Network Management Tool
Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

2. Architectural goals and philosophy

The system is just a prototype for a new product. It will run in a single computer for simplicity.Configuration management should be simple. So the cost should be minimized and the system must be timely delivered, that is by April 23, 2012. The system should be reliable and simple to use.

Below are design goals listed in descending priority order.

- **Minimized Cost and timely delivery**
- **Simplicity**
- **Reliability**
- **Usability**
- **Performance**
- **Availability**

Best practices and design principles will be applied in two main areas

1. Presentation services to individual desktops should be uncoupled
    a. a common look and feel for the Network Management toll
    b. data should be validated before any further processing
    c. the user interface will be designed in such a way that common user interface functionality will be implemented in a similar manner across the board
2. Business Rules should be encoded within the application development framework
    a. Business rules will need to be separated from the presentation and database
    b. Adoption of a component based framework needs to be considered to promote reuse.

3. Assumptions and dependencies

- This tool will be developed with NetBeans for fast GUI development which means the AbsoluteLayout.jar has to be included in the build path.
- **mysql-connector-java-5.1.19-bin.jar** for JDBC

4. Decisions, constraints, and justifications

[List the decisions that have been made regarding **architectural approache**s and the constraints being placed on the way that the developers build the system. These will serve as guidelines for defining architecturally significant parts of the system. Justify each decision or constraint so that developers understand the importance of building the system according to the context created by those decisions and constraints. This may include a list of DOs and DON'Ts to guide the developers in building the system.]

I[1]: how is the data managed going through SocialNetworkNode? Is it necessary to have a database to store some persistent data for SocialNetworkNode?

P[1.1]:memory All data needed by SocialNetworkNode will only be stored in dynamic memory.
It's simple and fast since we don't need to keep the data in sync.
But it reduces reliability of the system. If the system crashes or the computer is shut down, all the accumulated data from SocialNetworkNode will be lost.

P[1.2]:in-memory database configured with disk-based database backup.
In-memory database provide fast data access and modification. Disk-based database ensures the data won't be lost when the system crashes or unexpectedly shut down.
But nobody has good experience with in-memory database and configuration management. More time is needed.

P[1.3]: disk-based relational database.
Configuration is simpler. Reliability is ensured. Everybody has good experience with disk-based database, delivery time won't be delayed.
But performance is impacted. But the database management won't be substantial, so the performance won't be noticeable.

[1.4]:

R[1]: P[1.3] is better in overall.

I[2] Should the system set the priority based on some rules specified by the users or administrators?

P[2.1]: System should set the priority itself, the reason is that it is more realistic. The user or administrators cannot set the priority themselves, the priority should be set only by system.

P[2.2]: The users or administrators can set the priority themselves, because they can get more permissions, they can control well, and they can also help system share the workload.

R[2] : P[2.2] is better in overall.

I[3]. What kind of sorting algorithm should be used to sort the commObj lists?

R[3]: we didn't use any sorting algorithm to sort commObj lists. Instead, we use a method named container in java.util which can sort the lists automatically. It will be working as long as we achieve comparable.

I[4]. Should the system provide UI to E team ?

R[4]: Making UI is not a complicated work. We also have experienced people who is good at coding UI(NetBeans). It's also convenient for us to code UI since we are a coding group rather than group E, which is an evaluation team. So the answer is yes, since system can provide UI to

E team, E team no longer have to code the UI.

5. Architectural Mechanisms

Architectural Mechanisms are common solutions to common problems that can be used during development to minimize complexity. They represent key technical concepts that will be standardized across the solution. Architecture mechanisms facilitate the evolution of architecturally significant aspects of the system. They allow the team to maintain a cohesive architecture while enabling implementation details to be deferred until they really need to be made.

Architectural Mechanisms are used to satisfy architecturally significant requirements. Usually those are non-functional requirements such as performance and security issues. When fully described, Architectural Mechanisms show patterns of structure and behavior in the software. They form the basis of common software that will be consistently applied across the product being developed. They also form the basis for standardizing the way that the software works; therefore, they are an important element of the overall software architecture. The definition of architecture mechanisms also enable decisions on whether existing software components can be leveraged to provide the required behavior; or whether new software should be bought or built.

