Chapter 4

Component level Availability Metric: Design and Evaluation

4.1 Introduction

The traditional way of dealing with security was to employ the protection mechanisms after the developmental stages of an Information System. [114]. As a result, most of the research work in *Information* and *Computer/Network Security* is based on the detailed study of complex protocols or of complex systems and also given the fact that the genesis of the security holes is often backtracked to failures associated with such complex protocols and complex systems. In the last decade or so the security paradigm has shifted beyond the study of complex protocols, to the level were secure systems can be designed and evaluated in a connected and chronological order (evaluations of measurable components carried out individually) and also how secure systems can be designed in a manner that in spite of the adversarial environment, the system may perform its intended function [115,116,117,118 and 119]. The approach of evaluating the security of measurable components at system-design level focused on the mechanisms and design of components in such a way that the components facilitated security measurement [120]. The formulation of a methodology for the composing of such individually evaluated components of systems such that the security is ensured is still a research question with no concrete answers and furthermore, no system-design level methodology exists to compose such individuality. Also, very few methodologies exist that quantify the amount of security provided by a particular system [121, 122] and not much either that talk about quantifying security at the design level. The main reason is the fact that most of the security validation attempts are qualitative in nature, focused more on the processes and functionality of the system.

Given the dearth of a solid quantitative security metrics, there exists no quantitative method for measuring a systems *Availability* from the security perspective, but various measurement schemes do exist which measure *Availability* in terms of functionality and performance [103], furthermore there is no measurements of *Availability* at the design level. Given the importance of Availability as a security attribute [123], there is a need to quantify *Availability* as a security attribute. Quantifying *Availability* at an early stage i.e. system design level for systems with component based design would serve the purpose of

security evaluation better because security evaluation at an early stage of system design would facilitate the process of making changes in the design accordingly keeping in view the security and performance of the overall system. This thesis proposes a metric for *Availability* that quantifies *Availability* at the system-design level or for an already working system the metric is applied to the individual working components (software/program code), which are brought into the picture using the system modelling tools.

Why is the metrics software based? The answer is simple, because of the fact that, the hardware of the system is usually more secure, reason being the physical restrictions in attacking the hardware. Since the goal is to measure *Availability* from the security perspective, the hardware that way is affected indirectly, basically by exploiting the operating code of the system. Also whenever we talk about *Availability* of the hardware we are more focused on the functional aspects of the system, rather than the security i.e. system is much better functional (high availability) with redundancy in the hardware.

4.2 Dependability and Availability

Availability is one of the integrative attributes of *Dependability*, as shown in figure 4.1. Dependability is a computer system property such that the service delivered by the system can be trusted and justified for the same. The service delivery is actually the behavior of the system as it is observed by its user(s); a user is a different system (human or physical) which collaborates with the erstwhile [125]. The world today is showing ever-growing reliance and dependence on information computing systems, which has put forward many questions and challenges regarding the limits to their dependability. To counter such questions various global terminological and conceptual frameworks came into existence over the past two decades and a half. As came the concept and terminology of *Dependability* and has undergone various changes since its introduction in the early standard documents of security. Some of the early definitions that were adopted back then are well explained in [124]. With the passage of time and changes in the technological world a more standard definition of *Dependability* was established, based on the classical

notions of *security, reliability, maintainability* and *safety,* which are since then seen as the *dependability attributes* [124 and 125].

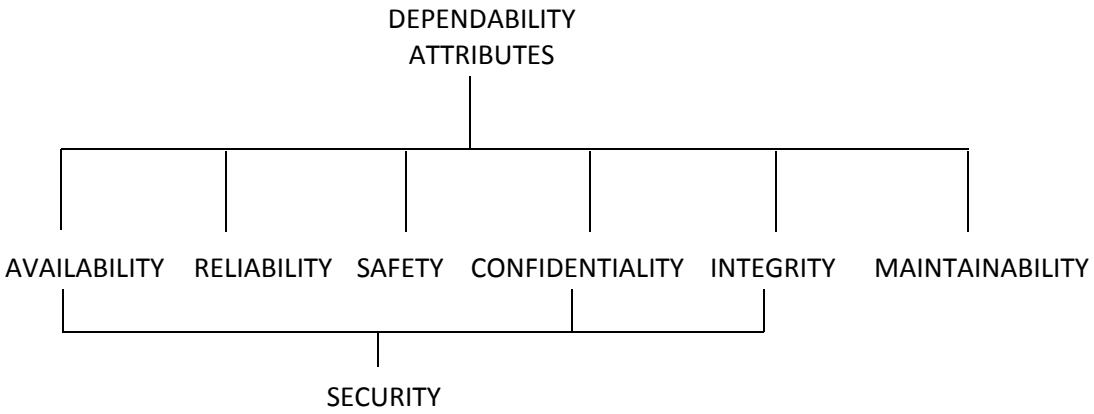

Figure. 4.1: Attributes of Dependability and Security

When we talk about a system being a dependable one, it certainly means that all the attributes of dependability exist in that system. Any alteration or deviation in the values of the attributes will certainly result in the system being lesser dependable. One such deviation can occur in the *Availability* attribute of the system. If the system has a component-based design (CBD) and has a large number of interacting components (i.e. long chains of dependencies), the system may require additional disk space and processing. The long chain of dependencies may result in degrading the performance of the system or in worse case result in a dependency hell [126], which may ultimately result in rendering a system unavailable, thus impacting the *Availability* the Information System. The effects on Availability can impact other security attributes as well, as is explained in [123]. In order to counter such a problem, two things need to be done. First is to see to what extent a system can handle the growing dependencies. Secondly to come up with a measurement scale that gives an idea about the system being stable or unstable based on the dependencies among the components. Lesser the dependencies more are the chances of the system to

work in a stable state, which in other words means a good score for the Availability attribute of the system.

