니즘, 즉 일반화차트(Generic Chart)의 고정적인 형식매개변수 자료형 문제 해결을 위해 재사용 모듈의 범위를 확장하기로 한다.

Ada가 설계언어(Program Design Language)로서의 기능에 충실함에도 불구하고 Statemate와 같은 별도의 설계도구를 사용한 것은 그래픽이 강조된 Statemate의 정형명세가 자연어 명세나 설계언어 수준의 코딩보다 개발자의 의도를 보다 명확하게 표현할 수 있으며, 또한 신뢰성의 확보를 위해서도 효과적이기 때문이다. 이러한 특성들은 궁극적으로 모호성과 불완전성의 배제를 통한 품질향상과 개발 일정의 단축을 의미한다.

2. Ada 재사용 컴포넌트의 개념 및 구현

(그림 1)에서와 같이 레이더 A 및 시스템 A와 연동중 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 방안이 필요하다.

본 논문에 인용된 ATCS는 레이더의 입력을 처리하여 항공기의 위치정보를 포함하는 'Plot'이라는 특수한 데이터객체를 생성한다. 연동하는 레이더 별로 ATCS로 보내는 자료의 형태가 다를 경우 각 레이더 형태 별로 서로 다른 자료형을 가지는 Plot 객체를 별도로 생성할 수도 있다. 그와 달리 아래 (예제 1)과 같이 모든 레이더 자료처리에 필요한 데이터를 하나의 객체에 포함시킴으로써 두 가지 레이더 타입에 모두 적용 가능한 통합 자료형 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;
    DATA_1 : Common_Data_Type;
    DATA_2 : Common_Data_Type
    DATA_3 : A_Specific_Type; --type A specific data
    DATA_4 : B_Specific_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 설계 작

업을 유발하며 변경추적 또한 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;
    DATA_1 : Common_Data_Type;
    DATA_2 : Common_Data_Type;
    case Radar is
      when A =>
        DATA_3 : A_Specific_Type
      when B =>
        DATA_4 : B_Specific_Type
    end case;
  end record;
```

(예제 2) 선택적 자료형 객체

그러나 위 두 가지 모두 정도의 차이가 있을 뿐, 변경 요구 발생 시 그 객체 전체가 영향을 받게 된다.

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

```
type Basic_Plot is
  record
    X_Position : Coordinate_Type;
    Y_Position : Coordinate_Type;
    DATA_1 : Common_Data_Type;
    DATA_2 : Common_Data_Type;
  end record;
```

(예제 3) 공통 자료형 객체

공통객체를 선언한 후에는 (예제 4)와 같이 각 레이더의 고유 데이터 필드를 정의한 객체를 선언해야 한다.

