Aspect-Oriented Development Method for Non-Functional Characteristics of Cyber Physical Systems Based on MDA Approach

Lichen Zhang
Guangdong University of Technology, Guangzhou, China
Email: zhanglichen1962@163.com

***Abstract*—Cyber physical systems have many non-functional requirements, which always crosscut the whole system modules. That may cause the code tangle and scatter, make the systems hard to design, reuse and maintain, and affect performance of systems badly. AOP is a new software development paradigm, which could attain a higher level of separation of concerns in both functional and non-functional matters by introducing aspect, for the implementation of crosscutting concerns. Different aspects can be designed separately, and woven into systems. In this paper, we propose an aspect-oriented MDA approach for non-functional properties to develop cyber physical systems. An aspect-oriented UML profile is built to develop cyber physical systems. Aspect-oriented UML models are designed as Platform Independent Models (PIM) for target-platform implementation, which deal with non-functional properties. OCL formal language is used to restrict the model in every stages of MDA, and the real-time extension of OCL formal language is made to describe the timing constraints of cyber physical systems. Finally, the model- based development and aspect-oriented approach, the formal methods and the cyber physical system are integrated effectively. A case study illustrates the aspect oriented MDA development of cyber physical systems.**

***Index Terms*—Non-Functional Properties, Aspect-Oriented, MDA**

I. INTRODUCTION

Cyber-physical systems (CPSs)[1] are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core. Recent years have witnessed the growing applications of CPSs in numerous critical domains including healthcare, transportation, process control, factory automation, smart building and spaces etc. By seamlessly integrating sensing, networking, and computation components with the control of physical processes, CPSs are expected to transform how we interact with and manipulate the physical world.

Model Driven Architecture (MDA)[2] is based on a series of industry-standard software development frameworks, model drives the software development process, and using support tool model can to achieve automatic conversion among the models, between the model and the code. Its core idea is to establish a Platform Independent Model (PIM) with complete description of system requirements and specific platform implementation technology, through a series of model transformation rule set, the platform independent models to be able to transfer to complete presentation system requirements, and specific implementation techniques related to platform specific model (PSM), finally, using MDA tools will be making platform specific model automatically transferred to code.

Aspect-oriented software development methods[3] make up object-oriented software development methods in system development needs of non-functional characteristics of the existing limitations question problem. Use separate technology of concerns separates all the crosscutting concerns of the system, and then analyzed, designed, modeled for each cross-cutting concerns, to address crosscutting concerns in object-oriented software development, the code tangling and scattering problems, enhancing the system's modular degree, lowering coupling between modules.

In this paper, we propose an aspect-oriented MDA for non-functional properties to develop cyber physical systems.

II. NON-FUNCTIONAL REQUIREMENTS OF CYBER PHYSICAL SYSTEMS

Non-functional requirements[4] address important issues of quality and restrictions for cyber physical systems, although some of their particular characteristics make their specification and analysis difficult: firstly, non-functional requirements can be subjective, since they can be interpreted and evaluated differently by different people; secondly, Non-functional requirements can be relative, since their importance and description may vary depending on the particular domain being considered; thirdly, non-functional requirements can be interacting, since the satisfaction of a particular non-functional requirement can hurt or help the achievement of other non-functional requirement.

A set of ISO/IEC standards are related to software quality, being standards number 9126 [5], 14598-1 and 14598-4 the more relevant ones [6]. The main idea behind these standards is the definition of a quality model and its use as a framework for software evaluation. A quality model is defined by means of general characteristics of software, which are further defined into sub-characteristics in a multilevel hierarchy; at the

doi:10.4304/jsw.7.3.608-619

bottom of the hierarchy appear measurable software attributes. Quality requirements may be defined as restrictions over the quality model. The ISO/IEC 9126 standard fixes 6 top level characteristics: functionality, reliability, usability, efficiency, maintainability and portability. Furthermore, an informative annex of this standard provides an illustrative quality model that refines the characteristics as shown in Fig 1 [5].

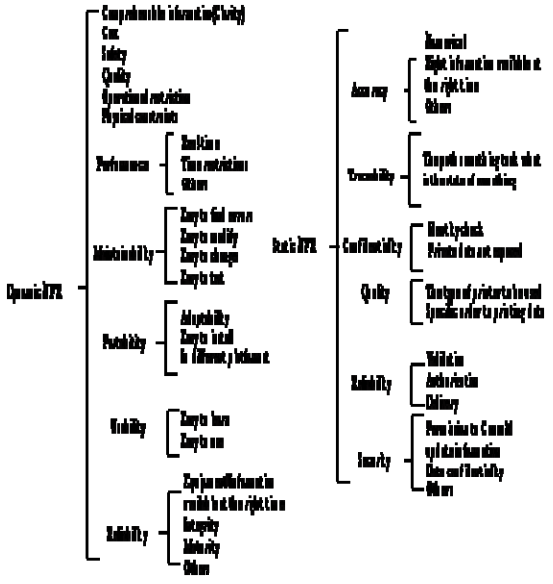

Figure 1. ISO 9126 Standard

In order to evaluate these attributes, a metric must be selected and rating levels have to be defined dividing the scale of measurement into ranges corresponding to degrees of satisfaction with respect to the attribute. The rating levels must be defined for each specific evaluation depending on the quality requirements. Finally, a set of assessment criteria combining the measures of attributes are necessary to obtain the rating of the intermediate and top characteristics and, finally, the quality of the product.

According to their relationships with the primary functionality of system, Non-functional requirements of system can be classified as follows (see Fig. 2)[5].

Figure 2. Non-functional requirements classification for system

Application aspect can change the internal behavior of. They are additional operational design requirements which a system should be configured to support specific target platforms. Examples of such non-functional requirements are memory optimization, error handling, fault tolerance, real-time property, and real-time policy. Since optimizing memory usage is one of the key issues in real-time system and it crosscuts the structure of system, Memory optimization is viewed as an application aspect of the system. Error handling, entangled in the entire system, is encapsulated and represented by an error handling aspect. Fault tolerance is another application aspect that influences behavior and structure of a system. Additionally, real-time properties and policies are viewed as application aspects as they influence the overall structural behavior of the system. Depending on the requirements of a system, real-time properties and policies could be further refined. The common characteristic of those aspects is that they extend the primary functionality of system.

