

CASE Tool Design for Graph Semantic Based Aspect Oriented Model



Anirban Sarkar¹, Narayan C Debnath²

¹Department of Computer Applications
National Institute of Technology
Durgapur, India

²Department of Computer Science
Winona State University
MN, USA

sarkar.anirban@gmail.com, ndebnath@winona.edu

ABSTRACT: Powerful CASE tools for Aspect Oriented System (AOS) are useful to improve the developer productivity and ensure syntactical correctness of the AOS model for effective system design. This paper has proposed a systematic approach to develop CASE tool for graph semantic based AOS model called, GAM, using meta-configurable environment called, Generic Modeling Environment (GME). Graph – semantic based Aspect Model (GAM) is a formal aspect oriented model to conceptualize the different facets of cross cutting concerns in AOS systems like join points, advices, PointCut etc. using graph based semantics. The proposed approach in this paper is comprised of set of semantic transformation rules for systematic mapping of concepts, semantics and notation syntax of GAM into the GME modeling paradigm elements. The approach is also supported with the correctness checking of such mapping using structural correspondence method. Moreover the expressiveness of the GME based CASE tool implementation of GAM has been illustrated using case study.

Keywords: Aspect Oriented System, Aspect Oriented Model, Conceptual Model, Graph Data Model, Generic Modeling Environment

Received: 28 August 2011, Revised 2 October 2011, Accepted 9 October 2011

©2011 DLINE. All rights reserved

1. Introduction

In the advent of increasingly complex, large and sophisticated software systems, the need for abstraction and modularization of system functionalities grow enormously. Separation of concerns is an added complexity and has gained parallel importance throughout the software life cycle. In recent days, one of the major goals in developing software system is separating and modularizing the set of concerns (cross-cutting concerns) those commonly pertain to multiple functional units. Unfortunately, the traditional techniques including object oriented paradigm may not be powerful enough to solidify common, global concerns that may encompass multiple aspects of functionality. Separation of concerns during design is no less important than at other stages of development. Overlooking the concerns at design phase, it will be hard to manage the complexity, the evolution and composition of the system with related cross-cutting concerns.

Aspect-orientation provides a new way of modularization, by clearly separating cross-cutting concerns from non-crosscutting ones. Although originally emerged at the programming level, aspect-orientation meanwhile stretches also over other development phases including the design phase. The goal of Aspect-Oriented Software Design (AOSD) is to promote advanced separation of concerns in the software design phase. From the software engineering point of view, the adoption of aspect-orientation in

design models is foremost important. As it may provides higher level insight about the software system with different level of views, abstraction, modularization and composition of application structure and behavior. It is important to work towards a general purpose AOSD that meets certain goals like, (i) Implementation language independent, (ii) Design-level composability and (iii) Compatibility with existing design approaches [16]. In the context of aspect oriented software development, two major properties, *quantification* and *obliviousness* are necessary [18]. This in no difference in case of aspect oriented software design. *Obliviousness* is the key properties that states, the base system components are not aware of the aspects that will eventually cross-cut those. On the other hand, *Quantification* is another key property of aspect oriented software that states, aspects may cross-cut an arbitrary number of base components simultaneously.

Aspects related concepts from programming level point of view have been introduced in [1, 5]. From a software development point of view, aspect-orientation originally emerged at the programming level with AspectJ [14]. Some key aspects related concepts are described below:

- **Concern:** A *concern* is an interest which pertains to the system's development, its operation or any other matters that are critical or otherwise important to one or more stakeholders. A concern may be represented as *Base concern* or *Cross-cutting concern*. A *base concern* is a unit of modularization, formalizing a non-crosscutting concern underlying to specific component design and which can be modeled using available general purpose paradigm like Object Oriented paradigm. Base concerns can be encapsulated cleanly and the components based upon base concerns are decomposable vertically through its hierarchical order. Whereas, *cross-cutting concerns* are system wide and pertained to multiple functional units (mainly components with base concerns).
- **Aspect:** Aspect is a unit of modularization, formalizing Cross-cutting concerns. Aspect also describes where and how the cross-cutting concerns can be adopted into the base concern.
- **Weaving:** Weaving refers to the mechanism of composition of the aspects to other concerns, either base concerns or other cross-cutting concerns or both. Weaving at programming level, the existing codes of concerns are enhanced with new code from related aspects. But at design level weaving mechanism deploy new *structural of behavioral semantics* from the related aspects to the target concerns in design model. Thus model level weaving in one hand can provide separate level of abstraction on aspect adoption and on other hand the well formed rule can be described for weaving mechanism which will consequently effect at programming level.
- **Join Point:** A *join point* species *where* an aspect needs to be weaved in the target concerns. Thus, a join point is a representation of an identifiable element in the target concern of the underlying system. At the design level join points are representation of *structural elements* (like attributes, structure or class) or *behavioral elements* (like methods or events) conceptualized by structural or behavioral semantics of the software. While at the same time, they can be also represented as *static* or *dynamic elements* in design model level.
- **Advice:** An advice refers to the service need to provide for the extension of structural or behavioral semantics to the target concerns on weaving. At programming level the method body of service needs to be executed when selected joint points reached and governed by the specified weaving rule.
- **PointCut:** PointCut refers to a relationship between a specific join point or a set of join points (selected join points) and advices. In higher abstraction level at design phase PointCut relation specifies how aspect can interact with the base concerns. The selected join points are the PointCut designator in the target concerns.

This paper has proposed a design CASE tool for Graph – semantic based Aspect Model called, GAM. The preliminary version of GAM has been published in [25]. In the proposed approach, the modeling concepts of GAM have been implemented using a meta-configurable modeling environment called, Generic Modeling Environment (GME) [19], using semantic mapping techniques. Such metamodel level implementation can be used as prototype CASE tools for modeling AOS using GAM. For appropriate applicability of conceptual level aspect model, besides the beneficiary of high level representation of the related concerns and cross-cutting features, detailed specification at a low level of abstraction is necessary for an automated execution of the design model to a code level through some CASE tool. In this context, GAM is a formal aspect oriented model to conceptualize the different facets of cross cutting concerns in AOS systems like join points, advices, PointCut etc. using graph based semantics. The model is revealed with a set of concepts to the conceptual level design phase of AOS, which are understandable to all stake

holders of such system, independent of implementation issues. The GAM is capable to visualize the cross-cutting concerns at high level of abstraction and even at low level of abstraction with finer detail of aspect interactions.

2. Related Research

The Aspect-oriented design model intends to provide broader and formal descriptions of the aspects related concepts and their relationship. A comprehensive design model should be accompanied with guideline for design of such system and weaving mechanism at model level. Moreover, a conceptual level design model for AOS is closer to the user perception about software system and related cross-cutting concerns.

