Security Metric Framework for the Software Architecture and Design Level

An Empirical Evaluation

Irshad Ahmad Mir, S.M.K Quadri

Abstract—***the field of security metric and security evaluation is multifaceted and multidimensional in nature, which needs great care and systematic approach to evaluate. The security evaluation is a continuous process that should be carried out throughout the different software development stages and also in the operational phases. In practice the secure software development is based upon the guidelines and rules for secure design and coding. Even if the secure software development process and guidelines are to be followed, the resultant level of security remains unknown to the development team. A security evaluation framework that can be applied at the early system development stages,the derived metrics that act as indicators of security level of the system and point out the most critical component of the system , in order to provide the basis for the system developers to take the design decisions regarding security is the foremost requirement of secure software development. In this study we haveproposed the extended security evaluation framework which strikes at the architectural and design phase of the software lifecycle, along with the empirical evaluation on a running system. In The proposed framework the mathematical modeling to derive the security metrics has been adopted. The empirical evaluation is carried out on a Finger Print Attendance Automation system (FASS) developed for the department of computer science UoK.***

Keywords:-*Software architecture, Security Metrics, Security Evaluation.*

I. Introduction

The main focus in the field of software security remained towards the application of protection mechanism after the system development and the security consideration during the development phase mainly dependent upon the secure design and coding guidelines. Lord Kelvin observed that if we can measure it we can improve it.

Information Security also comes under the same principle of measurement.Traditionally security is treated as an afterthought, in which the protection mechanisms employed after the development stages of the software [1]. Keeping in view the current vulnerable networked environment, security issues must be taken into consideration right from the early software development phases. If secure software development is to be followed, still the level of security remains unknown, which goes against the principle of Lord Kelvin i.e. " if you can't measure it then you can't improve it". Security evaluation framework that provide the quantitative or qualitative indicators of the security and point out the most critical element of the system in the early development phase can considerably help in sound decision making regarding the security of the system. The question now is where in the development phase the security evaluation is to be carried out? Since the design and the architecture of a system act as a blueprint of the overall system, so it becomes the best level for the evaluation security. In this study we have proposed an extension of our previously proposed security evaluation framework [1] for the component based software Architecture and Design (CBAD) and the derived metrics for the four main attributes of security, the *confidentiality, integrity , availability* and *dependency* from it. Our evaluation framework is based on component composition, dependencies among the components in the system composition and data/information flow across the components. We carried out the empirical evaluation of the proposed framework by reverse engineering process on a "Finger print Attendance Automation system" (FAAS) developed for the department of computer science, UoK. It is a web application developed in .NET framework 2008 using C# and Oracle as the backend database.

In section II we present an overview and terminology for software architecture and design and the selection architecture and design selection for the security evaluation. In Section III we present our security evaluation framework. Section IV presents the empirical evaluation of the proposed framework, finally the conclusion and references in section V and references.

II. Architecture and Design Selection

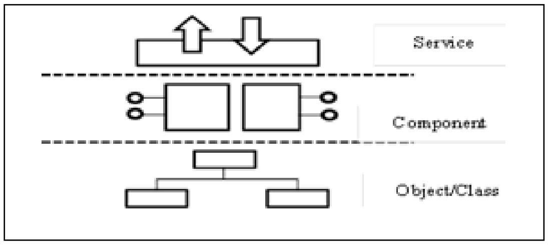
Since there are various software architectural and design modeling techniques present, such as Object oriented Analysis and design (OOAD), Enterprise Architecture Framework (EA), Service oriented Architecture and Design (SOAD) and Component Based Design(CBD) are the most common. The question is which of the architectural and design modeling technique to be followed for the specification of system structure so that it will be feasible for the analysis and

Irshad Ahamd Mir.
PhD. Research Scholar University of Kashmir
India.
irshad.mir@hotmail.com

S.M.K. Quadri.
Head Department of Computer Science University of Kashmir.
India.
quadrismk@hotmail.com

evaluation of quality attributes. Each of the modeling technique have their own merits and range of tools used for the specification of the architecture and design. Adopting one style over other is a very difficult task. Among the above mentioned architectural and designs Service Oriented Architecture (SOA), Component based development (CBD) and Object Oriented Analysis and Design (OOAD) are the core software architectural and design modeling techniques extensively adopted in software development. The SOA, CBD, and OOAD are not isolated from one another; instead they can be applied progressively to in the system development, each with different level of abstractions. Below figure (2) [2] shows the application architecture layers of the software architectures. As depicted in the figure (2), three level of technology layers are there for application architecture. At the higher level there is a service level which act as a great way to expose an external view of a system, with internal reuse and composition using traditional component design which may in turn use the object oriented design. The three layers (fig. 1) of application architecture are: Service Level, Component Level and Object/Class Level.

Figure 1. Application Architecture Layers



Choosing particular software architectural and design approach for the evaluation of quality attributes in general and for the evaluation of security in particular, is very difficult. On one extreme the SOAD is much coarse-grained in nature withe granularity focused on the loosely coupled services which are at the very high level of abstraction hiding the internal functional units of the service. Selection of SOAD of a system for the evaluation of security restricts the evaluation process from capturing the lower level details of the system. On the other extreme of Class/Object level provides the granularity at the class and object level which is very fine-grained view of a software architecture and design. Selection of Class/Object level architecture for the evaluation of security , no matter provides the lower level details about the design and architecture of the system but it makes the evaluation process too complex and bind the lower level details too early at the architectural and design phase of the system. The Component Layer in above diagram 4.3 is in between the two extremes. A service internally may exhibit the components composed together in a composition to provide the required service, these components may internally build by the lower level classes and objects. The position of Component Based Architecture and design in the architectural hierarchy are neither too coarse-grained like services that hide the internal functional units nor too coarse-grained to make the evaluation process too complex to bind the lower implementation level details. From the literature survey and our experience, component layer is the best suited for the evaluation process of any quality attribute of the system. Keeping in view the various factors, like granularity, abstraction, level of functionality exposed, flexibility provided and required efforts we have chosen Component Based Architecture and Design approach for the proposed Security Evaluation Framework.

A. *Component Based Architecture & Design*

Component Based Architecture and Design or Component Based Software Engineering (CBSE) is a successor of OOAD [3] and has been supported by commercial component frameworks such as Microsoft’s COM, Sun’s EJB and CORBA. In Component based Architecture and Design (CBAD) the fundamental unit of a large scale software construction is a component. The system in CBAD is structured as a collection of components and their interconnection and composition. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies [4].The abstract view of a component is shown in below figure 2. It consists of three main parts

- Component Name: The specified name of the component
- Code: The functionality of the component
- Interfaces: Both the required and provided interface to reveal the usage and functionality of the component in the system composition.