Maintenance aspect is characteristics that relate to the maintainability of system. Examples of such non-functional requirements are logging, tracing and coding rule enforcements aspect. Those non-functional requirements do not carry any operational purposes and they could consume considerable computing resources and major development efforts. The common characteristic of those non-functional requirements is that they are related to the human factors in software engineering.

Composition aspects refer to non-functional requirements that need to be considered when integrating components into system. Examples of such non-functional requirements are resource demand, temporal constraints, portability, and flexibility. Each component should have declared resource demands and information of its temporal behavior in its resource demand and temporal constraints respectively. Additionally, the information, such as real-time operating system supported, and other hardware related information, is contained in the portability. Possibilities of extending the component are contained in the flexibility.

The first task is to combine the non-functional services using patterns he/she identified during his/her expertise. The new artifacts are UML architectural models of the container and a UML framework for using the non-functional services in UML models[7]. The UML framework can be a profile for using the non-functional services, it uses UML extension mechanisms such as stereotypes, tagged values, and constraints. The second task is to help the application developer in transparently integrating the non-functional requirements to implementation components A UML framework can provide a profile to architects and designers for modeling non-functional dependencies. One or more non-functional requirements can be attached to the operations (i.e., pre-/post-conditions) and other constraints in OCL[8], the attributes, and other model elements of a software component. Non-functional requirements can also be attached to the links between the software components in order to configure the system for instance. Attaching constraints and tagged values is the simplest way to add non-functional requirements to a model element.

A constraint consists in specifying more semantics as an expression in a designated constraint language. Constraints are gaining more and more importance in UML. A tagged value consists of a name and its associated value.

Dependability is that property of a system that justifies placing one's reliance on it. The dependability of a system is the collective term that describes the availability performance of a system and its influencing factors: reliability, safety, maintainability and maintenance support performance. These non-functional properties are highly important for cyber physical systems as they are designed to operate in environments where failure to provide functionality or service can have enormous cost both from financial, influential or physical aspects. Therefore it is essential that these properties are calculated as precisely as possible during the design and operation of such systems. Reliability is the ability of a system or component to provide its required functionality or services under given conditions for a specified period of time. Availability is the ratio of total time that a system or a component is functional during a specified period and the length of the period. Maintainability can be specified as the probability that a component or system will be restored to a given condition within a period of time. Safety is described as the absence of serious consequences on the user or environment in case of failure. Safety can be defined as "a property of a system that it will not endanger human life or the environment" .A system is safety-critical if safety cannot be ensured when it fails to provide correct service. Integrity can be specified as the absence of improper alterations on the target system or component. Survivability can be defined as the ability of the system to remain functional after a natural or man-made disturbance. The threats to dependability are faults, errors and failures. There is a relationship between these threats: A fault is a defect in the system which, when activated, leads to an error. An error is an incorrect system state that may affect the external behaviour, thereby causing a failure. A failure occurs when the delivered service deviates from what is considered correct. There exist four general means to achieve dependability: fault prevention, fault tolerance, Fault removal, and fault forecasting. Fault prevention deals with the objective of avoiding to introduce faults during the software development process. There exist four general means to achieve dependability: fault prevention, fault tolerance, fault removal, and fault forecasting. Fault prevention can be considered as an inherent part of it. Fault removal deals with uncovering faults that have happened at any phase of the development process. Fault forecasting is aimed at evaluating the behaviour of the system under the occurrence of faults such that it can be concluded which ones would lead to system failure. Fault tolerance techniques are the means to allow a system to provide correct service even when faults occur. Such techniques use diverse forms of redundancy to detect and recover from faults. The most common approaches use either hardware redundancy, software redundancy, time redundancy or information redundancy to identify erroneous conditions. The subsequent recovery process relies on the remaining fault-free parts of the system to correct the errors and/or prevent them from reappearing.

Timeliness requirements [9] apply to computations in which correctness depends not only on the results produced but also the time at which they become available. Soft real-time requirements are general performance goals, typically expressed via some measure of average response time. Such goals have a probabilistic or statistical flavour which takes them outside the scope of this study. More tangible are hard real-time requirements in which particular events must occur at, or before, certain times. A periodic requirement states that some action must be performed at regular intervals, while a sporadic requirement states that some action must be performed immediately following an external "triggering" event. In cyber physical systems components do not only have to perform operations correctly, but also have to meet certain timing requirements. General purpose components like graphical user interface frameworks are often not design with a real-time scenario in mind and thus real-time programmers are many times forced to build large parts of their applications from scratch. Building components suitable for real-time applications is a difficult task, as besides the functional requirements attention has also to be paid to the non-functional timing requirements. This additional complexities makes building real-time components more expensive and error prone than general purpose components. Real-time and fault tolerance constraints can impose conflicting requirements on a distributed system. Real-time operation requires an application to be predictable, to have bounded request processing times, and to meet specified task deadlines. This predictability is often the most important characteristic of cyber physical systems. In contrast, fault tolerant operation requires that an application continue to function, even in the presence of unanticipated, potentially time-consuming events such as faults and fault recovery. Faults are often viewed as asynchronous unpredictable events that can upset a cyber physical system's scheduled operation. Sustained operation with consistency of application data in the face of faults is often the single most important characteristic of fault-tolerant systems. Thus, there is a fundamental conflict between the philosophies underlying the two system properties of real time and fault tolerance. While real-time performance requires a priori knowledge of the system's temporal operation, fault tolerance is built on the principle that faults can and do occur unexpectedly, and that faults must be handled through some recovery mechanism whose processing time is uncertain. When both real-time and fault tolerant operation are required in the same system today, trade-offs are made at design time, not at run-time.

III. MODEL-DRIVEN ARCHITECTURE

MDA is a framework proposed by OMG for the software development, driven by models at different abstraction levels. MDA relies on the separation of the

business logic of a system from its implementation. To achieve this, MDA defines two types of models: the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM). PIM captures system behavior and functionality, while PSM captures information about details of system implementation. The MDA development process primarily involves three steps. First the PIM is developed. The objective is to capture the high level functional requirements of the application. In the second step, transformation rules are used to transform the PIM into one or more Platform-Specific Models. A PSM is customized to describe the system in terms of the particular implementation platform. The third step involves the conversion of the PSM into application code. Typically, steps two and three are automated by the use of automated tools. It is the first step in the process that involves creativity and manual work. In general, MDA is a useful approach towards reasoning about the impact of system on the behavior of software systems[10].

