Recovering Runtime Architecture Models of Large Object-oriented Software Systems

Séminaire LATECE-UQAM – 24 mars 2021

Chouki Tibermacine

Outline

Outline

Migrating Legacy OO SW Systems

1. Recovering an architecture model from SW artifacts[1] (source code, config. files, execution traces, ...): identifying components and their inter-dependencies (clustering classes, extracting their interfaces, ...)

2. Refactoring the source code to match the architecture

[1] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. IEEE Tran. on SW Eng., 2009

A Larger Context for Arch. Recovery: SW Maintenance

- Maintenance of existing object oriented software systems represent significant investments (65%[2] to 90%[3] of the total project cost, depending on the projects/companies)

IEEE Std 1219: "Maintenance = Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

[2]Schach, R. (1999), Software Engineering, Fourth Edition, McGraw-Hill, Boston, MA, pp. 11.

[3]Erlikh, L. (2000), Leveraging legacy system dollars for E-business, IEEE IT Pro, May/June 2000, pp. 17-23.

SW Maintenance

It requires high level views of SW structure and behavior (architecture models)

- to assist developers in understanding the system
- because approximately 50% of maintainers time is spent in code understanding[4]

[4] Standish, T. (1984), An essay on software reuse. IEEE Trans. on SW Engineering SE-10 (5), pp. 494-497.

Architecture Recovery

- Architecture Recovery is a challenging and time-consuming task
  - Medium-sized systems of 70 to 280 KLOC require on average 100 person-hours[5]
- Automatic or semi-automatic architecture recovery methods are needed
- Several methods have been proposed for architecture recovery (Ducasse et al. TSE'09 and[5])
  - Most of them recover the static (class-based) architecture (source code/file clustering)
- Our challenge: investigate the use of runtime architectures

[5] J. Garcia et al. Obtaining ground-truth software architectures. In ACM/IEEE ICSE, 2013.

Runtime architecture

What?

- Runtime architecture = system's objects (not classes) and dependencies between them

Why?

1. Reflects system's running and concrete interacting entities
2. Theoretically, we may have "less complex" models than class-based ones (dead-code is eliminated)
3. The importance of having runtime architecture models was stressed in several experimental studies[6]

---

[6] S. Lee et al. How can diagramming tools help support programming activities? In VL/HCC, 2008

N. Ammar et al. Empirical evaluation of diagrams of the run-time structure for coding tasks. In WCRE, 2012.

Problem and Goal

Problem

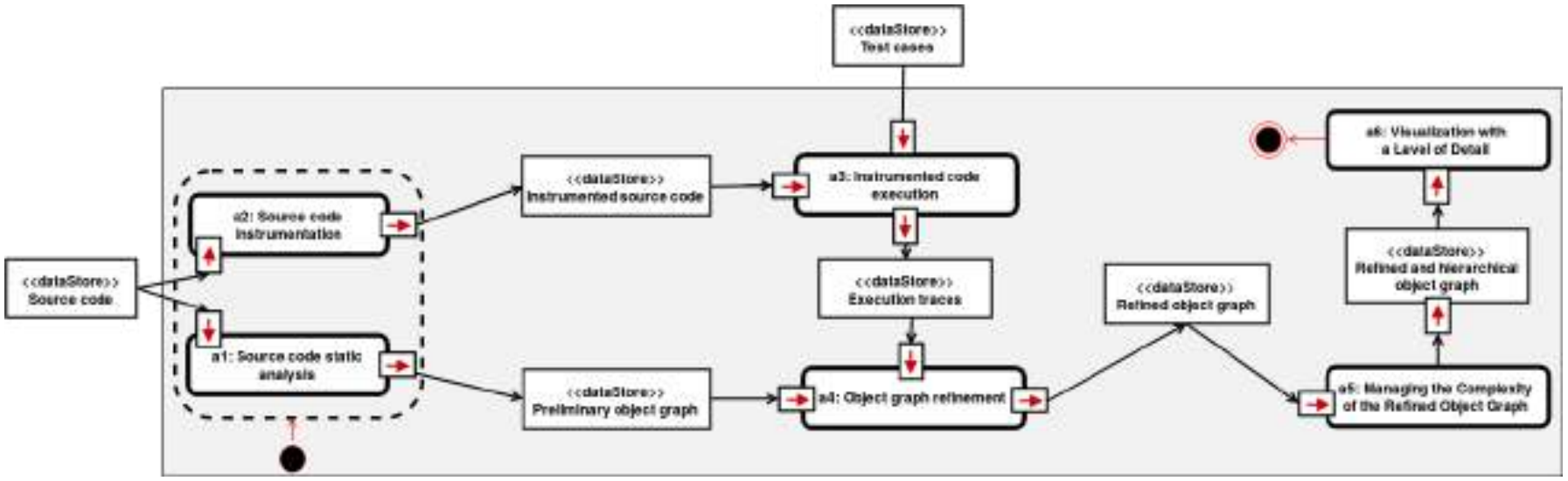
- Few methods target the recovery of runtime architectures
- They fail most of the time to provide models of a reasonable size that effectively help in understanding the structure of the system during its execution

