Software Component Identification and Composition

Presented By

Anshuman Sekhar Dash

M.Tech. 3rd Sem

Information Security

Reg No: 2020IS04

Computer Science & Engineering Department

Motilal Nehru National Institute of Technology

Allahabad, Prayagraj

October 28, 2021

Outline

1. Introduction
2. Software Component Problem
3. Previous Approaches
4. Formal Method Approach
5. Conclusion
6. References

Introduction I

- From the last 10 years use of computer software has entered into every part of economy which has in turn generated new demand from the software industry to develop reliable and cost effective software quickly and efficiently.
- Modern software systems are becoming more and more large scale which further increases the development cost , decreases the productivity and increases the cost of maintenance.
- A software system need not be developed all the way from scratch and instead the software system can be developed by piecing together small pieces of software from previously developed software system and assembling them to make a final product.
- In the context of CBSE a component will be defined as a unit of composition with contractually specified interfaces and explicit context dependencies only.

Introduction II

- A software component can be deployed independently and is subject to composition by third parties.
- The primary role of CBSE is to direct making of software system as stitching of parts or components , development of the individual components as reusable entities and maintenance of the system developed with timely replacement of components[1].

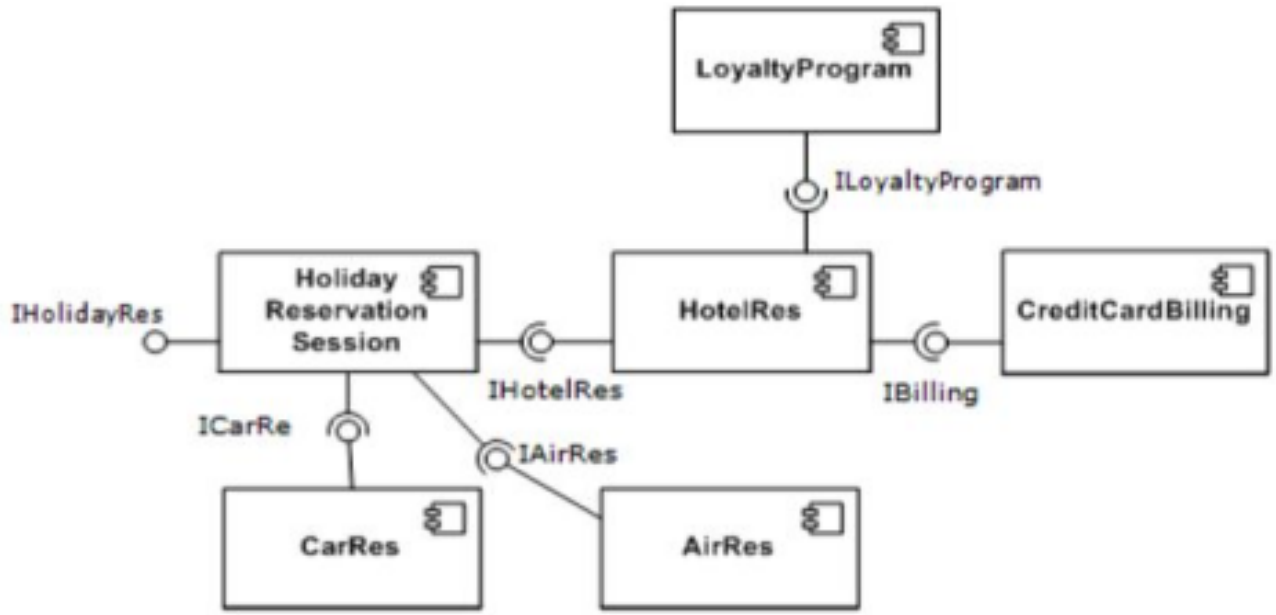

Figure: Software Components Example

Software Component Identification Problem

- Component identification during the software design phase is a process in which the functionalities of a given system are partitioned into non intersecting logical components to provide the starting point for designing software architecture [2][3].
- It is a complex problem and depends on different parameters with variable values and limiting conditions.
- This component identification problem belongs to Non-polynomial time problem which can only be solved by exhaustive search of all the different possibilities[4].

Software Composition Problem

- Effective reuse of software components not only depends on identification of software components but also the ways to merge the components.
- The other problem in CBSE is that of Component composition.
- Presently object oriented programming languages provide ad-hoc mechanisms for composing software using components.
- By providing a proper semantic foundation definition we will be able to cleanly integrate all the requirements of the software system into one system[5][6].