MDA attempts to raise the level of abstraction by which software and systems engineers carry out their tasks. This is done by emphasis the use of models (i.e., abstractions) of the artifacts that are developed during the engineering process. Models are representations of phenomena of interest, and in general are usually easier to modify, update, and manipulate than the artifact or artifacts that are being represented. Models are expressed using a suitable modeling language; UML is a widely used standard in MDA. MDA is not a development method or process; it can be implemented in a number of ways, e.g., via Extreme Programming, the Rational Unified Process, the B-Method, or a refinement calculus. The key element in MDA is the construction and transformation of models that are fit for the purposes of the development project. The languages and processes used in construction and transformation will vary from project to project.

The specification of the Object Constraint Language (OCL) is a part of the UML specification, and it is not intended to replace existing formal languages, but to supplement the need to describe the additional constraints about the objects that cannot be easily represented in graphical diagrams, like the interactions between the components and the constraints between the components' communication. In object-oriented modeling, a graphical model, such as a class diagram, is not enough for a precise unambiguous specification. OCL is designed to solve this problem. It facilitates the specification of model properties in a formal yet comprehensive way. By combining the power of the straightforward, graphical UML modeling and the textual, accurate OCL constraints, these kinds of information can be specified in this formal way.

IV. Related Works

The SAE Architecture Analysis and Design Language [11] is a design-by-committee standard promoted to help the space and avionics domain. It now extends to a much broader audience, and this language is used in many domains related to Cyber-Physical Systems. AADL is an ADL promoted in the context of Model-Driven Engineering which has now gained a significant momentum in the industry. Models are a valuable asset that should be used and preserved down to the construction of the final system; modeling time and effort should be reduced to focus directly on the system and its realization. Yet, validation and verification may require many different analysis models, involving a strong theoretical background to be mastered. The SAE AADL has been defined to match the concepts understood by any engineer (interface, software or hardware components, packages, generics). From these concepts, typical behavior elements (scheduling and dispatch, communication mechanisms) have been added using both formal and informal description, always bound to theoretical frameworks for V&V. In parallel, the AADL allows one to attach user-defined properties or languages for specific analysis. This enables the application of many different techniques for the analysis of AADL models, among which schedulability, safety, security, fault-propagation, model-checking, resource dimensioning, etc.; but also code generation.

Model-based design is a powerful design technique for cyber-physical systems, but too often literature assumes knowledge of a methodology without reference to an explicit design process, instead focusing on isolated steps such as simulation, software synthesis, or verification. Jeff C. Jensen, Danica H. Chang and Edward A. Lee combine these steps into an explicit and holistic methodology for model-based design of cyber-physical systems from abstraction to architecture, and from concept to realization. They decompose model-based design into ten fundamental steps, describe and evaluate an iterative design methodology, and evaluate this methodology in the development of a cyber-physical system[12].

Cyber-physical systems (CPSes) couple their cyber and physical parts to provide mission-critical services, including automated pervasive health care, smart and secure electricity grid, green cloud computing, and surveillance with unmanned aerial vehicles (UAV). CPSes can use the information available from the physical environment to provide such ubiquitous, energy efficient and low cost functionalities. Their operation needs to ensure three key properties, collectively referred to as S3: i) safety, avoidance of hazards, ii) security, assurance of integrity, authenticity and confidentiality of information, and iii) sustainability, maintenance of long term operation of CPS using green sources of energy. Ensuring S3 properties in a CPS is a challenging task given the spatio-temporal dynamics of the underlying physical environment. In this paper, the formal underpinnings of recent CPS S3 solutions are aligned together in a theoretical framework for cyber-physical interactions, which enables CPS researchers to systematically design a solution for ensuring safety, security, or sustainability. The general applicability of this framework is demonstrated with various example solutions for safety, security, and sustainability in diverse

CPS domains. Further, insights are provided on some of the open research problems for ensuring S3 in CPSes.[13].

Solberg Arnor et al. presents an MDD framework that uses aspect oriented software development techniques to facilitate separation of concerns. The proposed framework can simplify both the model development task and the task of specifying transformations. The conceptual model of the framework is presented and illustrated using distributed transactions at the PIM and PSM levels[14].

Aniruddha Gokhale and Jeff Gray illustrate the tangling of concerns in the deployment and configuration of distributed real-time and embedded systems. Model driven generative technologies help address these concerns by alleviating several accidental complexities arising in the modeling process. Yet, MDD tools alone are not sufficient since they cannot scale in some cases. Additionally, some of the modeling activities can become tedious and repetitive while addressing crosscutting concerns. Aspect weaving at the modeling level resolves these problems. Their paper describes ongoing work along with a short case study on integrating the C-SAW aspect weaving tool with the CoSMIC model driven development tool suite[15].

Sven Burmester, Holger Giese, and Wilhelm Schafer propose one approach that consists of components and Real-Time Statecharts, and permits to specify complex real-time systems following UML notations and the MDA approach at the PIM level. This platform independent description can then be mapped automatically to a platform specific model, provided that a target platform description in form of annotations describing real physical behavior (WCETs) are given. The PSM describes real-time threads, which are of general nature and not bound to a specific programming language or RTOS environment. Thus, an implementation can be realized in any programming language that provides real-time priority scheduling. Different analysis methods are applied on the different levels to achieve correct models[16].

MARCO AURÉLIO WEHRMEISTER proposes an automated integration of distributed embedded real-time systems design phases focusing on automation systems. The proposed approach uses Model- Driven Engineering (MDE) techniques together with Aspect-Oriented Design (AOD) and previously developed (or third party) hardware and software platforms to design the components of distributed embedded real-time systems. Additionally, AOD concepts allow a separate handling of requirement with distinct natures (i.e. functional and non-functional requirements), improving the produced artifacts modularization (e.g. specification model, source code, etc.). In addition, this thesis proposes a code generation tool, which supports an automatic transition from the initial specification phases to the following implementation phases. This tool uses a set of mapping rules, describing how elements at higher abstraction levels are mapped (or transformed) into lower abstraction level elements. In other words, such mapping rules allow an automatic transformation of the initial specification, which is closer to the application domain, in source code for software and hardware components that can be compiled or synthesized by other tools, obtaining the realization/ implementation of the distributed embedded real-time system[17].

