LIX, École Polytechnique

Software Modelling and Architecture: Exercises

Leo Liberti

Last update: November 22, 2006

Contents

Chapter 1

Introduction

This exercise book is meant to go with the course INF561 given at École Polytechnique by Prof. D. Krob. The current course edition is 1st semester 2006/2007. It contains a series of exercises in software modelling and architecture.

1.1 Structure of this book

Software modelling and software architecture are concepts needed when planning complex software systems. Consequently, it does not really make sense to concentrate on a sequence of self-contained, modestly-sized and independent exercises as would be usual for an exercise book. The difficulty of this discipline resides in the large size of the tasks at hand. We shall therefore pretend that the class is a team in a small software engineering firm (called *VirtualClass Ltd.*) engaged in a job contract. We shall enforce the more or less usual "real-life" constraints and impediments encountered in the business world, like ambiguous customer requirements, lack of crucial technical information, over-inundation of unimportant facts, and so on.

1.2 The setting

T-Sale is a large multinational firm which is often employed by national governments and other large institutions to provide very large-scale services. They will secure contracts by responding to the prospective customers' public tenders with commercial offers that have to be competitive. The upper management of T-Sale noticed some inefficiencies in the way these commercial offers are put together, in that very often the risk analysis are incorrect. They decided that they could improve the situation by trying to use stored information about past projects. More precisely, T-Sale keeps a detailed project database which allows one to see how an initial commercial offer became the true service that was eventually sold to the customer. The management hope that the preliminary customer requirements contained in the public tender may be successfully matched with the stored initial requirements to draw some meaningful inference on how the project actually turned out in the past.

T-Sale wants to enter into a contract with VirtualClass to provide the following service, which was expressed in very vague terms from one senior vice-president of T-Sale to VirtualClass.

> *We want a sort of "Google" for starting projects. We want to find all past projects which were similar at the initial stage and we want to know how they developed; this should give us some idea of future development of the current project.*

VirtualClass must estimate the cost and time required to complete this task, and make T-Sale a competitive offer. Should T-Sale accept the offer, VirtualClass will then have to actually plan and implement the system.

1.3 Aim

As was mentioned previously, we shall pretend that the class is the team of VirtualClass' software engineers that need to put together a meaningful and competitive offer for T-Sale, and then to plan the system.

1.4 Tools

The software engineers are supposed to use the following tools to attain their aims:

- *brainstorming meetings* to jump-start a phase or to try to get around a dead point;
- *normal meetings* to agree on a plan;
- *DBDesigner* as a database design tool;
- *Poseidon for UML* as a UML modelling tool;
- *oXygen* as an XML modelling tool.

Notes, diagrams, documentation and specifications should be standardized, uniform, and consistent.

1.5 Roadmap

1. The commercial offer needs to be drawn quickly. The associated risks should be assessed. It should be as close as possible to the delivered product.
2. In general, the software engineering team should follow the "V" development process (left branch) for planning the system, as shown in Fig. 1.1.

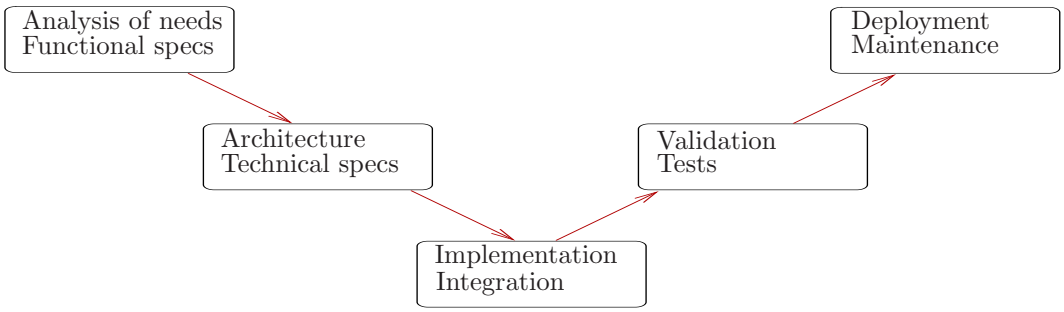


Figure 1.1: The "V" development process.

Chapter 2

Initial offer

2.1 Kick-off meeting

Aims of the meeting:

1. Formalize the customer requirements as much as possible
   (a) What is the deliverable, i.e. what is actually sold to the customer?
   (b) What is the first coarse "common-sense" system breakdown?
2. What data is needed from T-Sale's databases?

Length: 45 min.

2.1.1 Meeting output

1. Given some meaningful key-words or other well-defined indicators in the description of a new project, we want to classify it by some quantitative indices and look in a project database for all projects which were sufficiently similar at the initial stage and proceeded to completion with a uniform degree of success; we should then display a list of such projects so that the user can immediately glance at all important information concerning risk-assessment.
   (a) The deliverable is a software module that must be plugged in the existing T-Sale back-office network; it should have query access to some of the T-Sale databases and should be usable through a web interface.
   (b) At a first analysis, we shall need:
      - I/O user interface through a web browser;
      - a way to find meaningful indicators in the given project;
      - a system to query the databases for those indicators and return information about initial, intermediate and final cost, time and resources estimates.
2. We shall need T-Sales' data concerning:
   - project descriptions;
   - project schedules;
   - project costs;

- teams involved;
- people involved;
- other resources involved.

T-Sale's answer, as often happens, is rather vague.

> Dear VirtualClass Team,
> We are sorry to have to tell you that the structure of our databases is classified information, and we will only be able to give it to you at a later stage when and if we choose to employ your services. We can however describe the main features of what we think is useful to your job. We have an HR database detailing the usual information (salary, rank,...) abilities and skills. We have a technical database with project information (nature and cost of project, teams, people, schedule and associated changes). We naturally have a commercial database detailing customers and payments. Unfortunately the database which details hardware resources and costs may not be accessed as it contains some information classified at national level.
> Best regards,
> A. Smith

2.2 Brainstorming meeting

Aims of the meeting:

1. propose ideas for a system plan with sufficient details for a rough cost estimate;
2. collect these ideas in a formal document;
3. decide on a sexy project name.

Length: 45 min.

2.2.1 Meeting output

1. User will input project indicators known at early stage
2. **Functionality**: an input web form (user interface)
3. Which among these "early indicators" are quantitative, which qualitative?
4. What sort of clustering of the project space do they lead to?
5. According to what other indicators ("late indicators") can project be clustered?
6. **Functionality**: cluster projects according to a given quantitative/qualitative indicator (computational engine)
7. **Functionality**: access the customer database (database module)
8. How do we assess the quality of a clustering?
9. How "clear-cut" is a clustering?
10. **Functionality**: clustering significance evaluator (computational engine)
11. How are the early/late clusterings used later on?

12. **Functionality**: record a clustering (database module)

13. New projects must be classified according to early indicators: how do we use the information given by the clusterings obtained with late indicators? More in general, how do we pick a set of significant late clustering (which give the useful risk assessment information) given an early clustering?

