Intelligent consistency checking method for the use case model

Eun-young LEE[a], Woo-gon Shim[a], In-sup Paik[b],

[a] *Graduate school of Information and Communication, Ajou University*
San 5, Wonchun-dong, Paldal-gu, Suwon,442-749, Korea
Tel: +82-31-219-2638, Fax: +82-31-219-1614, E-mail:{Nescafe, startear}@ajou.ac.kr

[b] College of Information Technology, Ajou University
San 5, Wonchun-dong, Paldal-gu, Suwon,442-749, Korea
Tel: +82-31-219-2638, Fax: +82-31-219-1614, E-mail:ispaik@ajou.ac.kr

Abstract

In the development of complex software system, it is important to use hierarchical use case model due to the complex scope of development procedure. The use case model is core factor of the OMG (Object Management Group)'s UML (Unified Modeling Language) diagrams. In this paper, we propose a novel method to check syntactic consistency automatically in use case models at the different level of abstraction. This method is a rule-based approach which utilizes actor tree, use case tree and use case description. The proposed method is simulated on ITS (Intelligent Transportation System) architecture for the verification.

Keywords:

Use case model, Level of abstraction (Refinement), Vertical consistency, Rule based solution, Actor tree, Use case composition diagram, Use case description.

1. Introduction

In the development of very large and complex software system, software architectures have been recognized, as a means of improving the dependability of large complex software products, while reducing development times and costs [10].

In the '90s, software architectures appeared to structure complex software systems, describe a high-level system [9]. But in the case of very massive and complicate system, despite the high level of abstraction, often software architecture is too complex to be managed. To tackle system complexity like this, it is desirable to consist of the representation the system by several level of abstraction. Software architecting begins with requirement analysis on the definition of the conceptual services [8]. Although many different methods can be applied to this task, Use case model of the UML [24] approach selected as the standard language recently is highlighted for the easiness of notation.

Simple expression of the architecture at the first stage, is refined in detail according to the development stage [10], use case model is expressed different level of abstraction. Models of architecture produced at different levels of abstractions in the development process are supposed to inconsistent. Therefore, managing consistencies between models has been recognized as a major challenge in requirements engineering [15]. The reason for working with model consistency and improving model consistency is clearly to improve the quality of the engineered software and to improve the effectiveness of the work [11]. That is why *consistency* is of major importance [5].

But the consistency checking performed by human can probably not be executed perfectly, especially in large models with many diagrams and artifacts. The tasks such on searching out errors or certain omissions are tedious and likely to be easy to miss when inspecting manually. That is why automated model checking is necessary. The benefits from automated model checking are the followings; i) the chance of detecting subtle errors is increased. ii) ability handling complexity is effective. iii) the detection of errors is likely early. iv) basic errors identified simply [11].

Consistency checking about diagrams of UML has been acting researched; *horizontal consistency* between models of different parts or perspectives at the same level of abstraction and *vertical consistency* throughout models of the same part at different levels of abstraction exist. But it is only too true that automated model consistency checking methods in use case levels is not in reality [5] [11]. Unfortunately, the vertical consistency checking is difficult to automate and it is only short time since the concept of use case refinement became powerful [11].

The aim of this paper is to propose a method to check vertical consistency automatically throughout use case model at the level of abstractions and to implement it. To verify our method, we apply this method to one part of Intelligent Transportation System (ITS) architecture as an example. According to ITS standards ISO/TC 204 [23], in the establishment of ITS architecture, use case model is the

essence to discover and record functional requirements. And due to the properties of nation-wide scope and complexity of the system, there should be involved many different field experts. Before long, they recognized the system couldn't be covered with only one use case diagram. So, it needs to work with use case diagrams at different level of abstraction. Use case diagram comes to be complex as gradually the refinement. It is supposed to inconsistent because of communication problems among the people, system scale, and complexity. Therefore, ITS architecture is a good case needs the consistency management.

