Software Architecture Decomposition Using Attributes

Chung-Horng Lung, Xia Xu
Department of Systems and Computer Engineering
Carleton University, Ottawa, Ontario, Canada
email:{chlung, xiaxu}@sce.carleton.ca

Marzia Zaman
Cistel Technology
Ottawa, Ontario, Canada
marzia@cistel.com

Abstract. Software architectural design has an enormous effect on downstream software artifacts. Decomposition of functions for the final system is one of the critical steps in software architectural design. The process of decomposition is typically conducted by designers based on their intuition and past experiences, which may not be robust sometimes. This paper presents a study of applying the clustering technique to support decomposition based on requirements and their attributes. The approach can support the architectural design process by grouping closely related requirements to form a subsystem or module. In this paper, we demonstrate our experiences in applying the approach to a communication protocol software system.

1. Introduction

Alexander [1] demonstrated the application of the partitioning/clustering technique to building a village in India. Clustering and partitioning are conceptually similar. Partitioning or decomposition is a top-down approach to divide a system into subsystems with an aim of high cohesion within a subsystem and low coupling among subsystems. Clustering, on the other hand, is a bottom-up approach to group similar objects as clusters. Collection of clusters forms a subsystem or a system.

Alexander [1] postulated that the major design principle which is common to all engineering disciplines is the relative isolation of one component from other components. Effective decomposition is also a paramount goal in many disciplines, such as mechanical engineering and manufacturing. Clustering techniques have been successfully used in many areas to assist grouping of similar components and/or effective decomposition of a system. For instance, the technique has been used to classify botanical species and mechanical parts. The key concept of clustering is to group similar items together to form a set of clusters, such that intra-cluster similarity is high but inter-cluster similarity is low.

Software engineering is a relatively new area compared to other well established disciplines. This idea of decomposition and clustering has also been intensively discussed in software engineering. Decomposition plays a vital role in system design, as it has tremendous effects on the downstream artifacts and development phases. Software decomposition is often conducted by designers based on their intuition and past experience. While it may work well for some; in reality, however, many systems failed to meet the requirements as a result of poor design.

A key point of an effective clustering or decomposition technique is to maximize cohesion within a module and minimize coupling between modules. Clustering techniques have also been intensively studied in software engineering, particularly in the area of reverse engineering [11-13,16]. Clustering technique can also be used for forwarding engineering early in the life cycle. Inspired by Alexander, Andreu, et al. [3] and Lung, et al. [15] presented applications of clustering to requirement analysis or use cases prioritization. However, the main problem with this idea is that identification of the interdependencies of use cases or requirements is difficult due to ambiguities of the abstract description or understanding at the requirements stage.

The objective of this paper is to apply the clustering technique to support software decomposition based on *attributes* described in the requirements document and to mitigate the problem just stated. Identification of the relationship between requirements and attributes is more effective and efficient. The main idea is to help the designer develop a more robust software architecture or support evaluation of the architecture from the quality aspect at the early stage.

The clustering techniques adopted in this paper are based on numerical taxonomy or hierarchical agglomerative clustering (HAC) method. HAC uses numerical methods to make classifications of components. Each method has potential for revealing insight that may be lacking in other methods and no scientific study has shown that numerical taxonomy is inferior to other more complex multiversity methods [17]. Therefore, we adopt numerical taxonomy mainly because of its conceptual and mathematical simplicity, as will be demonstrated in Section 2.

The paper is organized as follows: Section 2 is a brief overview of the clustering technique adopted for this research. Section 3 highlights some related work. Section 4 demonstrates an industrial application of the technique. Finally, Section 5 is the summary.

2. Clustering

Clustering has been discussed in many disciplines. This paper adopts the hierarchical agglomerative clustering (HAC) method. The main idea behind this approach is to calculate the resemblance coefficients for a number of components based on a set of attributes. Components are the entities that we want to group based on their similarities. Attributes are the properties of the components. For example, components could be mechanical parts; the attributes, their features.

A resemblance coefficient for a given pair of components indicates the degree of similarity between these two components. A resemblance coefficient could be qualitative or quantitative. A qualitative value is a binary representation; e.g., the value is either 0 or 1. A quantitative coefficient measures the literal distance between two components.