14. **Functionality**: clustering compatibility evaluator (computational engine)

15. Literature review on clustering

16. How do we classify a new project according to the stored clusterings?

17. **Functionality**: query clusterings for

18. How do we present the output to the user?

19. **Functionality**: output form (user interface)

20. Name: how about "proogle" (the "project google"?)

The Proogle system will require the following functionalities:

- input/output web user interface;
- a computational engine for clustering according to a quantitative or qualitative indicator;
- a computational engine for evaluating clustering significance;
- a database module for storing clusterings;
- a computational engine for evaluating clustering compatibility;
- a database module for querying the stored clusterings.

Computational engines will require expertise in clustering techniques; database modules should be sufficiently straightforward; presenting output in a meaningful way will likely pose problems.

2.3 Formalization of a commercial offer

Aims of the meeting:

1. write a document (for internal use) which gives a rough overview of the system functionalities and of the system breakdown into sub-systems and interdependencies;

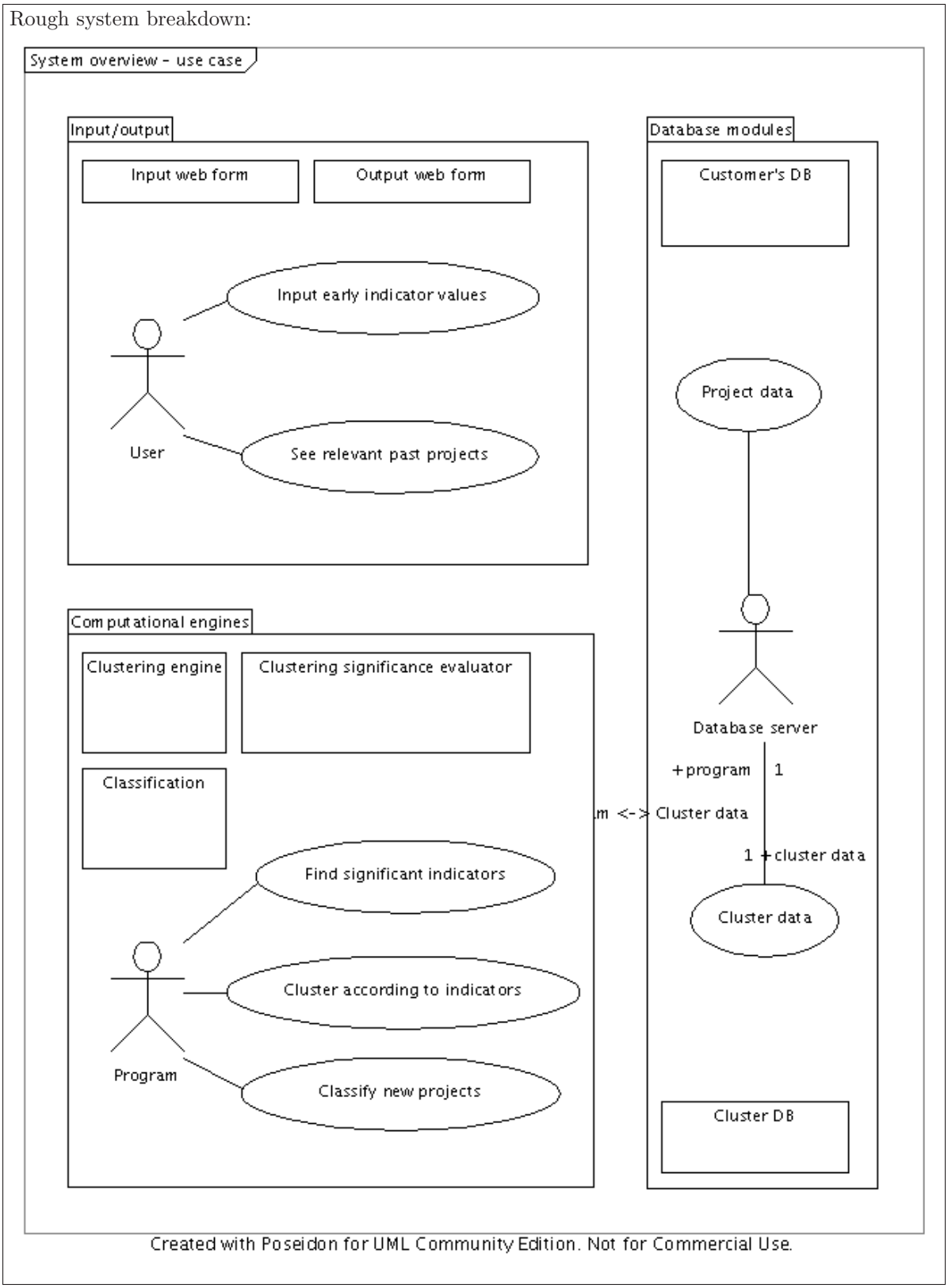
2. write a document (for internal use) with projected sub-system costs (complexity) and a rough risk assessment;

3. write a commercial offer to be sent to T-Sale with functionalities and the total cost.

Length: 1h.

2.3.1 Meeting output

Rough system breakdown:



Risks:

1. Failure to obtain necessary data/clearance from T-Sale — catastrophic, low probability
2. Not enough specific in-house clustering expertise — serious, high probability
3. Results not as useful as expected — low, medium probability

Address risks:

1. Insert clause in contract
2. Plan training
3. Insert clause in contract

Chapter 3

System planning

We shall now suppose that T-Sale accepted VirtualClass' offer and is now engaged in a contract. The next step is to actually plan the system. The contract clearly states that T-Sale is under obligation to provide T-Sale with database details, which are shown in Fig. 3.1.

3.1 Understanding T-Sale's database structure

Aims of the meeting: analysis and documentation of T-Sales' database structure. Note that the project's *condition* contains information about whether the project was a success or a failure, and other overall properties. Make sure every software engineer understands the database structure by answering the following questions:

1. How do we find the main occupation of an employee?
2. How do we find the expertises of an employee?
3. How do we find the condition of a project?
4. How do we find how many times a project was changed?
5. How do we find whether a project was paid for on time or late?
6. How do we find whether a customer usually pays on time or late?
7. How do we verify that the cost of all phases in a project sums up to the total project cost?
8. How do we evaluate the cost in function of time during the project's lifetime?
9. How do we discriminate between the phase cost due to human resources and the cost due to other reasons?
10. How do we find the expertises (with their levels) that were necessary in a given project?
11. How do we find out the abilities and skills (with their levels) that were necessary in a given project?
12. How do we find out which teams were most successful?
13. How do we find out the most dangerous personal incompatibilities?

Length: 1h.