Table: Artefacts used

Artefact	References
Static/dynamic Call graphs	[7]
Use Cases	[8][9][10]
Source Code	[7]
Sequence Diagram	[10][4]
Relationship between classes	[9]
Business Model Documents	[7][11]
Class Diagrams	[9][10][12]
Requirements Definition	[13]
Collaboration Diagram	[11]

Table: Component Identification Techniques

Methods	References
High cohesion and Low coupling	[13]
Use Case	[14]
Fuzzy based analysis	[15]
Artificial Neural Network	[8]
Genetic Algorithm	[9]
Particle Swarm Optimization	[16][17]
Other Methods	
CRUD Based	[10]
Graph Partitioning	[7]

Clustering Approaches

1. **High cohesion and Low Coupling:**-[13] used high cohesion and low coupling metric to cluster classes into logical components.Key classes were selected by an expert.Other classes were assigned any one among these classes based on level of dependency.It was very much dependent on the expert since key classes were required to be chosen
2. **Use Case:-** Use case,object models and collaboration diagrams were used to identify components.Clustering was performed on functional dependencies of use cases.In [18][19] all the different types of clustering algorithms present were used and compared.It required weighting by an expert[20].

Artificial Neural Network I

- [8] used designing phase use case diagrams and components was generated using neural network that was self organized.
- The criterion for neural network was cohesion and coupling.
- The number of neurons required initially is static and any change in use case the process needs to be repeated from starting.

Genetic Algorithm

- In genetic algorithm both the cohesion and coupling between two components is combined into a single objective function.
- This metric is then used to reject components whose fitness value is low[9].
- Genetic algorithm generated good results in small scale systems but failed to reciprocate good results in large scale system because in large scale systems probability of getting a good initial population is very low[21].

Particle Swarm Optimization

- PSO is a computational and artificial intelligence technique[17].
- The important factor of PSO which distinguishes it from other is its fast convergence in compression with other algorithms of global optimization.
- Feature vector is used to represent a class.
- Shortcomings of this approach is that it can only be used with object oriented systems.

Formal Concept Analysis

- FCA is a framework which uses mathematics that is used to explain and analyse data and their relationships[22].
- [11] uses a framework based on FCA that divides class diagram into logical components which is like that of clustering techniques.Adding on this paper [11] used a new method that was based on fuzzy FCA.Dispersion and distance concept was used to calculate cohesion and coupling metrics.
- Fuzzy clustering is useful when the boundaries between clusters are not well separated.
- Disadvantage of this method was that the threshold for dispersion and distance was set manually and it impacted heavily on the result, depends on the number of clusters and is susceptible to outliers in the components data.

Other Methods

1. **Graph Partitioning:-**In paper [7] object models was related with vertices and edges of graph. In graph partitioning technique the business model or the domain model is mapped into a weighted graph by an expert.This weighted graph is then used to apply graph segmentation technique .weights are manually assigned by experts.The graph is divided using heuristic from graph theories.The one disadvantage of this method is that it depends a lot on the weights given by the expert.The paper didn't justify the conclusion.
2. **CRUD Based:-**The 4 relationship used in this method is Create, Read , Write ,Update and Delete with their priority set as $C > D > U > R$ and these relationship are used to give a semantic relationship.This relationship is then transformed into a matrix .[10] Used business events as inputs.

Constructing Software Library using formal Methods

We selected [23] and [2] as the base paper for our research work.[23] used formal language predicate logic to specify software components which can be used for both object oriented and function based systems.Since it only requires the source code to specify software components no domain level elements are required from the software system.
Steps used in identification of software components using formal methods:-

1. Formal Specification of Software Components
2. Generating Lower Level Hierarchy
3. Measuring Similarity
4. Higher Level Hierarchy using Clustering

Formal Specification of Software components

- Interface represents syntactic specification of the method.
- Type declarations represents the type of input output and local variables used.
- Precondition describes the condition of the variables before the method is called.
- Post condition describes the condition of the variables after the method is called.

type Array:
  method ‖E‖ assign_element: Array × E → Array **is**
    in(s: Array, e: E)
    local()
    out(s': Array)
    { **pre**: true }
    { **post**: $len(s') = len(s) + 1 \wedge s'(len(s')) = e$
    % len: index of the last element in array with a value
    }
  method sort: Array → Array **is**
    in(s: Array)
    local(i, j, min, max: Int)
    out(s': Array)
    { **pre**: true }
    { **post**: $s' = \text{permutation}(s) \wedge$
      $(\forall\, i : \text{min} \leq \text{i} \leq \text{max} :: (\forall\, \text{j} : \text{min} \leq j < i :: s'(j) \leq s'(i)))$
    }
  method ‖E‖ last_element: Array → E **is**
    in(s: Array)
    local()
    out(e: E)
    { **pre**: $len(s) \neq 0$ }
    { **post**: last_element(s) $\stackrel{\text{def}}{=} s(len(s))$
    }

