'97 춘계학술발표회 논문집
한국원자력학회

필수안전 소프트웨어 프로그래밍 언어로서의 C++

박 종모+, 이 상범+, 이 장수++

+ 단국대학교 충남 천안시 안서동 산 28번지
++ 한국원자력연구소 대전광역시 유성구 덕진동 150번지

요약

C++는 C의 장점을 가지고 있으면서 객체지향적 요소를 포함하고 있어 현재 객체지향적 소프트웨어 개발에 가장 많이 사용되어지는 언어 중에 하나이다. 본 고에서는 원자력 발전소 제어 시스템과 같은 안전 시스템 개발에 사용될 언어가 보장해야할 속성에 대하여 정의하고 이러한 속성들을 C++는 어떻게 지원할 수 있는 가에 대하여 기술하였다. 80년 이후에 소개된 객체지향 기술이 계속적으로 발전하고 소프트웨어 개발에 많은 장점을 제공하고 있기 때문에 많은 분야의 소프트웨어 개발에 적용되어지고 있다. 따라서 safety-critical 시스템과 같이 고신뢰성과 안전성이 요구되는 시스템 개발에도 적용되어 진다면 소프트웨어 개발 생산성에 많은 도움이 될 것이다.

1. 서론

Safety 소프트웨어는 다른 응용 소프트웨어에 비해 신뢰성과 안정성이 보장이 되어져야 하며 개발 프로그래밍 언어의 특성과 프로그래밍 구조에 직접적으로 영향을 받게 되므로 언어의 선택이 중요하다. 현재 많이 사용되어지는 언어로서는 Ada, Fortran, Jovial, C, C++, PL/M 등 수십 여종에 이른다. 최근 객체지향적 개발 방법론이 소프트웨어 전 분야에 적용되어지고 있는 추세인데, 그 이유로는 모듈화, 추상화, 정보은폐 등 프로그램 개발에 있어 많은 장점을 제공하고 있기 때문다. 또한 프로그램의 복잡도와 개발비용을 줄일 수 있다는 장점이 있어 대형 소프트웨어 개발에 많이 사용되어지고 있다. 그렇다면 C++ 역시 safety-critical 소프트웨어 개발에 적합한 언어인가? 본 논문에서는 안전 소프트웨어 개발에 사용될 프로그래밍 언어가 가지고 있어야할 필요한 속성들(attributes)을 살펴보고, C++가 이러한 속성들을 만족시키고 있는 가를 살펴보고 아울러 safety-critical 소프트웨어에 적합한 언어로 사용되어지기 위한 프로그래밍 기법에 대하여 기술하고 있다.

제 2장에서는 안전 소프트웨어를 위한 프로그래밍 언어가 요구되는 속성들에 대하여 기술하였고 제 3장에서는 C++이 이러한 요구사항을 만족하고 있는 가에 대하여 설명하고 있다. 제

4장에서는 결론을 기술하고 있다.

2. 안전 소프트웨어 프로그래밍의 속성들

원자력 발전소 보호 및 제어 시스템과 같은 safety-critical 시스템에 사용되는 소프트웨어는 다른 일반 소프트웨어에 비해 사용할 프로그래밍 언어가 가져야할 속성들이 까다로울 수밖에 없다. 우선 오류의 발생을 억제할 수 있는 장치가 보장되어야 하며 문제 발생 시 프로그램을 추적할 수 있는 특성을 가져야 한다. 최근 소프트웨어 안전성을 수학적으로 증명하기위한 시도로 정형적 기법들이 많이 연구되고 있으나 소프트웨어의 규모가 조금만 커져도 증명이 불가능해 지거나 비용증가로 실용성이 없어진다. 안전 소프트웨어를 객체지향기법으로 개발할 수 있게 된다면 재사용 가능성을 높여 경제성 향상이 기대되고, 디지털화 가속화로 인한 복잡한 안전 소프트웨어 개발 요구증가에 대한 대책이 될 수 있을 것이다. 그러나 이를 위해서는 여러 가지 환경 변화나 문제 발생에 적절히 대응하고 유지보수를 쉽게하면서도 안전성을 보장할 수 있는 속성을 가져야 한다. 다음은 원자력 규정 위원회 (Nuclear Regulatory Commission:NRC)에서 정의한 안전 시스템을 위한 프로그래밍 언어의 몇 가지 속성들에 대하여 기술하였다. 따라서 안전 소프트웨어 개발을 위한 언어는 이러한 속성들을 잘 만족시켜야 한다.