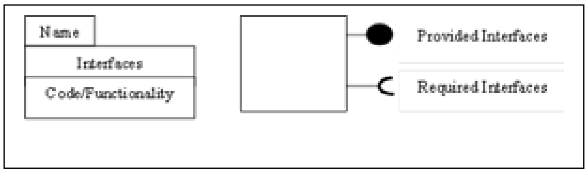


Figure 2. Software Component Model

B. *Component Composition*

In Component Based Software Development (CBD), a single component provides a unit of functionality and it may consume the services provided by other components and produce the output which may be consumed by other components in the system. Such interaction among components results in a system with components interlinked with one another in certain hierarchy which is also known as component composition. Composition is a central issue in Component Based Software Development (CBD). In CBD a component may be composed of several components and the entire system forms a component hierarchy. This notion of component composition allows developers to organize a software system in composition hierarchy. Since component provides a mean of reusability, it is fairly possible that an existing component from a component repository be fetched and plugged into a system. In a composition in which verities of components are involved, a composition language is desired. The composition language should have suitable semantics and syntax compatible with the components in the component model[4].There is no composition language in the most current component model. For instance Kola uses connecter as glue code for composition. Korba and UML 2.0 use the UML notations.

According to[5], component composition can take place during different stages of the component life cycle, like *design phase, deployment phase* and *runtime phase*:

Since the component composition depicts the overall structure of the system including interaction, cooperation and coordination. In our previous work [1], component composition of some of the well-known component modeling approaches have been presented.

III. Security Evaluation Framework: Architecture and Design Phase

In this section we present the extended security evaluation from work which is initially based up on our previously proposed security evaluation framework for component based software system [1]. The proposed security evaluation framework is based on the following factors

1. *Component Composition and Dependencies*
2. *Inter-Component Data/information and resource sharing.*

We have derived the security metrics for the main four attributes of security, the ***dependency,confidentiality, integrity*** and ***availability*** based on the above two factors.

A. *Component Composition and Dependencies*

A single component seems as atomic in nature. To provide the services to its client a component normally calls upon the service of other component. Such a scenario where a component calls upon the service of other components which in turn may call upon the service of other components and so forth, have to be inter-linked together in certain order to properly and efficiently provide the required functionality to its clients is known is composition of the system or component composition. With the advent of a networked environment the composition of such components may be either local (bound to local server) or remote (on multiple servers). Similar to the object-oriented systems, in which the object is the basic building block, in CBSs, component is the basic, but usually a black box building block. As new component gets plugged into the composition, it has the effect on that part or overall system. The newly added component in the composition may refer to other components and can be used by others in the composition. The compositions of the components certainly incur the dependencies both the direct and indirect (via intermediate components) among the components.

The composition of components results in cooperation, coordination, and interaction among them which in turn results in the dependencies among them in order to provide the complex system functionality. At the top there exist two types of dependencies which are further four types of dependencies.

- *Direct Dependency* : involves a direct association between two components
- *Indirect Dependency*: involves the association between component through intermediate components

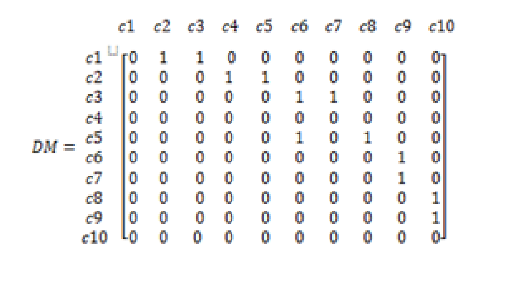
According to[6], there are at least four types of dependencies which are *explicit direct dependency*, *explicit indirect dependency*, *implicit direct dependency* and *implicit direct dependencies.* These dependencies show the nature of dependencies which are either direct or indirect and also implicit or explicit. Based upon the functionality and the business logic there are several dependencies that get incurred into the system among the components [7].

A system composed of several components (functional units) can be represented in graphical form using a particular modeling approach. In our case we have adopted the UML based component modeling, due to the diversity of tools present in it. In order to simplify the process we take into an account an illustrative exampleof a system composed of multiple component for illustrative purpose shown in figure (4)below using Visual paradigm for UML 9.0. As depicted in figure (4), beside the implicit dependencies that occurs by the interconnection of provided and required interface (includes both data and interface dependencies) for the provided and required functionalities, the dependencies are also specified explicitly by dashed arrow lines. The direction of the line shows that the source and the destination of a dependency.

We depict the dependencies among the components into an adjacency matrix (AM) representation**[7]**.The Components are organized as rows and columns with index i *and* j respectively. If a component C_i in row i is dependent on other component C_j in row j then the corresponding element in the adjacency matrix (AM) is marked as 1 , otherwise it is0. In general the values for each of the element of adjacency matrix $DM_{(n*n)} = dij_{(n*n)}$. Where:

$$dij = \begin{cases} 1, & if\ ci \to cj \\ 0, & otherwise \end{cases} \quad --- \quad (1)$$

From The above equation 1 the dependency matrix (DM) for the system composition figure (3) is



$$DM = \begin{array}{c|cccccccccc} & c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 & c10 \\ \hline c1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c2 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ c3 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ c4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ c6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ c7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ c8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ c9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ c10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 3. Direct Dependency Matrix

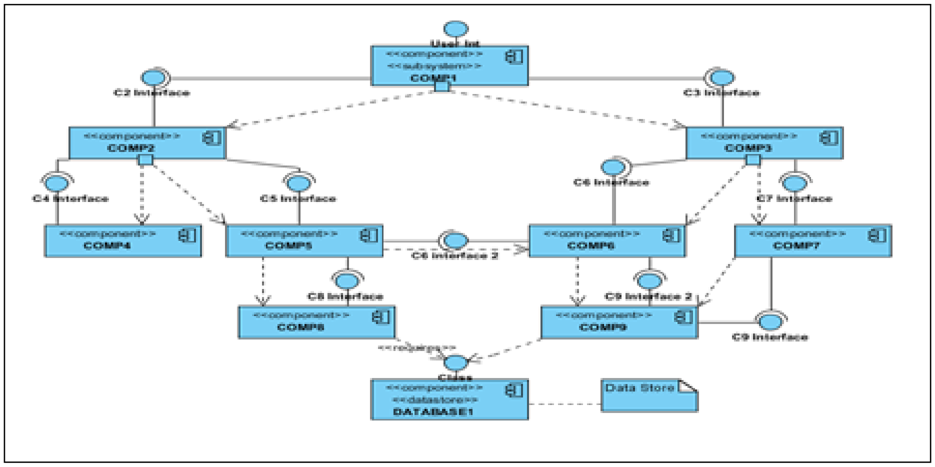
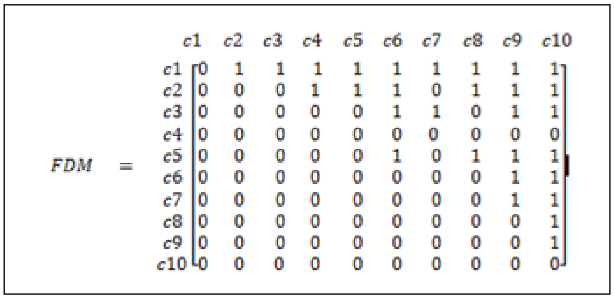


