제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

Use case model 의 상세화에 따른 consistency checking 방법에 관한 연구

이은영, 백인섭, 심우곤
아주대학교 정보통신전문대학원
e-mail : nescafe@ajou.ac.kr

Automated consistency checking method in use case model at the level of abstraction

Eun-young Lee, In-sup Paik, Woo-gon Shim
Graduate school of Information and Communication AJOU University

요 약

객체지향 환경에서 복잡한 소프트웨어 시스템을 개발하기 위해서는, 그것의 복잡성과 대규모성 때문에 추상화에 의한 다계층적인 use case model 의 사용이 불가피하다. 이러한 경우 모델의 consistency 유지가 매우 주요하고 어려운 이슈가 된다. 본 논문에서는 각 추상화 단계에 따른 use case model 들 사이에서 자동적으로 형식적인 consistency 를 체킹할 수 있는 방법을 제안한다. 이 접근 방법은 rule 을 기반으로 하여 actor tree, use case composition diagram 및 use case description 을 활용한다. 본 접근법을 검증하기 위하여, ITS 아키텍처 (Intelligent Transportation System architecture)의 한 파트를 예로 들어 적용하였다..

1. 서론

대규모의 복잡한 시스템을 개발할 때, 소프트웨어 아키텍처는 복잡한 소프트웨어 시스템을 구조화하고, 높은 수준에서 시스템을 묘사하여 시스템의 신뢰성 향상과 개발시간 및 비용을 줄이기 위한 수단으로 인식되었다[9][10]. 그러나 높은 추상화 수준의 아키텍처로도 관리하기가 어려운 대규모 시스템의 경우에는 여러 계층으로 추상화하여 나타내는 것이 바람직하다.

개념적인 서비스들을 정의하는 요구사항분석으로 시작하는 시스템 아키텍팅은 최근 표준모델링 언어로 선정된 UML[24]의 use case model 이 표기의 용이성으로 인하여 각광을 받고 있다.

초기에 간단하게 표현된 아키텍처는 개발단계에 따라 점점 상세하게 표현되고, 이에 따라 시스템이 서비스해야 할 요구사항을 표현하는 use case model 도 추상화 계층에 따라 점점 상세하게 표현된다. 각 추상화 계층에서 만들어지는 아키텍처 모델들 사이의 consistency 는 깨지기가 쉽고, 이는 후속하는 모든 단계에 엄청난 영향을 끼칠 수 있다. 모델들 사이의 inconsistency 는 시스템 개발 비용의 증가와 지체, 심지어 프로젝트의 실패로까지 이어질 수 있으므로 consistency 의 관리는 요구공학에서 중요한 도전이 되고 있다[5]. 모델들 사이의 consistency 를 유지하는 것은 명백하게 공학적으로 소프트웨어의 질을 향상시키고, 작업의 효율성을 개선하는 것으로 매우 중요한 것이다. 그러나 사람이 consistency checking 을 완벽하게 수행하는 것은 매우 어렵다. 특히 많은 다이어그램과 산출물을 가진 매우 큰 모델에서 에러나 특정 생략부분을 찾는 것은 사람에게 지루하고 놓치기 쉬운 작업이기 때문에, 자동화된 모델 consistency checking 방법이 요구된다. 자동 모델 consistency checking 의 이점은 다음과 같다: 1) 포착하기 어려운 에러를 찾아낼 기회가 증가한다. 2) 복잡성을 효과적으로 다룰 수 있다. 3) 에러의 탐지가 쉬어진다. 4) 기본적인 에러를 간단하게 밝혀낼 수 있다[11].

최근 UML 다이어그램들 사이의 consistency checking 방법이 활발히 연구되고 있다; consistency 에는 크게, 같은 추상화 계층에서 다른 파트를 나타내는 모델들 사이의 horizontal consistency 와 다른 추상화 계층에서 같은 파트를 나타내는 모델들 사이의 *vertical consistency* 가 존재한다. 그러나 use case diagram 들간의 계층에 대한 자동화된 consistency checking 방법은 아직 존재하지 않는다[5][11]. 안타깝게도, vertical consistency checking 은 자동화하기가 어렵고[11], use case 상세화(refinement)의 개념이 강하게 대두 된지도 얼마 되지 않았기 때문이다.