In this decade, several aspect oriented modeling approaches [2, 3, 4, 6, 8, 9, 11, 12, 17] has been proposed to adopt the aspect orientation in design phase of software lifecycle. Majority of these approaches has been extended the UML notations. UML and extensions to UML represent software elements using a set of language elements with fixed implementation semantics (e.g. methods, classes). Henceforth, the proposed approaches using extension of UML, in general, are logically inclined towards implementation of cross-cutting concern. This may not reflect the cross-cutting concerns related to the base concerns with high level of abstraction to the user. In other word, aspect oriented design model with UML extension cannot be considered as semantically rich conceptual level model. Only two approaches [6, 9] are based on non-UML notations. In [6] a model has been proposed to address the early design phase of aspect oriented software on two constructs Concept and Composite. Whereas, in [9] more emphasize has been given on separation of concerns and resulted with general purpose

concern space modeling which includes logical concerns, physical concerns and their relationships. But both of these approaches are informal in nature. Further the approaches are not rich enough to model different facets of aspect orientation like PointCut relation, Join Points, Weaving mechanism etc. Also few approaches [2, 8, 10] provide aspect-oriented software design guidelines. Moreover, few approaches [3, 8, 10] have been addressed the mechanism of weaving at model level. A proper semantic of weaving mechanism at model level is important because, firstly, it will provide a well formed guideline of code weaving at programming level and secondly, many weaving related crucial issues like possible effects, conflicts etc. on weaving can be identified and modeled at the design level. Thus it will reduce a considerable complexity at code level for the developer of AOS system. In this context, in [7] an approach has been proposed for addressing domain specific cross-cutting concern and weaving in typed-system.

In general, most of the approaches are lacking from proper formalization of aspect oriented concepts which includes the aspect related basic concepts like join point, advice, pointcut etc. Also most of the approaches have not addressed a formal operational model for aspect oriented software which includes the operational semantics for join point selection, concern projection, aspect adoption etc.

Further, very few attempts have been made to develop CASE tools for aspect-oriented software analysis and design. The research effort for these are important because, *firstly*, AOS design modeling should be supported by powerful CASE tools to improve developer productivity and ensure syntactical correctness of the model; *secondly*, such CASE tools will guide direct mapping between the model notation and the supported implementation language and further will allow automatic code generation based on the design model [20]. The THEME approach described in [10] has been implemented as Theme/UML [15] with the aim to develop a Theme/UML marking profile that facilitates development of the models within standard UML editors. It also includes a transformation package which can transform composed Theme/UML model into structural code using template-based approach. In [20] a CASE tool has been described to generate aspect code template in AspectJ language from on UML notation based aspect model. The approach has emphasized more on the aspect code than the validation of the semantic constructs for AOS. A graph transformation based AOS model called MATA has been described in [21] and it extends the UML notation for design of AOS. MATA is implemented as a vendor-independent tool based on Rational Software Modeler (RSM) where each aspect is modeled as a package.

3. GAM: The Graph-semantic Based Aspect Model

The GAM [25] is the extension of the comprehensive object oriented model to conceptualize both the base and crosscutting concerns of aspect oriented software system. It contains all the formal details, those are necessary to specify the base concerns, their properties and set of services provided by base concerns. Also it describes the facets of system wide crosscutting concerns those will commonly pertained to the base concerns. To conceptualize the cross-cutting concerns, the model facilitates

the formal description of join points, aspects, advices and pointcut relations.

The proposed model allows the system to be viewed as *Two Tier* architecture namely *Base System Tier* and *Aspect Tier*. The *Base System Tier* use to model the base concerns of the software system mainly using object oriented paradigm. And the second tier, *Aspect tier* use to model the cross-cutting concerns of software system. Two tiers will be associated with special type of relation called *PoinCut* relation. Each tier can be viewed as Graph $G(V, E)$ in layered organization. For each tier, at the lowest layer, each vertex represents an occurrence of an attribute or a data item to conceptualize the possible instances of elementary properties, parameters etc. Each such basic attribute is to be represented as separate vertex. The concepts of tiers are important to conceptualize the separation of concerns. Each tier separately can be organized as set of layers to provide the different level of abstraction of the concerns like base and cross-cut and can be deployed using the object oriented paradigm. The inter-tier association and interactions can be conceptualized using special type of relation called *PointCut* relation, description of join points and advices, which exactly not follow the object oriented principles.

In the model, any one tier can be deployed separately regardless of implementation and abstraction of another tier. Thus the components of the base system tier are *Oblivious* about the components of aspect tier. On the other side, the description of join points, advice and *PointCut* relation in together realize the properties of *Quantification*.

3.1 Components of GAM

Since from the topmost layer, a set of vertices V is decided on the basis of level of data abstraction whereas the set of edges E is decided on basis of the association between different semantic groups. The basic components for the model are as follows,

(i) *Elementary Semantic Group (ESG)*: An elementary semantic group is an encapsulation of all possible instances or occurrences of an attribute, that can be expressed as graph $ESG(V, E)$, where the set of edges E is a null set \emptyset and the set of vertices V represent the set of all possible instances. ESG is a construct to realize the elementary property, parameter, kind etc. of some related concern. The graphical notation for the any ESG is *Circle*.

(ii) *Service Semantic Group (SSG)*: The service semantic group is a construct to characterize an abstraction of method or action or event that can be invoked to provide a specific service. The SSG is important to realize the functional behavior of some concerns. Typically a SSG can be expressed as graph $SSG(V_s, E_s)$, where $V_s = RetParam \cup InputParams$, *RetParam* is return parameter of type Elementary Semantic Group (ESG) or Concern Semantic Group (CSG) [see Section 3.1 (iii)] and *InputParam* is a set of ESGs or CSGs. The *RetParam* and *InputParam* can be equal to null set for some SSG and correspondingly $ES = \emptyset$. Otherwise, the *ES* is the association within V_s and can be expressed as a bijective (one-to-one and onto) function $f: InputParam \rightarrow RetParam$. The graphical notation for the any SSG is *Oval*.

(iii) *Concern Semantic Group (CSG)*: The lowest layer concern semantic group is an encapsulation of zero or more related ESGs and zero or more related SSGs to represent one elementary concern of the software system. Any CSG can be represented as a graph (V_C, E_C) where vertices $V_C \subseteq E_G^* \cup S_G^*$, where E_G^* and S_G^* are the set of ESGs and SSGs with zero or more occurrences and the set of edges E_C represents the association amongst the vertices. Also for SSG $S_i \in S_G$, normally, $RetParam \cup InputParams \in E_G \cup C_G$, where C_G is the set of all CSG. The E_C is the set of edges to exhibit the association between the S_G and related EG within the CSG. Association of multiple CSGs can be realized in two categories. In case of *simple association*, two or more associated CSGs either of same layer or of adjacent layers will share a common set of ESGs. Cardinality also can be mentioned in this association. Otherwise, lower layer CSGs will maintain an *aggregation association* with the adjacent upper layer CSG. The upper layer CSG can be formed by inheritance or composition of one or more lower layer CSGs along with encapsulation of zero or more related ESGs and SSGs. For any CSG, it is also possible to designate one or more encapsulated ESGs or CSGs as *determinant vertex (DESG)* which may determine an unordered set of instances of encapsulated ESGs or CSGs. The graphical notation for any CSG is *square* and determinant vertex is *Solid Circle*.

