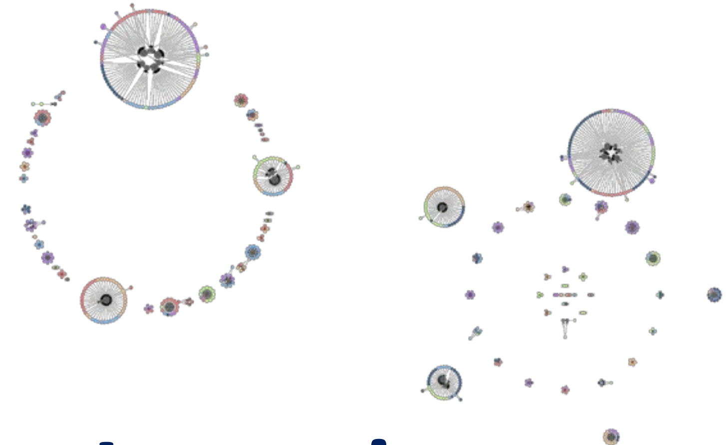
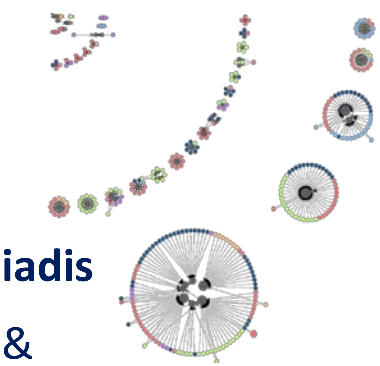


Visual Maps for Data-Intensive Ecosystems



Efthymia Kontogiannopoulou*

Petroleum Geo-Services

Oslo, Norway

Petros Manousis, Panos Vassiliadis

Dept. of Computer Science & Engineering,

Univ. Ioannina, Hellas

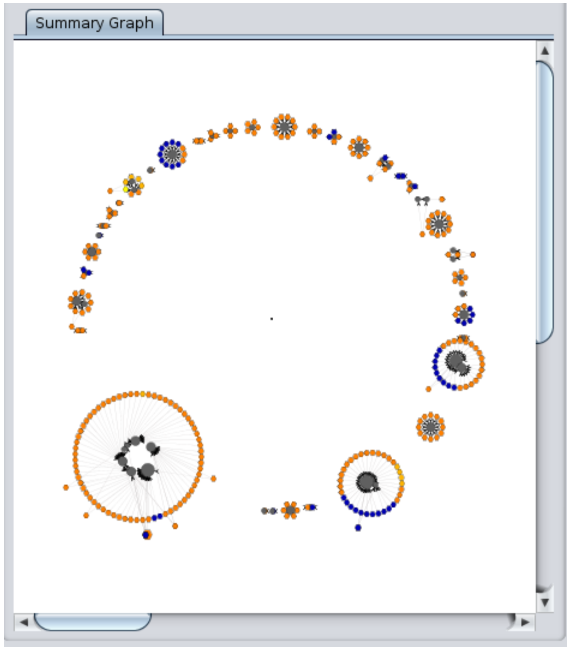
*work conducted while in the Univ. Ioannina

Univ. of Ioannina

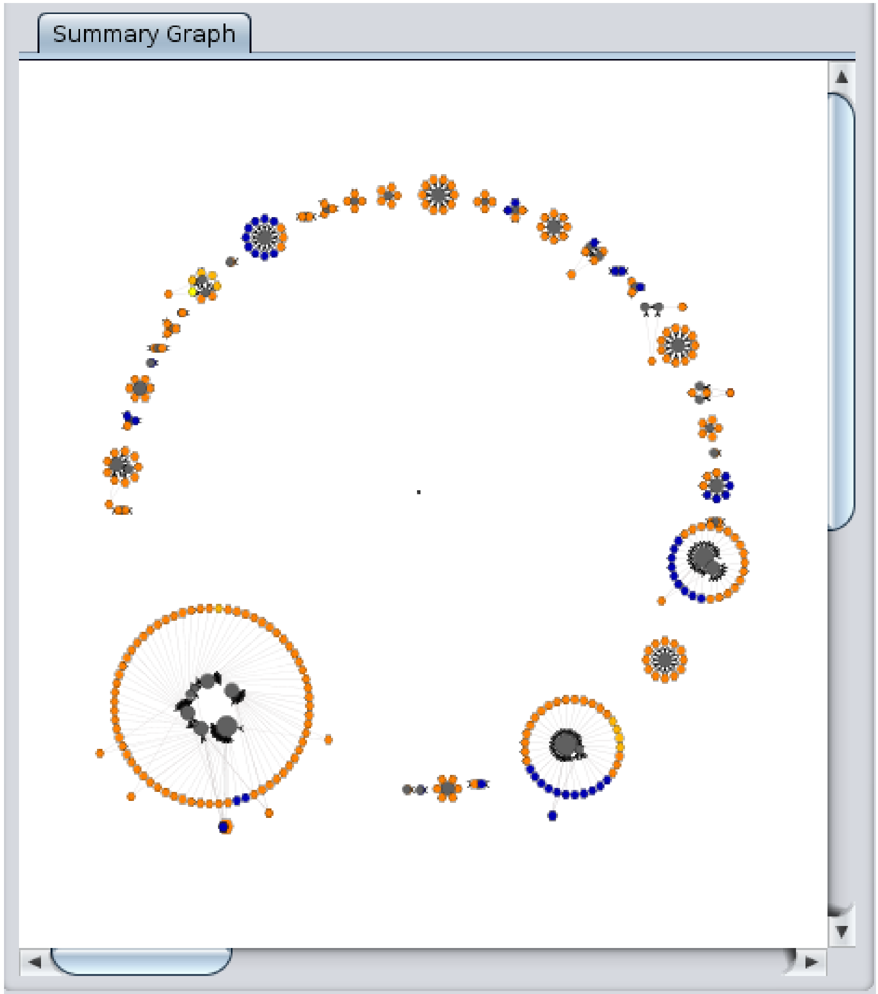
This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

*How do we make a map for a data-intensive software ecosystem? **

Information system with applications **built around a central db and **lots of queries** blended in their code, thus having **strong code-db dependencies***



Why do we need these maps?

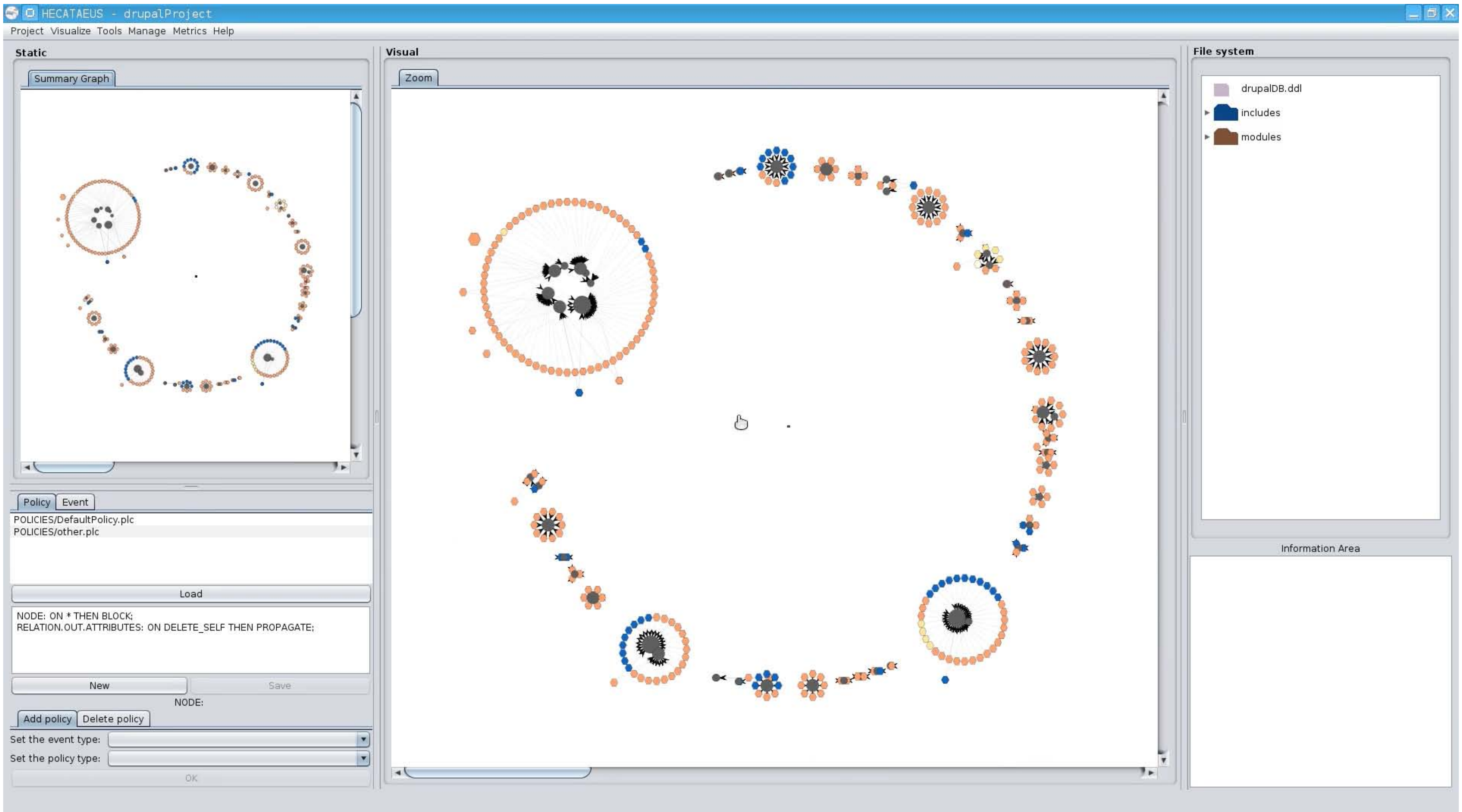


- Documentation,
- Program comprehension
- impact analysis

"Programmers spend between 60-90% of their time reading and navigating code and other data sources ... Programmers form working sets of one or more fragments corresponding to places of interest ...
Perhaps as a result, programmers may spend on average 35% of their time in IDEs actively navigating among working set fragments ..., since they can only easily see one or two fragments at a time."

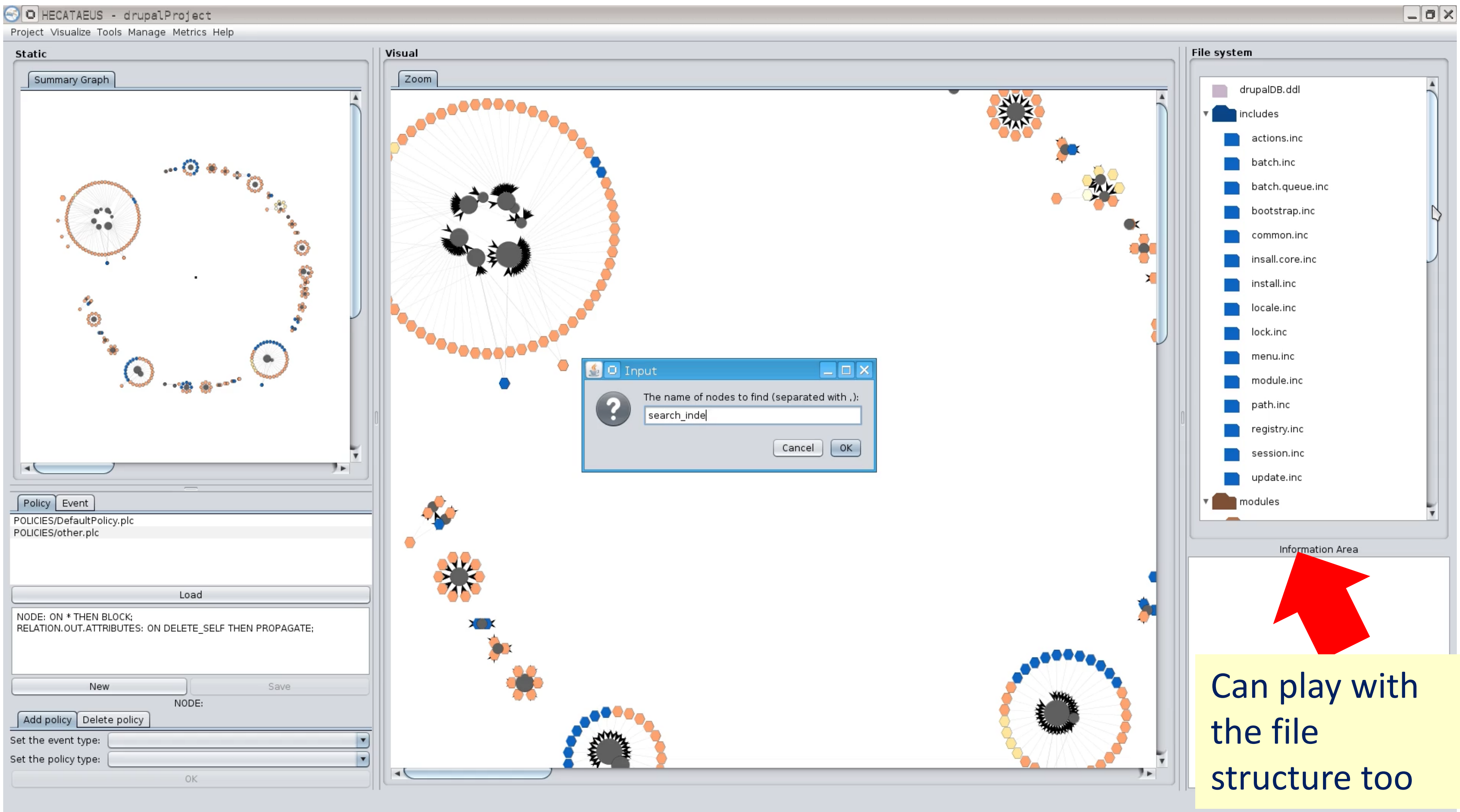
Bragdon et al. Code bubbles: rethinking the user interface paradigm of integrated development environments. ICSE (1) 2010: 455-464.

Circular placement for Drupal



Hecataeus tool:
http://www.cs.uoi.gr/~pvassil/projects/hecataeus/

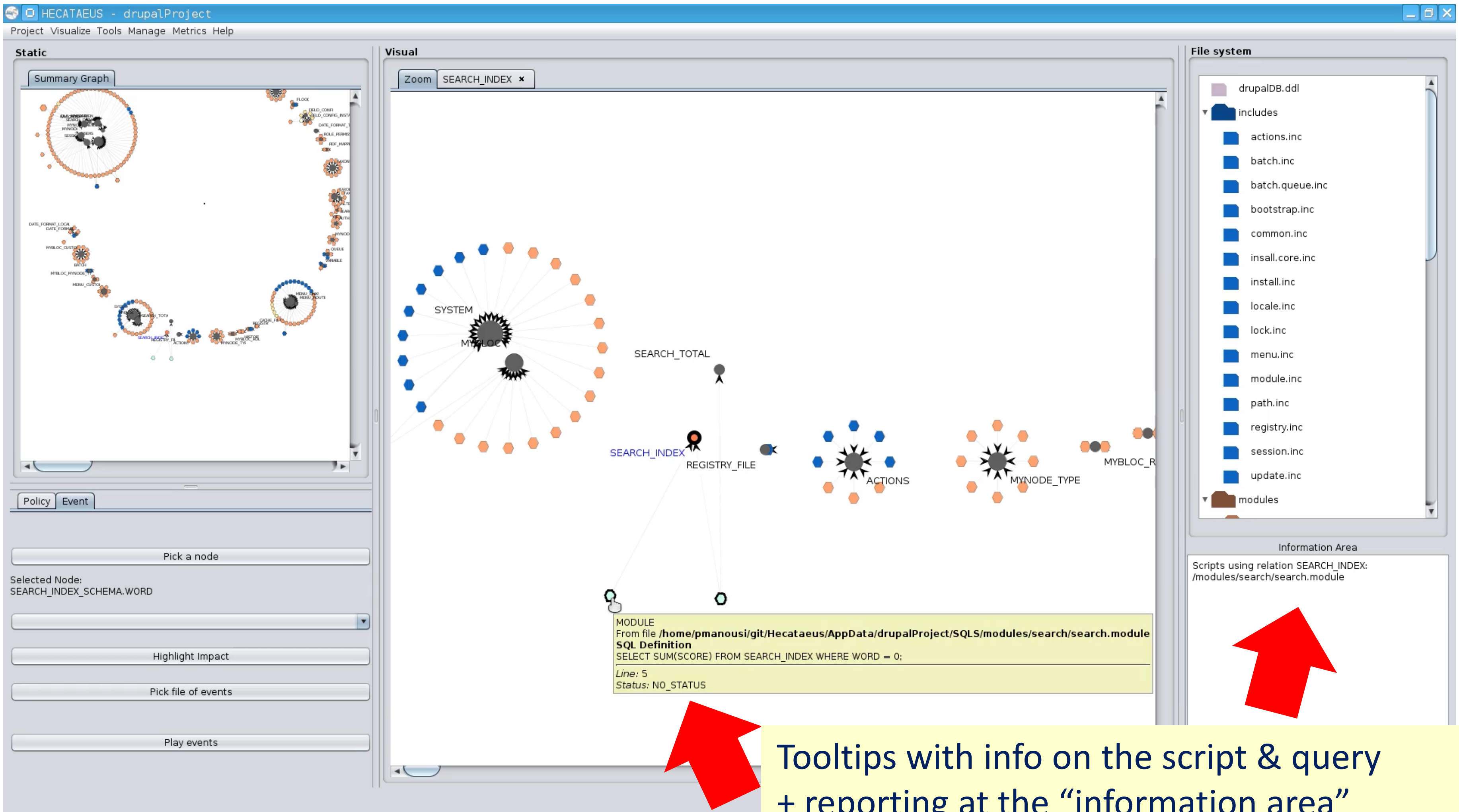
What happens if I modify table search_index? Who are the neighbors?



Can play with the file structure too

Hecataeus tool: **http://www.cs.uoi.gr/~pvassil/projects/hecataeus/**

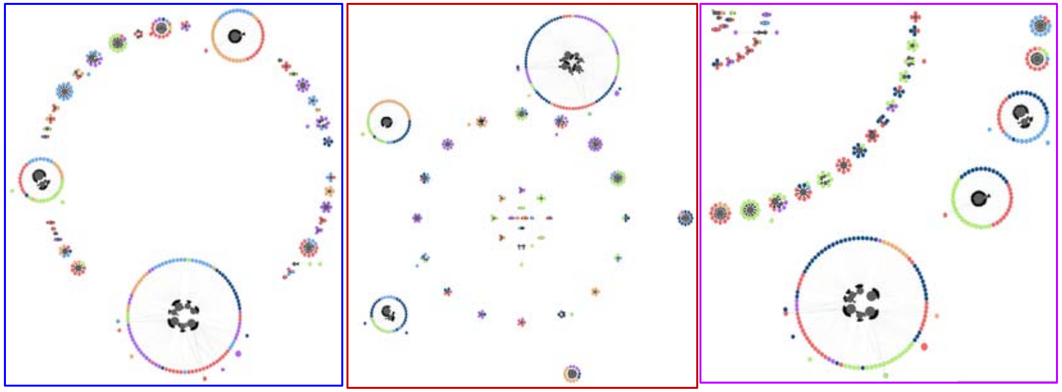
What happens if I modify table search_index? Who are the neighbors?

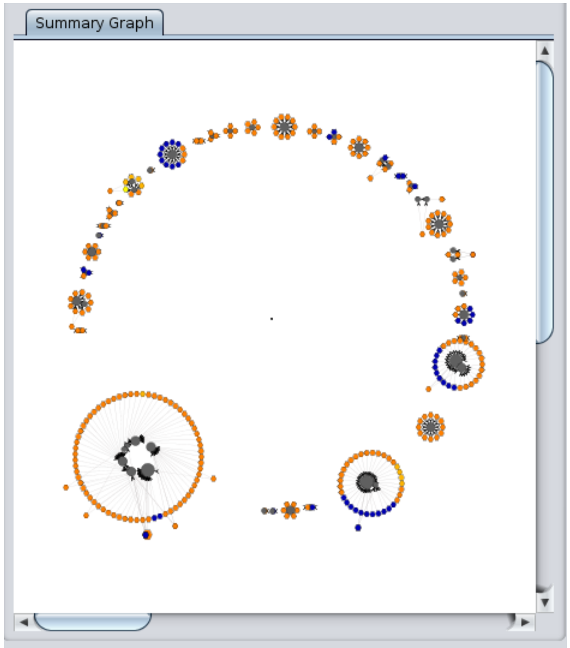


Tooltips with info on the script & query + reporting at the "information area"

Hecataeus tool: **http://www.cs.uoi.gr/~pvassil/projects/hecataeus/**

So, … how do we make a map for a data-intensive software ecosystem?





A **charting method for data-intensive ecosystems,** with a clear target to **reduce visual clutter**.

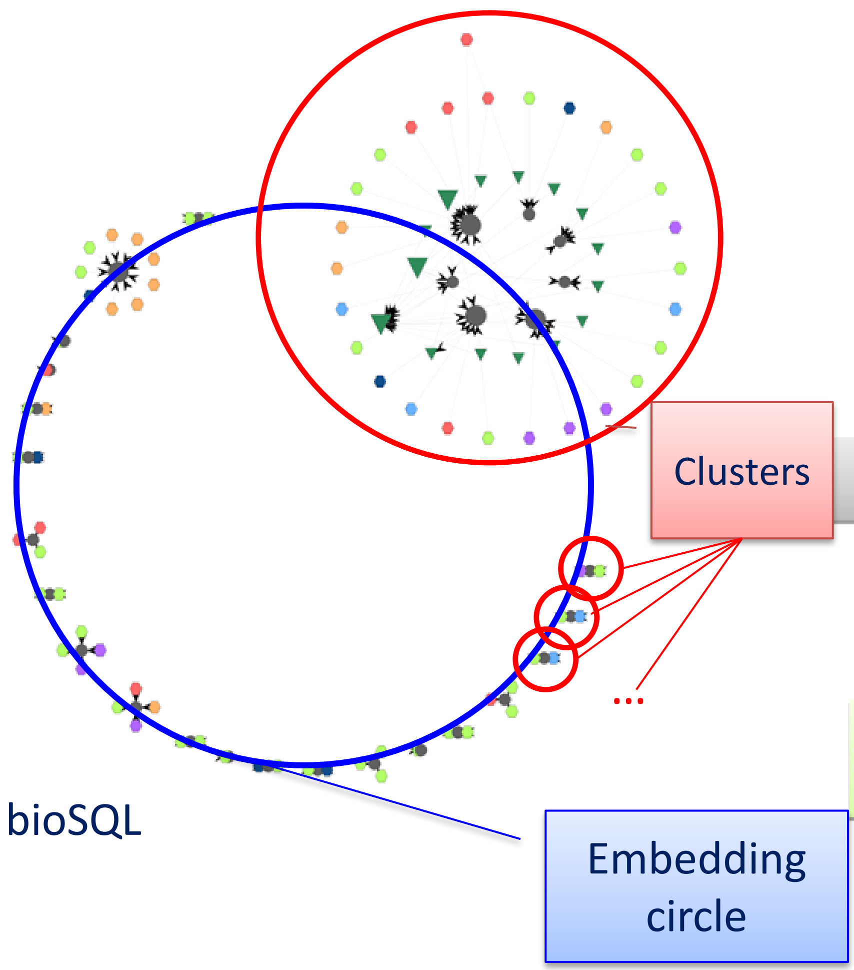
- We exploit a **rigorous, graph-based model** on **code-db dependencies**
  - modules (tables and queries embedded in the applications) as nodes and data provision relationships as edges
- We **cluster** entities of the ecosystem in groups on the basis of their strong interrelationship
- We chart the ecosystem via a set of **radial methods** and provide solutions for
  - … cluster placement …
  - … node placement within clusters …
  - … tuning of visual representation details (shapes, colors, …) …

all **with the goal of reducing visual clutter**

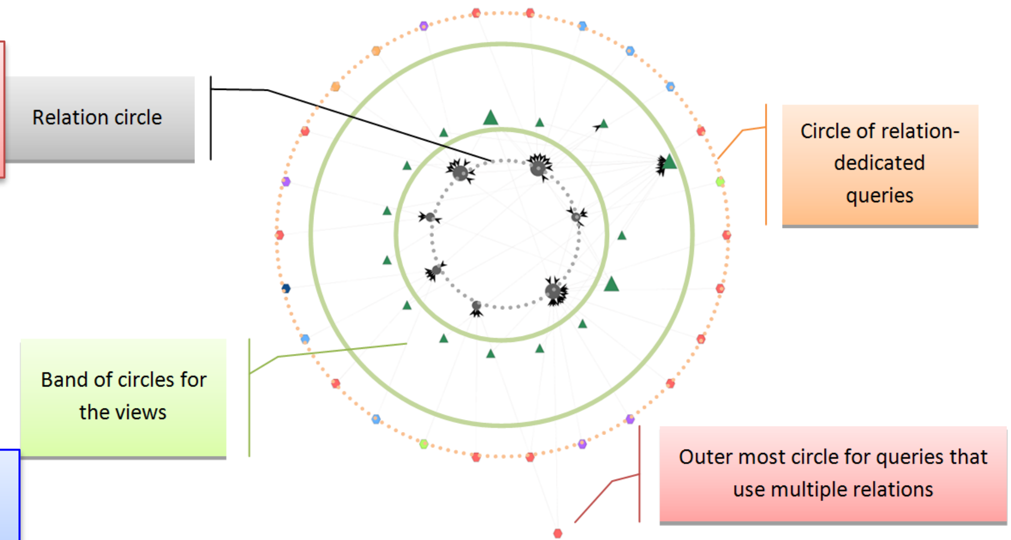
Graphical Notation

- One (or more) **embedding circle** for cluster placement
- **Clusters** are **internally** arranged over **concentric circles**, too

- Node colors: acc. to script for queries; fixed for **tables** & views,
- Node shape: relation / view / query
- Node **size**: acc. to **degree**



The internal structure of a cluster



4 bands of circles, within a cluster:

- 1 circle for relations
- As many as needed for views
- 2 circles for queries

“Transparent” edges for less visual clutter

- **Edges are the main source of visual clutter!**
- So, we reduced the intensity of the edges’ presence of the visual map:
  - we picked a light gray color for the edges and
  - we made them very thin, in terms of weight (almost invisible).
- To retain their info: every time a particular node is selected by the user its neighboring nodes are highlighted with a blue transparent color so, instead of emphasizing edges, we emphasize neighbors.