Goal

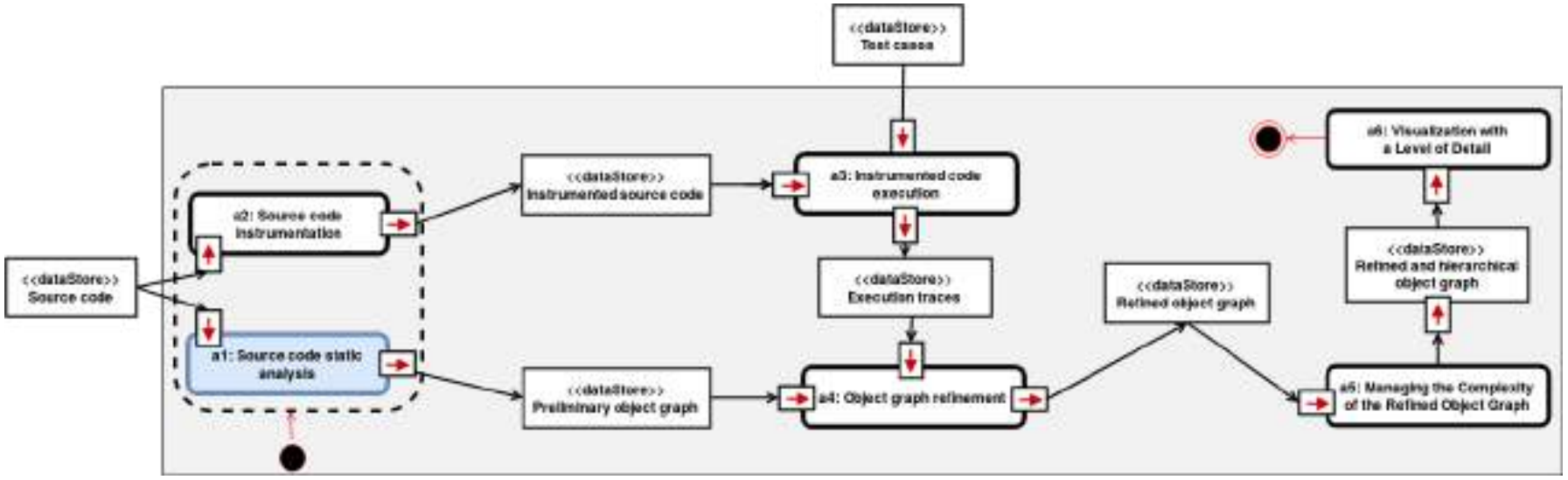
- We propose a method that recovers the runtime architecture of large OO systems and helps managing the complexity of the recovered architecture

Outline

Process in a nutshell



Process in a nutshell



a1: Source code static analysis

- To build a preliminary Object Graph (OG)
- An OG is a Directed Labeled Graph (DLG) where:
  - Nodes represent objects and
  - An edge between two nodes n1 and n2 indicates that the object modeled by n1 references the object modeled by n2 via a field
  - Edges are labeled with field names
- The flow of objects must be tracked
  - This is achieved by using an Object Flow Graph (OFG)[7]
- An OFG is a DLG where:
  - Nodes denote objects and program variables
  - Edges represent assignments between these variables

[7] Paolo Tonella. Reverse engineering of object oriented code. In ICSE, 2005.

a1: Source code static analysis

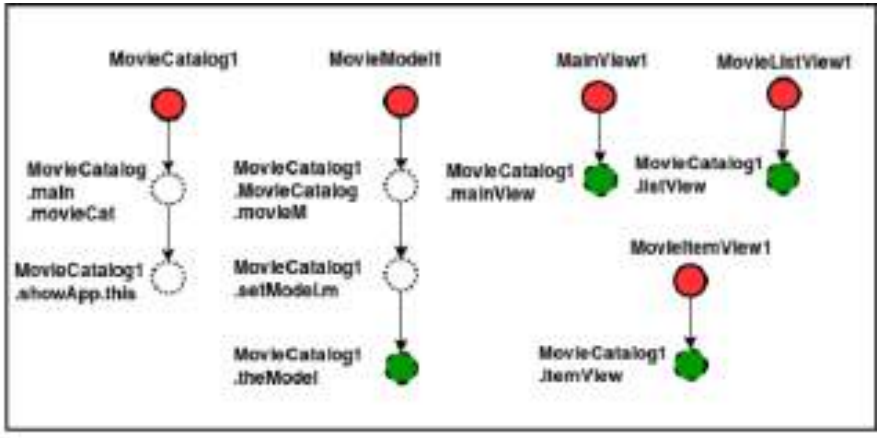
- Statements of interest for OFG recovery

```
<sta> ::= x = new constr([a 1 , a 2 , ..., a n ]); % Allocation site
        | x = y;                                    % Assignment site
        | [x =] y.meth([a 1 , a 2 , ..., a n ] );   % Invocation site
```

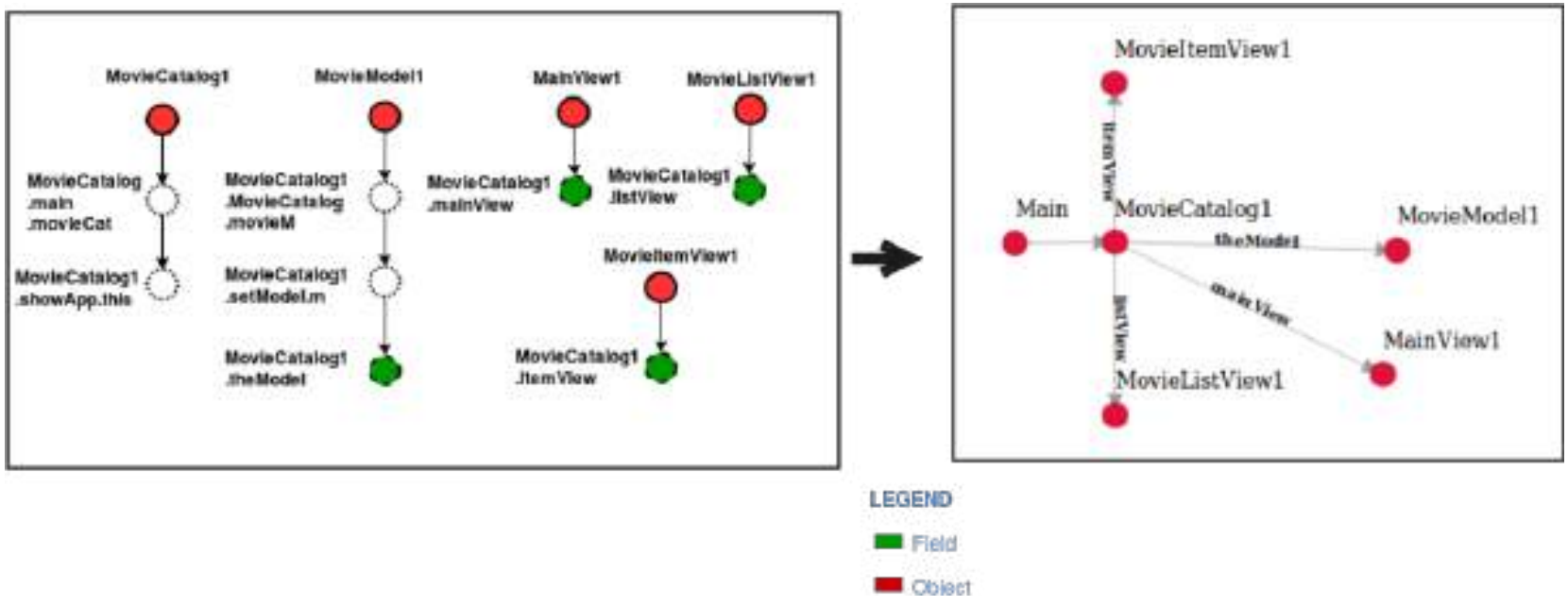
- Variables x, y, a_1 , a_2 and a_n are typed by a user-defined classes/interfaces
- We ignore types from libraries
- The OG is recovered by analyzing the output sets of the OFG nodes that correspond to fields