The structure of this paper is as follows: Section 2 describes the concept of requirement and use case model. And we introduce dimensions of consistency and the use case levels of abstraction. Section 3 proposes a novel method for consistency checking, defines the consistency rule and describes the system architecture. In section 4 applies the methods to one example of Transport Information Control System (TICS) architecture. Finally, section 5 draws conclusions and summarizes future work.

2. Related Work

Requirement engineering covers all of the activities discover, document, and maintain a set of requirements for a software system.

The first step of system development is requirement analysis. Requirements are the requests of stakeholders, are defined during the early stage of a system development as a specification of what should be implemented. Requirements describe how the systems should behave, are application domain information, constraints on the system's operation and specifications of a system property. In general, requirements can be divided into functional requirements and nonfunctional requirements [15] [18]. The *functional requirements* are services that the system is required to perform. The *nonfunctional requirements* are constraints on the operation of the system like usability, reliability, and performance. Functional requirements can be expressed as use cases. Use case model is an essential course that makes clear and reports functional requirement to help fulfill various stakeholders goal [13]. Use case model is explained in next section.

2.1 Use case model

As the first step when developing a system, the functional requirements of the system can be expressed with use case models. Two key elements of use case model are *use case* and *actor*. A use case is used to define the behaviors of the system, the functional requirements. An actor is a role outside of a system that interacts directly with a use case. In diagram, use case is described as an oval and actor represents as a stick man, they have associations between actors and use cases [Fig. 1].

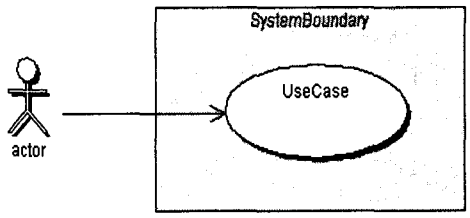

Figure 1 - This is a sample of Use Case Diagram.

The use case model helps technical experts and non-technical people alike understand that behavior. In other words, it provides a simple overview of the system functionality without the excessive details, which often confuse people with a less non-technical background. That sequences unambiguously spells out both what the user and the system will do, in the correct order, to accomplish the goal of the system

For this paper we will align (but not restrict) the notation of use case diagram to the OMG (Object Management Group)'s Unified Modeling Language (UML). There are two standard relationships between actor and use case, and between use cases. One is "communicates" that represents the participation of an actor in a use case. This is the only relationship between actors and use cases. The other is "uses" that is relationship between use cases [23]. But, in this paper, we do not consider "uses' association to reduce complexity.

2. 2 Dimension of Consistency

Dimension of consistency mainly divided into horizontal consistency and vertical consistency. *Horizontal consistency* is done between models of different system parts, user views at the same level of abstraction like a class diagram and a set of state charts which specifying the dynamic behaviour of the classes. *Vertical consistency* means between models of the same part or aspect at different levels of abstraction. It's typically refinements [11] [5].

Horizontal consistency checking is clear and will be most meaningful to automate, but then the vertical consistency checking often involves comparison of textual information with diagrams, and checking of different abstraction levels is not in reality yet. Consistency defined in this research is the vertical consistency about use case model at different level of abstraction of same view. Inconsistencies can lead to severe wrong development that is only detected much later in the process. It leads to increase the development cost, delay and fail of system development.

3. A method on rule-based solution

There are almost no methods to vertical consistency checking between use case diagrams. The reason is only short time since the concept of use case refinement became powerful and checking is not easy because use case notation is very informal. In addition, there are two

approaches of using UML diagrams: first approach is focused on the *class diagram* (static structure) and/or other dynamic behavioral diagrams, whereas it used to use the *use case diagram* just as guideline. On the contrary, second approach is mainly focused on the use case diagram, which called *use case-centric approach*. One of the reasons why the vertical consistency is rarely considered is that most of current research focused on the first one.

Complex and large system needs the use case models at the level of abstraction; we should not neglect a vertical consistency hold between them. In this section we propose a method on rule-based solution. We only consider the syntax error of use case levels and do not consider the relationship like extend and include to reducing complexity.