… and (finally) here is the method to construct the map:

1. **Cluster similar nodes in groups (clusters)**
   - A cluster is a set of relations, views and queries
   - Similarity is determined by the edges
2. **Estimate the space required for each cluster**
   - … to avoid cluster overlaps later
3. 3 alternative methods to **place clusters on a 2D canvas**
   - Single circle
   - Concentric Circles
   - Concentric Arcs
4. **Place the nodes of each cluster in concentric circles, internally in the cluster**

Roadmap

1. Cluster similar nodes in groups (clusters)
2. Estimate the space required for each cluster
3. Three alternative methods to place clusters on a 2D canvas
   - Single circle
   - Concentric Circles
   - Concentric Arcs
4. Place the nodes of each cluster in concentric circles, internally in the cluster
5. Summing up

Roadmap

1. Cluster similar nodes in groups (clusters)

2. Estimate the space required for each cluster

3. Three alternative methods to place clusters on a 2D canvas
   - Single circle
   - Concentric Circles
   - Concentric Arcs

4. Place the nodes of each cluster in concentric circles, internally in the cluster

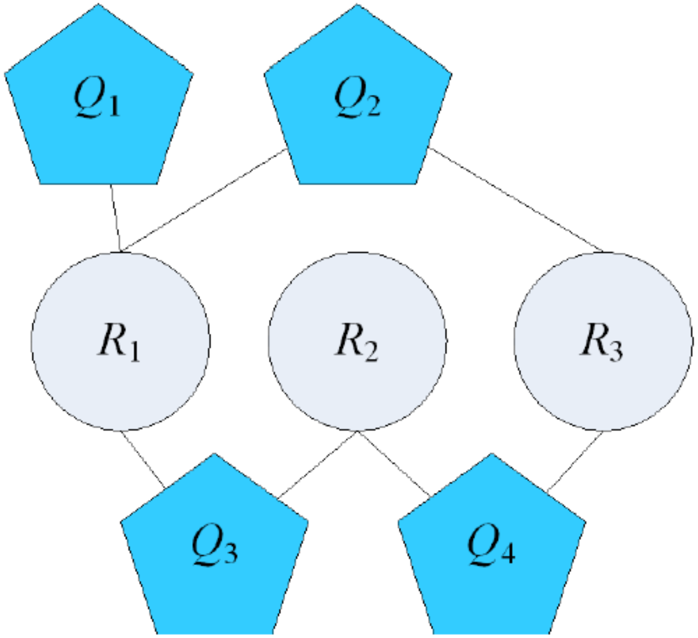
5. Summing up

Step 1: Clustering

- We use agglomerative hierarchical clustering to **group objects with similar semantics** in advance of graph drawing.
- Why? To reduce the amount of visible elements, visualization methods place them in groups, thus
  - reducing visual clutter
  - improving user understanding of the graph
- Principle of proximity: similar nodes are placed next to each other

Step 1: Clustering

$$dist(M_i, M_j) = 1 - \begin{cases} \dfrac{|neighbors_i \cap neighbors_j|}{|neighbors_i \cup neighbors_j|}, & if \ \nexists \ Edge(i,j) \\ \dfrac{|neighbors_i \cap neighbors_j| + 2}{|neighbors_i \cup neighbors_j|}, & if \ \exists \ Edge(i,j) \end{cases}$$



$$s(Q_1, Q_3) = \frac{|\{R_1\} \cap \{R_1, R_2\}|}{|\{R_1\} \cup \{R_1, R_2\}|} = \frac{1}{2}$$

$$s(R_1, Q_3) = \frac{|\{Q_1, Q_2\} \cap \{R_1, R_2\}| + 2}{|\{Q_1, Q_2, Q_3\} \cup \{R_1, R_2\}|} = \frac{2}{5}$$

$$s(R_1, Q_4) = \frac{|\{Q_1, Q_2\} \cap \{R_2, R_3\}|}{|\{Q_1, Q_2\} \cup \{R_2, R_3\}|} = 0$$

Roadmap

✓ 1. Cluster similar nodes in groups (clusters)

2. Estimate the space required for each cluster

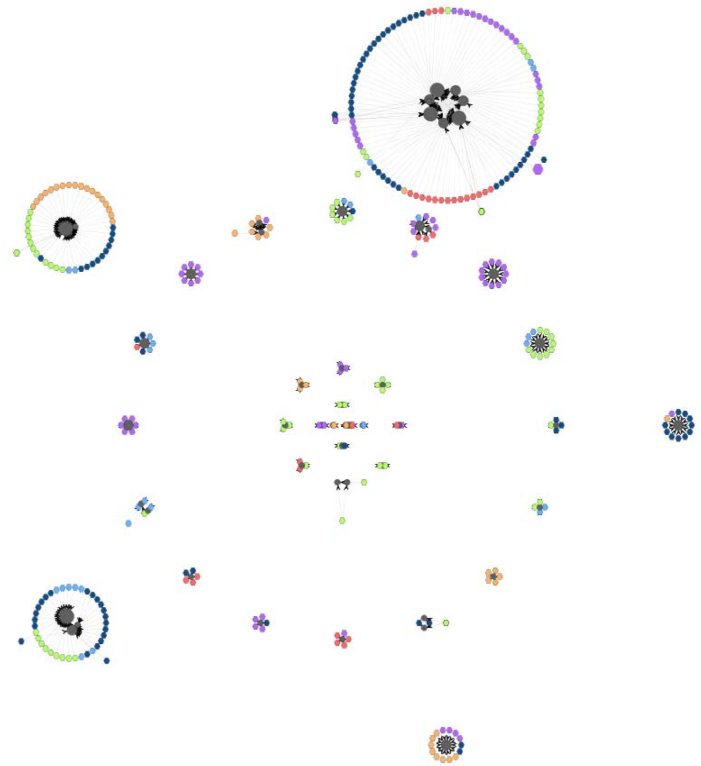
3. Three alternative methods to place clusters on a 2D canvas
   - Single circle
   - Concentric Circles
   - Concentric Arcs

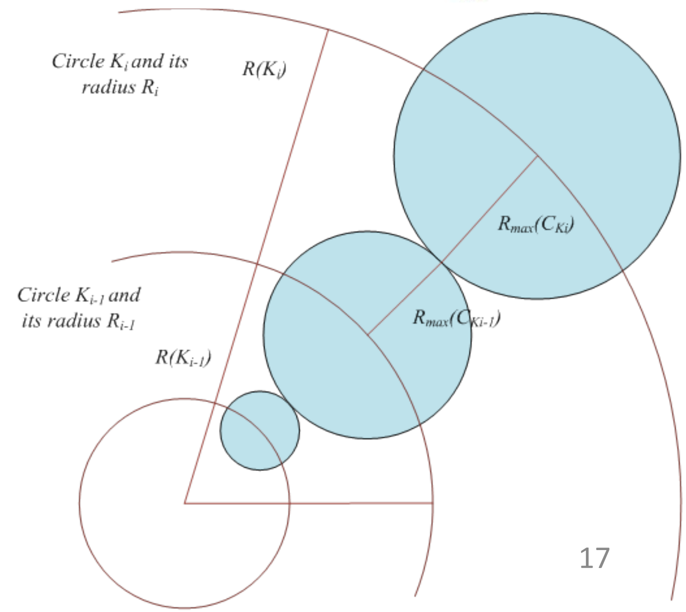
4. Place the nodes of each cluster in concentric circles, internally in the cluster

5. Summing up

Step 2: Estimate the area of each cluster

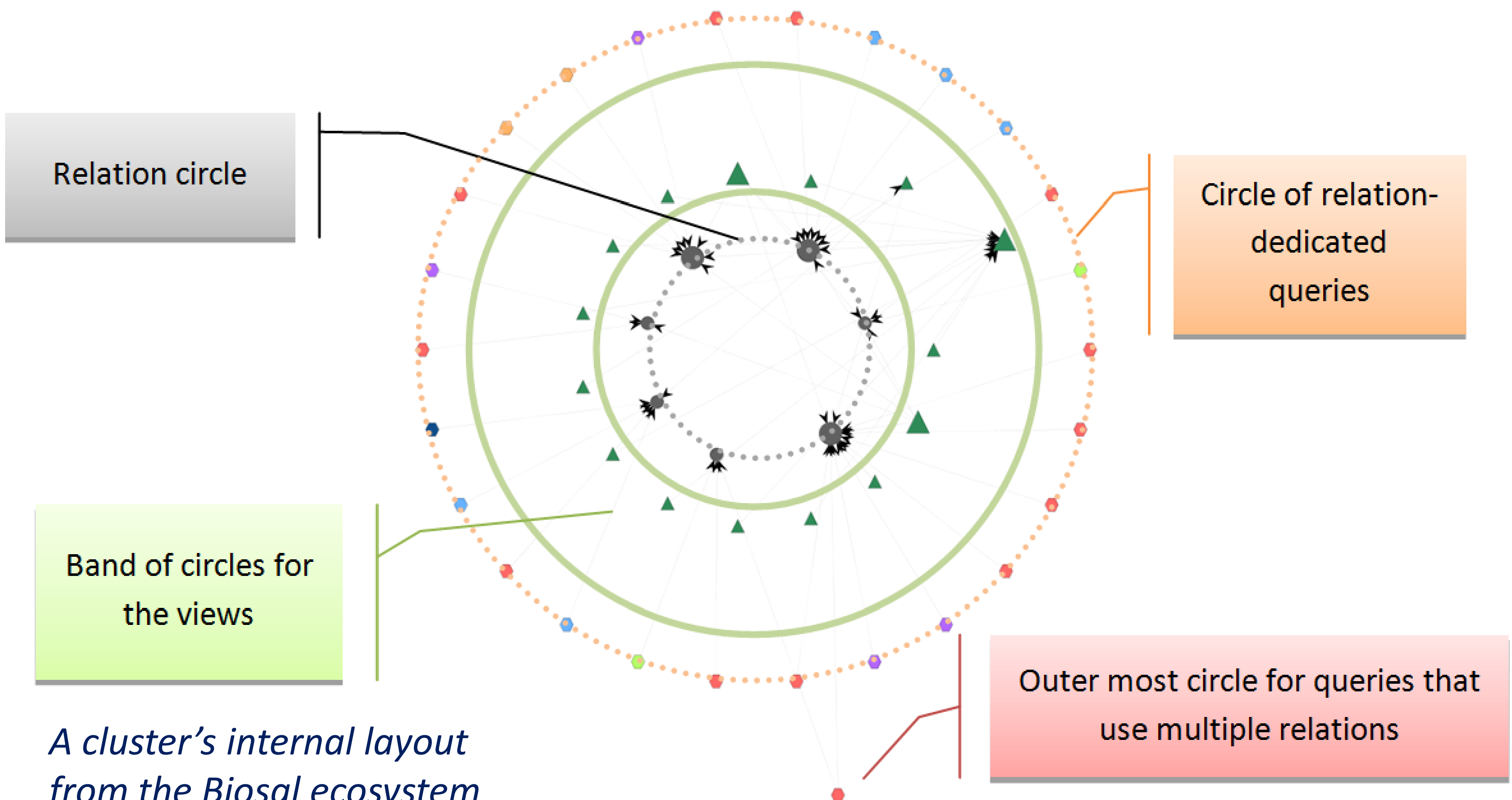
- Once the clusters have been computed, before placing them on the 2D canvas, the next step is to estimate the space required for each cluster
- This step is crucial and necessary for the subsequent step of cluster placement, in order to be able to
  - **calculate the radius and area each cluster,** and thus,
  - **arrange the clusters without overlaps**





Step 2: Estimate the area of each cluster

Each cluster includes **3 bands of concentric circles: relations (1 circle), views, queries (2 circles)**



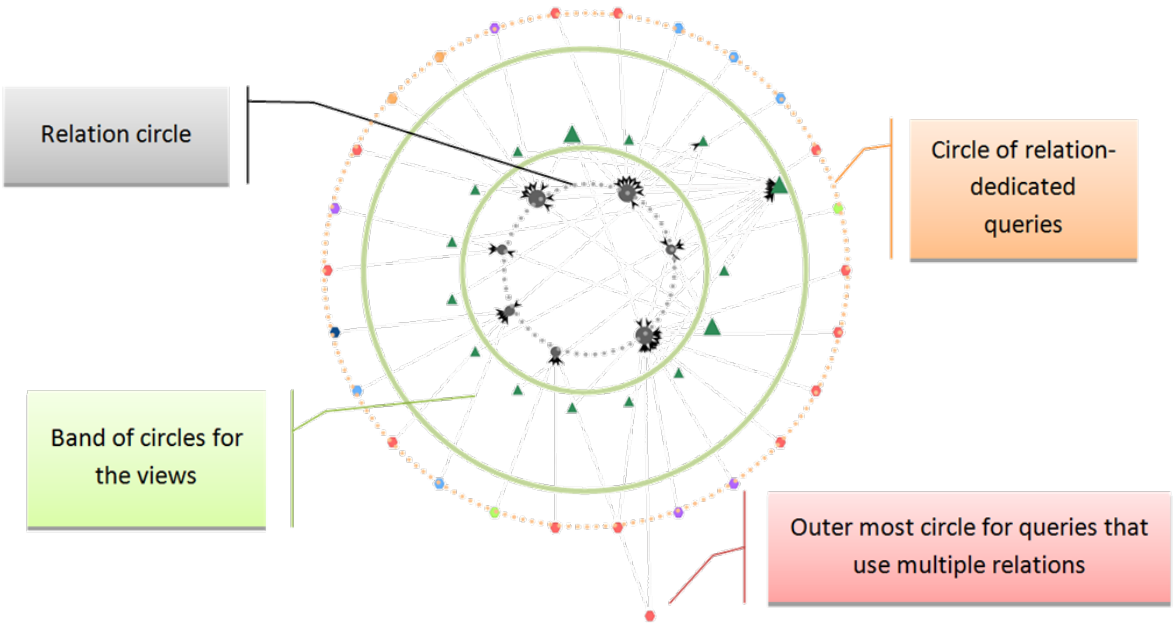
A cluster's internal layout from the Biosql ecosystem

Step 2: Estimate the area of each cluster

1. We determine the clusters' circles and their nodes:
   – We **topologically sort** cluster nodes in **strata** – each stratum becomes a circle
2. Then, we compute the radius for each circle:

$$R_i = 3 * \log(nodes) + nodes$$

The outer circle gives us the radius of this cluster



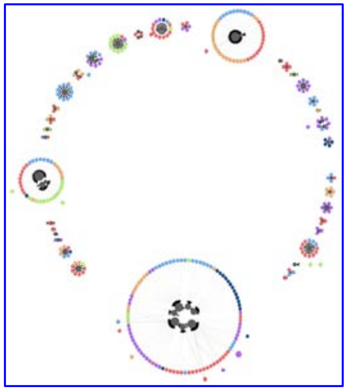
A cluster's internal layout from the Biosql ecosystem

Roadmap

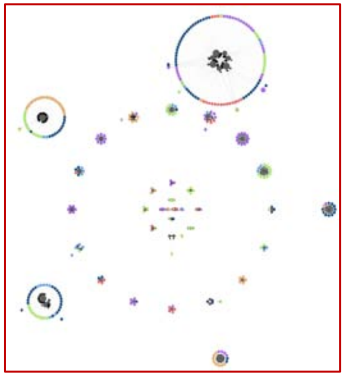
- ✓ 1. Cluster similar nodes in groups (clusters)
- ✓ 2. Estimate the space required for each cluster
- **3. Three alternative methods to place clusters on a 2D canvas**
  - Single circle
  - Concentric Circles
  - Concentric Arcs
- 4. Place the nodes of each cluster in concentric circles, internally in the cluster
- 5. Summing up

Step 3: Laying out the Clusters

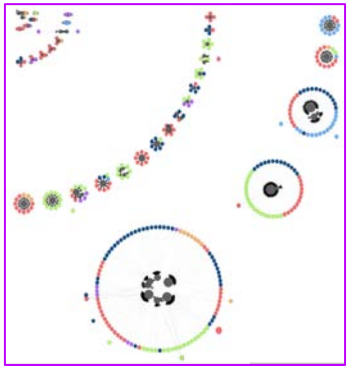
3 alternative methods for placing the clusters on a 2D area



- Circular placement
  - all clusters on a single embedding circle

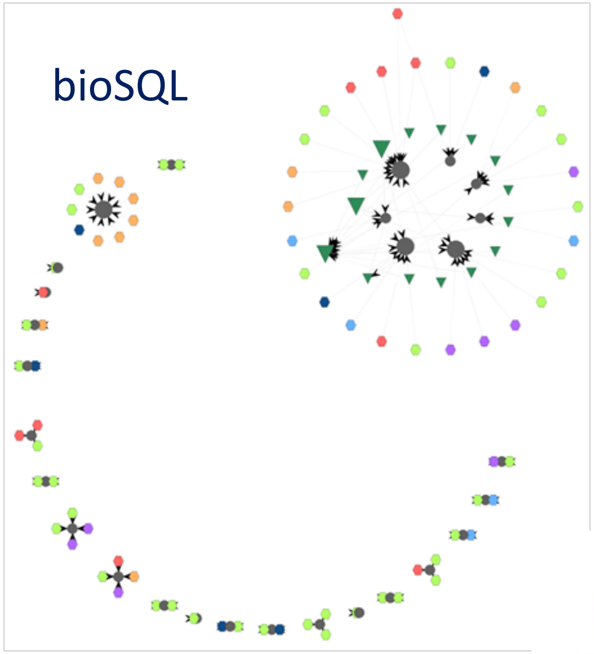


- Concentric circles
  - trying to reduce the intermediate empty space

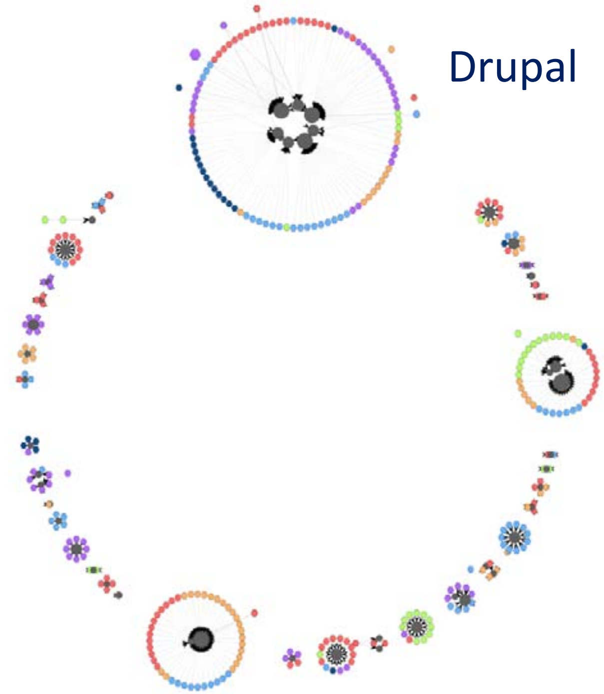


- Concentric arcs
  - a combination of the previous two methods

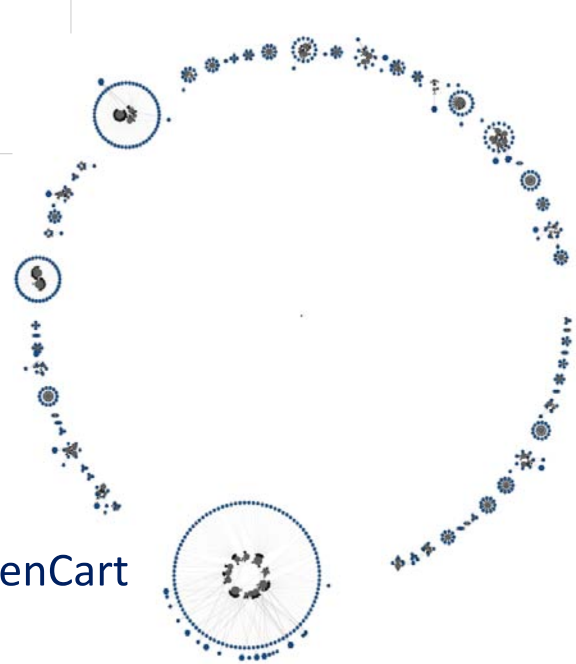
Circular Layout



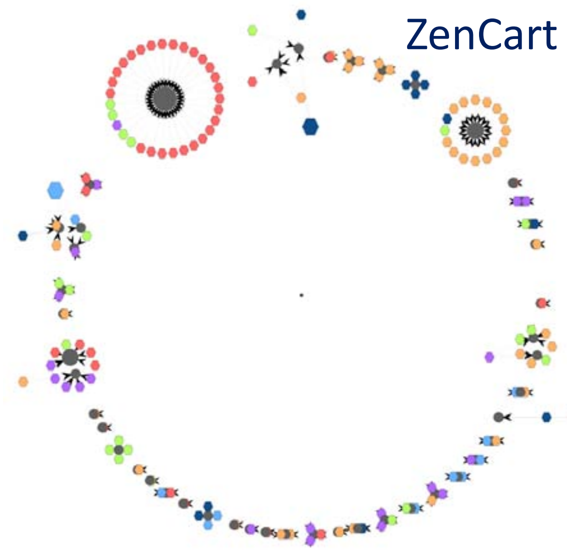
bioSQL



Drupal



OpenCart



ZenCart

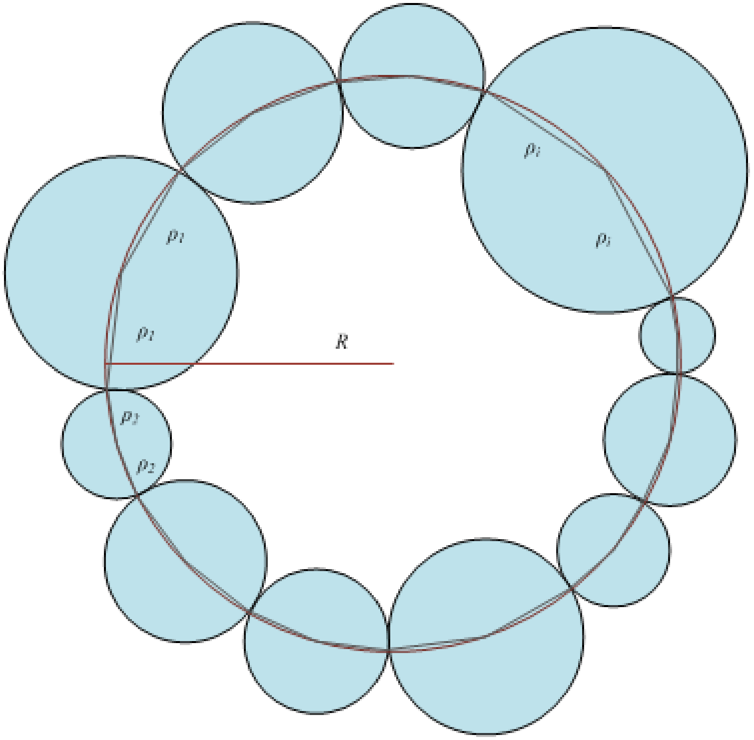
Circular cluster layout

- We use a single embedding circle to place the clusters.
- One sector of the circle per cluster
  - with its angle varying on the cluster's size (#nodes)
  - remember: each cluster is also a circle, approximated by its outermost constituent circle of nodes
- Steps:
  1. Compute *R*, the radius of the embedding circle
  2. Compute φ_i, the angle of each cluster's sector
  3. Add some extra whitespace
  4. Compute the coordinates of all clusters

```
compute R;
compute φi;
whitespace;
coordinates.
```

Circular Layout: Embedding Circle determination

- Given: the radius r_i of each cluster i
  Compute: R, the radius of the embedding circle.
- **Method**:
- approximate the circles periphery *($2\pi R$)* by the sum of edges of the **embedded polygon**
- divide this sum by 2π to calculate the radius R of the embedding circle



$$2\pi R \cong \sum_{i=0}^{|C|} 2\rho_i => R \cong \sum_{i=0}^{|C|} 2 * \rho_i / 2\pi$$

```
compute R;
compute φi;
whitespace;
coordinates.
```

Circular layout: calculation of the angle for each of the segments

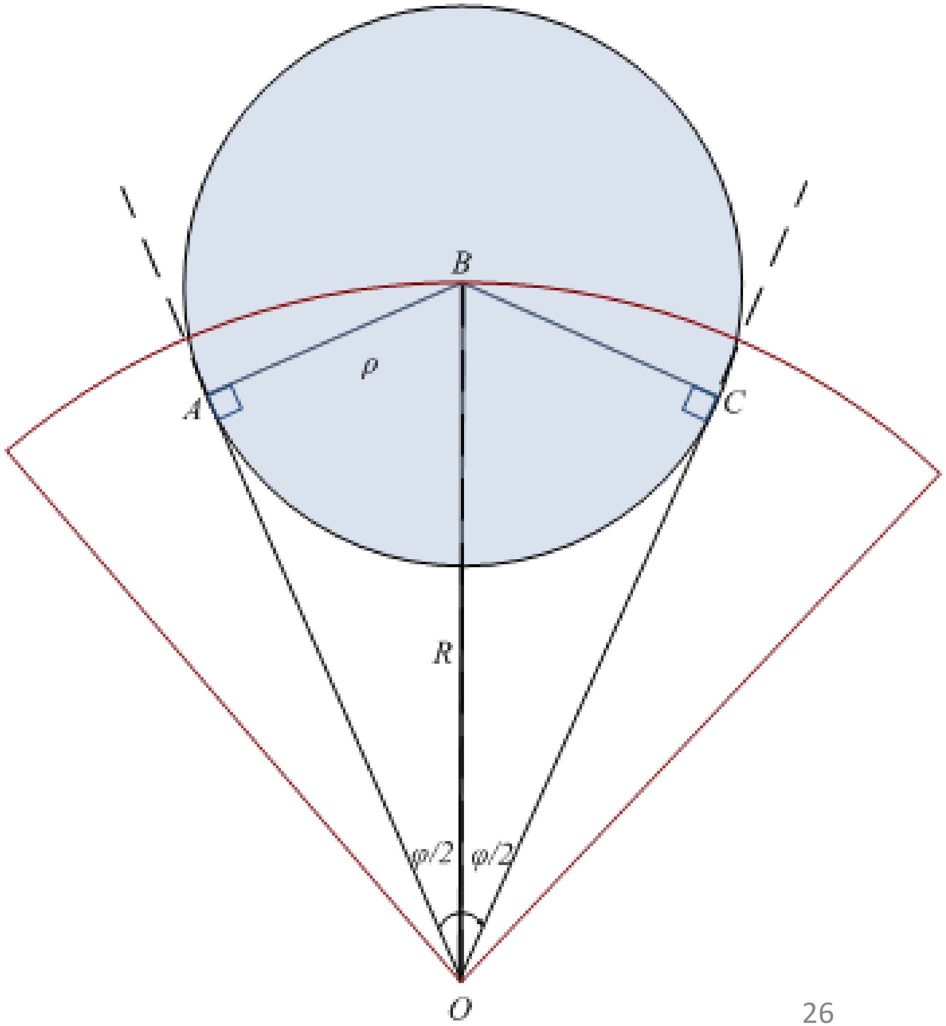
- Goal: assign each cluster to a segment of the circle depending on the cluster's radius (size).
- Each these segments is defined by an angle φ over the embedding circle.
- Not as obvious as it seems – we have to consider two cases:
  - The radius ρ of the cluster we want to place is smaller or equal to the radius of the embedding circle R
  - The radius ρ of the cluster we want to place is greater than the radius of the embedding circle R

compute R;
~~compute φ_i;~~
whitespace;
coordinates.