그러므로 본 연구에서는 계층적인 use case diagram 에서 자동으로 consistency 을 checking 할 수 있는 방법을 제안하고, 이를 구현하고자 한다. 우리가 제안한 방법을 검증하기 위하여, 한 예로 ITS 아키텍처 (Intelligent Transportation Systems Architecture)의 한 부분을 취해 이 방법을 적용하여 보도록 한다.

ITS 에 대한 국제 표준인 ISO/TC 204 [23]에 의하면, ITS 아키텍처 수립 시 use case model 은 기능적 요구사항 (functional requirements)을 밝히고 기록하기 위한 필수적

인 과정으로 명시되어 있다. 또한 국가적인 규모와 복잡도로 인하여, 여러 명의 각 분야별 전문가들이 시스템을 분석하게 되는데, 하나의 use case diagram 만으로는 커버할 수 없기 때문에 use case diagram 간의 계층화를 통하여 시스템에 대한 명세작업을 수행하였다. 상세화 작업을 진행함에 따라 다이어그램은 점차 복잡해지고 여러 사람이 작업하기 때문에, 모델들 사이의 consistency 가 깨질 가능성이 점차 높아진다. 그러므로 ITS 아키텍처는 consistency management 가 필요한 좋은 예이다.

본 논문은 2 장에서 consistency 의 종류와 use case model 추상화 계층의 개념을 소개한다. 3 장에서는 consistency checking 을 위한 rule 기반의 방법을 제안하고 시스템 아키텍처를 묘사한다. 4 장은 TICS 아키텍처의 한 예에 방법을 적용하고 5 장에서 결론 및 향후 작업을 요약한다.

2. 관련 연구

UML 에는 actor 와 use case 사이의 관계를 나타내는 "communicate"와 use case 들 사이의 관계를 나타내는 "use" 두 가지 표준 관계가 존재한다. 그러나 복잡도를 줄이기 위하여 이 논문에서는 "use" 는 고려하지 않도록 한다.

2.1 Consistency 의 차원

Consistency 의 차원은 horizontal consistency 와 vertical consistency 로 분류된다. horizontal consistency 는 시스템의 다른 파트를 나타내는 모델들 사이에서, 같은 수준의 추상화로 사용자의 관점을 나타낸다. vertical consistency 는 시스템의 같은 파트를 다른 추상화 계층으로 나타내는 모델들 사이의 consistency 를 의미한다. 이는 전형적인 상세화(refinement)이다. vertical consistency checking 은 종종 다이어그램에서 문자정보의 비교와 다른 추상화 계층을 포함하는 애매모호성을 가지고 있어서 아직 실현되지 않았으나, horizontal consistency checking 은 비교적 명확하여 자동화로의 많은 연구가 이루어져있다.

이 연구에서 정의하는 consistency 는 시스템의 같은 부분을 나타내는 use case model 을 다른 추상화 계층에 따라 나타내는 모델간의 vertical consistency 를 의미한다. ITS 아키텍처는 매우 복잡하고 거대한 시스템이기 때문에 요구사항이 매우 많고 다양하므로, use case diagram 이 복잡해진다. 상위 모델에서 나타난 use case 는 다음 계층에서 use case 의 set 으로 refine 된다. 즉 위 계층의 use case 는 다음 계층의 use case 의 set 을 형성한다. 만약 다음 하위단계의 상세화 계층에, 상위단계의 use case 에 해당하는 세부 use case 가 존재하지 않으면 그것은 consistency 가 결여되었음을 의미한다. inconsistency 는 개발 과정에서 한참 후에 탐지되어 잘못된 개발을 유도하고, 개발 비용의 증가와 지체 및 시스템 개발 실패의 원인이 된다. 그러므로 시스템 개발이 더 진전되기 전에 요구사항 분석 단계에서부터 inconsistency 를 찾고 제거하는 것이 좋다.