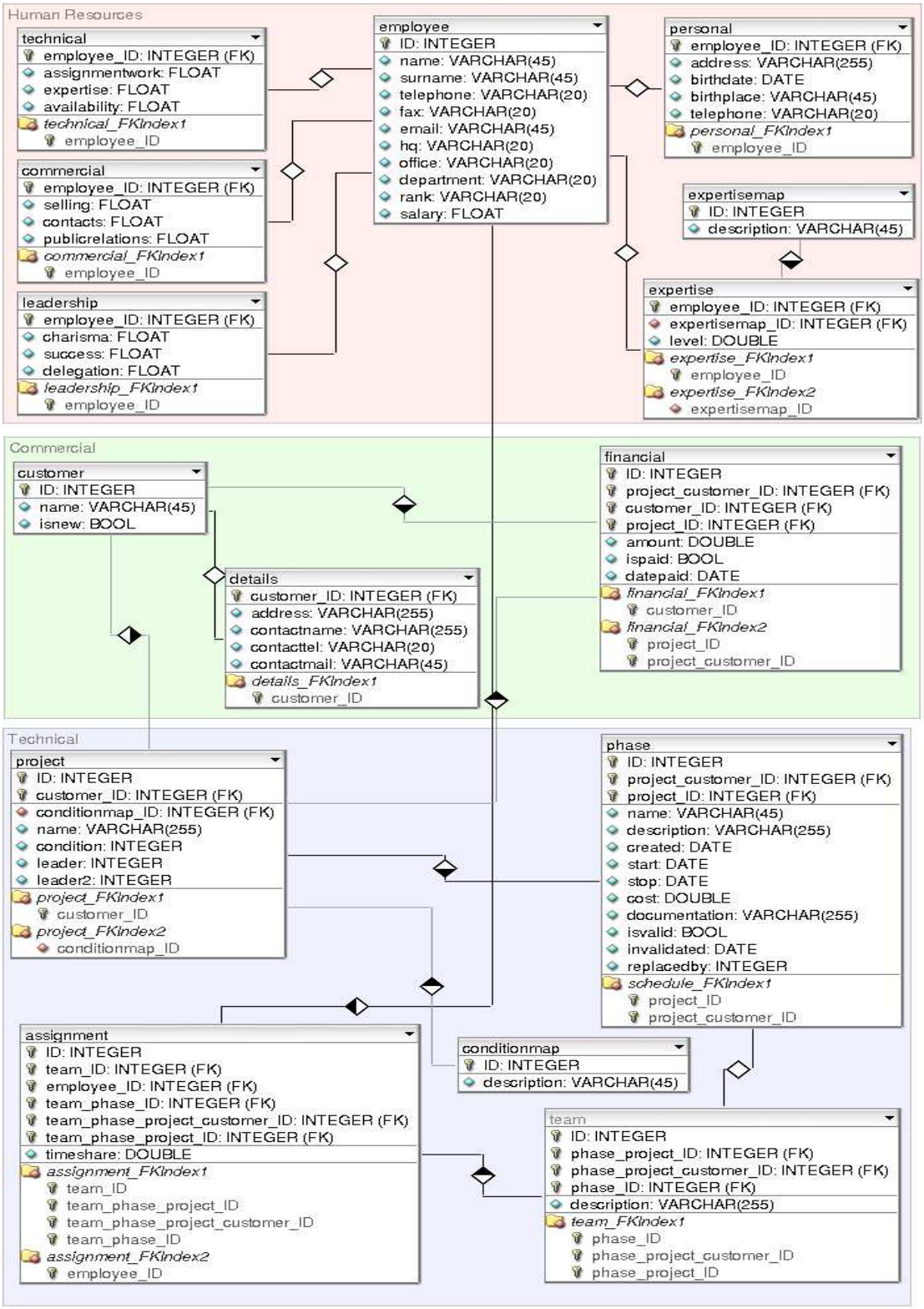


Figure 3.1: T-Sales' database structure.

3.1.1 Meeting output

1. The table among `technical`, `commercial`, `leadership` which contains the employee's ID gives its main occupation.
2. We consult the `expertise` table. For a description of the expertise, we consult `expertisemap`.
3. We simply look at the `condition` field of the `project` table, whose description is in `conditionmap`.
4. We scan the `phase` table for a given project and count the times the `isvalid` field contains 'true'.
5. We find the last phase of the project looking at the `phase` table and we compare the `stop` field with the `datepaid` field of the `financial` table.
6. We scan the `financial` table for a given customer, and find whether the completion date (`stop` field) of the last phase in the project (table `phase`, accessed through `project`) corresponds with the `datepaid` field of the `financial` table.
7. We sum the costs of all the non-invalidated phases in the project and compare it to the total project cost (`amount` field in `financial` table).
8. The function changes every time a phase is invalidated or created. The cost is the cumulative cost of all the phases which are valid at any given time.
9. Since we only know the costs due to human resources, we must find the salaries of all the people involved in the project and scale them by the percentage of their time they devoted to the project. In other words, we must sum the scaled salaries over all phases of the project, over all teams involved in the phase, and over all people associated to the team.
10. We find the people involved in the project through phases and teams, and we compute their expertise level vector.
11. Similar to the above.
12. We match the teams involved in a project with indicators such as the project's condition and the number of invalidated phases (the fewer, the better).
13. We find the subsets of people from a team which occur most often in the most unsuccessful projects.

3.2 Brainstorming meeting

The commercial offer quotes: "Given some meaningful key-words or other well-defined indicators in the description of a new project, we want to classify it by some quantitative indices [...]". Such concepts as "meaningful key-words or other well-defined indicators" and "quantitative indices" are not well-defined, and therefore pose the most difficult problem to be solved in order to arrive at a software architecture. In order to solve the problem, a brainstorming meeting is called.

Aim of the meeting:

1. find a set of well-defined new project indicators which are suitable for searching similar terms in the T-Sale database;
2. find a set of quantitative indices to be computed using the T-Sale database information, which should shed light on the future life cycle of the new project;
3. document all ideas spawned during the meeting in a formal document.

Length: 1h.

3.2.1 Meeting output

Here is one possible approach to solving these problems:

1. find all project indicators which are known at the initial stage (details of first phase, customer history, personal compatibilities in teams);
2. propose a sizable number of quantitative indices that can be associated to a project (initial projected cost, cost curve, required skill levels, total human resources cost, number of teams, number of people, total cost, ...);
3. cluster all projects with similar degree of success (i.e. look at the `condition` field and at the number of invalidated phases) and produce a partition of the set of projects such that all projects in a partition subset have the same degree of success;
4. the most meaningful quantitative indices in the proposed set are those having the least variance in each partition subset;
5. finding the variance of the project indicators in the partition subsets will give an idea of the indicator reliability.

3.3 Functional architecture

Propose a functional architecture for the software. This should include the main software components and their interconnections, as well as a break-down of the architecture into sub-parts so that development teams can be formed and assigned to each project part. Since system-wide faults arise from badly interacting teams, it is naturally wise to minimize the amount of team interaction needed.

