Applying Social Network Analysis Centrality to Detect Software Design Patterns

García-Hernández, Alejandra
Velasco-Elizondo, Perla
Zamarrón, Juan M.

Autonomous University of Zacatecas (UAZ)
Centre for Mathematical Research (CIMAT)

CIMPS, 2014.

Content

Introduction

Software systems can be seen as a set of interdependent software components forming a network

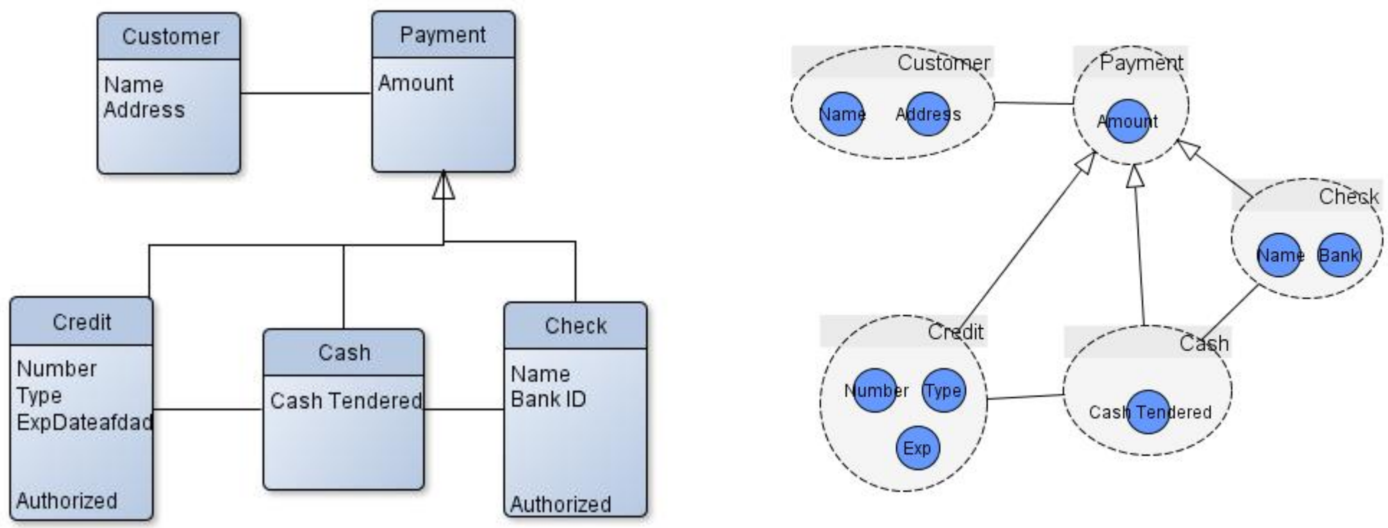


Figure 1. Software Network

Pattern: Is a general structure of software componentents useful for solving a recurring software design problem (Gamma, Helm, Johnson & Vlissides 2005)

Introduction

- Problem: Systems do not have documentation
- Much time is spent on studying the software to understand it

- Patterns could help to comprehend the behaviour of the systems.

We consider that a design pattern detection approach can be defined based on SNA techniques.

Research Objectives

- Describe a first effort in defining a design pattern detection approach.
- Apply SNA centrality metrics to automatically indentify design patterns in object oriented systems' source code.
- Describe and evaluate the results of applying SNA centrality metrics.

Background

Social Network Analysis has been applied to:

- Explain organizational performance .
- Study behaviours in online social networks.
- Understand organizational aspects of developers teams.

And Recently:

- To understand the structure and behaviour of software systems.

Background

Social Network Analysis metrics used in this research:

- Degree Centrality: is the number of relationships that are incident with the node.

    InDegree Centrality: denotes the number of ingoing nodes related to a node.

    OutDegree Centrality: denotes the number of outgoing nodes related to a node.

Background

Object-Oriented Software Systems (OOSS)

OOSS: are designed and implemented as a group of interacting objects.

Object: represents a real world entity that is meaningful to the system being constructed.

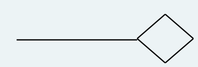
Class: is a template for defining objects' instances.

Every object is built from a class.