V. Applying AOP and MDA to Non-Functional Requirements

The use of models based development to assist in the development of software for general purpose computes is not a new research topic. In addition, there are already some works on the “model-driven engineering” topic proposing solutions to some problems. However, the employment of MDE in the design of Cyber physical systems can be considered a recent research topic, which still has several gaps to be fulfilled. MDA distinguishes several types of models. Platform In-dependent Models (PIM) specify the software system in an independent way from the technology platform chosen to implement it. Platform Specific Models (PSM) refine the PIM to specificities of the implementation platform. That is, two different implementations of the same system would share the same PIM but have two different PSMs, each one adapted to the technological capabilities of each platform. A third type of model, Computation Independent Models (CIM, a kind of business model), exists, but in this paper, we will focus on the transformation from PIM to PSM. To reflect the NFRs properly in the implementation of a software system, it is necessary to specify them at design level. Modeling helps the developers to work in a higher level of abstraction by hiding the details. The model representation of a system provides a high-level view, where the developers can focus on different aspects of a software system. UML can be used to represent FRs and NFRs of the system because UML has emerged as the industry standard for software modeling notations. Various diagrams are available in UML models, and using several types of diagrams, several views of a system can be captured. Another major advantage of using UML is that many UML tools are available in the market.

Fig.3 shows the UML extensions for non-functional requirements (NFR) modeling. The NFR Modeling package (stereotyped as profile) defines how the elements of the domain model extend meta-classes of the UML meta-model. A UML framework can provide a profile to architects and designers for modeling non-functional dependencies. One or more non-functional requirements can be attached to the operations (i.e., pre-/post-conditions) and other constraints in OCL, the attributes, and other model elements of a software component. Non-functional requirements can also be attached to the links between the software components in order to configure the system for instance. The main UML extension mechanisms are constraints, tagged values, and stereotypes. These extensions are used for

documentation purposes and for directing code and configuration descriptors generation. Attaching constraints and tagged values is the simplest way to add non-functional requirements to a model element. A constraint consists in specifying more semantics as an expression in a designated constraint language. Constraints are gaining more and more importance in UML. A tagged value consists of a name and its associated value. By definition, constraints and tagged values are simple and very extensive-since there are very few limitations on their usage. The resulting contract between the service and the software component can be too fine-grained and spread over multiple model elements. Therefore, they may be complex to use for configuring non-functional services.

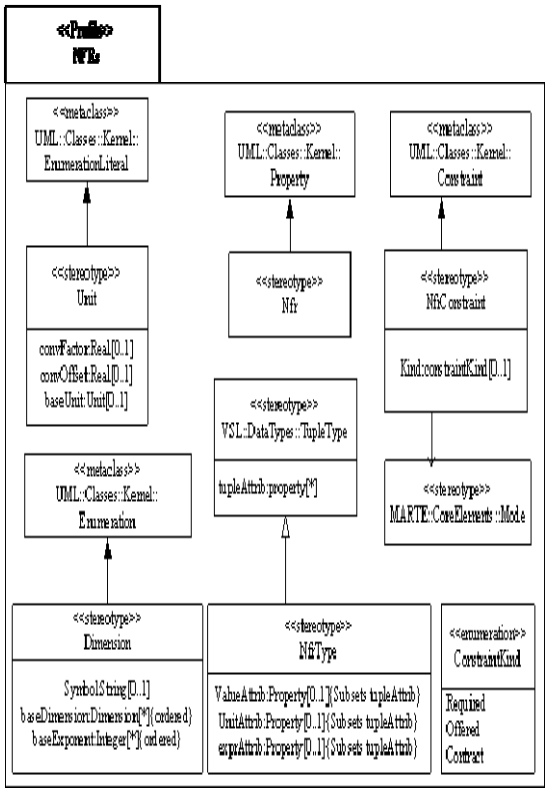

Figure 3. UML profile diagram for NFRs modeling

Object-Oriented Programming (OOP) has been the dominant programming methodology that is being used in all kinds of software development today. The main focus of OOP is to find a modular solution for a problem by breaking down the system into a collection of classes that encapsulates state and behavior. However, In Object-Oriented Programming, crosscutting concerns are elements of software that can not be cleanly captured in a method or class. Accordingly, crosscutting concerns has to be scattered across many classes and methods. OOP fails to provide a robust and extensible solution to handle these crosscutting concerns. AOP is a new modularity technique that aims to cleanly separate the implementation of crosscutting concerns. It builds on Object-Orientation, and addresses some of the points that are not addressed by OO. AOP provides mechanisms for decomposing a problem into functional components and aspectual components called aspects. An aspect is a modular unit of crosscutting the functional components, which is designed to encapsulate state and behavior that affect multiple classes into reusable modules. Distribution, logging, fault tolerance, real-time and synchronization are examples of aspects. The AOP approach proposes a solution to the crosscutting concerns problem by encapsulating these into an aspect, and uses the weaving mechanism to combine them with the main components of the software system and produces the final system. We think that the phenomenon of handling multiple orthogonal design requirements is in the category of crosscutting concerns, which are well addressed by aspect oriented techniques. Hence, we believe that system architecture is one of the ideal places where we can apply aspect oriented programming (AOP) methods to obtain a modularity level that is unattainable via traditional programming techniques. To follow that theoretical conjecture, it is necessary to identify and to analyze these crosscutting phenomena in existing system implementations. Furthermore, by using aspect oriented languages, we should be able to resolve the concern crosscutting and to yield a system architecture that is more logically coherent. It is then possible to quantify and to closely approximate the benefit of applying AOP to the system architecture.

MDA is a framework proposed by OMG for the software development, driven by models at different abstraction levels. MDA relies on the separation of the business logic of a system from its implementation. To achieve this, MDA defines two types of models: the Platform-Independent Model (PIM) and the Platform-Specific Model (PSM). PIM captures system behavior and functionality, while PSM captures information about details of system implementation. The MDA development process primarily involves three steps. First the PIM is developed. The objective is to capture the high level functional requirements of the application. In the second step, transformation rules are used to transform the PIM into one or more Platform-Specific Models. A PSM is customized to describe the system in terms of the particular implementation platform. The third step involves the conversion of the PSM into application code. Typically, steps two and three are automated by the use of automated tools. It is the first step in the process that involves creativity and manual work. In general, MDA is a useful approach towards reasoning about the impact of system on the behavior of software systems.