Figure 4. An Illustration of system's Component Composition and Dependencies

The matrix above figure (4) depicts the direct dependency matrix i.e. it depicts the direct association among the components in the composition. In order to calculate all the indirect possible dependencies among the components (direct as well as indirect) we apply the Warshall's algorithm of transitive closer[8] to compute the Full Dependency matrix (FDM)of the illustrative system composition of figure (4) and the resultant matrix is shown in below figure (5).



$$FDM = \begin{array}{c|cccccccccc} & c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 & c10 \\ \hline c1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ c2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ c3 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ c4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ c6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ c7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ c8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ c9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ c10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 5. Full Dependency Matrix (FDM)

The Full Dependency matrix in above figure (5) shows all possible dependencies (direct as well as indirect dependencies of each of the component of the system depicted in figure (4).

Beside the dependencies types of presented earlier we define two more types of dependencies associated with a component in the composition, the ***In-Dependency*** and ***Out-Dependency***.

- ***In-Dependency:*** In-Dependency of a component (C_i) is defined as the other components in the composition that are directly or indirectly dependent up on the component(C_i).
- ***Out-Dependency:*** Out-Dependency of a component $(C_i)$is defined as the other components in the composition up on which component (C_i) depends for the functionality.

We further define two more terms related to the *In-Dependency* and *Out-dependency* of a component which are:

- ***Degree-In***: denoted by $\boldsymbol{DegIN(C_j)}$ is defined as the number of component in the *In-Dependency* of the component (C_j) . Degree-In of a Component can be easily calculated by counting the number of $1's$ in the corresponding column j of the Full Dependency Matrix(FDM). Mathematically:
- ***Degree-Out:*** denoted by $\boldsymbol{DegOUT(C_i)}$of component (C_i) is defined as the number of other components in the system composition the (C_i) is dependent upon. Degree-Out of a component can be easily calculated as the number of $1's$ in the corresponding row i of the Full Dependency Matrix (FDM).

Mathematically:

$$degIN(C_i) = \sum_{j=1}^{n} (FBD_{ji}) - - - - - - (2)$$

And

$$degOUT(C_i) = \sum_{j=1}^{n} (FBD_{ij}) - - - - (3)$$

The total degree of dependencies denoted $degTOT(C_i)$ of a component C_i can be calculated as

$$degTOT(C_i) = degIN(C_i) + degOUT(C_i) - - - (4)$$

Where$degIN(C_i) + degOUT(C_i) \geq 1$.

In order to keep the result on the similar scale (0,1) the $depTOT(C_i)$ is

$$degToT(Ci) = \frac{1}{degIN(C_1) + degOUT(C_i)} - - - - (5)$$

Where$degIN(C_i) + degOUT(C_i) \geq 1$.

The aggregated level of dependencies of the overall system denoted by $Dep\ (Sys)$ is:

$$Dep(Sys) = \frac{\sum_{i=1}^{N} degTOT(C_i)}{N} \quad - - - (6)$$

Where N is the total number of components in the system. Based upon the dependencies (direct as well as indirect) we derive and propose the security matric for the ***availability*** for each of the component and then for overall system.

B. *Availability*

Availability ensures that the service provided by software, its component or network should remain available to its clients in timely manner. In the current networked environment the software architectures are now shifting from simple standalone application to large distributed architectures based on OSI or J2EE n-tiers. In case where the composition of components is remote the provided functionalities of the components are accessed by remote procedure call (RPC) which requires the invocation, marshaling and unmarshallingg of the parameters. In a functional dependency graph of component, one component may invoke a call and wait for the provided services of second component which in turn requires the functionality of third component and so forth. Such a chain of functional dependencies among components will certainly results in delayed response time at each hop of the dependency chain. The delay involved at each hop of the dependency chain is due following factors:

- ***Processing Delay:*** Time taken by a component to successfully process the request of its clients and return the result from the invocation to the end result returned.
- ***Transmission Delay:*** In case the composition of components is remote then the transmission delay is the transmission time alone excluding the processing delay.

The delayed response at each of the component can certainly affect the *availability* (one of the main attribute of security) of the system. Software developers must need to know preferably quantitatively the possible delay incurred at each of the components in the system composition and the aggregated level of availability level of overall system.

C. *Availability Metric*

As mentioned previously a component can act as a hub in which it handles the request from one group of components for the required functionality and may in turn call up on the provided services of other group of components on their behalf .This process can form a chain which results in a system with a delayed response time especially in distributed and multiuser environment which affects the level of availability of the system. The measurement of availability results in to find the availability critical components, so that the alternative corrective measures can be applied. In this section we derive the availability metric for a component (C_i) by taking into account the following factors.

- $DegIN(C_j)$
- $DegOUT(C_i)$
- Processing Delay P
- Transmission delay T

We put forward the above theoretical concept of availability metric into mathematical form. There may be a $1 - N$ relationship between each of the component in $\boldsymbol{DegIN}(\boldsymbol{C_i})$ and $\boldsymbol{DegOUT}(\boldsymbol{C_i})$, i.e. for each of the component in $\boldsymbol{DegIN}(\boldsymbol{C_i})$, $\boldsymbol{C_i}$ may call some or all of the components including $\boldsymbol{C_i}$ itself for required services on of behalf of invoking components . Based on these factors we propose the availability matrix of a component $\boldsymbol{C_i}$ denoted by $\boldsymbol{A}(\boldsymbol{C_i})$ as:

$$A(Ci) = \frac{1}{DegIN(Ci) + \sum_{j=1}^{DegIN(Ci)} DegOUT(Ci) + \sum_{j=1}^{DegOUT(Ci)} Pj + Tj} \quad - - - - (7)$$

Where

$\boldsymbol{DegIN}(\boldsymbol{C_i})$is the in-degree of component $\boldsymbol{C_i}$ and $\boldsymbol{DegIN}(\boldsymbol{C_i}) \geq \mathbf{1}$

$\boldsymbol{DegOUT}(\boldsymbol{C_i})$ is the out-degree of component $\boldsymbol{C_i}$ and $\boldsymbol{DegOUT}(\boldsymbol{C_i}) \geq \mathbf{1}$

$\boldsymbol{P_j}$ is the processing delay of the j^{th} component in $\boldsymbol{DegOUT}(\boldsymbol{C_i})$.

