Homework 3

System: Log4j 2.7

Mukesh Dangi

(1) Architecture Recovery Method: RELAX

Reason: The more information --specially in graphical form-- we have, more likely a user will be able to comprehend the system' architecture and the less explicit information we have, the person who has deeper knowledge will be able to understand the system architecture.

Given all the result, A ***user*** will be able to find the useful information by analyzing some of the files from RELAX recovery method. A user is very likely to comprehend the system if we provide the graphical representation of system recovered from one there recovery methods instead of cluster files or any other textual or files written in different programming languages. RELAX recovery method has a lot to offer in terms of system recovery results and information which a user can find very comprehensive and useful.

Many users treat system as a black box except the information for which they are care about. Out of three recovered results of a system, a user --assuming that (s)he has used the system previously and not a new bee to software-- will be able to absorb the system's abstract level architecture like directory structure, connectivity of different components etc.

log4j-2.7-api_directories.pdf: This graphical representation gives a very good view of the system's package structure. From this file a user can understand information like project hierarchy, directories and connectivity.

From log4j-2.7-api_RELAX_clusters_Concerns.pdf a user will be able to find some information about system's component and their connectivity. However, User will not able to identify the internal implementation of the system. it's more likely that a user chooses RELAX results to describe the system's architecture. There are some improvements which are described in bonus question's answer which are will be very helpful for a user than developer or architect.

(2) Architecture Recovery Method: ARC

Reason: In the next level, ***software developer*** comes into the picture. Software developer knows more about programming languages and best practices to implement the provided architecture However, has less knowledge to envision and model big software.

Software developer not only able to trace the software architecture from RELAX recovery results but also from ARC result as well. Along with RELAX results, ARC result will be very useful for him because he has very good understating of programing languages and has substantial understanding of architectures. A software developer is more likely to pick the ARC results which describe system in the more of technical terms like clusters, components, connector, cluster dependencies.

From log4j-2.7-api_30_top_words_per_topic.txt, a developer will be able to describe the clusters and their most relevant words. Moreover (s)he can provide more information like Log4j 2.7's relevant files, package collections like *util*, *message*, *log* and *param*. In log4j 2.7 pairs of clusters which are not connected to each other are *org.apache.logging.log4j* and *org.apache.logging.log4j.spi* and second - *org.apache.logging.log4j.util.SortedArrayStringMap* and *org.apache.logging.log4j.message.ReusableMessageFactory*. Here a developer can confirm that there are some clusters with high connectivity or cohesiveness. So, a software developer is more likely to pick results from ARC recovery method.

(3) Architecture Recovery Method: ACDC

At the end, ***Software architecture*** comes in to play the role who has broad and deep technical knowledge. Software architecture sits on top of other two –user and developer—and generally envision, collaborate, develop models/prototypes of a complex system. An architect can find useful information from ARC, RELAX as well as ACDC results.

ACDC gives very less information about the system in terms of graphical representation of system's architecture so a user cannot get anything from this method. However, an architect can draw some conclusions based on log4j-2.7-api_acdc_clustered.rsf for example Log4j 2.7 has 145 entities in it. Most congested clusters like *spi.ss* and *message.ss* have 30% and 25% entities encapsulated in it. If architect want to analyze the architecture of the system, (s)he can find useful information from ARC results like *log4j-api_deps_clusters.pdf*, *log4j-api_30_top_words_per_topic.txt* and RELAX result like *log4j-2.7-api_directories.pdf, log4j-2.7-api_relax_clusters.pdf*. Overall, he could be the person who get most out of these results.

Describe the system by its visualization (ARC and RELAX):

In order to maintain and document the system,
(a) the new team has to first recover its architecture using available recovery method(s).
(b) The next step would be identifying system components and their tasks, connector and their roles, information flow, system's logical and deployment view.

Given the results from all three recovery methods, A user will find it very difficult to re-construct an overall abstract image of the system except getting some of the ideas from graphical representation of the system *like log4j-2.7-api_directories.pdf and log4j-2.7-api_RELAX_clusters_Concerns.pdf*. As all three methods gives more of a technical sense of a system hence a user will not be able to understand the internal implementation or architecture very well. While comparing ARC and RELAX from previous work, A user will be able to find most noticeable difference like new directory has created or removed, a new cluster has been added or destroyed etc. In term of providing noticeable changes to a user, RELAX method's results will be good fit as compare to ARC.

However, a developer and architect will be able to some of the principal design decisions like project structure, hierarchy, clusters and connectivity with among them, most congested cluster, less useful cluster, Changes in cluster dependency, which cluster requires service other one etc. from ARC and RELAX results. Moreover, they can also mark the clusters which could be impacted majorly by future changes. More reasoning for this part has been explained in the above part as well.