MDD attempts to raise the level of abstraction by which software and systems engineers carry out their tasks. This is done by emphasis the use of models (i.e., abstractions) of the artifacts that are developed during the engineering process. Models are representations of phenomena of interest, and in general are usually easier to modify, update, and manipulate than the artifact or artifacts that are being represented. Models are expressed using a suitable modeling language; UML is a widely used standard in MDD. MDD is not a development method or process; it can be implemented in a number of

ways, e.g., via Extreme Programming, the Rational Unified Process, the B-Method, or a refinement calculus. The key element in MDD is the construction and transformation of models that are fit for the purposes of the development project. The languages and processes used in construction and transformation will vary from project to project.

The MDA guide is vague in its definition of MDA and the notion of refinement. The guide defines MDA in terms of PIM, PSM, and additional models such as domain models. Refinement is defined informally as a process of transforming MDA models (e.g., PIM to PSM, PSM to code, PIM to PIM). The MDA guide distinguishes PIM and PSM as models at different levels of abstraction, e.g., a PIM is at a higher level of abstraction than a PSM. However, years of research on refinement calculi and programming methodology, particularly on wide-spectrum languages, suggest that distinctions such as this are not helpful: it is more productive to think in terms of specifications that have different properties. For example, in predicative programming, programs are a special kind of specification. They are implementable and immediately executable on a machine. Similarly, in refinement calculus, specifications are a special kind of program. they are not always executable, but one can test for feasibility, and they are written in a unified language. To formally define refinement in MDA, there are four alternatives. One could translate the core languages used in MDA i.e., UML, or a subset of UML into a formal language such as Z, B, LOTOS or specification statements. Work has been carried out on expressing such translations, but it all suffers from limitations, e.g., incompleteness, difficulties in achieving consistency, etc. A second alternative is to promote a formal definition of refinement e.g., weakest preconditions and express it in MDA terms, e.g., in UML. It is debatable whether UML is well suited to expressing formal definitions of refinement.

The specification of the Object Constraint Language (OCL) is a part of the UML specification, and it is not intended to replace existing formal languages, but to supplement the need to describe the additional constraints about the objects that cannot be easily represented in graphical diagrams, like the interactions between the components and the constraints between the components' communication. In object-oriented modeling, a graphical model, such as a class diagram, is not enough for a precise unambiguous specification. OCL is designed to solve this problem. It facilitates the specification of model properties in a formal yet comprehensive way. By combining the power of the straightforward, graphical UML modeling and the textual, accurate OCL constraints, these kinds of information can be specified in this formal way.

OCL has the characteristics of an expression language, a modeling language and a formal language. An OCL expression is guaranteed to be without side effects since it is an expression language, and thus cannot change anything in the model, although an OCL expression can be used to specify the state changes of the system. OCL is not a programming language, but a modeling language. So it is impossible to write program logic or flow-control in OCL. All implementation issues are likewise out of the scope of OCL. OCL is also a formal language where all constructs have a formally defined meaning; in other words, it is unambiguous. Furthermore, OCL is strongly typed.

The main idea behind OCL is "Design By Contract". By applying this, the responsibility of the parties is made unambiguous and can be formally described. An OCL constraint consists of the precondition, the post-condition and the invariant. The contract is a way of establishing that does what by stating, first, what must be true for the caller (client) to request a service from the callee (server) (precondition), and, what must be true when the callee finishes providing the service (post-condition). The invariant must be true when a routine is called and when it terminates, but not necessarily when it is executing. By the principle of "Design By Contract", and specifying these three constraints, the services provided by the server are exposed, but not the details of the implementation of the services.

On the other hand, the callee will know when exactly a service can be provided (available), and the caller will know when exactly it can request the service. In case of exceptions, it is easy to find out who caused the exception: if the precondition is false, the caller broke the contract; if the post-condition is false, the callee broke the contract; if the invariant is false, the callee class broke the contract.

Since OCL is a textual extension of the graphical UML modeling language, an OCL specification is always unambiguous and precise. It also provides better documentation to the visual models. It can be used during the modeling and specification. Since OCL is an expression language, it can be checked without an executable system. All these features turn out to be useful in representing non-functional properties, which can be represented by the combination of precondition, post-condition and invariant in OCL. The Non-functional attributes are represented by the member variables of the class, and the Non-functional actions are represented by the methods. They are checked at run time, before and after the calls so that the change of the Non-functional parameters of the system is monitored in a timely basis.

The precondition has to be satisfied before the method can be called, and the post-condition has to be satisfied at the time the method returns. It is easy to find out which step causes exceptions if any. The methods are called in a loop-like fashion, so, whenever a change of the Non-functional parameter is observed, the corresponding methods are called and the changes are made accordingly and the necessary notification is done at the same time. The Non-functional specification is integrated in the overall system design in this fashion. In this way, the satisfaction of the Non-functional requirements is guaranteed and the change of the Non-functional properties is under observation and control, as well.

Although non-functional properties and associated metrics have been widely used in networking, there is no standard vocabulary for discussing the Non-functional as it relates to the distributed computing and component-based solutions, especially when the Non-functional properties are applied on variant platforms and when the different aspects of the Non-functional interact with each other. A standard vocabulary is the first step toward progressing Model Driven Architecture that includes Non-functional parameterization and/or Non-functional contracts. MDA provides an open, vendor-neutral environment for the integration of different distributed application software. MDA aims to separate the business or application logic from the underlying platform technology. Its standards are made up of the UML, Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI), and Common Warehouse Meta-model. Platform-independent applications built using MDA and the associated standards can be realized on a range of platforms.

The MDA design initiative assists during the interaction between the different platforms and different system. System environments started out providing the interoperability using the architectures that are standard, proprietary, or somewhere in the middle. Progressively, more and more services and more powerful system have been added to the overall architecture, thus, it is more difficult to ensure the interoperability of these system. To efficiently solve this problem, MDA is designed by applying the component and modeling technology and putting the whole picture together.

The distributed systems software development process based on aspect-oriented system is divided in five phases. Fig. 4 depicts the whole software development process[18].