■ 신뢰성(Reliability)

신뢰성이란 정의된 시간 이내와 정해진 조건하에서 성공적인 실행 확률 또는 요구 사항에 대한 성공적인 실행의 확률을 말한다. 소프트웨어의 성공적인 수행은 시스템의 메모리와 프로그램 로직에 의한 소프트웨어의 적절한 행위의 결과이다. 신뢰성을 높일 수 있는 방법으로는 메모리 활용의 예측성, 제어 흐름에 대한 예측성 그리고 타이밍에 대한 예측성을 높임으로 높일 수가 있다.

■ 건전성 (Robustness)

건전성이란 비정상적인 상태나 사건이 발생하더라도 안전 시스템의 소프트웨어가 받아드릴 수 있는 방법으로 운행을 할 수 있는 능력을 말한다. 즉, 예외의 상태에 대처하거나 내부적인 문제를 해결하거나 그리고 비일상적인 상황으로부터의 발생하는 오류의 확산을 막을 수 있는 소프트웨어의 능력을 말한다.

■ 추적성(Traceability)

추적성은 소스 코드와 라이브러리 요소들의 기원 그리고 개발과정을 검토하거나 규정하는 것에 대한 신뢰에 관련되어 있다. 또한 추적성은 상위 단계의 설계 문서와 더불어 소스 코드에 관련될 수가 있을 뿐만 아니라 소프트웨어 V&V와 품질의 확신에 대한 다른 요소들을 쉽게 해주기 때문에 중요하다.

■ 유지보수성(Maintainability)

유지보수성이란 소프트웨어가 개발 완료 후에 발생하는 변경과정에서 발생할 수 있는 오류의

가능성을 줄일 수 있는 수단이 된다.

3. Safety-Critical System 개발 언어로서의 C++

C++는 C를 기반으로 확정된 언어로 AT&T에서 개발되어 시스템 소프트웨어를 비롯하여 많은 분야 응용 소프트웨어 개발에 많이 사용되어지고 있다. C++는 C의 장점인 간편성, 기술력, 이식성 등 그대로 유지하면서 클래스, 상속성, 다형성 등을 지원하기 때문에 점차 그 사용 범위가 늘어날 것으로 기대하고 있다. Safety-critical 소프트웨어 개발에 C++를 사용하여 앞장에서 정의한 속성을 만족시키고자 할 때는 다음과 같은 지침을 따르는 것이 좋다. 본 논문에서는 일반적인 내용은 제외하고 C++ 언어의 특성에 관련된 내용만을 기술하였다.

3.1 신뢰성

소프트웨어의 신뢰성을 높이는 방법으로는 다시 세분하면 메모리 활용의 예측성, 제어 흐름의 예측성 그리고 타이밍의 예측성을 높이면 된다. 이러한 예측성을 높이기 위한 방법으로 C++에서는 다음과 같은 방법들이 유효하다.

▸ 클래스 라이브러리 크기의 조정

객체지향 프로그램은 객체의 타입인 클래스들로 구성되어져 있다. 클래스는 데이터들과 이를 활용하는 메소드들로 구성되어져 있는데, 분석과 설계과정에서 적절한 클래스를 정의하는 것이 프로그래밍에서 너무나 중요하다. 왜냐하면 잘못된 클래스의 정의로 전체적으로 프로그램 구조가 안정되지 못하고 클래스간의 결합도가 높아지면 그 만큼 오류의 가능성이 높이지기 때문이다. 따라서 클래스 크기와 수를 조정하는 것이 중요하다.

▸ 연산자 overloading의 조정

확실한 의미가 없는 연산자의 overloading은 코드에 에러를 발생할 수가 있다. 예를 들어 Complex::operator+와 List::operator+가 정의되었다면 "+"라는 연산자는 정수와 부동 소수의 합산뿐만 아니라 복소수와 list의 원소들끼리의 합산에 적용될 수가 있는데, 이로 인한 오류가 발생할 수가 있다. 만약 overloading을 사용하더라도 연산자의 우선 순위의 혼돈이 초래될 수 있기 때문에, 언어에서 정해진 우선 순위(precedence)를 따르기보다는 괄호를 사용하여 우선 순위를 확실히 하는 것이 좋다.

▸ 내부적인 다양성의 조정

내부적인 다양성을 지원하기 위해서는 같은 메시지를 처리하는 여러 개의 클래스를 작성하되 이들 클래스 내부의 메소드는 다르게 구현해야 한다. 그렇게 되어야만 클래스간의 상호교환성이 높아진다. 또한 이러한 클래스들은 다른 내부 데이터 속성을 가져야 한다.