4.3 Dependencies in Component Composition

In a scenario where there are many interacting components of an Information System, a component may call the service of any other component which may in turn call services of other components and so on until the required task is accomplished. The components are interlinked in a well-organized manner in order to provide the required functionality in an efficient and balanced manner. Such a scenario is known as component composition or composition of the system. In the case of distributed/networked environment, the component composition is located over remote information systems. The component composition, in this case, can be both local bound (standalone system) and remote bound. In component based system architecture the component is the basic building block of the system, more precisely a component usually is a black box building block that's only concerned with inputs and outputs, without any knowledge of the internals of the component. In a component composition, components interact, collaborate and participate with each other to carry out the required system functionality, resulting in dependencies among various interacting components. The associations that exist between interacting components can be either direct or indirect [111]:

i. *Direct Dependency*: when the components interact directly.
ii. *Indirect Dependency*: when the components interact through intermediate components

The dependency between components is categorized into four types [112], *implicit dependency (direct and indirect), explicit dependency (direct and indirect)*. Implicit dependencies are related to the systems environment while as Explicit dependency is the clearly defined dependency i.e. a component may refer to other components and may be

used by many components. In a component composition while the components interact, collaborate and participate, the system contains following types of dependencies:

1. *Input / Output Dependency:* when a component provides/requires information to/from another component.
2. *Interface Dependency:* when the interface of a component is invoked by another component using a message to trigger an event through the mentioned interface. It is produced by the integration of user and interface.
3. *Data Dependency:* when the exchange of data occurs between various interacting components. Data that belongs to a component is used in another component.
4. *Time Dependency:* when the components interact with each other in a timely sequence. That is when the working of a component precedes / follows the working of another component.
5. *Control Dependency:* when there is control integration between components. Control integration in CBS's is done either by *remote procedure calls* or by general passing.
6. *State Dependency:* the behavioral action of a component will not take place unless the system or it's any part is in a described state.
7. *Context Dependency:* the component runs must be under appropriate context environment.
8. *Cause and Effect Dependency:* represents that the action of one component implies the action of another component.

4.4 Quantifying Dependencies

To model the dependencies between various components in the system and to derive a metric for Availability based on the components we make use of an *Adjacency Matrix* (AM_{nxn}) [112] aka *Dependency Matrix* or the C*omponent Dependency Graph*. To construct the matrix we need to represent the system components in a graphical form. We make use of UML modeling for the representation of components in a graphical form. In figure 4.2

is shown the structure of a component based system using the UML paradigm. The boxes represent the various interacting components of the system. As shown in the figure the dependencies appear as a result of linkage between the *provider* and *required interfaces* (any type of dependency as mentioned in the list above), these are the *implicit* dependencies. The *explicit* dependencies are shown by the dotted arrow, tail represents the source component that is dependent on the component connected by the arrow head.

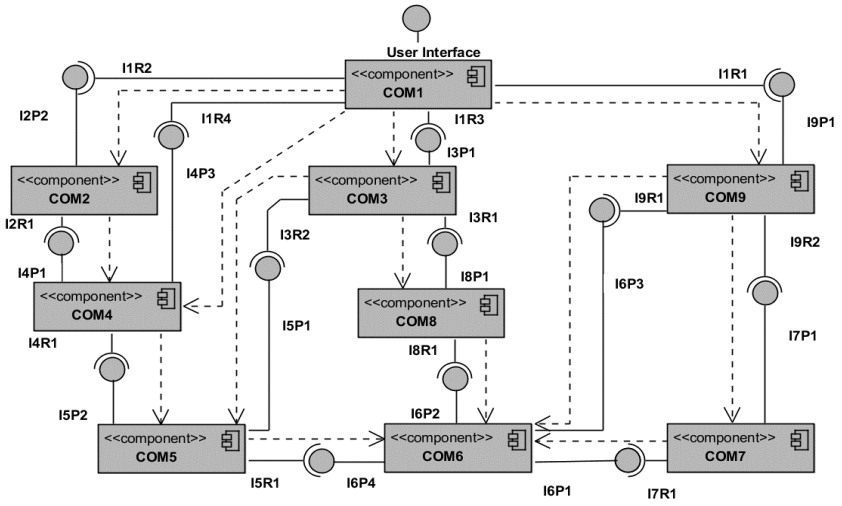

Figure 4.2: Illustration of Components and their Dependencies in a System

In the adjacency matrix denoted by AM_{nxn} each component is represented by a column and a row with indices as "*i*" and "*j*" respectively. Let's assume that a component C_i depends on another component C_j, then the comparable element in the adjacency matrix AM_{nxn} is denoted as "1", otherwise the value is denoted as "0". If an element in the matrix is represented by d_{ij}, then all the values in the matrix AM_{nxn} can be generalized as:

$$AM_{nxn} = (d_{ij})_{\text{nxn,}} \quad \text{where } d_{ij} = \begin{cases} 1, & if\ ci \rightarrow cj \\ 0, & otherwise \end{cases} \tag{1}$$

