내장형 실시간 시스템의 소프트웨어 아키텍처 평가 절차 및 성능 평가 기준 고려사항

권도형°, 최윤석, 정기원
숭실대학교 컴퓨터학과
kdh906@empal.com, secooling@hanafos.com, chong@ssu.ac.kr

A Software Architecture Evaluation Procedure and Performance evaluation criteria In Realtime-Embedded Systems

Dohyung Kwon°, Yunseok Choi, Kiwon Chong
Soongsil University

요 약

내장형 실시간 시스템은 점차 소형화, 다기능화 하여 그 복잡도가 증가하고 있다. 이로 인해 소프트웨어 아키텍처의 적용이 요구되고 있다. 아키텍처를 적용하기 위해서는 내장형 실시간 시스템에 적합한 아키텍처를 선정해야 하며, 선정한 아키텍처는 처리속도와 같은 비기능적 요구사항을 만족할 수 있어야 한다. 이에 본 논문에서는 내장형 실시간 시스템 개발 중 아키텍처 선정 시 처리속도에 기반하여 아키텍처를 선정할 수 있는 성능평가 절차와 고려사항을 제안한다. 이를 통해 사용자가 제시한 성능적 요구사항에 적합한 아키텍처를 선정할 수 있다.

1. 서 론

내장형 실시간 시스템은 특정 기능을 수행하기 위한 소프트웨어와 이를 구동하기 위한 하드웨어로 구성된다. 내장형 실시간 시스템의 응용범위는 점차 확대되고 있으며, 시스템 자체의 구성은 소형화, 다기능화 함에 따라 시스템의 복잡도가 증가하고 있다. 이로 인해 내장형 실시간 시스템의 개발에도 소프트웨어 아키텍처의 적용이 필요하다. 내장형 실시간 시스템에 소프트웨어 아키텍처를 적용하기 위해서는 해당 아키텍처가 시스템의 기능적인 요구사항을 충족할 뿐만 아니라 안정성, 확장성 및 처리속도와 같은 비기능적 요구사항을 충족할 수 있어야 한다. 특히 내장형 실시간 시스템 경우 시간제약을 충족할 수 있도록 빠른 처리속도를 갖는 아키텍처가 요구된다. 이에 본 논문에서는 내장형 실시간 시스템의 소프트웨어 아키텍처 평가절차 및 성능평가 고려사항을 제시한다. 이를 통해 여러 후보 아키텍처 중 빠른 처리속도를 갖는 아키텍처를 정량적으로 평가, 선정할 수 있기 위해 고려해야 할 평가항목을 제공한다. 논문의 구성은 다음과 같다. 2장에서는 소프트웨어 아키텍처에 대해 기술하며, 3장에서는 처리속도를 고려한 아키텍처 성능평가 절차 및 고려사항을 제시한다. 4장은 제시한 기법을 적용한 사례를 살펴보며, 5장에서 결론을 맺는다.

2. 소프트웨어 아키텍처

소프트웨어 아키텍처는 복잡한 시스템을 분할하여 구조화하여 전체 시스템의 가시성을 제공하고, 이해 관계자가 시스템의 목적, 범위,기능 등을 명확히 이해할 수 있도록 도와준다. 그러므로 시스템의 복잡도가 증가할수록 아키텍처 기술적용이 필요하다.

2.1 소프트웨어 아키텍처 뷰타입

아키텍처를 기술할 때 다양한 뷰타입과 스타일을 사용할 수 있다 .다음은 소프트웨어 아키텍처 스타일의 분류이다. [2]
Module 뷰타입은 목표시스템의 구조적인 모습을 표현하고 Component-and-Connector 뷰타입은 시스템의 실행시간의 행위를 보여준다. Communicating-Processes 스타일은 제어의 흐름을 표현하기에 적절하므로 내장형 실시간 시스템을 표현하는데 효과적이다. Allocation 뷰타입은 소프트웨어 모듈이 실제 실행되는 물리적인 요소를 나타내는데 사용된다.