3.1 A rule-based method

The advantages of rule-based solution are that it can store and process hundreds or thousands of rules and that it can be updated, by simply adding or modifying rules in its rule base. We do not propose to make hundreds or thousands of rules, but using a rule based point of view, one can make new rules as one sees a need for consistency in a new area of the model [13].

For the consistency checking between use case models at different level of abstraction, it utilizes actor tree and use case composition diagram, use case description [Table 1].

Table 1 - Some artifacts needed to consistency checking in use case model at different level of abstraction

Artifacts	Definition	Contents
Actor Tree	Generalization relationship of actors	Root actor, Superordinate actor, Subordinate actor
Use Case Composition Diagram	Generalization relationship of use cases	Superordinate use case, Subordinate use case
Use Case Description	Text document, terse summary of the main success scenario	Actor's name

The *actor tree* means the representation of actor generalization relationship. Actor is an external class of the system, defines a role performed by a human or type of system. These actors are represented as hierarchies based on root actor. Actor in topmost level is root actor. Actors inherits from root actor are represented in actor tree. Figure 2 is a sample of actor tree. Actor tree represents to stickman or class type with stereo type <<actor>>. It is a part of actor tree in TICS reference architecture. The actor of 'User' represents the entire human, which obtain end user service from the transport system. Actors identified with these roles are shown hierarchically in Figure 2.

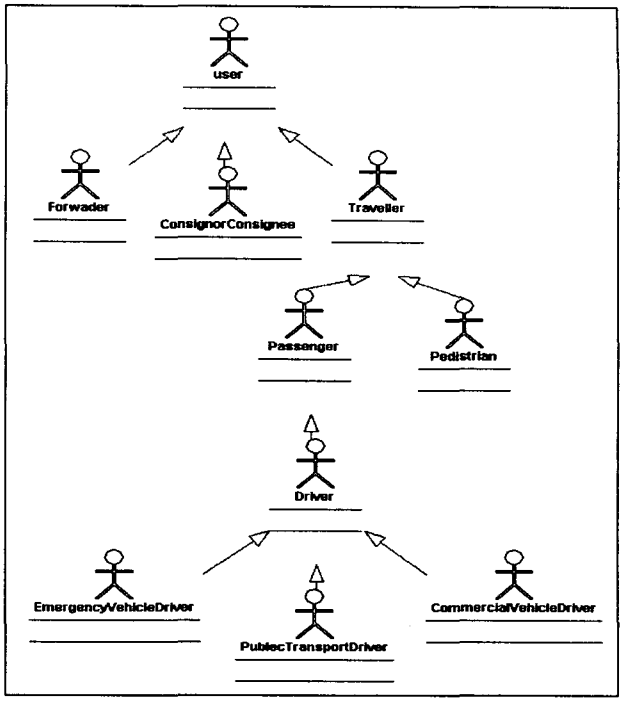

Figure 2 - This is a sample of actor tree.

The *use case composition diagram* means relationship between parent use case and child use cases according to the use case refinement. Use case in super level represents a smaller use case set in sub level. Specifying use cases by level make a sharp distinction compared with actor. The use case composition diagram is used to clarify the relationship between super ordinate use case and subordinate use cases. Figure 3 is a sample of use case composition diagram. It is a part of use case composition diagram in TICS reference architecture that we apply a method to ITS.

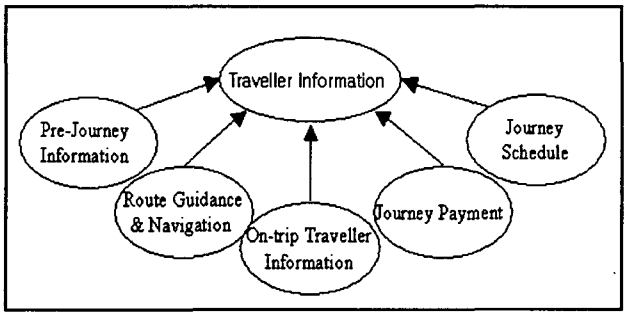

Figure 3 - This is a sample of use case composition diagram of TICS reference architecture.