There are many HAC algorithms of calculating the resemblance coefficients [2,8,17]. This paper does not discuss those in detail. The idea adopted in this paper is similar to Lung, et al. [16]. Typically, for binary data, HAC methods examine each pair of attributes between two components and keep track of the number of similarities or dissimilarities. A formula is then applied to the calculation of the resemblance coefficient between these two components.

For instance, let a, b, c, and d represent the number of the pair of 1-1, 1-0, 0-1, and 0-0 matches between two components and assume the following component-attribute input data set for an eight-attribute case.

i = {1, 0, 1, 1, 0, 0, 0, 1}
j = {1, 1, 1, 0, 0, 0, 1, 0}
k = {0, 1, 1, 0, 1, 0, 1, 0}

A 1-1 match between two components indicates that they share this specific attribute. Based on the definition, we can obtain for components i and j that a = 2, b = 2, c = 2, and d = 2. Similarly, for components i and k, we obtain that a = 1, b = 3, c = 3, and d = 1; components j and k, a = 3, b = 1, c = 1, and d = 3.

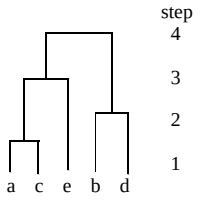
There are various ways to calculate the coefficient. The following are three typical examples:

- Jaccard Coef: $c_{xy} = a / (a+b+c)$
- Simple Matching Coef: $c_{xy} = (a+d) / (a+b+c+d)$
- Sorrenson Coef: $c_{xy} = 2a / (2a+b+c)$

Given a resemblance matrix, the next step is to group similar components. In essence, a clustering method consists of iterative operations that incrementally groups similar components into clusters. The sequence begins with each component in a separate cluster. At each step, the two clusters that are closest to each other are merged and the number of clusters is reduced by one. Once these two clusters have been merged, the resemblance coefficients between the newly formed cluster and the rest of the clusters are updated to reflect their closeness to the new cluster.

An algorithm known as UPGMA (unweighted pair-group method using arithmetic averages) is commonly used to find the average of the resemblance coefficients when two clusters are merged [17]. Two other algorithms have often been studied are SLINK (single linkage) algorithm, CLINK (complete linkage algorithm). The results are usually represented using a dendrogram for HAC. Figure 1 illustrates the concept. In this example, the clustering steps are (a, c), (b, d), ((a, c), e), and finally ((a, c, e), (b, d)). The dendrogram grasps the relative degree of similarity among components or clusters. In general, the lower the level, the more similar the components or clusters are.

Figure 1. An Example of a Dendrogram



3. Related Work

A lot of research on clustering has been conducted in software engineering. Most of these approaches are proposed to support reverse engineering or at the code level. More discussion of the related work can be found in [16]. In this section, we focus on research efforts that are specifically used for high level artifacts and in the forward engineering process.

As pointed out in Section 1, Alexander devised a clustering approach to the building of an Indian village. He demonstrated 141 requirements based on 13 categories (e.g., religion, water, agriculture, and etc.). He then identified the relationships or interactions between requirements. For example, requirement 1 interacts with 8,9,12, 13, … Based on the interactions, the complete list of requirements were decomposed into four major subsets or subsystems, and those four subsets in turn are broken into twelve minor subsets. Finally, he identified an architectural style for each subset. Together, all the subsets form the entire village.

Andreu and Madnick [3] applied the concept to a data base management system. Requirements and their interdependencies were first identified and were converted to a graph problem. Various partitioning alternatives were examined and a quantitative metric was calculated for each alternative. The alternative with the lowest value of coupling was chosen as the optimal partitioning. The system was divided into 5 partitions, each constituting a subsystem in the architecture.

Heyliger [10] proposed to use N square charts to partition a large system. The objective was to refine the design incrementally to maximize cohesion and minimize coupling. He has identified a set of patterns that characterize specific interfaces among system elements. This process, as depicted by the author, is labor intensive and the rearrangement of the elements is a major problem even for small or modest systems.