Therefore the Adjacency matrix AM_{nxn} (aka Direct Dependency matrix DD_{nxn}) for a component composition involving N components would look like this:

$$DD = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ . \\ C_N \end{array} \begin{array}{c} \begin{array}{cccccc} C_1 & C_2 & C_3 & . & . & C_N \end{array} \\ \begin{bmatrix} d_{11} & d_{12} & d_{13} & . & . & d_{1n} \\ d_{21} & d_{22} & d_{23} & . & . & d_{2n} \\ d_{31} & d_{32} & d_{33} & . & . & d_{3n} \\ . & . & . & . & . & . \\ d_{n1} & d_{n2} & d_{n3} & . & . & d_{nn} \end{bmatrix} \end{array}$$

Where,
$C_1, C_2 . . . . C_N$ are components
*d_{ij} is either **0** (no dependency) or **1**(dependency)*

Figure 4.3: Matrix Direct Dependency

The matrix drawn above is a *Direct Dependency Matrix* that represents the direct interactions between various interacting components in the system. Using *Warshall's algorithm* of *transitive closure* [113] we create one more matrix called as Full Dependency Matrix, that contains all possible interactions (direct and indirect) between components. The algorithm for computing the complete dependencies of a component C_i is:

Algorithm1: Warshall's Algorithm for Transitive Closure (FD_{nxn} zero-one matrix)

Input: Adjacency Matrix *DD* of dependency (Relation) *D* on a set of *n* components.
Output: Adjacency matrix *FD* of the transitive closure of *D.*

1. $FD := DD$ [initializing FD to DD]
2. **for** $j := 1$ **to** n
3. **for** $i := 1$ **to** n
4. **if** $T_{i,j} = 1$ **then**
5. $d_i := d_i \vee d_j$ [Boolean OR of row *i* and row *j*, update d_i]
6. **next** *i*
7. **next** *j*

end Algorithm

The input to the Algorithm is the direct dependency matrix and the output after applying the Warshall's Algorithm is the full dependency matrix that looks like:

$$FD = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ . \\ C_N \end{array} \begin{array}{c} \begin{array}{cccccc} C_1 & C_2 & C_3 & . & . & C_N \end{array} \\ \begin{bmatrix} fd_{11} & fd_{12} & fd_{13} & . & . & fd_{1n} \\ fd_{21} & fd_{22} & fd_{23} & . & . & fd_{2n} \\ fd_{31} & fd_{32} & fd_{33} & . & . & fd_{3n} \\ . & . & . & . & . & . \\ fd_{n1} & fd_{n2} & fd_{n3} & . & . & fd_{nn} \end{bmatrix} \end{array}$$

Where,
$C_1, C_2 . . . . C_N$ are components
fd_{ij} *is either* **0** *(no dependency) or* **1***(dependency)*

Figure 4.4: Matrix Full Dependency

The Full Dependency Matrix represents all possible dependencies that a component can have in a component composition. For the dependency(whether direct or indirect) between any two components C_i and C_j belonging to column and row with indices as "*i*" and "*j*" respectively, the comparable element "fd_{ij}" in the full dependency matrix FD_{nxn} is denoted as "1", otherwise as "0".

Related to the dependency matrices, we define the following dependency determinants of an individual component C_i in the composition as follows:

- *Total-Dependency:* of a component C_i is defined as the overall associations of the component C_i with other components in the component composition.
- *Inward-Dependency:* of a component C_i is the number of components in the composition that are directly or indirectly dependent up on the component C_i.
- *Outward-Dependency:* of a component C_i is defined as the components in the composition upon which component C_i depends directly or indirectly for its provided functionalities.

Next, we quantify *Inward-Dependency* and *Outward-Dependency* as *Inward-Degree* and *Outward-Degree* respectively in Full Dependency Matrix.

- *Inward-Degree:* ***inDeg(C_i)*** of a component C_i is the number of components in *Inward-Dependency* of component C_i. It is calculated simply by counting the number of 1's in the corresponding column *j* in the *FD_{nxn} Matrix.* Mathematically the above statement can be written as:

$$inDeg(C_i) = \sum_{j=1}^{n} (fd_{ji}) \quad (2)$$

- *Outward-Degree:* ***outDeg (C_i)*** of a component C_i is the number of components in *Outward-Dependency* of component C_i. It is calculated by counting the number of 1's in the corresponding row *i* in the *FD_{nxn} Matrix.* Mathematically the above statement can be written as:

$$outDeg(C_i) = \sum_{j=1}^{n} (fd_{ij}) \quad (3)$$

4.4.1 Availability Metric Design

When the components of an *Information System* interact, collaborate and participate with each other, a long chain of dependencies can create issues [126] in the system. In order to keep an eye on that, we need to analyze the dependency levels of each of the components in the system. This will give us the indications about the critical behavior of the components and based on such data we can analyze the effects that it can have on the functioning of the overall system from the security perspective.