$\boldsymbol{T_j}$ is the transmission delay involved of j^{th} component in $\boldsymbol{DegOUT}(\boldsymbol{C_i})$

The proposed *availability* metric provides the software developer the early indicator about the level of availability of each of the component and can easily identify the *availability critical* component of the system and enable to provide the necessary corrective or preventive measures accordingly. The range of output values of the above availability matric in equation (4 and 5) will remain in a range of (0,1). More the value tends towards 0 on the scale higher the effect on the availability of the component. The aggregated *availability* of the overall system denoted $\boldsymbol{A}(\boldsymbol{Sys})$can be computed as:

$$A(Sys) = \frac{\sum_{i=1}^{N} A(Ci)}{N} \quad - - - - (8)$$

Where:

$\boldsymbol{N}$ is the total number of components in the system.

The so proposed *availability* metrics is for the design and architectural phase of the software development lifecycle but it can also be applicable to the already developed systems by performing reverse engineering to get to the design and architecture of the system.

D. *Inter-Component Data/Information Flow.*

The dependencies among the component (data dependency, functional dependency, interface dependency, etc.) results in the flow of data/information across the components. Such a flow of data/information must be analyzed for the secure system operations. In this and the net subsequent section we look at the data/ information flow among the components and derive the metrics for the two fundamental security attribute , the *confidentiality* and *integrity* for each of the component and in the composition and then for the overall system.

The flow of the data/information takes places through the component interface (provided and required). These interfaces can be defined as the component access point[9], enable the clients to access the provided functionality of the components. A component normally has multiple access points for the different functions provided in the interface [10]. Since in component Based Software Development (CBD) each component is a separate entity designed with varying level of protection then plugged together to provide the overall system functionality. It becomes the most critical to analyze the flow of data/information and resource sharing among the components. Flow of data/information is characterized by two types of flows, Inter-component flow and Intra-component flow [11]:

– *Inter-Component Flow:* The exchange of data/information (both *In-flow* and *Out-flow*) across the components through provided and required interfaces. A component provides information to its clients (a user, a required interface or an engineering device) by an *Out-flow* through its provided interfaces through a list of *Out-parameters* and receives data/information from the client by an In-flow through its required interfaces by a set of *In-parameters*. Since only the interfaces are visible to the clients of a component, such an exchange of data/information takes place across the components interface boundary.

– *Intra-Component Flow:* As a result of *Inter-Component* data/information flow, provided and required interfaces of a component pass /receive the data to/from component body. For instance, in case of an In-flow in Inter-component data/information flow the required interfaces of the interface pass the data/information to the component body for further processing or to update a backend data source or a data structure. So in case of Intra-component information and data/flow boundary is confined to the component body.

Below figure (6) shows the both types of data/information flow of a component.

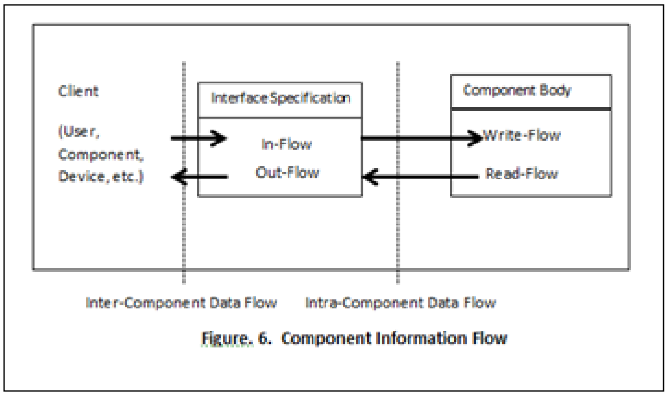


Figure 6. Component Information Flow

Like dependencies, the data/information flow among the components can be either *direct-flow* or *indirect-flow.*

- *Direct Data/Information-flow:* In the direct information flow the flow of information to/from between components occur directly without passing through an intermediate node or component. For instance if a component C_1 invokes other component C_2 and passes the information to it or C_2 returns something back to C_1 is known as direct data/information flow
- *Indirect Data/Information-flow:* In the indirect flow of data/information the flow takes place indirectly through intermediate nodes or components. As an example if a component C_1 invokes component C_2 which in turn invokes C_3 and passes the data/information passed by C_1 to C_3 or C_3 returns something to C_1 via C_2 is known as indirect-data/information flow.

Software development team must need a concrete way to analyze the flow of data/information (both direct and indirect) across the components in a system. UML component modeling provides a variety of tools and notations for the specification of components, components interaction, and interface specifications in order to design a system. The so design of the system using UML modeling, the flow of data / information can easily be analyzed.

Each component in the system composition possesses certain resource such as data database access, files, data/ information. Components in the composition can access the resources of other components with certain privileges.

As the design and architecture of a system evolve it becomes complex and cumbersome to keep track of the flow of data/information across the components. Such a scenario results in security threat to the system especially the two main concerns of the security of any system, the ***confidentiality*** and ***integrity***. In the next subsequent sections we propose the metrics for the confidentiality and integrity of a component and then for the overall system based on the following parameters.

- Component dependencies
- Data/Information flow.
- Component interfaces.

E. Confidentiality Metric

Confidentiality of any system, resource or a network ensures that unauthorized disclosure of data/information should not occur. In the large complex software systems (comprised of multiple components both local and remote) the flow of data/information across the components becomes a critical factor for the security of the overall system. Even if a system builds with the protection mechanisms in place, the data/information breach at one of the component can cause the serious problem to the overall system. The derived metric aim to analyze and provide the quantative indicators about the level of *confidentiality* of each of the component and further for the overall system.

As mentioned earlier the inter-component data/information takes placed through the provided and required interfaces of a component. For the simplicity we redefine these two types of interfaces as:

- *Write-Interfaces:* denoted by I_w , are the required interfaces of a component through which the *In-flow* of data/information takes place.
- *Read-Interfaces:* denoted by I_r , are the provided interfaces through which the *Out-flow* of information takes places.