3.3.1 Solution

The only available point of departure for this analysis is the sketched architecture design contained in our commercial offer, which at this point should be used and expanded into a detailed and fully implementable software architecture. The following components are apparent:

1. **Input web form** (IWF): user inputs early indicator values concerning a new project
2. **Output web form** (OWF): user sees similar projects with relevant indicator values
3. **Clustering engine** (CE): given a set of objects and their pairwise distances, perform a clustering minimizing the inter-cluster distances
4. **Clustering significance evaluator** (CSE): Given a clustering, does it match well to another given clustering?
5. **Classification** (CLS): given an indicator for a given type of clustering, find the cluster it belongs to in the given and all similar clusterings
6. **Customer's DB**: split in *Commercial* (CDB), *Human resources* (HRDB), *Technical* (TDB) data repositories
7. **Clustering DB** (CLDB): repository for existing clusterings.

Fig. 3.2 shows a high-level activity diagram based on this modularization. Vertices are either logic anchors (black), actions (yellow), important data (green) and databases (blue). Arcs denote logic flow (black) or data flow (blue).

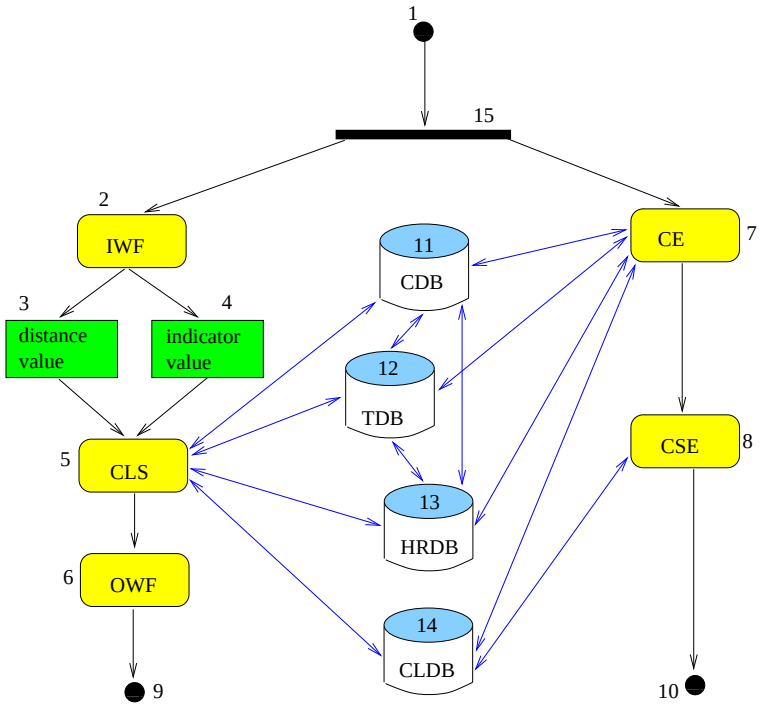


Figure 3.2: Initial activity diagram.

3.3.1.1 Interfacing

We apply the *interfacing* operator to this graph on the blue arc color (data flow arcs, coded by the label 2 in the AMPL data file). The AMPL model file is as follows.

```
# interface.mod
# AMPL model for interface creation

# graph
param n >= 1, integer;
set V := 1..n;
set E within {V,V};

# edge weights
param c{E};

# edge inclusions
param I{E};

# vertex colours
param lambda{V};

# arc colours
param kmax default 10;
param k <= kmax, >= 0, integer, default 1;
param mu{E} >=0, integer, <= kmax;

# variables
var x{V} binary;
var y{(u,v) in E} >= 0, <= min(max(0, mu[u,v]-k+1), max(0,k-mu[u,v]+1));

# model

maximize densesubgraph : sum{(u,v) in E} I[u,v] * c[u,v] * y[u,v] -
                 sum{v in V} x[v];

# linearization constraints
subject to lin1 {(u,v) in E} : y[u,v] <= x[u];
subject to lin2 {(u,v) in E} : y[u,v] <= x[v];
```

```
subject to lin3 {(u,v) in E} : y[u,v] >= x[u] + x[v] - 1;
```

The AMPL data file is as follows.

```
# activity1.dat
# AMPL dat file from UML activity diagram 1

param n := 15;
param : E :   c I mu :=
       1  15  1 1 1
       2  15  1 1 1
       2   3  1 1 1
       2   4  1 1 1
       3   5  1 1 1
       4   5  1 1 1
       5   6  1 1 1
       5  11  1 1 2
       5  12  1 1 2
       5  13  1 1 2
       5  14  1 1 2
       6   9  1 1 1
       7   8  1 1 1
       7  11  1 1 2
       7  12  1 1 2
       7  13  1 1 2
       7  14  1 1 2
       7  15  1 1 1
       8  10  1 1 1
       8  14  1 1 2
       11 12  1 1 2
       11 13  1 1 2
       12 13  1 1 2
;

param lambda :=
1  1
2  2
3  3
4  3
5  2
6  2
7  2
8  2
9  1
10 1
11 4
12 4
13 4
14 4
15 1 ;
```

The AMPL run file is as follows.

```
# file interface.run
model interface.mod;
data activity1.dat;
let k := 2; # choose the colour
option solver cplexstudent;
solve;
display y;
display x;
```

We solve the problem by issuing the command `cat interface.run | ampl`. We find the interface subgraph $H = (U, F)$ where $U = \{5, 7, 11, 12, 13, 14\}$ and $F = \{\text{all blue arcs}\}$. We add a new vertex 16 representing the interface, remove the arcs F and add the (bidirected) arcs $F' = \{\{u, 16\} \mid u \in U\}$. Since 16 is a database interface, we assign it the database vertex colour (blue). The activity diagrams evolves into Fig. 3.3.

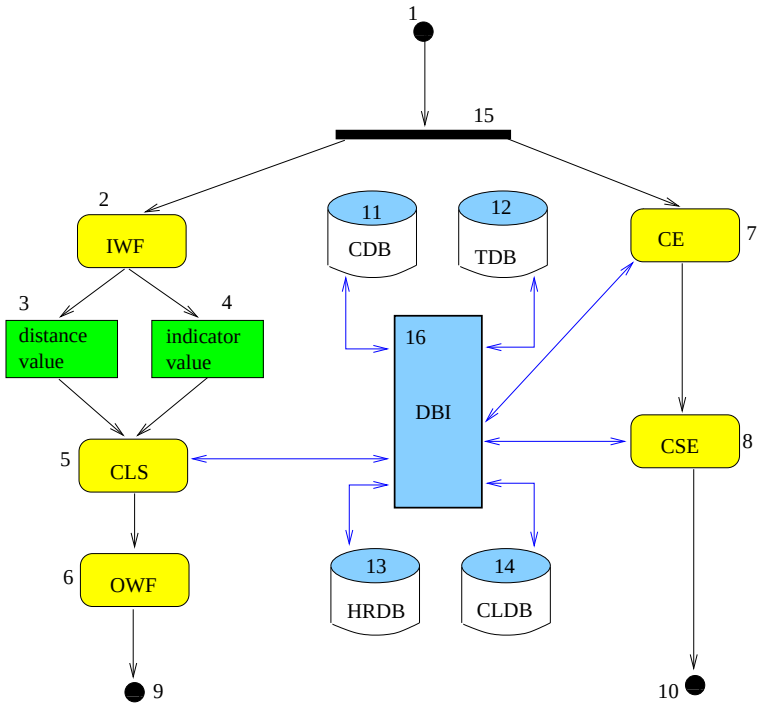


Figure 3.3: Activity diagram after interfacing.