The difference between actor tree and use case composition diagram is that use case composition diagram represents clearly number, name of use case by level. A use case, 'Traveler information' in the top level has to represent 5-use case and to equal the use case's name to the use case composition diagram in next level.

The *use case description* defined in this paper is text document, terse one-paragraph summary [12]. It is usually represents the main success scenario, includes the actors' name that directly communicates with each use cases.

A tor's name in use case diagram compare with actor's n:me in use case description, verify that all actor linked use c:se in use case diagram exist on use case description. As n:orhologic construction analysis of description, if the a:tor - not to be described on the description - links to use c:se in use case diagram, it is inconsistent.

Table 2 - This is sample of use case description

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

3.2 System Overview

First, rule based system exists to store consistency rule. In this system, modify or update the rule. Also there are use case diagrams at the level of abstraction and three kinds of documents: actor tree, use case composition diagram and use case description. They are compared with use case diagrams. Consistency checking in use case model is on the basis of consistency rules and compare with above 3 artifacts.

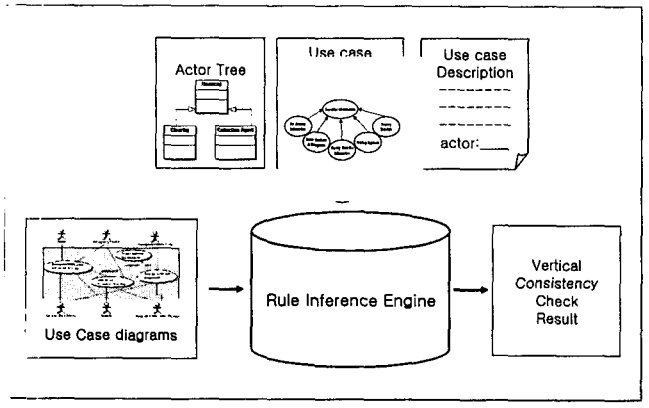

Figure 4 - System architecture of rule based consistency checking

3.3 Consistency Rules

In this section, we introduce several examples of rule for automatic consistency checking. Consistency rule shown in his paper, is very simple and only about syntax error. As use case model is representation of system requirements, is produced at early stage when develop the system architecture. Inconsistencies happened at the early stage of development system, make great influence on all following step. It may cause very expensive cost and delay and so far as software project failure. Therefore it is very important use though it's simple.

Ex) Rule1. All of actor in the top-level use case diagram is root actor.

To clarify the levels, only root actor can represent in topmost level. But line between levels of actor in the use case diagram is not clear, it just has the possibility of potential problem. Actor existed in top-level use case diagram compare with root actor of actor tree. If the actor is not a root actor, then it has a possibility of inconsistency.

Ex) Rule 2. Actor in super level exist one actor with generalization relationship in sub level.

Actor in super level represents equal or higher level than actors in sub level. Actor in sub level use case diagram inherits from the actor in super level use case diagram. So actors in sub level use case diagram are equal or represents with subordinate actor. Compare with actor tree as each actor linked subordinate use case appears as an actor of at least one linked subordinate use cases.

Ex) Rule 3. Use case in super level is mapping the child use case in sub level.

Number, name of use cases correspond to the child use cases in the use case composition diagram. When use case in super level represent in sub level, it has to represent by level clearly as the use case composition diagram. From super level to sub level a use case is refined to smaller use cases, verify the number and name of use cases. If number of use case isn't equal, or name of use case is different, then it is not consistent.

Ex) Rule 4. Actor of generalization relationship doesn't represent in the same level.

Search into actor tree. Generalization relationship means that inheritance of features and associations. If actors of generalization relationship represent in one use case diagram, it means overlap, can be inconsistent potentially

Ex) Rule 5. Actors linked use case is existed in use case description.