Circular layout: calculation of the angle for each of the segments

- Typical case, where $\rho \leq R$
- Consider the left triangle *ABO*
- Then:

$$\varphi/2 = \sin^{-1}\left(\frac{\rho}{R}\right)$$



```
compute R;
compute φi;
whitespace;
coordinates.
```

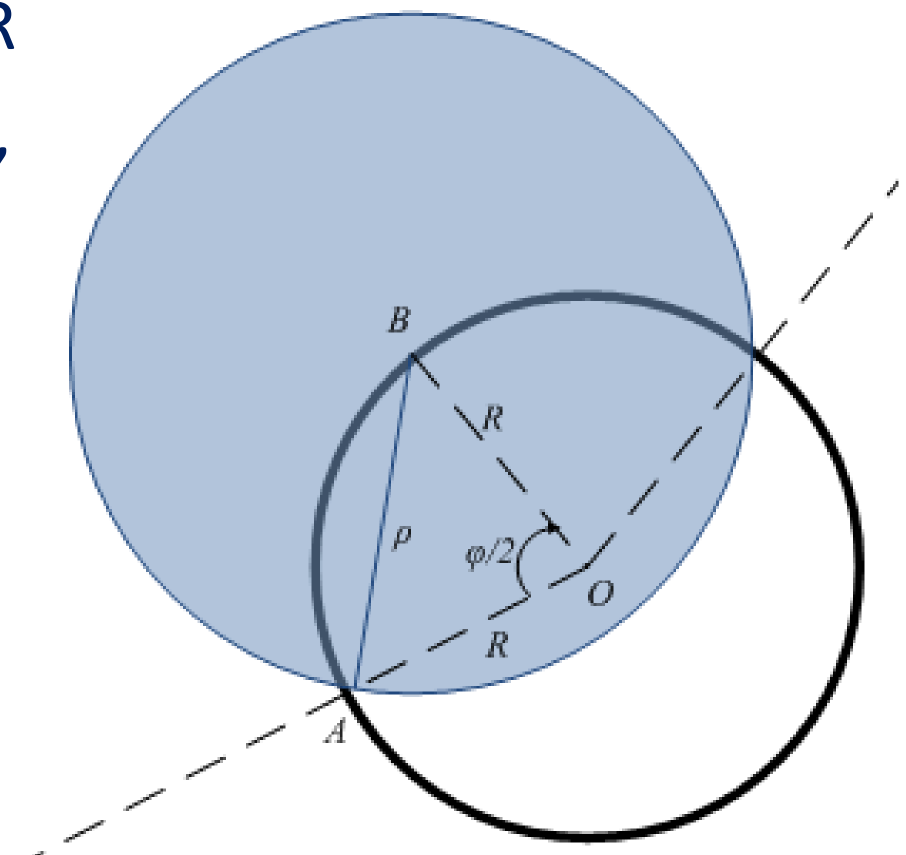
Circular layout: calculation of the angle for each of the segments

- A large cluster occurs ρ > R
- Assume the isosceles ABO, both AO,BO = R
- Then:

$$\varphi/2 = \cos^{-1}\left(\frac{2R^2 - \rho^2}{2R^2}\right)$$

- due to

$$cos\varphi = (b^2 + c^2 - a^2)/2bc$$



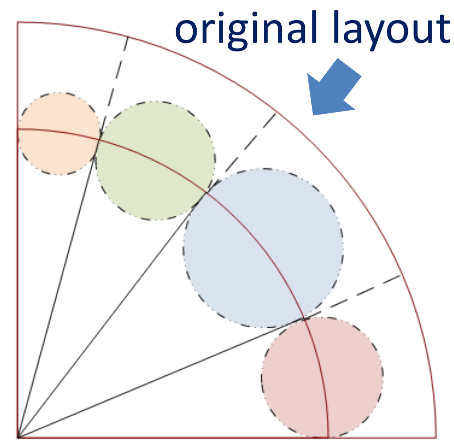
Note: cannot avoid to discriminate the two cases

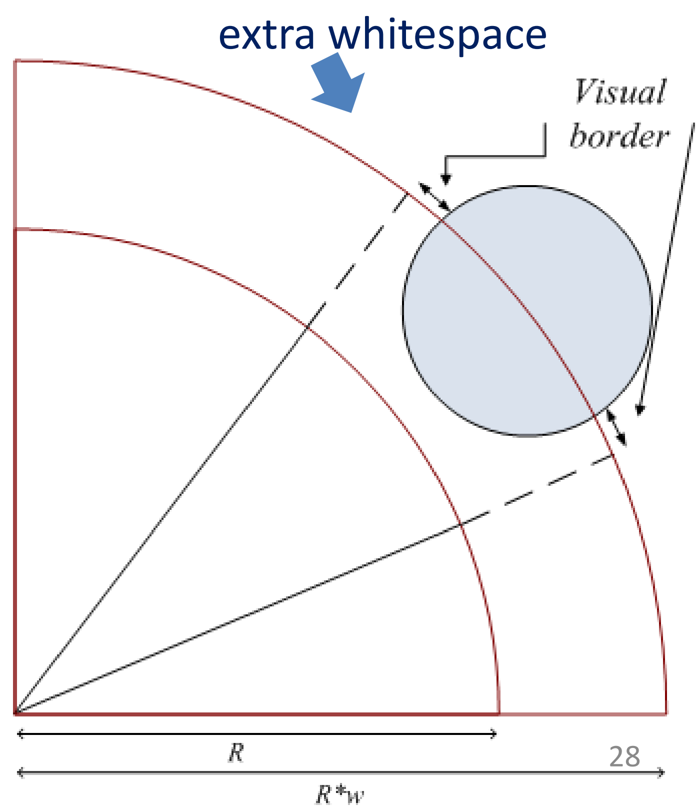
```
compute R;
compute φi;
whitespace;
coordinates.
```

Circular layout: avoid cluster overlap!

- We introduce a white space factor *w* that enlarges the radius *R* of the circle
- Each cluster is approx. by a circle, with
  - radius r (known from step #1)
  - center [c_x, c_y] determined by *φ*, *R*, and *w*.

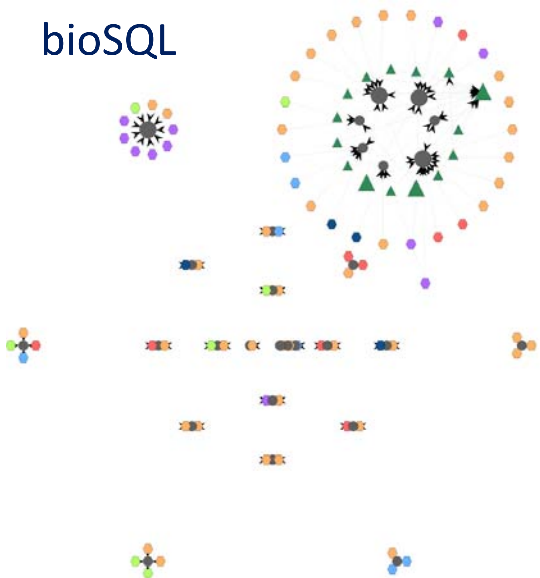
$$c_x = \cos\left(\frac{\phi}{2}\right) * R * w,\ c_y = \sin\left(\frac{\phi}{2}\right) * R * w$$



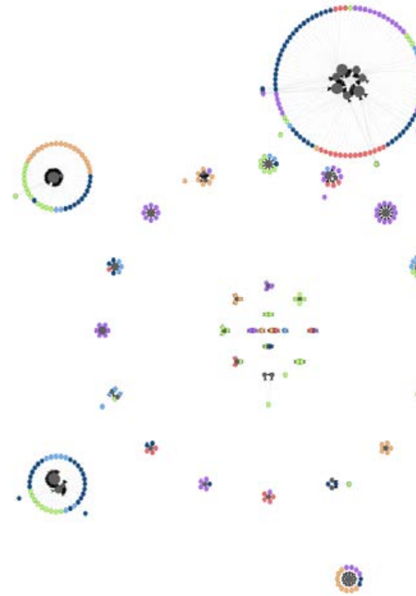


Concentric circles

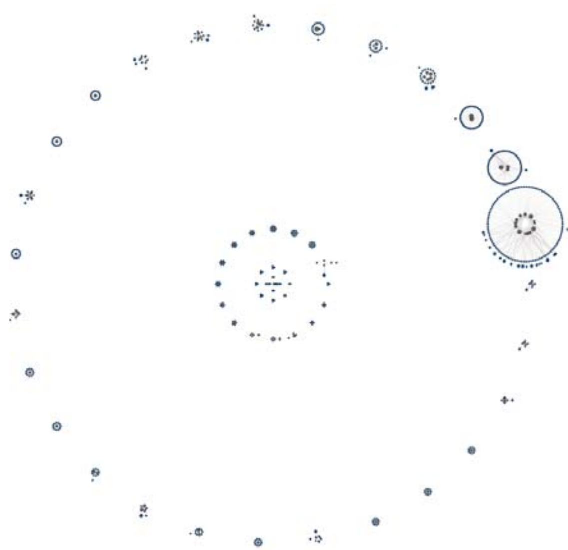
bioSQL



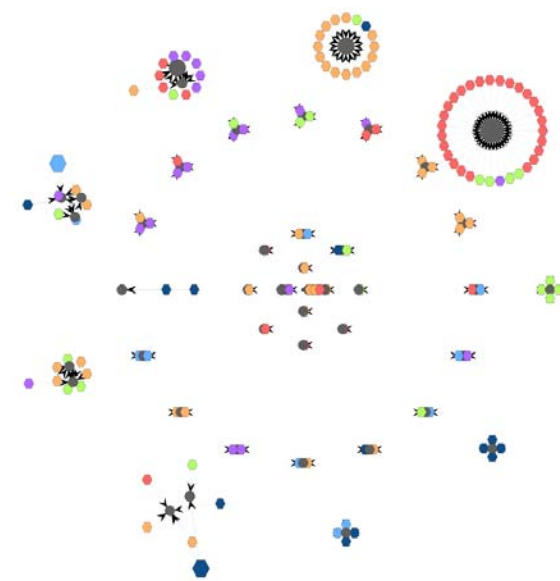
Drupal



OpenCart

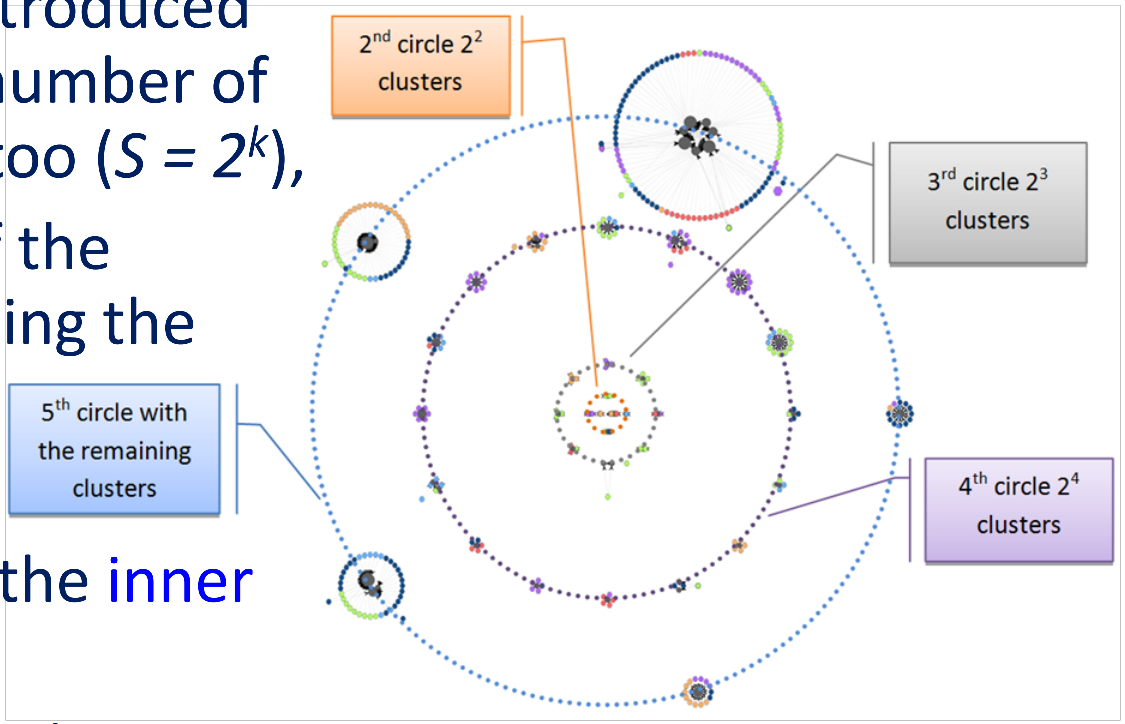


ZenCart



Concentric Circles Layout

- Each circle is split in fragments of powers of 2
  - as the order of the introduced circle increases, the number of fragments increases too ($S = 2^k$),
  - with the exception of the outermost circle hosting the remaining clusters
- This way, we can place
  - the small clusters on the inner circles, and
  - bigger clusters (occupying more space) on outer circles



Concentric Circles Layout

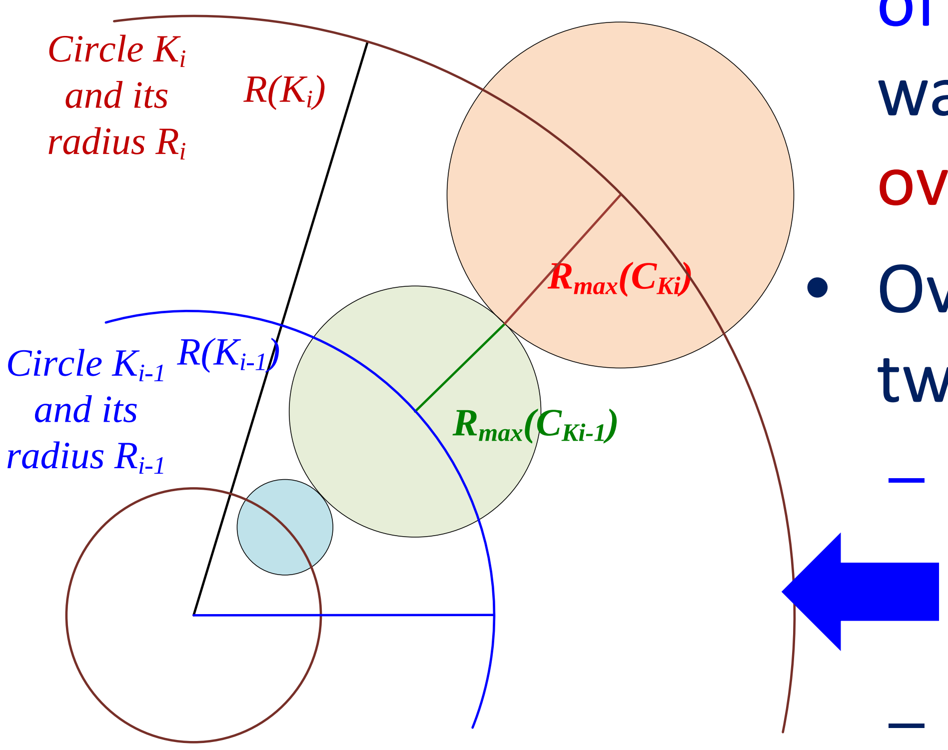
Method:

1. Sort clusters by ascending size in a list L^C
2. While there are clusters not placed in circles
   1. Add a new circle and divide it in as many segments as $S = 2^k$ with k being the order of the circle (i.e., the first circle has 2^1 segments, the second 2^2 and so on)
   2. **Assign the next *S* fragments from the list L^C to the current circle and compute its radius according to this assignment**
   3. Add the circle to a list L of circles
3. Draw the circles from the most inward (i.e., from the circle with the least segments) to the outermost by following the list L.

Main challenge

Concentric Circles: radius calculation

$R(K_i) = R(K_{i-1}) + R_{max}(C_{Ki-1}) + R_{max}(C_{Ki})$



- Instead of having to deal with just one circle, we need to compute the radius for each of the concentric circles, in a way that clusters do not overlap
- Overlap can be the result of two problems:
  - clusters of subsequent circles have radii big enough, so that they meet, or,
  - clusters on the same circle are big enough to intersect.

Concentric Circles: radius calculation for each circle

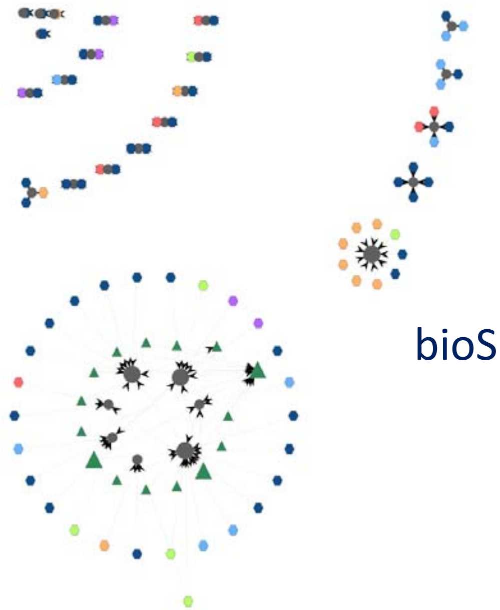
- Finally, **to calculate the radius of a circle**:
  - we take the maximum of the two values of the two aforementioned solutions and
  - we use an additional whitespace factor *w* to enlarge it slightly (typically, we use a fixed value of 1.2 for *w*).

$$R(K_i) \quad = w * max \begin{cases} R_{i-1} + R_{max}(C_{K_{i-1}}) + R_{max}(C_{K_i}) \\ \frac{1}{\pi}\sum_{j=1}^{|C|} R(C_{jK_i}), R(C_{jK_i}) : \text{radius of cluster } C_j \text{on circle } K_i \end{cases}$$

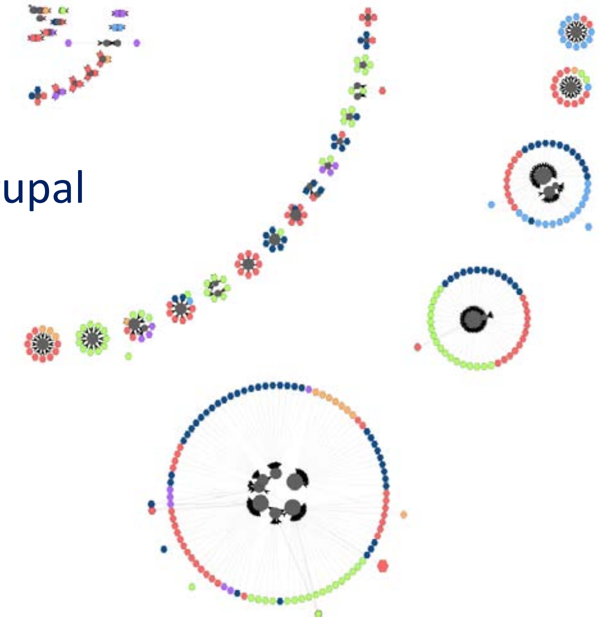
- **Clusters of the same circle** have **equal** segments with an **angle**:

$$\varphi_i = 2\pi/nK_i$$

where *n*: the number of clusters on circle K_i



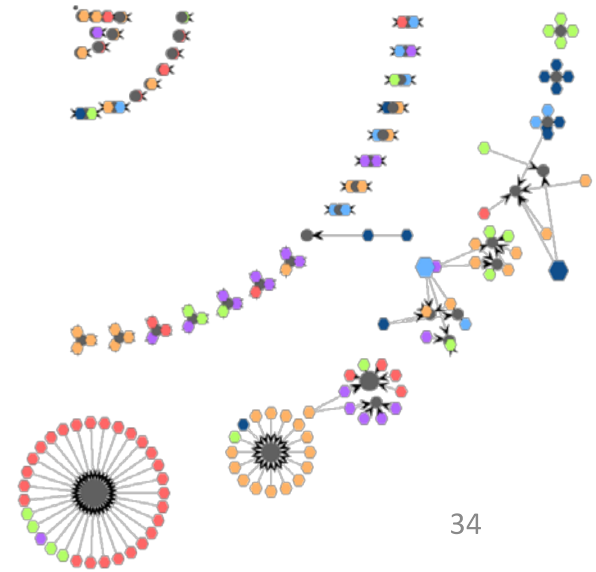
bioSQL

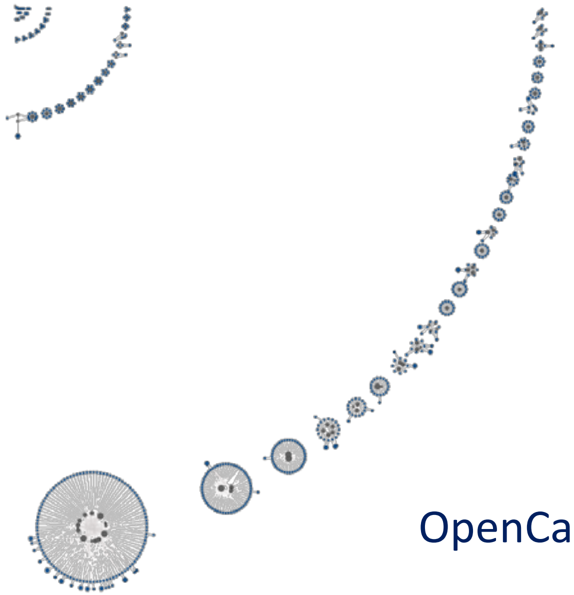


Drupal

Concentric arcs

ZenCart

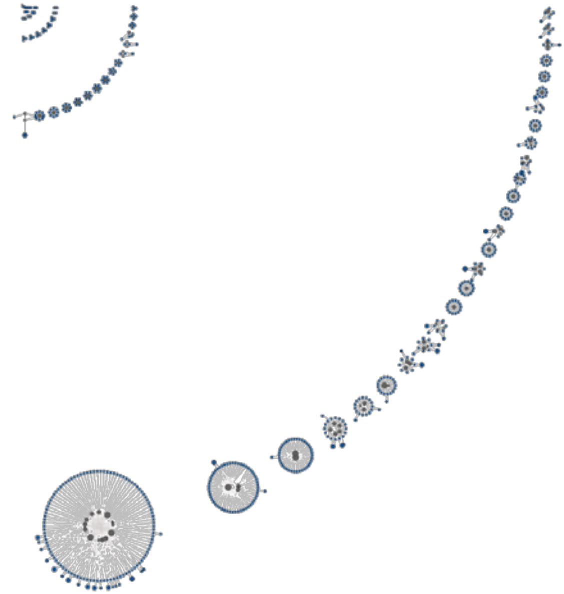


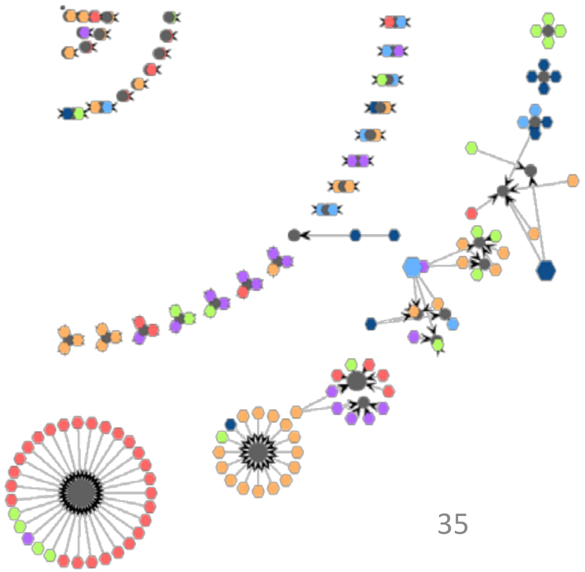


OpenCart

Concentric Arcs Layout

- To attain better space utilization
  - small clusters placed in the upper left corner
  - less whitespace to guard against cluster intersection
- Just like concentric circles:
  - we deploy the clusters on concentric arcs A_i of size π/2
  - we place 2^i clusters on the i^{th} arc
  - to avoid cluster overlaps, we use exactly the same radius optimization technique we used before.
- Unlike the concentric circles,
  - the partition assigned to each cluster is proportionate to its size (as in the case of the single circle), again taking care to avoid overlaps

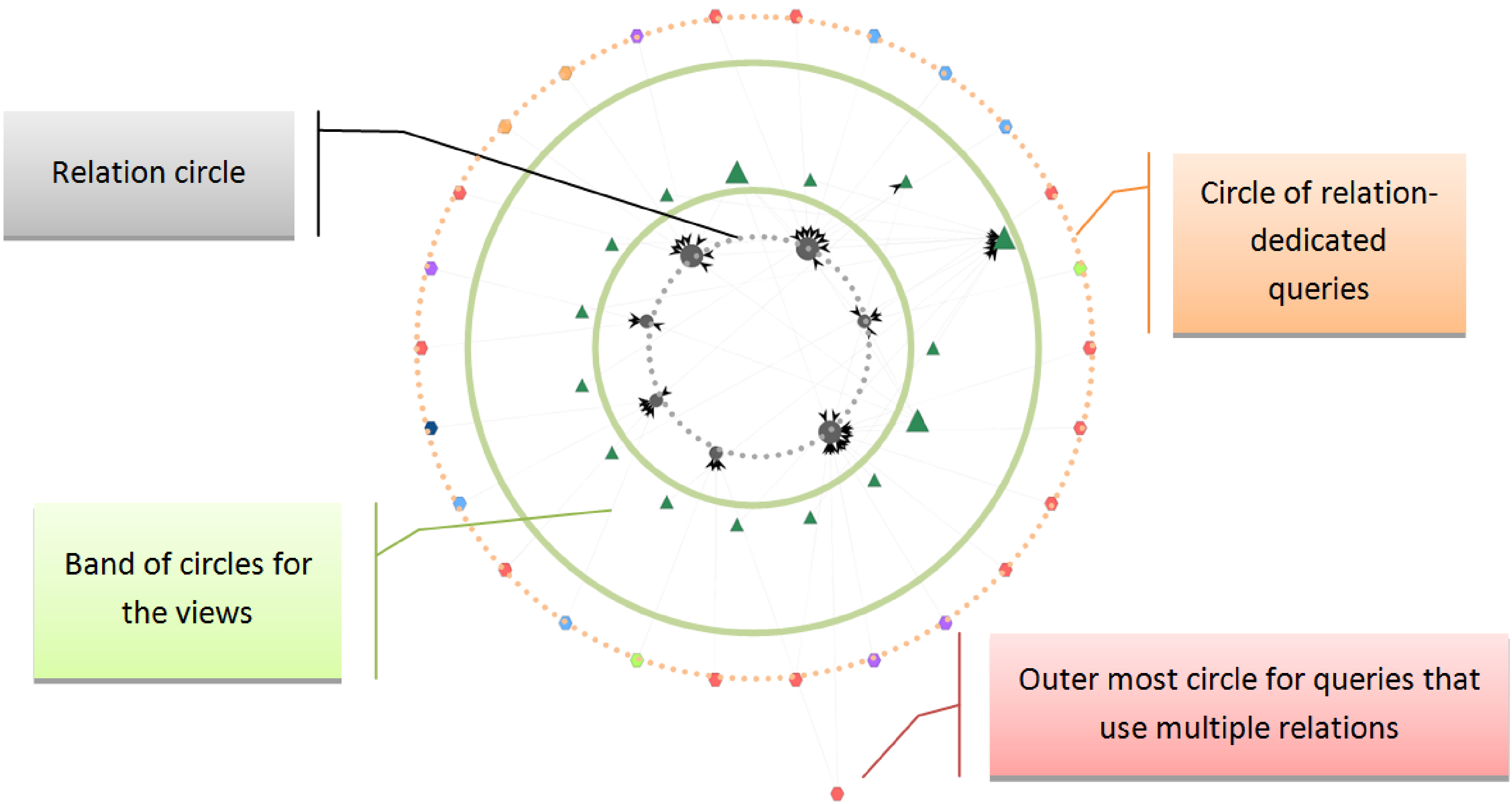




Roadmap

- ✓ 1. Cluster similar nodes in groups (clusters)
- ✓ 2. Estimate the space required for each cluster
- ✓ 3. Three alternative methods to place clusters on a 2D canvas
  - – Single circle
  - – Concentric Circles
  - – Concentric Arcs
- **4. Place the nodes of each cluster in concentric circles, internally in the cluster**
- 5. Summing up

