International Journal of Software Engineering
and Knowledge Engineering
Vol. 17, No. 5 (2007) 599–613
© World Scientific Publishing Company

SOFTWARE ARCHITECTURE DECOMPOSITION USING ATTRIBUTES

CHUNG-HORNG LUNG* and XIA XU†
*Department of Systems and Computer Engineering,
Carleton University, Ottawa, Ontario, K1S 5B6, Canada*
* *chlung@sce.carleton.ca*
† *xiaxu@sce.carleton.ca*

MARZIA ZAMAN
Cistel Technology, Ottawa, Ontario, K2E 7V7, Canada
marzia@cistel.com

Received 31 March 2006
Accepted 16 October 2006

Software architectural design has an enormous effect on downstream software artifacts. Decomposition of function 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 system 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 experiments in applying the approach to an industrial communication protocol software system and comparing several clustering algorithms. The result obtained from WPGMA (weighted pair-group method using arithmetic averages) shows closer resemblance than other clustering methods to the one developed by the designer.

Keywords: Software architecture; requirements; attributes; decomposition; clustering.

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 dividing a system into subsystems with the 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 components 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. In fact, classification or clustering analysis is one of the most fundamental methods used in science and engineering to facilitate better understanding of the observations and the subsequent construction of complex knowledge structures from features and component clusters. 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 (cohesion) is high but inter-cluster similarity (coupling) is low.

Software engineering is a relatively new area compared to other well established disciplines. This idea of decomposition and clustering has also been a topic of interest 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 experiences. While it may work well for some; in reality, however, some 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 extensively studied in software engineering, particularly in the area of reverse engineering [20, 22, 27, 30]; and it has often been a topic of interest in closely related conferences on reverse engineering [12–14]. Partitioning or clustering technique can also be used for forwarding engineering early in the life cycle, as postulated by Alexander [1], and Ulrich and Eppinger [29] in other engineering disciplines. Inspired by Alexander, Andreu [2] and Lung *et al.* [19] presented applications of clustering to software engineering, e.g., use cases prioritization or requirement analysis. However, the main problem with this idea is that identification of interdependency of use cases or requirements is difficult due to ambiguities of the abstract description or understanding of requirements at the early 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. A requirement attribute is a concrete descriptive characteristic or object associated with a requirement. Requirement attributes can be:

(i) items directly specified in the requirements, e.g., a specific type of message used in a network protocol;
(ii) user-defined attributes (defined by the designer), e.g., a data structure or a resource; or
(iii) system attributes that relate to the operating environments or other systems.

Attributes generally are basic elements that relate to a requirement. In other words, a requirement usually consists of multiple attributes. Therefore, identification of the relationship between requirements and attributes is more effective and efficient than identification of the relationship between requirements at this stage. 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. Section 2 describes those algorithms adopted in our experiment. 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 [23]. Therefore, we adopt numerical taxonomy mainly because of its conceptual and mathematical simplicity, as will be demonstrated in Sec. 2.

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

2. Clustering

Clustering has been discussed in many disciplines, including software engineering and data mining in knowledge engineering. This paper adopts the numerical taxonomy or 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 mammals; the attributes, their dental formulas. A further example: the components could be mechanical parts; the attributes, their features. The components and attributes can be many things in various areas, to which clustering analysis can be applied.

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 could be a binary representation; e.g., the value is either 0 or 1. A quantitative coefficient measures the literal distance between two components when they are viewed as points in a two-dimensional array formed by the input attributes.

Many HAC algorithms exist for calculating the resemblance coefficients [9, 23]. This paper does not discuss those in detail. 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 these two components share this specific attribute in common. 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 Coefficient: $c_{xy} = a/(a + b + c)$
- Simple Matching Coefficient: $c_{xy} = (a + d)/(a + b + c + d)$
- Sorenson Coefficient: $c_{xy} = 2a/(2a + b + c)$.

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 fused, the resemblance coefficients between this newly formed cluster and the rest of the clusters are updated to reflect their closeness to the new cluster.

