Algorithm for intelligent prediction of requests in business systems

Igor Podolak, Adam Roman, Piotr Kalita, Bartosz Bierkowski
Institute of Computer Science
Jagiellonian University, Poland

WWW.SOFSEM.SK
=:SOFSEM 08:=
High Tatras, Slovakia
January 19-25, 2008

Agenda

- Aims and motivation – reference to ASK-IT
- Overview of system architecture
- Data Structure (RPG Graph)
- Example
- Algorithms (PredictRequest & UpdateGraph)
- Tests

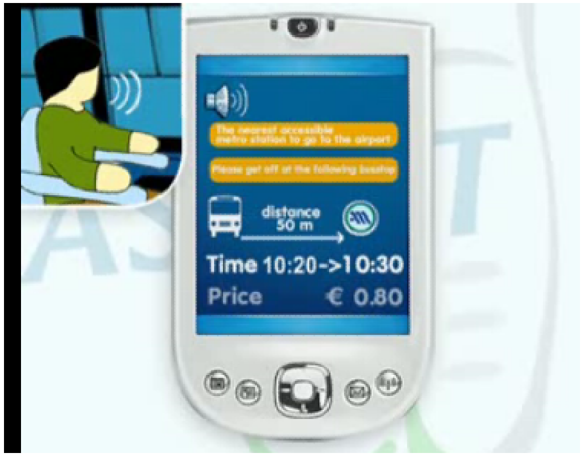
Application

Ambient Intelligence
System of Agents for
Knowledge-based and
Integrated Services for Mobility Impaired users
(IST-2003-511298 6 Framework Project)

User request prediction: WorkPackage 3.4, Activity 3.5.4

European Commission

Mobility Impaired person equipped with PDA



Access to various services (for instance route planning)

