국가 ITS 아키텍쳐 발전방안 : 객체지향 국제표준 아키텍쳐 수용 방안

백인섭
(아주대학교 정보통신대학, 교수)

목 차

Ⅰ. 아키텍팅 개념 및 방법론에 대한 고찰
Ⅱ. 국가아키텍쳐 발전방안 : 복합적 아키텍팅
Ⅲ. 객체지향 아키텍쳐 수용 방식
Ⅳ. 결론

Ⅰ. 아키텍팅 개념 및 방법론에 대한 고찰

시스템 아키텍쳐에 대한 개념과 방법론은 90년대 후반 미국의 지능형 교통체계에 실제로 수용되면서 그 중요성이 세계적으로 인식되게 되었다.[4,5] 그러나 이는 아직까지 학문적 수준에서 조차도 명확한 상태로 성숙되지 못한 상태로서 매우 다양한 개념과 방법론들이 혼재하고 있는 상황이다.[10,11,12,13] 따라서 세계 각국의 ITS 아키텍쳐는 그것의 개념수준은 물론 기술적 방법론 수준에서 까지도 상당한 수준의 차이를 가지고 있는 것이 현실적 상황이다. 이러한 혼돈은 우선 개념적 차이에서 초래되며 다음은 기술적 방법론적 차이에서 초래된다고 볼 수 있다. 개념적인 차이는 각국의 국가 아키텍쳐 서두에서 강조되는 아키텍쳐의 역할에서 묵시적으로 나타나고 있다. 미국,호주등의 국가아키텍쳐에서 그 역할은 첫째 이해당사자(Stakeholders)간의 협의수단으로서 강조되고 있고 다음은 다양한 주체들에 의해서 이루어지는 다양한 서비스 시스템들간의 통합을 강조하고 있다.[4,5,8] 그러나 유럽의 경우나 일본의 경우는 후자를 강조있다.[7,9] 그러나 이 두 가지 역할은 매우 다른 것으로서 전자는 다양한 조직간의 역할분담을 협의하기 위한 것이기 때문에 방법론적 사안이나 기술론적 사안은 배제되어야하고 간단 명료해서 모든 이해당사자가 어렵지 않게 이해할 수 있어야 한다. 따라서 주로 정보교환 수준의 협동(Inter-Operability)에 초점이 맞추어져야 하는 반면에 후자의 경우는 다양한 시스템들 간의 통합(Integration)을 위한 것이기 때문에 방법론적 사안이나 기술론적 사안들이 상당수준 포함될 수 밖에 없어 복잡성과 난해성을 가지게된다. 그러나 이러한 두 가지 다른 수준의 개념을 구별 없이 모두 한꺼번에 포함하고 있는 것이 현재의 아키텍쳐 개념이며 따라서 방법론적으로나 기술론적으로 매우 다양한 방식들이 사용되고 있는 것이다. 이들 중 대표적인 것에 대해서 그 특성과 장단점을 분석해 보면 다음과 같다.

정보공학적 방법론(I.E : Information Engineering) [J. Martine][13]

- 제1수준 : World View Architecture
- 제2수준 : Domain View Architecture
- 제3수준 : Application View Architecture
- 제4수준 : Element View Architecture : Application Components !!!

정보공학적 방법론은 비지네스 영역 체계에 적합한 방법론으로서 거시적 관점에서의 하향식 시스템 틀을 구상하기에 적합한 방법론이다. 그러나 미시적 관점에서의 물리적 자원의 효율적 배분이나 실시간 제어성을 가지는 도메인의 시스템 틀을 구상하기에는 한계성을 가진다.

제품공학적 방법론(P.E: Production Engineering) [System Engineering] [13]

- 제1수준 : World View Architecture
- 제2수준 : (H/W, S/W, People, Data) : Technical-Domain View Architecture
- 제3수준 : HE, SE, HE, DE : Technical-Element View Architecture
- 제4수준 : Component View : Component-View Architecture

제품공학적 방법론은 서비스 체계가 아닌 제품 체계의 틀을 구상하기에 적합한 방법론이다. 그러나 거시적 관점에서의 시스템 상호연동체계를 구상하기에는 한계성을 가진다.