(iv) *Base Concern Semantic Group (BCSG)*: The Base Concern Semantic Group is a specialized form of CSG to conceptualize the Base Concerns of the system. A set of BCSGs composed through association or inheritance relationship to form the *Base System Tier*. The graphical notation for any CSG is *Square* with tag *Base*.

(v) *Advice Semantic Group (AdSG)*: Advice Semantic Group is a special form of SSG and is a construct for advice services with kind description. AdSG can typically expressed as the tuple $\langle ESG_{Kind}, SSG_{Advice} \rangle$, where SSG_{Advice} is the advice service need to provide for the refinement of structural or behavioral semantics of the target BCSGs to incorporate the cross-cutting concern and ESG_{Kind} provides the further information about the kind of refinement will be performed on incorporating the

cross-cutting concerns. The graphical notation for any AdSG is *Dotted Oval* with the tag of relevant ESG_{Kind} value. ESG_{Kind} typically holds the values like, *Replace, Create, Delete*.

(vi) *Join Point Semantic (JPS)*: A join point semantic species the identifiable components within any BCSG where an advice is required to incorporate. On the effect of incorporation of the advice, the specific BCSG will be enhanced either by replacement or by creation or by deletion of one structural or behavioral semantic. JPS is a reference of ESG or SSG or any CSG related to target base concern. It can be typically expressed as $\langle NameReference, ESG_{PosType} \rangle$, where *NameReference* is the alias name of the referred ESG or SSG or CSG and $ESG_{PosType}$ provides the further information regarding the relative position where the cross-cutting concern can be incorporated. In GAM a JPS can be treated as a referential vertex. The graphical notation for any JPS is *Solid Triangle* with the tag of relevant $ESG_{PosType}$ value. $ESG_{PosType}$ typically holds the values like, *Before, After, Around, BeforeInstance, AfterInstance*.

The *BeforeInstance* and *AfterInstance* values of $ESG_{PosType}$ can be typically used for identification of dynamic JPS. Static JPS are used to redefine the related construct and dynamic JPS are used to express that advice will be incorporated at instance level. Whereas, other values of $ESG_{PosType}$ are suitable for static JPS. Using reference for the join point semantic enables the usage of both static and dynamic join point concepts.

(vii) *Aspect Semantic Group (ASG)*: The Aspect Semantic Group is a specialized form of CSG to conceptualize the crosscutting concerns of the system. An ASG can be created by encapsulating several related *AdSGs*, *ESGs* and other *SSGs*. A set of *ASGs* can be composed through association or inheritance relationship to form the *Aspect Tier*. The graphical notation for any CSG is *Dotted Square*.

(viii) *Relation Types*: The proposed GAM provides a graph structure to represent any system with its cross-cutting concerns. The edges of the graph represent relationships between or within the constructs of the model. In GAM, we have used *four* types of edges to represent different relationships. The type of edges and their corresponding meanings are as follows,

(a) *Association*: Associations are defined either between determinant ESG and other encapsulated ESGs, SSG and constituent CSG, or between two associated CSGs of similar type (either BCSG type or ASG type), or between InputParam and RetParam of some SSG. Graphically association can be expressed using *Solid Undirected Edge*. With any CSG, this represents a bijective mapping between determinant ESG and other ESGs or composed CSG.

(b) *Link*: Links are defined between two adjacent layer parent CSG and inherited CSG of similar type (either BCSG type or ASG type). Links are used to represent the inheritance relationships. Graphically link can be expressed using *Solid Directed Edge*.

(c) *Reference*: References are defined between *JPS* and the referred identifiable components (like ESG or SSG or constituent BCSG) of some *BCSG*. Reference edges are used to represent the name alias of the related components of the BCSG. Graphically reference can be expressed using *Dotted Edge*.

(d) *PointCut*: At the lower layer, PointCuts are defined between an AdSG and a specific JPS or a set of JPSs. From the topmost layer it can be expressed between related BCSG and ASG. Multiple JPSs can be related with one AdSG. On the other hand multiple AdSG can be related with one JPS. From the topmost layer PointCut exhibit the crosscutting relation between BCSG and ASG. In GAM, the interactions between the Base System Tier and Aspect Tier are expressed using a set of PointCut relationships. Since, at conceptual level design, *Base System Tier* is oblivious about the *Aspect Tier*. So in the model, PointCut has been treated as *first class relationship*. Besides the exhibition of abstraction properties, as a first class relationship, dynamic construction of PointCut relation is supported by the model. Graphically PointCut can be expressed as *Doted Directed Edge* with name tag.

The summary of GAM model constructs and their graphical notations have been given in Table 1 and Table 2.

3.2 An Example of GAM

As example, we have applied the Observer Pattern as an aspect to a simpler version of *Hospital Management System*. Let, a hospital has a group of *Doctors* and *Nurses*, who are served to the relevant *Departments*. *Patients* are admitted in some certain *Ward* in the hospital. The admitted *Patient*, *Doctor* and *Nurse* in the hospital have unique *id* and *name*. The Doctors manage a list of patients through the services like *admit(Pat)* and *Discharge(Pat)* and associate a *Nurse* to a *Patient* using the service *Assign (Nurse, Pat)*. The status of availability of each *Nurse* should be kept up-to-date. Thus, each time a *Patient* is admitted or discharged, the concerned *Doctors* and *Nurses* have to be notified. This notification functionality is not provided by the base system, but is applied using the observer pattern. *Hospital Management System* forms the *Base System Tier* of the proposed model.