3. rule 을 기반으로 한 consistency 유지 방법

Use case diagram 들 간의 vertical consistency 를 checking 하는 방법은 거의 없다. 그 이유로는 use case 상세화 개념이 주목 받기 시작한 것이 얼마 되지 않았으며, use case 를 기술하는 방법이 너무나 비정형적이어서 checking 자체가 쉽지 않기 때문이다. 더구나 use case 를 지침 정도로 활용하면서 class diagram 이나 UML 의 기타 다이어그램으로 해결하는 것과, use case 에 중점을 두고 기타 diagram 들을 use case 의 모호함을 해소하기 위한 부수적인 도구로 활용하는 방법 중 전자의 것에 연구가 치중해 있는 것도 그 원인으로 작용한다. 그러나 복잡하고 규모가 큰 시스템에서는 use case diagram 들간의 계층화가 필수적이기 때문에, 각 계층 간 vertical consistency 의 유지를 소홀히 해서는 안 된다. 본 절에서는 rule-based solution 에 기반을 둔 방법을 제안한다. 본 논문에서는 오직 use case 계층의 형태론적 에러만을 고려하고, extend 와 include 와 같은 use case 사이의 relationship 은 복잡성을 감소하기 위해 고려하지 않았다.

3.1 Rule based solution

Rule based solution 의 장점은 백여 개, 천여 개의 rule 을 저장하여 진행하고, 간단히 rule base 에서 추가하고 수정하는 것에 의해 업데이트 할 수 있는 것이다. Rule base 에서 새로운 rule 을 만들고, 새로운 모델영역에서 consistency 유지를 위한 필요성에 따라 새로운 rule 을 만들 수 있다[2]. use case 추상화단계의 consistency checking 을 위해서는, actor tree, use case composition diagram, use case description 이 필요하다.

Actor tree 는 actor generalization relationship 의 표현으로 actor 는 root actor 를 기반으로 한 hierarchies 로 나타난다. 가장 최상위 단계의 actor 를 root actor 라 하고, 이 root actor 에서 상속을 받는 actor 들이 나타낸다.

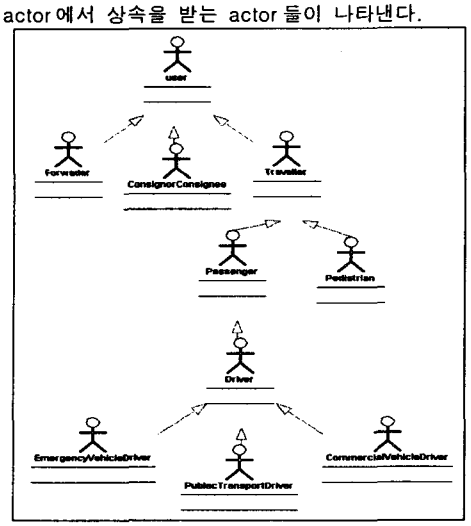

그림 1. actor tree 의 예.

Use case composition diagram 이란 use case 상세화에 의하여 상위 use case 와 하위 use case 의 관계를 나타낸 것이다. 상위 use case 는 하위 시스템에서, 여러 개의 작은 use case 의 set 으로 나타내어진다. 계층에 의하여 나타내어지는 use case 의 세부 항목은 actor 에 비하여 엄격하게 나타난다.

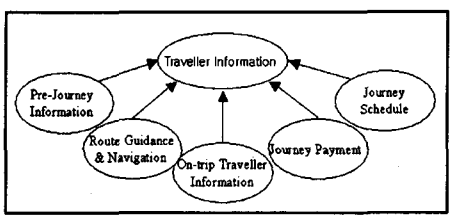

그림 2. use case composition diagram 의 예

이 논문에서 정의하는 Use case description 이란 use case 를 나타내는 간단한 요약을 의미한다. 주요성공 시나리오를 표현하고, 각 use case 와 직접적으로 대화하는 actor 의 이름을 모두 포함하고 있다. Use case diagram 의 actor 의 이름과 use case description 상의 actor 의 이름을 비교하여, use case diagram 에서 use case 와 연결된 모든 actor 가 use case description 에 존재하는지 확인한다. Description 의 형태소 분석을 통하여, description 에 기술되지 않은 actor 가 use case diagram 상에서 연결관계를 가지면, 이것은 inconsistency 하다고 할 수 있다.

표 1. use case description 의 예

Journey payment
These transactions provide for trip conformation and enable the traveler to make advanced payments. They provide payment for each segment (trip) of a pre-planned journey. *The Traveller actor* must provide the means of payment at the Travel Terminal interface. Actor: *Traveller*

3.2 시스템 개요

consistency 를 유지하기 위한 rule 이 저장되어 있는 rule 기반 시스템이 존재한다. 여기서는 rule 을 수정하거나 업데이트할 수 있다. 여기에 consistency 를 checking 하기 위한 use case diagram 이 존재하고, 이 다이어그램을 consistency 의 rule 에 의하여 확인하기 위한 actor tree, use case composition diagram, use case description 의 문서가 존재한다. Consistency rule 을 기반으로 use case diagram 과 위의 세 문서를 비교하여, consistency 를 checking 한다.