Recovery method to re-create the system's architecture and documenting it.

To create the new architecture and document it, I would prefer to use RELAX method which gives both abstract and deep understating of static aspect of the system. RELAX's graphical representation gives overall system architecture, nodes/clusters, entities, relationship/connections among them. Moreover, RELAX's recovery methods give information about architecture drift, re-designed components, directories, core clusters like SQL, UI, IO and connectivity of the. Here in 2.7, *"message"* and *"util"* clusters are more connected with others than any other cluster hence most of other clusters are depended on "message" and "util" clusters. This tells us that if anything goes wrong in any of the most congested node or cluster then changes could affect it's connected clusters.

RELAX encapsulates related files into a cluster for a predefined topic. Using RELAX recovery method we can capture the grouped clusters which in return can really help us defining the package structure or hierarchy. In Apache log4j 2.7, we have total 170 entities with 5 topics named as Graphics, IO, Networking, SQL and No Match for unspecified entities. Additionally, in log4j 2.7 *message* and *ThreadConext* clusters have more connectivity with other clusters which indicates that any change in the message and *ThreadContext* clusters could impact many other components of the system.

RELAX is very helpful in reconstructing the principal design decision of log4j. In Order to re-construct the prescriptive architecture of this system, we need to have system structure, functional/NF properties, components, connectors and their interaction. Let's capture these from RELAX method's results of log4j 2.7: -

(a) Files and Directory structure: This can be reconstructed from log4j-2.7-api_directories.pdf. Moreover, this file can really help us in ascertaining domain of the system. After a successful categorization to domain specific architecture we can move ahead to analyze rest of the design decisions.
(b) Functional and Non-Functional properties: Functional and nonfunctional properties of the system is not clearly visible clearly in RELAX results unless architect analyze textual visualization. However, even from textual visualization it is very difficult to list down all the properties. This could be one of the improvement in the architecture recovery techniques where we are not getting any abstract architecture
(c) Components, Connectors and their interaction: log4j-2.7-api_RELAX_clusters_Concerns.pdf gives clear picture of clusters and connections among other clusters. Visibly there are many nodes which has many-to-one or one-to-one mapping with each other.

Architectural style(s) best suited to capture the results (by RELAX, ARC and ACDC) of log4j 2.7:

By analyzing log4j-2.7-api_deps.rsf from ARC and ACDC, log4j-2.7-api_RELAX_clusters_Concerns.pdf, log4j-2.7-api_directories.pdf(Tree) from RELAX, we can see that there are dependencies among clusters and each dependency contribute to layering in the system. We have many entries like "contains a b" or "depends a b" which really gives some sense of layered style in log4j.
Moreover, log4j-2.7-api_deps.rsf result file resembles as client-server style where each "depends a b" means 'a' means that 'a' component depends on some service of component 'b' so component 'a' can be assumed as client and 'b' as server.

We don't find information such as central computation system or memory/board which can substantiate the argument for blackboard style. Additionally, we have dependency information However we don't any explicit information about publisher – subscriber like which component has subscribed an event from which component.

Additionally, we can say that there is no trace of CORBA style because CORBA is meant for heterogeneous programming languages and components which are written in different languages. We don't find two components which are written in two different languages. These methods captured static aspect of the log4j system so we are unable to get any information about data flow except "depends a b" where information flows from component 'b' to 'a'.

we can determine information flow based on dependency information from log4j-2.7-api_deps.rsf whether we have bi-directional or unidirectional information flow. In log4j, we have both types of entries depends a b", depends b a" which substantiate that information is flowing in both direction and both components are interdependent on each other and there are few entries with "depends c d" unidirectional flow.

Modeling Language(s) best suited to capture the result (by RELAX, ARC and ACDC) of log4j 2.7:

Log4j is logging system which is being used across multiple domain specific and general-purpose applications. Most of our recovery methods capture static or structural aspect of the system. So, we'll not able to use any of the modeling languages like *Rapide* or *Write* which capture dynamic or behavioral aspects of a system.

Moreover, log4j is not a domain specific application as explained earlier hence domain specific modeling languages like Koala, Weaves and AADL are not going to play any major role. However Generic modeling languages and architecture description languages will be very helpful in capturing static aspects of the log4j system.

<u>Generic modeling languages:</u> These modeling techniques are very common for recovery methods which capture static aspects of the system hence results from **ARC**, **ACDC** and **RELAX** can be modeled using generic languages.