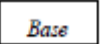


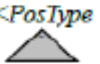
GAM Constructs	Description	Graphical Notation
ESG	Elementary Semantic Group	
Determinant ESG	Determinant vertex of any CSG which will determine the other member vertices in the CSG	
CSG	Concern Semantic Group	
BCSG	Base Concern Semantic Group to realize any Base system tier component	
SSG	Service Semantic Group	
AdSG	Advice Semantic Group	
JPS	Join Point Semantic	

Table 1. Summary of GAM constructs and their graphical notations





GAM Relation Types	Description	Graphical Notation
Association	Defined between determinant ESG and other encapsulated ESGs as well as between two associated CSGs of similar type	
Link	Defined between two adjacent layer parent CSG and inherited CSG of similar type	
Reference	Defined between JPS and the referred identifiable components of BCSG	
PointCut	Defined between AdSG and JPS at lower layer and between related BCSG and ASG at topmost layer	

Table 2. Summary of GAM relations types and their graphical notations

The *Observer Pattern* defines a one-to-many dependency between objects in a way that whenever a *Subject(Patient)* changes its state, all its dependent Observers (related Doctors and Nurses) are notified using *update(Subject)*. While any *Observers* can register and un-register with their *Subjects* of interest using the services *start(Subject)* and *stop(Subject)*. A *Subject* keeps a list of interested Observers using *add(Observer)* and *remove(Observer)*. Applying the observer pattern, however, affects the base system tier's modularity. In particular, the SSGs *getState()* and *update(Subject)* have to be implemented by Patient and Doctor respectively. Additional modifications are necessary to call *start(Subject) / stop(Subject)* and *notify()*. Therefore, the observer functionality can be regarded as crosscutting concern and, thus be realized with the Aspect Tier of the proposed model.

The Hospital Management System with Observation Aspect using GAM notation has been shown in Figure 1. At the top level view [Figure 1.(a)] two tiers are connected using PointCut relation only. The view is constituted with topmost layer BCSGs, ASGs and the set of PointCut relations. In the hierarchical view [Figure 1.(b)], the PointCut relations are represented between JPSs and AdSGs. In the system four services of aspect tier have been represented as advise services namely, *getState()* of *Subject* ASG and *Start(Subject)*, *Stop(Subject)* and *Update(subject)* of *Observer* ASG. Among them *getState()* and *Update(Subject)* will refine the *Patient* and *Doctor* BCSG structure respectively. The *Start(Subject)* and *Stop(subject)* will be invoked dynamically on the state changes of some *Patient* instance. Correspondingly, *getState()* and *Update(Subject)* are connected with static JPS *J1* and *J2* respectively using PointCut relation. Whereas, *Start(Subject)* and *Stop(subject)* are connected using dynamic JPS *J3* and *J4* respectively using PointCut relation.

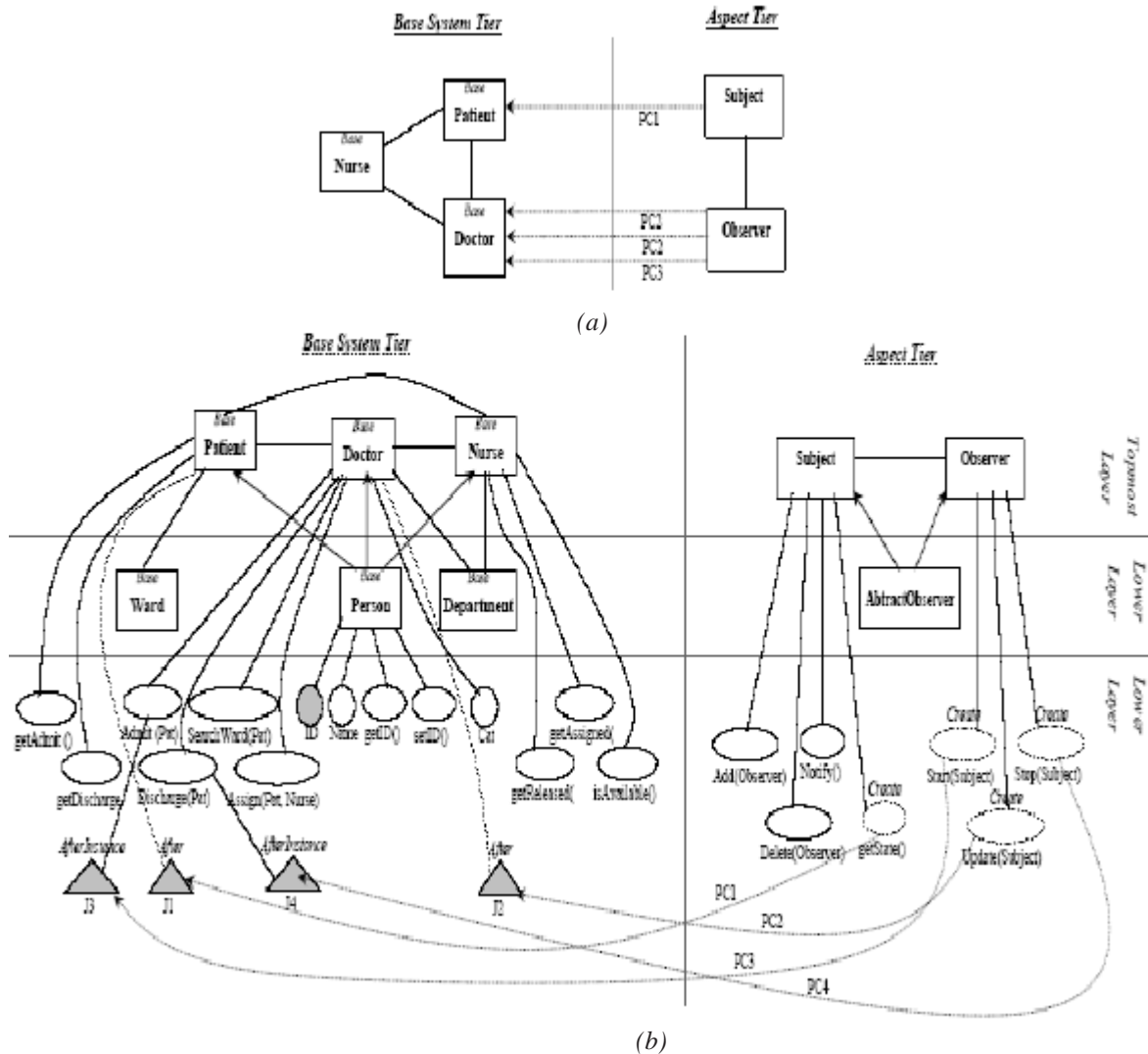


Figure 1. Hospital Management System with Observer Aspects using GAM notations
 (a) Top Level View, (b) Hierarchical View

3.3 Features of GAM

Using GAM, an Aspect Oriented System can be viewed as a Graph (V, E) in layered organization with two tier architecture. The graph structure maintains the referential integrity inherently. Apart from these, one of the major advantages of the model is that it defines each level of structural detail on the constructs which are independent of code level representation of AOS.

The features of the proposed model are as follows,

(a) *Explicit Separation of structure and Content*: To model AOS, the proposed model reveals a set of structures and relationships, those are not instance based. So, the nature of contents that corresponded with the instances has been separated from the system's structural descriptions.

(b) *Separation of Concerns*: The proposed GAM is based on *Two Tier* architecture namely *Base System Tier* and *Aspect Tier* which are related with different type of concerns. Both the tier can be design independently. The interactions between these two tiers can be established through PointCut relations. The concept of tier is fully capable to separate the concerns for such system.