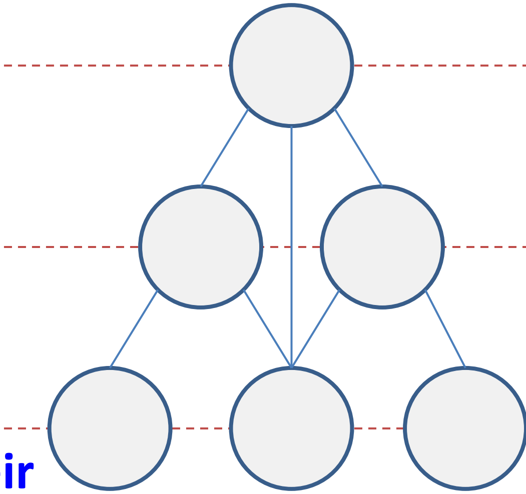
Step 4: arrangement of nodes within the circular clusters



Remember: 4 bands of circles to place nodes

Step 4: arrangement of nodes within the circular clusters

- We want to follow a barycenter based method, which can work successfully for layered bipartite graphs
- The standard barycenter method works with linear layers with the principle that once you have laid out layer i, you can lay out layer i+1 wrt the previous one
  - … **practically placing nodes in the barycenter of their neighbors in the previous layer i**



- Here, we have two challenges:
  - adapt this to our radial, concentric circles
  - decide the initial order of the process (here: relations in the inner circle)

```
order R;
place R & Q_R;
place V & Q-Q_R.
```

Step 4: arrangement of nodes within the circular clusters

1. Order the relations
    1. Count the frequency of each combination of tables as hit by the queries
    2. Place tables in popular combinations sequentially
2. Decide the position of relations and relation-dedicated queries
    1. Locate relation dedicated queries, decide the arc they need and position them sequentially
    2. Place relation in the middle of this arc
3. Decide the position of the rest of the queries and the views
    1. Stratify views and queries – each stratum has a dedicated circle
    2. Place views and queries via a barycenter method on their angle
    3. Adjust overlapping nodes (e.g., queries hitting exactly the same tables)

Roadmap

- ✓ 1. Cluster similar nodes in groups (clusters)
- ✓ 2. Estimate the space required for each cluster
- ✓ 3. Three alternative methods to place clusters on a 2D canvas
  - Single circle
  - Concentric Circles
  - Concentric Arcs
- ✓ 4. Place the nodes of each cluster in concentric circles, internally in the cluster
- **5. Summing up**

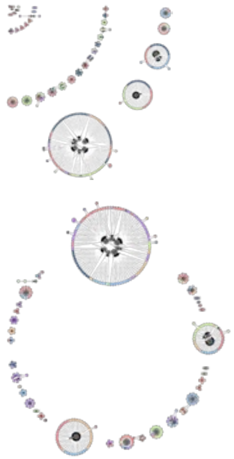
Not covered in this talk / paper…

- … failures & other tries ….
- Algorithmic details and geometrical issues
  - … esp., concerning the intra-cluster placement
- Relationship to aesthetic principles
- Experiments

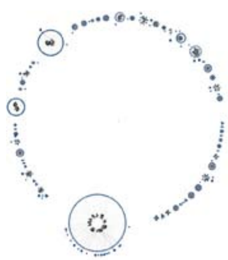
To probe further (code, data, details, presentations, …)

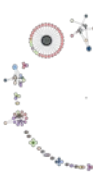
http://www.cs.uoi.gr/~pmanousi/publications/2014_ER/

We can **tame code-db interdependence** via **rigorous modeling** and **visual methods**



- **Visual methods to chart ecosystems** explored on the grounds of:
  - … radial deployment
  - … grouping, coloring, placement
  - **… visual clutter reduction**
  - all aiming to better **highlight code-db relationships**





To probe further (code, data, details, presentations, …)

http://www.cs.uoi.gr/~pmanousi/publications/2014_ER/

AUXILIARY SLIDES

Why bother?

- The problem is …
  - **Important**, as its implications relate to productivity and development effort
  - **Hard to solve**, not solved by SotA, as standard graph drawing methods do not seem to work well
  - **Interesting**, as it requires a large amount of technical solutions to visualization problems
- … and, of course, we have not only solved it, but also, we have incorporated the solution to an actual system…

In a nutshell

Fundamental modeling pillar: *Architecture Graph G(V,E) of the data-intensive ecosystem. The Architecture Graph* is a skeleton, in the form of graph, that traces the dependencies of the application code from the underlying database.

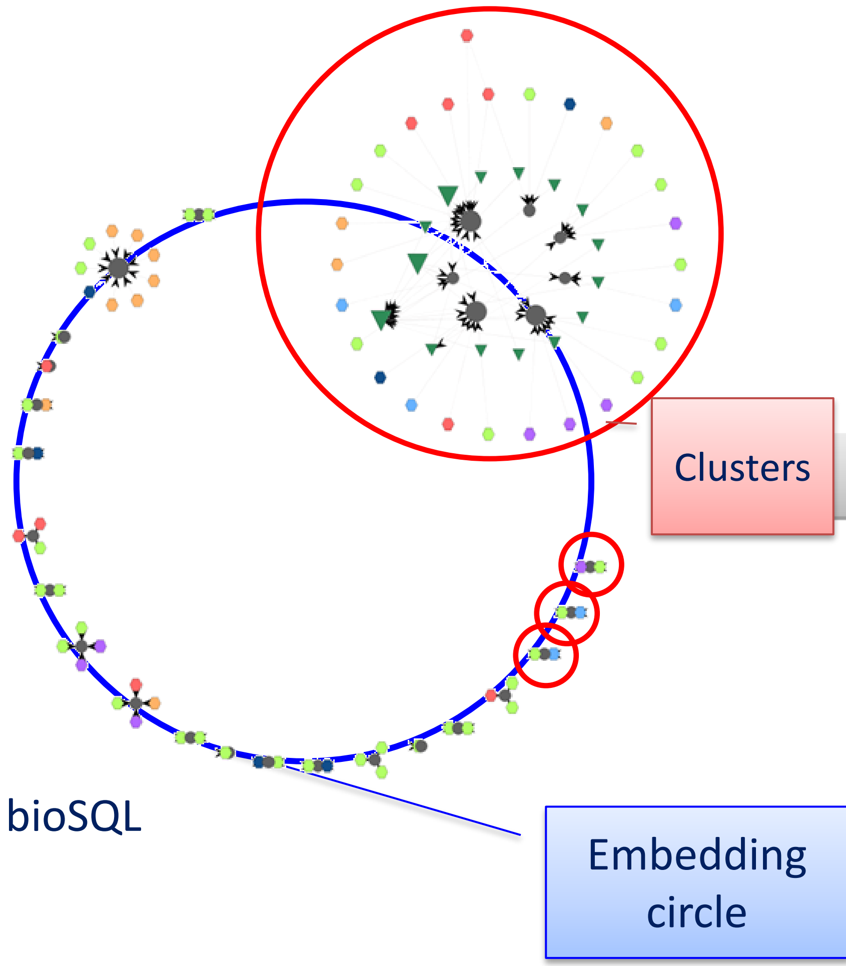
- *modules (relations, views* and queries) as nodes and
- edges denoting data provision relationships between them.

Visualization choices:

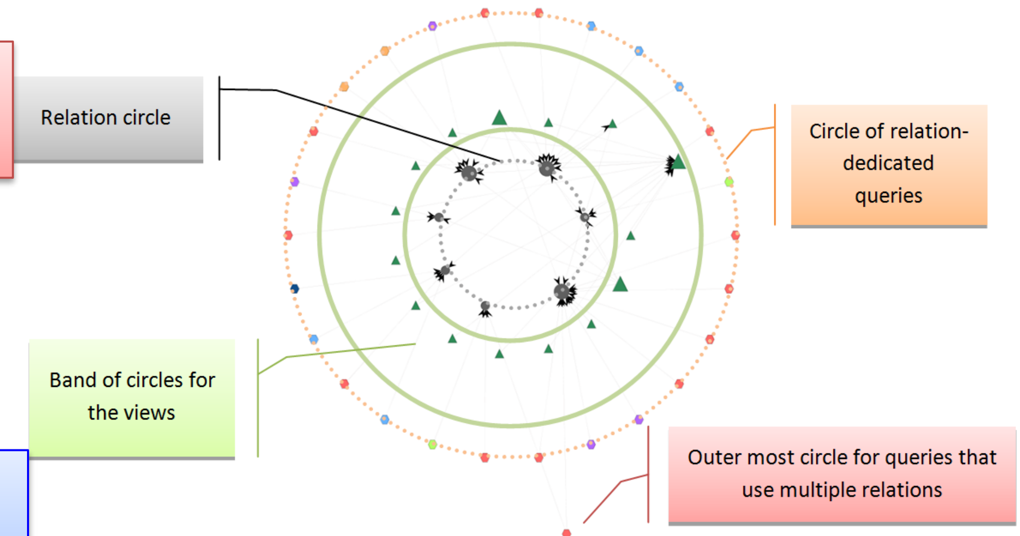
- *Circular layout.* Circular layouts give*:*
  - better highlight of node similarity*,*
  - less line intersections, i.e., less clutter
- *Clustered graph drawing.* We place clusters of objects in the periphery of an embedding circle or in the periphery of several concentric circles or arcs. Each cluster will again be displayed in terms of a set of concentric circles, thus producing a simple, familiar and repetitive pattern.

Graphical Notation

- One (or more) **embedding circle** for cluster placement
- **Clusters** are **internally** arranged over **concentric circles**, too
- Node colors: to which script queries belong
- Node shape: relation / view / query



The internal structure of a cluster



4 bands of circles, within a cluster:
- 1 circle for relations
- As many as needed for views
- 2 circles for queries

Aesthetics and design choices

- Node shape: different shapes to visually distinguish the different type of nodes. Relation nodes have circular shape, view nodes have triangular shape and query nodes are depicted as hexagons.
- **Node size: scaled according to their node degree**
  - the most used modules are more conspicuous.
- Node color: we distinguish node types with different colors.
  - **Relations are grey** and **views are dark green**. (db's are dark)
  - **Query nodes have different colors**, depending on the folder their embedding script in the applications belongs.
  - Thus, the difference in color provides another way of grouping queries.

Visual clutter introduced by edges

- **Edges are the main source of visual clutter!**
- So, we reduced the intensity of the edges' presence of the visual map:
  - we picked a light gray color for the edges and
  - we made them very thin, in terms of weight (almost invisible).
- To retain their info: every time a particular node is selected by the user its neighboring nodes are highlighted with a blue transparent color so, instead of emphasizing edges, we emphasize neighbors.

Steps of the method

Our method for visualizing the ecosystem is based on the principle of clustered graph drawing and uses the following steps*:*

1. Cluster the queries, views and relations of the ecosystem, into clusters of related modules. Formally, this means that we partition the set of graph nodes *V into a set of disjoint subsets, i.e., its clusters,* $C_1, C_2, \ldots, C_n$.
2. Estimate the necessary area for each cluster.
3. Position the clusters on a two-dimensional canvas in a way that minimizes visual clutter and highlights relationships and differences.
4. For each cluster, decide the positions of its nodes and visualize it.

Related Work

Gestalt principles

See for example *C. Ware. "Information Visualization: perception for design", Morgan Kaufmann, 2nd edn., 2004*

- *Proximity* - objects close to each other tend to be perceived as similar.
- *Similarity* - objects of the same shape, color, orientation and size are perceived as similar by individuals.
- *Connectedness* - to express semantic relationship among visually connected objects.
- *Closure* - the eye tends to create perceptions of closed space, even if they do not exist -- best served when the depicted objects tend to create a "border" around similar objects along with blobs of whitespace.
- *Continuity* - the eye tends to perceive as related objects that are aligned together intersections create the perception of single uninterrupted groups.
- *Symmetry* - as a means to emphasize non-typical behavior or emphasis when symmetry is broken by an object. In principle asymmetry is used for emphasis while symmetry is used in cases where we do not want to target on something specific.
- *Contrast* - creates emphasis in sharp antithesis to the similarity principle. Contrast can be achieved in terms of chromatic, size or shape choices.
- *Proportion* - where an object placed in an area of the visualization is scaled according to its semantic significance, as the difference in proportion creates a visual attraction to the eye

Best practices

- *Clutter avoidance* - the avoidance of noise on the diagram via uninterrupted areas of whitespace that act as separators of the groups of objects
- *Isolation* - to promote emphasis for an object in sharp antithesis to the continuity of the vast majority of the "regular" objects
- *Visual hierarchy* - to denote a semantic hierarchy in the depicted objects
- *Focal points* to guide visual flow (i.e., objects that intentionally stand out in the representation and whose sequence guides the eye in the visual flow of exploring the diagram).

Jenifer Tidwell. "Designing interfaces - patterns for effective interaction design", O'Reilly, 2006

The eyes have it

"Visual Information Seeking Mantra": Overview first, zoom and filter, then details-on-demand

- **Overview**: Gain an overview of the entire collection. Overview strategies include zoomed out views of each data type to see the entire collection plus an adjoining detail view.
- **Zoom** : Zoom in on items of interest. Users typically have an interest in some portion of a collection, and they need tools to enable them to control the zoom focus and the zoom factor. Smooth zooming helps users preserve their sense of position and context. Zooming could be on one dimension at a time by moving the zoom bar controls or in two dimensions. A very satisfying way to zoom in is by pointing to a location and issuing a zooming command, usually by clicking on a mouse button for as long as the user wishes or clicking on a node or edge to view further details.
- **Filter**: filter out uninteresting items.
- **Details-on-demand:** Select an item or group and get details when needed. Once a collection has been trimmed to a few dozen items it should be easy to browse the details about the group or individual items. The usual approach is to simply click on an item to get a pop-up window with values of each of the attributes, also helpful to keep a history of user actions and support other actions the user may need like undo or replay.

Ben Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations", Proc. of the 1996 IEEE Symposium on Visual Languages, pp 336-343, 1996

Code visualization

- Brian Johnson and Ben Shneiderman. **Tree-Maps**: a space-filling approach to the visualization of hierarchical information structures. In Proceedings of the 2nd conference on Visualization '91 (VIS '91), pp. 284-291. 1991. IEEE Computer Society Press, Los Alamitos, CA, USA.
- S.G. Eick, J. L. Steffen, E. E. Sumner Jr. **Seesoft**: a tool for visualizing line-oriented software statistics. IEEE Transactions on Software Engineering 18(11): pp. 957-968, 1992.

- Andrew Bragdon, Robert C. Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra,Joseph J. LaViola Jr. "**Code bubbles**: a working set-based interface for code understanding and maintenance", Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI 2010), pp 2503-2512, 2010
- Robert DeLine, Kael Rowan. "**Code canvas**: zooming towards better development environments", Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010), pp 207-210, 2010

- **Pierre Caserta and Olivier Zendra. Visualization of the Static Aspects of Software: A Survey. IEEE Transactions on Visualization and Computer Graphics (TVCG), 17(7), July 2011**

(Radial) graph drawing

- Ivan Herman, Guy Melancon, and M. Scott Marshall. “Graph Visualization and Navigation in Information Visualization: A Survey”, IEEE Transactions on Visualization and Computer Graphics 6, pp 124-43. 2000
- Takao Ito, Kazuo Misue, Jiro Tanaka. “Drawing Clustered Bipartite Graphs in Multi-circular Style”, 14th International Conference on Information Visualisation (IV 2010), pp 23-28, 2010
- Kazuo Misue. “Drawing bipartite graphs as anchored maps”, Asia-Pacific Symposium on Information Visualisation (APVIS) pp 169-177, 2006

Method Internals

Bird’s eye view of the Method

1. **Cluster similar nodes in groups (clusters)**
   - A cluster is a set of relations, views and queries
   - Similarity is determined by the edges
2. **Estimate the space required for each cluster**
   - … to avoid cluster overlaps later
3. 3 alternative methods to **place clusters on a 2D canvas**
   - Single circle
   - Concentric Circles
   - Concentric Arcs
4. **Place the nodes of each cluster in concentric circles, internally in the cluster**

Step 1: Clustering

- To reduce the amount of visible elements, visualization methods place them in groups, thus
  - reducing visual clutter
  - improving user understanding of the graph
- Principle of proximity: similar nodes are placed next to each other
- Here: we use clustering to group objects with similar semantics in advance of graph drawing.

Step 1: Clustering

- Average-link agglomerative clustering algorithm
- First, we compute the distances for every pair of nodes in the graph.
- Then, we iteratively perform cluster merging:
  - find the minimum distance pair of clusters,
  - merge the components of the pair into a new cluster, and,
  - calculate the new distances.

- This process starts with each node being a cluster on its own and stops when the minimum distance of all pairs of clusters is greater than a user-defined threshold of cluster distance.

Algorithm 1. Clustering

Input: *G* : all the graph objects (relations, queries, views), list with solutions (initially every object as a cluster) *T*: the user defined threshold for the distance of two clusters (below which the user deems that the merge of the clusters is without meaning)

Variables: *mindist*: the min distance between clusters

Output: *C:* a set of clusters

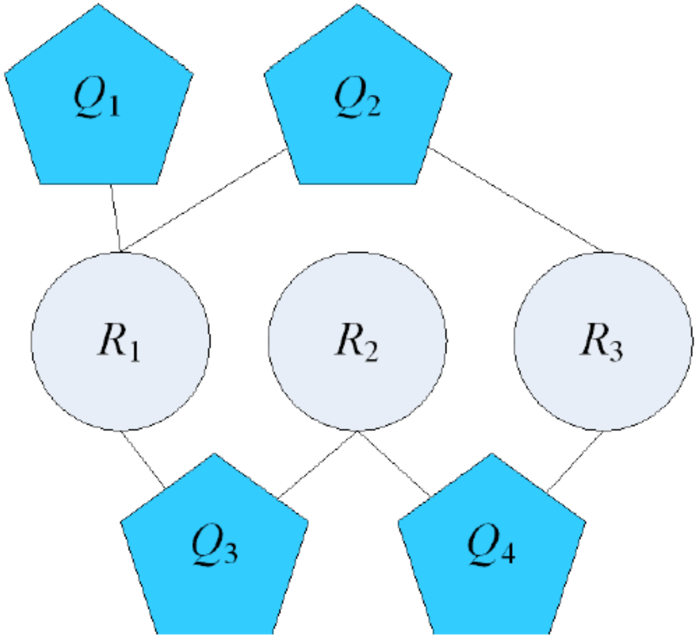
Begin

```
Create a set C = { {t_1}, {t_2}, …, {t_n} } with all the objects of G as clusters
Do
    mindist = ∞
    For each pair c_i , c_j, i ≠ j
        Compute pairwise distances between them
        If a pair has smaller distance than mindist
            Update mindist with smaller distance
            Update mindist pair
        End if
    End for
    Merge mindist pair
    Add pair to C
    Remove mindist objects from C
    If mindist >= T return C
While number of clusters != 1
Return C
```

End

Step 1: Clustering

$$dist(M_i, M_j) = 1 - \begin{cases} \frac{|neighbors_i \cap neighbors_j|}{|neighbors_i \cup neighbors_j|}, & if \ \nexists \ Edge(i,j) \\ \frac{|neighbors_i \cap neighbors_j| + 2}{|neighbors_i \cup neighbors_j|}, & if \ \exists \ Edge(i,j) \end{cases}$$



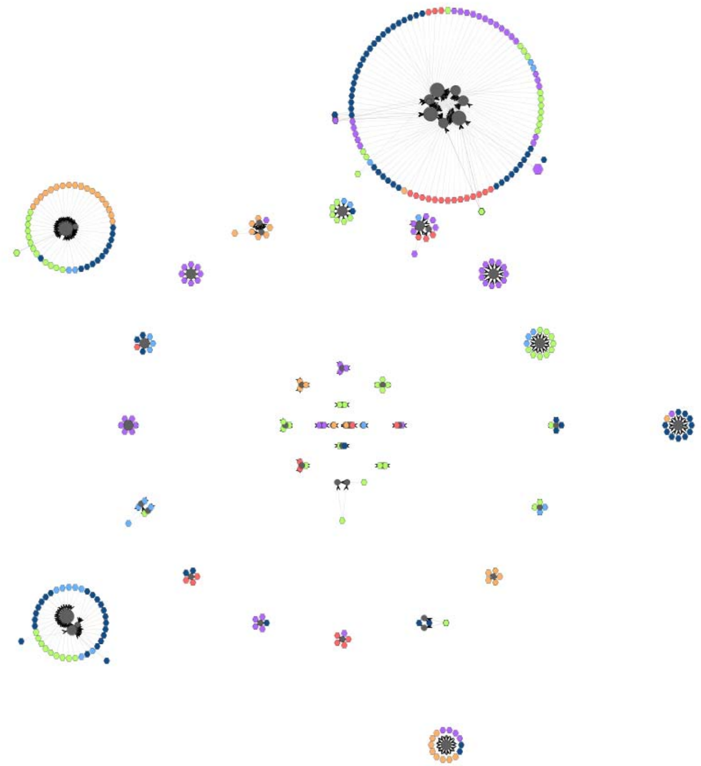
$$s(Q_1, Q_3) = \frac{|\{R_1\} \cap \{R_1, R_2\}|}{|\{R_1\} \cup \{R_1, R_2\}|} = \frac{1}{2}$$

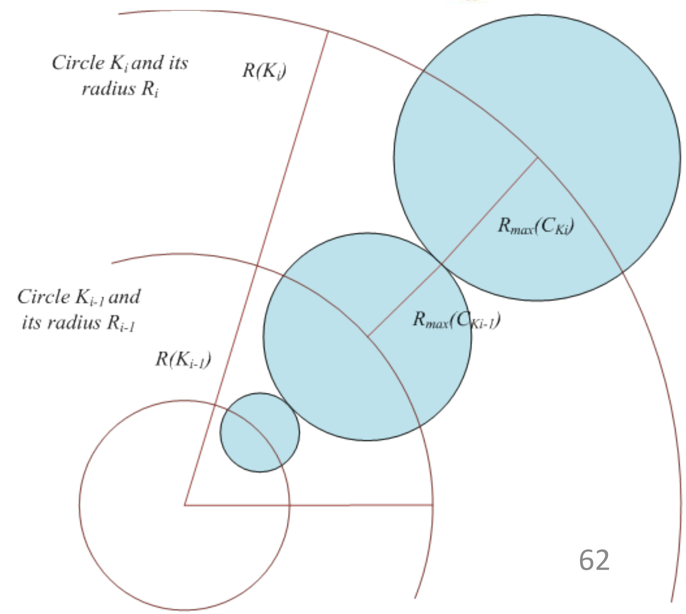
$$s(R_1, Q_3) = \frac{|\{Q_1, Q_2\} \cap \{R_1, R_2\}| + 2}{|\{Q_1, Q_2, Q_3\} \cup \{R_1, R_2\}|} = \frac{2}{5}$$

$$s(R_1, Q_4) = \frac{|\{Q_1, Q_2\} \cap \{R_2, R_3\}|}{|\{Q_1, Q_2\} \cup \{R_2, R_3\}|} = 0$$

Step 2: Estimate the area of each cluster

- Once the clusters have been computed, the next step is to <u>estimate the space required for each cluster</u>
- This step is crucial and necessary for the subsequent step of cluster placement, in order to be able to
  - **calculate the total area of the overall graph** and
  - **arrange the clusters without overlaps**

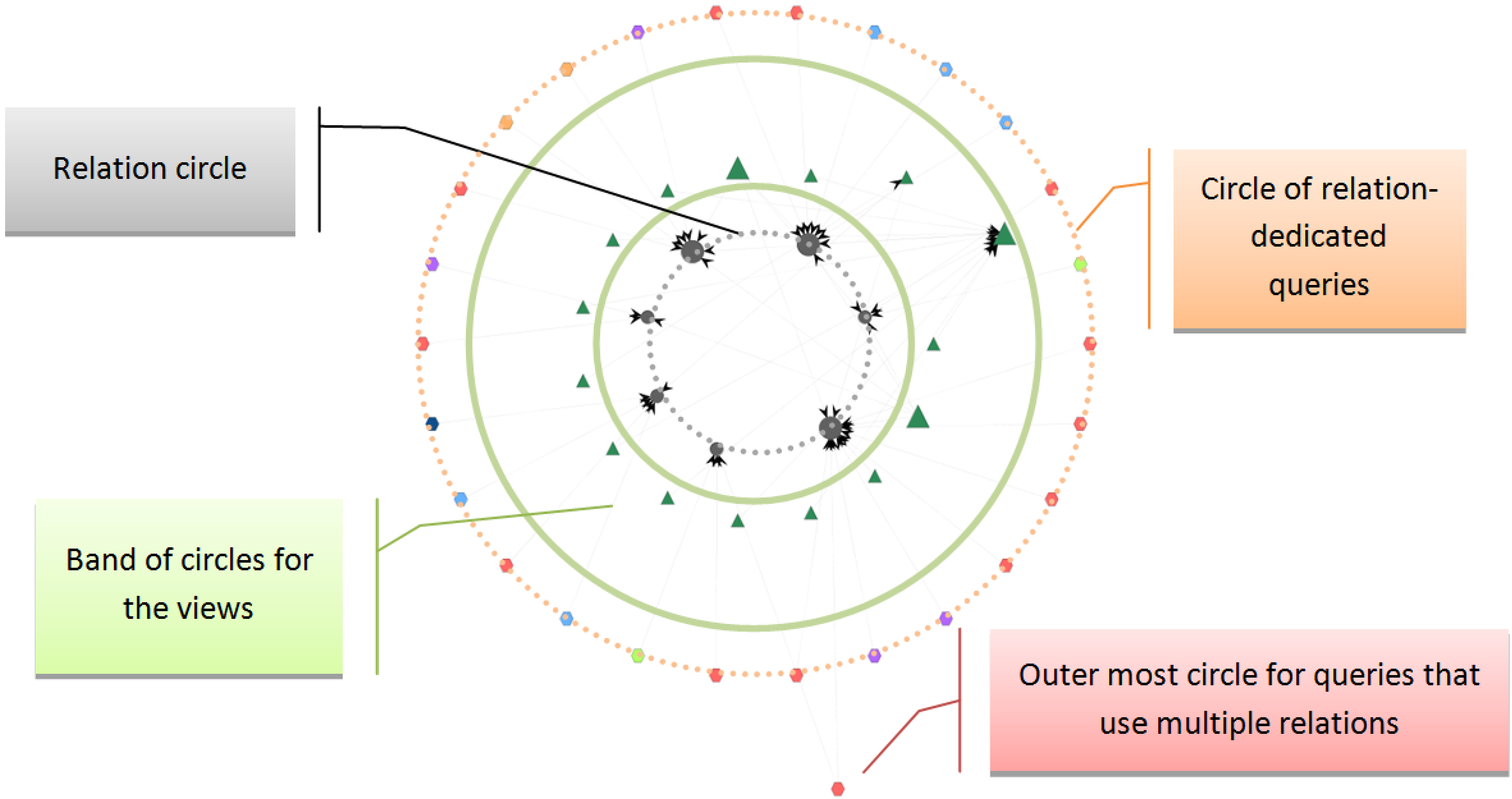




Step 2: Estimate the area of each cluster

- Each cluster includes **3 bands of concentric circles**:
  - the innermost (single) circle for the relations,
  - an intermediate band of circles for the views, and
  - the outermost band of circles for the queries, organized as
    - a circle of relation-dedicated queries (i.e., queries that hit a single relation) and
    - an outer circle for the rest of the queries.