▸ 동적 메모리 할당의 최소화

동적 메모리 할당이 메모리 효율성을 증가시킨다 하더라도 메모리 예측을 불가능하게 하기 때문에 할당되지 않는 메모리를 접근하거나 이미 할당에서 해방된 메모리를 요구하거나 또한 요구된 동적 메모리가 부족한 경우 등 많은 문제점이 생길 수가 있다. C++의 new()와 delete()를 사용하여 동적 메모리 할당을 할 수가 있으나, 가급적 꼭 필요하지 않는 경우를 제외하고는 사용하지 않는 것이 좋다. 그리고 이러한 연산자들은 overloading을 하지 않는 것이 안전하다. 또한 메모리의 잘못된 사용을 방지하기 위해서는 클래스는 동적으로 생성되고 소멸되기 때문에 모든 클래스는 반드시 소멸자 (destructor)를 포함해야 한다. 생성자 (constructor) 역시 어떤 문제 발생시 클래스가 차지하고 있는 메모리의 문제를 없애기 위한 방법을 고려하여 정의되어져야 한다. 또한 모든 유도 클래스 (derived class)역시 가상 소멸자를 포함하고 있어야 한다.

▸ 제어 흐름 복잡도의 최소화

복잡한 제어 흐름은 프로그램을 복잡케 만들어 이해하거나 유지보수에 많은 어려움이 따른다. 따라서 switch문 이외의 반복문에서는 break와 continue의 사용을 피해야 한다. 또한 if-then문보다는 switch 문을 사용하는 것이 좋다. 일반적으로 객체지향 프로그램에서 큰 사이즈의 함수 대신에 많은 작은 함수들을 사용하면 제어 흐름의 복잡도를 줄이는데 도움이 된다.

▸ 변수와 포인터의 초기화

모든 변수와 포인터는 사용되어지기 전에 초기화시키는 것이 오류 방지에 도움이 된다. static 변수는 컴파일러가 0으로 초기화를 시켜주지만 그 외 전역 변수 같은 것은 컴파일러에 따라 초기화가 될 수도 있고 아닐 수도 있다. C++에서는 클래스의 생성자에서 모든 클래스의 속성을 초기화시키는 것이 좋다. 포인터 역시 초기화를 시키지 않으면 원치 않는 값을 가질 수가 있기 때문에 잘못된 메모리 위치를 가리킬 수가 있기 때문이다.

▸ 클래스 자료형의 사용

C++에서는 구조체(structure)를 사용하기보다는 클래스라는 자료형을 사용함으로써 더 나은 접근제어를 보장한다. 또한 연합체(union)를 사용하기보다는 클래스의 계층구조를 사용하는 것이 좋다. 또한 기본 자료형인 int, float, char는 진정한 클래스가 아니기 때문에 경우에 따라선 클래스로 재정의를 하면 access control을 제공하고 일관성을 높일 수가 있다. 하지만 기본형보다는 클래스가 실행속도는 저하시킬 수가 있다. 일반적으로 C++는 다른 언어에 비해 강력한 타입 규제를 보장하기는 하지만 type-cast pointer를 포함하고 있기 때문에 이러한 것은 피해야 한다.

3.2 건전성(Robustness)

Robustness는 비정상적이거나 예측되지 않는 상황에서 살아남는 소프트웨어의 속성을 말한다. 특히 안전 소프트웨어는 예측 못한 사건들이 발생할 수가 있기 때문에 이러한 사고에서도 시스템이 다운되지 않고 계속적으로 실행을 할 수 있는 기능이 필수적이다. 건전성을 보장하기 위해서는 exception 즉, 정상적인 프로그램 실행의 중단을 일으키는 사건을 처리하는 기능이 보장되어야 한다. exception은 지역적으로 처리되어져야 하는데, exception의 종류로는 addressing

exception, data exception, I/O exception, overflow/underflow exception, operation exception protection exception 등이 있다.

C++에서 이러한 exception을 지역적이고 일관성 있게 처리하기 위해서는 우선 운영체제가 지원하고 있는 기능에 의존하기보다는 signal과 trap에 의존하는 것이 좋다. 왜냐하면 운영체제에서 지원하는 기능의 신뢰성이 의문시 될 수도 있기 때문이다. 따라서 exception에 관련된 signal과 trap을 직접 처리하는 기능을 자체적으로 갖고 운영체제의 기능은 보조적인 수단으로 사용하는 것이 좋다. C++의 throw()와 catch()같은 기능을 이용하여 exception handing을 하면 C의 setjmp()와 longjmp()보다 뛰어나다.