Actor linked to use cases exists on use case description. As morphologic construction analysis of description, if the actor not to be described on the description linked to use case in use case diagram, it is inconsistent.

Until now, we defined simple and rough consistency rule. In next section, describes the application of these rules to consistency checking

4. Simulation: Intelligent Transportation System (ITS) Architecture

ITS defined as the electronics, communications and information processing used singly or integrated into the transportation system infrastructure, and into vehicles themselves to improve the efficiency or safety of surface transportation [25]. And architecture is highest-level structural and functional organization of a system.

4.1. ITS Architecture

An ITS architecture defines how systems functionally operate and the interconnection of information exchanges that have to take place between these systems to deliver transportation services [25]. In the international ITS standards, ISO/TC204 taken from the UML, apply for the semantic definitions. In this paper, we present the process of transformation from ITS architecture to use case model according to international ITS architecture standards and consistency checking. For the ITS service, development system is Traffic Information Control System (TICS). This example refers to reference model of international ITS standards, ISO/ TC 204 [23]. Association between use cases abbreviated to decrease the complexity. So "uses" association is not represent in a case study.

The next diagram represent top-level use case diagram of TICS, ISO/TC204 the reference architecture

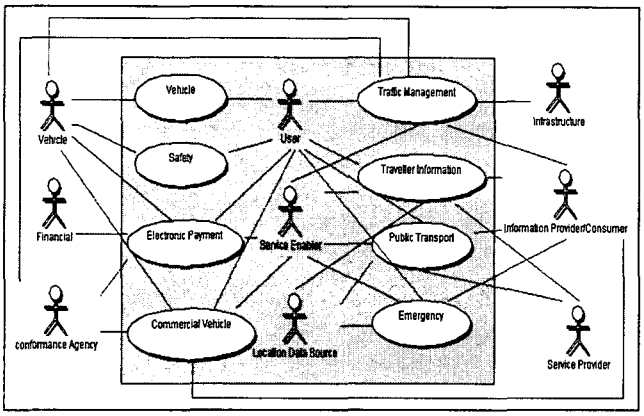

Figure 5 - Core TICS reference architecture Top-level use case diagram.

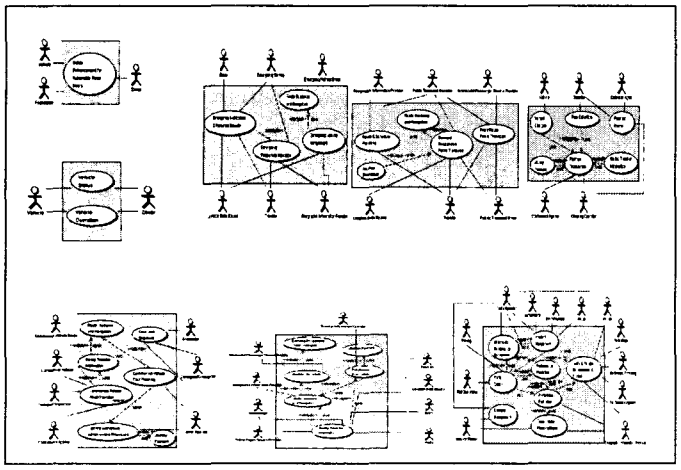

Figure 6 - Core TICS reference architecture 2nd level use case diagram.

Although just only one level is refined, the diagram came to be very complex and grew bigger. So use case model is getting bigger and more complicate with refinement. The consistency checking by human may be difficult.

4.2 Application of the Rule 1

By applying the Rule 1 to this reference architecture, we can check the consistency.

Rule1. All of actor in the top-level use case diagram is root actor.

The diagram in Fig 5 is top–level use case diagram. Actors are all of user includes vehicle, infrastructure, financial and so on. And use cases are all of service that ITS provides.

This diagram compares with actor tree as follows, the actor, 'Conformance Agency' and 'Location Data Source' is subordinate actor of 'Service Enabler'. They are not root actor. Use case linked to the actor 'Conformance Agency' and 'Location Data Source' both linked to 'Service Enabler'. Sticky applying the consistency by level, it's overlap. So it is inconsistent potentially.