E – learning services

Social Events

Bus Services

Navigation Services

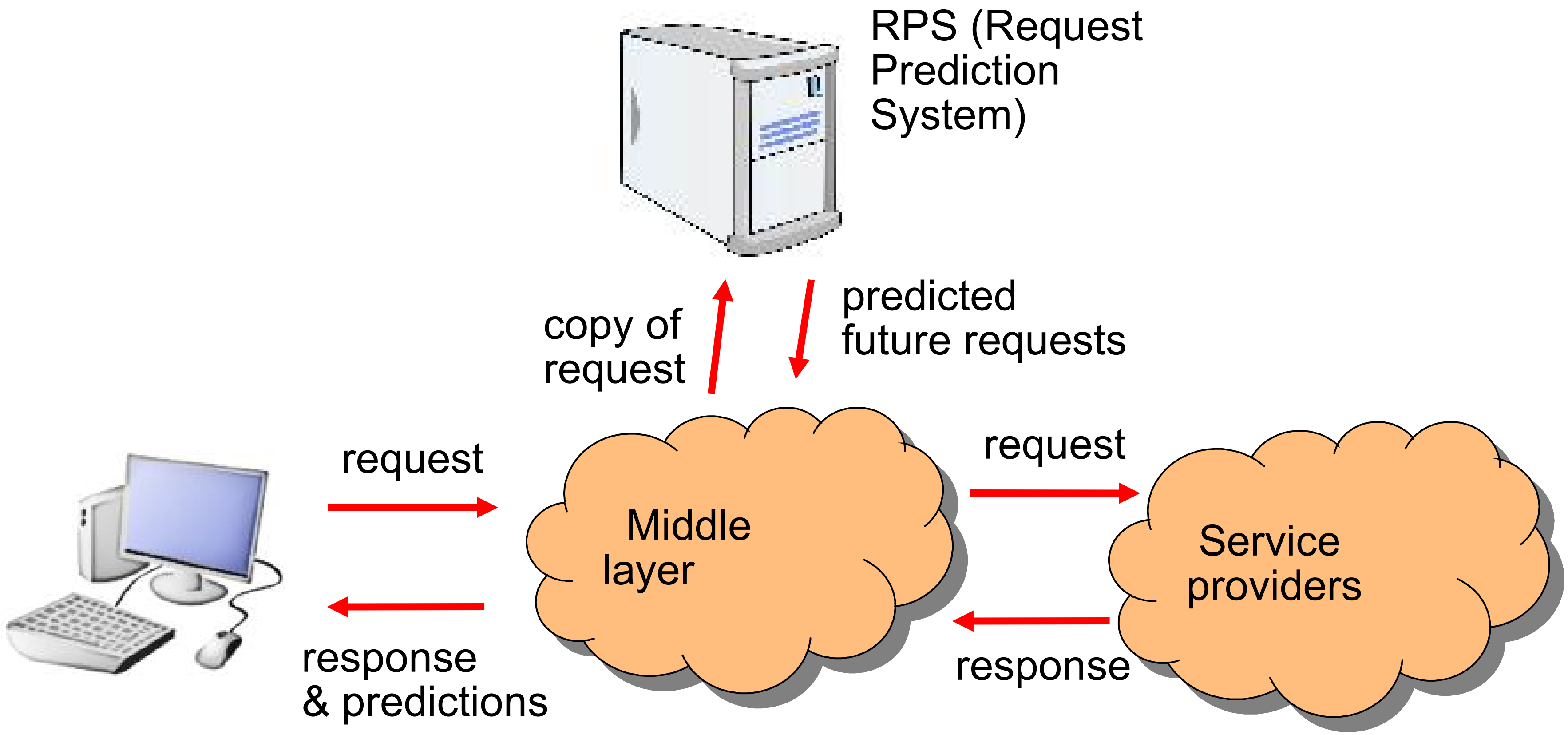
Traffic Events

5 services, together about 20 operations (e.g. FindPOI, FindRoute ....)

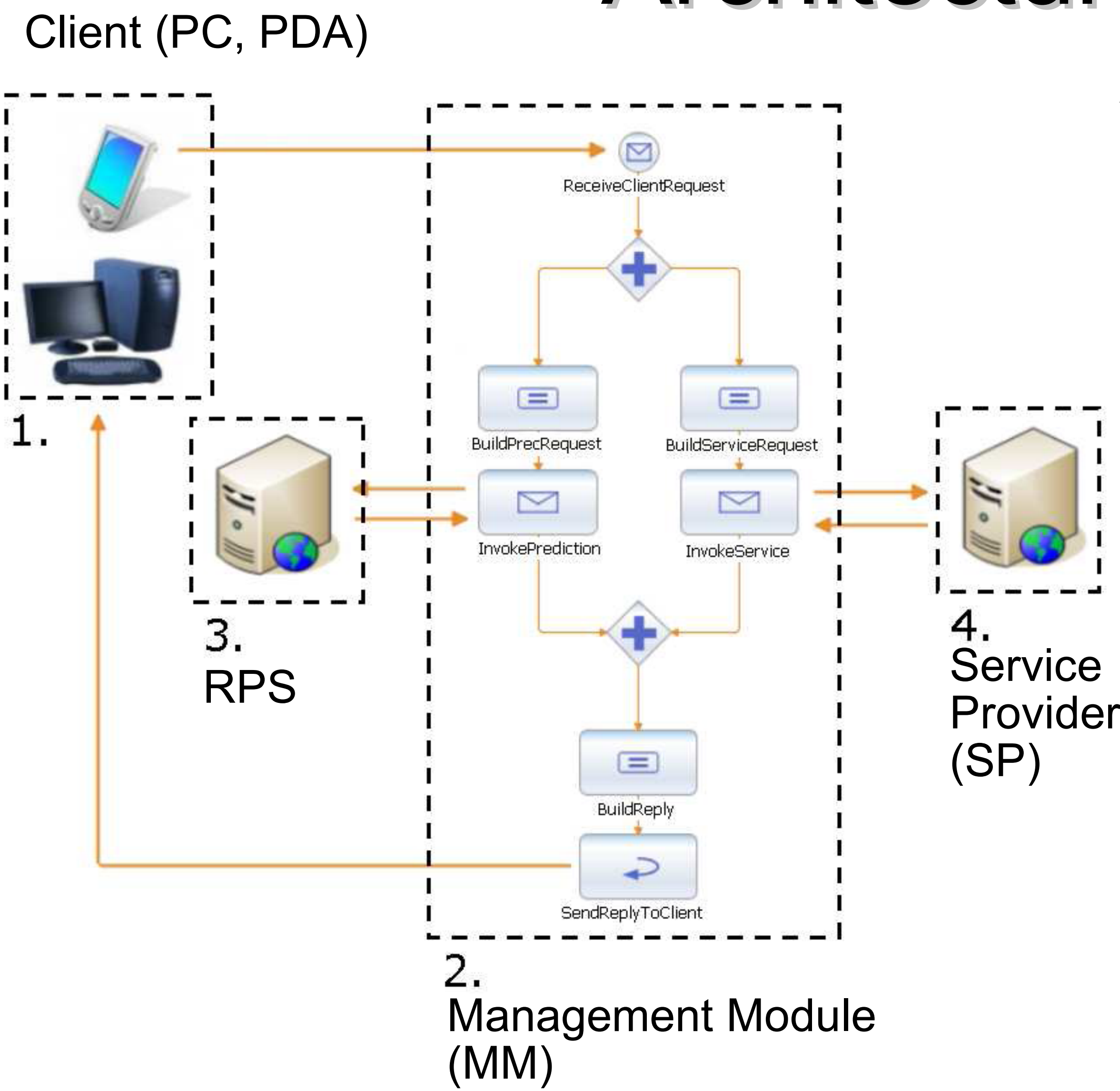
Purposes of prediction

- To accelerate the response time
  - execution in idle time
  - intelligent caching
- To suggest the user possible service requests
  - RPG graph structure included in RPS system reflects the popularity and mutual dependencies between requests
- To aid other modules of the system
  - Ranking of services
  - Enables user requests data mining

The idea of request prediction



Architecture



A)Client system (1) issues a service request to MM (2).