3.3.1.2 Clustering

We now apply the *clustering* operator to the new activity diagram, to identify some clusters such that interconnections are minimized. Such clusters may help break down the architecture in logically disconnected parts; as system-wide faults usually emerge from inter-team lack of communication, assigning such parts to different teams will minimize the chances of ending up with system-wide faults.

The AMPL model is as follows.

```
# flexcolour_clustering.mod
# flexible coloured clustering (colours on vertices) - AMPL model

# graph
param n >= 1, integer;
set V := 1..n;
set E within {V,V};

# edge weights
param c{E};

# edge inclusions
param I{E};

# vertex colours
param lambda{V};
param gamma{u in V, v in V : u != v} :=
  if (lambda[u] = lambda[v]) then 0 else 1;

# arc colours
param mu{E};

# max number of clusters
param kmax default n;
set K := 1..kmax;

# original problem variables
var x{V,K} binary;
```

```
# linearization variables
var w{V,K,V,K} >= 0, <= 1;

# cluster existence variables
var z{K} binary;

# model
minimize intercluster :
  sum{k in K, l in K, (u,v) in E : k != l} I[u,v] * c[u,v] * w[u,k,v,l] +
  sum{k in K} z[k];

subject to assignment {v in V} : sum{k in K} x[v,k] = 1;

# use (ceil(card{V}/kmax)+1) as RHS for balanced multi-cluster cardinality
subject to cardinality {k in K} : sum{v in V} x[v,k] <= ceil(card{V}/2);

subject to existence {k in K} : sum{v in V} x[v,k] >= z[k];

subject to diffcolours {u in V, v in V, k in K, l in K : u != v and k != l} :
  w[u,k,v,l] <= gamma[u,v];

### linearization constraints
subject to lin1 {u in V, v in V, h in K, k in K : (u,v) in E or (v,u) in E} :
  w[u,h,v,k] <= x[u,h];
subject to lin2 {u in V, v in V, h in K, k in K : (u,v) in E or (v,u) in E} :
  w[u,h,v,k] <= x[v,k];
subject to lin3 {u in V, v in V, h in K, k in K : (u,v) in E or (v,u) in E} :
  w[u,h,v,k] >= x[u,h] + x[v,k] - 1;
```

The AMPL data file is as follows.

```
# activity2.dat
# AMPL dat file from UML activity diagram 2

param n := 16;
param : E :   c I mu:=
       1  15  1 1 1
       2  15  1 1 1
       2   3  1 1 1
       2   4  1 1 1
       3   5  1 1 1
       4   5  1 1 1
       5   6  1 1 1
       5  16  1 1 2
       6   9  1 1 1
       7   8  1 1 1
       7  16  1 1 2
       8  10  1 1 1
       8  16  1 1 2
       11 16  1 1 2
       12 16  1 1 2
       13 16  1 1 2
       14 16  1 1 2
;

param lambda :=
1  1
2  2
3  3
4  3
5  2
6  2
7  2
8  2
9  1
10 1
11 4
12 4
13 4
14 4
15 1
16 4 ;
```

The AMPL run file is as follows.

```
# file flexcolour_clustering.run
```

```
model flexcolour_clustering.mod;
data activity1.dat;
let kmax := 4; # maximum number of clusters
option solver cplexstudent;
solve;
display y;
display x;
```

We solve the problem by issuing the command `cat flexcolour_clustering.run | ampl`. We ask for at most 4 clusters (`let kmax := 4;`). We obtain two clusters: $C_1 = \{1, 2, 3, 4, 5, 6, 9, 15\}$ and $C_2 = \{7, 8, 10, 11, 12, 13, 14, 16\}$. The activity diagram is now as in Fig. 3.4.

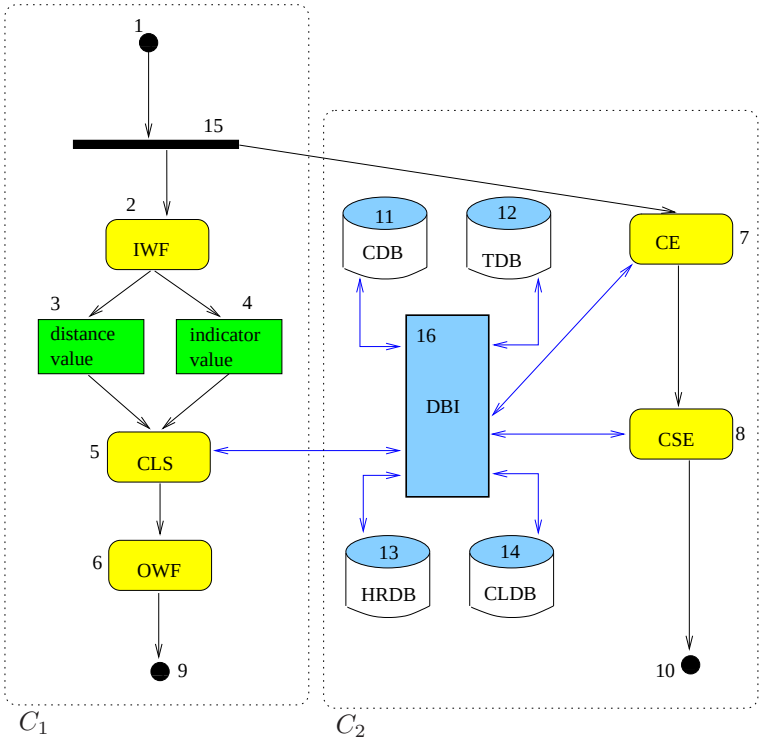


Figure 3.4: Activity diagram after clustering.