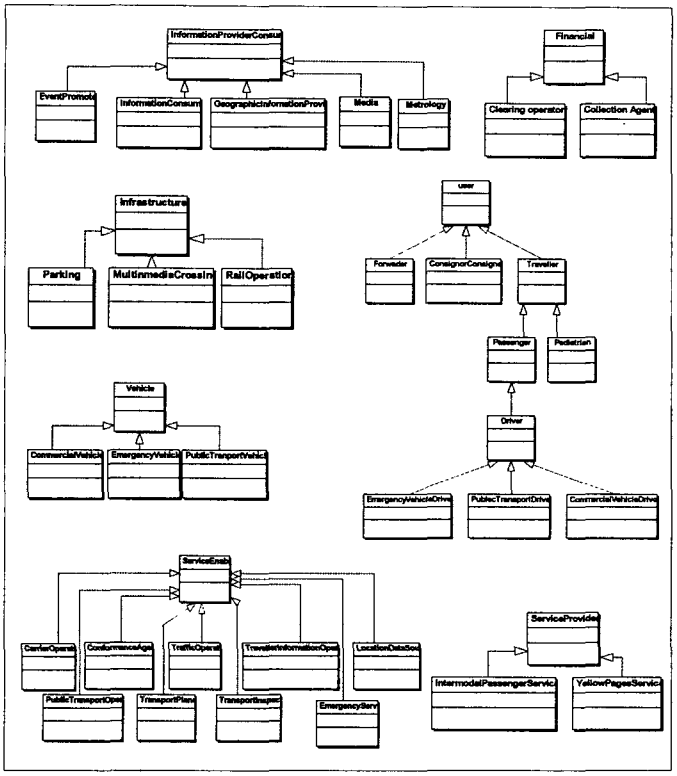

Figure 7 - Actor tree of TICS reference architecture. 'Conformance Agency' and 'Location Data Source' are subordinate actors of 'Service Enabler'.

Based on the novel method we proposed above, this diagram compares with actor tree; a result of automatically consistency checking represents as follows:

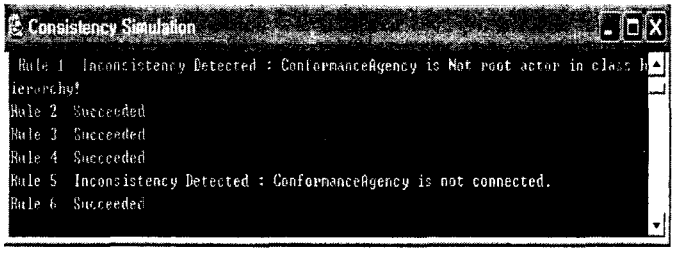

Figure 8 - Consistency checking result for Rule 1

Although our method is very simple, it is very useful. Inconsistencies are easy to happen and really happen in the reference model, as the architecture produces many diagrams and comes to complicate. Till now looking into, we can find the several inconsistencies with application of our method in refinement process from topmost level to 2nd level

5. Conclusion

In this paper, we proposed a method to automate consistency checking in use case model at the level of abstraction. In the development of large and complex system, use case model at the level of abstraction is necessary, since all services of expecting system cannot be represented in one use case model. But there are almost no methods to check consistency throughout use case models. Our rule-based solution utilizes actor tree, use case composition diagram and use case description. We also show several examples of consistency rules. It is simple, whereas very important to consistency management of software system development. To verify our approach, we applied the method to one part of ITS architecture as an example and we could find several syntax errors in reference model of international ITS standards, ISO/ TC 204.

In the future, actor tree and use case composition diagram that make out with the manual currently will be generated automatic with use case diagrams. And then it expands to consider "uses" association between use cases that we do not treat in this time. Making the range with apply field, the research needs to reinforce the rule of syntax error and expands the rule to detect semantic errors.