Lung, et al, [14] reported an experience of building a reusable simulation framework in manufacturing. The approach surveyed over 100 simulation models and identified their features or attributes. Clustering was then conducted based on those features to group similar or related features into a set of generic models. Each generic model was further decomposed into a number of specific models. A framework was then constructed based on the generic and specific models, which could support over 100 simulation models in manufacturing.

Lung, et al., [15] proposed the usage of HAC to use cases and requirements analysis. The concept followed Alexander's idea. However, the main challenge with this approach is the identification of interdependencies between requirements. It is time consuming and labor intensive to conduct the exercise. More importantly, requirements may be ambiguous or too general at this stage. It may be very difficult to clearly identify the relationships between requirements at that stage.

In fact, we applied the approach to a new project in an advanced network communications technology. It was a technology-driven approach for the development of next generation networking equipments, where there were no clear or specific requirements. During the process, we encountered practical problems at times due to the fact that the requirements were specified in a very high-level general fashion. Specifically, they are:

- Difficult to judge if two requirements are actually interdependent, because some parts are not clear;
- Many requirements seemed to be interdependent at that level; and
- Requirements are incomplete; therefore, many interdependencies between requirements may be missing.

The intension of this paper is to simplify the previous process by using requirements and attributes relationships. The main idea is that if there are specific attributes or features that are known or can be identified, it will be easier to identify the interdependencies between requirements indirectly through attributes. Lung, et al. [16] demonstrated the concept in software architecture decomposition. However, that case study was a reverse engineering effort. In this paper, we apply the concept to study a network communication protocol in the forward engineering process.

4. Industrial Application Experience

This section illustrates the application of the clustering to an industrial software system. Section 4.1 briefly describes the problem domain. Section 4.2 demonstrates the experience.

4.1 Background of Case Study

The problem under study is a real network protocol, RSVP-TE [4], in industry. RSVP [6] is a resource reservation control protocol that enables Internet applications to obtain different qualities of service (QoS). RSVP-TE is a signaling protocol that extends the RSVP to support multiple protocol label-switching (MPLS) [18] traffic engineering applications. RSVP-TE provides a mechanism to establish and maintain explicitly routed label switched paths (LSPs).

RSVP-TE has two fundamental messages: Path and Resv (reservation request) messages, which are used to set up LSPs and also used as refresh messages to maintain existing LSPs. Both Path and Resv messages comprise a number of optional objects describing traffic parameters, QoS, and so on. These parameters are used to support advanced traffic engineering. In addition, there are also PathTear (path teardown), ResvTear (reservation teardown), PathErr (path error) and ResvErr (reservation error) messages. The PathTear and ResvTear messages are used to tear down existing LSPs and release reserved resources. The PathErr and ResvErr messages deal with the errors that occur during Path and Resv message-processing, respectively.

The protocol under study is part of a network system which consists of a suite of protocols and base facilities to support communications of network elements.

4.2 Application of Clustering to Software Decomposition

This section demonstrates the application of clustering to software decomposition based on attributes specified in the requirements document. The following outlines the *iterative* process that we adopted:

- Identify functional requirements
- Identify attributes
- Identify the relationship between requirements and attributes
- Apply clustering
- Develop a conceptual architecture based on the decomposition and architectural styles or patterns

Identify Functional Requirements:

The first step identifies critical requirements. In this study, we focus on functional requirements. In RSVP-TE, there are different types of messages. Each message could have a variable number of objects embedded in it. For example, the protocol is primarily used to support network traffic engineering by creating an explicit path from a source to a destination. The information of the

explicit path is captured in the ERO (explicit route object) inside of a Path message described in the RFC. Therefore, it is required to process the ERO. Another object that could be embedded in messages is RRO (record route object), which is used to record the IP addresses at every hop or the label used at every hop. Similarly, there is a need to process the RRO. Table 1 lists twenty-eight important requirements (rows) specified in the RFC document.

Identify Attributes:

The second step is to identify the attributes identified in the requirements, use cases, or scenarios. Attributes, in this context, closely resemble objects or features. For RSVP protocol, some typical attributes are stated in the previous section. Examples include various types of messages, e.g., Path message and Resv message. The attributes are presented in the columns in Table 1.