The confidentiality is likely to be affected through a component when both the read and write operations by different components in the composition takes place. The idea is to identify the level of such operations on a component and accordingly quantify the resultant indicators. With respect to our confidentiality metric we put forward the following argument.

- *Argument 1:* The confidentiality of a component is likely to be affected more as the number of reading components in the system composition and dependency (direct/indirect) with respect to the writing components. For instance a component C having n number of reading components (the components that are functionally dependent on the provided services of C and m number of writing components (the components in the system composition whose services are required by C). As the n increases for each of the component in m the confidentiality of is affected more, i.e. each of component in n is likely to have read access to component C and all the components in the dependency (direct/indirect) that are capable of writing to .

The specified argument doesn't mean that that the effect of writing components m is completely undesirable. If there is no writing component the confidentiality will not be affected at all, instead the argument states that an increase in n (reading components) has relatively higher impact on the confidentiality than an increase in m (writing components). Also as specified earlier a component may have multiple interfaces (both provided and required) which we defined earlier as *Write-Interface* (I_w) and *Read-Interface*(I_r) through which the *in-flow* and *out-flow* of data/information takes place. Because these interfaces are the ports of a component for the flow of data/information across the component, so the number of these ports is also the candidate for the overall confidentiality level of a component.

In order to derive the *confidentiality* metric for a component $\boldsymbol{C}$we have following parameters at hand.

- From equation (3), number of components that can likely make an *In-flow* (write operation) directly or indirectly on $\boldsymbol{C}$is$\boldsymbol{DegOUT}(\boldsymbol{C}\ \ )$.
- From above equation (2), number of components that can likely responsible for the *Out-flow* (read operation) data/information directly or indirectly from $\boldsymbol{C}$is$\boldsymbol{DegIN}(\boldsymbol{C}\ \ )$.
- Possible number of read-interfaces (provided interfaces) through which the *Out-flow* (read operation) can take place on $\boldsymbol{C}$is$\boldsymbol{I_r}$.
- Possible number of write-interface (required interfaces) through which the *In-flow* (write operation) can takes place on $\boldsymbol{C}$ is$\boldsymbol{I_w}$.

Keeping in view the above argument (1), that in both the read and write operation the confidentiality will be affected more as the number of reading components ($\boldsymbol{DegIN}(\boldsymbol{C}\ \ )$)for each of the writing component ($\boldsymbol{DegOUT}(\boldsymbol{C}\ \ )$)the confidentiality metric is :

$$COD(C_i) = \frac{1}{(DegIN(C_i)^2 * I_r)/\ DegOUT(C_i)\ + I_w} -- \\ -(9)$$

Where $\boldsymbol{COD}(\boldsymbol{C_i})$is the confidentiality of component $\boldsymbol{C_i}$.

$\boldsymbol{DegIN}(\boldsymbol{C_i}\,)$is the degree of in-dependency (number of components that directly or indirectly read from) component $\boldsymbol{C_i}$and $\boldsymbol{DegIN}(\boldsymbol{C_i}\,) \geq \boldsymbol{1}$

$\boldsymbol{DegOUT}(\boldsymbol{C_i})$is the degree of out-dependency (number of components that directly or indirectly write to) component $\boldsymbol{C_i}$and $\boldsymbol{DegOUT}(\boldsymbol{C_i}) \geq \boldsymbol{1}$

$\boldsymbol{I_r}$and$\boldsymbol{I_w}$are the read and write (provided and required) interfaces of component $\boldsymbol{C_i}$and $\boldsymbol{I_r}$,$\boldsymbol{I_w} \geq \boldsymbol{1}$.

The so derived confidentiality metric, equation (8) provides the quantative indicator about the criticality of each of the component with respect to the confidentiality. As stated earlier the numbers of reading components have the higher impact on the confidentiality of the component with respect to the writing components. In the proposed confidentiality metric such a scenario is taken into the consideration by squaring the $\boldsymbol{DegIN}(\boldsymbol{C_i}\,)$.

The aggregated confidentiality metric for of the overall system is

$$COD(S) = \frac{\sum_{i=1}^{N} COD(Ci)}{N} --- (10)$$

Where, $\boldsymbol{N}$is the total number of components in the composition.

F. *Integrity Metric*

Integrity in the third main pillar of the security of any software, network, or any other system. The main objective of the *integrity* is to ensure that unauthorized modification to the data/information and information processing resource should not take place. As with *confidentiality*, as the architecture and design of the system evolves from the scratch to the full-fledged design, it becomes very difficult to keep track of the information flow and modification to the information and the resources. System developers need some tools that can be applied at the early design and after the design phase of the system development to identify the most critical elements of the system and preferably the quantative indicators of the level of availability of each of the component and for the whole system in order to make necessary decisions and adjustment early to reduce the cost and efforts needed in further system life cycle. The integrity of a component C is likely to be affected when multiple component in the composition perform a write operation (*In-flow* of data/information) on C though the *write-Interfaces* (I_w) *(Required interfaces) of* a component. For instance a component C may having certain resources say a backend database or a file, other components can call upon the services of C and pass the data through the *Write-interfaces* (I_w) to update the data/base and file. A weaker component in the system composition can be attacked to violate the integrity of the system. Bothe software developer and the user must need to know the potential risk associated with each of the component. Taking into account all the parameters we propose the *integrity* metric of a component as:

Let $\boldsymbol{C}$be a component whose integrity level is to be evaluated.

The possible number of components in the system composition that are responsible for the *In-flow* (write) of data/information to $\boldsymbol{C}$ would be in the *Degree-Out*of **.** As mentioned earlier degree-out of a component $\boldsymbol{C}$ is the number of components in the system composition on which $\boldsymbol{C}$depends for their provided services. These components are bound to $\boldsymbol{C}$ through its Write-Interfaces (I_w) or simply the required interfaces. Mathematically if $\boldsymbol{m}$be the number of components that are responsible for the *In-flow* data/information then:

$$m \ \in DegOUT(C) - - - - - \quad (11)$$

Note that as from the above equation (3) $\boldsymbol{DegOUT(C)}$represents all the possible components on which $\boldsymbol{C}$ depends (directly or indirectly) for the provided services and that can perform a write-operation on $\boldsymbol{C}$directly or indirectly.