(a) Natural language/plain English are best suited for an average size system like log4j where we don't have many complex services. We've plentiful tools like word processor or other text editors to capture the system's architecture. Natural languages are easily understood by all stakeholder even who don't have technical knowledge. Many of results like log4j-2.7-api_deps.rsf, log4j-2.7-api_directories.pdf, log4j-2.7-api_RELAX_clusters_Concerns.pdf etc. can be easily modeled in natural languages using text editors.

(b) Graphical modeling like PowerPoint is highly expressive in terms of graphical representation of a system's static aspects/architecture. They are limited in space. RELAX Result like log4j-2.7-api_directories.pdf, **ARC** result like log4j-2.7-api_deps_clusters.pdf are using graphical modeling techniques to capture system aspects. **ACDC** has no graphical represented output however we can still use graphical modeling technique to model the results. For example, log4j-2.7-api_acdc_clustered.rsf has many entries in "*contains a b*" format which can be drawn as arrow from 'a' to 'b' component of the system.

(c) UML: Although our recovery techniques doesn't capture dynamic aspects of the log4j system, we have rich set—13-- of modeling languages in UML itself. We can use class diagram to capture the aspects of the system. From **RELAX** result -- log4j-2.7-api_deps.rsf which has output in the "depends a b", we can create a class diagram in which 'a' and 'b' are classes and 'depends' can signify the connectivity in between them. Moreover, log4j-2.7-api_RELAX_clusters.pdf gives us graphical representation of the clusters and relationship among them. Class diagram modeling technique can be used to capture this result by taking each cluster a class and each connection between two clusters is a connector between related classes. Each cluster—IO, SQL, Networking, graphics and no_match are treated as components and each dependency among them is connectors. IO and graphics clusters are analogues to layout in actual log4j architecture.

Similarly **ARC** results can help us to model log4j 2.7. For example, *log4j-api_deps_clusters.pdf* has cluster and connections among them. We can each cluster as a class and a connection as a connector between those classes. However, *log4j-api_deps_clusters.pdf* doesn't give any behavioral aspect or information log4j.

Architecture Description languages:

As explained earlier, most of our recovery methods capture static or structural aspect of the system. So, we'll not able to use any of the modeling languages like *Rapide* or *Write* which capture dynamic or behavioral aspects of a system.

(a) Darwin: In Darwin, we have component, connector and configuration with 'required' and 'provided' interface. We can use Darwin modeling language with help of RELAX's log4j-2.7-api_deps.rsf result in which "depends a b" can be used as required-provided service. In this example 'b' is provider of some service to (require) 'a' and at the end of this process we get a complete log4j configuration.

(b) AADL are specifically used for modeling hardware and software aspect of a system. Log4j doesn't have much of hardware related to dependencies so AADL is not suitable to model log4j. Acme is similar to Darwin—capturing structural aspect of the system-- and can be used to model log4j. In order to model in Acme, each cluster—*IO, SQL, Networking, graphics* and *no_match* is analogues to a component and each dependency among them is connector.

Bonus Qs:

Improvements: Let's take **RELAX** recovery method. Even though RELAX produces very insightful information of a system, but it fails to give clearer picture of a system. Few of them are listed here and could be improved:

(a) RELAX gives many graphical represented output files which are very useful and comprehensive However RELAX doesn't give dynamic behavior of system which be very useful in many ways to understand the system thoroughly like which component is currently executing the tasks, which component has using more memory and threads, where out threads are getting stuck. Moreover, this can help us in identifying the crucial components of the systems.

(b) Producing a dynamic behavior could potentially help us to understand the system in better way. Moreover, we can simulate the system in virtual environment which could benefit us in catching bugs in the system before launching it out.

(c) Implementing a small system which captures a system's dynamic behavior is not a challenging task. We can have a simple python script or java program which capture the snapshot a system during run time. And at the end of the execution of the task we can analyze the system behavior by comparing snapshots all together either manually or by image processing algorithms.

(d) Graphical or textual representation of the overall system: RELAX gives information like directory or package structure, clusters and their connectivity with other components, however we can add a tool which produce a start-to-end overview/abstract architecture of a system because at present none of these three recovery methods give us an abstract architecture of the system. This is result can really help us in understating the user navigation like start and end point of the user activity.

(e) RELAX can help us in finding components and connecters However, we cannot categories them by their role like convertor, coordinator, facilitator and communication explicitly. Moreover, getting a logical and deployment viewpoint of the system also important which is not provided by our three recovery methods.

(f) Nonfunctional properties are also very useful and none of our three mentioned techniques give any information about it. it's very important to note them However, it could be very difficult to document them unless we have a recovery method which can capture dynamic aspects of the system.