In addition to the attributes identified in the requirements document, those attributes in other existing subsystems that are closely related to the protocol are also identified. Those attributes are listed from columns 15 to 18. A typical example is that RSVP-TE protocol is on top of the IP (Internet Protocol). In other words, it has to interwork with the IP module. Other subsystems that are related are connection management module, traffic control module, and forwarding engine module.

Identify the Relationship between Requirements and Attributes:

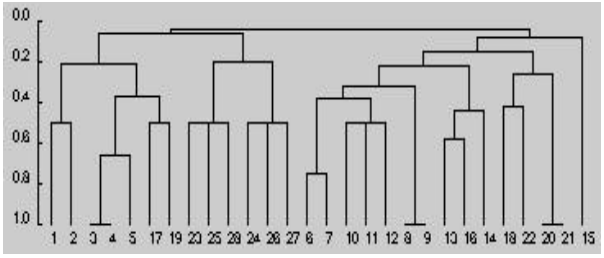
The third step is to identify the relationships between requirements and attributes. As mentioned earlier, this task is primarily used to simplify the step – identification of the relationships between requirements – described in the Section 3. Requirements may be depicted in very general or high-level terms which are difficult to interpret precisely or many requirements may seem to be related. On the other hand, it is easier to check if a requirement is related to some attributes identified in the previous step.

For example, the two objects, RRO and ERO, involved in the first two requirements may seem independent, since they are used for different purposes. However, both of them directly interact with common attributes, e.g., PathMsg and in-Intf as shown in Table 1. Similarly, requirements 6 and 8 are indirectly related through attributes ResvMsg and Out_intf.

Apply Clustering:

The next step is to apply the clustering technique to the requirement-attribute matrix. Selection of an appropriate algorithm may not be trivial, because there is no clear distinction between various resemblance coefficients (Jaccard, Sorenson, and so on) and clustering methods (UPGMA, SLINK, CLINK, and etc.). Figure 2 demonstrates the clustering results using the Jaccard coefficient and the UPGMA algorithm. Results obtained from SLINK and CLINK are not shown due to space limitations. For this particular case, the result obtained from the Sorensan coefficient is very similar to Figure 2, except that the resemblance coefficients are different. Therefore, we are not repeating the diagram.

Figure 2. Decomposition of Requirements into Subsystems Based on Clustering Using UPGMA



In Figure 2, the numbers along the horizontal line correspond to the requirements listed in Table 1. The numbers on the vertical line are resemblance coefficients. The results, reported by the designer, obtained from UPGMA gives the best result. Based on the clustering, there are five main clusters:

- Cluster 1: requirements 1-5 and 17, 19. Requirements 1-5 are directly related to the PathMsg processing; while requirements 17 and 19 are for PErrMsg, which is used to send error code for the PathMsg.
- Cluster 2: requirements 23-28 are related to the functionality of sending messages to its neighbors.
- Cluster 3: requirements 6-12. Those requirements are related to ResvMsg processing.
- Cluster 4: requirements 13, 14 and 16. This group is used to tear down LSPs.
- Cluster 5: requirements 18, 20, 21, 22. Those requirements are related to RErr processing.

Requirement 15 is not grouped with other requirements clearly. It has to do with cluster 4 which processes teardown. It is not uncommon to see some components that are not clustered clearly with other components. In such cases, domain knowledge plays a vital role to manually group those with other clusters. As stated earlier, the process could be iterated if necessary based on the input data and validation of the results. For instance, requirements 17 and 19 in cluster 1 could be further divided into a separate cluster that handles specifically for the error processing for PathMsg. The design decision will be made by the designers. Nevertheless, the results could be used as a guideline to facilitate decomposition. For this case study, the requirements can be decomposed into five major subsystems as outlined above.