Example: a cluster from BioSQL



Step 2: Estimate the area of each cluster

- We need to:
  - determine the circles of the drawing and the nodes that they contain, and
  - compute the radius for each of these circles.
  - Then, the outer of these circles gives us the area of this cluster

Step 2: Estimate the area of each cluster

- To obtain the bands we **topologically sort** the nodes of the cluster and organize them in **strata**.
  - Relations: the 0-th stratum (no dependencies whatsoever)
  - Views: each stratum V_i defines an equivalence class in the graph with all the nodes of the graph that depend only from nodes in strata V_j previous to V_i, $j < i$.
  - Queries: we heuristically split them in two pseudo-strata: (a) relation-dedicated queries and (b) all the rest of the queries.
- For each stratum, we add a circle with radius

$$R_i = 3 * \log(nodes) + nodes$$

Step 2: Estimate the area of each cluster

Algorithm 2. Circle Identification

Input: a cluster C

Variables: T: the tables of C, V: the views of C, Q: the queries of C, S: a list of strata (to be topologically sorted) over $T \cup V \cup Q$, *nodes:* $T \cup V \cup Q$

Output: a list of circles $K=\{K_0, \ldots, K_n\}$, each annotated with its nodes, $nodes(K_i)$, and its radius R_i

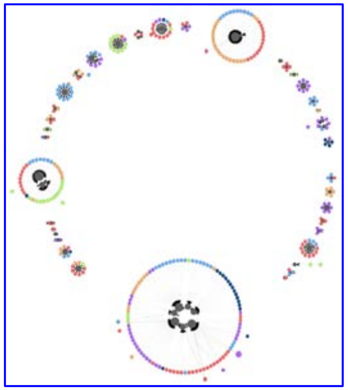
Begin

1. Topologically sort T, V and Q and organize their nodes in strata; then, S is a list of strata, $S = T \cup V \cup Q$, with $T = \{S_0\}$, $V = \{S_1, \ldots, S_m\}$, $Q = \{S_{m+1}, S_{m+2}\}$
2. For every stratum S_i of $S = \{T \cup V \cup Q\}$
3. Append a new circle K_i to K, $nodes(K_i) = V_i$
4. Compute its radius $R_i = \log(nodes^3) + nodes$

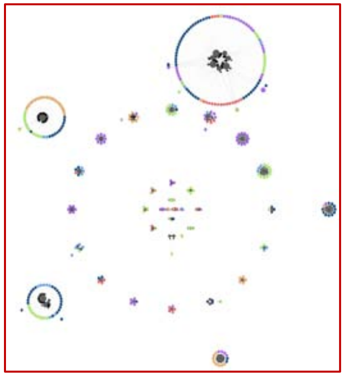
End

Step 3: Laying out the Clusters

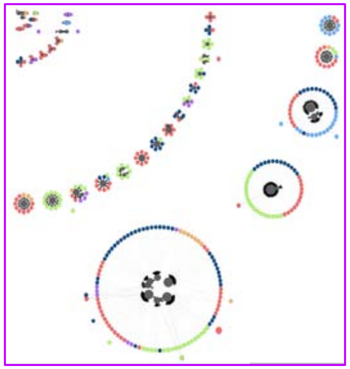
3 alternative methods for placing the clusters on a 2D area



- Circular placement
  - all clusters on a single circle



- Concentric circles
  - trying to reduce the intermediate empty space



- Concentric arcs
  - a combination of the previous two methods

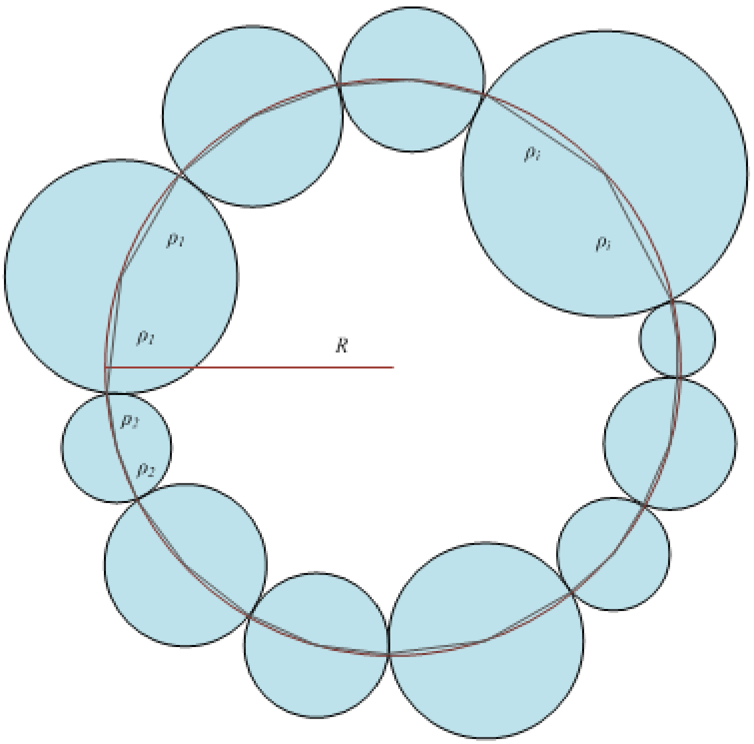
Circular cluster layout

- We use a single embedding circle to place the clusters.
- One sector of the circle per cluster
  - with its angle varying on the cluster's size (#nodes)
  - remember: each cluster is also a circle, approximated by its outermost constituent circle of nodes
- Steps:
  1. Compute *R*, the radius of the embedding circle
  2. Compute φ_i, the angle of each cluster's sector
  3. Add some extra whitespace
  4. Compute the coordinates of all clusters

compute R;
compute φ_i;
whitespace;
coordinates.

Circular Layout: Embedding Circle determination

- Given: the radius r_i of each cluster i
  Compute: R, the radius of the embedding circle.
- **Method**:
- approximate the circles periphery ($2\pi R$) by the sum of edges of the embedded polygon
- divide this sum by 2π to calculate the radius R of the embedding circle



$$2\pi R \cong \sum_{i=0}^{|C|} 2\rho_i => R \cong \sum_{i=0}^{|C|} 2 * \rho_i / 2\pi$$

```
compute R;
compute φi;
whitespace;
coordinates.
```

Circular layout: calculation of the angle for each of the segments

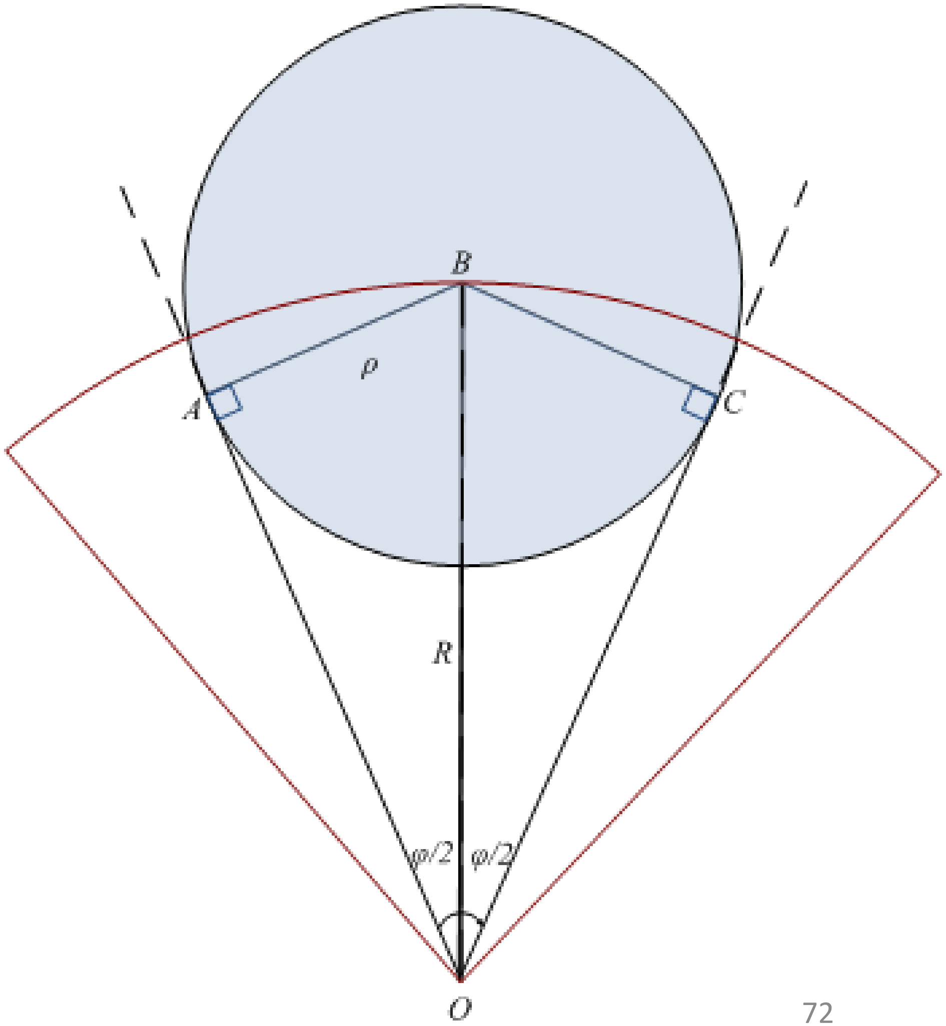
- Goal: assign each cluster to a segment of the circle depending on the cluster's radius (size).
- Each these segments is defined by an angle φ over the embedding circle.
- Not as obvious as it seems – we have to consider two cases:
  - The radius ρ of the cluster we want to place is smaller or equal to the radius of the embedding circle R
  - The radius ρ of the cluster we want to place is greater than the radius of the embedding circle R

```
compute R;
compute φi;
whitespace;
coordinates.
```

Circular layout: calculation of the angle for each of the segments

- Typical case, where $\rho \leq R$
- Consider the left triangle *ABO*
- Then:

$$\varphi/2 = \sin^{-1}\left(\frac{\rho}{R}\right)$$



In case you are wondering: is it possible that $\rho > R$??

- $R = (1/\pi) * \Sigma(\rho_i)$
- Assume $\rho_i \in \{250, 2, 3, 5\}$ => $R = 260 / 3.14 = 82.80$
- Therefore, there is a cluster with larger radius than the surrounding circle…

```
compute R;
compute φi;
whitespace;
coordinates.
```

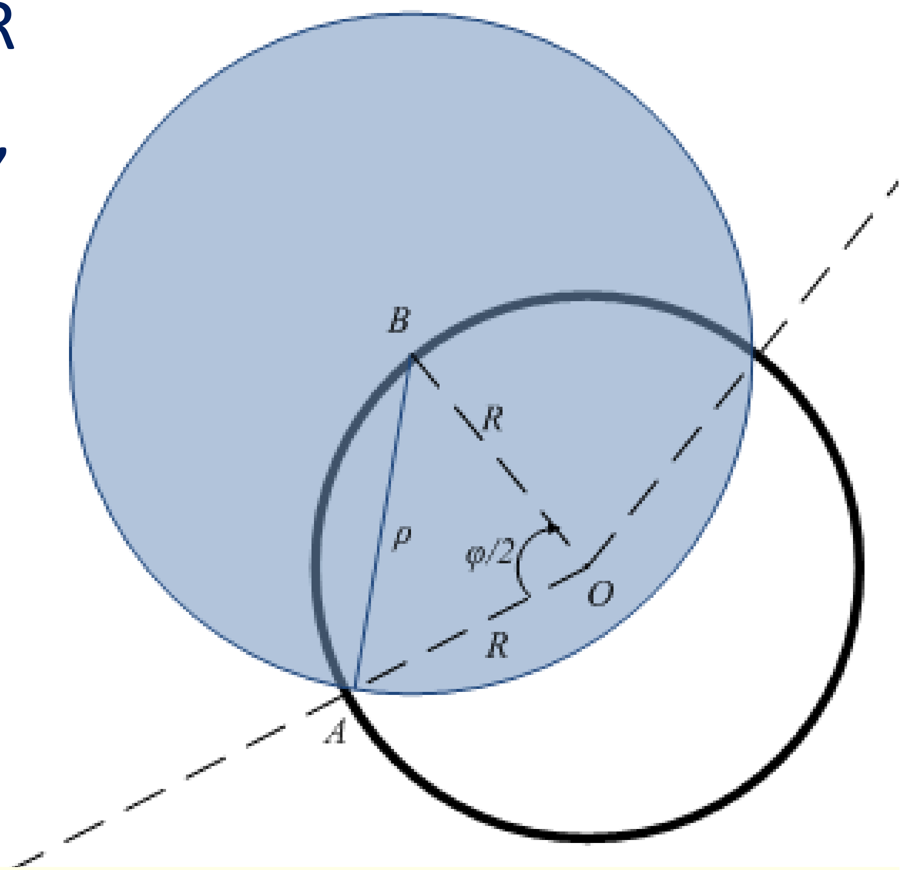
Circular layout: calculation of the angle for each of the segments

- A large cluster occurs $\rho > R$
- Assume the isosceles ABO, both AO,BO = R
- Then:

$$\varphi/2 = \cos^{-1}\left(\frac{2R^2 - \rho^2}{2R^2}\right)$$

- due to

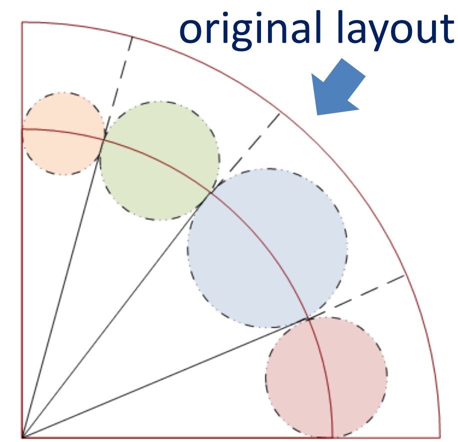
$$cos\varphi = (b^2 + c^2 - a^2)/2bc$$



Note: cannot avoid to discriminate the two cases

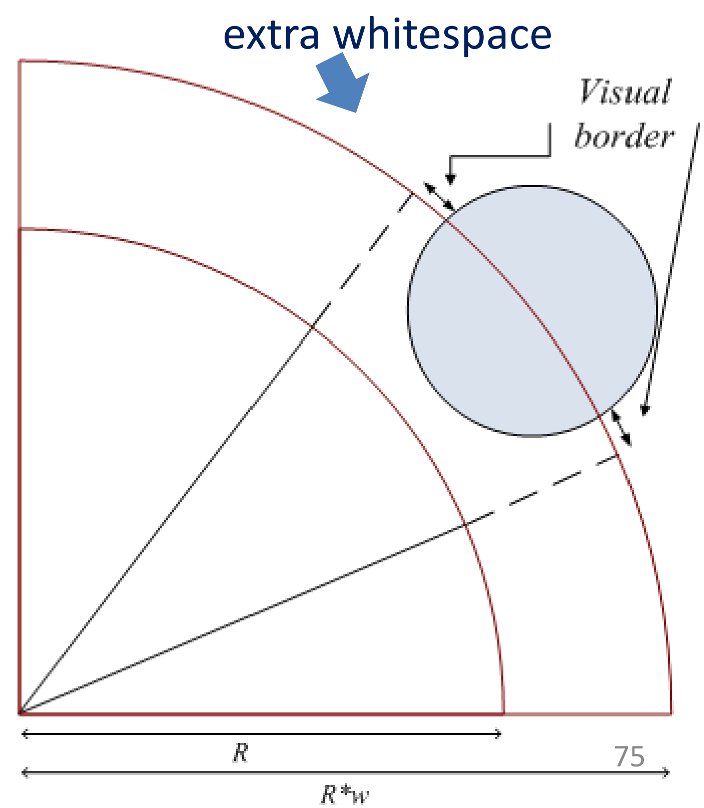
```
compute R;
compute φi;
whitespace;
coordinates.
```

Circular layout: avoid cluster overlap!

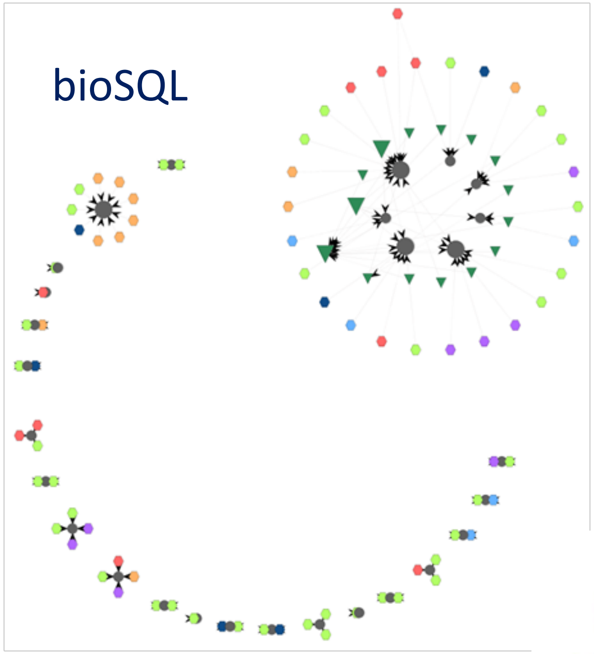


- We introduce a white space factor *w* that enlarges the radius *R* of the circle
- Each cluster is approx. by a circle, with
  - radius r (known from step #1)
  - center [c_x, c_y] determined by *φ*, *R*, and *w*.

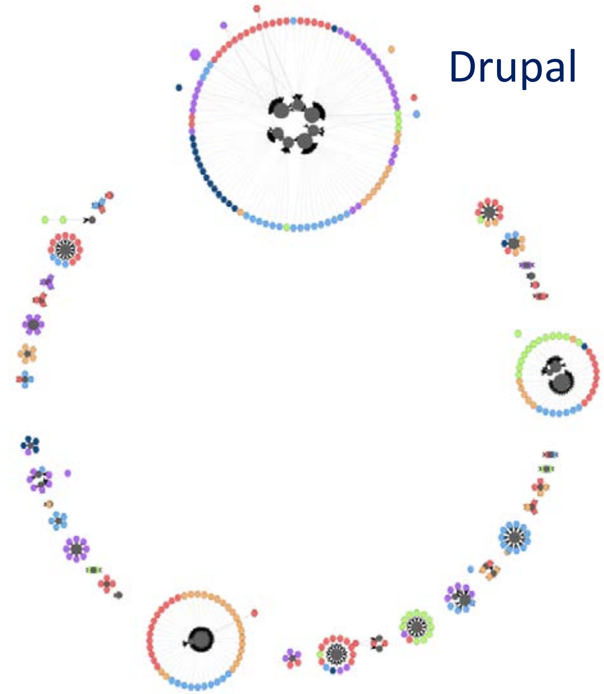
$$c_x = \cos\left(\frac{\phi}{2}\right) * R * w, \;\; c_y = \sin\left(\frac{\phi}{2}\right) * R * w$$



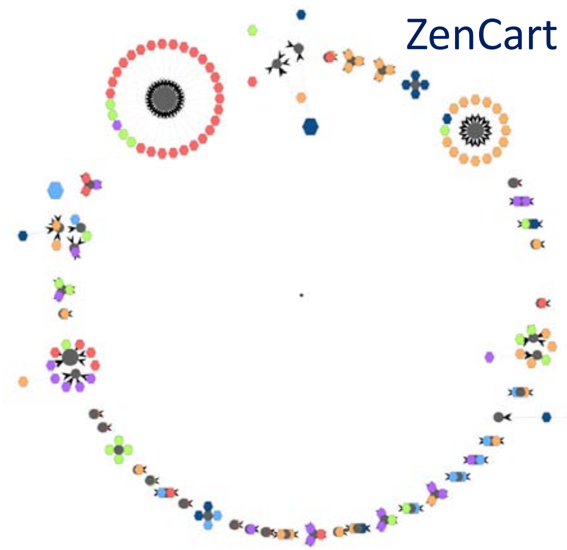
Circular Layout



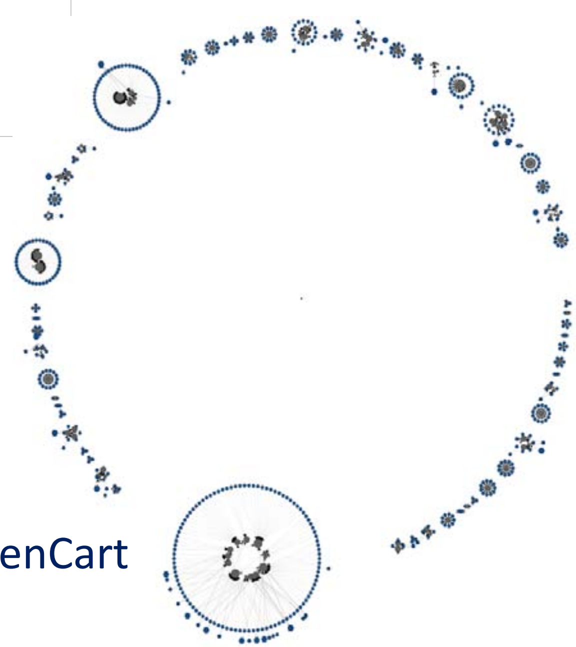
Drupal



ZenCart

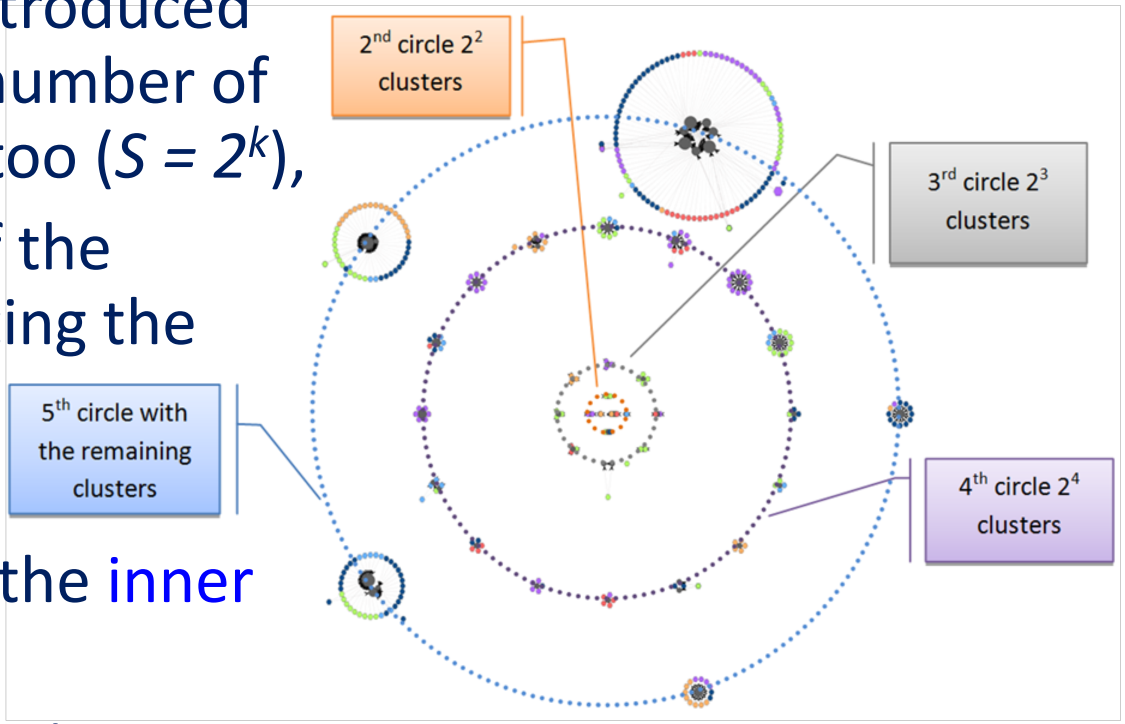


OpenCart



Concentric Circles Layout

- Each circle is split in fragments of powers of 2
  - as the order of the introduced circle increases, the number of fragments increases too ($S = 2^k$),
  - with the exception of the outermost circle hosting the remaining clusters
- This way, we can place
  - the small clusters on the inner circles, and
  - bigger clusters (occupying more space) on outer circles



Concentric Circles Layout

Method:

1. Sort clusters by ascending size in a list L^C
2. While there are clusters not placed in circles
   1. Add a new circle and divide it in as many segments as $S = 2^k$ with k being the order of the circle (i.e., the first circle has 2^1 segments, the second 2^2 and so on)
   2. Assign the next S fragments from the list L^C to the current circle and compute its radius according to this assignment
   3. Add the circle to a list L of circles
3. Draw the circles from the most inward (i.e., from the circle with the least segments) to the outermost by following the list L.

Concentric Circles Layout

Method:

1. Sort clusters by ascending size in a list L^C
2. While there are clusters not placed in circles
   1. Add a new circle and divide it in as many segments as $S = 2^k$ with k being the order of the circle (i.e., the first circle has 2^1 segments, the second 2^2 and so on)
   2. **Assign the next S fragments from the list L^C to the current circle and compute its radius according to this assignment**
   3. Add the circle to a list L of circles
3. Draw the circles from the most inward (i.e., from the circle with the least segments) to the outermost by following the list L.

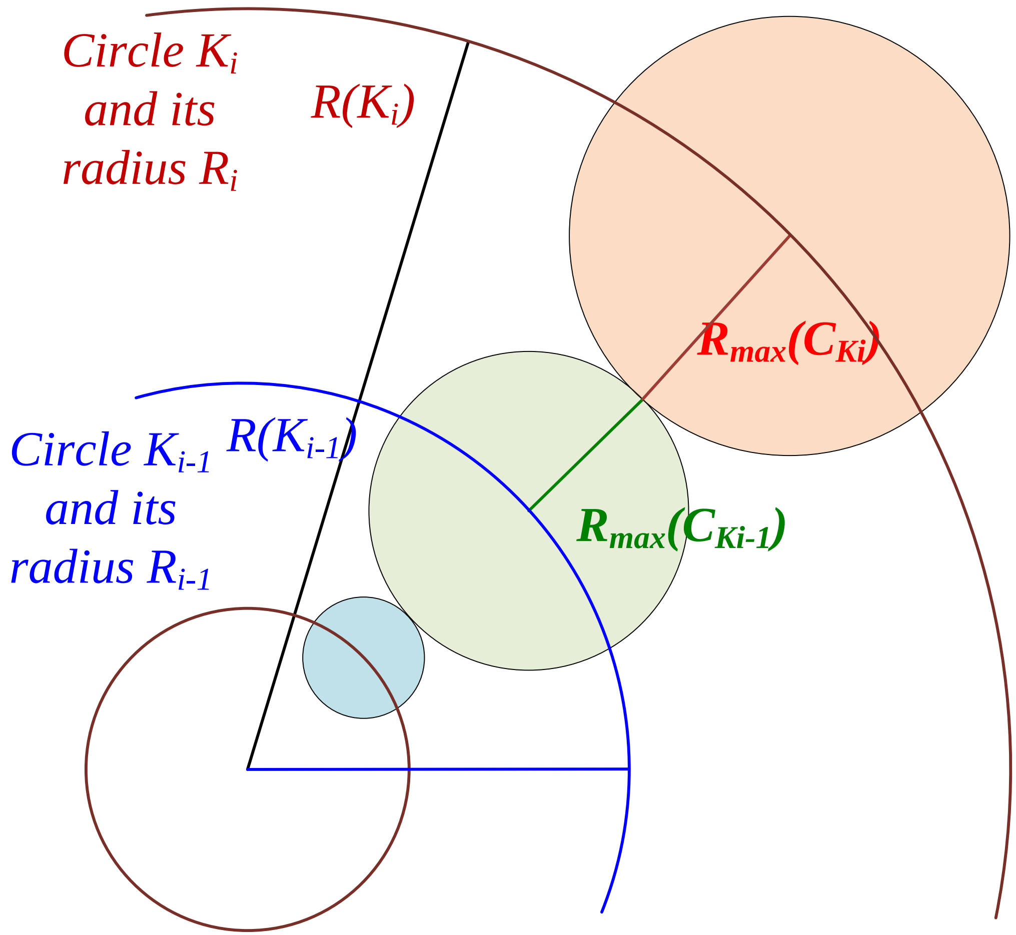
Main challenge

Concentric Circles: radius calculation

- Instead of having to deal with just one circle, we need to compute the radius for each of the concentric circles, in a way that clusters do not overlap
- Overlap can be the result of two problems:
  - clusters of subsequent circles have radii big enough, so that they meet, or,
  - clusters on the same circle are big enough to intersect.