객체지향 소프트웨어 공학(OOSE)적 방법론[Ivar Jacobson OOSE][12]

- 제1수준 : Requirement Model (USECASE 모형)
- 제2수준 : Analysis Model (CLASS 모형)
- 제3수준 : Design Model (SEQUNCE 모형)
- 제4수준 : Implementation Model

OOSE 방법론은 거시관점의 시스템 환경에서의 요구사항(기능적 및 비 기능적)은 합리적으로 통합조정되어 정의됨을

가정하고 그 이후의 미시적 관점에서의 시스템 기본 틀을 구상하는 것에 초점이 맞추어 있다. 그러나 이러한 가정은 바로 거시아키텍처 사항으로서 그것의 어려움과 중요성을 간과한 것이다.

객체지향 방법론[UP(Unified Process) 모형]에서의 아키텍처 구상 방법[16]

-제1수준 아키텍처 : 요구사항 정의 수준(Requirement)
- USECASE 모형

-제2수준 아키텍처 : 분석 수준 (Analysis)
- CLASS 모형 : 논리적 객체 (기술 독립적 & 패키지 수준)
- COLLABORATION / SEQUENCE 모형

-제3수준 아키텍처 : 설계 개발 수준(Design)
- CLASS+ 모형 : 기술적 객체 (기술 종속적 & 개별 객체 수준)
- COLLABORATION+ / SEQUENCE+ 모형
- PACKAGE 모형

-제4수준 아키텍처 : 구현 수준(Implementation)
- CLASS++ 모형 : 물리적 객체 (기술환경 종속적)
- COLABORATION++ / SEQUENCE++ 모형
- COMPONENT 모형
- DEPLOYMENT 모형

UP에서는 이러한 Work-Flow(Iteration Cycle)가 4개의 Phase(Inception, Elaboration, Construction, Transition) 각각에서 반복적으로 수행된다.
또한 아키텍처는 위에서 제시된 모형 중에서 단지 아키텍처 관점에서 중요성을 가지는 부분들로 정의되며(Skeleton of System), Elaboration 단계의 마지막에서 아키텍처가 구상된다.(Base-Line Architecture) 즉 UP에서의 아키텍처는 "가동성을 가지는 시스템의 골격"(Runable System Skeleton)으로 볼 수 있다. 이것은 Construction/Transition Phase에서 계속 살이 부쳐져서 결국에는 완전한 시스템으로 성장하게 된다.(Iterative Development)

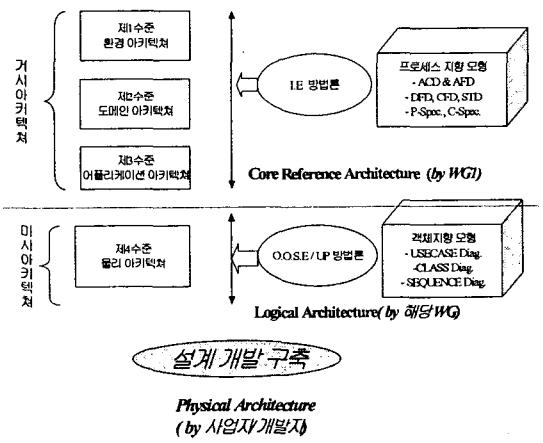
<분석 평가>

이상에서 살펴본 바와 같이 모든 방법론에서 아키텍처는 대개 4개 수준으로 구체화 과정을 가지며 방법론적/기술론적으로 볼 때는 크게 두가지 형태(프로세스 지향 방식과 객체지향 방식)로 대별된다.
미국 ITS 아키텍처의 경우는 프로세스 지향 방식으로 구상되어 있으며 그것의 목표 개념은 큰 의미의 이해당사자간의 협의와 작은 의미의 시스템 통합 모두를 포함하고 있기 때문에 4개 수준 모두에서 비교적 상세화 되어 대단히 대규모적이고 복잡한 특성을 가지고 있다.[4,5] 최근들어 새롭게 구상된 유럽연합의 프레임 아키텍처 또한 프로세스 지향 방식을 취하고 있으나 그것의 목표 개념은 미국과는 달리 시스템 통합을 위한 수단으로서 강조되고 있기 때문에 미국 경우와 같이 4개 수준 모두에서 상세화 되고 따라서 매우 대규모적이고 복잡한 것이 특징이다.[7] 그러나 국제표준에서는 상위 2개 수준만이 핵심 참조아키텍처로 이루어져 있으며 나머지 수준은 다음 단계의 영역별 논리아키텍처에서 구상되도록 되어 있지만 사용된 2개 수준이 명확한 수준에 대한 정의 없이 사용됨으로서 모호성이 있다고 하겠다.[6] 우리나라 국가 ITS 아키텍처의 경우는 미국이나 유럽의 경우처럼 프로세스 지향 방식을 취하고 있지만 그 목표개념이 큰 수준에서의 조직간(또는 서비스)의 역할 분담 정립에 초점이 맞추어 있다. 따라서 독립적으로 운영가능한 응용 시스템간 흐름정보 수준의 상호연동성을 구상 정의하는 수준에 머물고 있기 때문에 미국이나 유럽의 아키텍처와는 달리 비교적 간단 명료성을 가진다. 따라서 이해당사자간의 협의조정 수단으로서는 적합성을 가지지만 향후 구축될 수 많은 ITS관련 시스템들을 원활하게 효율적으로 통합하는 수단으로서는 취약성을 가진다 하겠다. [1,2]