Table 1. Relationships between Requirements and Attributes

	Attributes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		Path Msg	Resv Msg	PTear Msg	RTear Msg	PErr Msg	RErr Msg	PSB	RSB	TCSB	BSB	In_ intf	Out_ intf	Nhop	Phop	IP_ mod	CM_ mod	TC_ mod	FE_ mod
1	Process RRO	x	x									x	x						
2	Process ERO	x										x	x	x			x		
3	Create PSB	x						x				x							
4	Update PSB	x						x				x							
5	Build PathMsg	x						x											
6	Create RSB		x						x				x						x
7	Update RSB		x						x				x						
8	Create TCSB		x							x			x					x	
9	Update TCSB		x							x			x					x	
10	Reserve resource		x						x	x									
11	Merge flowspec		x					x		x									
12	Build ResvMsg		x					x	x										
13	Time out PSB							x	x	x	x								
14	Process PTearMsg			x				x	x	x	x			x			x	x	x
15	Time out RSB				x				x										
16	Process RTearMsg		x		x			x	x	x	x							x	x
17	Generate PErr	x				x													
18	Generate RErr		x				x												
19	Process PErrMsg	x		x		x		x											
20	Create BSB						x				x								
21	Update BSB						x				x								
22	Process RErrMsg		x			x	x	x	x		x								
23	Send PathMsg	x												x		x			
24	Send ResvMsg		x												x	x			
25	Forward PTearMsg			x										x		x			
26	Forward RTearMsg				x										x	x			
27	Forward PErrMsg					x									x	x			
28	Forward RErrMsg						x							x		x			

PSB: path state block, RSB: reservation state block, TCSB: traffic control state block, BSB: blockade state block
In_intf: incoming interface, Out_intf: outgoing interface
Nhop: next hop, Phop:previous hop
IP_mod: IP module, CM_mod: Connection Manager Module, TC_mod: traffic control module, FE_mod: forwarding engine module

For this specific example, UPGMA demonstrates better results based on the designer's judgment. The tool can generate clusters automatically if the user specifies the number of clusters or threshold values of the resemblance coefficients. However, we recommend that the designer make the final decision by examining the overall clustering result, since it will play an influential role for the design.

Develop a Conceptual Architecture

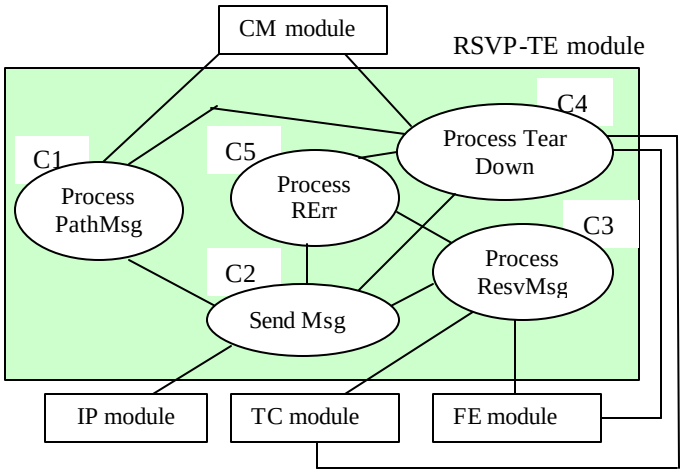
This step involves more knowledge in software architecture and design. Decomposition obtained from the previous step and architectural styles or patterns are useful in this step. A conceptual architecture in this context is similar to that described in [5]. It is not a final system, but a representation of high-level design with critical components and connectors. For RSVP-TE design, the conceptual architecture is shown in Figure 3. There are five main components or clusters, C1, C2, …, C5, corresponding to those stated above.

Iterative refinement is needed for the conceptual architecture by adding some attributes listed in Table 1 or more connections of specific types between components. For instance, shared data structures, PSB, RSB, and etc. could be added to the diagram. In addition, input/output queues can be inserted for incoming/outgoing interfaces. For instance, if simply based on requirements, PathMsg (C1) and ResvMsg (C3) may seem independent, since they are used in opposite directions in the protocol. But they are related through some attributes after further analysis. In fact, each will trigger the generation of the other message type when the message reaches the end router. The connection between C1 and C3 will then be added to the design. The iteration process will make the conceptual architecture more concrete.