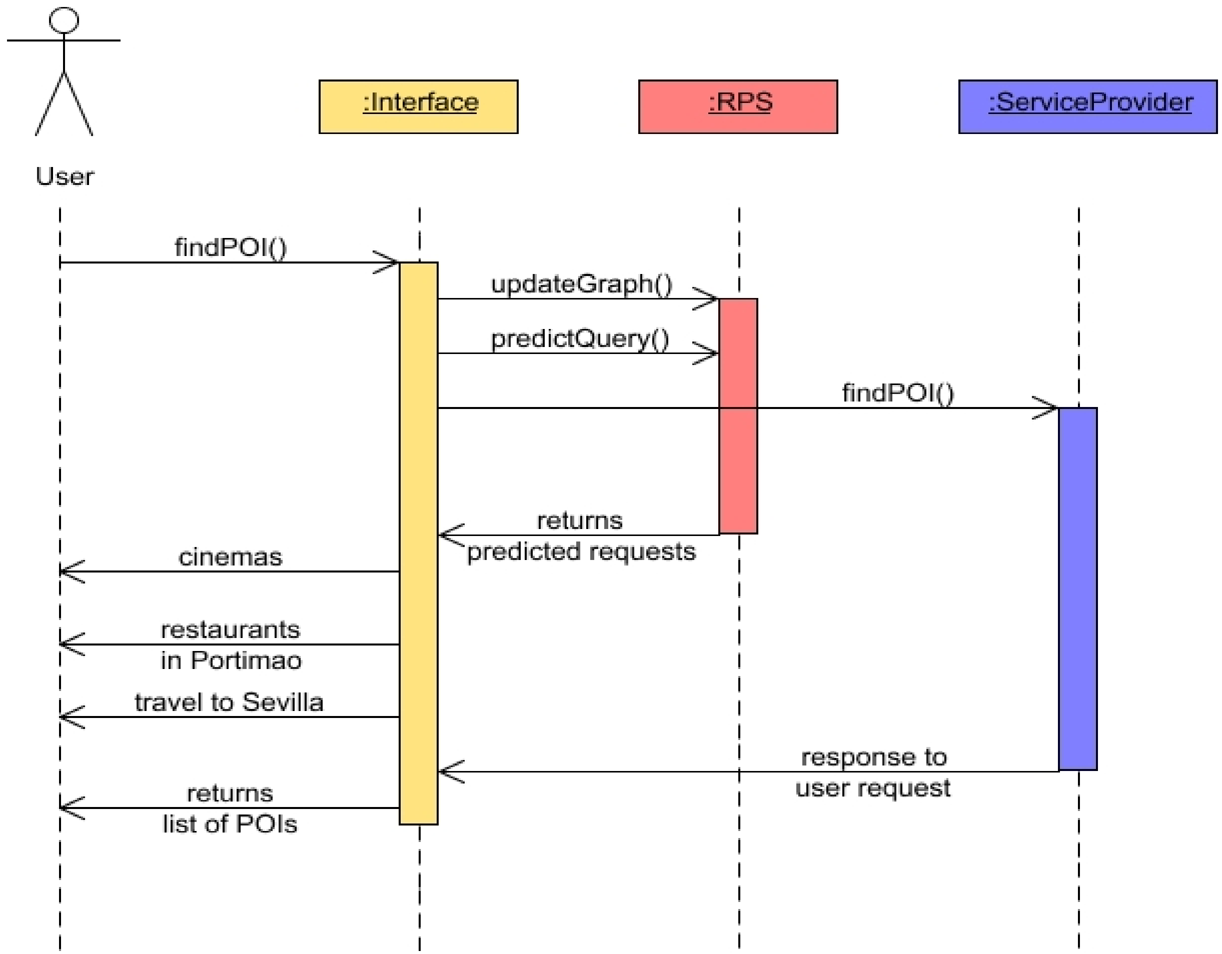
B)MM associates one of SP (both RPS (3) and SP (4) are available to MM as Web Services).

C)The request is issued by MM to SP and in parallel to RPS.

D)The reply arrives from SP and predictions from RPS.

E)MM gathers replies and sends them to the client.