II. 국가아키텍처 발전방안 : 복합적 아키텍팅

앞에 살펴본바와 같이 기존의 아키텍팅 방법론으로서의 정보공학적 방법론과 제품공학적 방법론은 각각 ITS 유형의 체계를 아키텍팅하기에는 적합성과 부적합성을 동시에 가진다. 따라서 이 두 가지 방법론을 적절하게 상호 보완적으로 사용하면 그 효과성을 극대화할 수 있을 것이다. 즉 거시적 관점의 아키텍팅은 기존의 시스템 지향 방식의 아키텍처 모델(ACD & AFD)과 정보공학적 방법론으로 구상하고 미시적 관점의 아키텍팅은 객체 지향 방식의 아키텍처 모델과 OOSE/UP 방법론을 확장 보완해서 구상하는 것이다. 이를 도시하면 <그림-1 >과 같다.

<그림-6> 복합 아키텍처 모형 및 방법론



<그림-1> 복합적 아키텍팅 방법

즉 우리나라 경우 기 수립된 국가아키텍쳐의 3 수준까지를 거시수준의 참조 아키텍쳐로 수용하고 제 4 수준의 물리 아키텍쳐는 미시 아키텍쳐로서 기존의 프로세스 지향 방식으로 구상하든지 또는 객체 지향 방식(OOSE/UP의 방법론을 확장 보완)으로 구상하는 것이다. 이러한 과정을 요약해 보면 다음과 같다.

- 핵심 참조 아키텍쳐 : 시스템간의 협동성 정의(시스템 & 흐름정보)
  - 시스템 지향 방식(ACD & AFD)
    : 기존의 국가아키텍쳐의 3 수준에 해당.
  - 객체 지향 방식(USE-CASE' & CLASS & SEQUENCE)

- 논리 아키텍쳐 : 시스템별 논리적 객체 및 상호 연동 관계 정의
  - 프로세스 지향 방식(DFD & CFD)
  - 객체 지향 방식 (USE-CASE+ & CLASS+ & SEQUENCE+& PACKAGE(K.ITS 4 수준 아키텍쳐에 해당))

- 물리 아키텍쳐 : 시스템별 물리적 객체 및 상호 연동 관계 정의.
  - 프로세스 지향 방식(물리적AFD)
  - 객체 지향 방식(CLASS++, SEQUENCE++, COMPONENT, DEPLOYMENT)

1. 프로세스 지향 방식의 거시 아키텍쳐(Macro-Architecture) 구상.

(국제표준에서 핵심-참조 아키텍쳐(Core Reference Architecture)에 해당)

거시 아키텍쳐란 거시적 관점에서의 논리적 아키텍쳐를 의미하며 그 목적은 거시적 관점에서의 기능(비지네스 기능)을 관련 어플리케이션 시스템들에 적절하게 배분 할당하고 상호 관계를 가지도록 체계를 논리적으로 정의하는 것이다. 이 경우 아키텍쳐적 구성요소는 어플리케이션 시스템 수준이기 때문에 객체로 표현하기에는 한계가 있다. 따라서 기존의 프로세스 지향 방식의 ACD 와 AFD 모델을 사용하는 것이 바람직하다. 아키텍쳐를 구상하는 방법론으로서는 기존의 정보공학적 방법론에 근거하여 상위 3개 수준의 아키텍쳐를 구상한다. 즉 제1수준으로 시스템과 환경의 관계를 정의하는 전체환경 수준의 아키텍쳐를 구상하고 제2수준으로 시스템에 해당하는 도메인(사업영역)과 도메인간의 바람직한 관계를 도출정의하는 도메인 아키텍쳐를 구상한다. 제3수준에서 도메인별 해당하는 어플리케이션 수준의 시스템들을 도출정의하고 이들간의 바람직한 상호관계를 정의하는 어플리케이션 아키텍쳐를 구상한다. 거시적 관점에서의 아키텍팅시에 고려되어야 하는 중요한 이슈들은 다음과 같다:

<거시 아키텍쳐 구상시 고려 사항>

- 서비스 또는 제공기능 차원에서의 중복/사각/상충 방지 : 거대 시스템을 정의하는 경우는 그 규모적 방대성과 복잡성으로 해서 서비스나 기능이 중복된든지 아니면 누락되든지 아니면 서로 상충하는 정의가 이루어지기 쉽다. 이러한 문제를 사전에 방지하기 위한 것이 거시적 아키텍쳐 정의이다.

- 거시적 수준에서의 합리적인 상호협동 체계 정의 : 거대 시스템은 다수의 서브시스템으로 구성되며 그 들간의 관계는 원래 결정되어 있는 것도 아니고 그렇다고 자연 발생적으로 이루지는 것도 아니다. 바로 거시적 아키텍쳐 구상시에 여러 관점에서 합리적으로 정의되어야 하는 것이다.

- 거시적 관점에서 어플리케이션들과 그들간의 관계를 도출정의 할 시에 기능적 합리성 뿐만이 아니라 법제도 조직적 합리성내지는 수용성까지도 고려되어 기능적으로 옳게 구현된 시스템들이 이러한 이슈로 인해서 운영되지 못하는 것을 사전에 예방하는 방향으로 아키텍쳐가 구상되어야 한다.

- 거시적 관점에서 어플리케이션들을 도출정의함에 있어 서비스의 유효성(Availability)과 신뢰성(Reliability)이 극대화되도록 하여야 한다.

- 거시적 관점에서 어플리케이션들을 도출정의함에 있어 특별한 경우 어플리케이션의 독자적 운용성(Self Sustainability)을 가지도록 구상되어야 한다. 독자적 운용성 부여는 그것과 상호 보완토록 정의된 타 시스템이 어떤 이유로 구현되지 안는 경우에도 독자적으로 서비스가 가능할 수 있게 하기 위함이다.

- 거시적 관점에서 어플리케이션들을 도출정의함에 있어 그것의 규모가 사업성을 가질 수 있도록 구상되어야 한다. 이는 추후 관련 개발 구현 사업의 원활한 추진을 위함이다.

이러한 관점에서 구상된 거시아키텍쳐는 ITS관련 모든 어플리케이션 시스템의 환경(Context)를 정의하는 것으로서 해당 시스템에 대한 거시적 관점의 요구사항을 도출 정의하는 것이다.

2. 객체 지향 방식의 미시 아키텍쳐(Micro-Architecture) 구상.

(국제표준에서 논리아키텍쳐(Logical Architecture) & 물리아키텍쳐(Physical Architecture)에 해당)

미시 아키텍쳐란 거시적 관점에서 도출 정의된 어플리케이션 시스템 각각에 대한 구성 틀을 단기는 물론 중장기적 차원에서 구상하는 것으로서 그 목적은 정의된 서비스 기능을

원활하게 수행하기 위해서 적절한 논리적 및 물리적 자원들을 합리적으로 할당하기 위함이다. 이 수준에서의 아키텍처를 구상함에 있어 주요 이슈는 어플리케이션 시스템의 성능이나 신뢰성 그리고 내부 요소간또는 외부요소와 상호운용성 등이고 또한 재사용성 도모 등 비교적 상세수준의 이슈들이기 때문에 객체지향 방식의 모형과 방법론을 통해서 명확하게 도출 정의하는 것이 바람직하다 하겠다. 모델 형식으로서는 이미 Defacto 표준으로서 자리를 굳힌 UML[11]을 따른다.(국제표준 TR14813 과 같음) 그러나 방법론으로서는 OOSE 방법론은 USECASE 모형의 상세화 과정이 분석/설계 모형에 어떻게 해당하는 가가 모호하기 때문에 이를 지양하고 대신 UP 방법론을 따르기로 한다. 다만 이 중 상위 2개 수준을 미시 아키텍처로 정의하고 하위 2개 수준은 설계 사항으로 정의한다. 미시적 아키텍처 구상시 고려사항을 살펴보면 다음과 요약될 수 있다.