The value in defining architecture mechanisms is that they:

1. Explicitly call out aspects of the solution mechanics that are common across the system. This helps you plan.
2. Put down markers for the developers to build those aspects of the system once a nd then re-use them. This reduces the workload.
3. Promote the development of a consistent set of services. This makes the system easier to maintain.

An Architectural Mechanism can have three states: Analysis, Design and Implementation. These categories reflect the maturity of the mechanism's description. The state changes as successive levels of detail are uncovered during when you refine Architecturally Significant Requirements into working software. The categories are summarized in the table that follows.

States of an Architectural Mechanism

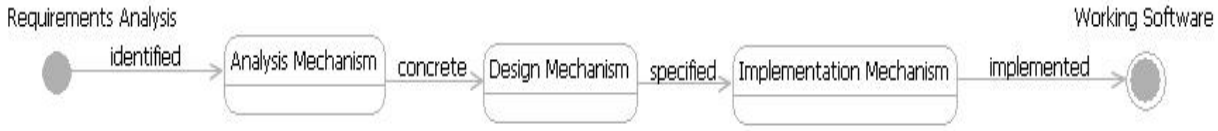
State	Description
Analysis	A conceptual solution to a common technical problem. For example, persistence is an abstract solution to the common requirement to store data. The purpose of this category is simply to identify the need for an Architectural Mechanism to be designed and implemented; and capture basic attributes for that mechanism.
Design	A refinement of an Analysis Mechanism into a concrete technology (for example, RDBMS). The purpose of this category is to guide precise product or technology selection.
Implement ation	A further refinement from a design mechanism into a specification for the software. This can be presented as a design pattern or example code.

For more information on these different types of mechanisms, see the attached concepts.

Be aware that these states are frequently referred to themselves as Analysis, Design and Implementation mechanisms. These are synonyms and merely represent the architecture mechanisms in different states of development. The transition from one state to another can often be obvious or intuitive. Therefore, it can be achieved in a matter of seconds. It can also require more considered analysis and design, thus take longer. The important point here is

that these categories of mechanisms apply to the same concept in different states. The only difference between them is one of refinement or detail.
The following diagram illustrates the transition of Architectural Mechanisms from one state to another.

State Machine for Architectural Mechanisms



1. Persistence

java utility class to handle the reading and writing of stored data.

2. Process Management

Provides support for the management of processes and threads.

6. Key abstractions

While there is an infinite number of issues that could be investigated with respect to NMTs that could address disaster management, this customer is particularly interested in one aspect: socialNetwork, informationNetwork, communicationNetwork after an establishment of a link between two disconnected communication networks.

6.1 communicationNetwork: The software installed in devices which can connect to internet.

6.2 informationNetwork: for customers to input and save their GPS, medicalCall, ICEcontacts information.

6.3 socialNetwork: for customers to input and save their Name, Address, DOB, Phone#, Friend List, Occupation, StatusBlurb information.

7. Layers or architectural framework

Application architecture defines the various components and their interactions in context of a whole system. the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating

system in the presentation layer would only affect the user interface code.Layered structure offers us benefits as described below:

- STRUCTURE: Organizing applications along business-level boundaries and not technical boundaries
- SPEED & FLEXIBILITY: Making application changes through configuration and not programming
- CONTROL: Modifying, extending or overwriting any architectural element.
- REUSE: Achieving greater reusability and integration by loosely coupling application logic to infrastructure.

1. Presentation layer and content layer:

This is the topmost level of the application. The presentation tier displays information related to SocialNetworkNode and InfomationNetworkNode. It communicates with other tiers by outputting results to user interface.In this layer, developer provides GUI which is for clients.

2. Business Objects Layer

The business Objects Layer is pulled out from the presentation tier and, as its own layer, it controls an application's functionality by performing detailed processing.

The Business layer will constitute of: Java Beans and Java Classes: Java Beans are used to manage the data flow between the layers. Java classes on the other hand are simple java objects that provide utilities to the application. They may also contain business logic and provide other supporting services.Business layer are consist of InformationNetwork node, SocialNetwork node and CommunicationNetwork node.

3. Data Access Layer

This tier consists of database servers and utility class to access the database. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.
The Data Utility Class allows for the abstraction of the persistence from the business component. The Data Utility Class manages the connection to the data source to obtain and store data. It encapsulates all access to the data store.