In the previous section we defined a term *Total-Dependency*, which can be put mathematically as:

$$tDep(C_i) = inDeg(Ci) + outDeg(Ci) \quad (4)$$

$$\Rightarrow tDep(C_i) = \sum_{j=1}^{n} (fd\ ji) + \sum_{j=1}^{n} (fd\ ij)$$

Where,
inDeg(C_i) is the *Inward-Degree* of the component C_i
outDeg(C_i) is the *Outward-Degree* of the component C_i

To control the results in the region of 0 and 1, the above equation can be written as:

$$tDep(C_i) = \frac{1}{inDeg(Ci) + outDeg(Ci)} \quad (5)$$

Where,
inDeg(C_i) or inDeg(C_i) > 0.

The dependency of components $C_1 + C_2 \ldots\ldots\ldots + C_n$ for the overall system *tDep(SyS)* becomes:

$$tDep(SyS) = \frac{\sum_{i=1}^{N} tDep(Ci)}{N} \quad (6)$$

Where,
N is the number of components in the system.

The main trait of Availability is timely access to resources, a delayed response is no response given the speed at which information systems operate these days. In a scenario of a component composition, a component or a group of components may be dependent upon another component or a group of components, which may, in turn, be dependent upon another component or a group of components. Such type of dependency chains may result in delayed responses. This may ultimately impact the Availability of the system. There are more delays if the interacting components are located over remote information systems, in such component compositions the functionality provided by the components is accessed by the client components via the remote procedure calls (RPC's) which start with a client stub call (invocation), then the parameter packing (marshalling) and sending the message

from the client to the server machine. The incoming packets are fed into the server stub and then the parameter unpacking (un-marshalling). Finally the call by server stub to the server procedure. The delay involved is mainly due to the following factors [174] [177] [178]:

i. *Processing delay:* component's processing time measured from its invocation to the return of the results [173].
ii. *Propagation delay:* in the case of remote component composition the time taken by the message to travel from the calling component to the destination component over the network, excluding the processing and queuing delay [173].
iii. *Transmission delay:* in the case of the remote component composition is the time taken to transmit the message from the calling component to the destination component over the network [176].
iv. *Queuing delay:* in the case of remote component composition the time taken by the message to enter the queue or leave the queue of a node on the network [175].

From the above discussion, it's clear that the factors that can impact Availability of the system in a component composition are:

- ***inDeg*** of the component C_i.
- ***outDeg*** of the component C_i.
- ***Delay*** involved in the dependency chain.
    i. *Processing delay*.
    ii. *Propagation delay*.
    iii. *Transmission delay*.
    iv. *Queuing delay*

The metric for *Availability* that we are proposing in the thesis is based on the factors mentioned above. Recall from the figure 4.2 and the definitions of ***inDeg*** and ***outDeg*****,** the number of components that may request the services of a component C_i for their required functionality is $inDeg(C_i)$. The number of components requested by component C_i for its required functionality is $outDeg(C_i)$. As the dependency chain grows and also given the delays associated with the remote/networked nature of the composition, it is certainly going

to show effects on the performance of the component (delayed response or no availability) and the *Availability* of the overall system**.**

Using the above-mentioned factors and equation 5 as the base, availability of the component C_i becomes:

$$IAV(C_i) = \frac{1}{inDeg(Ci)+outDeg(Ci)+Delay} \quad (7)$$

the fact that relationships among every component either in ***inDeg*** or ***outDeg*** are the factor of **1 – N** i.e. for the required functionality, $\mathbf{C}_i$ may call some or every component in ***outDeg*****(C*i*)**, on behalf of the calling components. Therefore in the component chain, the calling components (components in ***inDeg***(***C****i*)) invoking ***C****i* ,accumulate the ***outDeg*****(C*i*)** component by ***inDeg***(***C****i*) number of times. Therefore the above equation becomes:

$$IAV(C_i) = \frac{1}{inDeg(Ci) + \sum_{j=1}^{inDeg(Ci)} (outDeg(Ci)) + Delay} \quad (8)$$

Where,
$\sum_{j=1}^{n} (f\,d\,ji) = inDeg(C_i)$ *and* $\sum_{j=1}^{n} (f\,d\,ij) = outDeg(C_i)$
$inDeg(C_i)$ *or* $outDeg(C_i) > 0$.

Furthermore the metric also take into account the delay associated with the component chain. The delay here is twofold i.e. for systems with *local bound component compositions* and for *systems with remote component compositions.*

For the former (local bound) processing delay $\Delta \boldsymbol{P_j}$ for each component which ***C****i* calls for its service (Components in ***outDeg(C****i*)) is:

$$Delay = \sum_{j=0}^{outDeg(Ci)} \Delta Pj \quad (9)$$

Where,
$j = 0$ *for the processing delay of the component itself*

Therefore the equation 8 for *Availability* becomes:

$$IAV(C_i) = \frac{1}{inDeg(Ci) + \sum_{j=1}^{inDeg(Ci)}(outDeg(Ci)) + \sum_{j=0}^{outDeg(Ci)}(\Delta P j)} \quad (10)$$

For the later (remote bound) we make use of the *delay metric* (used for measuring network performance), the metric comprises of processing delay ΔP, the propagation delay ΔR, the Queuing delayΔQ and the transmission delay ΔT. For each component which ***Ci*** calls for its service (Components in ***outDeg(C_i)***) and also the delay of processing the component $\boldsymbol{C_i}$ itself, the metric for delay of the dependency path can be calculated as:

$$\text{Delay} \quad = \sum_{k=0,l=k+1}^{outDeg(Ci)} (\Delta \mathrm{P}k + \Delta \mathrm{Q}k + \Delta \mathrm{R}kl + \Delta \mathrm{T}kl + \Delta Pl + \Delta Ql) \quad (10)$$