Movie Catalog example

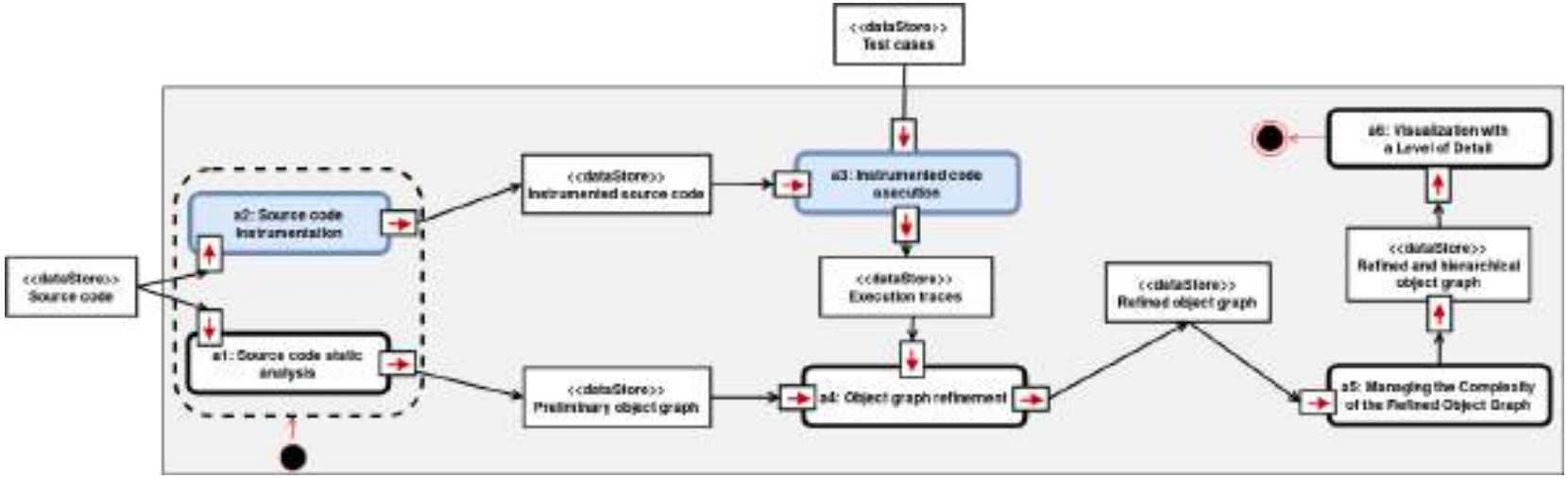
```
public class MovieCatalog extends WmvcApp {

private  MainView mainView ;
private  MovieListView listView ;
private MovieItemView itemView ;

public MovieCatalog ( String name ) {
MovieModel movieM = new MovieModel ( ) ;
setModel ( movieM ) ;
mainView = new MainView ( ) ;
listView = new MovieListView ( ) ;
itemView = new MovieItemView ( ) ;
}
public static void main ( String[] args) {
MovieCatalog movieCat =new MovieCatalog( " " )
movieCat. showApp ( ) ;
}
}
```