(c) *Obliviousness*: In GAM, the awareness of aspects is not required apriori to design the base system tier.

(d) *Quantification*: GAM allows this quantification property of cross-cutting semantics. The description of JPS, AdSG and PointCut relation in together can realize this property.

(e) *PointCut Relation*: In the proposed model, the PointCut relation has been treated as first-class entity. So the relation can be designed with several level of abstraction. From the topmost layer PointCut exhibit the cross-cutting relation between BCSG and ASG and at the lower layer it expresses the relation between JPS and associated AdSG.

(f) *Dynamic Join Points*: The proposed model supports both static and dynamic join point semantics. Different *PosType* values like *BeforeInstance* and *AfterInstance* are used to express the *Dynamic Join Point Semantics*.

3.4 Usage of GAM in Effective System Modeling

GAM is graph semantic based aspect oriented model which can formally conceptualize the different facets of cross cutting concerns in AOS systems like join points, advices, PointCut etc. Besides the capability to represent the cross-cutting concerns at high level of abstraction, the model can visualize such concerns even at low level of abstraction with finer detail of aspect interactions using layered approach. This layered architecture is useful to model large scale complex system with crosscutting concerns.

Several non-functional requirements (NFR) such as security, usability, integrity, performance etc. are also critical for large scale complex system and those NFRs cross-cut multiple modules of the base system. In classical software engineering, the cross-cutting concerns are modeled and integrated with the base system at very latter stage of software development process (construction phase), which results tangled representations at the code level and those are difficult to understand and maintain. AOS aims to identify and specify such cross-cutting concerns in separate modules and results better support for modularization with the reducing cost of development, maintenance and evolution of such software. Thus “aspects” are fundamental modeling primitives for cross-cutting concerns to provide improved support for separation of cross-cutting functional and non-functional properties during early stage of software development process (requirements and design phase), hence offering a better means to identify and manage conflicts arising due to tangled representations [26]. Several research proposals are there to model the NFRs using AOS fundamentals [27, 28]. But those proposals are not formal and also not supported with CASE tools implementations. But in case of large and complex system, formal analysis is very useful for detecting possible errors early in the design phase. In this context, GAM is fully capable to model NFRs formally at the design level for the large and complex software. Moreover, the proposed GME based CASE tool (discussed in the next section) for GAM is capable to implement such NFR models as cross-cutting concerns.

4. Mapping GAM in Generic Modeling Environment (GME)

The GME provides meta-modeling capabilities that can be configured and adapted from meta-level specifications (representing the Conceptual model) that describe the domain concept [7]. It is common for a model in the GME to contain several numbers of different modeling elements with hierarchies that can be in many levels deep. The GME supports the concept of a viewpoint as a first-class modeling construct, which assists a modeler in separating the concerns of multiperspective views using aspects [19]. Each GME viewpoint or aspect describes a partitioning that selects a subset of conceptual modeling components as being visible. Although GME offer a powerful implementation of viewpoints, however, it does not fit completely within the definition of aspects. The AOS property of quantification is not any way supported by the viewpoints concept. For example, designer cannot quantify over a join point and related advices.

4.1 Introduction of GME

The Model Integrated Computing (MIC) developed at the Institute for Software Integrated Systems at Vanderbilt University that provides a framework for developing domain artifacts for computer-based systems and relies heavily on the use of domain-specific languages to describe the final system implementation. MIC is a way to develop systems while addressing problems of system integration and evolution. MIC is used to create and evolve integrated, multiple-view models using concepts, relations and model-composition principles used in the given domain. The core tool in MIC is the Generic Modeling Environment (GME), which stemmed from earlier research on domain-specific visual programming environments [22]. GME is a generic and configurable toolkit for creating domain-specific modeling and program synthesis environments. The configuration is accomplished through metamodel specifying the modeling paradigm of the application domain. The *modeling paradigm* or *meta-model* specification in GME contains all the syntactic, semantic, and presentation information regarding the domain. The modeling paradigm defines the family of models that can be created using the resultant modeling environment. The generic modeling concepts of GME have

been well described in [19]. It includes several well defined meta-meta-modeling elements to create the domain specific modeling paradigm. Among all meta-meta-modeling elements, *Atoms*, *Models*, *Connections*, *References* and *Set* are commonly called *First Class Objects*, or *FCOs*, emphasizing their central role in any modeling project. The summary of the major meta-meta-modeling elements [23] are as follows,

(i) *Atoms*: Atoms are basic, limited type of entity. The name “atom” indicates that it has no internal structure (i.e. contained objects). Every feature of an atom that can be represented in a model is contained in the atom’s name, attributes, and the relations it participates in.

(ii) *Models*: This is second generic type of entity, are very similar to atoms. The main difference lies in the ability of models to contain atoms, other models, and other types of objects. Thus, models have internal structure. When viewed together, they form tree-like containment hierarchies of entities. Models can be opened, showing a diagram of their internal structure using level.

(iii) *Connections*: These are the primary concepts that represent relationships. Connections normally describe a relation between two objects, and this relation is represented as a line in a particular color and style connecting the two objects. Connections can also have their own attributes.

(iv) There are two other important generic concepts, *references* and *sets*. A *reference* is an entity with a built-in association for a single object. This association allows it to act like a pointer or an alias for that object. A *set* forms an association between several similar objects. Sets are often thought of as collections or categories.

(v) *Attributes*: Attributes are “bound” to FCOs and are used to store information in an FCO. In GME, attributes can contain text, integers, real numbers, or boolean values.

(vi) *Folders*: These are containers for different sections of a modeling project. Folders are auxiliary objects and are used merely for organization. Each modeling project contains at least one folder, called the root folder, located at the very top of the hierarchy.

(vii) *Aspects/Viewpoints*: These represent different “views” of the structure of a model. A model with several aspects will display different subsets of its contained entities depending on the aspect selected.

(viii) *Constraints*: These are validity rules applied to a model. They are expressed using OCL, a predicate language and can be checked on-line (while the model is being built) or in on-demand mode.

4.2 Semantic Mapping of GAM to GME

In this section, the semantic mapping of the concepts of GAM to the elements of GME has been described. This will result in developing the GAM modeling paradigm (meta-model) in GME and which will realize the concepts, semantics and notation syntax of GAM for designing and analyzing the AOS. For the purpose, a set of transformation rules have been proposed to draw the structural correlation between the GAM concepts and intended modeling paradigm in GME. The transformation rules are drawn from the concepts of GAM and the basic meta-meta-modeling elements of GME. The rules are as follows,

TR1: The *ESG*, *DESG*, *InputParam* and *RetParam* of GAM should be mapped as *Atom* in GME. The *type* and *size* of the *ESG* or *DESG* should be described as *Attributes* in GME.