표1. 소프트웨어 아키텍처 스타일

ViewTypes	Styles
Module	Decomposition, Uses, Generalization, Layered
C & C	Pipe-and-Filter, Shared-Data, Publish-Subscribe, Client-Server, Peer-to-Peer, Communicating-Processes
Allocation	Deployment, Implementation, Work Assignment

2.2 소프트웨어 아키텍처 평가

소프트웨어 아키텍처 평가의 목적은 소프트웨어 아키텍처 분석을 통해 해당 아키텍처가 가지는 위험을 발견하고, 이해당사자가 원하는 특정 품질속성을 보다 잘 반영하기 위함이다. 따라서 특정 품질속성에 민감한 지점(Sensitivity Point)을 발견하고 여러 품질속성 간의 Trade-off 지점을 발견하는 것이 중요하다. 대표적인 아키텍처 평가 방법으로 ATAM(Architecture Tradeoff Analysis Method)이 있다. ATAM 의 프로세스는 다음과 같다[1].

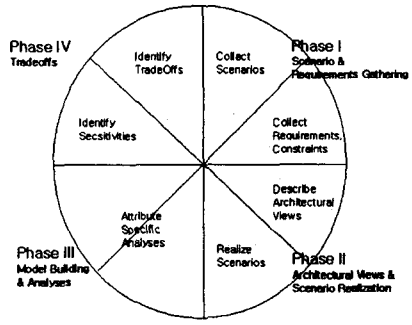

그림 1 ATAM 프로세스

그림1에서 보는 바와 같이 ATAM은 총 4개의 단계로 구성되어 있으며,각 단계는 두 개의 세부 단계로 구성된다. ATAM은 아키텍처 분석결과를 정량적인 형태로 제공하지 않으므로 처리속도 평가와 같은 객관적인 정보를 요구하는 분석의 경우 어려움이 있을 수 있다. 본 논문에서는 처리속도를 중심으로 내장형 실시간 아키텍처 선정을 위한 성능평가절차와 평가기준 고려사항을 제시한다.

3. 내장형 실시간 시스템의 소프트웨어 아키텍처 평가절차 및 평가기준 고려사항

3.1 아키텍처 평가절차

본 논문에서 제시하는 아키텍처 평가 절차는 다음과 같다.

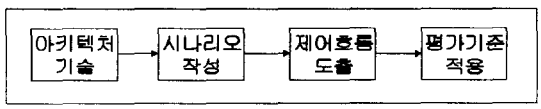

그림 2 아키텍처 평가절차

- 아키텍처 기술

목표시스템에 적합한 아키텍처 스타일을 적용하여 여러 개의 후보 아키텍처를 작성한다.소프트웨어 관점과 하드웨어 관점을 모두 고려해야한다. 소프트웨어의 구조를 나타내는 Module 뷰타입의 Decomposition 스타일과 제어의 흐름을 나타내는 C&C 뷰타입의 Communicating-Processing 스타일, 실제 물리적으로 어떻게 할당되어 있는지를 나타내는 Allocation 뷰타입의 Deployment Style을 사용하여 하드웨어 관점을 기술한다.

- 시나리오 작성

아키텍처에 적용할 시나리오를 작성한다. 시나리오는 특정 속성을 잘 나타낼 수 있는 시나리오를 구상한다. 이 시나리오를 기반으로 하여 제어의 흐름을 찾는다. 목표 아키텍처를 충분히 고려하기위해 다양한 시나리오를 작성하여 적용한다.

- 제어흐름 식별

작성된 시나리오를 후보 아키텍처에 적용하여 제어 흐름을 식별한다. 또한 제어 흐름에 하드웨어를 어떻게 매핑해야 하는지를 분석한다. 각 후보 아키텍처들은 다른 제어의 흐름을 갖게 되고, 이로 인해 성능의 차이가 나타나는 지를 살펴볼 수 있다. 제어 흐름에 따라 특정 소프트웨어 모듈이 불려지는 횟수나 특정 하드웨어를 접근하는 횟수 가 달라질 수 있다.

- 평가기준 적용

식별한 제어흐름을 평가기준에 적용하여 성능을 측정한다. 아키텍처 상에서 시나리오를 따라 제어 흐름을 찾고 그것을 좇아가면서 평가기준에 값을 부여한다. 부여된 값을 계산식을 사용 하여 수치화 한다. 이런 작업을 각각의 후보 아키텍처별로 수행한다. 이렇게 후보 아키텍처 별로 수치화된 정보를 표로 만들어 각각의 후보아키텍처들을 비교하고 이것을 근거로 하여 내장형 실시간 시스템의 아키텍처를 선정한다. 이렇게 함으로서 직관만이 아닌 좀 더 정량적으로 아키텍처를 성능면에서 분석하여 아키텍처를 선정할 수 있도록 한다.