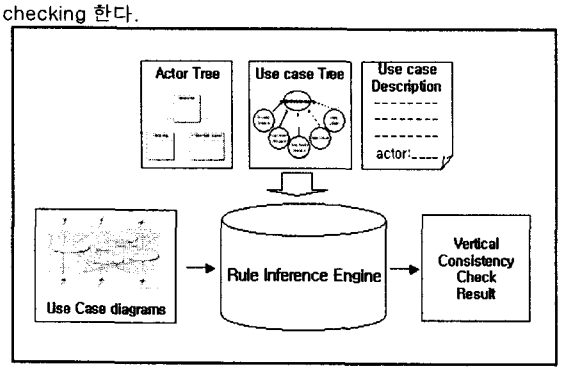

그림 3. 시스템 아키텍처

3.3 Consistency rules

여기서 제시하는 consistency checking rule 은 형태론적 에러에 한정한 것으로 매우 간단하다. 그러나 시스템 아키텍처 개발 시에 시스템의 요구사항을 표현하는 use case model 의 consistency 이므로, 여기서 발생하는 inconsistency 는 시스템의 초기에 발생하여 후속하는 모든 단계에 엄청난 영향을 끼칠 수 있으므로 간단하지만, 매우 중요한 rule 이라고 할 수 있다. 여기에 적용할 수 있는 consistency checking rule 의 예를 들면 다음과 같다.

Ex) Rule1. 최상위 use case diagram 에서 모든 actor 는 root actor 이다.

계층간의 관계를 명확하게 나타내기 위하여, 최상위계층에서는 root actor 만이 나타날 수 있다. 그러나 use case diagram 에서 actor 간의 계층별 구별은 아주 명확하게 이루어지지 않으므로, 단지 잠재적인 문제의 가능성을 가지고 있는 것이다. 최상위수준의 다이어그램에 존재하는 actor 를 actor tree 와 비교한다. actor 가 root actor 가 아니면 그것은 inconsistency 의 가능성이 있다.

Ex) Rule 2. 상위단계에 있는 actor 는 하위 단계에서 generalization relationship 있는 actor 중의 하나로 존재한다.

상위계층의 actor 는 하위계층의 actor 와 같은 계층에 있거나 하위 계층에서 나타난다. 하위 계층의 use case diagram 에서 나타나는 actor 는 상위 use case diagram 에서 나타낸 actor 의 상속을 받는 것으로 상위 계층에서 나타난 actor 와 동일하거나 하위 actor 로 표현되어야 한다. 그러므로 actor tree 와 비교하여 상위 use case 의 각 actor 는 하위 use case 와 연결되는 최소 하나의 actor 로 나타난다.

Ex) Rule 3. 상위 계층에 있는 use case 는 하위 계층의 child use case 와 mapping 된다.

use case 의 이름과 수는 use case composition diagram 에서 child use case 와 부합한다. 상위 계층에서 표현된 use case 가 하위 계층에서 나타날 때, use case composition diagram 에 나타난 명확한 계층별 구분에 의하여 표현되어야 한다. 상위 계층에서 하위로 갈 때, 한 use case 가 4 개의 use case 로 세분화 된다면, 그 use case 의 수와 이름이 모두 같은지 확인한다. Use case 의 수가 모자라거나 많거나, 이름이 다르면 이것은 inconsistent 한 것이다.

Ex) Rule 4. Generalization 관계의 actor 는 같은 계층에서 나타나지 않는다.

Generalization 관계는 특성과 연결관계를 상속 받았다는 것을 의미한다. 그러므로 generalization 관계의 actor 가 한 다이어그램에서 동시에 나타나는 것은 중복을 의미하는 것으로, 잠재적인 오류의 가능성이 있다. 이는 use case 경우에도 적용될 수 있으나, 이미 각 계층별 명확한 use case 관계를 정의 했으므로 고려하지 않아도 된다

Ex) Rule 5. use case 에 연결된 actor 는 use case description 에 존재한다.