TR2: The *SSG* construct of GAM should be mapped as *Model* in GME and should encapsulate the atoms declared for *ESG*, *InputParam* and *ReturnParam*.

TR3: Any *CSG* construct (either *BCSG* or *ASG*) should be mapped as *Model* in GME and should encapsulate the atoms declared for *ESG* and *DESG* and model declared for *SSG*.

TR4: The *Advice Semantic Group (AdSG)* construct of GAM should be mapped as *Model* in GME and should encapsulate the atoms declared for *InputParam* and *RetParam*. The *Kind* of *AdSG* should be mapped as *Attribute* in GME with enumeration of the values like, *Replace*, *Create*, *Delete*.

TR5: The *JPS* construct of GAM should be mapped as *Model* in GME. The *PosType* concept of *JPS* should be mapped as *Attribute* in GME with enumeration of the values like, *Before*, *After*, *Around*, *BeforeInstance*, *AfterInstance*.

TR6: The *Relationships (PointCut, Reference, Link and Association)* of the GAM are mapped as *Connections* in GME with the source and destination as described in the GAM concepts.

TR7: The *Layers* concepts of GAM will be mapped using the *Level* concept of GME. Thus all details of *ESG*, *DESG* and *SSG*

belongs to some CSG will be shown in inner layer. The internal structure of any CSG will be indicated using the *Port* concepts of GME.

TR8: The *Tier* concepts of GAM will be mapped using the *Aspect* concept of GME. In accordance with the *Base System Tier* and *Aspect Tier* concept two aspects has been used in GME with the relevant model elements in GME. Using the *Base System* aspect in GME, the atoms/models described for ESG, DESG, BCSG, SSG, JPS and all related Connections will be visualized. On the other hand, using the *Aspect System* aspect, in GME the atoms/models described for ESG, DESG, AdSG, ASG and all related Connections will be visualized.

The summary of the semantic mapping from the GAM constructs to GME elements has been given in Table 3.

4.3 Correctness of Mapping

The set of proposed transformation rules described in previous sub-section (section 4.2) facilitates the systematic transformation of conceptual level AOS model like GAM to the equivalent modeling paradigm or meta-model specification in GME. The correctness of the model transformation can be proved using the structural correspondence approach described in Narayanan et al. [24]. In every model transformation, there is a correlation or correspondence between parts of the input model and parts of the output model. One can specify these correlations in terms of the abstract semantics of the source and target model constructs. The approach of Narayanan et al. describes that, if a transformation has resulted in the desired output models, there will be a verifiable structural correspondence between the source and target model instances that is decidable.

Moreover, the transformation can be accepted as correct, if a node in the source model and its corresponding node in the target model satisfy some correspondence conditions.

In case of GAM, Table 3 can be use to specify the correspondence between conceptual model level identifiable constructs with GME modeling paradigm elements. Moreover the Table 3 can be treated as the look-up table for the cross links between the source model (GAM) and target model (Paradigm in GME).

Further, the proposed set of transformation rules will realize the correctness condition of such mapping. In our proposed approach, the correspondence rules must ensure that semantics and syntax for every constructs in the GAM and its equivalent meta-modeling elements in GME modeling paradigm.












GAM Constructs	Graphical Notation	GME Elements
ESG		"Atom"
Determinant ESG		"Atom"
CSG		"Model"
BCSG		"Model"
SSG		"Model"
AdSG		"Model" with <i>Kind</i> as enumerated "Attribute"
JPS		"Model" with <i>PosType</i> as enumerated "Attribute"
Association		"Connection" to realize the encapsulation in GAM Concept
Link		"Connection" to realize the inheritance in GAM concept
Reference		"Connection" between JPS and {BCSG, SSG}
PointCut		"Connection" between JPS and AdSG
Layer		"Level"
Tier		"Aspect"

Table 3. Semantic Correspondences among GAM Constructs and GME Meta-meta-modeling Elements