Where,
k and l are two adjacent nodes.
transmission delay from k to l, $\Delta \mathrm{T}kl = \frac{\mathrm{b}}{\rho}$,
b: bits in the packet, ρ: bandwidth between node k and l
$\Delta \mathrm{R}kl$ Propagation time from node k to l
Queuing delay of k: ΔQk, queuing delay of l: ΔQl
Processing delay of k: $\Delta \mathrm{P}k$, Processing delay of l: ΔPl

Therefore the equation for *Availability* for the system with remote component composition becomes:

$$\Rightarrow IAV(C_i) = \frac{1}{inDeg(Ci) + \sum_{j=1}^{inDeg(Ci)}(outDeg(Ci)) + \sum_{k=0,l=k+1}^{outDeg(Ci)}(\Delta \mathrm{P}k + \Delta \mathrm{Q}k + \Delta \mathrm{R}kl + \Delta \mathrm{T}kl + \Delta Pl + \Delta Ql)}$$

Where,
$\sum_{j=1}^{n}(fd\ ji)$ = *inDeg(C_i),* components in in-dependency of C_i.
$\sum_{j=1}^{n}(fd\ ij)$ = *outDeg(C_i),* components in out-dependency of C_i.
inDeg(C_i) or outDeg(C_i) > 0.

The range of values for the Availability metric of the component C_i will be in the region of 0-1. The proposed metric for Availability will serve as an indicator about the critical components of the system. If the value of the availability of a component is somewhere near 0 then the component is rendered as a critical one, higher values nearing 1 means

otherwise. More the number of dependencies, more the value will tend to 0. A lesser value higher risks to the availability of the component. Based on the above equation the *Availability* metric for the overall system would be:

$$IAV(SyS) = \frac{\sum_{i=1}^{N} IAV(Ci)}{N}$$

Where,
N *is the number of components in the system.*
IAV(Ci) is the availability level of the component C_i

The range of values for the Availability metric $IAV(SyS)$ for the system will be in the region of 0-1. Based on this value different designs of the system can be considered and the best design chosen would be the one whose score would be nearing 1. A score nearing 1 would mean stability in terms of analyzing the growing dependencies in the system. The algorithm for measuring Availability is as follows:

Algorithm2: Availability evaluation Algorithm for a Component-based system.

Input: Component-based design of a system with *n* components.
Output: Quantitative Availability measurement ***IAV(SyS)*** of the system.

1. Convert the design of the system into an adjacency matrix $\mathbf{AM_{nxn}}$ (aka direct dependency matrix $\mathbf{DD_{nxn}}$).
   If dependency exists $ci \rightarrow cj$ **then**
   $d_{ij} = 1$
   Else
   $d_{ij} = 0$
   End If
2. Using Warshall's Algorithm for transitive Closure convert the matrix $\mathbf{AM_{nxn}}$ into full dependency matrix $\mathbf{FD_{nxn}}$.
3. Quantify ***inDeg*** and ***outDeg*** of component C_i
   $inDeg(C_i) = \sum_{j=1}^{n} (fd_{ji})$
   $outDeg(C_i) = \sum_{j=1}^{n} (fd_{ij})$

4. Calculate Delay

$Delay = \sum_{j=0}^{outDeg(Ci)}(\Delta Pj)$ // for local bound $\mathbf{C_i}$

or

$Delay = \sum_{k=0,l=k+1}^{outDeg(Ci)}(\Delta \mathrm{P}k + \Delta \mathrm{Q}k + \Delta \mathrm{R}kl + \Delta \mathrm{T}kl + \Delta Pl + \Delta Ql)$//for remote bound $\mathbf{C_i}$

5. Measure Availability of every component $\mathbf{C_i}$ in the composition of systems with the two versions of component composition (local and remote) on the scale 0-1.

$$IAV(C_i) = \frac{1}{inDeg(Ci) + \sum_{j=1}^{inDeg(Ci)}(outDeg(Ci)) + \sum_{j=0}^{outDeg(Ci)}(Pj)}$$

Or

$$IAV(C_i) = \frac{1}{inDeg(Ci) + \sum_{j=1}^{inDeg(Ci)}(outDeg(Ci)) + \sum_{k=0,l=k+1}^{outDeg(Ci)}(\Delta \mathrm{P}k + \Delta \mathrm{Q}k + \Delta \mathrm{R}kl + \Delta \mathrm{T}kl + \Delta Pl + \Delta Ql)}$$

6. Quantify Availability for the system with N number of components.

$$IAV(SyS) = \frac{\sum_{i=1}^{N} IAV(Ci)}{N}$$

end Algorithm

4.5 Empirical Evaluation and Analysis

For the validation and analysis of the usefulness, appropriateness of the Availability metrics, the empirical evaluation of the proposed metric is carried out in this section. The accepted and prevailing methods used by the research community for empirical evaluation are, case studies, surveys and experiments (127). The best method suited for our case is the third one i.e. the experimental way. The system that we are using for the purpose of evaluation is a web/network based video monitoring system "EES". It is a motion activated video monitoring system that follows the movements of the object if any motion is detected. The system makes use of a webcam for capturing data (video) and then stores the data in a database. The video stream can be watched online from the web interface or later in a passive mode. At the highest level of abstraction (service level) the system consists of three software segments:

- A database
- Application program on the client's computer that controls the webcam.
- Web interface for remote access to the video stream and settings.