Concentric Circles: radius calculation to avoid cluster overlap in subsequent circles

$$R(K_i) = R(K_{i-1}) + R_{max}(C_{Ki-1}) + R_{max}(C_{Ki})$$



We need to make sure that the radius of a circle K_i is larger than the sum of

(i) the radius of its previous circle K_{i-1},

(ii) the radius of its larger cluster $R_{max}(C_{ki-1})$ on the previous circle,

(iii) the radius of the larger cluster of the current circle $R_{max}(C_{ki})$.

Concentric Circles: radius calculation to avoid cluster overlap in the same circle

- To avoid the overlap of clusters on the same circle, we compute R_i via the encompassing circle's periphery ($2\pi R_i$) that can be approximated as the periphery of the inscribed polygon (sum of twice the radii of the circle's clusters)

Concentric Circles: radius calculation for each circle

- Finally, **to calculate the radius of a circle**:
  - we take the maximum of the two values of the two aforementioned solutions and
  - we use an additional whitespace factor *w* to enlarge it slightly (typically, we use a fixed value of 1.2 for *w*).

$$R(K_i) = w * max \begin{cases} R_{i-1} + R_{max}(C_{K_{i-1}}) + R_{max}(C_{K_i}) \\ \frac{1}{\pi}\sum_{j=1}^{|C|} R(C_{jK_i}), R(C_{jK_i}) : \text{radius of cluster } C_j \text{on circle } K_i \end{cases}$$

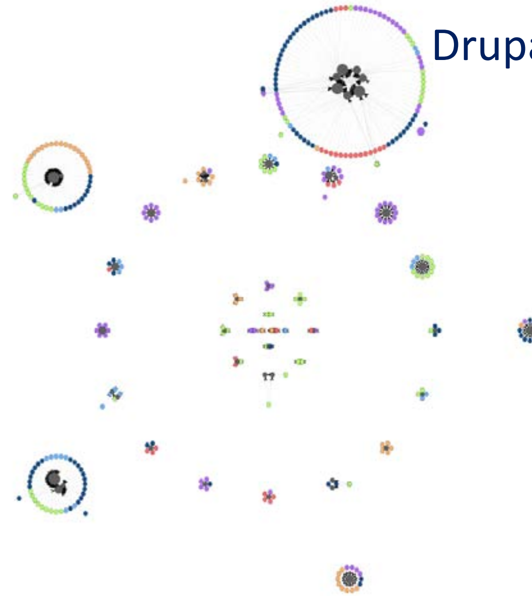
- **Clusters of the same circle** have **equal** segments with an **angle**:

$$\varphi_i = 2\pi/nK_i$$

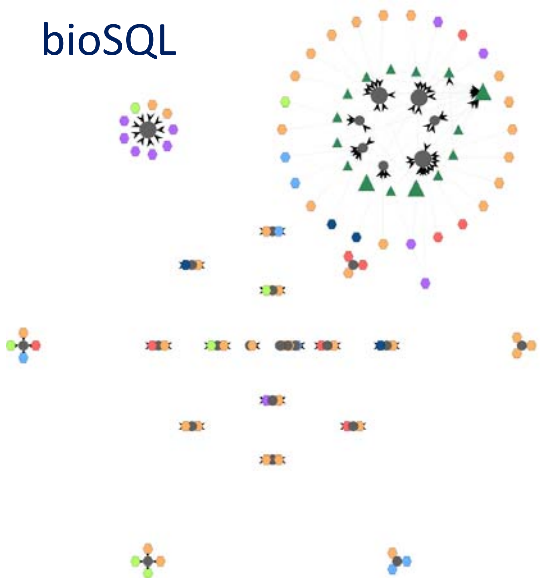
where *n*: the number of clusters on circle K_i

Concentric circles

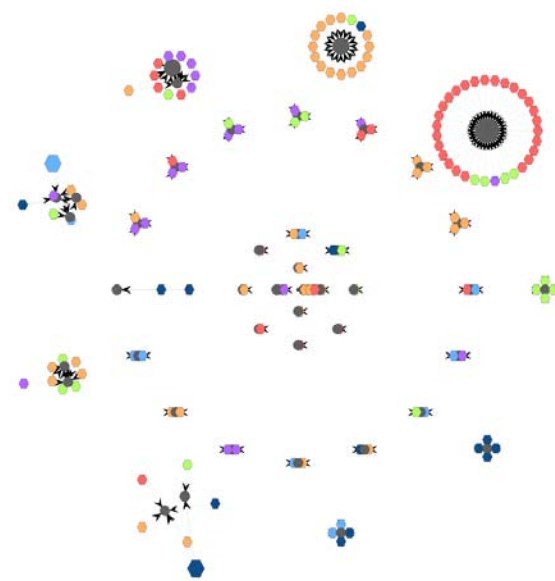
Drupal



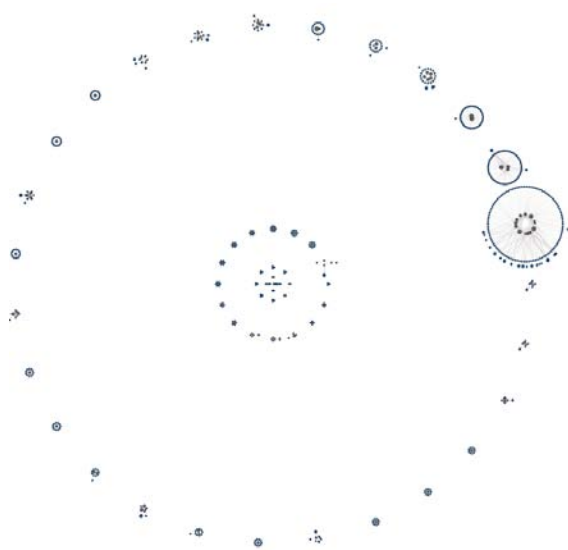
bioSQL



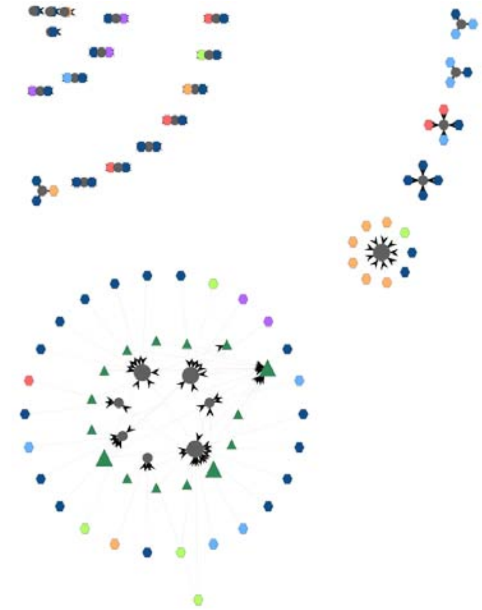
ZenCart



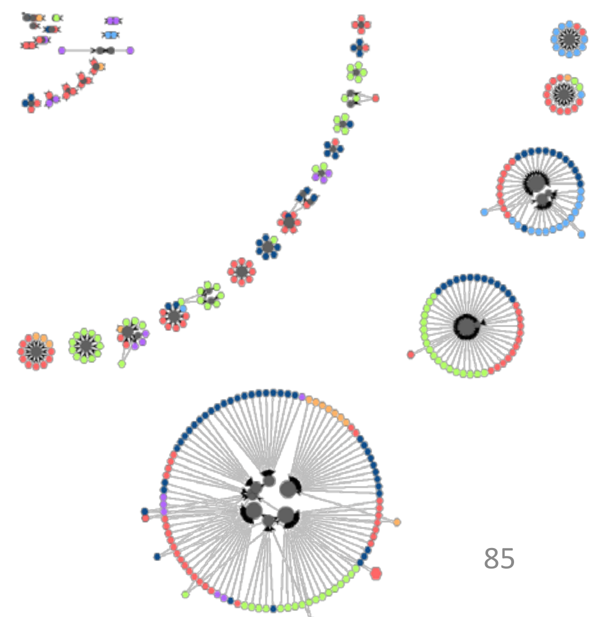
OpenCart



Concentric Arcs Layout

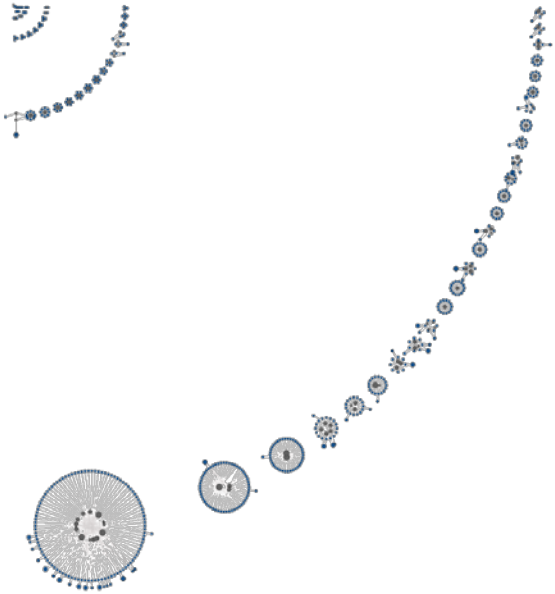


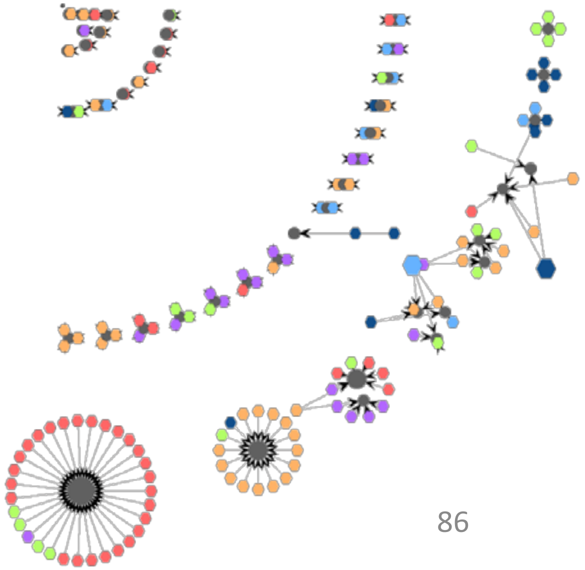
- To attain better space utilization, we can place the clusters in a set of concentric arcs, instead of concentric circles.
  - the small clusters are placed in the upper left corner
  - there is less whitespace devoted to guard against cluster intersection

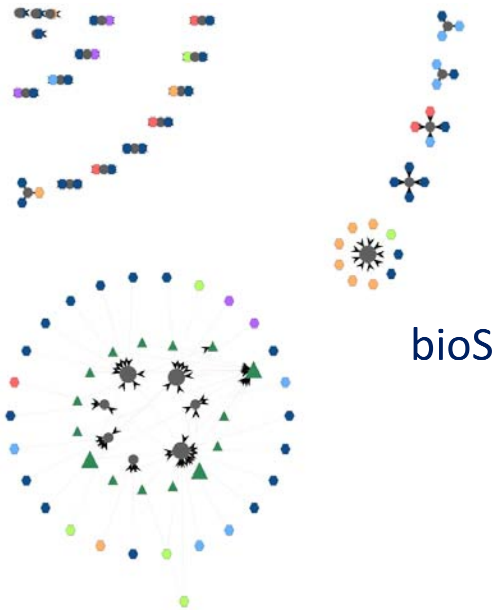


Concentric Arcs Layout

- Just like concentric circles:
  - we deploy the clusters on concentric arcs A_i of size $\pi/2$
  - we place 2^i clusters on the i^{th} arc
  - to avoid cluster overlaps we use exactly the same radius optimization technique we used before.
- Unlike the concentric circles,
  - the partition assigned to each cluster is proportionate to its size (as in the case of the single circle), again taking care to avoid overlaps

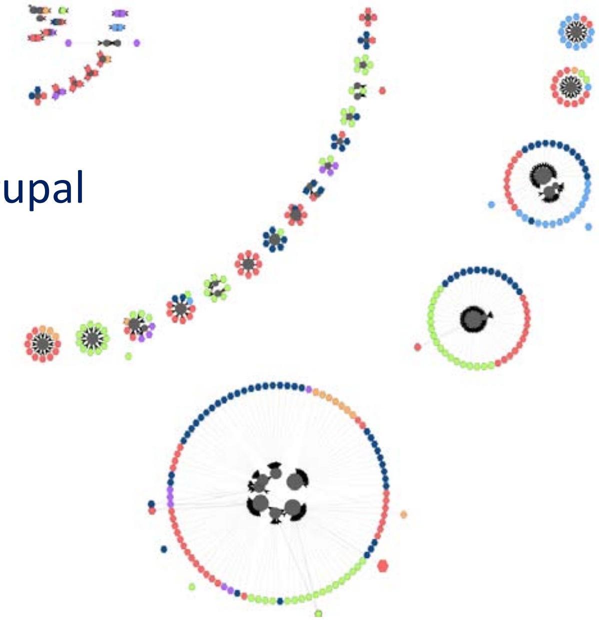




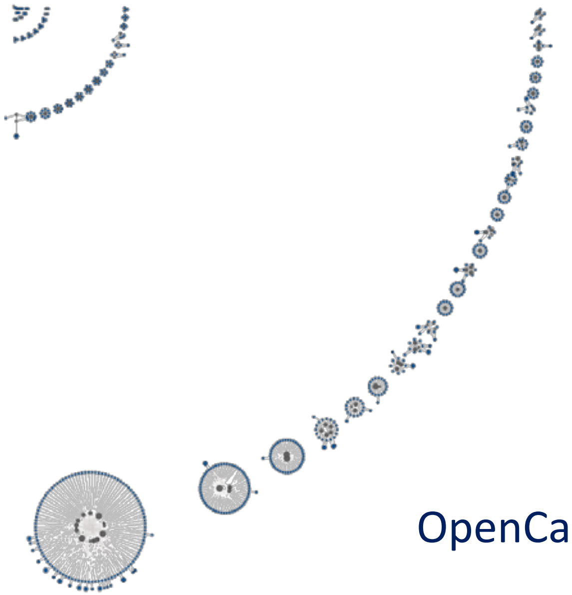


bioSQL

Drupal

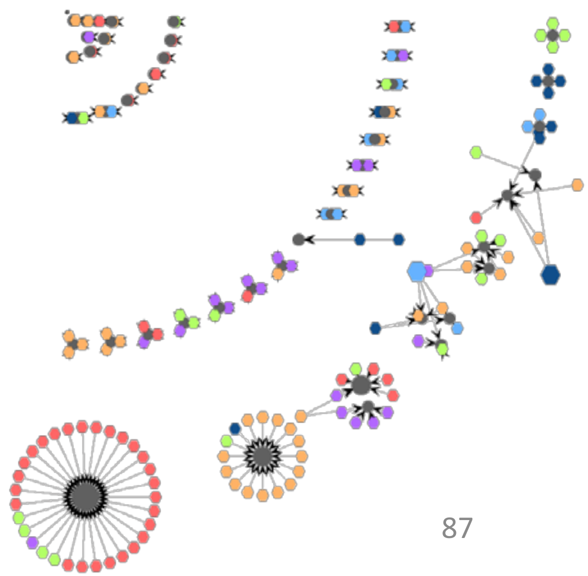


Concentric arcs

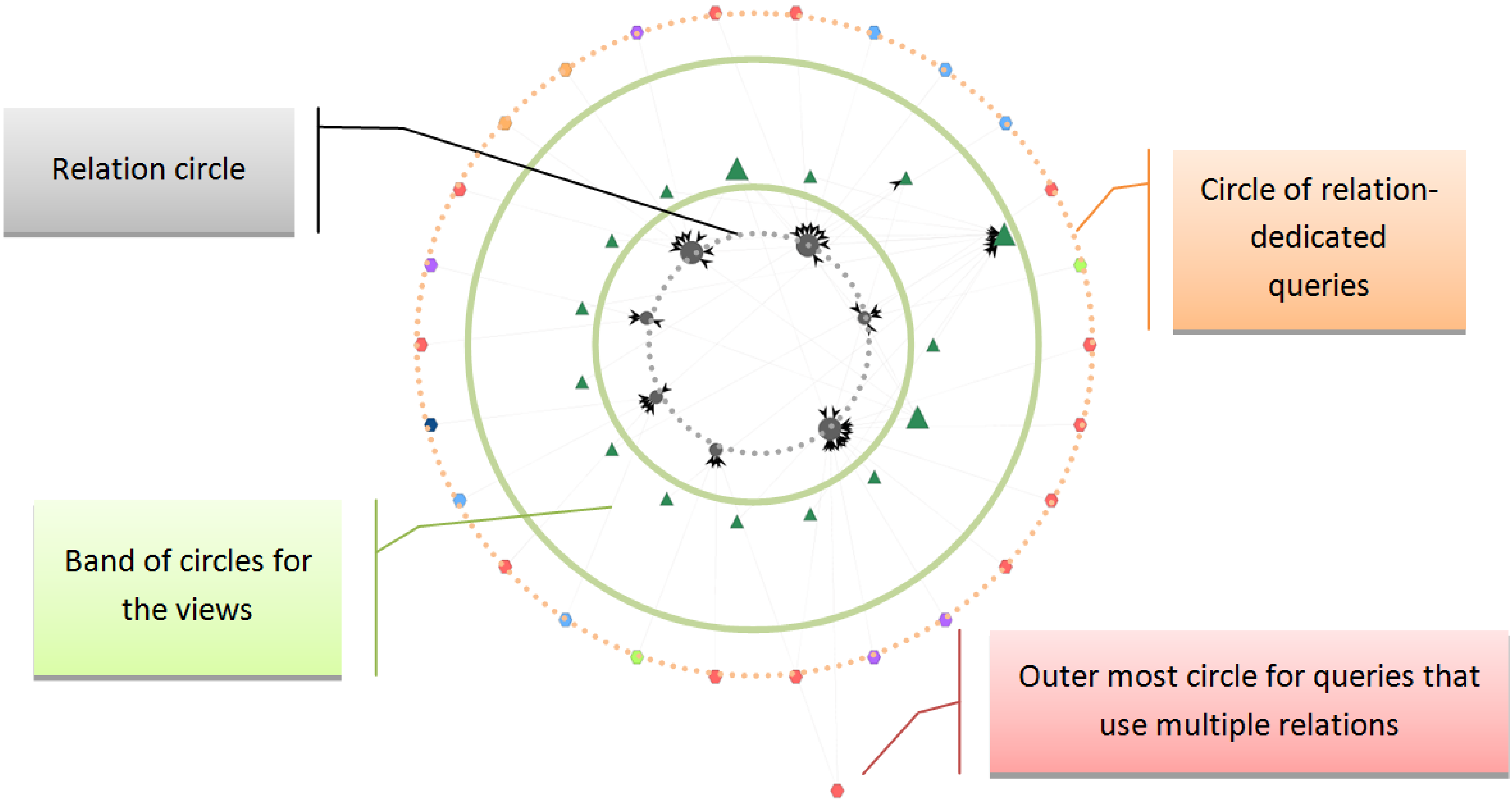


OpenCart

ZenCart



Step 4: arrangement of nodes within the circular clusters



Remember: 4 bands of circles to place nodes

Step 4: arrangement of nodes within the circular clusters

- We want to follow a barycenter based method, which can work successfully for layered drawings
- The standard barycenter method works with linear layers with the principle that once you have laid out layer i, you can lay out layer i+1 wrt the previous one
  - … practically placing nodes in the barycenter of their neighbors in the previous layer i
- Here, we have two challenges:
  - adapt this to our radial, concentric circles
  - decide the initial order of the process (here: relations in the inner circle)

```
order R;
place R & Q_R;
place V & Q-Q_R.
```

Step 4: arrangement of nodes within the circular clusters

1. Order the relations
   1. Count the frequency of each combination of tables as hit by the queries
   2. Place tables in popular combinations sequentially
2. Decide the position of relations and relation-dedicated queries
   1. Locate relation dedicated queries, decide the arc they need and position them sequentially
   2. Place relation in the middle of this arc
3. Decide the position of the rest of the queries and the views
   1. Stratify views and queries – each stratum has a dedicated circle
   2. Place views and queries via a barycenter method on their angle
   3. Adjust overlapping nodes (e.g., queries hitting exactly the same tables)

Algorithm 3. Circle Layout

Input: a cluster *C*

Variables: *T*: the tables of *C*, *V*: the views of *C*, *Q*: the queries of *C*, *S*: a list of (topologically sorted) strata over $T \cup V \cup Q$

Output: an angle φ_i for each node *i* of the cluster *C*

Begin

1. //Order relations
2. Let *L* be a list of pairs $[c, c_n]$, with *c* being a combination of relations hit by queries in the cluster and c_n the frequency of *c*; $L = \varnothing$
3. Iterate over Q to compute *L*
4. Sort *L* in decreasing order
5. Let T* be the list of relations (ultimately in sorted order); $T^* = \varnothing$
6. **For** each combination *l* in L, add its contents to T*, if not already in T*
7. //Place relations and relation-dedicated queries
8. **For** each table t_i in T
9. Compute $Qd(t_i)$ = the relation-dedicated queries of t_i
10. Compute the angle of the segment that pertains to $Qd(t_i)$

    $\omega_i = |Qd(t_i)| * d\phi$,

    //*dφ is the angle per node computed as 2π / #nodes of the circle*
11. Place t_i in the middle of its segment, next to the previous:

    $\phi_i = 0.5 * \omega_i + \phi_{i-1}$
12. Place the queries of $Qd(t_i)$ sequentially in their circle
13. //Place views and queries
14. **For** every circle k_i of the cluster's circles, in the order determined by Algorithm *Circle Identification*
15. **For** each node v_j in k_i
16. Sum all the angles of the nodes belonging to the cluster that v_j accesses in the previous circles and divide by their number to compute the node's angle ϕ_j

End

```
order R;
place R & Q_R;
place V & Q-Q_R.
```

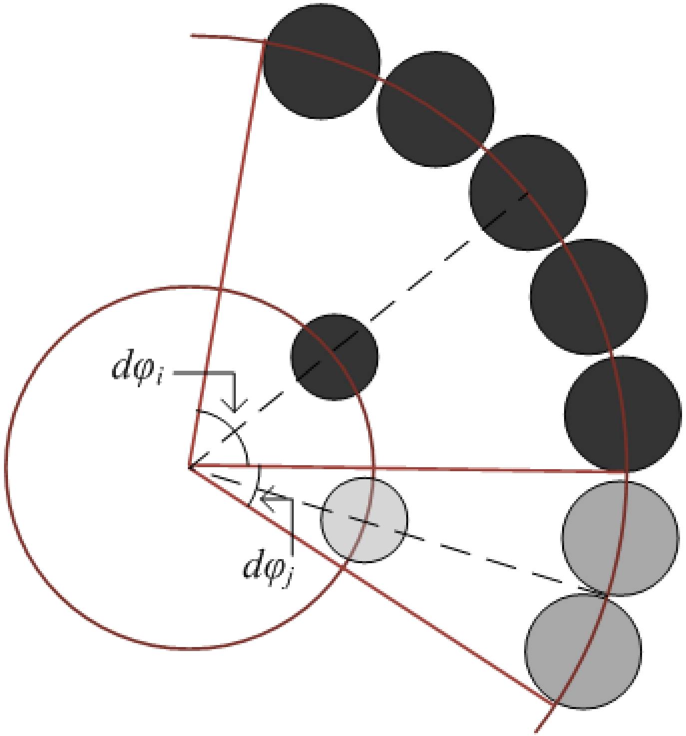
Step 4: arrangement of nodes within the circular clusters

- Greedy arrangement that places sequentially tables in frequent combinations (as they appear in the FROM clause of queries)
- Assume L= {{*T4*, *T3*}, {*T1*, *T5*}, {*T1*, *T2*, *T3*}} in decreasing order of frequency.
- Then, the final order of the relations will be {*T4*, *T3*, *T1*, *T5*, *T2*}

```
order R;
place R & Q_R;
place V & Q-Q_R.
```

Step 4: arrangement of nodes within the circular clusters

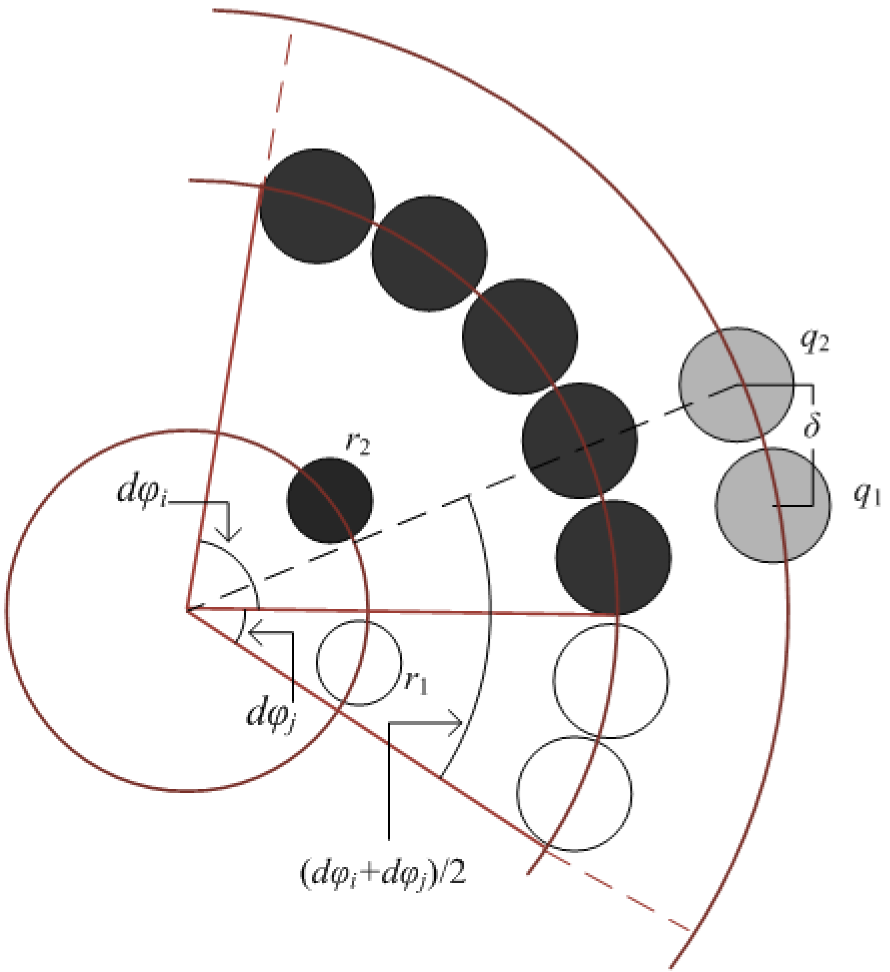
- Place relation-dedicated queries (accessing only one table) sequentially
- The segment for each relation-dedicated query is the same, proportional to the #nodes of the circle (#relation-dedicated queries)
- The relation is placed in the middle of the corresponding segment



```
order R;
place R & Q_R;
place V & Q-Q_R.
```

Step 4: arrangement of nodes within the circular clusters

- We have stratified views and the rest of the queries in step 2.
- Visit each circle (stratum) in turn: each node of circle k_i is defined over nodes in the previous circles
- Traditional barycenter-based method: place the node in an angle = avg. value of the sum of the angles of the nodes it accessed
- Rearrange nodes in occupied positions by a very small value δ (in our case δ=0.09 radians).