Average clustering methods have often been used. UPGMA (unweighted pair-group method using arithmetic averages) and WPGMA (weighted PGMA) are commonly adopted to find the average of the resemblance coefficients when two clusters are merged [23]. In UPGMA, averages are weighted by the number of components in each cluster. In other words, it weights each item in the candidate cluster equally. For instance, let $rc(A, B)$ represent the resemblance coefficient between clusters A and B, and $n(A)$ represent the number of components in cluster A. Based on the UPGMA method, if two clusters X and Y are to be merged, then the resemblance coefficient between cluster A and the newly formed cluster (X, Y) is $(n(X)^*rc(A, X) + n(Y)^*rc(A, Y))/(n(X) + n(Y))$.

WPGMA differs from UPGMA by weighting the component most recently included to a cluster equal with all previous components without considering the number of components in clusters X and Y. In other words, WPGMA down-weights the larger group by giving equal weights to the two branches of the dendrogram that are to be merged. WPGMA is calculated using $(rc(A, X) + rc(A, Y))/2$.

Two other algorithms which have often been studied are SLINK (single linkage) algorithm and CLINK (complete linkage algorithm). SLINK is also called the nearest neighbor method. It defines the similarity measure between two clusters as

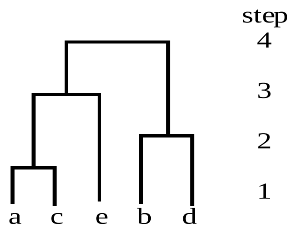


Fig. 1. An example of a dendrogram.

the maximum resemblance coefficient among all pair entities in the two clusters. CLINK, also called the furthest neighbor method, on the other hand, defines the similarity measure between two clusters as the minimum resemblance coefficient among all pair entities in the two clusters.

The clustering 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.

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 in reverse engineering can be found in [20, 22]. In this section, we focus on research efforts that are specifically used for high level artifacts and in the forward engineering process. In addition, coupling has a vital role in software architecture and design or engineering design in general. Many papers either have addressed the idea of low-coupling or devised a method to reduce coupling. For instance, [6, 15, 21, 26] discussed the principle of low coupling for system or software architecture; [1, 29] presented approaches to reducing coupling for engineering design or product development. Good design typically exhibits low coupling between modules, so that modules are more self-contained that can be tackled separately, and are more portable or reusable. High coupling often implies more complex products and greater unexpected side effects due to changes. Our approach also follows the same school of thought. In other words, our aim is to support software architecture development by grouping closely related requirements so that they can be realized by one module. The following paragraphs in this section illustrate related research that has been used to support system decomposition specifically from the forwarding engineering perspective.

As pointed out in Sec. 1, Alexander devised a partitioning approach to the building of an Indian village. He demonstrated 141 requirements based on 13 categories (e.g., religion, water, agriculture, etc.). He then identified the relationships or interactions between requirements. For example, requirement 1 interacts with requirements 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 [2] applied a similar concept to a database management system to maximize system strength and minimize coupling. 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 five partitions, each constituting a subsystem in the architectural design.

Heyliger [11], on the other hand, proposed to use N square charts to partition a large system into subsystems. The objective of Heylinger's approach was to incrementally refine the design 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.* [17] reported an experience of building a reusable simulation framework in manufacturing. The approach surveyed over 100 simulation models used in manufacturing 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 be used to support over 100 simulation models proposed for various purposes in manufacturing.

Lung *et al.* [19] proposed the usage of HAC to use cases and requirements analysis. The approach 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, too general, or described in various levels of details at this stage. It may be difficult to clearly identify the relationships between requirements at this level.

In fact, we applied the approach proposed by Lung *et al.* [19] to a new project in an advanced network communications technology. It was a technology-driven approach for the development of the next generation networking equipment, where there were no clear or specific user requirements. During the process, we encountered some practical problems at times due to the fact that the requirements are specified at a very high-level general fashion. Specifically, those problems include:

- It was difficult to judge if two requirements were actually interdependent because some parts were not clear;
- Many requirements seemed to be interdependent at that level; and
- Requirements were incomplete; therefore, many interdependencies between requirements may be uncertain or missing.