References

[1] Amyot, D. and Mussbacher, G. (2000). *On the extension of UML with use case maps concepts.* UML 2000, York, UK, volume 1939 of LNCS. Springer

[2] Boehm, B. (1981). *Software Engineering Economics.* Prentice-Hall, Englewood Clips, NJ

[3] Dorfman, M. (1997). *Requirements Engineering,* Software Requirements Engineering, 2 nd Edition, Thayer, R. and Dorfman, M. editors, IEEE Press, Los Alamitos, CA, 1997.

[4] Eeles, P. (2001). *Capturing Architectural Requirements*, the rational edge November 2001. rational software

[5] Engels,G., Kuster, J. and Heckel, R.(2001). "A Methodology for specifying and Analyzing Consistency of Object-Oriented Behavioral Models," *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 186-195

[6] Giandini, R., Pons, C., Pérez, G. (2002). "Use Case Refinements in the Object Oriented Software Development Process," *Proceedings of CLEI 2002, ISBN 9974-7704-1-6, Uruguay. November 2002.*

[7] Gryce, C., Finkelstein, A. and Nentwich, C. (2002). "xlinkit: Lightweight Consistency checking for the UML," *UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development, Blekinge Institute of Technology, volume Research Report 2002:06.*

[8] Hausmann, J., Heckel, R., Taentger, G. (2002). "Detection of Conflicting functional Requirements in a Use Case Driven Approach (A static analysis technique based on graph transformation) ," *Proceedings of the 24th international conference on Software,* engineering pp. 105 - 115

[9] Inverardi, P., Muccini, H. and Pelliccione, P. (2001). "Automated Check of Architectural Models Consistency using SPIN," *16th IEEE International Conference on Automated Software Engineering (ASE'01),* pp.346-349

[10]Inverardi, P., Muccini, H. and Pelliccione, P.(2001). "Checking consistency between architectural models using SPIN", *Proceeding of The First Int. Workshop from Software Requirements to Architectures*

[11]Kielland, T., Børretzen, J. (2001). "UML consistency checking," SIF8094, Institutt for datateknikk og informasjonsvitenskap

[12]Larman, C. (2001). *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. (2001). "Rule-based detection of inconsistency in UML models," *Workshop on Consistency Problems in UML-Based Software Development, At the Fifth International Conference on the Unified Modeling Language,* pp. 106--123.

[14]Loucopoulos, P., Karakostas, V. (1995). *System Requirement Engineering*, McGraw-Hill, pp.6-10.

[15]Nuseibeh, B., Easterbrook, S. and Russo, A. (2001). "Making inconsistency respectable in software development," *The Journal of Systems and Software*, 58(2):171–180,

[16]Pérez ,G., Giandini, R. and Pons, C. (2002). "Model Refinements in the object oriented software development process," *Argentine Symposium on Software Engineering*, 1666-1087 pp.61 - 73

[17]Quatrani, T. (2001). "Introduction to the Unified Modeling Language," The Rational Developer network, Ratinal software.

[18]Rasch, H. and Wehrheim, H. (2002). "Consistency between UML Classes and Associated State Machines," *UML 2002 - Workshop on Consistency Problems in UML-based Software Development*, volume 06, pp 46-60.

[19]Robinson, W., Pawlowski, S. (1999) "Managing Requirements Inconsistency with Development Goal Monitors," *IEEE Transactions on Software Engineering*, 25(6) pp.816--835,

[20] Spanoudakis, G. and Zisman, A. (2001). "Inconsistency Management in Software Engineering: Survey and Open Research Issues," handbook of software engineering and knowledge engineering, World Scientific Publishing Co. Pte. Ltd.

[21]Thayer, R., Dorfman, M. (1997). *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 (1999) Transport information and control systems-Reference model architecture(s) for the TICS sector – part 2: Core TICS reference architecture, ISO/DTR 14813-2

[24]OMG: *Unified Modeling Language Specification,* March 2003, Version 1.5, formal/03-03-01

[25]ITS-Intelligent Transportation Systems, http://www.its.dot.gov