Movie Catalog example

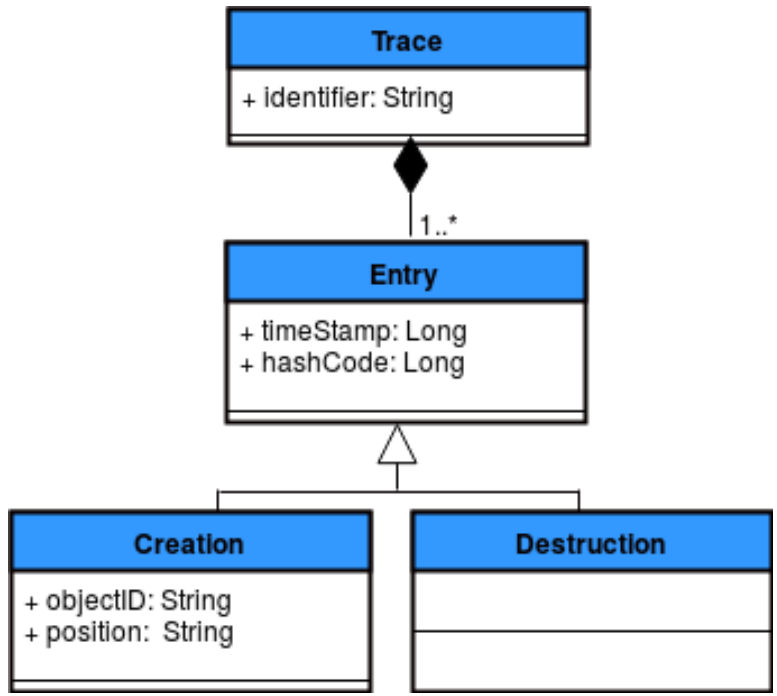


Process in a nutshell



a2 + a3: Source code instrumentation and instrumented code execution

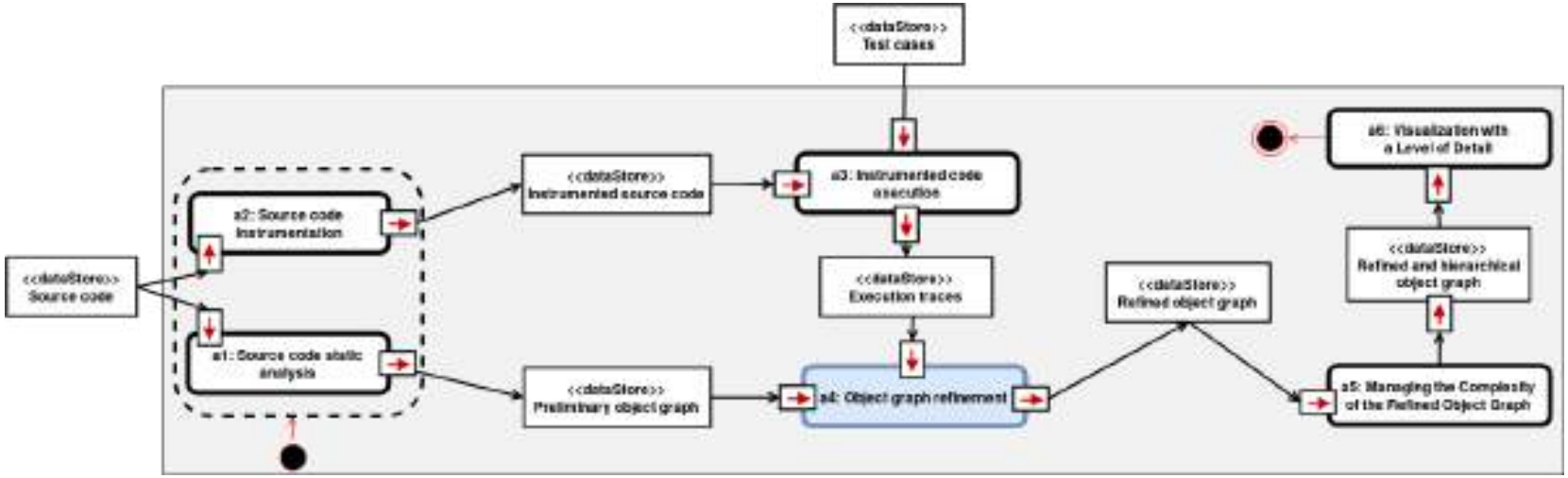
- Instrumentation consists of adding automatically statements that produce execution traces



Movie Catalog example

```
TimeStamp= 1493642333681, new:ObjectID = MovieModel1     Position = MovieCat: 15, hashCode= 186370029
TimeStamp= 1493642333797, new:ObjectID = MovieEditor1    Position = MovieEditor: 52, hashCode= 1768305536
TimeStamp= 1493642333861, new:ObjectID = MainView1       Position = MovieCat: 17, hashCode= 1915503092
TimeStamp= 1493642333862, new:ObjectID = MovieListView1  Position = MovieCat: 18, hashCode= 1567581361
TimeStamp= 1493642333866, new:ObjectID = MovieItemView1  Position = MovieCat: 19, hashCode= 1688376486
TimeStamp= 1493642333889, new:ObjectID = MovieCatalog1   Position = MovieCat: 38, hashCode= 1793329556
TimeStamp= 1493642368150, new:ObjectID = Movie1          Position = MainView: 137, hashCode= 20541719
TimeStamp= 1493642507878, finalize: 20541719
```