```
type Fields_For_A is
  record
    DATA_3 : A_Specific_Type;
  end record;
type Fields_For_B is
  record
    DATA_4 : B_Specific_Type
  end record;
```

(예제 4) 고유 자료형 객체

위와 같은 Plot 객체를 사용하는 서브프로그램을 작성하는 경우, 만약 Plot의 공통객체만을 처리하는 것이라면 그 서브프로그램은 완전한 재사용성을 확보할 수 있다. 그렇지 않을 경우 공통부분과 고유부분을 모두 처리하도록 작성해야 하는데 이때는 작성하고자 하는 서브프로그램에서 Plot의 고유 필드를 처리하는 부분을 별도의 섹션으로 분리하고, 다음과 같은 일반화(Generic) [4] 프로그램을 선언한다.

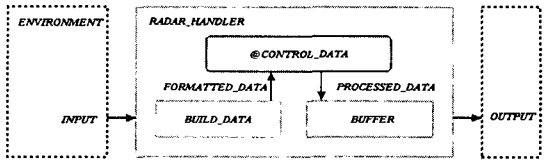
```
generic
  type Other_Fields is private;
  with procedure Process (These_Other_Fields :
    Other_Fields);
procedure Process_Generic (This_Plot : Basic_Plot;
                           And_These_Fields : Other_Fields);
```

(예제 5) 일반화(generic) 서브프로그램의 명세

일반화 형식프로시저인 Process와 함께 Other_Fields가 일반화 형식매개변수로 선언되었으므로 어떤 환경에 대해서도 이 일반화 컴포넌트는 재사용 가능하다.

3. Statemate 일반화차트(Generic chart) 설계

본 장에서는 실제 운영 중인 ATCS 사례를 고려하여 (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 일반화차트로 작성한다.

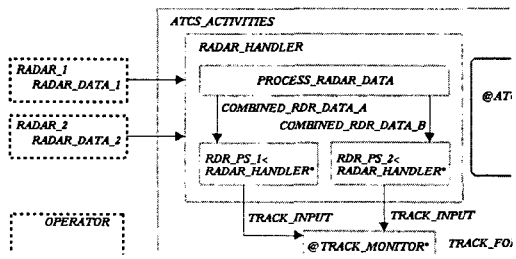


(그림 2) Generic chart RADAR_HANDLE

Statemate의 Generic Chart의 인스턴스는 기본적으로 Generic Chart의 형식매개변수와 동일한 데이터 타입을 사용해야 한다. 이러한 기본 규칙을 고려할 때 (그림2)에서 선언된 INPUT의 가장 단순한 형태는 (예제 1)과 같이 모든 레이더 타입의 데이터를 포함하는 통합 자료형이 될 것이다.

그러나 이 경우 공간적 낭비의 문제는 무시하더라도 Statemate에서 사용하기 곤란한 근본적 문제가 있다. 즉, 레이더 타입에 따라 실제로는 실제매개변수가 형식매개변수의 일부 필드를 사용하지 않게 되고 Statemate는 이를 오류로 처리하게 된다. 아래(그림 3) 에서 DATA_1과 DATA_2가 서로 다른 데이터 필드를 가진다고 가정하면 실제매개변수와 형식매개변수의 자료형 불일치에 따라 (그림 2)의 일반화차트는 재사용할 수 없다.

(그림 3)과 같이 정의된 RADAR_HANDLER는 일반화 인스턴스인 RDR_PS_1, RDR_PS_2 외 새로운 Activity Chart를 포함하고 있다. 이 차트의 역할은 외부 환경으로부터 들어오는 다양한 형식의 데이터를 일반화차트가 처리할 수 있도록 형식매개변수의 자료형에 맞도록 변환시키는 것이다.



(그림 3) Generic 인스턴스를 포함한 재사용 모듈

이와 같이 추가적인 기능이 필요한 이유는 일반화 인스턴스들이 자신의 형식매개변수와 다른 타입의 데이터를 직접 처리할 수 없기 때문이며 따라서 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 형식매개변수로 선언할 수 없다. 그 결과 선택적 자료형이 가지는 효과를 얻기 위해 새로운 Activity가 요구되는데 (그림 3)의 PROCESS_RADAR_DATA가 그 역할을 수행한다.

PROCESS_RADAR_DATA는 입력 데이터의 자료형에 상관없이 일반화차트의 형식입력매개변수와 동일한 유형의 출력 데이터를 생성해야 한다.

이 과정에서 (그림 3)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 포함하고 있다. 그리고 각 레이더 별로 DATA_3, 혹은 DATA_4를 고유데이터로 가지고 있다.

다음으로 각 레이더로부터 서로 다른 데이터 유형을 입력 받은 PROCESS_RADAR_DATA는 일반화 인스턴스인 RDR_PS_1 또는 RDR_PS_2에 형식매개변수와 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

이를 위해 PROCESS_RADAR_DATA는 자신의 Mini-Spec 내에 다음과 같은 루틴을 포함하게 된다.

```
if RADAR_DATA_1.RADAR_TYPE==A then
  COMBINED_RDR_DATA_A.RADAR_TYPE=A;
  COMBINED_RDR_DATA_A.SPECIFIC_DATA
    .SPECIFIC_DATA_KEY=3;
  COMBINED_RDR_DATA_A.SPECIFIC_DATA.
    SPECIFIC_DATA_VALUE=RADAR_DATA_1.DATA_3;
end if;
if RADAR_DATA_2.RADAR_TYPE==B then
  COMBINED_RDR_DATA_B.RADAR_TYPE=B;
  COMBINED_RDR_DATA_B.SPECIFIC_DATA.
    SPECIFIC_DATA_KEY=4;
  COMBINED_RDR_DATA_B.SPECIFIC_DATA.
    SPECIFIC_DATA_VALUE=RADAR_DATA_2.DATA_4;
end if;
```

TRACK_MONITOR로 최종 전달되는 TRACK_INPUT은 TRACK_ID와 RADAR_1 및 RADAR_2의 모든 데이터를 통합한 데이터 필드로 구성된다.

<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 일반화차트를 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 조건별로 데이터를 구분하는 역할과 (예제 5)에서와 같이 공통 및 고유데이터를 처리하여 원하는 출력형태로 내보내는 서브프로그램 [5]의 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 조정 작업만을 거쳐 사용 가능해야 하며 이 때 자원의 낭비, 액세스 빈도, 향후의 환경 변화에 대해도 가장 효과적으로 대응할 수 있어야 한다.

소프트웨어 컴포넌트를 재사용 하는 것은 생산 작업의 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.

참고문헌

[1] Colin Atkinson, "Object-Oriented Reuse, Concur-rency and Distribution", Addison -Wesley, 1991

[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] J. G. P Barnes, "Programming in Ada, 4th Edition", Addison-Wesley, 1993

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