3.3 추적성 (Traceability)

추적성이란 개발된 프로그램이 설계 명세서의 내용과 비교되어 정확성과 완벽성을 검증하는데 도움이 된다. 이러한 속성을 만족시키기 위해서는 다음과 같은 가이드라인을 따르는 것이 좋다.

. built-in 함수의 사용을 최소화한다.
. 라이브러리 함수의 사용을 최소화한다.
. 형상관리 도구를 활용한다.
. 주석과 내부 문서를 잘 정의한다.

3.4 유지보수성

소프트웨어는 사용과정에서 주변환경의 변화, 오류의 발생, 요구사항의 변화 등으로 인해 끊임없이 변화가 요구되는데, 이러한 변화 과정에서 오류의 발생을 최소화하도록 개발되어져야 한다.

▸ 판독성 (Readability)

요지보수에 있어서 readability의 중요성은 NASA 연구 보고서 찾아 볼 수 있는데, 이를 통하여 변경하여야할 코드 식별을 향상시킬 수 있고 새로운 결점을 줄일 수가 있다. 판독성에 대한 일반적인 표준은 없지만, 들여 쓰기(indentation), 주석(comments), 혼합 언어사용의 최소화, 부프로그램 크기의 제한, literal 사용의 최소화 등을 통하여 어느 정도 향상시킬 수 있다. 이러한 관점에서 Visual C++는 환경 적으로 판독성을 높이도록 도와주고 있다.

▸ 자료의 추상화 (Data Abstraction)

자료의 추상화는 C++는 클래스로 정의되는데, 같은 범주 안에 데이터와 함수의 조합으로 묶어져 있다. 실행 시나 소프트웨어 유지보수시에 변수를 변경하였을 때의 잠재된 부가적인 영향을 줄이거나 제거하는 것이 안정성에 있어서 아주 중요하다. 이와 더불어 전역변수 사용의 최소

화하는 것이 좋다.

▸ 이식성 (Portability)

이식성이란 다른 운영 플랫폼에서도 예측가능하고 일관성 있는 결과를 산출하는 표준적인 프로그래밍 구조의 말한다. 이식성을 높이기 위한 방법으로는 built-in 함수, 컴파일러된 함수 사용, 동적 바인딩, 인터럽트의 최소화 등으로 가능하다.

4. 결론

객체지향 언어인 C++가 safety-critical 소프트웨어 개발 언어로 사용되어지기 위해 따라야 할 guideline를 몇 가지 제시하였다. 복잡하고 대형의 안전시스템 소프트웨어 개발에 객체지향 기술을 이용하게 되면 분석에서 코딩까지 여러 가지 이점과 재사용 증가에 따른 경제성 제고가 기대되지만, 아직까지 객체지향 기술의 테스팅 기법과 안전성 기법에 대해서는 많은 연구가 필요하다. C++가 최근 객체지향 언어로 각광을 받고 있으며 특히 C언어를 확장했기 때문에 C 개발자에게 환영을 받고 있다. 본 논문에서는 C++가 사용되어질 때 고려해야 할 사항을 언어적 속성에 따라 몇 가지 제시하고 있다. 일반적인 내용보다는 객체지향적 관점에서의 특성에 대한 내용이 많이 기술하였다. 종합적인 관점에서는 C++가 앞으로 안전 시스템 개발에 적절한 프로그래밍 언어로 평가되며, 본 논문에서 살펴본 바와 같은 특성들을 보완한다면 safety-critical 시스템 개발 언어로 발전될 수 있으리라 기대된다.

참고 문헌

[And84] O.Anderson and P.G. Petersen, "Standard and regulations for software approval and certification," Elektronik Centralen Report ECR 154 (Denmark), 1984.

[Bin95] D.W. Binkley, "C++ in Safety Critical Systems," NIST-IR 5769, National Institute of Standards and Technology, November, 1995.

[Car92] T.C. Cargil, "C++ Programming Style," Addison Wesley, 1992.

[Cut93] B. Cuthill, "Applicability of Object Oriented Design Methods and C++ to Safety Critical Systems," Proc. of the Digital System Reliability and Nuclear Safety Workshop, NUREG/CR-0136, NIST SP 500-216, 1993.

[Hec96] H. Hecht 외 7명, "Review Guidelines on Software Languages for Use in Nuclear Power Plant Safety Systems," SoHar Inc., 1996.

[Eck95] B. Eckel, "Exception Handling in C++," Embedded Systems Programming, Vol. 8, No.1, Jan., 1995.

[Par90] D.L. Parnas 외 2명, "Evaluation of Safety Critical Software," CACM, Vol.33, No.6, June, 1990.