The aim 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.* [20] demonstrated the concept in software architecture decomposition. In that approach, a set of functional components and a list of features or attributes were identified. System decomposition can then be conduced based on the component-attribute clustering. However, that case study was a reverse engineering effort. In this paper, we apply the concept to study a network traffic engineering communication protocol in the forward engineering process.

Bachmann *et al.* [4] presented an experience report to assist architectural design with an emphasis on modifiability. A list of responsibilities and their relationships were identified. Conceptually, assigning responsibilities to modules in their method is similar to our approach. However, they adopted a cost model which consists of rules to assist the architect. Responsibilities with a high probability of propagation of modification are allocated to the same module; otherwise, responsibilities are assigned to different modules. Our approach, on the other hand, performs clustering of requirements based on common requirement attributes.

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 [3], in the telecommunication industry. RSVP [7] 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) [24] traffic engineering applications. RSVP-TE provides a mechanism to establish and maintain explicitly routed label switched paths (LSPs), with or without resource reservation.

RSVP-TE has two fundamental messages: the Path message and the Resv (reservation request) message, which are used to set up LSPs and are 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. The requirements are specified in IETF (Internet Engineering Task Force) RFC (Request for Comments) [3, 24].

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 process that we adopted:

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

Note that the steps do not have to be carried out in a strict sequential fashion. Rather, the process is iterative in nature, since requirements are rarely complete at the early stage in practice. An iterative plan for requirements elicitation and analysis is always necessary. Each step is elaborated in more detail as follows.

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 28 important requirements (rows) specified in the RFC document.

Identify attributes

The second step is to identify the attributes described in the requirements, use cases, or scenarios. Attributes, in this context, closely resemble objects or features described in the requirements. As stated in Sec. 1, a requirement attribute can be:

(i) items, typically objects, explicitly specified in the requirements;
(ii) user-defined attributes (defined by the designer); or
(iii) system attributes that relate to the operating environments or other systems.

For RSVP protocol, some typical attributes are explicitly described in requirements document as described in the previous section. Examples include various types of messages, e.g., Path message (PathMsg) and Resv message (ResvMsg). Some attributes are user-defined to support a functional requirement or the attributes identified above. Examples for RSVP protocol include data structures related to those message types such as path state block (PSB) and reservation state block (RSB). The attributes are presented in the columns in Table 1.

In addition to the attributes identified in the requirements document or user-defined attributes, the attributes that are closely related to the supporting environments or need to interwork with other sub-systems (protocols) are also uncovered. Those attributes are listed from columns 15 to 18 in Table 1. A typical example is that RSVP-TE protocol is on top of the IP (Internet Protocol) layer. In other words, it has to interwork with the IP module. Other subsystems that are closely 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 Sec. 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 specific attributes identified in the previous step.

For instance, 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 (incoming interface) as shown in Table 1. Similarly, requirements 6 and 8 are indirectly related through attributes ResvMsg and Out_intf (outgoing interface).

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 advantage among various resemblance coefficients (Jaccard, Sorenson, and so on) and clustering methods (UPGMA, WPGMA, SLINK, CLINK). Therefore, we applied all those methods and compared the results against the actual design. Figure 2 demonstrates the clustering results using the Jaccard coefficient as explained in Sec. 2. For this particular case, the result obtained from the Sorenson coefficient is very similar to Fig. 2, except that the resemblance coefficients are different. The results obtained from Jaccard and Sorenson coefficients are better than that of the Simple Matching coefficient. Therefore, we are not showing the diagrams for Sorenson and Simple Matching coefficients.

Table 1. Relationships between requirements and attributes.