Background

Classes and objects interact via relationships that denote specific forms of interactions:

Table 1. Common relationships in context of object-oriented systems

Name	Semantics	UML Syntax
Association	It denotes that one object can cause another to perform an operation on its behalf.	
Aggregation	It denotes a whole-part relationship between the an aggregated object (the whole) and their constituents (the parts).	
Generalization	It denotes that one of two related classes (the subclass) is considered to be a specialized form of the other (the superclass).	
Implementation	it denotes that one of the related classes realizes the operations specified by the other.	

Background

Design Patterns.

Patterns come as structures of software elements and specific relationships among them.

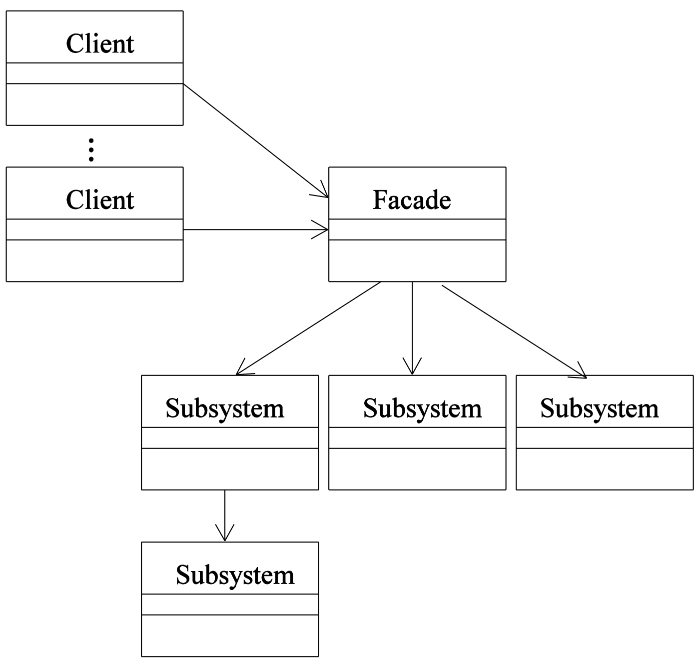


Figure 2. Façade Pattern

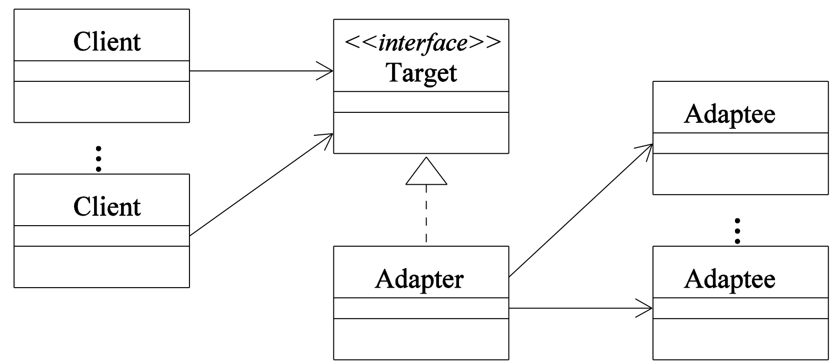


Figure 3. Adapter Pattern

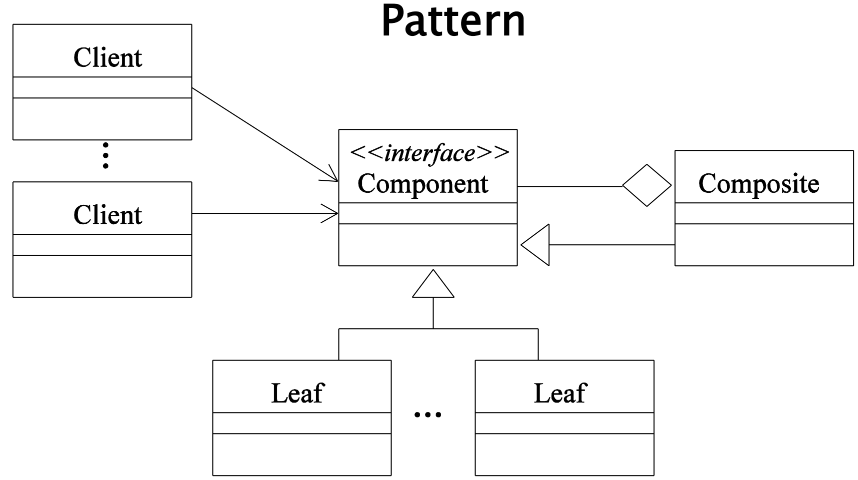


Figure 4. Composite Pattern

Related Work

Approaches to design pattern detection from a systems' source code:

- **Mathematical formalisms**: each pattern is represented by class diagrams. The resulting diagrams are translate into a set of facts and rules, that are introduced in a software for detect the design patterns.
- **Graph theory**: detect design patterns by analysing similarity between the classes in different systems or two entire systems (Tsantalis et al. 2006; Dong et al. 2009; Akshara et al. 2010; Balanyi & Ferenc 2003).
- **Object-Oriented Software metrics**: are used to determine pattern constituent's candidate sets (Antoniol et al. 1998).
- **Ontology-based approach**: Use an analyser to construct the input code as ontology individuals and asks the analyser to clasify them.