Further these segments are broken down into different modules and our evaluation is focused on these modules of the code. These modules constitute the components in our metric for Availability. The system is developed in Java (J2SE 1.6) and the back end is Oracle Database 11g version 11.1.0.6. The architecture of the system is designed keeping in view the UML 2.0 specifications for component level design and all the notions are in accordance with the said modelling language. The design tool used is "Visual paradigm 9.0 for UML".

4.5.1 Experiment Setup

For the illustration of *delay* in the proposed *Availability* metric we conduct a small scale experiment using a simple network topology given in figure 4.5.

Table 4.1: System Configurations used in Experiment.

Machine	Operating System	Hardware Configuration
Web Server (WINR2)	Windows server 2012 R2 (6.3 build 9600)	Intel® Core™ 2 Duo 2GHz, 1 GB RAM
Application Server	Windows 8 (6.3 build 9600)	Intel® Core™ i5 2.8 GHz, 4 GB RAM
User	Windows 8 (6.3 build 9600)	Intel® Core™ i3 1.8 GHz, 2 GB RAM

The EES Sys_Database running on oracle 11g is native to the client machine. Router is configured to relay data from WINR2 to the client machine. Using the web interface the user gets the access to the video footage in real time.

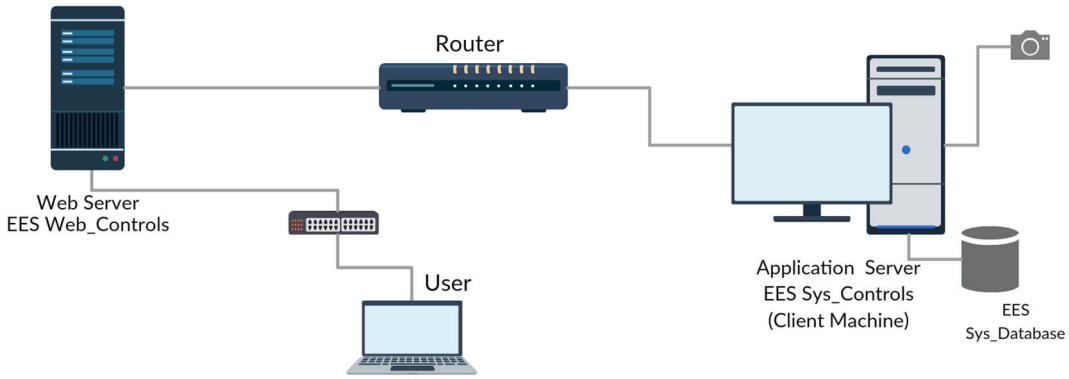

Figure 4.5: Experimental Setup of EES

4.5.2 Results and Discussion

We start with the first step in the algorithm 2, converting the following design into an adjacency matrix $\mathbf{AM}_{nxn}$:

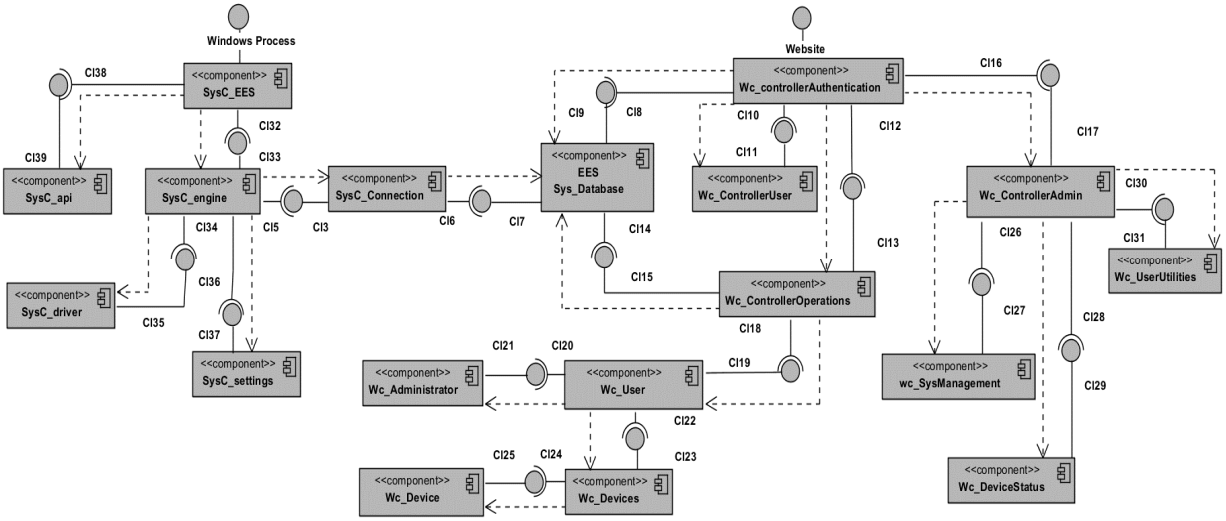

Figure 4.6: Component Based Design of EES

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1
11	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.7: Adjacency Matrix (Direct Dependency) AM_{nxn} of EES

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1
11	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	1	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.8: Full Dependency Matrix FD_{nxn} of EES

The Adjacency Matrix $\mathbf{AM_{nxn}}$ in figure 4.7 represents the direct interactions between all the interacting components in the EES system. After applying the Warshall's Algorithm for transitive Closure, we convert the matrix $\mathbf{AM_{nxn}}$ into the Full Dependency Matrix $\mathbf{FD_{nxn}}$. This matrix gives us all the possible interactions (direct and indirect) of every component in EES. The resultant matrix is shown in figure 4.8.