Also the number of ports or *Read-Interfaces* through which the In-flow of data and information can take place is$\boldsymbol{I_w}$**.** The complete integrity metric for a component $\boldsymbol{C_i}$is:

$$INT(Ci) = \frac{1}{Iw * degOUT(Ci)} - - - (12)$$

Where $\boldsymbol{I_w}$ are the write-Interfaces of component $\boldsymbol{C_i}$and $\boldsymbol{I_w \geq 1}$

$\boldsymbol{DegOUT(C_i)}$is the out-dependency of $\boldsymbol{C_i}$ and $\boldsymbol{DegOUT(C_i) \geq 1}$

The so proposed metric in equation (12) provides the early indicator of the level of integrity of each of the system. As with the earlier equations, the resultant value lie in between (0,1) with the lower values on the scale the higher chance of integrity breach. The aggregated *integrity* metric for the overall system $\boldsymbol{S}$ is

$$INT(S) = \frac{\sum_{i=1}^{N} INT(Ci)}{N} - - - (13)$$

Where $\boldsymbol{N}$ is the total number of components in the system composition.

IV. Empirical Evaluation

In section (III) we have proposed a security evaluation framework and derived the metrics and derived the metrics for the four main attributes of the security, *Dependency, Confidentiality, Integrity* and *Availability.* In order to analyze the applicability and to check the feasibility of the proposed frameworks and metrics, in this section we perform an empirical evaluation of the proposed framework. The three most common methods used in empirical evaluation are, experiments, case studies and surveys [12]. Our empirical evaluation falls under first category, the experimental approach to present the process of applying the proposed security evaluation framework.

A. *Data Collection*

The empirical evaluation is carried out on a running system "Fingerprint Attendance Automation System "(**FAAS)** developed by the author, for the department of computer science UoK. Below figure (9) depicts the component based Architecture and Design of FASS, The main features of the system are:

- It is a web application that can be accessed over the network.
- Having a backend database for the storage of data from various components.
- Provides facility for the automated attendance through finger print scanner attached to the server through in the LAN. The output from the device gets stored to the back end database. And also the information from the system flows out to the Fingerprint scanner.
- Beside the general attendance through Fingerprint scanner, it provides an interactive interface to faculty for individual class attendance for each of the subject taught.
- Provides automated shortage generation.
- Account management for administrator faculty and users with necessary privileges

We have performed reverse engineering process to get to the design and architectural level of the system, and captured the component based design and architecture of the system in figure (9). The reverse engineering is carried out using tools provided with the .NET framework to collect the various classes and objects , their relations and dependencies with each other in the system. Further the component level design and architecture of the system is prepared using UML 2.0 modeling in "Visual Paradigm for UML 9.0" design tool. In empirical evaluation we focused less on the specification of the functionality of the system and collected the data required for the evaluation which is.

- *In-Dependency*: denoted by $\boldsymbol{DegIN(C_i)}$ in equation (2) for each of the component of FAAS in figure (8).
- *Out-Dependency:* denoted by $\boldsymbol{DegOUT(C_i)}$in equation(3) for each of the component FAAS in figure (8)

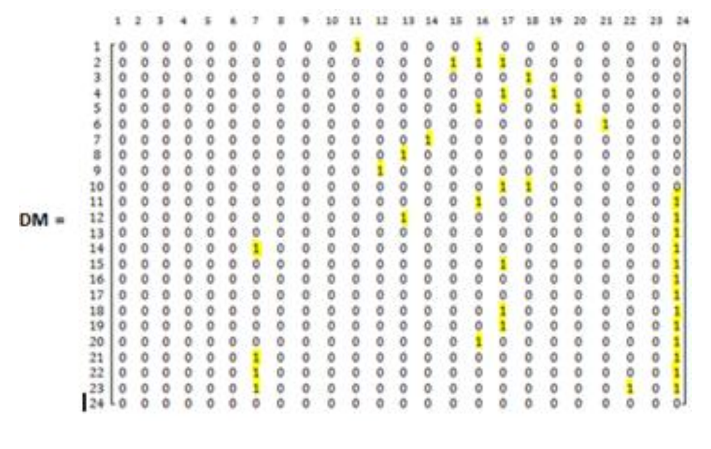
- *Read-Interfaces:* denoted by $\boldsymbol{I_r}$, the number of the reading (provided) interfaces for each of the component of FAAS in figure (8).
- *Write-Interfaces:* denoted by $\boldsymbol{I_w}$, the number of the writing (required) interfaces for each of the component of FAAS in figure (8) .
-

In data collection table (1) the components $\boldsymbol{C_i}$, where $i = 1\ to\ 6\ and\ 8\ to10$ are the components through which the users interacts with the system and their names are suffixed by "INT". From the FDM figure (10) the $\boldsymbol{DegIN}$ of these components is 0 . Because the users depend up on the provided services of these components so we change their $\boldsymbol{DegIN}$ **form** 0 to1. In general any component having $(\boldsymbol{DegIN}(\boldsymbol{C_i}),\boldsymbol{DegOUT}(\boldsymbol{C_i}),\boldsymbol{I_r},\ \text{and}\boldsymbol{I_w}) = \boldsymbol{0}$, then we set it to 1 in order to eliminate any divide by 0 error .

As depicted in the figure (9) there are 24 fine grained components in the system composition. Beside the implicit interface dependencies among the components, the dependencies (functional,control and interface) are also specified explicitly by dashed arrow lines with the direction of arrow specifies the source and destination of a particular dependency. The components of the system are numbered from1 to 24 as shown in below data collection table (1). From equation (1) the direct dependencies matrix DM of the system architecture and design (FASS) depicted in below figure (9) is shown in figure (7). From the proposed framework (section (III)), the full dependency matrix (FDM) of the direct dependency matrix (DM) is calculated and depicted in below figure (8).

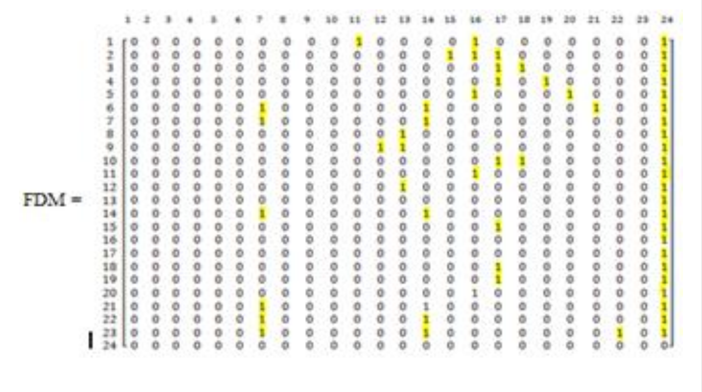
Applying the derived metrics of the security evaluation framework of above section (III), we calculate the level of security with respect to the following attributes for each of the component in the system composition.