Experiments and Results

Experimental method

- 4 data sets: well-known open source projects that contain database queries
- The source code of the last version of each tool was downloaded. We retrieved the database definition from the source code.
- We grepped the source code for the occurrences of SELECT and FROM terms in it, and out of the resulting text, we isolated lines actually encompassing queries.
- The actual queries were automatically isolated via a dedicated java application we constructed for this purpose
- Finally, they were post-processed in order to be parsable by our tool, Hecataeus that ultimately converts the ecosystem to an architecture graph and visualizes it for the user.

Data sets

Dataset	Description	R	V	Q	E
Biosql	A generic relational schema covering sequences, features, sequence and feature annotation, a reference taxonomy, and ontologies	28	15	79	104
ZenCart	An open source shopping cart software	106	0	149	158
Drupal	An open source content management platform	75	0	355	379
OpenCart	An open source shopping cart software	115	0	650	824

Compliance to aesthetic criteria

Aesthetic Criteria	**Circular Layout**	**Concentric Circle Layout**	**Concentric Arc Layout**
Proximity	Clustering		
Connectedness	graph representation and clustering		
Similarity and Proportion	Same shape and color for same type of nodes, size proportionate to connections		
Closure and Isolation	Circular visualizations, white border between clusters and nodes		
Clutter Avoidance	**In clusters**: different angle for each cluster depending on its size	**In clusters**: concentric circle radius optimization	**In clusters**: different angle for each cluster, radius optimization
	In graph: clustering algorithm produces "perfect" clusters – no intra cluster edges **In edges**: barycentric method for node placement, light edge coloring		

Compliance to the Visual Information Seeking Mantra

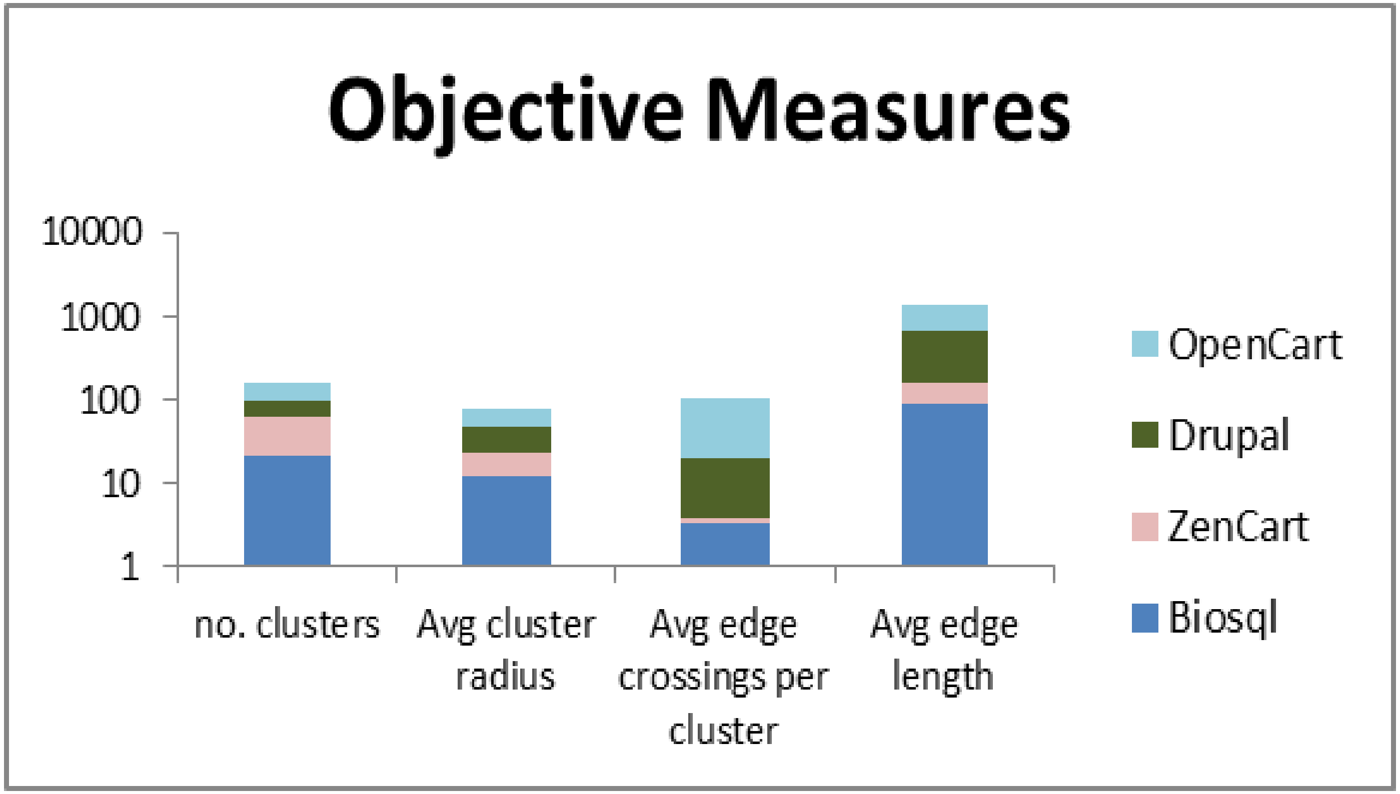
Visual Information Seeking Mantra principles	**How we address them**
Overview	We give an overview of the graph by visualizing only the higher level nodes. For an overview of the clusters, we created an overview map that represents each cluster as a single node
Zoom	We implemented the zooming feature by providing a more detailed view of the selected parts of the graph in a different tab
Filter	The fact that we support zooming in user specified areas of interest and provide a new tab with only the user selected modules in higher detail is a way of filtering information.
Details-on-Demand	This principle is also available via the zoom in feature we implemented

Clustering effectiveness

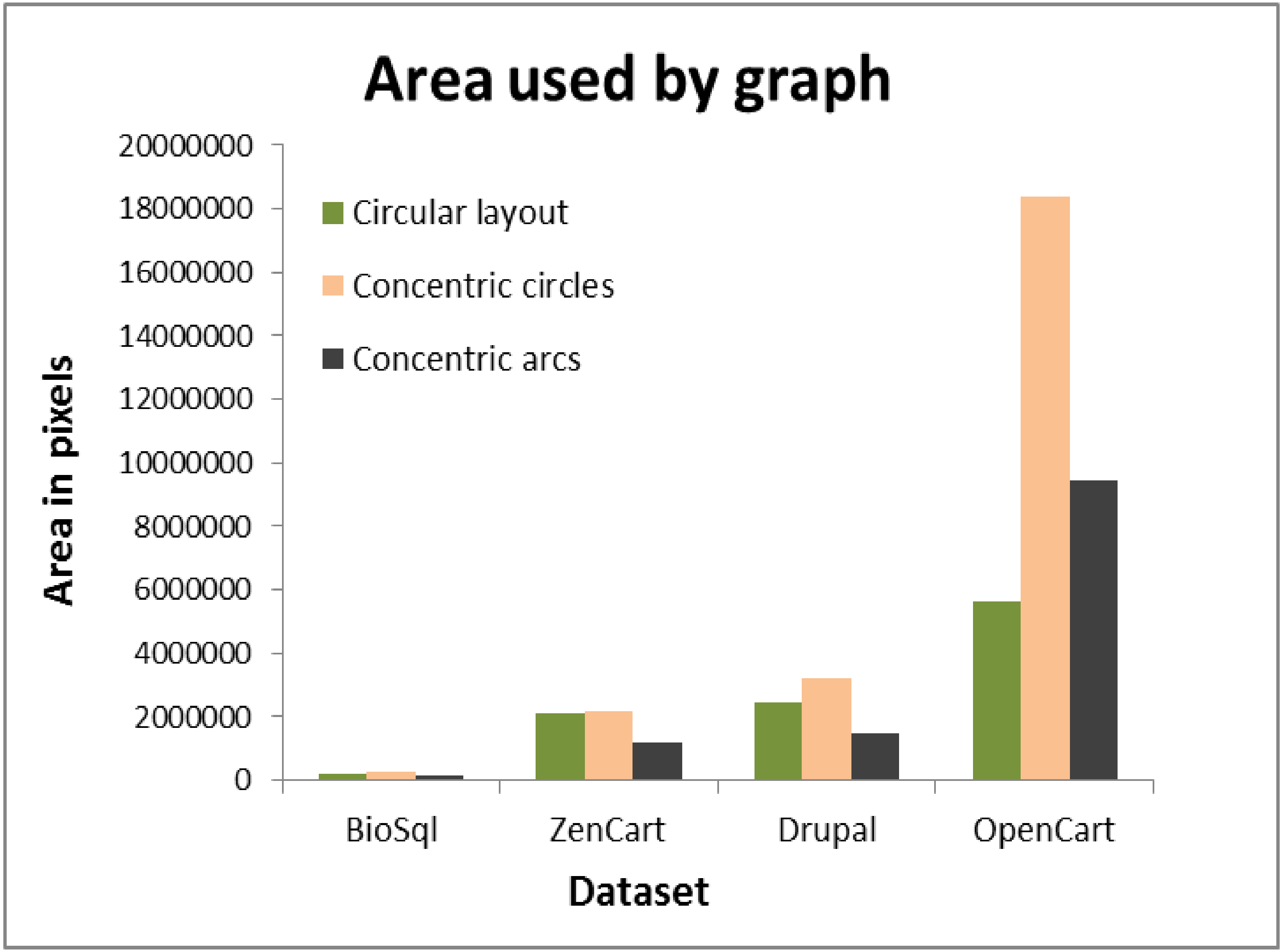
- Clean separation of clusters in all cases!
- However, this can be due to the nature of the ecosystems – must not hurry to generalize
- In all cases, 20 – 60 clusters
  - We find this reasonable for a 2D screen canvas

Data set	# clusters	# nodes	Avg nodes/cluster
BioSql	22	112	5,55
ZenCart	41	255	4,8
Drupal	37	429	11
OpenCart	59	765	12,9

Method independent measures for all data sets



Area occupied by the graph: no clear winner

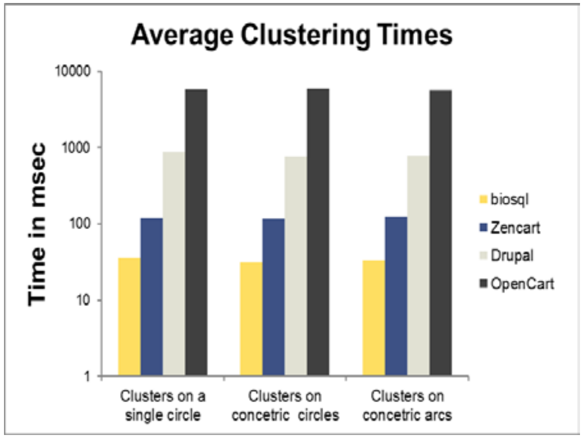


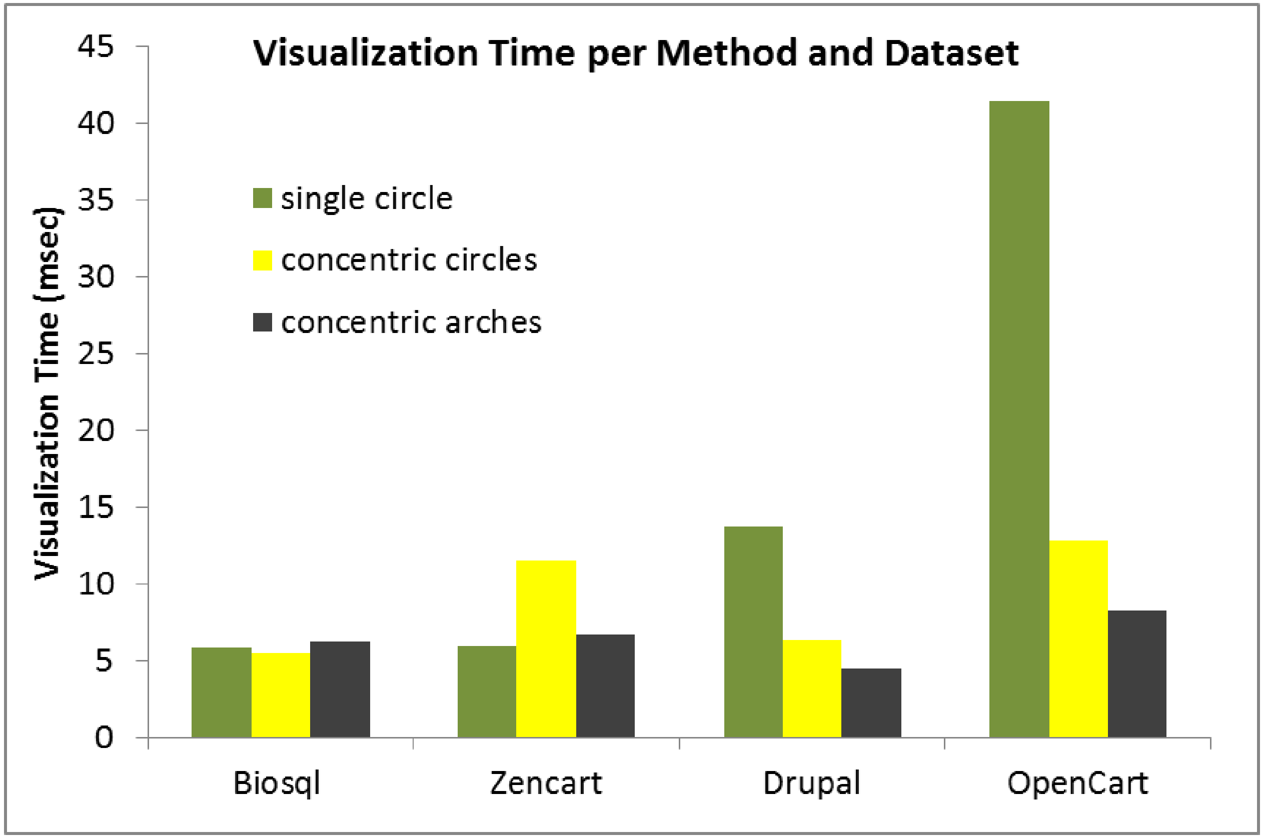
Area occupied by the produced graph (absolute and pct over MBR)

Data set	Circular layout	Concentric circles	Concentric arcs	Covered area
BioSql	196243.49	**275943.12**	193640.75	44549.6
ZenCart	2007635.67	**2162419.52**	1238295.66	50739.45
Drupal	2329122.25	**3232092.83**	1612675.06	253172.6
OpenCart	5775976.36	**18392055.26**	9711796.44	461424.53

Data set	Circular layout	Concentric circles	Concentric arcs	Covered area
BioSql	22%	16%	23%	44549.6
ZenCart	2%	2%	4%	50739.45
Drupal	10%	7%	15%	253172.6
OpenCart	7%	2%	4%	461424.53

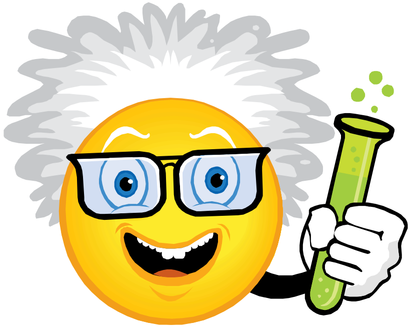
Time considerations





Any other useful slides

Future work



- Ongoing:
  - Still experimenting with coloring schemes

- Open:
  - Alternative visualization methods
  - Other types/cases of ecosystems

  - Forward engineering
  - Incorporate full range of “entities” / “concepts” (actors, roles, …) to make it really an “ecosystem”…

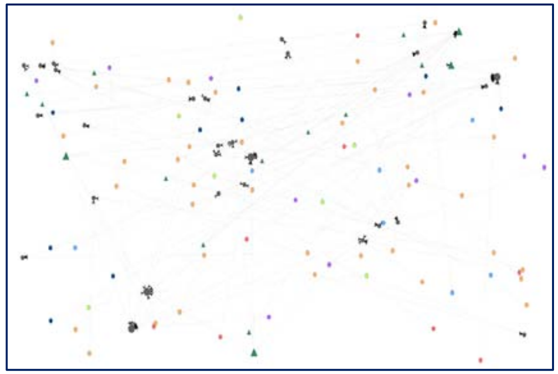
Alternative methods provided by Jung

- A simple Circular Layout that places vertices randomly on a circle
- The Fruchterman-Reingold algorithm [FrRe91]
- Meyer's "Self-Organizing Map" layout [Meye98]
- The Kamada-Kawai algorithm [KaKa89]
- A simple force-directed spring-embeder [dETT99]

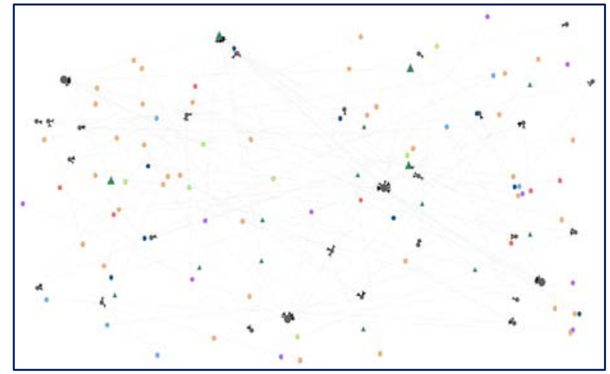
Alternative methods provided by Jung

Biosql visualized by Jung's built – in's

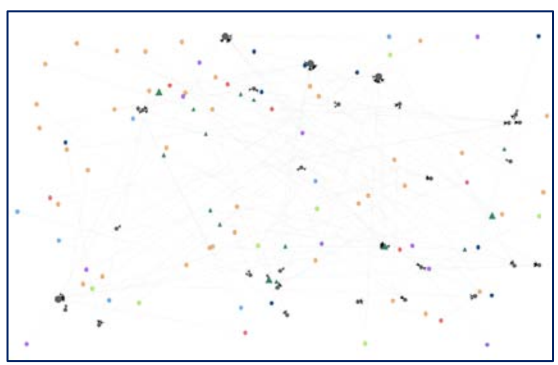
Random Circular



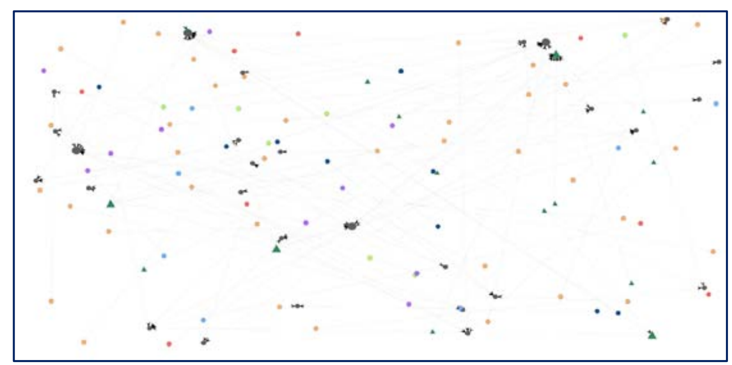
Fruchterman-Reingold



Meyer



Kamada-Kawai

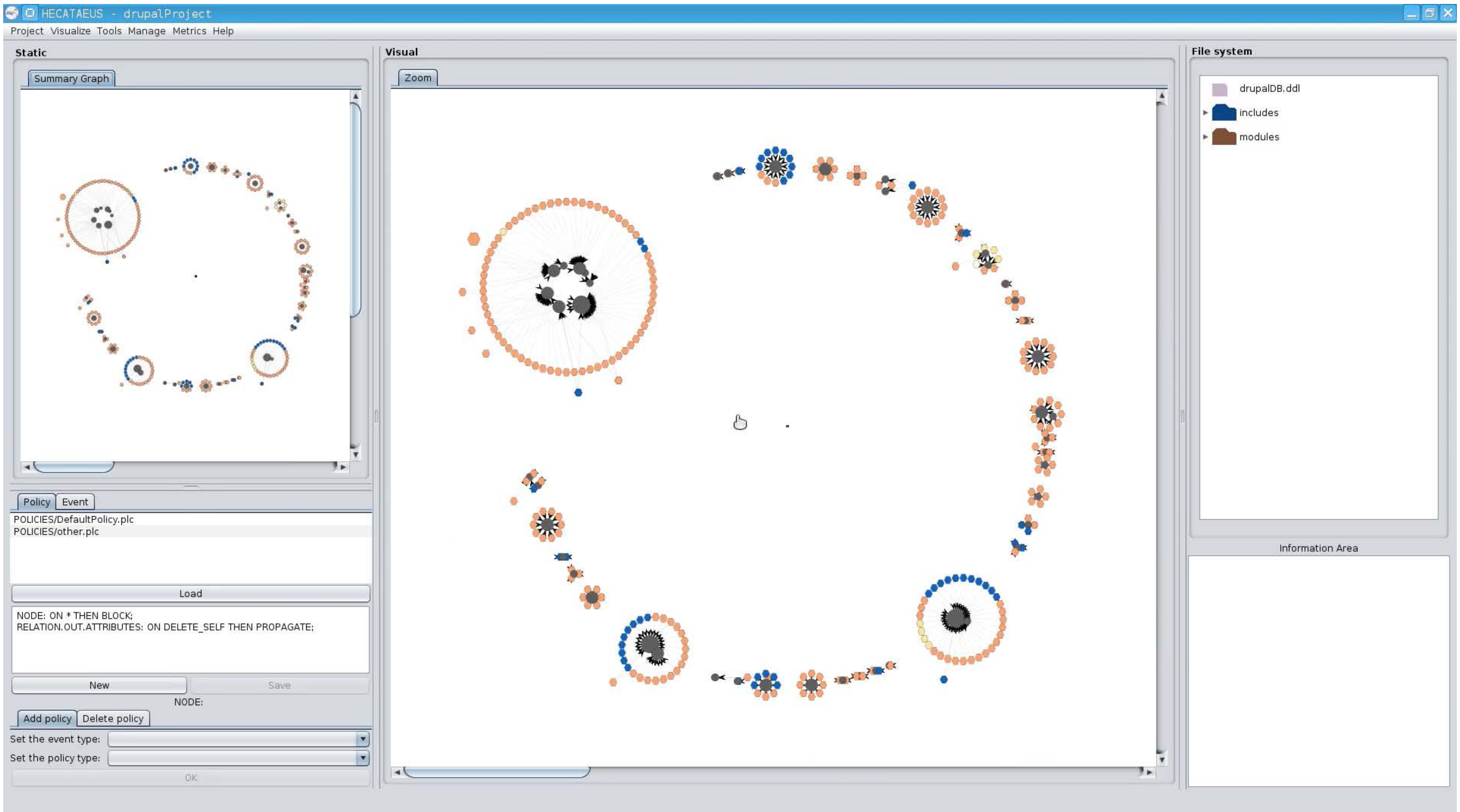


Force-directed springs

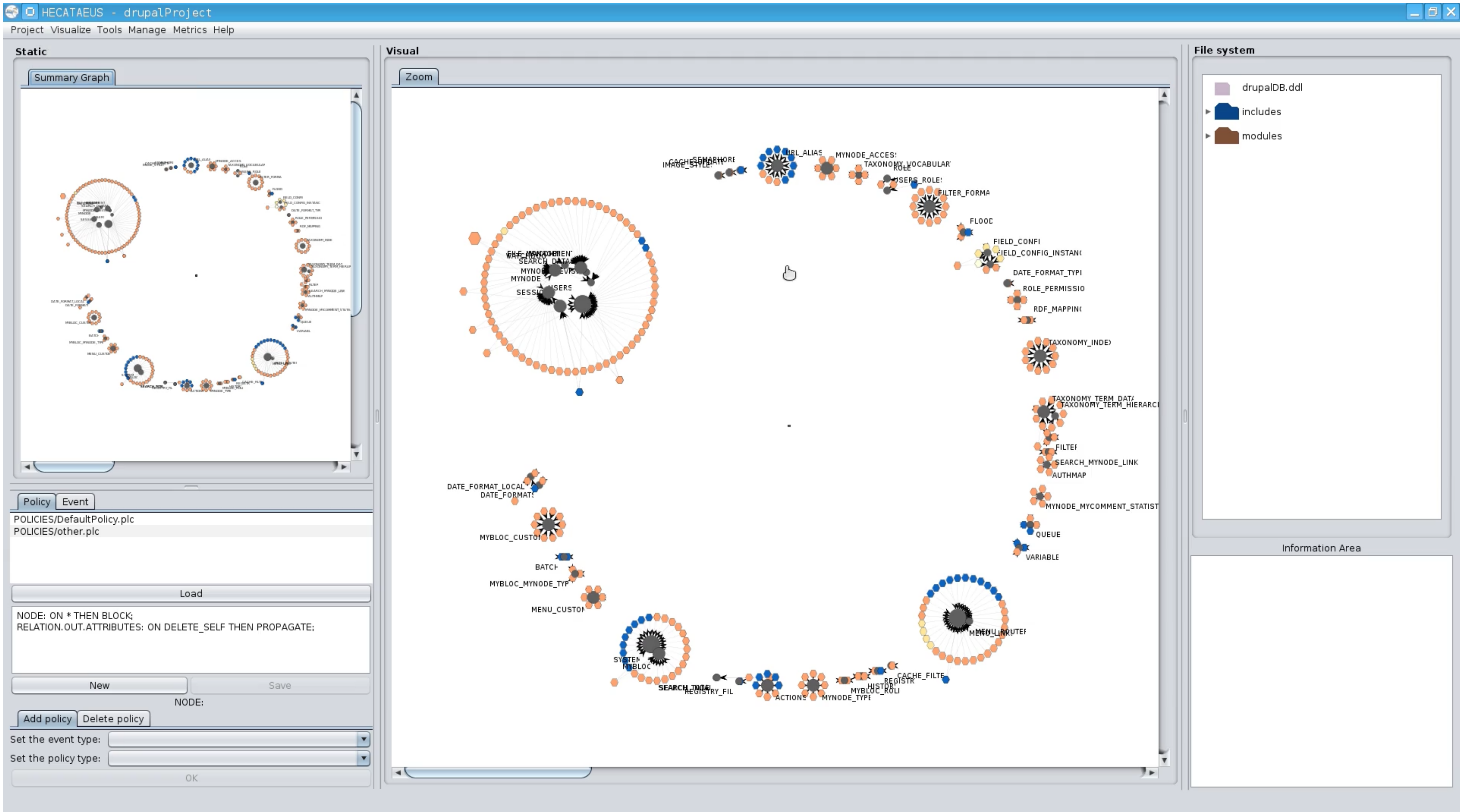
A short story executed over one of the latest v. of Hecataeus