Figure: Example of predicate logic specification of component[23]

Generating Lower Level Hierarchy I

- The lower level hierarchy will generate fine-grained components on which logical reasoning can be done on the basis of specifications.
- The subsumption relationship between 2 components will be used to classify and group together components which can be reused.
- A component is said to subsume another component when the first component is more general than the second.
- This abstract data type can be implemented on classes as class can also be termed as an abstract data type.

Generating Lower Level Hierarchy II

Algorithm 1: Pair Wise Algorithm

Input: The set of components from subsumption test algorithm

$S = \{Cmp_1, Cmp_2, ..., Cmp_n\}$

Output: Hierarchy of components by applying generality principle

while $S \neq \{\}$ **do**

- choose some $Cmp_i \mathcal{E} S$;
- $S \leftarrow S \div Cmp_i$;
- $set \leftarrow S$;
- **while** $set \neq \{\}$ **do**
  - choose some $Cmp_j \mathcal{E} S$;
  - $S \leftarrow S \div Cmp_j$;
  - **if** $C_i \sqsupseteq C_j$ **then**
    - C_i is set as parent of C_j
  - **else if** $C_j \sqsupseteq C_i$ **then**
    - C_j is set as parent of C_i

Measuring Similarity

- Similarity of 2 components is represented as $S(x, y)$.
- Example $X = (C_1 \wedge C_2) \vee (C_2 \wedge C_3)$ *and* $Y = (C_3) \vee (C_2 \wedge C_3)$
- In the solution a component is represented in disjunctive normal form consisting of conjuncts. Each conjunct is to be associated with an equivalence class.
- For every component a matrix will be constructed whose dimensions are $m \times (T + 1)$ where m is the number of conjunction and T is the number of equivalence classes.
- $X(i, 0) = u_i$ $0 \leqq i \leqq m \leqq 1$ *and*$X(i, j) = l$ x_i *has* l *terms in eq class* j
- *for all* i, j *if* $X(i, 0) = Y(j, 0))$
  then $s^{'}(i, j) = \frac{\sum_{t=1}^{T} 2 * min(X(i,t), Y(j,t))}{(X(i,t) + Y(j,t)}$
  else $s^{'}(i, j) = 0$
- $s(X, Y) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} s^{'}(i,j)}{N}$

Higher Level Hierarchy Using Clustering

- In divisive algorithms the set X is further divided into a set of clusters using dissimilarity measure to form a finer partition.
- In agglomerative algorithms a group of finer partitions are merged together using similarity measures to form a courser partition.
- Each intermediate steps helps in generating a hierarchy of clusters.
- It provides an architectural understanding of the system.
- The agglomerative hierarchical clustering algorithms (AHCA) begin from a set of individual entities that are first allocated into small clusters which are then in turn allocated into larger clusters until reaching a final all inclusive clustering.

Agglomerative Clustering

Algorithm 2: Agglomerative Clustering

Input: A Set of disjoint Lattices

Output: Set of Clusters Unified into one

1. each top element of the lattice is set a cluster with only single element
2. search for a pair of clusters which have maximum similarity
3. merge the the top 2 most similar clusters into a new cluster
4. calculate new similarity scores for the clusters left
5. proceed until only one cluster is left