Process in a nutshell



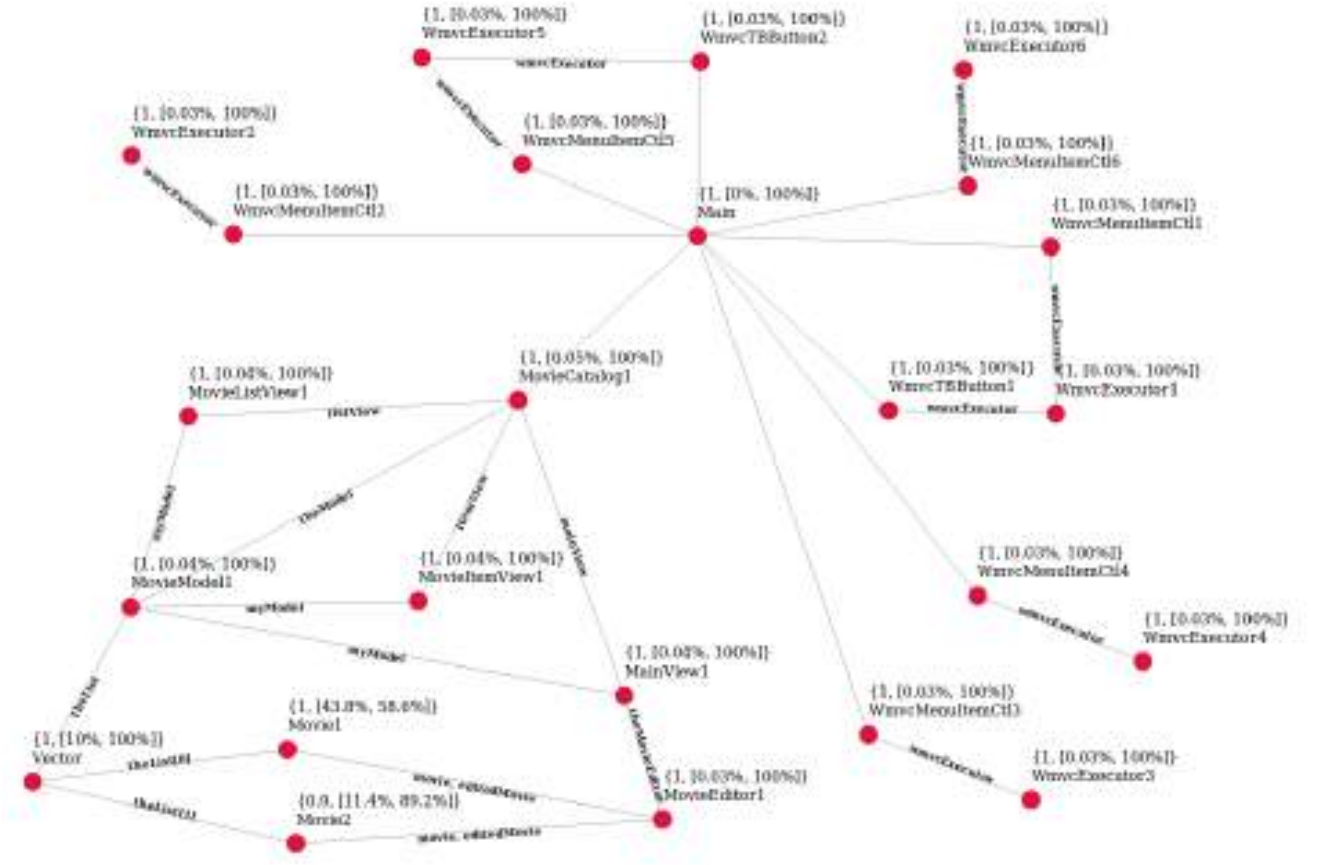
a4: Object Graph Refinement

- Add two kinds of labels on nodes of the OG: empirical probabilities and lifespans
- Probability is the ratio of the number of occurrences of this object in execution traces to the total number of execution traces
- Lifespans are measured using the creation and destruction timestamps

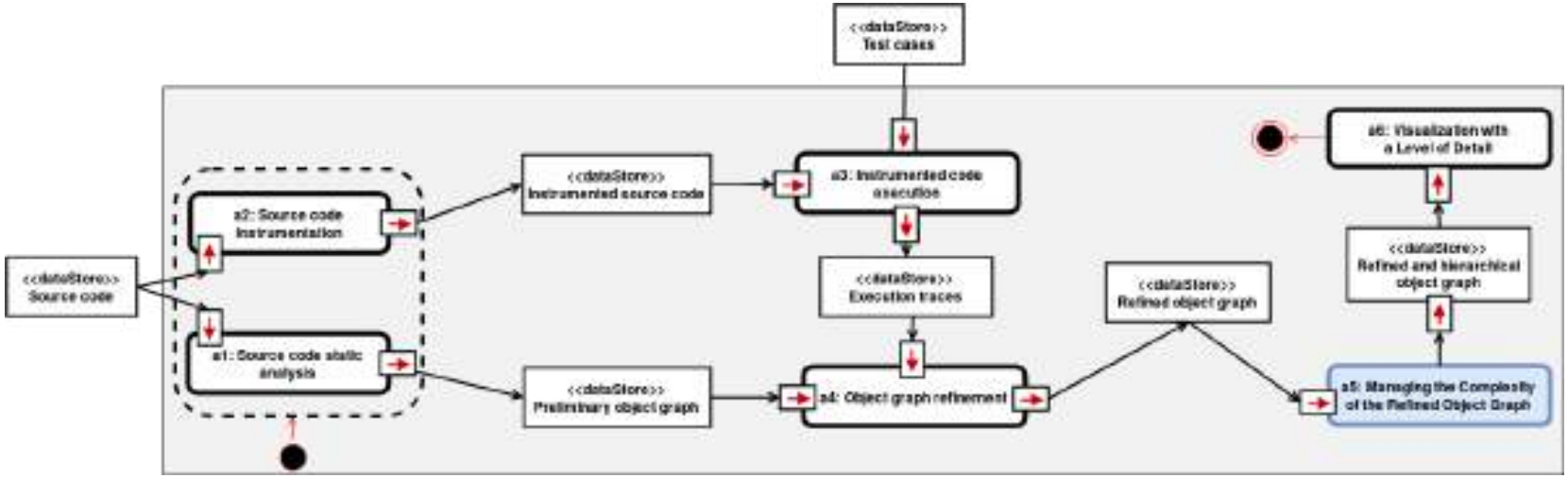
$$sts_c(n) = \frac{1}{m} * \sum_{i=0}^{m} \frac{(cts_i(n) - st_{sys_i})}{length_{sys_i}} * 100 \qquad (1)$$

$$sts_d(n) = \frac{1}{m} * \sum_{i=0}^{m} \frac{(dts_i(n) - st_{sys_i})}{length_{sys_i}} * 100 \qquad (2)$$