The first phase is a profound analysis of the requirements. The phase includes three steps:

Step one handles the non-functional requirements and then identifies which of those are crosscutting.

Step two performs a traditional specification of functional requirements, in this case, using an UML-like approach where the use case model is the main specification technique.

Step three starts by composing functional requirements with aspects; then it identifies and resolves conflicts that may arise from the composition process.

The concepts of overlapping, overriding and wrapping can be adopted to define the composition part of the model. Overlapping indicates the requirements of the aspect modifies the functional requirements they transverse. In this case, the aspect requirements may be required before the functional ones, or they may be required after them. Overriding indicates the requirements of the aspect superpose the functional requirements they transverse. In this case, the behavior described by the aspect requirements substitutes the functional requirements behavior. Wrapping indicates the requirements of the aspect encapsulate the functional requirements they transverse. In this case, the behavior described by the functional requirements is wrapped by the behavior described by the aspect requirements[3].

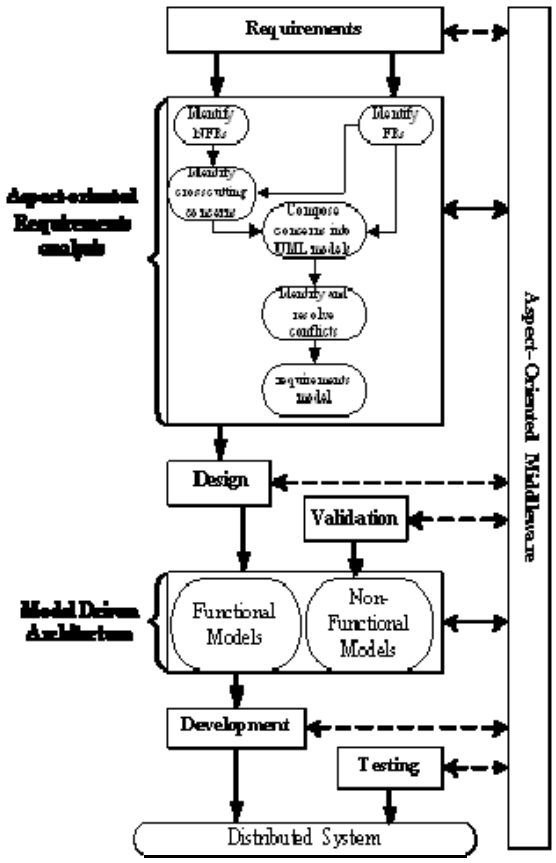

Figure 4. Distributed systems software process based on aspect-oriented system

In the design phase, the distributed system will be designed considering both the requirements and the constraints posed by the system and system. Using the MDA approach to produce the platform specific models includes five steps (see Fig. 5):

Step one: Create the PIM for the distributed system.

Step two: Select the target system and create the generic system aspects.

Step three: Transform PIM to enhanced PIM using the application converter.

Step four: Transform the generic aspects to enhanced aspects using the aspect converter.

Step five: Weave the enhanced aspects into the enhanced PIM to produce the PSM.

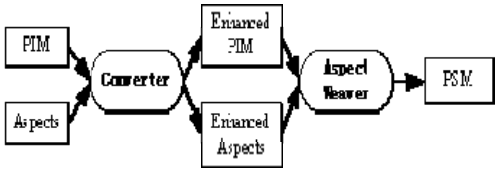

Figure 5. Process view of the PSM generation

The validation phase is in charge of validating the application design against both functional and non-functional models. Also in this phase, the system

characteristics have to be considered since they can affect the application validation. Model-based analysis techniques can be used for validation purposes . Because it provides a way for the design-time analysis of distributed systems enabling rapid evaluation of design alternatives with respect to given performance measures before committing to a specific platform.

In the development phase, the source code of classes and aspect is generated. And the distributed system is built on top of the aspect-oriented system platform.

Since aspect may affect the behavior of one or more classes through advice and introduction. Traditional testing techniques, such as unit testing, integration testing, are not applicable to test aspect in the testing phase. Some aspect-oriented testing approaches , such as data-flow-based unit testing, state-based testing approach, and model-based testing approach can be used in this phase.

To address the system development, principled methods are needed to specify, develop, compose, integrate, and validate the application and system software used by cyber physical systems. These methods must enforce the physical constraints cyber physical systems, as well as satisfy the system's stringent functional and non-functional requirements. Achieving these goals requires a set of integrated Model Driven System (MDM) tools that allow developers to specify application and system requirements at higher levels of abstraction than that provided by low-level mechanisms, such as conventional general-purpose programming languages, operating systems, and system platforms. Different functional and systemic properties of cyber physical systems via separate system and platform-independent models are applied. Domain-specific aspect model weavers can integrate these different modeling aspects into composite models that can be further refined by incorporating system and platform-specific properties. Different but interdependent characteristics and requirements cyber physical system behavior (such as scalability, predictability, safety, and security) are specified via models. Model interpreters translate the information specified by models into the input format expected by model checking and analysis tools. These tools can check whether the requested behavior and properties are feasible given the specified application and resource constraints. Tool-specific model analyzers can also analyze the models and predict expected end-to-end QoS of the constrained models. Platform-specific code and metadata that is customized for a particular QoS-enabled component system and DRE application properties, such as end-to-end timing deadlines, recovery strategies to handle various run-time failures in real-time, and authentication and authorization strategies are modeled at a higher level of abstraction. System and applications by assembling and deploying the selected components end-to-end using the configuration metadata are synthesized by MDM tools. In the case of legacy components that were developed without consideration of QoS, the provisioning process may involve invasive changes to existing components to provide the hooks that will adapt to the metadata. The changes can be implemented in a relatively unobtrusive manner using program transformation systems.

VI. Case Study: Intelligent Transportation Systems

Intelligent Transportation systems(ITS)[19] – automotive, aviation, and rail – involve interactions between software controllers, communication networks, and physical devices. These systems are among the most complex cyber physical systems being designed by humans, but added time and cost constraints make their development a significant technical challenge. MDA approach can be used to improve support for design, testing, and code generation. MDA approach is increasingly being recognized as being essential in saving time and money. Transportation systems consist of embedded control systems inside the vehicle and the surrounding infrastructure, as well as, the interaction between vehicles and between vehicle and the infrastructure. The dynamics of these interactions increases complexity and poses new challenges for analysis and design of such systems.