Use case 와 연결된 actor 는 이미 use case description 에 존재하여 있다. 형태소 분석을 통하여, use case description 상에 나타나는 actor 의 이름과 use case diagram 상에 존재하는 actor 의 이름을 비교한다. 만약 use case description 에 나타나지 않는 actor 가 use case 와 연결되어 있다면, 이는 잘못된 연결이다. 그러므로 이는 consistency 에 위배된다.

이 장에서는 consistency rule 을 정의하고 기본적인 rule 을 예로 들었고, 이를 적용하여 consistency checking 하는 것을 다음 장에서 제시하고자 한다.

4. 시뮬레이션: ITS 아키텍처

4.1 ITS architecture

국가 ITS architecture 는 반드시 수행해야 하는 기능성을 정의하고, 이러한 기능을 제공하는 하위시스템과 정의자가 서비스를 지원하기 위해 변화되어야 하는 정보를 정의한다.

이 논문에서 우리는 ITS 아키텍처에서 국제 ITS 표준에 따라 use case model 이 계층별로 변화되는 과정을 나타내고, consistency 를 checking 한다. ITS 서비스에 의한 개발 시스템은 TICS 로, 국제 ITS 표준인 ISO/TC204 의 참조모델을 참고로 했다. Use case 들 사이의 관계는 복잡성을 줄이기 위하여 생략했으므로 " use" association 은 이 논문에서는 나타내지 않았다. 다음에 나타나는 다이어그램은 ISO/TC204 의 참조 모델인 TICS 의 최상위단계의 use case diagram 을 나타낸 것이다.

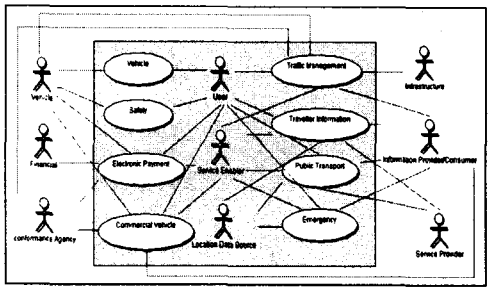

그림 4. TICS 최상위 use case diagram

최상위에서는 8 개의 use case 서비스와 9 개의 actor 로 표현된다. 이 최상위수준의 use case diagram 이 다음 계층으로 상세화하여 표현되면 아래와 같다.

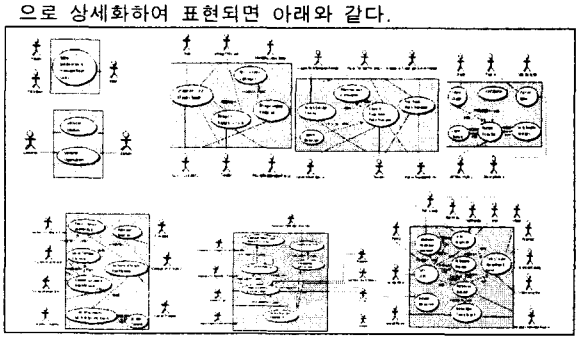

그림 5. 두번째 단계의 TICS 의 use case diagram

단지 한 수준만 상세화되었음에도 불구하고 매우 복잡해지고, 커졌음을 확인할 수 있다. 그러므로 use case diagram 이 점점 상세화 되어감에 따라 diagram 은 더 커지고 더 복잡해지며, 사람이 consistency 를 checking 하는 것은 매우 어려워진다고 할 수 있다.

4.2 Rule 1 의 적용

이 참조 모델에 rule1 을 적용하여 consistency 를 checking 한다.

Rule1. 최상위 use case diagram 에서 모든 actor 는 root actor 이다.

그림 4 의 다이어그램을 그림 1 의 actor tree 와 비교하면, conformance agency 와 location data source 는 service enabler 의 하위 actor 로서 root actor 가 아님을 알 수 있다. Conformance agency 와 location data source 가 연결된 use case 는 service enabler 도 모두 연결되어 있으므로, 엄격하게 계층별 consistency 을 적용하면, 중복되어 나타난 것으로 inconsistency 의 가능성이 존재한다. 이를 제안한 방법을 바탕으로 하여 자동적으로 checking 하면 아래와 같이 검출됨을 알 수 있다.