Another point that is worth mentioning is that architectural styles or patterns [9,19] may be identified for the target system during the analysis. This, however, requires knowledge in both the problem and the solution domains. For this particular example, no specific style

or pattern was selected. In other cases, we have identified suitable architectural patterns for two telecommunications systems. One was based on the Observer [9] pattern; the other one was derived from Half-Sync and Half-Async pattern [19] before design.

Figure 3. Conceptual Architecture for RSVP-TE Based on Clustering Analysis



5. Conclusion

We have presented an approach to support software architecture decomposition using clustering of requirements and attributes. The approach was extended from previous research which identified the relationships between requirements. Based on our experience, identification of relationships between requirements may be difficult and confusing early in the life cycle. The relationships between requirements and attributes can be identified more easily.

We have applied the technique to a real system in network protocol. We have also compared different clustering approaches: UPGMA, SLINK, and CLINK. In our study, UPGMA generated the best result based on the designer's evaluation. Also, we studied Jaccard and Sorensan algorithms for resemblance coefficients. The results were very similar for this case study.

The clustering results are useful to support the architect or designer for decomposition. The relationship between requirements and attributes, as shown in Table 1, are also useful, because it reveals how requirements relate to subsystems through attributes or objects. Reading across the row, we can associate the attributes made up the requirements. Reading down the column, we can find out which requirements the attribute participates in [20].

Currently, we are working on the integration of the clustering tool with a commercial requirements management tool, called Telelogic DOORS [7]. The tool can capture relationships among various requirements as well as between requirements and attributes.

Acknowledgements

We would like to thank M. Zaid and R. Crawhall of NCIT, Ottawa and R. Munikoti and K. Kalaichelvan of EION, for supporting this work. We also thank A. Srinivasan and P. Dhakal of EION for supporting RSVP-TE implementation.

References:

[1] C. Alexander, *Notes on the Synthesis of Form*, Harvard University Press, 1964.

[2] M.R. Anderberg, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[3] R.C. Andreu and S.E. Madnick, *A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation*, TR CISR 32, MIT Sloan School of Management, 1977.

[4] D. Awduche, et al., *RSVP-TE: Extensions to RSVP for LSP Tunnels*, IETF RFC 3209, 2001.

[5] L. Bass, M. Klein, and F. Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method", *Proc. of the 4th Int'l Workshop on Software Product Family Eng.*, 2001, pp. 169-186.

[6] Braden, R., et al., *Resource ReSerVation Protocol (RSVP)*, RFC 2205, 1997.

[7] Telelogic DOORS, http://www.telelogic.com/products/doorsers/doors/index.cfm, Feb 2005.

[8] B. Everitt, *Cluster Analysis*, Heinermann Educational Books, Ltd., London, 1980.

[9] E. Gamma, et al., *Design Patterns*, Addison, 1995.

[10] G. Heyliger, "Coupling", *Encyclopedia of Software Engineering*, J. Marciniak (ed.), 1994.

[11] *Int'l Conf. on Software Maintenance.*

[12] *Int'l Working Conf on Program Comprehension.*

[13] *Int'l Working Conf on Reverse Engineering.*

[14] C.-H. Lung, et al., "Computer Simulation Software Reuse by Generic/Specific Domain Modeling Approach", *Int'l J. of Software Eng. and Knowledge Eng.*, vol. 4, no. 1, March 1994, pp. 81-102.

[15] C.-H. Lung, A. Nandi, and M. Zaman, "Applications of Clustering to Early Software Life Cycle Phases", *Proc. of Int'l Conf. on Software Eng. Research and Practice*, June, 2002, pp. 625-631.

[16] C.-H. Lung, M. Zaman, and A. Nandi, "Applying Clustering Techniques to Software Architecture Partitioning, Recovery and Restructuring", *J. of Systems and Software*, vol. 73, no. 2, Oct 2004, pp. 227-244.

[17] H. C. Romesburg, *Cluster Analysis for Researchers*, Krieger, Malabar, Florida, 1990.

[18] E. Rosen, et al., *Multiprotocol Label Switching Architecture*, IETF RFC 3031, 2001.

[19] D. Schmidt, et. al., *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*, John Wiley and Sons, 2000.

[20] G. Schneider and J. P. Winters, *Applying Use Cases A Practical Guide*, Addison-Wesley, 2001.