The architecture is composed of two main subsystems C_1, C_2, corresponding to two activity processes IWF→CLS→OWF (performed by the user) and CE↔CSE↔DBI (performed by the program) that we may call respectively the *foreground* and *background* processes. The foreground subsystem consists of three main components (input, classifier and output); the background subsystem consists of a database sub-subsystem (with an interface and four databases) and two main components (clustering engine and clustering significance evaluator). Very high-level specifications may now be given as follows:

1. IWF: input indicator(s) and clustering distance(s) from the user
2. CLS: classify new project according to given indicator(s) and distance(s) using a database of existing clusterings with cluster-matching information
3. OWF: output set of existing projects close to the new project w.r.t. given indicator(s)
4. CE: given a set of indicator values and an associated distance metric, cluster the values; pass the clustering to the DB interface for storage
5. CSE: given two clusterings, match them and verify their compatibility; pass the matching information to the DB interface for storage

6. DBI: interface to customer and clustering DBs.

The two processes (corresponding to C_1, C_2) are linked by arcs $(15, 7)$ (a logical flow arc) and $(5, 16)$ (a data flow arc). The logical path choices $(1, 15, 2)$ and $(1, 15, 7)$ identify the foreground and background processes respectively. If we consider two separate starting points for the two processes we can eliminate vertex 15 and all its adjacent arcs (including $(15, 7)$). We then introduce a starting vertex (labelled 15, since the old vertex 15 was reformulated out of the graph) for the background process (see Fig. 3.5). The data flow arc $(5, 16)$ is crucial to the process interplay and cannot be eliminated. It actually gives

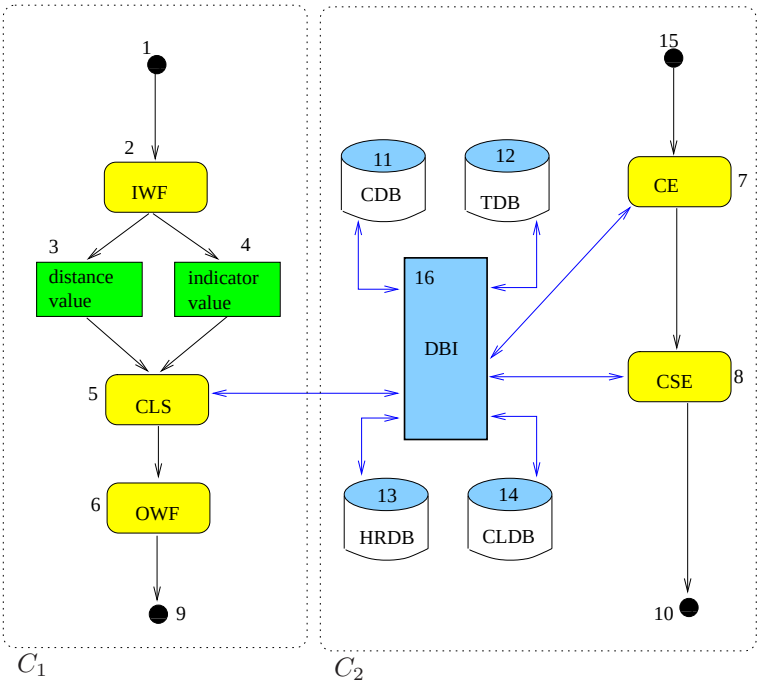


Figure 3.5: Final activity diagram: separating the processes.

the extent and the type of interconnection between the processes. It also suggests where the two teams developing the different process will need to interact, namely in the design of the database interface (DBI, vertex 16): more precisely, the background process team will need to explain to the other team what data is made available by the interface, and the foreground process team will need to require the appropriate data exchange formats and protocols.

The precise breakdown of each component into classes and methods is part of the technical architecture.

3.4 Technical architecture

Propose a technical architecture detailing the inner working of each system component, as well as the system as a whole. This should include a class diagram and component APIs (application programming interfaces).

3.4.1 Solution

In order to build a class diagram and the APIs, we need to know how input data are transformed into the output data, and exactly which data is passed from one component to another. As the background process is in some way a server to the foreground one, we shall model the latter first (top-down approach).

Informally, the data flow for the foreground process is as follows:

$$\begin{aligned}
\textbf{input} &\xrightarrow{\text{IWF}} \text{(early indicator, indicator value, distance value)} \xrightarrow{\text{CLDB}} \\
&\rightarrow \text{(corresponding early clustering, cluster)} \xrightarrow{\text{CLS}} \\
&\rightarrow \text{(matching late clustering(s), cluster(s))} \xrightarrow{\text{OWF}} \\
&\rightarrow \textbf{output} \text{ projects in identified cluster(s).}
\end{aligned}$$

In order for the foreground process data flow to make any sense, the CLDB database must contain all the clusterings relative to the given early indicator value and distance, and the CLS component must be able to match early and late clusterings (and to draw the appropriate "close" clusters from the matched clusterings). The background process must therefore supply the necessary information. Recall that the foreground process is ran by each user, and so should be as fast as possible. It is therefore necessary to delegate most of the computational work to the background process: all data transformation should draw from information that was pre-computed by the background process. In particular, finding a matching late clustering should be as simple as looking up a pre-computed boolean value in an array; in turn, this means that the background process must pre-compute all possible matching information and store it in the CLDB database.

Informally, the data flow for the background process is as follows:

$$\begin{aligned}
\textbf{start} &\rightarrow \text{all possible pairs } ((\text{early indicator}, \text{distance}), (\text{late indicator}, \text{distance})) \xrightarrow{\text{CE}} \\
&\rightarrow \text{(clustering)} \xrightarrow{\text{DBI}} \\
&\rightarrow \textbf{store} \text{ (clustering)} \xrightarrow{\text{CSE}} \\
&\rightarrow \text{(do clustering match?, list of matching clusters)} \xrightarrow{\text{DBI}} \\
&\rightarrow \textbf{store} \text{ (matching info)} \rightarrow \textbf{stop}.
\end{aligned}$$

To make the data flow descriptions more formal, we must make clear what we mean precisely by such concepts as indicator, distance, cluster, clustering, clustering comparison.

3.4.1.1 Indicators

An *indicator* is a non-negative real-valued function $v : P \rightarrow \mathbb{R}_+$ defined on the set of projects P. Given an indicator v, we let:

$$\begin{aligned}
\bar{v} &= \max_{p \in P} v(p) \\
\underline{v} &= \min_{p \in P} v(p).
\end{aligned}$$

Early indicators are indicators whose value can be defined before the project is started; *late* indicators may only be defined after the project ends. Consider early indicators v_i^E for $i \leq m$ and late indicators v_j^L for $j \leq n$. For each early indicator $i \leq m$ we also consider finite sets of distances[1] D_i^E with (and likewise for late indicators).

[1] By *distance* we mean here a generic measure of similarity, without implying the triangular inequality.

3.4.1.2 Clusterings

Given an indicator v on P and a distance value $d \in \mathbb{R}_+$, a *clustering* γ_{vd} of P is a set of $K_{vd} = |\gamma_{vd}|$ subsets γ_{vdk} of P, where $k \leq K_{vd}$ and

$$K_{vd} = \left\lceil \frac{\bar{v}_i - \underline{v}_i}{d} \right\rceil,$$

such that:

(a) $\forall p \in P \; \exists k \leq K_{vd} \; (p \in \gamma_{vdk})$ (covering condition).

(b) $\forall k \leq K_{vd} \; \forall p \in \gamma_{vdk} \; (\underline{v}_i + kd \leq v(p) \leq \underline{v}_i + (k+1)d)$ (cluster extent).