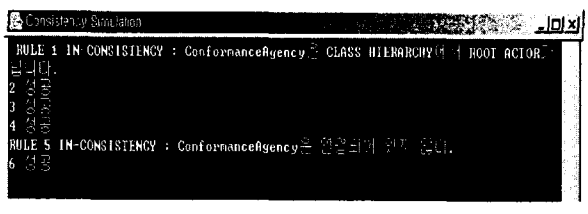

그림 6. Rule 1 의 consistency checking 결과

5. 결론

본 논문에서는 actor 간의 추상화 수준과 use case 의 구체화 수준을 각 단계별 use case model 에 대응시켜 consistency 파괴를 자동 검색하는 방법을 제안하였다. 이는 actor tree, use case composition diagram, use case description 의 새로운 개념의 문서가 필요한 rule 기반의 solution 이다.

향후에는 actor tree 와 use case composition diagram 의 개념 및 내용을 보완 확장하여 use case diagram 들을 통해 자동 생성할 수 있도록 하고, 적용 분야와 범위를 넓혀 현재 형태론적 에러(syntax error)를 checking 하기 위한 rule 을 보강함은 물론 의미론적 에러(semantic error)를 발굴해 낼 수 있도록 rule 을 확장해 나가는 연구가 필요하다.

참고문헌

1. Amyot, D. and Mussbacher, G.: On the extension of UML with use case maps concepts. UML 2000, York, UK, volume 1939 of LNCS. Springer (2000)
2. Boehm, B: Software Engineering Economics. Prentice-Hall, Englewood Clips, NJ (1981)
3. Dorfman, M.: Requirements Engineering, (1999)
4. Eeles, P.: Capturing Architectural Requirements, the rational edge rational software 2001
5. Engels,G., Kuster, J. and Heckel, R.: Methodology for specifying and Analyzing Consistency of Object-Oriented Behavioral Models
6. Giandini, R., Pons, C., Pérez, G.: Use Case Refinements in the Object Oriented Software Development Process
7. Gryce, C., Finkelstein, A. and Nentwich, C.: xlinkit: Lightweight Consistency checking for the UML http://www.xlinkit.com
8. Hausmann, J., Heckel, R., Taentger, G.: Detection of Conflicting functional Requirements in a Use Case Driven Approach (A static analysis technique based on graph transformation), ICSE 2002
9. Inverardi, P., Muccini, H. and Pelliccione, P.: Automated Check of Architectural Models Consistency using SPIN
10. Inverardi, P., Muccini, H. and Pelliccione, P.: Checking consistency between architectural models using SPIN
11. Kielland, T., Børretzen, J.: UML consistency checking SIF8094 (2001)
12. Larman, C.: Applying UML and patterns (An introduction to object-oriented analysis and design and the unified process), 2nd edition, Prentice Hall PTR
13. Liu, W., Easterbrook, S. and Mylopoulos, J.: Rule-based detection of inconsistency in UML models
14. Loucopoulos, P., Karakostas, V.: System Requirement Engineering, McGraw-Hill, (1995) page 6-10.
15. Nuseibeh, B., Easterbrook, S. and Russo, A.: Making inconsistency respectable in software development, The Journal of Systems and Software, 58(2):171-180, (2001)
16. Pérez ,G., Giandini, R. and Pons, C.: Model Refinements in the object oriented software development process, Argentine Symposium on Software Engineering
17. Quatrani, T.: Introduction to the Unified Modeling Language
18. Rasch, H. and Wehrheim, H.: Consistency between UML Classes and Associated State Machines
19. Robinson, W., Pawlowski, S.: Managing Requirements Inconsistency with Development Goal Monitors, GSU Working paper CIS 97-4
20. Spanoudakis, G. and Zisman, A.: Inconsistency Management in Software Engineering: Survey and Open Research Issues, handbook of software engineering and knowledge engineering (2001)
21. Thayer, R., Dorfman, M.: Software Requirement Engineering (2nd edition) (IEEE Computer Society Press, 1997). An excellent tutorial volume of research papers in requirement engineering.
22. National ITS Architecture, http://itsarch.iteris.com/itsarch/
23. ISO TC 204/SC WG: Transport information and control systems-Reference model architecture(s) for the TICS sector – part 2: Core TICS reference architecture
24. OMG: Unified Modeling Language Specification, March 2003, Version 1.5, formal/03-03-01,