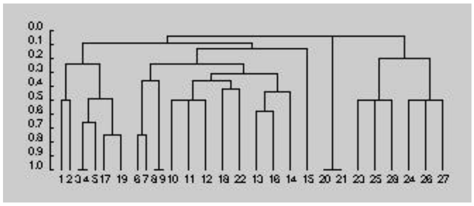
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Requirements	Attributes	PathMsg	ResvMsg	PTearMsg	RTearMsg	PErrMsg	RErrMsg	PSB	RSB	TCSB	BSB	In_intf	Out_intf	Nhop	Phop	IP_mod	CM_mod	TC_mod	FE_mod
1	Process RRO	×	×									×	×						
2	Process ERO	×										×	×	×			×		
3	Create PSB	×						×				×							
4	Update PSB	×						×				×							
5	Build PathMsg	×						×											
6	Create RSB		×						×				×						×
7	Update RSB		×						×				×						
8	Create TCSB		×							×			×					×	
9	Update TCSB		×							×			×					×	
10	Reserve resource		×						×	×									
11	Merge flowspec		×					×		×									
12	Build ResvMsg		×					×	×										
13	Time out PSB							×	×	×	×								
14	Process PTearMsg			×				×	×	×	×			×			×	×	×
15	Time out RSB				×				×										
16	Process RTearMsg		×		×			×	×	×	×							×	×
17	Generate PErr	×				×													
18	Generate RErr		×				×												
19	Process PErrMsg	×		×		×		×											
20	Create BSB						×				×								
21	Update BSB						×				×								
22	Process RErrMsg		×			×	×	×	×		×								
23	Send PathMsg	×												×	×	×			
24	Send ResvMsg		×												×	×			
25	Forward PTearMsg			×										×		×			
26	Forward RTearMsg				×										×	×			
27	Forward PErrMsg					×									×	×			
28	Forward RErrMsg						×							×		×			

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

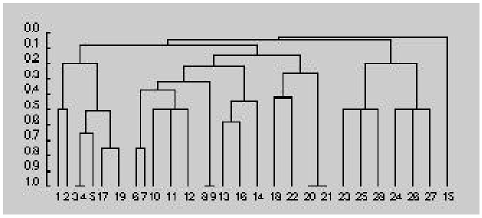
In Fig. 2, the numbers along the horizontal line correspond to the requirements listed in Table 1. The numbers on the vertical line are resemblance coefficients. The results were evaluated and compared with the actual design by the designer. Based on the assessment, the results obtained from the WPGMA, as described in Sec. 2, gives the best result. According to 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

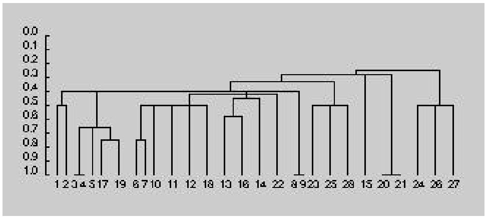
UPGMA



WPGMA



SLINK



CLINK

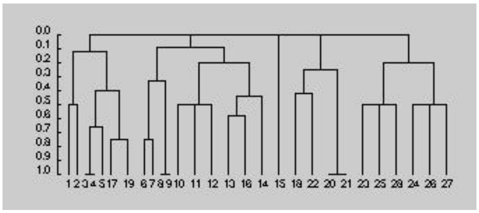


Fig. 2. Decomposition of requirements into subsystems based on clustering algorithms.

is used to send error code for the PathMsg.

- Cluster 2: requirements 23–28 (not necessarily in sequence as shown in the figure) 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, 20–22. 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

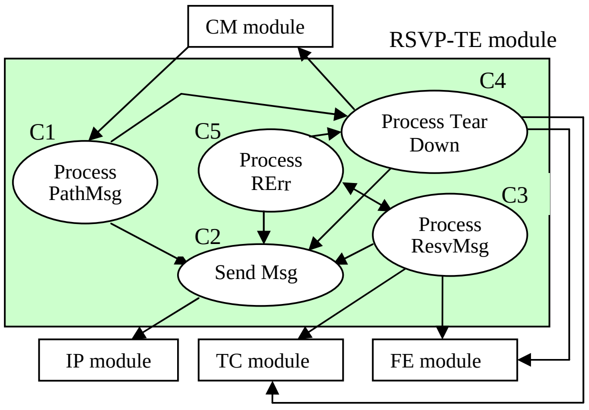


Fig. 3. Conceptual architecture for RSVP-TE based on clustering analysis.

that are not clearly clustered 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.