Next we quantify ***inDeg*** and ***outDeg*** of every component c_i using equation no. 2, and equation no. 3. The values are given in the table 4.2. After the quantification[1] of ***inDeg*** and ***outDeg*** we calculate the *delay* (Δ) associated with every component in the composition. For remote bound connections the delay is calculated by time-stamping the RPC call, from the invocation of the client stub to the execution of the server stub.

Table 4.2: Data Assemblage Table[2]

C_i	*Component Name*	*InDeg(C_i)*	*outDeg(C_i)*	*Delay* (μ)	*Standard Deviation* (σ_{Delay})
1	SysC_EES	1	6	0.208	0.025
2	SysC_api	1	1	0.108	0.023
3	SysC_engine	1	4	0.160	0.034
4	SysC_Connection	2	1	0.138	0.023
5	SysC_driver	2	1	0.146	0.029
6	SysC_settings	2	1	0.126	0.030
7	EES Sys_Database	5	1	0.248	0.038
8	Wc_controllerAuthentication	1	11	0.290	0.030
9	Wc_ControllerUser	1	1	0.102	0.017
10	Wc_ControllerAdmin	1	3	0.124	0.022
11	Wc_ControllerOperations	1	5	0.182	0.034
12	Wc_UserUtilities	2	1	0.118	0.023
13	Wc_Administrator	3	1	0.148	0.031
14	Wc_User	2	3	0.174	0.027
15	wc_SysManagement	2	1	0.142	0.038
16	Wc_Device	4	1	0.190	0.030
17	Wc_Devices	3	1	0.146	0.033
18	Wc_DeviceStatus	2	1	0.116	0.032

[1] To avoid infinite values we set *inDeg* and *outDeg* values to "1", wherever the value is "0".
[2] The values in red are the values changed from "0" to "1".

The difference between these two times gives us the time taken by a component to call a remote component for its required functionality. For the local bound connections delay is calculated by time-stamping the local component call in the program code. The process is repeated 5 times for the trueness of the results and mean ($\boldsymbol{\mu}$) of the values is calculated and then used in the evaluation of the *Availability* of the individual component. The values are given in the Table 4.2. For the validation of the mean delay we also calculate the standard deviation ($\boldsymbol{\sigma}_{\text{Delay}}$).

After the above mentioned calculations are done, we next calculate the Availability value of every component in EES i.e. we evaluate the step 5 of Algorithm 2. The calculations are given in the table 4.3.

Table 4.3: Availability score of individual components and the overall system.

C_i	Component Name	Availability	System Availability Indicator(EES)
1	SysC_EES	0.122940743	0.196181497
2	SysC_api	0.474383302	
3	SysC_engine	0.171880371	
4	SysC_Connection	0.227998176	
5	SysC_driver	0.241196334	
6	SysC_settings	0.242365487	
7	EES Sys_Database	0.097580016	
8	Wc_controllerAuthentication	0.071530758	
9	Wc_ControllerUser	0.475737393	
10	Wc_ControllerAdmin	0.222222222	
11	Wc_ControllerOperations	0.141083521	
12	Wc_UserUtilities	0.242836328	
13	Wc_Administrator	0.162654522	
14	Wc_User	0.115500116	
15	wc_SysManagement	0.241429261	
16	Wc_Device	0.122100122	
17	Wc_Devices	0.157828283	
18	Wc_DeviceStatus	0.242954325	

The results are divided into ten Availability zones ($\mathbf{A}_1$ to $\mathbf{A}_{10}$), with components lying in $\mathbf{A}_1$ being the most critical and unreliable components. Practically it's impossible for a component to achieve an Availability rating of $\mathbf{A}_{10}$ given the controlled nature of the values of equations between 0 and 1. All the components lying above $\mathbf{A}_4$ showed high performance in each of the runs that were conducted for calculating the delay. For such reason we have classified every component above $\mathbf{A}_4$ as the high performance components for EES. Most critical components are the values shown in red. For such components extra security features should be added at the code level of the system. Exploitation of these components can easily disrupt the working of whole system. Next to most critical are shown in blue and heavy usage of these components can also bring performance issues. The component values in black exhibited moderate processing times and the quickest of them all are the ones in green. The graphical analysis of table 4.3 is shown in figure 4.9. The Availability for the overall system is calculated after applying the step 6 of algorithm 2. The results shown in table 4.3 and graphically analysed in figure 4.10. This output is also categorized into different Availability levels ($\mathbf{A}_1$ to $\mathbf{A}_4$), lowest and critical level being $\mathbf{A}_1$. The EES system just lies below the $\mathbf{A}_3$ (.2 - .3) in the $\mathbf{A}_2$ zone.