Approach

Generating Software Networks

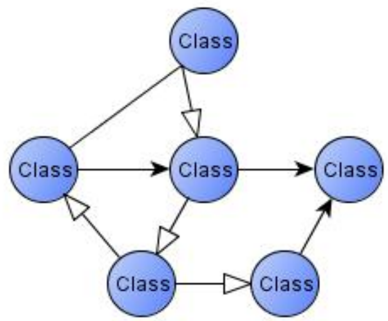


Table 2. Relationships

	Representation
Association	
Aggregation	
Generalization	
Implementation	

1. We automatically detected nodes and their relationship from source code by using the Gate tool (Gate, 2013).
2. The data is stored in a database.

Table 3. Database

ID Relationship	Relationship	Source Class	Class Type	Target Class	Class Type	Location
1	Generalization	Advertiser	Abstract	Account	Abstract	File:/G:/relationships
2	Association	Car	Public	Engine	Public	File:/G:/relationships
3	Composition	Apple	Abstract	Fruit	Public	File:/G:/relationships

Approach

3. The data is imported into the Jung Software to generate a software network.

4. With Jung was possible to identify nodes, links among nodes and their direction, and type of relation.

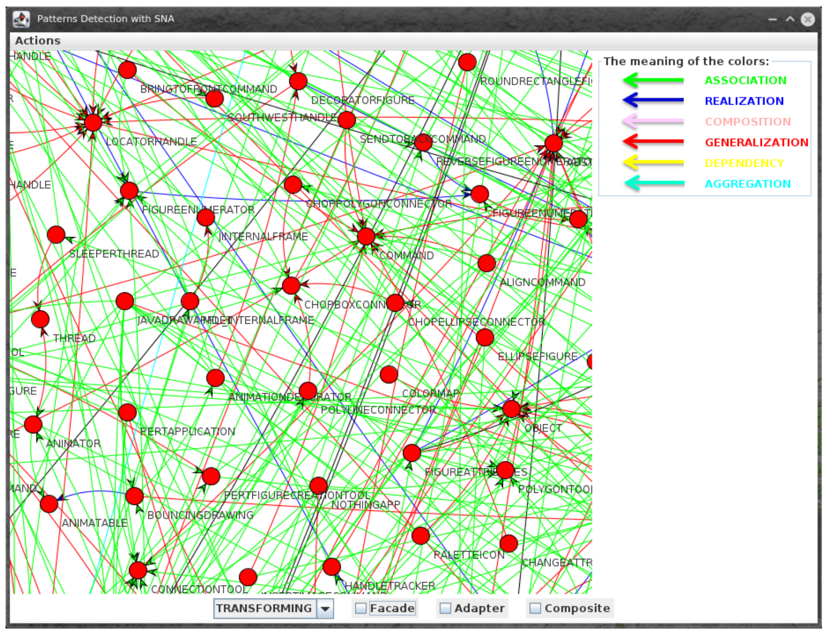


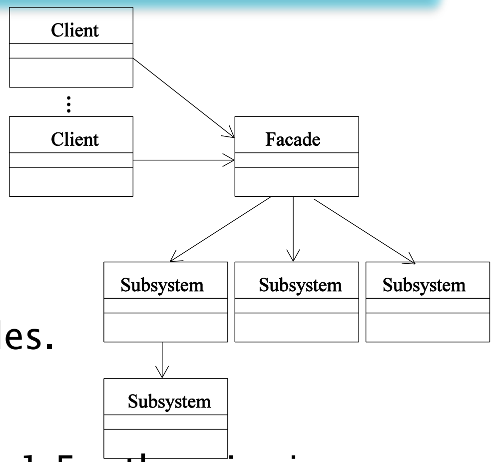
Figure 5. An excerpt of system network visualization

Approach

Analysing Software Networks

Metrics: degree centrality, in-degree, out-degree.