Figure 2 also demonstrates the results obtained from the UPGMA, WPGMA, SLINK and CLINK. For this specific example, the WPGMA demonstrates better results inspected visually by the designer. UPGMA generates similar result, except requirements 18, 20, 21, and 22 are mixed with cluster 3. 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 makes the final decision by examining the overall clustering result, since there are other factors or quality attributes to consider for a design in practice.

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 a representation of high-level design with critical components and connectors. Other quality issues, such as performance and reliability, could be considered for further sensitivity [18] or tradeoff analysis [16]. For RSVP-TE design, the conceptual architecture is shown in Fig. 3. There are five main components or clusters, C1, C2, . . . , C5, corresponding to those stated above.

Iterative refinement is needed for the conceptual architecture if more attributes are added or more connections between components as requirements are better understood or more requirements are captured. For instance, shared data structures, PSB, RSB, etc. could be added to the diagram. In addition, input and output queues can be inserted to serve the purpose of incoming and outgoing interfaces. For instance, based simply 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.

This technique is not advocated to drive the architecture design; rather, it is used to complement the manual design process by providing a different view. The technique is also useful to support architecture evaluation by identifying potential areas that demonstrate high coupling or inappropriate design. Those areas could be incrementally improved, made informed tradeoffs early, or simply brought to more attention. We make no claim that the clustering result is absolutely correct. Similar to [4], the conceptual architecture can be viewed as a "first cut". As stated, incremental and/or iterative development is advocated.

Another point worth mentioning is that architectural styles [28] or patterns [10, 25] may be identified for the target system based on the analysis. This, however, requires knowledge in both the problem domain and the solution domain. For this particular example, no specific style or pattern was selected. In other cases, we have identified architectural patterns that fit the target system in two telecommunications systems. One system or subsystem was based on the Observer [10] pattern; the other architectural structure was derived from Half-Sync/Half-Async pattern [25].

5. Conclusion and Future Directions

We have presented an approach to support software architecture decomposition using clustering of functional 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. On the other hand, the relationships between requirements and attributes can be identified more easily.

We have applied the technique to a real system in network protocol design as a case study. Three algorithms for resemblance coefficients calculation, Simple Matching, Jaccard and Sorensan, were studied. The results obtained from the last two coefficients were very similar and were better than that of the Simple Matching approach for our problem. In addition, we have also compared four different clustering approaches: UPGMA, WPGMA, SLINK, and CLINK. For this particular study, WPGMA generated the best result and CLINK, on the other hand, did not produce good result based on the designer's evaluation. This paper is an initial attempt for the comparison. More empirical investigations need to be conducted to further compare these clustering algorithms and validate the results. The comparison results may provide insights on which clustering algorithm(s) to choose in the future as we understand those algorithms better.

The clustering results are useful to support the architect or designer for system decomposition which plays a crucial role for downstream product development. The relationship between requirements and attributes, as shown in Table 1, is also useful because it reveals how requirements relate to subsystems through attributes or objects. Reading across the row, we can associate that the attributes made up the requirements. Reading down the column, we can find out which requirements the attribute participates in [26].

Currently, we are working with a commercial tool, Telelogic DOORS [8], for requirements management. This tool can capture relationships among various requirements as well as between requirements and attributes. It also provides traceability. We are working on the tool integration with our clustering tool. It could also be integrated with other requirements management tools that support document processing and integration with spreadsheet.

Another area that we are looking at is to consider non-functional requirements. Non-functional quality requirements have been advocated as important drivers for software architectures [5]. In the case of RSVP-TE, performance is the most important one and almost every functional requirement is closely related to it. Thus, including it or not does not make much difference. However, various non-functional requirements are needed to consider in general, in which case, including those in the clustering analysis may reveal more insights.