<미시 아키텍처 구상시 고려 사항>

- 기능적 적합성(Working System)
- 품질적 적합성(Workable System)
- 재 사용성(Reusability)
- 유연성(Modifyability & Extensibility)
- Project Consideration : 계획된 예산 및 기간 한도에 가능한가
- Business Consideration : 사업성
- Technical Analysis : 기술적 구현성
- Manufacturing Evaluation : 제품의 공급성
- Human Issues : 해당 인적 자원의 충분성
- Environmental Interfaces : 인테페이스의 합리성
- Legal Consideration : 법 제도 조직적 수용성

III. 객체지향 아키텍처 수용 방식

미국 유럽등 각 국의 아키텍처가 포용하는 영역은 거시적 관점에서 미시적 관점까지 모두 포용하고 있다.[4,5,7,8,9] 그러나 우리 나라의 경우는 국가 아키텍처가 거시 아키텍처에 초점이 맞추어져 있기[1,2,14] 때문에 미시 아키텍처를 객체지향 방식으로 구상하기 위해서 중첩됨이 많지 않다. 또한 미시 아키텍처는 거시 아키텍처와 설계의 중간 단계이기 때문에 아키텍팅 단계에서 구상될 수도 있고 설계에 선행하는 기본 설계 단계에서 구상될 수도 있다 하겠다. 따라서 다음과 같이 3 가지 수용 방식이 가능하다 하겠다.

(방식-1)

K.ITS를 위한 국가 아키텍처는 기 구상된 4개 수준 아키텍처 중 제3수준까지만 포함하여 거시 아키텍처/핵심-참조 아키텍처(Core Reference Architecture)로 확정하고 제4수준은 시스템별 미시적 아키텍처로서 논리 아키텍처와 물리 아키텍처를 객체지향 방식으로 구상한다. 이 경우 국가 아키텍처에서 기 구상된 제4수준 물리적 구성체계는 미시 아키텍처의 논리 아키텍처 구상시 패키지 모형화 단계에서 패키지로 사용될 수 있다. 미시 아키텍처 구상 개발은 영역별 표준화팀과 사업별 사업주체에 의해서 기본 설계 사항으로 구상하도록 한다. 본 방식의 주요 장단점을 평가해보면 아래와 같다:

- 기 개발된 국가 아키텍처(프로세스 지향)를 상당부분 그대로 수용.
- 소요 기간 및 소요 예산의 정확성 및 배정의 수월성 극대화
- 국제 표준 진행과정에 유연하게 대응할 수 있기 때문에 국제표준의 완성 전에
  조기 수용이 가능함.
- 국제표준을 부분적으로 또는 조기에 수용함으로서 국제적 호환성의 문제 소지가 있음.
- 전체적(사업간) 일관성이나 재사용성이 저해될 위험성 있음.

(방식-2)

방식-1에서 전체적 일괄성이나 재사용성 문제를 해결하는 방식으로서 미시 아키텍처에 대한 구상을 기존의 국가 아키텍처 개발 팀이 객체지향 방식으로 재 구상하여 국가 아키텍처 V.2로 재 공포하는 방식으로서 다음과 같은 장단점을 가진다:

- 전체적 일관성이나 재사용성 극대화
- 사업개발이 지연된다(아키텍처 확정 후 개발을 착수해야 함)
- 소요 기간 및 예산 배정의 어려움
- 2002.3월부터 재개될 국제 표준 진행과정에 대해서 유연한 대응이 어렵게 되기
  때문에 국제표준이 확정된 이후에 이 방식으로 수용하는 것이 바람직.(2005년 이후)

(방식-3)

방식-1에서 국제표준에 대한 부분적 호환성 문제를 해소하는 방식으로서 시스템 지향 방식으로 구상된 거시 아키텍처 또한 객체지향 형식으로 전환한다. 즉 제3수준에서 도출 정의된 60개 서브시스템을 use-case로 간주하고 영역별로 USE-CASE 모형화하고(확장된 USE-CASE 모형 사용) 각 use-case 별(또는 영역별)로 패키지 수준의 객체를 사용해서 고 수준 SEQUENCE 모형을 구상하여 이를 핵심 참조아키텍처(Core Reference Architecture)로서 국가 표준화한다. 이를 바탕으로 해서 영역별로 논리 아키텍처와 물리 아키텍처를 표준 개발 팀과 시스템 개발자가 공동으로 구상(Elaboration)하며 여기서 도출 정의되는 데이터와 메시지 형식을 표준화하여 국가 표준화한다. 본 방식은 여타 방식에 비해 상대적으로 여러 가지 강점을 가진다. 주요 강점을 요약하면 아래와 같다:

- 국제표준과 완전 호환성을 가짐(거시 및 미시 아키텍쳐를 모두 객체지향으로)

- 미시 아키텍쳐 구상과 표준화와 사업개발을 동시에 도모함으로서 상호 시너지효과

극대화

- 거시와 미시 아키텍쳐를 동시에 구상함으로서 시너지 효괄를 얻을 수 있다.

- 사업자의 개발비용 및 표준화 예산으로 아키텍쳐 구상을 지원(예산 지원의 수월성)

- ISO TC204의 아키텍쳐 표준화 작업 구조와 일치한다. WG1에서 핵심 참조아키텍쳐만

표준화하고 이를 바탕으로 영역별 WG에서 영역에 대한 논리 아키텍쳐와 물리

아키텍쳐를 개발해서 표준화하는 구조와 호환성을 가지기 때문에 국내외적으로 작업이

수월함

- 국제 표준 진행과정에 비교적 유연하게 대응할 수 있기 때문에 국제표준의 완성 전에

조기 수용이 가능함.

- 국가 아키텍쳐 구상 작업과 관련 표준화 작업 및 구축 사업들이 상호 연계되어

추진됨으로서 전체적(사업간) 일괄성과 사용성이 극대화 될 수 있음.

- 사업관리(프로젝트 관리)가 복잡하고 어렵다.(이에 대한 대책으로서는 표준화 팀

중심으로 일이 추진되고 또한 관리가 이루어진다면 수월할 것으로 사료됨)

이상의 세 가지 수용방식을 비교 분석해 보면 여러 가지 관점에서 (방식-1)이나 (방식-3)이 적합하다고 사료된다. 이 경우 응용시스템 별로 도출 정의되는 개체(Control Classes, Information Classes, Interface Classes)들과 개체간의 메세지들이 서로 상충되지 않고 일관성 있게 재 사용되기 위해서는 이들 개체들이 우선 영역내에서 일관성/호환성을 가지도록 영역별 데이터 사전(National Data Dictionary)에 엄정한 절차를 걸쳐 등록관리 되고 사용되어져야 하며 영역간 또는 국가간의 일관성/호환성을 유지하기 위해서는 국가의 중앙 등록소(National Data Registry)에 엄정한 절차를 걸쳐 등록되고 관리되고 그리고 이를 통해서 만 사용 되어지도록 해야 한다. 이를 위해서는 국제표준 ISO CD 14817(Requirement for DR/DD)[14]을 따르는 것이 필수적이라 하겠다.

IV. 결 론

객체지향 방식의 아키텍쳐 구상은 객체지향 아키텍쳐 모델링과 방법론에 준거하여 단계별로 추상화/구체화 과정을 통해서 이루어진다. 모델링 과정에서는 우선 정적 모델링으로서 시스템별 관련 모든 액터(Actor)와 이해당사자(Stake Holder)가 도출 정의되어야 하고 또한 액터별 쓰임새(Usecase)와 관련되는 객체 유형(Class)들이 도출 정의되어야 한다. 그리고 이들의 상호 관계를 정의하는 USECASE Diagram과 CLASS Diagram이 구상되어야 한다. 다음은 동적모델링으로서 Usecase 별 SEQUENCE 또는 COLLABORATION Diagram이 구상되어야 한다. 이러한 모델링 작업은 아키텍팅 방법론적 과정에서 단계별로 보다 구체화 되고 상세화 되며 최종적으로 물리아키텍쳐 단계의 모델링으로서 COMPONENT Diagram, PACKAGE Diagram 그리고 DEPLOYMENT Diagram등이 구상된다. 그러나 이렇게 여러 수준에서 여러 가지 모형으로 정의되는 객체지향 아키텍쳐는 그것의 명확성과 상세성 그리고 일관성등에서 강점을 가지기 때문에 관련되는 다양한 시스템들을 통합해서 상호연동성을 가지게 하는 수단으로서는 적합하지만 보다 큰 수준에서의 이해당사자 조직간의 역할적 이해 충돌을 합리적으로 조정하는 수단으로서는 취약성을 가진다.