Final Result

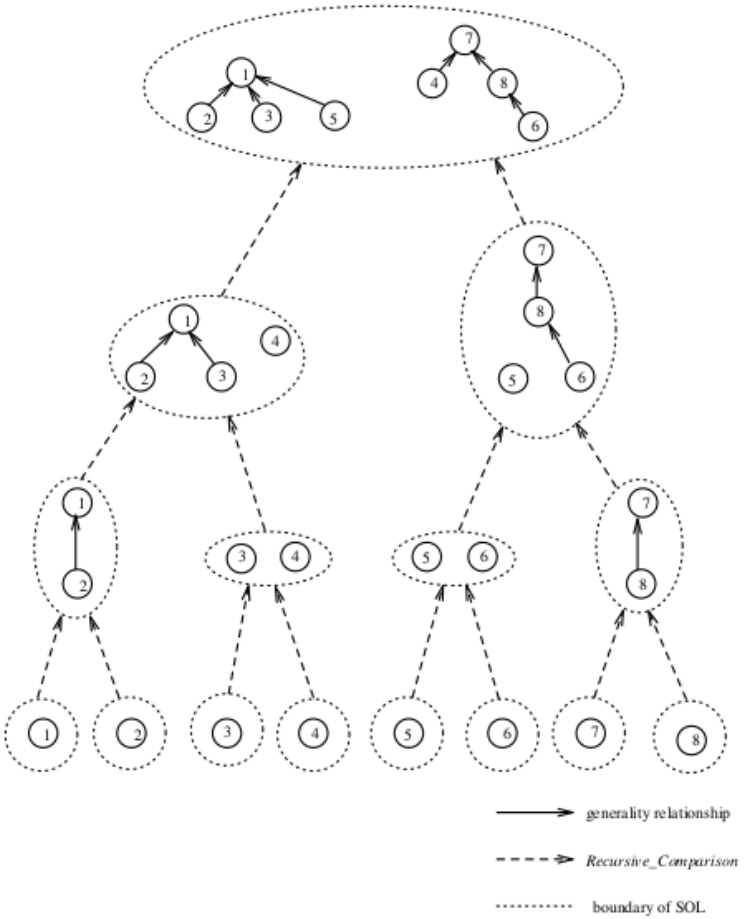

Figure: Lower Level Hierarchy Building using recursive comparison[23]

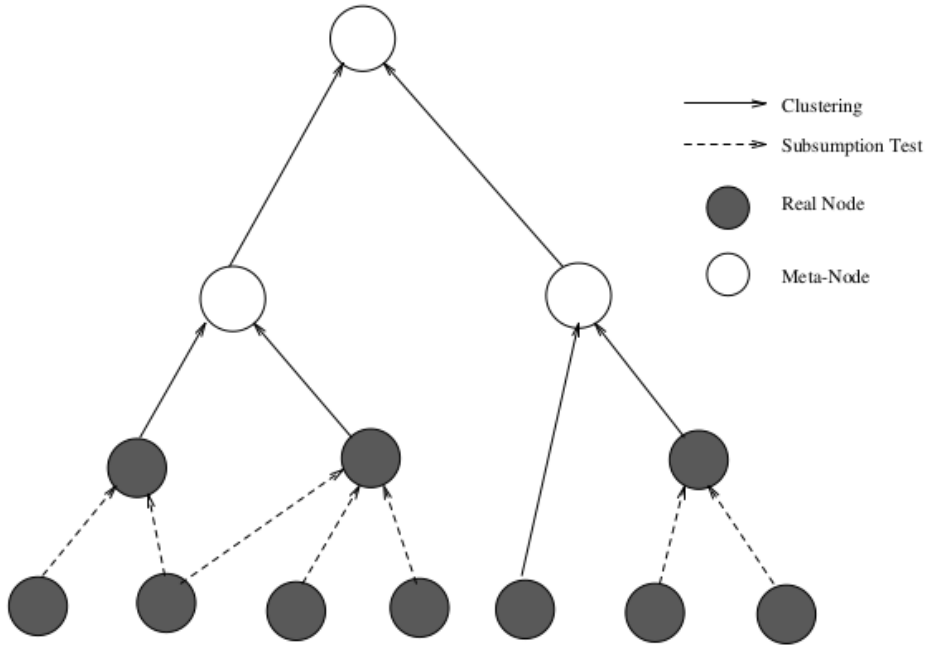

Figure: Final resultant hierarchy of software components[23]

Conclusion and Future Research

- Most of the Approaches used Clustering for component identification but focussed mainly on top down approach
- Evolutionary techniques like particle swarm optimization and genetic algorithm gives better result but are limited to object oriented programming.
- Using formal methods to describe the source code and then using it to identify software components is the approach we will use for software component identification.
- Verification of software components using the idea from [24]
- Next problem in CBSE is composing software using components where a providing a proper semantic definition will help in cleanly integrating all requirements of the software system into one system.

References I

[1] Fatihah Mohd, Suryani Ismail, Masita Masila Abdul Jalil, Fatin Izatul Nadia Mohd Zulkarnain, and Norshuhani Zamin.
A guidelines for controlled experimental design to evaluate the metrics of software component reusability.
In *2021 Fifth International Conference on Information Retrieval and Knowledge Management (CAMP)*, pages 85–90, 2021.