References

1. C. Alexander, *Notes on the Synthesis of Form* (Harvard University Press, Cambridge, MA, 1964).
2. R. C. Andreu and S. E. Madnick, *A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation*, Technical Report CISR 32, MIT Sloan School of Management, 1977.
3. D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, *RSVP-TE: Extensions to RSVP for LSP Tunnels*, IETF RFC 3209, 2001.
4. F. Bachmann, L. Bass, M. Klein, and C. Shelton, Experience using an expert system to assist an architect in designing for modifiability, in *Proc. 4th Working IEEE/IFIP Conf. on Software Architecture*, 2004, pp. 281–284.
5. L. Bass, M. Klein, and F. Bachmann, Quality attribute design primitives and the attribute driven design method, in *Proc. 4th Int. Workshop on Software Product Family Eng.*, 2001, pp. 169–186.
6. L. Bernstein and C. M. Yuhas, *Trustworthy Systems through Quantitative Software Engineering* (Wiley-IEEE Computer Society Press, 2005).
7. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *Resource ReSerVation Protocol* (*RSVP*), RFC 2205, 1997.
8. DOORS, Telelogic, http://www.telelogic.com/products/doorsers/doors/index.cfm, last accessed in Aug. 2006.
9. B. Everitt, *Cluster Analysis* (Heinermann Educational Books, London, 1980).

10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns* (Addison-Wesley, 1995).
11. G. Heyliger, Coupling, *Encyclopedia of Software Engineering*, J. Marciniak (ed.), 1994.
12. *Proceedings of the IEEE Int Conf. on Software Maintenance.*
13. *Proceedings of IEEE Int. Conf. on Program Comprehension.*
14. *Proceedings of IEEE Int. Working Conf. on Reverse Engineering.*
15. R. Kazman, G. Abowd, L. Bass, and P. Clements, Scenario-based analysis of software architecture, *IEEE Software* **13**(6) (1996) 47–55.
16. R. Kazman, M. Klein, and P. Clements, *ATAM: Method for Architecture Evaluation*, Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute.
17. C.-H. Lung, J. K. Cochran, G. T. Mackulak, and J. E. Urban, Computer simulation software reuse by generic/specific domain modeling approach, *Int. J. Software Eng. Knowledge Eng.* **4**(1) (1994) 81–102.
18. C.-H. Lung and K. Kalaichelvan, A quantitative approach to software architecture sensitivity analysis, *Int. J. Software Eng. Knowledge Eng.* **10**(1) (2000) 97–114.
19. C.-H. Lung, A. Nandi, and M. Zaman, Applications of clustering to early software life cycle phases, in *Proc. Int. Conf. on Software Engineering Research and Practice (SERP)*, June, 2002, pp. 625–631.
20. C.-H. Lung, M. Zaman, and A. Nandi, Applying clustering techniques to software architecture partitioning, recovery and restructuring, *J. Systems and Software* **73**(2) (2004) 227–244.
21. M. W. Maier and E. Rechtin, *The Art of Systems Architecting*, 2nd ed. (CRC, 2000).
22. B. S. Michell and S. Mancoridis, On the automatic modularization of software systems using the bunch tool, *IEEE Trans. on Software Eng.* **32**(3) (2006) 193–208.
23. H. C. Romesburg, *Cluster Analysis for Researchers* (Krieger, Malabar, Florida, 1990).
24. E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol Label Switching Architecture*, IETF RFC 3031, 2001.
25. D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture*, Vol. 2, *Patterns for Concurrent and Networked Objects* (John Wiley and Sons, 2000).
26. G. Schneider and J. P. Winters, *Applying Use Cases A Practical Guide* (Addison-Wesley, 2001).
27. R. W. Selby and R. M. Reimer, *Interconnectivity Analysis for Large Software Systems*, Technical Report, UCIrv-95-PROC-CSS-001, Univ. California at Irvine, 1995.
28. M. Shaw and D. Garlan, *Software Architecture Perspectives on an Emerging Discipline* (Prentice Hall, 1996).
29. K. T. Ulrich and S. D. Eppinger, *Product Design and Development*, 3rd ed. (McGraw Hill, New York, 2003).
30. T. A. Wiggerts, Using clustering algorithms in legacy systems modularization, in *Proc. 4th Working Conf. on Reverse Engineering*, 1997, pp. 33–43.