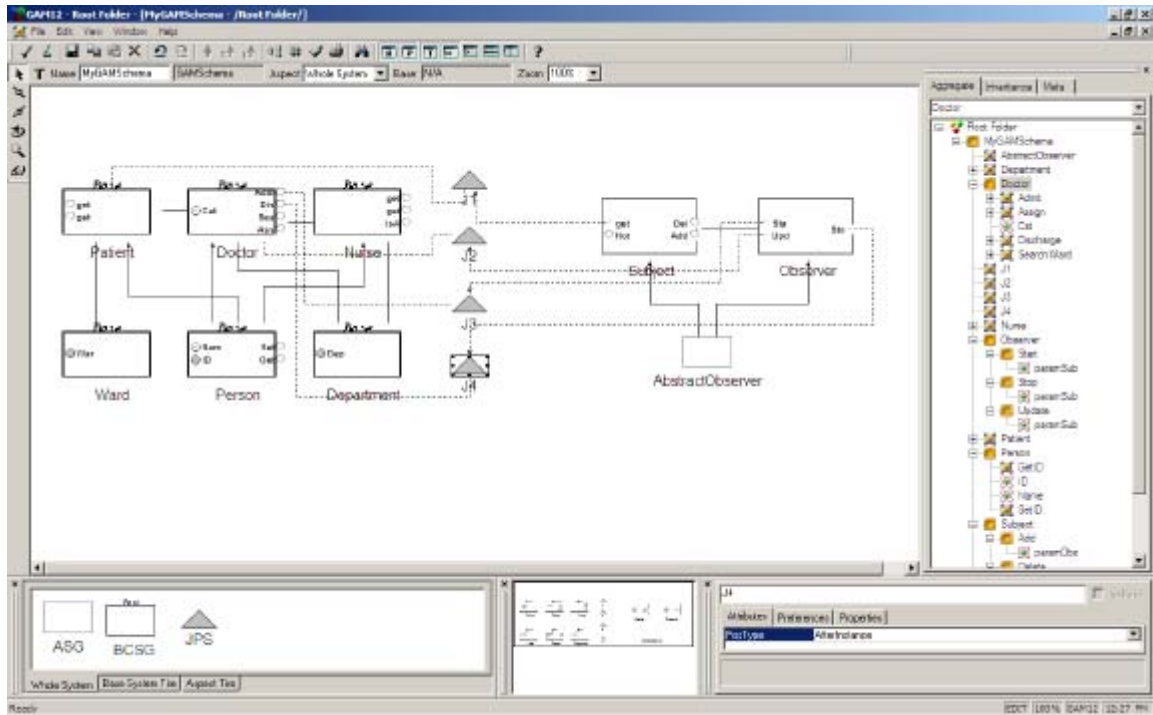


Figure 2. Hospital Management with Observer pattern in GME

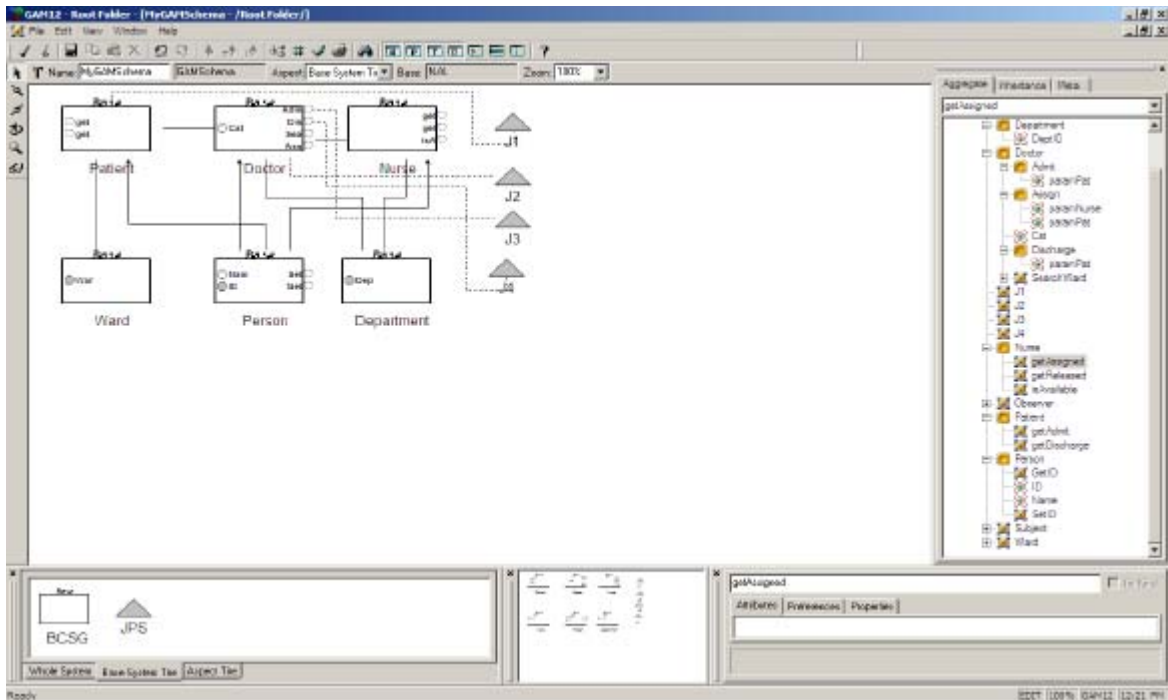


Figure 3. Base System Tier of Hospital Management in GME

5. Implementation of GAM using GME

In the context of GAM, the lower layers can be conceptualized using levels in GME. The tiers (Base System Tier and Aspect Tier) can be conceptualized using aspects or viewpoints in GME. The modeling paradigm for GAM has been implemented using the

proposed transformation rules. The GAM modeling paradigm describes a family of models in the aspect-oriented software domain and hugely useful for design of instance model in such domain. The implemented prototype is supported by the oblivious properties of AOS but is not supported with the quantification properties of AOS. Three viewpoints or aspects have been used in the proposed paradigm, namely, *Base System Tier*, *Aspect Tier* and *Whole System*. The Whole System viewpoints to realize the entire system which consists of base concerns and its cross-cutting concerns. The former two viewpoints are to realize Base System Tier and Aspect Tier respectively.

The Hospital Management System with Observer pattern described in Section 3.2 has been modeled using GAM implementation in GME [Figure 2]. Different possible *PosType* values for JPS and *Kind* values for AdSG has been implemented using Attribute meta-meta element of GME. The *Base System Tier* and *Aspect Tier* viewpoints/aspects have been shown in Figure 3 and Figure 4. The concept JPS has been treated as references of the identifiable components of BCSG or ASG, so the related JPSs are visible in relevant viewpoints.

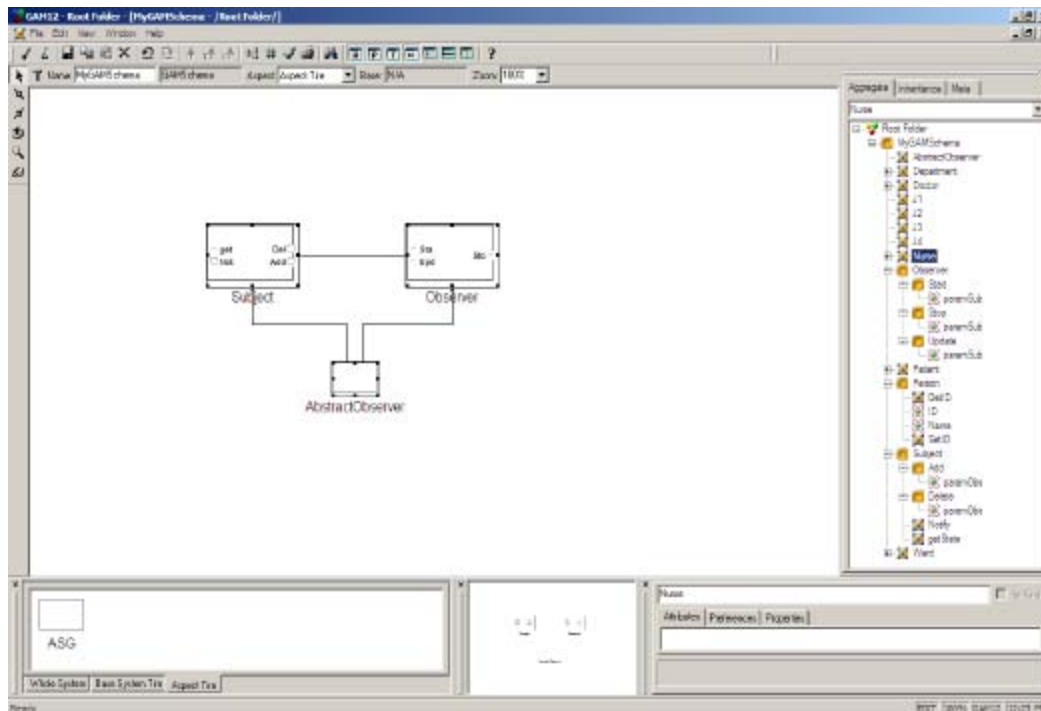


Figure 4. Aspect Tier of Hospital Management in GME

6. Conclusion

In this paper, an approach has been introduced for the conceptual level design of the aspect oriented system using graph based semantics called, GAM. The model allows viewing the entire aspect-oriented software system as a Graph (V, E) in layered organization with two tier architecture, namely, *Base System Tier* and *Aspect Tier*. The layered organization in both the tiers of the model facilitates to view the system structure from different level of abstraction. Moreover, the set of constructs for the GAM facilitates the user and designer of the system with better understandability which is again independent from the implementation issues.