3.2 처리속도 중심의 아키텍처 평가기준 고려사항

평가기준 항목으로 하드웨어 수, 하드웨어 접근 횟수, 하드웨어 속도(CPU, RAM, 네트워크 속도), 메모리 용량, 소프트웨어 모듈의 개수, 태스크 호출 수, 소프트웨어 모듈의 복잡도 등을 고려한다. 먼저 태스크 호출 수는 소프트웨어가 실행할 때 불려지는 특정 태스크의 호출 수를 의미한다. 이것은 동일한 기능을 하는 시스템일 지라도 다른 아키텍처일 경우 태스크의 호출 빈도수가 달라질 수 있다. 태스크의 복잡도는 태스크를 이루고 있는 모듈의 개수, 메시지 교환 회수, 메시지 가중치를 포함한다. 복잡도가 큰 태스크가 많이 호출되면 성능이 떨어지게 됨을 의미한다. 메시지 가중치는 메시지를 구성하는 데이터의 양을 의미한다. 메시지의 교환 횟수는 해당 태스크가 호출 될 때 마다 발생하는 메시지 교환 횟수를 의미한다. 모듈의 개수는 태스크 안에 구성되는 모듈의 수를 나타낸다. 모듈의 수는 태스크의 수와 같을 수도 있고 더 많을 수도 있다. 하드웨어의 속도는 성능에 가장 큰 영향을 미친다. 소프트웨어 태스크와 밀접한 관련이 있는 CPU속도, 메모리 접근 시간등을 고려한다. 메모리 용량 역시 고려하여야 한다.

4. 성능 평가 방법의 적용

앞에서 제시한 방법을 적용할 시스템으로 세차장 모형을 사용하였다[5]. 세차장모형은 자동차가 세차장 레일위로 올라가면 레일을 움직여 옳을 뿌려주는 위치까지 이동하고 브러쉬로 청소를 해주고 마지막으로 건조 실행를 통해 자동차를 건조시키는 장치다.

4.1 아키텍처 기술

후보아키텍처1의 Control Task(a) 는 Rail, Lamp, Brush, Dry의 기능을 하는 4개의 모듈이 하나의 태스크를 이루었다. 후보아키텍처2는 앞서 말한 4개의 모듈이 독립적으로 나뉘어져있다.

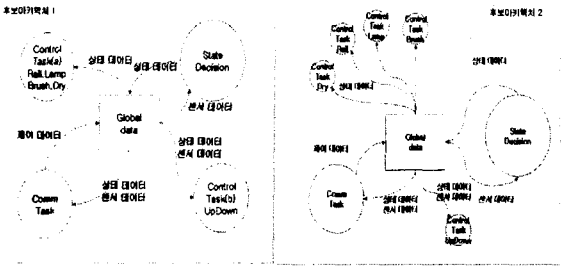

그림 3 세차장모형 소프트웨어 아키텍처

Comm Task : 통신 태스크로 분산 환경에서 웹 클라이언트와 통신을 한다.

Control Task : 약속된 프로토콜로 데이터를 받아 해당모듈을 실행하여 액추에이터를 제어한다.

State Decision Task : 센서로부터 감지된 데이터를 읽고 상태를 저장한다.

4.2 시나리오 작성과 제어 흐름 도출

- 시나리오
  - 웹 클라이언트에서 시작버튼을 누른다
  - 상태를 읽어 들이고 판단한다.(메모리 저장)
  - Rail이 진행한다.
  - Updown 컨트롤이 차의 높이를 체크해서 차의 높이에
  - 맞도록 제어한다.
  - 광센서를 돌린다(몇 세척)
  - 브러쉬를 돌린다.
  - 건조실행을 돌린다.(건조)

    Rail이 정지한다.