In real-time systems such as ITS, the passage of time becomes a central feature — in fact, it is this key constraint that distinguishes these systems from distributed computing in general. Time is central to predicting, measuring, and controlling properties of the physical world. The modeling process of non functional requirements time of ITS by aspect–oriented MDA[20][21][22] is shown as Fig.6, Fig.7,Fig.8 and Fig 9.

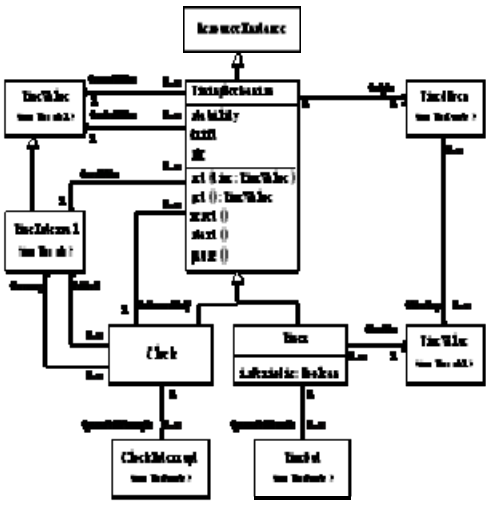

Figure 6.. Time mechanism model-CIM model[23]

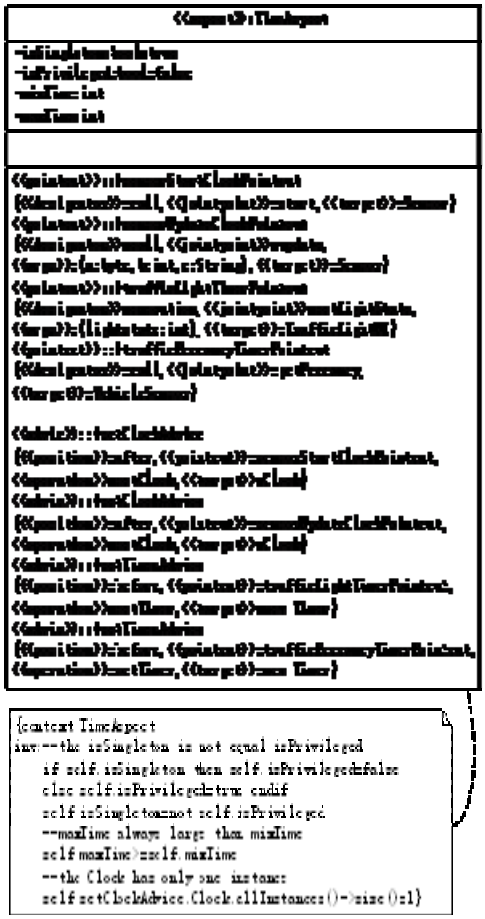

Figure 7. Aspect oriented time model: PIM model of time

OCL supplements UML by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics. Time aspect is specified by OCL as follows.

Context TimeAspect:
inv: *--the isSingleton is not equal isPrivileged*
if self.isSingleton then self.isPrivileged=false
else self.isPrivileged=true endif
self.isSingleton=not self.isPrivileged
--maxTime always large than mixTime
self.maxTime>=self.mixTime
--the Clock has only one instance
self.setClockAdvice.Clock.allInstances()->size()=1

The time constraints of phase is specified by OCL as following:

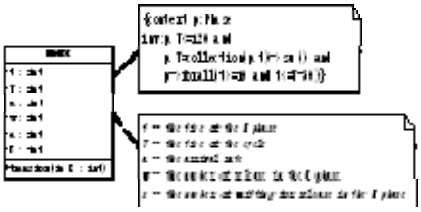

Figure 8. Time constraints of phase

Now we return to the model transformation, whose essential point is mapping to the special programming language code as shown in Fig.9.

Figure 9. Aspect code of time property: PSM model

Considering safety specification, the formal technique can be applied. The train control systems environment consists of train and road[9]. However, since just train is monitored, system environment is specified by MSV variable MSV_{train},which may have four states(distant, approached, on-crossinng and passed)[9].

$$n_1 = MSV_{train} = \{n_{11} = enter, n_{12} = approached, n_{13} = full\ h, n_{14} = distant\}$$

In addition, internal MSV variables, MSV_{timer} is considered to monitor passing of time[24].

$$n_2 = MSV_{timer} = \{0, 1, 2, \ldots, max\}$$

System-controlling component is just the system gate. It is shown by CSV_{gate} variable(called $C_{gate)}$, whose state is set by software as MoveDown ,MoveUp and closes/opens the road[9]. Thus,

$$c_1 = CSV_{gate} = \{c_{11} = open, c_{12} = close\}$$

We have the following safety constraints[9]:

$$approached_{train}(MSV_{train}, f_1) = [f_1 = A(2, MSV_{train})\ and\ f_2 = A(2, MSV_{train})]\ .\ h_1 = enter,\ h_2 = approached$$

$$full\_h_{train}(MSV_{train}, f_1) = [f_1 = A(3, MSV_{train})\ and\ f_2 = A(3, MSV_{train})]\ .\ h_1 = approached,\ h_2 = full\_h$$

$$distant_{train}(MSV_{train}, f_1) = [f_1 = A(2, MSV_{train})\ and\ f_2 = A(2, MSV_{train})]\ .\ h_1 = full\_h,\ h_2 = distant$$

The aspect -oriented fault-tolerant model is as shown in Fig.10.