Approach



a) Analysis for Detecting the Façade Pattern:

1. Detect the Façade node:
   1.1 Get the nodes that have association relationships with other nodes.
   1.2 Compute the **in-degree** centrality of each node.
   1.3 Compute the **out-degree** centrality of each node.
   1.4 If the node has in-degree>0 and out-degree>0 continue to step 1.5, otherwise ignore it and analyse the next node.
   1.5 If a node meets the above rules, extract the Façade node from the network. Otherwise there is not a Façade pattern in the network.
2. Detect the Client nodes:
   2.1 Get the ingoing nodes of each Façade node and extract.
3. Detect the Subsystem nodes:
   3.1 Get the outgoing nodes of each Façade nodes.
   3.2 Analyse each outgoing node and the nodes related to them.
   3.3 If a successor node of the Façade, is a client node of another Façade, that node is not considered as a subsystem node, otherwise all the nodes are part of the subsystem nodes.
   3.4 Extract the subsystem nodes.

Approach

b) Analysis for Detecting the Adapter Pattern.

1. Detect the Target node:
   1.1 Get the nodes that have association and implementation in-going relationships.
   1.2 Compute the in-degree centrality of each node.
   1.3 Compute the out-degree centrality of each node.
   1.4 If in-degree centrality >= 2 and out-degree centrality == 0 continue to step 1.5, otherwise ignore it.
   1.5 If a node meets the above rules, extract the candidate node target from the network.
   Otherwise, there is not an Adapter pattern in the network.
2. Get the Clients nodes:
   2.1 Get the target nodes identified in step 1.
   2.2 Get the in-going nodes of each target node.
   2.3 If in-going nodes have an association relationship with the target node, extract it (all the extracted nodes are client nodes).
   2.4 Otherwise, there is not an Adapter pattern in the network.
3. Get the Adapter nodes:
   2.1 Get the target nodes identified in step 1.
   2.2 Get the in-going nodes of each target node.
   2.3 If ingoing nodes have an implementation relationship with the target node, extract it (all the extracted nodes are Adapter nodes).
   2.4 Otherwise, there is not an Adapter pattern in the network.
4. Get the Adaptee nodes:
   4.1 Get the nodes Adapter identified in step 3.
   4.2 Get all the out-going nodes of the Adapter node and extract them.
   4.3 If outgoing nodes have an association relationship with the Adapter node, extract them.
   4.4 Otherwise, there is not an Adapter pattern in the network.

Approach

c) Analysis for Detecting the Composite Pattern.

1. Get the Component node:
   1.1 Get the nodes with aggregation, association and generalization edges.
   1.2 Compute the in-degree centrality of each node.
   1.3 Compute the out-degree centrality of each node.
   1.4 If in-degree centrality >= 3 and out-degree centrality == 0 continue to step 1.5 otherwise ignore it.
   1.5 If the node type is abstract, continue to step 1.6 otherwise ignore the node.
   1.6 If a node meets the above rules, extract the candidate node component from the network. Otherwise, there is not a Composite pattern in the network.
2. Get the Composite nodes:
   2.1 Get the Component nodes from step 1.
   2.2 Get the neighbours of the Component nodes that have aggregation and generalization edges.
   2.3 If a neighbour has an aggregation out-going edge and a generalization out-going edge with the Component node extract.
   2.4 Otherwise, eliminate the Composite node and there is not a Composite pattern in the network.
3. Get the Client nodes:
   3.1 Get the Component nodes from step 1.
   3.2 Get the predecessors of each Component node that have an association relationship with the Component node.
   3.3 Extract it (all extracted nodes are client nodes).
   3.4 Otherwise, eliminate the Client node and there is not a Composite pattern in the network.
4. Get the Leaf nodes:
   4.1 Get the Component nodes from step 1.
   4.2 Get the predecessor's nodes of each Component node that have a generalization relationship with the Component node.