Movie Catalog example



Process in a nutshell



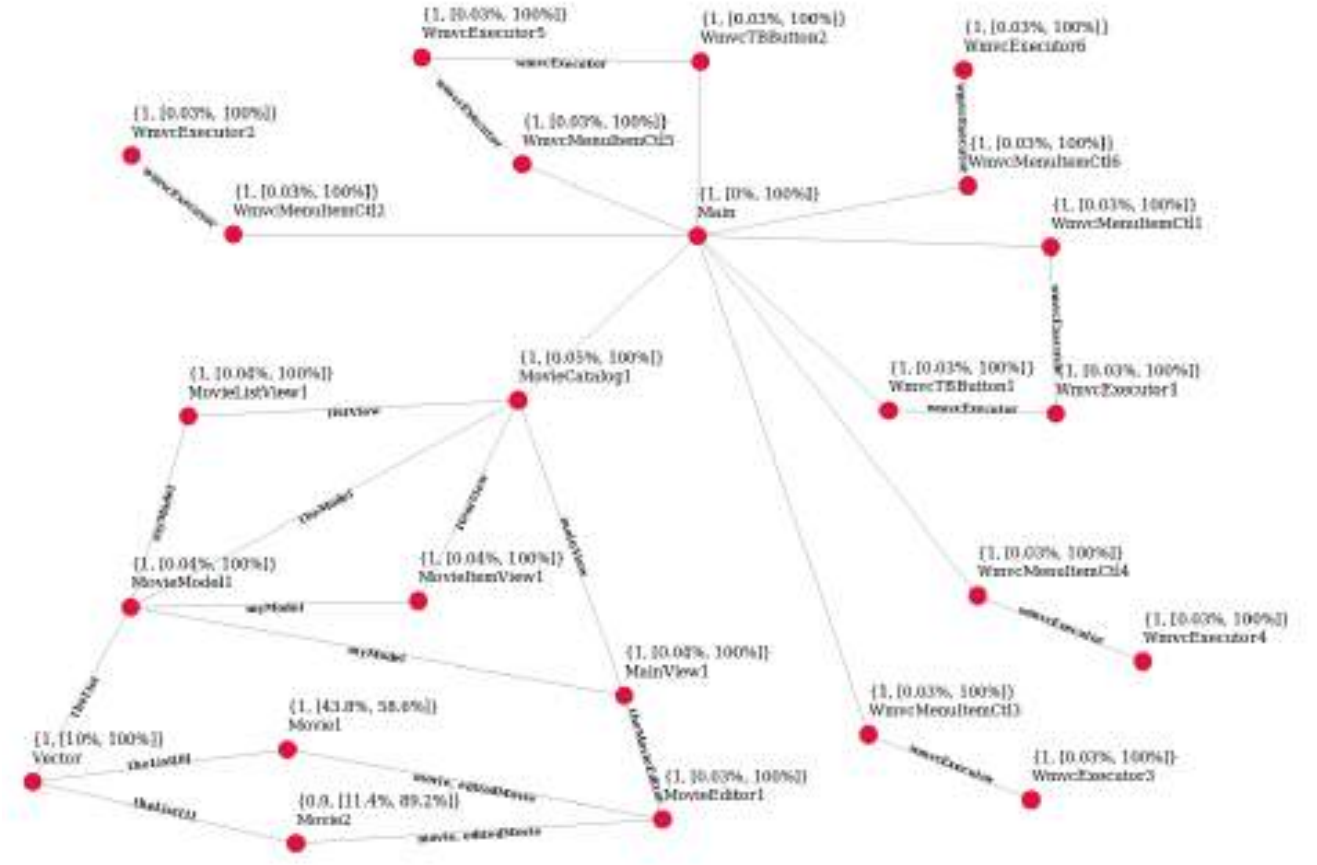
a5: Managing the Complexity of the Refined Object Graph

- Two techniques are used
  - Exploiting object lifespan and empirical probability for a visualization with a LOD
  - Identifying composite structures in the refined OG based on the owners-as-dominators ownership model

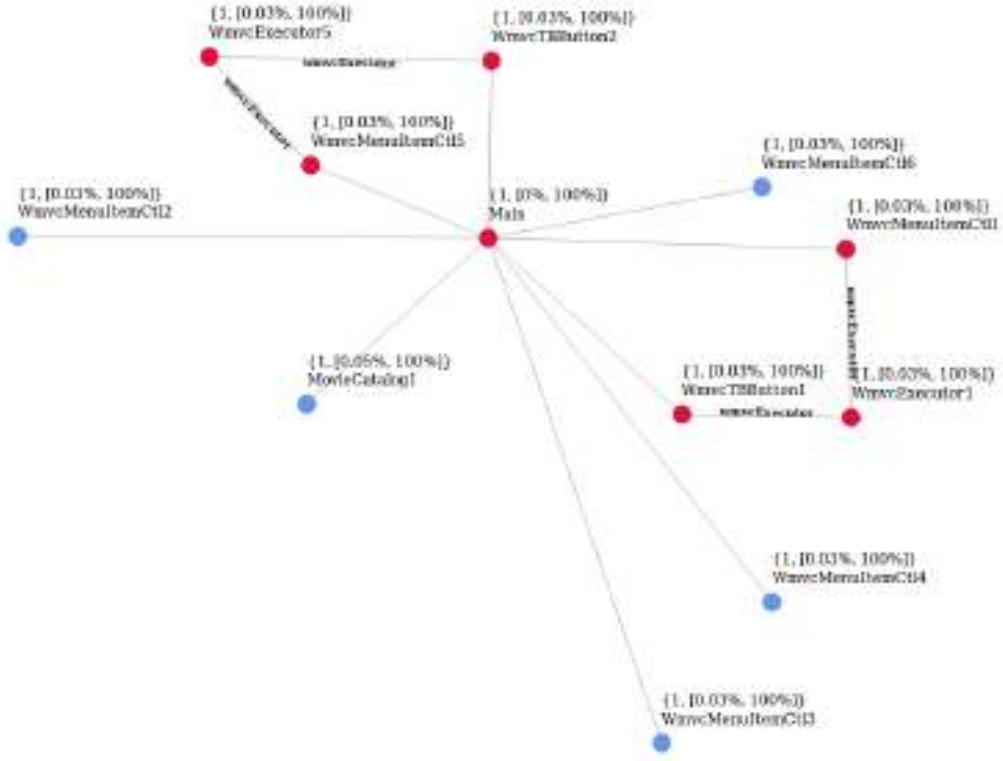
$$dom(n) = \begin{cases} \{n\} \text{ if n is the root of the graph} \\ \{n\} \cup (\cap_{(m \in predec(n)\}} dom(m)) \text{ otherwise} \end{cases} \quad (3)$$

where: *predec(n)* = the set of all predecessors of the node n.