- *Dependency*: Applying equation (5) we compute the dependency level of each of the component.
- *Availability*: Applying the derived metric for the availability equation (7), we have computed the level of availability of each of the component. In case of availability, since the system is a web application operating on the local LAN, we have calculated the processing and transmission delay together of each of the component through a Remote Procedure Call (RPC) and calculated the time elapsed from the invocation of a RPC request till the result returned. The elapsed time for each of the component is shown in below data collection table (1).The components from 1-9 below table (1) are responsible for user interaction, to act as an interaction between user and the rest of the system; there transmission and processing delay is similar.
- *Confidentiality:* Applying the derived metric for the confidentiality, equation (9) we computed the level of confidentiality of each of the component.
- *Integrity:* Applying the derived metric for the integrity, equation (12) we computed the level of availability of each of the component



$$DM = \begin{array}{r|cccccccccccccccccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
14 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
21 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
22 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
23 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
24 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 7. Direct Dependency Matrix of FASS



$$FDM = \begin{array}{r|cccccccccccccccccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
14 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
21 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
22 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
23 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
24 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 8. Full Dependency Matrix of FAAS.

The result of the above computed attributes for each of the component is presented in table (2). As mentioned earlier the range for output values is (0, 1). Lower the values on the scale higher the effect on the component.

In order to calculate the security posture with respect to the *Dependency*, *Availability*, *Confidentiality* and *Integrity* of the overall system, we apply the following derived metrics.

B. *Analysis*

The result of empirical evaluation of "FAAS" is depicted in below result table (2) . The resultant values marked as red are those components that are most critical with respect to the specified security attributes. We have analyzed the resultant values with the system architecture and design and it showed us the positive response that these components are actually the most critical in the system architecture and design and the malfunctioning of these components will affect the overall system functionality. The values marked as green are the next critical for the system with respect to specified security attributes. Comparing the result of empirical evaluation with the architecture and design of the system is according to our expectation if we look at the design and architecture of the system and also the experience with the system in the running environment, which shows the effectiveness of the proposed framework.

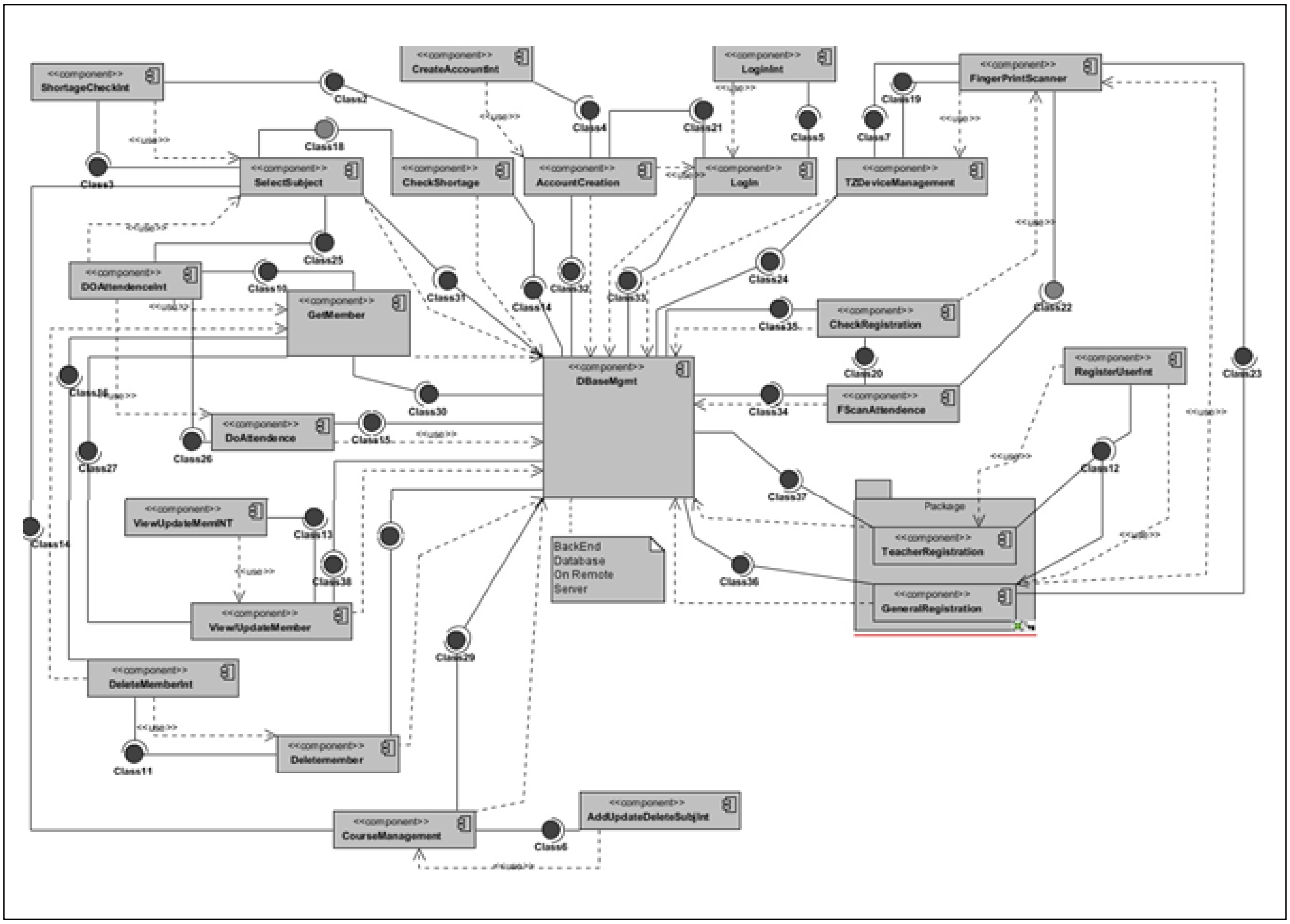


Figure 9. Component Based Architecture & Design of Finger Print Attendance Automation System (FASS)