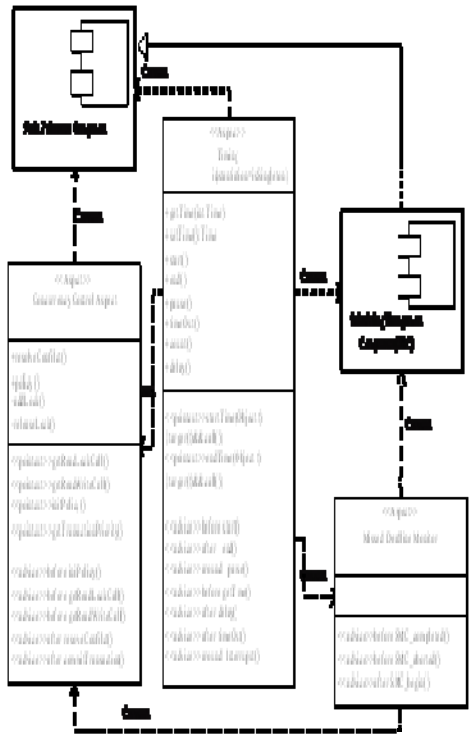

Figure 10. Aspect-oriented Fault-Tolerant Model: PIM model

```
public aspect Rule {
                pointcut TrainArrive (): call(approach
(MVS_name, CSV_name))
                after (): approach(){
                 if (MVS_name = approached && CSV_name =fault)
                }
}
```

Figure 11. Aspect code of train arrive: PSM model[24]

VII. CONCLUSION

In this paper, we proposed an aspect-oriented MDA for non-functional properties to develop real-time cyber physical systems. We illustrated the proposed method by the development of ITS and demonstrated aspect-oriented MDA approach that can be used for modeling non-functional characteristics of complex system, effectively reduce the complexity of software development and coupling between modules to enhance the system's modular.

The further work is devoted to developing tools to support the automatic generation of model and code.

ACKNOWLEDGMENT

This work is supported by the Major Program of National Natural Science Foundation of China under Grant No.90818008, National Natural Science Foundation of China under Grant No. 61173046 and Natural Science Foundation of Guangdong province under Grant No.S2011010004905.

REFERENCES

[1] Edward A. Lee, Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Published by authors, First Edition, 2011, 978-0-557-70857-4G.

[2] Object Management Group.OMG MDA guide v1.0.1[EB/OL].http://www.omg.org/docs/omg/03-06-01.pdf

[3] Wehrmeister, M.A., Freitas, E.P., and Pereira, C.E., et al., "An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems ", 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Santorini Island, Greece, May7-9, 2007, IEEE Computer Society, pp.428-432.

[4] Cancila, D., Passerone, R., Vardanega, T. and Panunzio, M., Toward Correctness in the Specification and Handling of Non-Functional Attributes of High-Integrity Real-Time Embedded Systems ， IEEE Transactions on industrial informatics, vol 6,No2,181-194, 2010.

[5] International Organization for Standarization, ISO/IEC Standard 9126: Software Engineering – Product Quality. 2004

[6] International Organization for Standarization, ISO/IEC14598 Information Technology - Software product evaluation. 2001

[7] Liming Zhu,Gorton, I. ,"UML Profiles for Design Decisions and Non-Functional Requirements",Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent, 20-26 May 2007 pp.8-9

[8] UML2.0 OCL Specification. http://www.comp.nus.edu.sg/~yangfei/HYP/UML2.0ocl.pdf

[9] Edward A. Lee. "Computing Needs Time". Talk or presentation, 12, November, 2010; Distinguished Lecture Series on Cyber-Physical Systems, Washington University, St. Louis.

[10] Frankel, D. S., "Model Driven Architecture:Applying MDA to Enterprise Computing", OMG Press.

[11] Jerome Hugues, "AADL for Cyber-Physical Systems: Semantics and beyond, validate what's next". Talk or presentation, 19, April, 2011.

[12] .Jeff C. Jensen, Danica H. Chang and Edward A. Lee. A Model-Based Design Methodology for Cyber-Physical Systems, Proc. Of first IEEE workshop on design.modeling and evaluation of cyber-physical systems (CYPHY), Instanbul, Turkey, 2011

[13] A. Banerjee, K. Venkatasubramanian, T. Mukherjee, and S.K.S. Gupta *Ensuring Safety, Security and Sustainability of Mission-Critical Cyber Physical Systems*, IEEE Proceedings special issue on cyber-physical systems, 2011.

[14] Solberg Arnor et al. Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development, Proceedings of the 29th Annual International Computer Software and Applications Conference. pp121-126 2005.

[15] Aniruddha Gokhale and Jeff Gray . An Integrated Aspect-oriented Model-driven Development Toolsuite for Distributed Real-time and Embedded Systems, AOSD Workshop on Aspect-Oriented Modeling Workshop, Chicago, IL, March 2005.

[16] Sven Burmester, Holger Giese, and Wilhelm Schafer, Model-Driven Architecture for Hard Real-Time Systems: From Platform Independent Models to Code.http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/ECMDA.pdf

[17] MARCO AURÉLIO WEHRMEISTER. An Aspect-Oriented Model-Driven Engineering Approach for

Distributed Embedded Real-Time Systems. Ph.D thesis, Porto Alegre: PPGC da UFRGS, 2009.

[18] Jingyong Liu, Yong, Zhong, Lichen Zhang, Yong Chen. Applying AOP and MDA to middleware-based distributed real-time embedded systems software process. 2009 Asia-Pacific Conference on Information Processing, APCIP 2009, pp 270-273, 2009.

[19] K.Ranjini, A.Kanthimath and Y.Yasmine. Design of Adaptive Road Traffic Control System through Unified Modeling Language. International Journal of Computer Applications, Volume 14– No.7, February 2011

[20] Yi Liu Yi Liu, Zhiyi Ma Zhiyi Ma and Weizhong Shao ,Integrating Non-functional Requirement Modeling into Model Driven Development Method Method, 2010 Asia Pacific Software Engineering Conference . p98-107, 2010

[21] Clemente, P. J., Sánchez, F., and Perez, M. A. "Modelling with UML Component-based and Aspect Oriented Programming Systems",Seventh International Workshop on Component-Oriented Programming at European Conference on Object Oriented Programming (ECOOP). Málaga, Spain.,2002,pp.1-7.

[22] G. Madl, S. Abdelwahed. “Model-based Analysis of Distributed Real time Embedded System Composition”, Proceedings of the 5th ACM international conference on Embedded software, New Jersey, USA , 2005.

[23] A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version 1.0.OMG Document Number:formal/2009-11-02.Standard document URL: http://www.omg.org/spec/MARTE/1.0

[24] Seyed Morteza Babamir and Saeed Jalili. Making real-time systems fault tolerant: a specification-based approach. Journal of Scientific & Industrial Research [J].2010.7, Vol.69:501-509.

[25] Robert Cartwright et al. Cyber-Physical Challenges in Transportation System Design. www.ee.washington.edu/research/nsl/aar.../WalidTaha-20081017192130.pdf