Research Results

- We apply our approach to design pattern detection on two existing software systems that have been used by other authors (Tsantalis et al. 2006):

  1. JHotDraw 5.2
  2. Jrefactory 2.6.24

- We compare our data with Tsantalis et al., data:

Table 4. Number of detected patterns by Tsantalis et al., and our approach

	Façade		Adapter		Composite	
	Tsantalis et al.	Our approach	Tsantalis et al.	Our approach	Tsantalis et al.	Our approach
JHotDraw	No support provided	42	18	21	1	2
JRefactory	No support provided	117	11	11	0	3

Research Results

- We also assessed the performance of our approach in terms of the time required for analysing the information of the system:

Table 5. Time required for analysing pattern information by Tsantalis et al., and our approach

	Façade		Adapter		Composite	
	Tsantalis et al.	Our approach	Tsantalis et al.	Our approach	Tsantalis et al.	Our approach
JHotDraw	No support provided	176 ms	209 ms	4 ms	4 ms	12 ms
JRefactory	No support provided	240 ms	2066 ms	13 ms	55ms	68 ms

Research Results

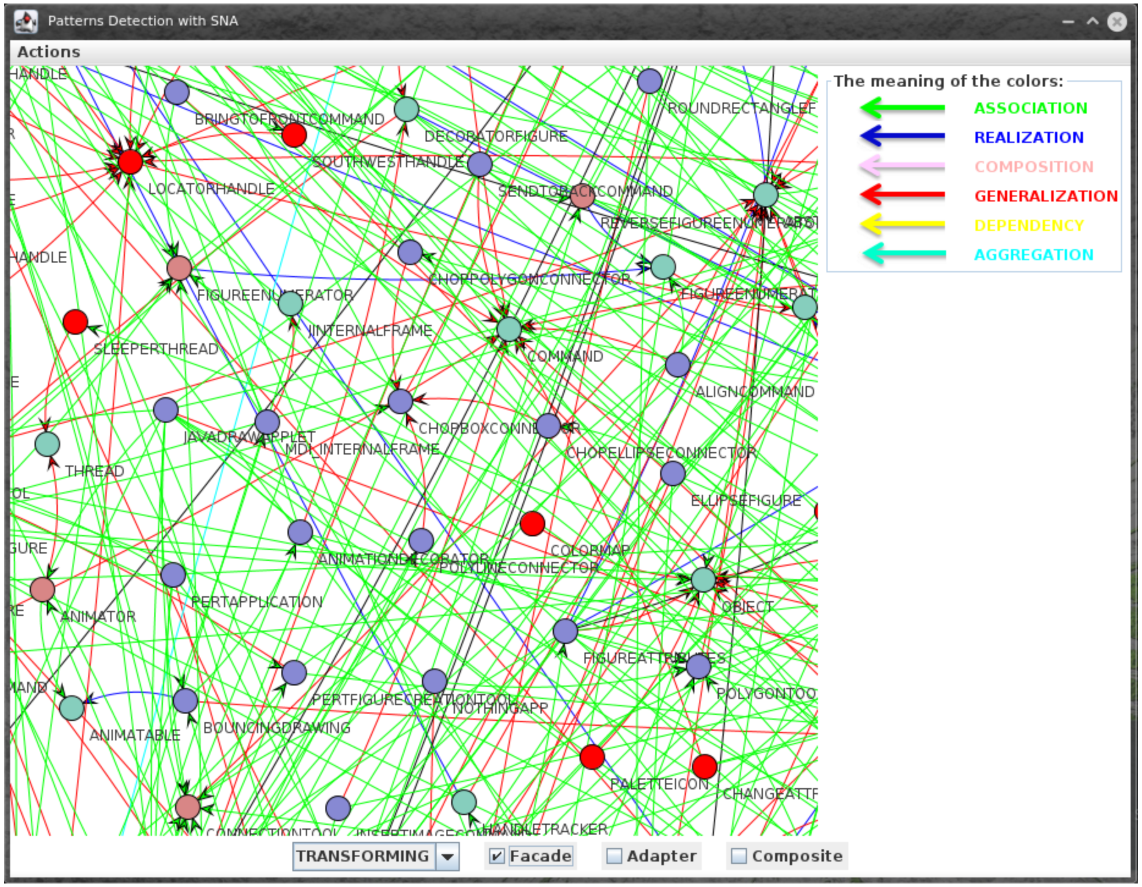


Figure 6. An excerpt of pattern detection visualization

Conclusions

- Our results show that SNA metrics work well in detecting software design patterns.

Future Work:

- Develop the means to detect more patterns in systems codified in other programming languages.
- Achieve our ultimate goal of defining an approach to reconstruct the architecture of a software system.

Thank you for your attention

CONTACT

Dra. Alejandra García Hernández

UNIVERSIDAD AUTÓNOMA DE ZACATECAS

alegarcia@uaz.edu.mx