[2] Shabnam Gholamshahi and Seyed Mohammad Hossein Hasheminejad.
Software component identification and selection: A research review.
Software: Practice and Experience, 49, 10 2018.

[3] Greg Baster, Prabhudev Konana, and Judy E. Scott.
Business components: A case study of bankers trust australia limited.
Commun. ACM, 44(5):92–98, May 2001.

References II

[4] Dominik Birkmeier and Sven Overhage.
On component identification approaches - classification, state of the art, and comparison.
In *Component-Based Software Engineering, 12th International Symposium, CBSE 2009, East Stroudsburg, PA, USA, June 24-26, 2009, Proceedings*, volume 5582 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[5] Jiafu Tang, Li feng Mu, C.K. Kwong, and X.G. Luo.
An optimization model for software component selection under multiple applications development.
European Journal of Operational Research, 212:301–311, 07 2011.

References III

[6] Iria Estevez-Ayres, Luis Almeida, Marisol Garcia-Valls, and Pablo Basanta-Val.
An architecture to support dynamic service composition in distributed real-time systems.
In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 249–256, 2007.

[7] Antonia Albani, Sven Overhage, and Dominik Birkmeier.
Towards a systematic method for identifying business components.
In *Component-Based Software Engineering*, pages 262–277, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

References IV

[8] Mohammad Ahmadzadeh, Gholamreza Shahmohammadi, and Mohammad Shayesteh.
Identification of software systems components using a self-organizing map competitive artificial neural network based on cohesion and coupling.
ANDRIAS Journal, 40:721–6513, 02 2016.

[9] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili.
Sci-ga: Software component identification using genetic algorithm.
The Journal of Object Technology, 12:3:1, 01 2013.

[10] R. Ganesan and S. Sengupta.
O2bc: a technique for the design of component-based applications.
In *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39*, pages 46–55, 2001.

References V

[11] Haitham S. Hamza.
A framework for identifying reusable software components using formal concept analysis.
In *2009 Sixth International Conference on Information Technology: New Generations*, pages 813–818, 2009.

[12] Maen Hammad and Rua'a Hasan Banat.
Automatic class decomposition using clustering.
2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C), pages 78–81, 2021.

[13] Jong Kook Lee, Seung Jae Jung, Soo Dong Kim, Woo Hyun Jang, and Dong Han Ham.
Component identification method with coupling and cohesion.
In *Proceedings Eighth Asia-Pacific Software Engineering Conference*, pages 79–86, 2001.

References VI

[14] Soo Dong Kim and Soo Ho Chang.
A systematic method to identify software components.
In *11th Asia-Pacific Software Engineering Conference*, pages 538–545, 2004.

[15] Zhengong Cai, Xiaohu Yang, Xinyu Wang, and Aleksander Kavs.
Afuzzy formal concept analysis based approach for business component identification.
Journal of Zhejiang University - Science C, 12:707–720, 09 2011.

[16] Seyed Mohammad Hossein Hasheminejad and Shabnam Gholamshahi.
Pci-pso: Preference-based component identification using particle swarm optimization.
Journal of Intelligent Systems, 28(5):733–748, 2019.

References VII

[17] Hamid Masoud, Saeed Jalili, and Seyed Mohammad Hasheminejad.
Dynamic clustering using combinatorial particle swarm optimization.
Applied Intelligence, 38(3):289–314, April 2013.

[18] Sunil Kumar, Rajesh Bhatia, and Rajesh Kumar.
K-means clustering of use-cases using mdl.
Communications in Computer and Information Science, 270:57–67, 01 2012.

[19] Jian Cui and Heung Chae.
Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems.
Information & Software Technology, 53:601–614, 06 2011.

[20] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili.
Ccic: Clustering analysis classes to identify software components.
Information and Software Technology, 57, 06 2014.

References VIII

[21] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili.
An evolutionary approach to identify logical components.
Journal of Systems and Software, 96, 10 2014.

[22] Gabriela Arévalo, Nicolas Desnos, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier.
Formal concept analysis-based service classification to dynamically build efficient software component directories.
International Journal of General Systems - INT J GEN SYSTEM, 38:427–453, 05 2009.

[23] Jun Jang Jeng and Betty H. C. Cheng.
Using formal methods to construct a software component library.
In Ian Sommerville and Manfred Paul, editors, *Software Engineering — ESEC '93*, pages 397–417, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

References IX

[24] Mehdi Hariati.
Formal verification issues for component-based development.
Informatica, 44, 12 2020.

Thank You!
Any Questions?