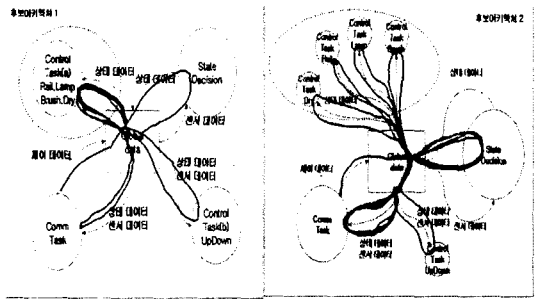

그림 4 제어의 흐름 식별

문제를 단순화하기 위해 점선 안에 있는 부분으로 한정하여 분석을 수행하였다. 후보아키텍처1의 점선 부분의 태스크를 후보아키텍처2의 점선안의 4개의 태스크로 표현하였다. 시나리오를 따라서 각 태스크를 따라가면서 제어의 흐름을 식별하는 그림이다.

4.3 평가기준 적용

앞서 제시한 평가기준 고려사항은 시나리오에 기반하여 수치화한다. 이때의 값은 각 후보 아키텍처 간의 상대적인 값이다. 그림 4의 후보아키텍처1은 시나리오를 따라가면 태스크의 호출이 4회이고 태스크가 호출 될 때 마다 교환되는 메시지는 1회이다 그러나 메시지 가중치 즉 메시지를 구성하는 데이터의 양은 4이다. 이것은 메시지 안에 4개의 모듈에 필요한 데이터를 포함하기 때문이다. 이것과 비교해서 후보아키텍처2는 시나리오 상의 태스크 호출 회수는 4로 같지만 메시지 가중치가 1이다. 이것은 각 모듈을 하나의 태스크로 분리하여 메시지를 구성하는 데이터의 양이 1로 줄어들었다. 이런 방법으로 태스크의 호출 수와 태스크의 복잡도를 고려한다. 이렇게 상대적인 수치로 아키텍처를 평가할 수 있는 정량적인 데이터들을 아키텍처 간에비교하여 성능우위의 아키텍처를 선정한다.

5. 결론 및 향후 연구방향

본 논문에서는 내장형 실시간 시스템의 소프트웨어 아키텍처 선정을 위한 처리속도 중심의 성능평가 절차와 평가기준 고려사항을 제안하였다. 제안한 평가방법을 사용하여 세차장 모형 시스템의 아키텍처 평가를 수행하였다. 제안한 방법은 아키텍처를 구성하는 태스크 각각의 구성 모듈 및 메시지를 분석하고, 이를 토대로 태스크 복잡도 계산 후, 태스크 호출 빈도를 고려하여 태스크의 처리속도를 산출한다. 산출된 처리속도는 여러 후보 아키텍처 중 특정 아키텍처를 선정할 때 사용할 수 있는 정량적인 평가 자료가 된다. 제안한 방법의 정량적인 평가결과를 통해 후보 아키텍처 중 보다 빠른 처리속도를 갖는 아키텍처를 선정할 수 있다. 향후 연구과제로 수치화 한 평가항목들이 서로 어떤 관계에 있는지를 연구해서 식을 도출하고 정량적 평가를 위한 메트릭을 만들 필요가 있다.

제시한 내장형 실시간 소프트웨어 아키텍처 평가방법은 처리속도 중심으로 아키텍처의 평가절차와 고려사항을 제시하였으나, 향후에는 처리속도 뿐만 아니라 확장성, 재사용성, 유용성 및 안정성 등 기타 다양한 품질속성별 고려사항을 제시하고 정량적으로 측정하여 아키텍처를 평가할 수 있는 방법에 대한 연구가 필요할 것이다.

6. 참고문헌

[1] Len Bass, Paul Clements, Rick Kazman "Software Architecture in Practice",Addison-wesley,2003.

[2] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, "Documenting Software Architectures Views and Beyond" Addison-wesley, 2003.

[3] Marco Castaldi, Paola Inverardi, Sharareh Afsharian, "A Case Study in Performance, Modifiability and Extensibility Analysis of a Telecommunication System Software Architecture", IEEE Int'l Symp. 2002.

[4] Wolfgang A. Halang, Krzystof M. Sacha, "Real-Time Systems", World Scientific, 1992

[5] 강순주 "임베디드 소프트웨어 교육 워크샵", ETRI