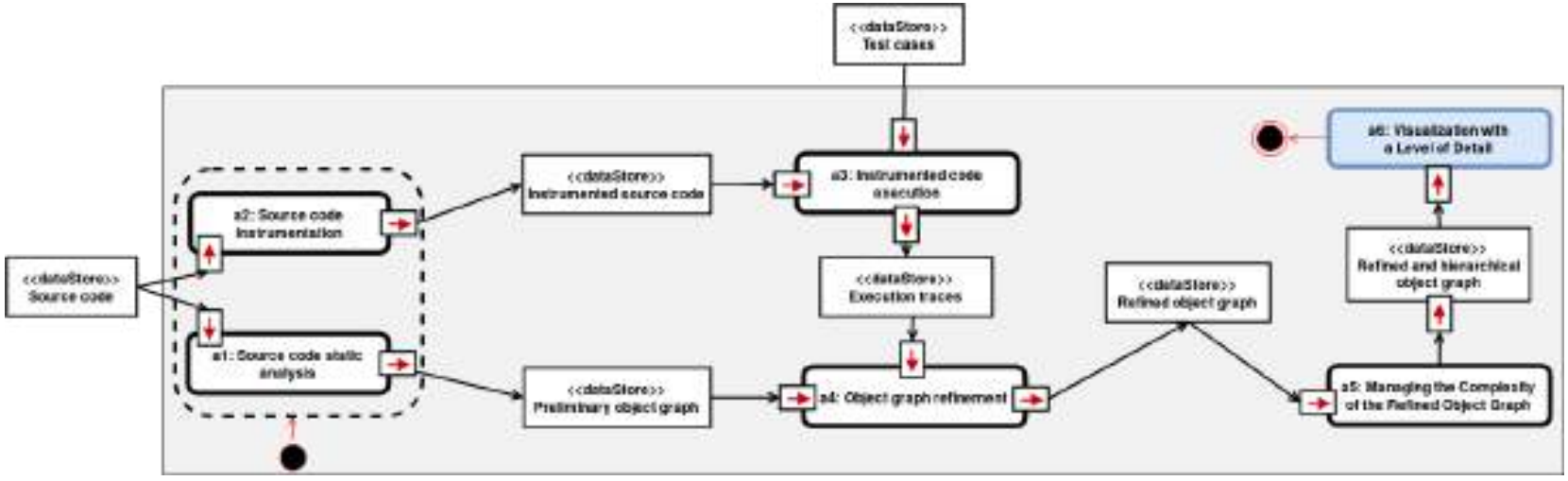
Movie Catalog example



Movie Catalog example



Process in a nutshell



a6: Visualization with a LOD

- A user-oriented visualization
  - The user steers the displayed view to concentrate on focal nodes
- Settings are defined through an interface
  - Settings include thresholds for lifespan interval and showing/hiding composite structures…
- Filtering is launched to remove nodes not fulfilling the user settings

Outline

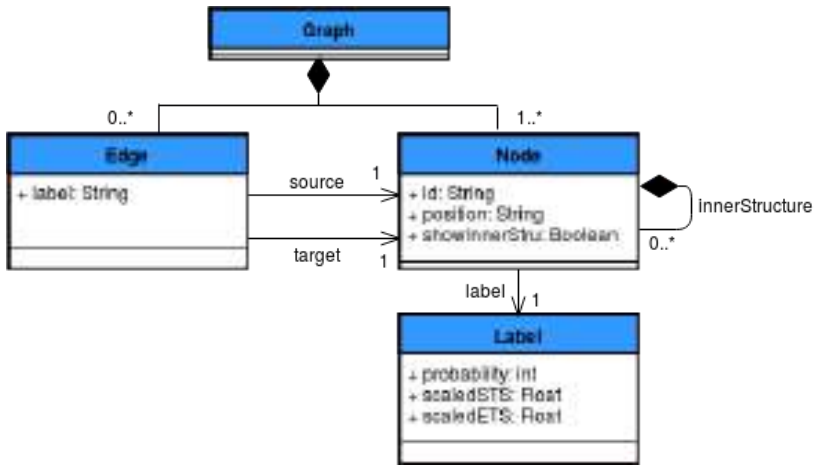
Case Study Setup: implementation

- DIALOG Tool:
  - DIALOG for *refineD and hIerArchicaL Object Graph generator*
  - a1 & a2 are implemented using Spoon[8]
  - For composite structure identification, the Lengauer-Tarjan algorithm was implemented
    - » The most widely used fast dominance algorithm[9]
  - Available here:
    `https://gite.lirmm.fr/zellagui/DIALOGTool/`

[8] R. Pawlak et al. Spoon: A library for implementing analyses and transformations of Java source code. Software: Practice and Experience, 2015. Wiley

[9] S Blazy et al, Validating Dominator Trees for a Fast, Verified Dominance Test. ITP 2015

Case Study Setup: implementation



- Input: Java Source Code > Output: graphs in JSON
- Object of the case study: Jext (211 classes)
  - Small sized and tractable for a case study session

Research Questions

- **Main question:** To what extent the output of the proposed method contributes to comprehension?
- This question was decomposed into three sub-questions:
  - To what extent the refined and the hierarchical object graph contributes in **reducing complexity**?
  - To what extent the refined and the hierarchical object graph contributes in **reducing the time** spent for completing typical program comprehension tasks?
  - To what extent the refined and the hierarchical object graph contributes to **identifying refactoring opportunities**?

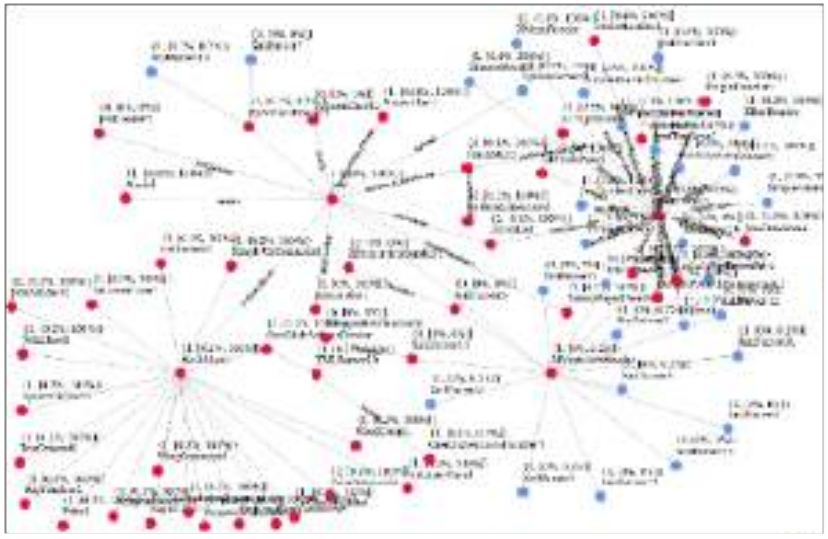
RQ1: Reducing complexity

- Number of nodes in the flat OG = 203
  - With the composite structures, Num nodes=12, HR=16.91
  - With the composite structures, lifespans and empirical probabilities, Num nodes=8, HR=25.4