TALES OF GLORY

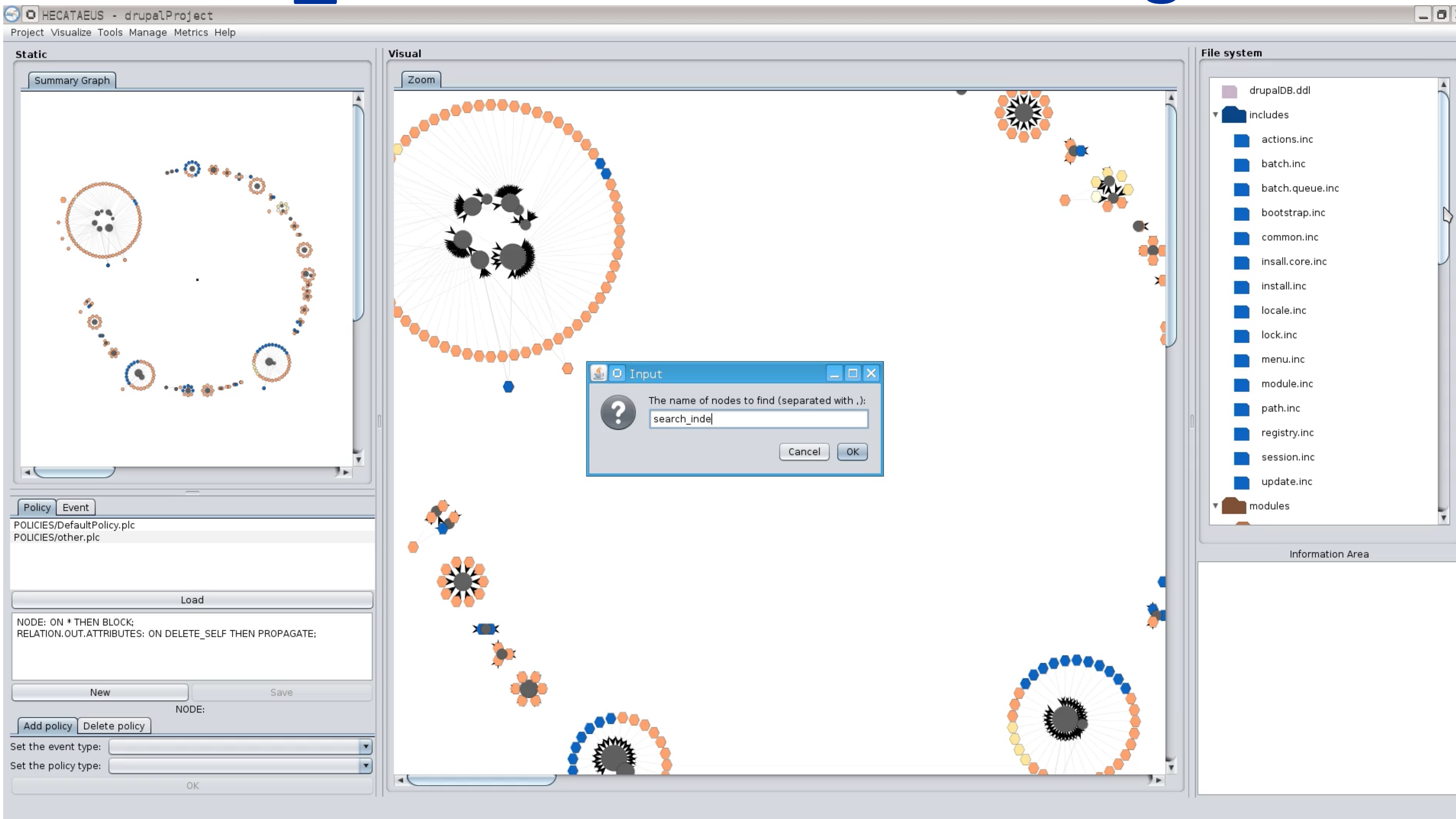
Circular placement



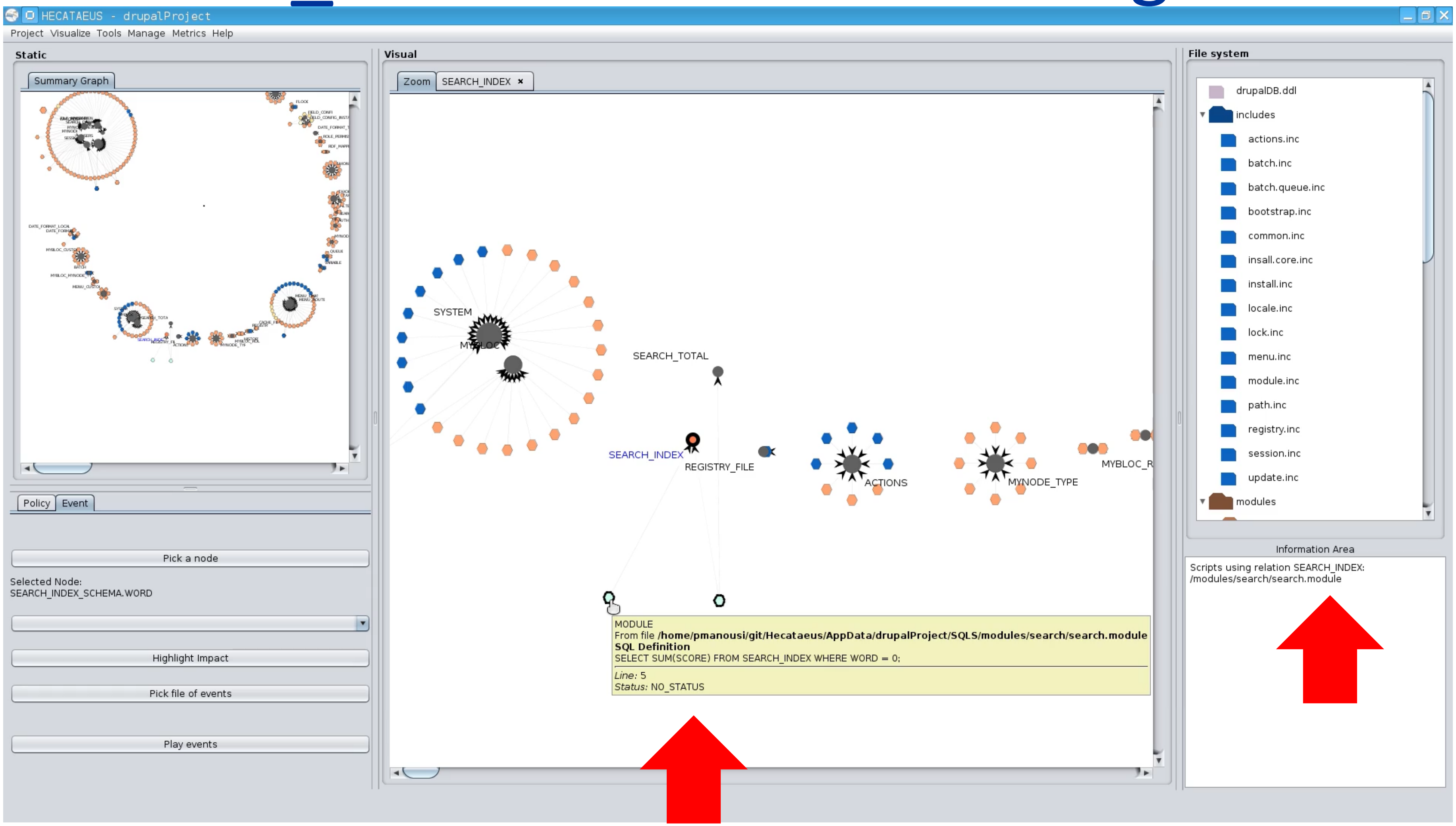
Labels on/off



What happens if I modify table search_index? Who are the neighbors?

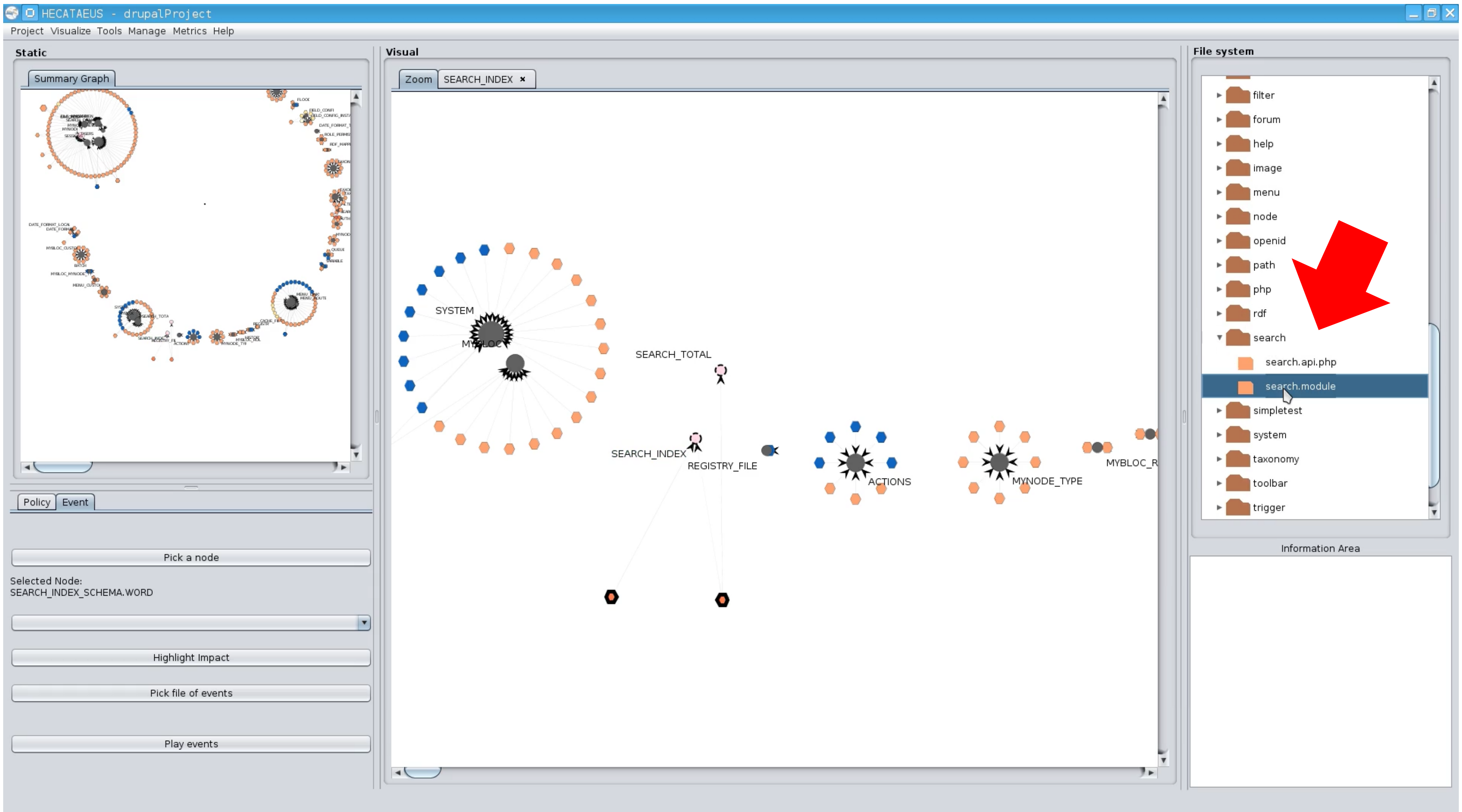


What happens if I modify table search_index? Who are the neighbors?

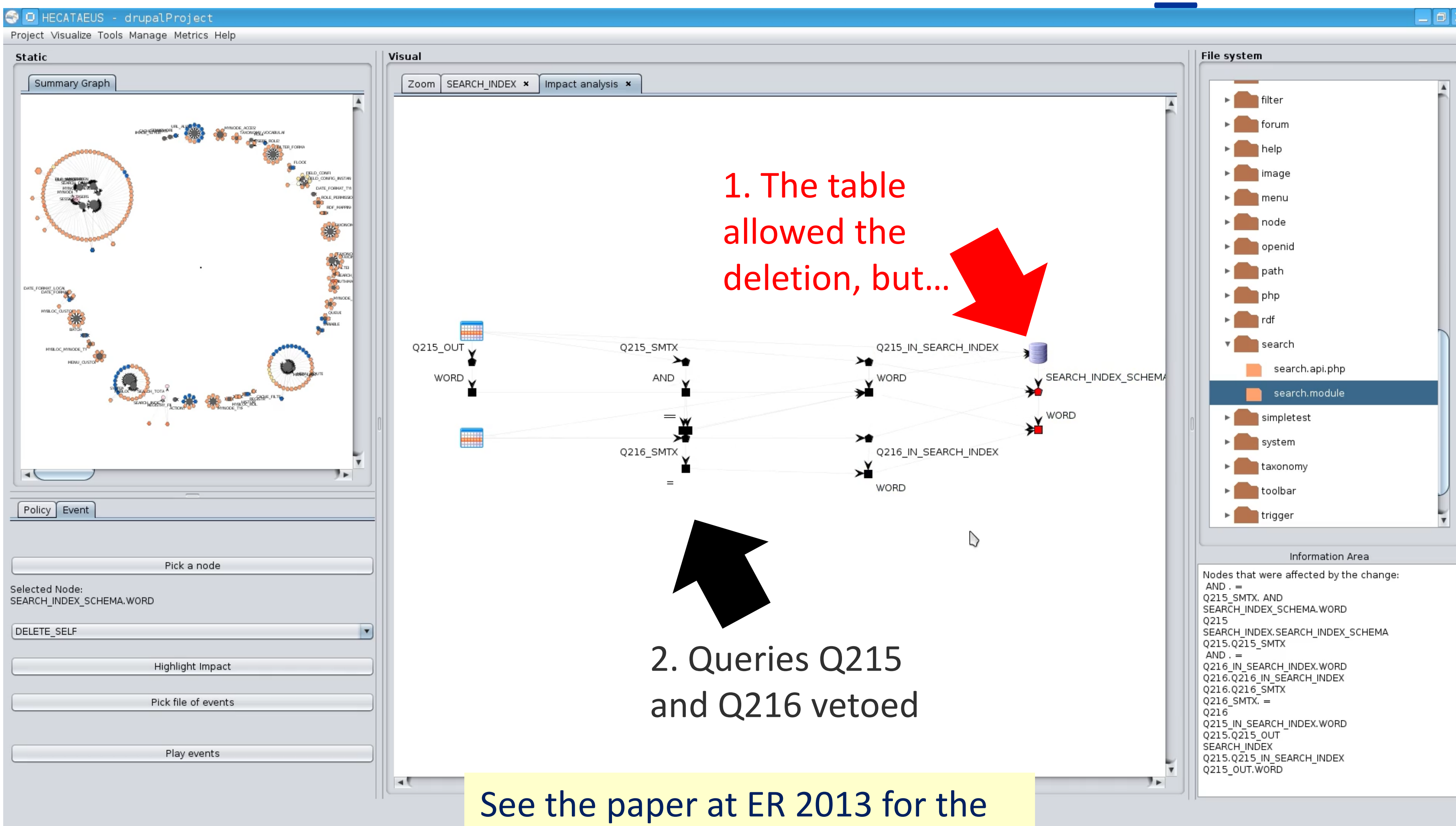


Tooltips with info on the script & query

In the file structure too…



Played an impact analysis scenario: delete attr. ‘word’ from search_index

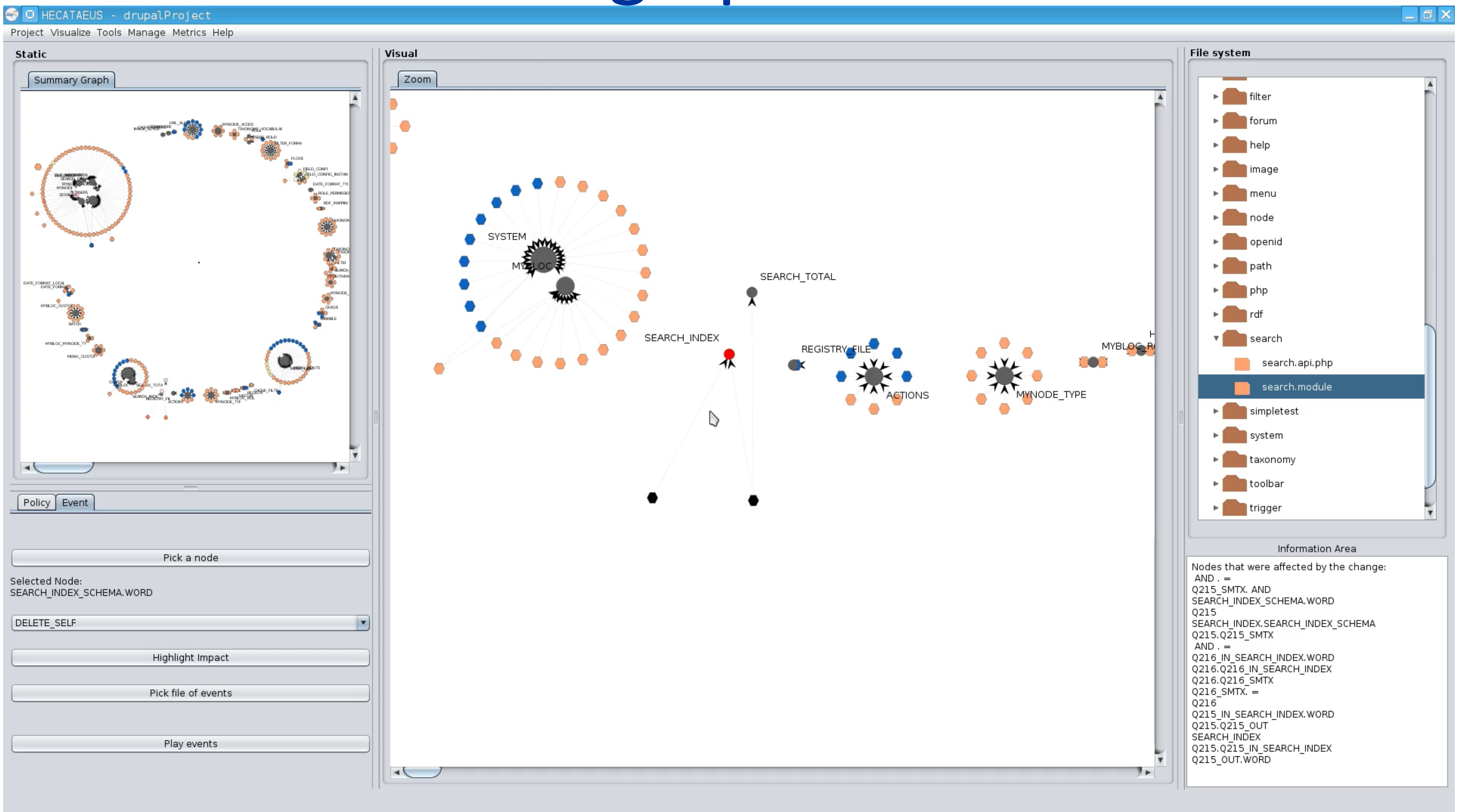


1. The table allowed the deletion, but…

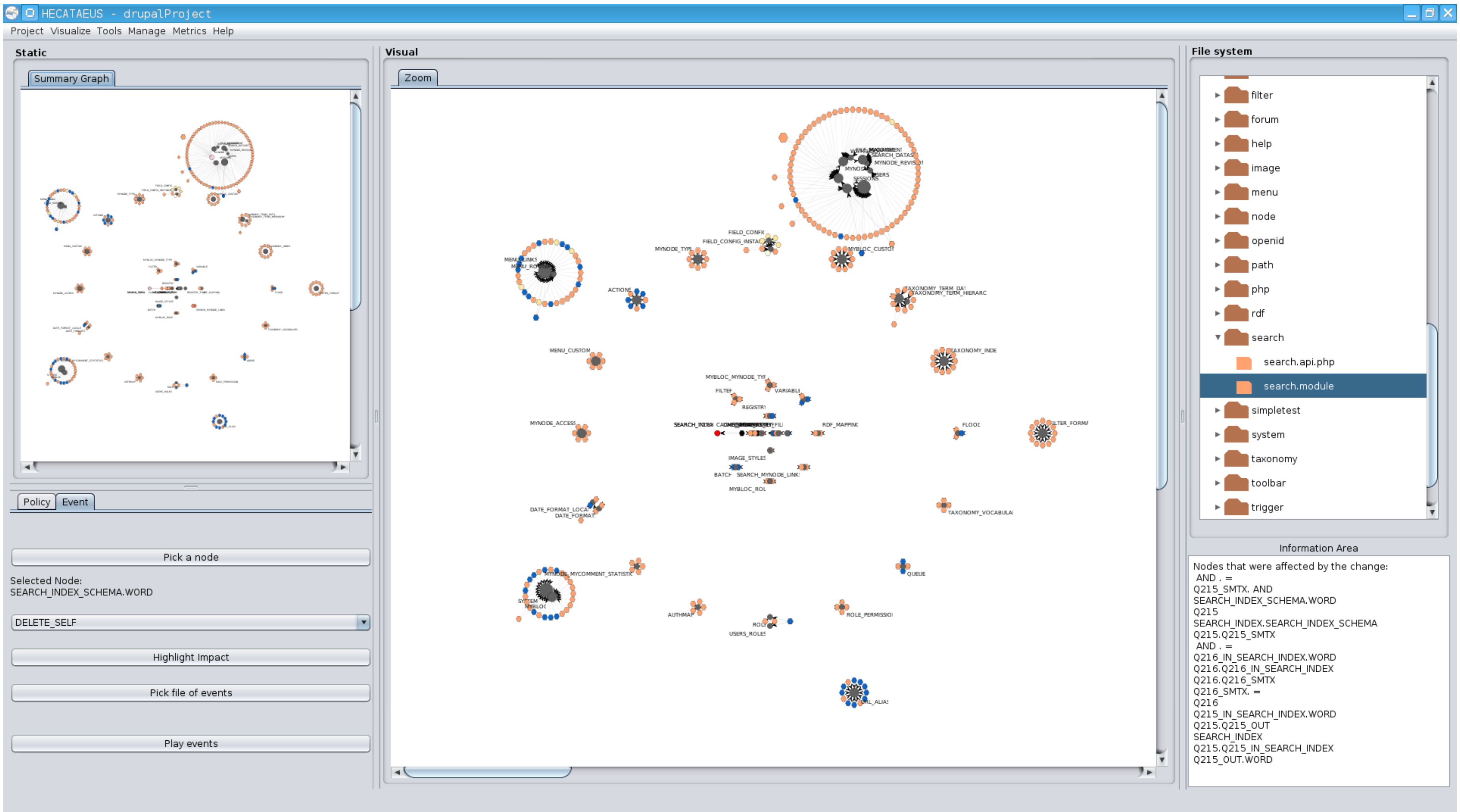
2. Queries Q215 and Q216 vetoed

See the paper at ER 2013 for the details of impact analysis

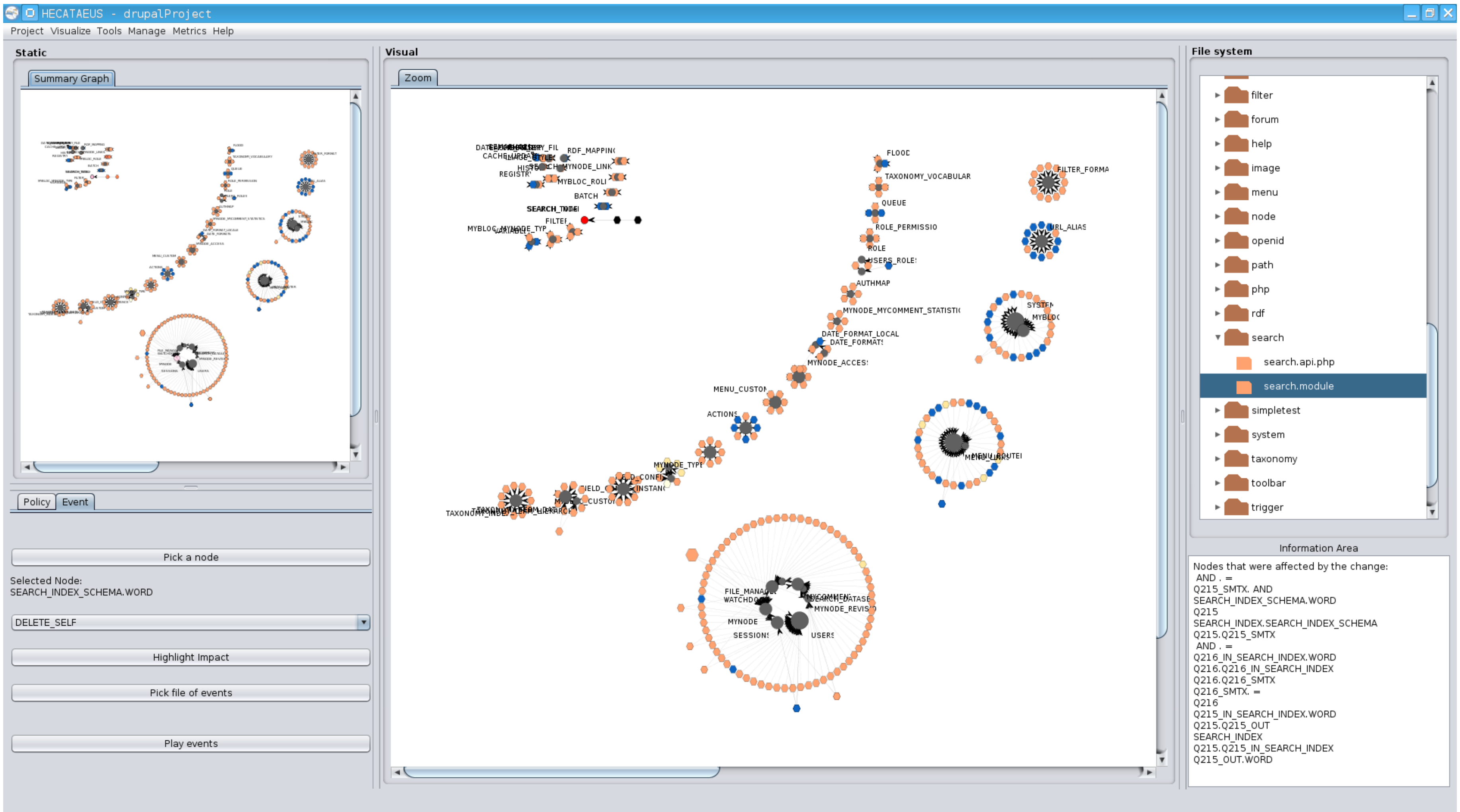
… and the impact at the master graph…



Concentric circles



Concentric Arcs



Zooming into a cluster