그러나 앞장에서 제시된 (방식-1)을 취하는 경우는 두 가지 차원에서(거시적 및 미시적)의 아키텍쳐를 두 가지 방식(프로세스 지향 방식과 객체지향 방식)을 사용해서 상호보완적으로 사용하기 때문에 기존의 프로세스/시스템 지향 방식 아키텍쳐에서의 취약점인 미시적 관점에서의 상호연동성 ,재사용성, 상세성, 일괄성의 문제가 크게 개선될 수가 있으며 또한 객체 지향방식 아키텍쳐에서 취약점인 거시적 관점에서의 합리적 정의가 이루어 질 수가 있다. (방식-3)을 취하는 경우는 프로세스 지향 방식으로 구상된 거시 아키텍쳐를 취하되 그 형식만은 객체지향 방식으로 전환하고(내용은 그대로 취함) 미시 아키텍쳐는 그 내용과 형식 모두를 객체지향 방식으로 구상함으서 (방식-1)과 내용적으로 거의 비슷하지만 형식적으로는 전체가 객체지향 방식을 취함으로서 국제표준과 완전 호환성을 가지게 되는 강점이 있다.

그러나 여기서 주의해야 될 점은 객체지향 방식을 취하더라도 최종 구축되는 시스템은 기존 국가 아키텍쳐에서 도출 정의된 63개의 서브시스템이 준수되어야 한다는 점이고 이들 간의 관계로서 도출 정의된 흐름정보 또한 준수되어져야 한다는 것이다. 다만 기 정의된 흐름 정보들은 ISO CD 14817에서 정의된 의미와 형식을 갖추도록 보완되어야 하고 객체간의 메시지 형식은 추가로 도출 정의되어야 한다.

또 다른 주의 점은 아키텍쳐 구상이 시스템 정의 사항이냐 아니면 설계 사항이냐 하는 문제이다. 특히 미시적 아키텍쳐의 구상은 기술 의존적이기 때문에 설계 (Design) 속성이 강한 정의(Definition/Specification)로 보아야 한다. 따라서 기존의 프로젝트 수행 방식에서 벗어나기 어려운 경우는 이를 정의 단계가 아닌 설계 단계 (기본설계) 에서 시스템 통합자 (System Integrator)에 의해서 수행하는 것도 바람직하다 하겠다. 이는 아키텍쳐 구상을 위한 예산 및 기간 설정에서 중요한 역할을 하기 때문에 신중하게 고려되어야 할 것이다.

참고문헌

1. 국토개발연구원, "국가 ITS 아키텍쳐 확립을 위한 연구", 최종보고서(Part-1), 1998.12
2. 국토개발연구원, "국가 ITS 아키텍쳐 확립을 위한 연구", 최종보고서(Part-2), 1999.12
3. IEEE P1471/D4.1, "Draft Recommended Practice for Architectural Description", Architecture WG of SESC IEEE, Dec., 1998
4. US DOT, FHA, "ITS Architecture: Physical Architecture", Oct., 1995
5. US DOT, FHA, "ITS Architecture: Logical Architecture", Oct., 1995
6. ISO/TC204/WG1/N300, Transport Information and Control System (TICS): Reference Architecture Tutorial, Oct., 1997
7. ERTICO, "European ITS Frame Architecture", V1.1, March, 2002
8. ITS-Australia, "Australian ITS Architecture",
9. VERTIS, "System Architecture for ITS in Japan", Nov., 1999
10. Leen Bass, Paul Clements, Rick Kazman, Software Architecture in Practice, Addison Wesley, 1998
11. Hans-Erik Eriksson, Magnus Penker, UML Toolkit, John Wiley & Sons Inc., 1998
12. I.Jacobson, Object Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992
13. R. S Pressman, Software Engineering (4th Edition), McGraw-Hill, 2000.
14. IS14817, "Requirements for an ITS/TICS Central Data Registry and ITS/TICS Data Dictionaries",ISO/TC204/WG1, 2002
15. 백인섭,이승환,이시복, "국가 지능형교통체계를 위한 아키텍쳐 :모형 및 방법론",대한교통학회 논문지,제19권,제6호,대한교통학회,2001.12
16. I.Jacobson, G.Booch, J.Rumbaugh, The Unified Software Development Process, Addison Wesley,1998