A systematic approach has been proposed in this paper to develop a CASE tool for GAM by implementing the modeling concepts of GAM in Generic Modeling Environment (GME). For the purpose, a set of semantic transformation rules have been proposed to facilitate the systematic mapping of GAM constructs, relationships and their semantics into the GME metamodeling elements. The resulted meta-modeling paradigm will be able to realize a family of models related to AOS systems using the concepts, syntax and semantics of GAM. The GAM modeling paradigm in GME supports multiple viewpoints or aspects to realize the separation of concerns or in other words to realize the *Tier* concepts of GAM. Further it supports the oblivious property of AOS. The GAM paradigm implementation in GME can be used as prototype CASE tools for analysis and design of

base and its cross-cutting concerns of AOS. Moreover, the modeling concepts of GAM along with the prototype CASE tools proposed in this paper can be proven to be useful for effective system modeling where the system is large, complex and accompanied with critical non-functional features like security, usability, performance etc.

The future study will include the enhancement of the GAM paradigm in GME for automatic generation of aspect code template using aspect oriented programming language like AspectJ.

References

- [1] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J. (1997). Aspect-oriented programming. Booktitle: ECOOP, SpringerVerlag, p. 220-240.
- [2] Aldawud, O., Elrad, T., Bader, A. (2001). A UML Profile for Aspect Oriented Modeling. Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA'01).
- [3] Stein, D., Hanenberg, S., Unland, R. (2002). Designing Aspect-Oriented Crosscutting in UML. 1st Workshop on Aspect-Oriented Modeling with UML (AOSD'02).
- [4] Pawlak, R., Duchien, L., Florin, G., Legond-Aubry, F., Seinturier, L., Martelli, L. (2002). A UML Notation for Aspect-Oriented Software Design. 1st Workshop on Aspect-Oriented Modeling with UML(AOSD'02).
- [5] Chavez, C. F. G., Lucena, C. J. P. (2003). A Theory of Aspects for Aspect-Oriented Software Development. 7th Brazilian Symposium on Software Engineering (SBES'2003).
- [6] Wagelaar, D. (2003). A Concept-Based Approach for Early Aspect Modeling. *In: Proc. of Early Aspects : Aspect-Oriented Requirements Engineering and Architecture Design*, Boston, Massachusetts
- [7] Gray, J., Bapty, T., Neema, S., Schmidt, D.C., Gokhale, A., Natarajan, B. (2003). An Approach for Supporting Aspect-oriented Domain Modeling. 2nd International Conference on Generative programming and component engineering, Vol. 48, p. 151 – 168.
- [8] France, R., Ray, I., Georg, G., Ghosh, S. (2004). Aspect-oriented Approach to Early Design Modeling. *IEE Proceedings Software*, 151(4), p. 173 – 185.
- [9] Sutton, S. M. Jr., Rouvellou, I. (2005). Concern Modeling for Aspect-Oriented Software Development, *Aspect-Oriented Software Development*. p. 479–505, Addison-Wesley.
- [10] Clarke, S., Baniassad, E. (2005). *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, Upper Saddle River.
- [11] Pawlak, R., Seinturier, L., Duchien L., Martelli, L., Legond-Aubry, F., Florin, G. (2005). Aspect-Oriented Software Development with Java Aspect Components. *Aspect-Oriented Software Development*. P. 343 – 369. Addison-Wesley, Boston.
- [12] Groher I., Bleicher S., Schwanninger C. (2005). Model-Driven Development for Pluggable Collaborations. 7th International Workshop on Aspect-Oriented Modeling (MODELS'05), Montego Bay, Jamaica.
- [13] Conejero J., Hernandez J., Rodriguez R. (2005). UML Profile Definition for Dealing with the Notification Aspect in Distributed Environments. 6th International Workshop on Aspect-Oriented Modeling (AOSD'05), Chicago, Illinois.
- [14] The AspectJ (TM) Programming Guide. (2005). <http://eclipse.org/aspectj/doc/released/progguide/index.html>, The AspectJ Team.
- [15] Carton, A., Driver, C., Jackson, A., Clarke, S. (2009). Model-Driven Theme/UML. *Transactions on Aspect-Oriented Software Development*. Vol. VI. p. 238-266, Springer.
- [16] Dahiya, D., Sachdeva, R. K. (2006). Approaches to Aspect Oriented Design: A Study. *ACM SIGSOFT Software Engineering Notes*. 31(5). P. 1-4.
- [17] Ballal, R., Hoffman, M. A. (2009). Extending UML for Aspect Oriented Software Modeling. *WRI World Congress on Computer Science and Information Engineering*. Vol. 7. p. 488 – 492.
- [18] Filman, R., Friedman, D. (2000). Aspect-oriented Programming is Quantification and Obliviousness. *Workshop on Advanced Separation of Concerns (OOPSLA 2000)*.

- [19] Lédeczi, Á., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G. (2001). Composing Domain- Specific Design Environments. *IEEE Computer*. p. 44-51.
- [20] Groher, I., Schulze, S. (2003). Generating Aspect Code from UML Models Workshop on Aspect-Oriented Modeling with UML. Aspect Oriented Software Design (AOSD 2003) Boston, USA.
- [21] Whittle, J., Jayaraman, P. (2008). MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation. Book Title: Models in Software Engineering, Springer-Verlag Berlin, Heidelberg. p. 16 – 27.
- [22] Molnár, Z., Balasubramanian, D. & Lédeczi, A. (2007). An introduction to the Generic Modeling Environment. *In: Proc. of TOOLS Europe. Workshop on Model-Driven Development Tool Implementers Forum.*
- [23] GME Tutorial. <http://w3.isis.vanderbilt.edu/Projects/gme/Tutorials/>
- [24] Narayanan, A., Karsai, G. (2008). Specifying the Correctness Properties of Model Transformations. Proc. of 3rd Int. workshop on Graph and model transformations (Int. Conf. on Software Engineering). p. 45-52.
- [25] Sarkar, A., Debnath, N. C. (2011). Graph Semantic Based Design of Aspect Oriented Software: A Conceptual Perspective. 26th Int. Conf. on Computers and Their Applications (CATA 2011). p. 78 – 83.
- [26] Rashid, A., Sawyer, P., Moreira, A., Araujo, J.(2002). Early Aspects: A model for Aspect-oriented Requirements Engineering. *IEEE Joint International Conference on Requirements Engineering*. p. 199 - 202.
- [27] Gao, S., Deng, Y., Yu, H., He, X., Beznosov, K., Cooper, K. (2004). Applying Aspect- Orientation in Designing Security Systems: A Case Study. *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, p. 360-365.
- [28] Wehrmeister, M.A., Freitas, E. P., Pereira, C. E., Wagner, F. R. (2007). 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. p. 428 – 432.