Notice we define clusterings so that γ_{vd} is unique for each choice of v, d and can be computed in $O(|P|)$. This is not the only possible such definitions. Other definitions allow for non-uniqueness and for higher computational complexity orders.

Each subset γ_{vdk} of a clustering is called a *cluster*; because of (b), we can assign to each cluster γ_{vdk} an interval $I_{vdk} = [\underline{v}_i + kd, \underline{v}_i + (k+1)d]$. Let γ^E_{id} be the clustering of P corresponding to the early indicator v^E_i and distance $d \in D^E_i$, with $K_{id} = |\gamma^E_{id}|$; let γ^E_{idk} be the k-th cluster of γ^E_{id} for $k \leq K_{id}$ (and likewise for late indicators). Fig. 3.6 shows an example of a clustering where $P = \{1, \ldots, 10\}, \underline{v} = 0, \bar{v} = 3, d = 1$.

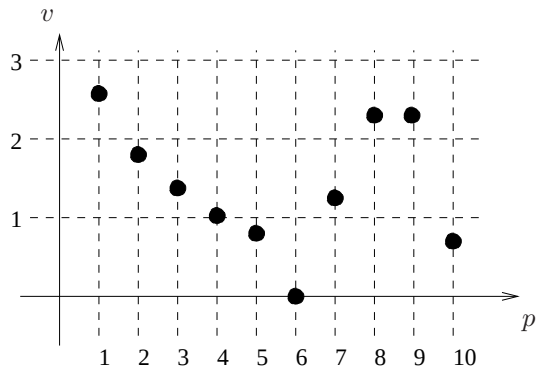


Figure 3.6: Clustering example. We obtain three clusters $\gamma_{vd1} = \{4, 5, 6, 10\}$ with $I_{vd1} = [0, 1]$, $\gamma_{vd2} = \{2, 3, 4, 7\}$ with $I_{vd2} = [1, 2]$ and $\gamma_{vd3} = \{1, 8, 9\}$ with $I_{vd3} = [2, 3]$.

3.4.1.3 Clustering comparison

Our software relies on our ability to succesfully compare early and late clusterings and say if they match or not. Given two indicators u, v and distances d, δ with relative clusterings γ_{ud} and $\gamma_{v\delta}$, we first scale the indicator values and distances so that they are comparable. This can be easily done by scaling the two clustering intervals I_{ud} and $I_{v\delta}$ to the interval $[0, 1]$: for all $p \in P$ let

$$\begin{aligned}
\tilde{u}(p) &= \frac{u(p) - \underline{u}(p)}{\bar{u}(p) - \underline{u}(p)} \\
\tilde{v}(p) &= \frac{v(p) - \underline{v}(p)}{\bar{v}(p) - \underline{v}(p)} \\
\tilde{d} &= \frac{d - \underline{u}(p)}{\bar{u}(p) - \underline{u}(p)} \\
\tilde{\delta} &= \frac{\delta - \underline{v}(p)}{\bar{v}(p) - \underline{v}(p)}.
\end{aligned}$$

We define the dissimilarity between the two clusterings $\gamma_{ud}, \gamma_{v\delta}$ as:

$$\Delta(\gamma_{ud}, \gamma_{v\delta}) = (\tilde{d} - \tilde{\delta})^2 + \sum_{p \in P} (\tilde{u}(p) - \tilde{v}(p))^2.$$

Notice this definition does not actually consider the clustering itself, but just the indicator and the distance: this occurs because of the way our clusterings are defined. More precisely, this occurs because the cluster each $p \in P$ belongs to is determined by p alone and not by the other elements of P.

Given an overall tolerance $\varepsilon > 0$, an early indicator clustering γ^E_{id} (where $i \leq m$ and $d \in D^E_i$) *matches* a late indicator clustering $\gamma^L_{j\delta}$ (where $j \leq n$ and $\delta \in D^L_j$) if either one of the two conditions below is satisfied:

1. $\Delta(\gamma^E_{id}, \gamma^L_{j\delta}) \leq \varepsilon$;
2. $\Delta(\gamma^E_{id}, \gamma^L_{j\delta}) = \min_{h \leq n, b \in D^L_h} \Delta(\gamma^E_{id}, \gamma^L_{hb})$;

we denote the matching by $M(\gamma^E_{id}, \gamma^L_{j\delta}) = 1$ and a mismatch by $M(\gamma^E_{id}, \gamma^L_{j\delta}) = 0$. If two matching clusterings satisfy the first condition, it is a close match. The second condition is a "catch-all" condition which ensures that we can match each early indicator clustering to at least one late indicator clustering.

Given two matching clusterings $\gamma_{ud}, \gamma_{v\delta}$, we must now find indices $h \leq K_{ud}$ and $k \leq K_{v\delta}$ such that γ_{udh} and $\gamma_{v\delta k}$ are "as close as possible". We extend the dissimilarity definition Δ to clusters as follows:

$$\Delta(\gamma_{udh}, \gamma_{v\delta k}) = (\tilde{d} - \tilde{\delta})^2 + \sum_{p \in \gamma_{udh} \cap \gamma_{v\delta k}} (\tilde{u}(p) - \tilde{v}(p))^2 + |\gamma_{udh} \triangle \gamma_{v\delta k}|,$$

where $A \triangle B = (A \cup B) \setminus (A \cap B)$ is the symmetric difference of two sets A, B. This definition is justified by the fact that the difference in normalized indicator value for a project p in $\gamma_{udh} \triangle \gamma_{v\delta k}$ is simply the diameter of the corresponding normalized interval $[0, 1]$, namely 1, and that $1^2 = 1$. With this extended definition, we can compute $\Delta(\gamma_{udh}, \gamma_{v\delta k})$ for each possible pair (h, k) and determine a pair of closest clusters. We denote the set of clusters $\gamma_{v\delta k}$ in $\gamma_{v\delta}$ closest to a given cluster γ_{udh} by $\Gamma(\gamma_{udh}, \gamma_{v\delta})$.

3.4.1.4 The foreground process

The data transformation model of the foreground process is as follows: we are given a new project π; we select an early indicator v^E_i, compute $w = v^E_i(\pi)$, select a meaningful distance $d \in D^E_i$, find the corresponding clustering γ^E_{id} and the cluster γ^E_{idk} such that $w \in I_{idk}$. We then find a late indicator clustering $\gamma^L_{j\delta}$ (where $j \leq n$ and $\delta \in D^L_j$) such that $M(\gamma^E_{id}, \gamma^L_{j\delta}) = 1$, and the corresponding closest clusters $\gamma^L_{j\delta h} \in \Gamma(\gamma^E_{idk}, \gamma^L_{j\delta})$. The formal data flow description of the foreground process is:

$$\begin{aligned}
\textbf{input} \quad & \overset{\text{IWF}}{\rightarrow} \quad (\pi, v^E_i, d) \overset{\text{CLDB}}{\longrightarrow} (\gamma^E_{id}, \quad \gamma^E_{idk} : v^E_i(\pi) \in I_{idk}) \overset{\text{CLS}}{\longrightarrow} \\
& \rightarrow \quad O = \{(j, \delta, h) \mid M(\gamma^E_{id}, \gamma^L_{j\delta}) = 1 \wedge \gamma^L_{j\delta h} \in \Gamma(\gamma^E_{idk}, \gamma^L_{j\delta}))\} \overset{\text{OWF}}{\longrightarrow} \\
& \rightarrow \quad \forall (j, \delta, h) \in O \ \textbf{output} \text{ projects in } \gamma^L_{j\delta h}
\end{aligned}$$