Table 1. Data Collection Table

C_i	Component Name	Direct In-Dep.	Direct Out-Dep	$degIN(C_i)$	$degOUT(C_i)$	(I_r)	(I_w)	Delay (P+T)
1	ShortageCheckINT	1	3	1	3	2	1	0.11
2	DoAttendanceINT	1	4	1	4	2	2	0.11
3	ViewUpdateMemINT	1	1	1	3	3	4	0.11
4	DeleteMemINT	1	3	1	3	2	3	0.11
5	AddUpdateDelSubINT	1	3	1	3	4	2	0.11
6	RegisterUserINT	1	2	1	4	2	3	0.11
7	FingerPrintScanner	5	2	6	3	2	3	0.30
8	LoginINT	1	2	1	2	1	1	0.11
9	CreateAccountINT	1	2	1	3	2	2	0.11
10	ViewUpdateMemINT	1	3	1	3	2	2	0.11
11	CheckShortage	2	3	1	2	2	1	0.17
12	AccountCreation	2	3	1	2	2	1	0.21
13	Login	3	2	3	1	2	1	0.23
14	TZDeviceManagement	2	3	6	3	4	4	0.33
15	DoAttendance	2	3	1	2	3	1	0.17
16	SelectSubject	6	2	5	1	2	2	0.15

17	GetMember	7	2	7	1	1	2	0.15
18	ViewUpdateMember	3	3	2	2	2	2	0.20
19	DeleteMember	2	3	1	2	2	2	0.21
20	CourseManagement	2	3	1	2	3	2	0.23
21	Registration	2	3	1	3	3	2	0.19
22	CheckRegistration	2	3	1	3	2	2	0.15
23	FscanAttendance	1	4	1	4	2	1	0.35
24	DBaseMgmt	14	1	24	1	3	5	0.43

Table 2. Individual Components Security Indicators

C. No	Component Name	Dependency	Availability	Confidentiality	Integrity
1	ShortageCheckINT	0.25	0.210526316	0.6	0.333333
2	DoAttendanceINT	0.2	0.169491525	0.4	0.125
3	ViewUpdateMemINT	0.25	0.209205021	0.2	0.083333
4	DeleteMemINT	0.25	0.208768267	0.272727273	0.111111
5	AddUpdateDelSubINT	0.25	0.207900208	0.3	0.166666
6	RegisterUserINT	0.2	0.16	0.285714286	0.083333
7	FingerPrintScanner	0.111111111	0.03990423	0.076923077	0.333333
8	LoginINT	0.333333333	0.273224044	0.333333333	0.25
9	CreateAccountINT	0.25	0.205338809	0.375	0.166666
10	ViewUpdateMemINT	0.25	0.209205021	0.272727273	0.111111
11	CheckShortage	0.333333333	0.279329609	0.5	0.5
12	AccountCreation	0.333333333	0.273224044	0.333333333	0.25
13	Login	0.25	0.155520995	0.052631579	1
14	TZDeviceManagement	0.111111111	0.03990423	0.019230769	0.083333
15	DoAttendance	0.333333333	0.279329609	0.4	0.5
16	SelectSubject	0.166666667	0.095877277	0.019230769	0.5
17	GetMember	0.125	0.069300069	0.019607843	0.5
18	ViewUpdateMember	0.25	0.151975684	0.166666667	0.25
19	DeleteMember	0.333333333	0.279329609	0.333333333	0.25
20	CourseManagement	0.333333333	0.279329609	0.285714286	0.25
21	Registration	0.25	0.197628458	0.333333333	0.166666
22	CheckRegistration	0.25	0.197628458	0.375	0.166666
23	FscanAttendance	0.2	0.161030596	0.666666667	0.25
24	DBaseMgmt	0.04	0.020648358	0.000577034	0.2

Table 3. Overall FASS Security Indicators

Security Attributes	Dependency	Availability	Confidentiality	Integrity
Overall Security Indicators	0.2355	0.1822	0.2759	0.2762

V. Conclusion

The main aim of the purposed security evaluation framework and derived metrics is to provide the early indicators of the security especially at the design and architectural phase of system life cycle. Such indicators enables the developers to take necessary decisions and to provide the necessary protection mechanisms if required. The proposed framework is flexible enough to be applied on existing developed system, which requires a reverse engineering process to get to the design and architecture of the system. In future we are looking forward to the formalism of the proposed framework and to include the other security attributes into consideration and to carry out a large scale experiment in order to check the feasibility of the framework.

References

[1]. Mir, Irshad Ahmad, and S. M. K. Quadri. "Analysis and Evaluating Security of Component-Based Software Development: A Security Metrics Framework." *International Journal of Computer Network and Information Security* (IJCNIS) 4.11 (2012): 21.

[2]. Brown, Alan, Simon Johnston, and Kevin Kelly. "Using service-oriented architecture and component-based development to build web service applications." *Rational Software Corporation* (2002).

[3]. C. Szyperski. Component Software: Beyond Object-Oriented Programming. *Addison-Wesley*, 1998.

[4]. Kung-Kiu Lau and Zheng Wang. Software component models. *IEEE Transactions on Software Engineering*, 33(10), October 2007, pp. 709-724.

[5]. B. Christiansson, Christiansson, Benneth, Lars Jakobsson, and IvicaCrnkovic. "CBD Processs." (2002).

[6]. Here we have some ideas similar to the idea in this draft, the difference is that we will construct dependency matrix, they construct component coupling graph. http://*www.cis.ohiostate.du/weide/sce/miscellaneous/CCDs.pdf*

[7]. B. Li, Managing Dependencies in Component-Based Systems Based on Matrix Model‖ *Proc. of Net.ObjectDays Conf.*, pp.22-25, 2003.

[8]. Rosen, Kenneth H., Discrete Mathematics and its Applications, Third Edition, *McGraw-Hill, Inc*, 1994.

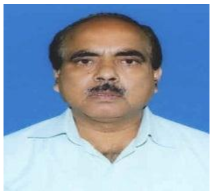
[9]. I. Crnkovic, B. Hnich, T. Jonsson and Z. Kiziltan, "Specification, implementation, and deployment of components," *Communication of ACM*, vol.45, 2002, pp. 35-40.

[10]. C. Szyperski, Component Software: Beyond Object Oriented Programming, Second Editioned, *Addison Wesley*, New York, 2002.

[11]. M. Abdellatief , Component-Based Software System Dependency Metrics based on Component Information Flow Measurement‖, *The Sixth International Conference on Software Engineering Advances*, ISBN: 978-1-61208-165-6 ICSEA 2011.1

[12]. Blom, Martin. Empirical Evaluations of Semantic Aspects in Software Development. *Diss. Karlstad University*, 2006.

Irshad Ahmad Mir is a PhD Research scholar in the department of Computer Sciences university of Kashmir, India. He has completed his bachelors (B.C.A) and masters (M.C.A) degrees in computer applications from the University of Kashmir.



Prof. S.M.K Quadri is presently the Head department of Computer Sciences University of Kashmir. He got his M.Tech from the Indian School of MinesDhanbad, and PhD from University of Kashmir, India

> "Measurement is the first step that leads to control and eventually to improvement. If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it."