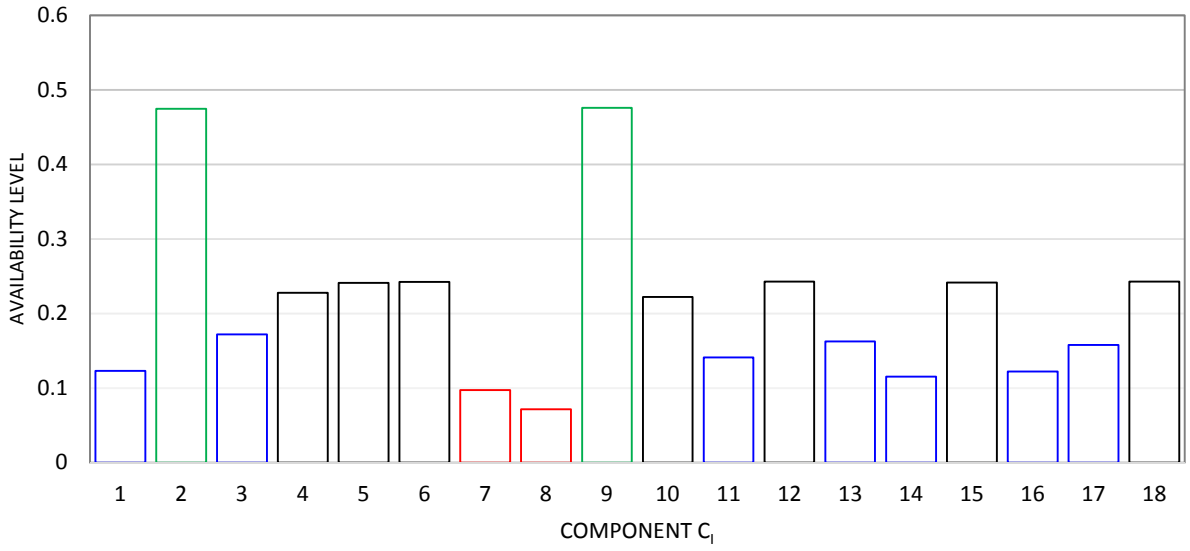

Figure 4.9: Availability graph of EES components.

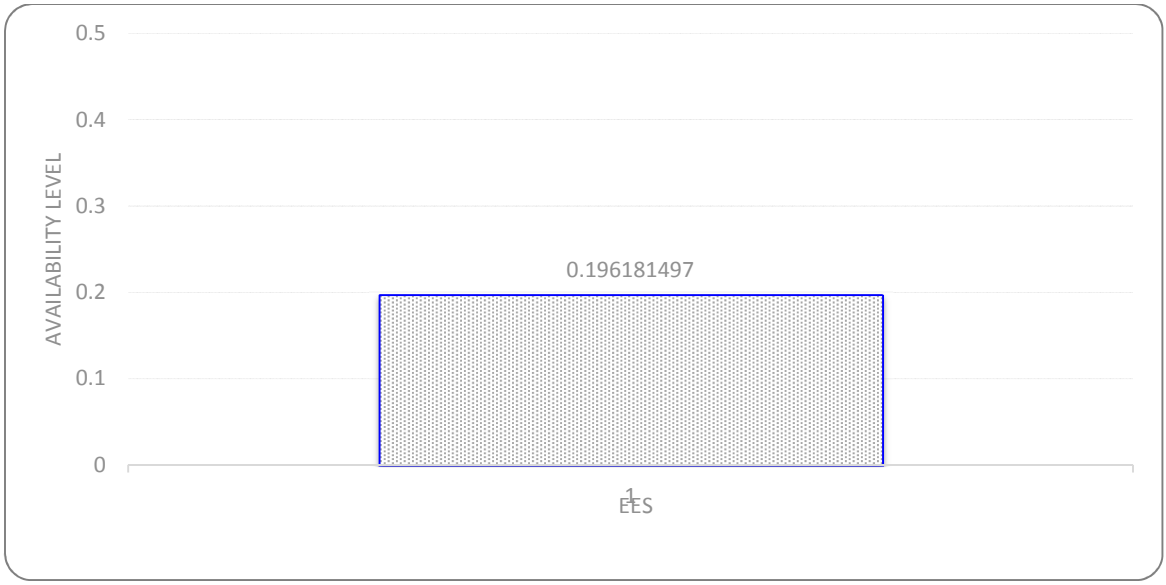

Figure 4.10: Availability graph of EES System.

4.5.3 Conclusion

While measuring the Availability if we go beyond the application level of an information system i.e. the component level, the dependencies among the various interacting components can be used to determine the availability/workability or risk analysis of an information system. The work in this chapter presented a novel metric of measuring the availability at the component level that gave us an idea about the risk involved (from the security perspective) in the particular design of the component composition. The metric is based on the various interactions among the components of the system, plus the processing time taken by each of the components whether components be local bound or remote bound. More the dependencies of a component on other components more complexity in the design which may ultimately result in low performance and may ultimately impact the workability/availability of the information system. The work in the thesis gives us an analysis of each component with respect to the dependency on other components and the processing times associated with those interactions. Using the results from the metrics as a

reference the design may be altered for better performance of the information system. Since the metric is more inclined towards the software part of the information system, the future scope lies in incorporating more of the other components (hardware, user and network) in the metric as well. Also in the future the work can be extended to distributed computing environment, which involves a complex component based architecture of hardware, software and the network.