The required data structures are:

- project (p, class): contains project attributes as defined in the T-Sale DB;
- cluster (list of projects);
- clustering (γ: list of clusters);

- indicator (v, class): contains
  - methods to retrieve the indicator value given a project
  - list of clustering distances D (floating point numbers)
  - extremal values $\bar{v}, \underline{v}$ (floating point numbers)
  - list of clusterings γ_{vd} for this indicator, relative to all distances $d \in D$
  - methods to scale the indicator values and distances in D to the interval $[0, 1]$
- foreground process (class): contains
  - list of early indicators $(v_i^E \mid i \leq m)$;
  - list of late indicators $(v_j^L \mid j \leq n)$;
  - matching information (M, array of booleans indexed on $i \leq m, d \in D_i^E, j \leq n, \delta \in D_j^L$);
  - matching cluster information (Γ, maps clusters γ_{idk} for varying $k \leq K_{id}$ to list of matching clusters $(\gamma_{j\delta h})$ for varying $h \in \{1, \ldots, K_{j\delta}\}$).

3.4.1.5 The background process

The data transformation model of the background process is as follows: given an early indicator v_i^E $(i \leq m)$, a distance $d \in D_i^E$, a late indicator v_j^L $(j \leq n)$ and a distance $\delta \in D_j^L$:

- if γ_{id}^E is present in the CLDB database retrieve it, else compute it and store it;
- if $\gamma_{j\delta}^L$ is present in the CLDB database retrieve it, else compute it and store it.

Determine $M(\gamma_{id}^E, \gamma_{j\delta}^L)$ and store it in the CLDB database; if $M(\gamma_{id}^E, \gamma_{j\delta}^L) = 1$, for each $k \in K_{id}$ compute the set $\Gamma(\gamma_{idk}^E, \gamma_{j\delta}^L)$ and store it in the CLDB database. The formal data flow description of the background process is:

$$\begin{aligned}
\textbf{start} &\rightarrow \{((v_i^E, d), (v_j^L, \delta)) \mid i \leq m \wedge d \in D_i^E \wedge j \leq n \wedge \delta \in D_j^L\} \xrightarrow{\text{CE}} \\
&\rightarrow \mathcal{C} = \{(\gamma_{id}^E, \gamma_{j\delta}^L) \mid i \leq m \wedge d \in D_i^E \wedge j \leq n \wedge \delta \in D_j^L\} \xrightarrow{\text{DBI}} \\
&\rightarrow \textbf{store } \mathcal{C} \xrightarrow{\text{CSE}} \\
&\rightarrow \mathcal{M} = ((M(c) \mid c \in \mathcal{C}), (\Gamma(\gamma_{idk}^E, \gamma_{j\delta}^L) \mid (\gamma_{id}^E, \gamma_{j\delta}^L) \in \mathcal{C}, k \in K_{id})) \xrightarrow{\text{DBI}} \\
&\rightarrow \textbf{store } \mathcal{M} \rightarrow \textbf{stop}.
\end{aligned}$$

The required data structures are all those listed in Section 3.4.1.4 aside from the foreground process class, plus a background process class containing:

- list of early indicators $(v_i^E \mid i \leq m)$
- list of late indicators $(v_j^L \mid j \leq n)$
- matching information (M, array of booleans indexed on $i \leq m, d \in D_i^E, j \leq n, \delta \in D_j^L$)
- matching cluster information (Γ, maps clusters γ_{idk} for varying $k \leq K_{id}$ to list of matching clusters $(\gamma_{j\delta h})$ for varying $h \in \{1, \ldots, K_{j\delta}\}$)
- methods for computing Δ applied to clusterings
- methods for computing intersections of clusters
- methods for computing symmetric differences of clusters
- methods for computing Δ applied to clusters.

3.4.1.6 Class structure

The class structure is detailed in Fig. 3.7.

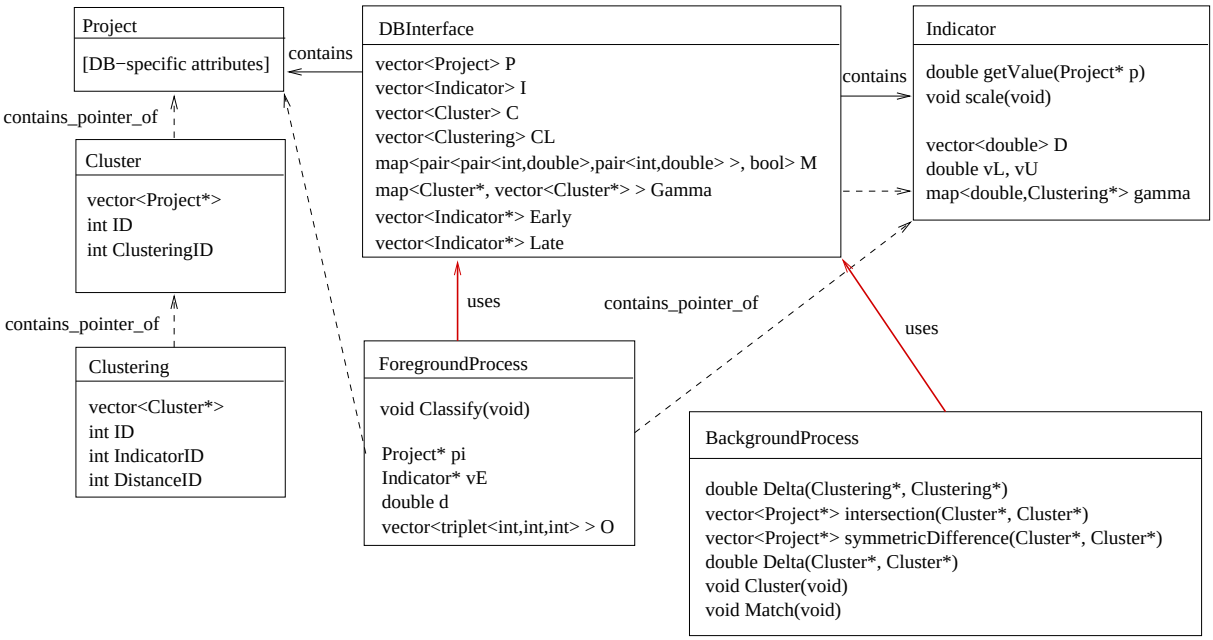


Figure 3.7: The class diagram of the fore- and background processes.