L	Exploiting composite structures			probability = 1 & lifespan length> 5%		
	#Rev Obj	Avg inner nodes per composite	HR	#Rev Obj	Avg inner nodes per composite	HR
0	12	-	6.42	8	-	6.5
1	65	10.83	2.14	44	13.66	2.17
2	88	2.67	1.15	61	6.77	1.18
3	26	1.13	1.05	21	1.33	1.02
4	10	2.5	1	3	1.5	1
5	2	2	1	2	2	1

- Comprehension task handling becomes effortful and time-consuming when the number of nodes and edges increases

RQ1: Reducing complexity



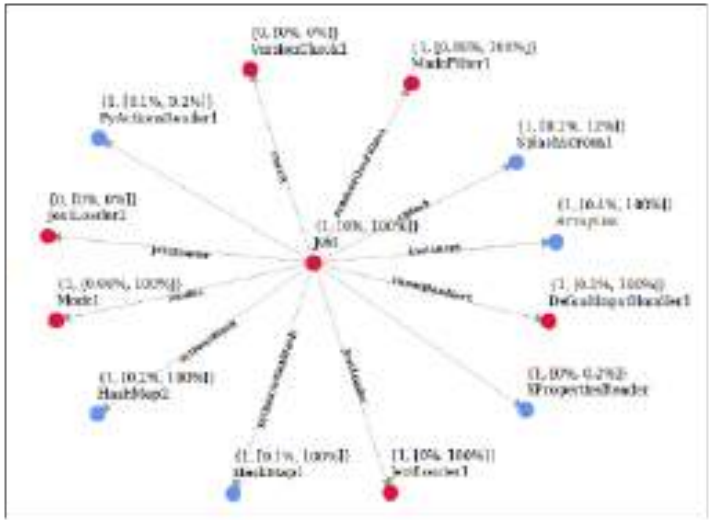
Flat OG

Number of nodes = 203

LEGEND

Composite node

Simple node



Hierarchical OG

Number of nodes = 12

RQ2: Time and correctness

- We followed the expe. design used by Cornelissen et al[10]
  - we adapted the comprehension tasks to the context of Jext
- 6 Ph.D students assigned to 2 groups (3 in each group)
  - SrcCode group: were given only the source code
  - SrcCode+OG group: were given the source code and the OG

Tasks
What are the main stages in a typical Jext scenario? (formulate your answer from a high level perspective)
Name five text treatments/actions supported by Jext: which classes are responsible on these treatments? when were they created? who created them?
In general terms describe the life cycle of the org.jext.Mode class: when is it created? who created it? how many languages are supported in Jext?

[10] B. Cornelissen et al. Trace visualization for program comprehension: A controlled experiment. In ICPC 2009.

RQ2: Time and correctness

Group	Time spent			
	Min	Max	AVG	Dif
SrcCode	73	146	111.7	-
SrcCode+OG	41	68	62.7	-43.9%

- SrcCode+OG group required (-43.9%) less time
  - this is due to the fact that the SrcCode group lost time in scrolling between source code files

RQ3: Refactoring opportunities

- We studied the impact of the availability of the OG on the detection of the *Poltergeist* anti-pattern
- Poltergeists are classes with limited responsibilities and roles to play in the system
  - their life cycle is quite brief
- We used an automatic detection by selecting only the objects that have a lifespan less than or equal to 12% in the recovered OG
- We asked a third party to make a manual detection based on a UML specification of this anti-pattern[11]
- Recall = Precision = 0.87

[11] M. T. Llano et al. UML specification and correction of OO antipatterns. In ICSEA 2009

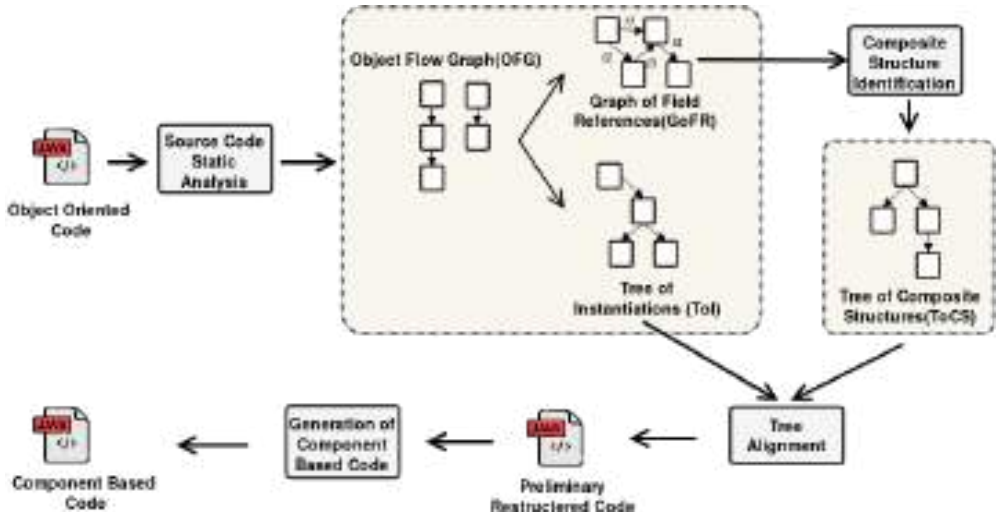
Outline

Conclusion

- The goal of this work is to support understanding during software maintenance by recovering the as-built architecture
- Towards that end, we proposed a five-step process to recover refined and hierarchical object graphs of object oriented software systems
- These object graphs have the following distinguishing features:
  1. Nodes are labeled with lifespans and probabilities of existence that allow a visualization with a level of detail.
  2. They support the collapsing/expanding of objects to hide/show their internal structure

Extension of this Work

- The recovered architecture is used to migrate object oriented software systems into component based ones



Perspectives

- Extraction of other kinds of information that can be used in the visualization with LoD (static info: size of classes, ...)
- Use the recovered architecture in the identification of **(micro-)services**, which are objects (atomic or composite) that are:
  - created and destroyed by the same objects (service providers)
  - and used by different objects (service consumers)

Thanks for your attention