User's view (prediction)



Data structure inside RPS (Request Prediction Graph RPG)

Request Prediction Graph (RCG) is a 6-tuple

$$RCG = (G, cons, popm, parw, kind, ord),$$

nodes

where:

vertices

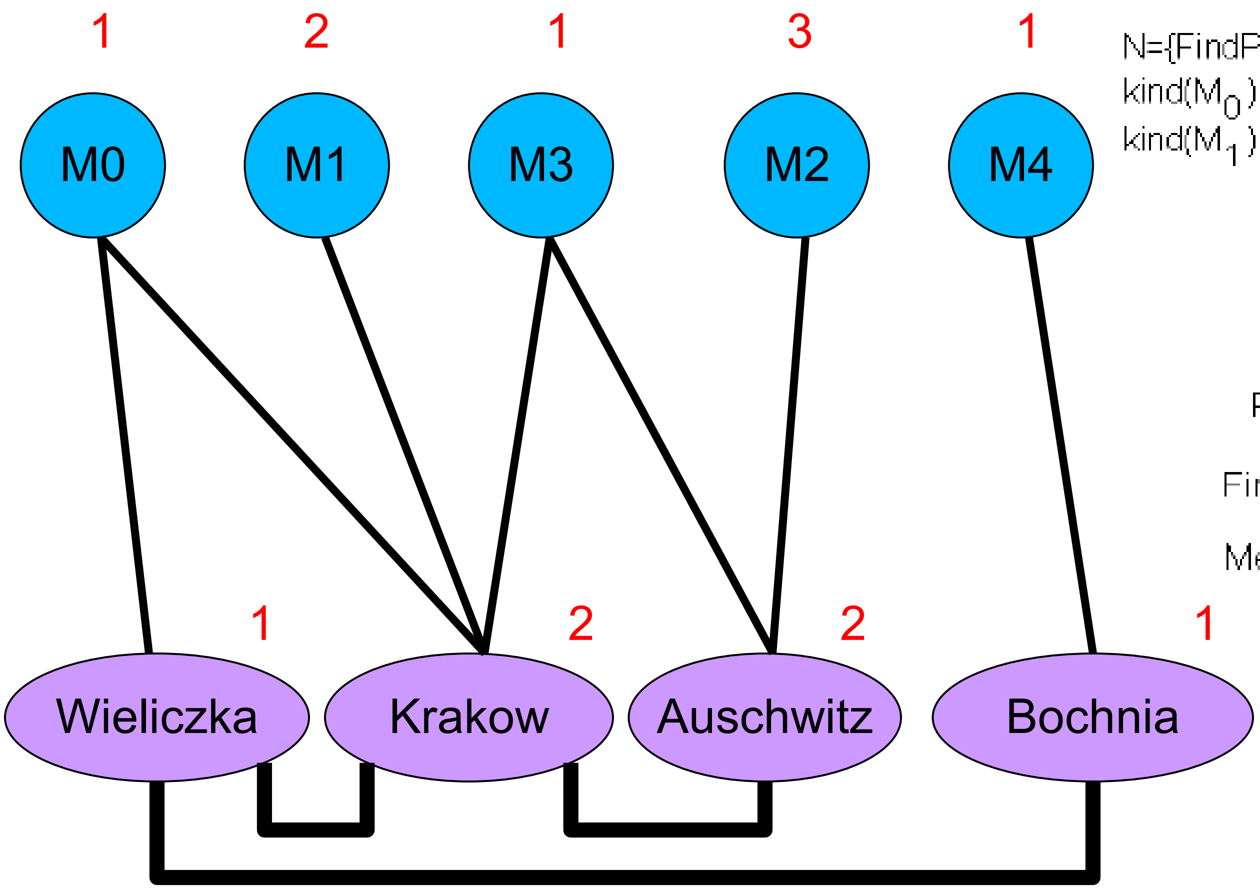
1. $G = (MET \cup PAR, WM \cup WP)$, $MET \cap PAR = \emptyset$, $WM \cap WP = \emptyset$;
2. $WM \subseteq MET \times PAR$, $WP \subseteq \binom{PAR}{2}$;
3. $cons: \ N^2 \to \mathbb{Q}$;
4. $popm: \ MET \to \mathbb{Q}$;
5. $parw: \ PAR \to \mathbb{Q}$;
6. $kind: \ MET \to N$;
7. $ord: \ MET \times PAR \to \mathbb{N}$;
8. $\forall m \in MET \ \{ord(m,p): \ (m,p) \in WM\} = \{1, 2, ..., deg(m)\}$.

Three types of weights that are used in prediction process

Each method has the associated name of operation

Parameter ordering

RPG - example



N={FindPOI, FindRoute}   MET={$M_0 \dots M_4$}
kind(M_0)=kind(M_3)=FindRoute
kind(M_1)=kind(M_2)=kind(M_4)=FindPOI

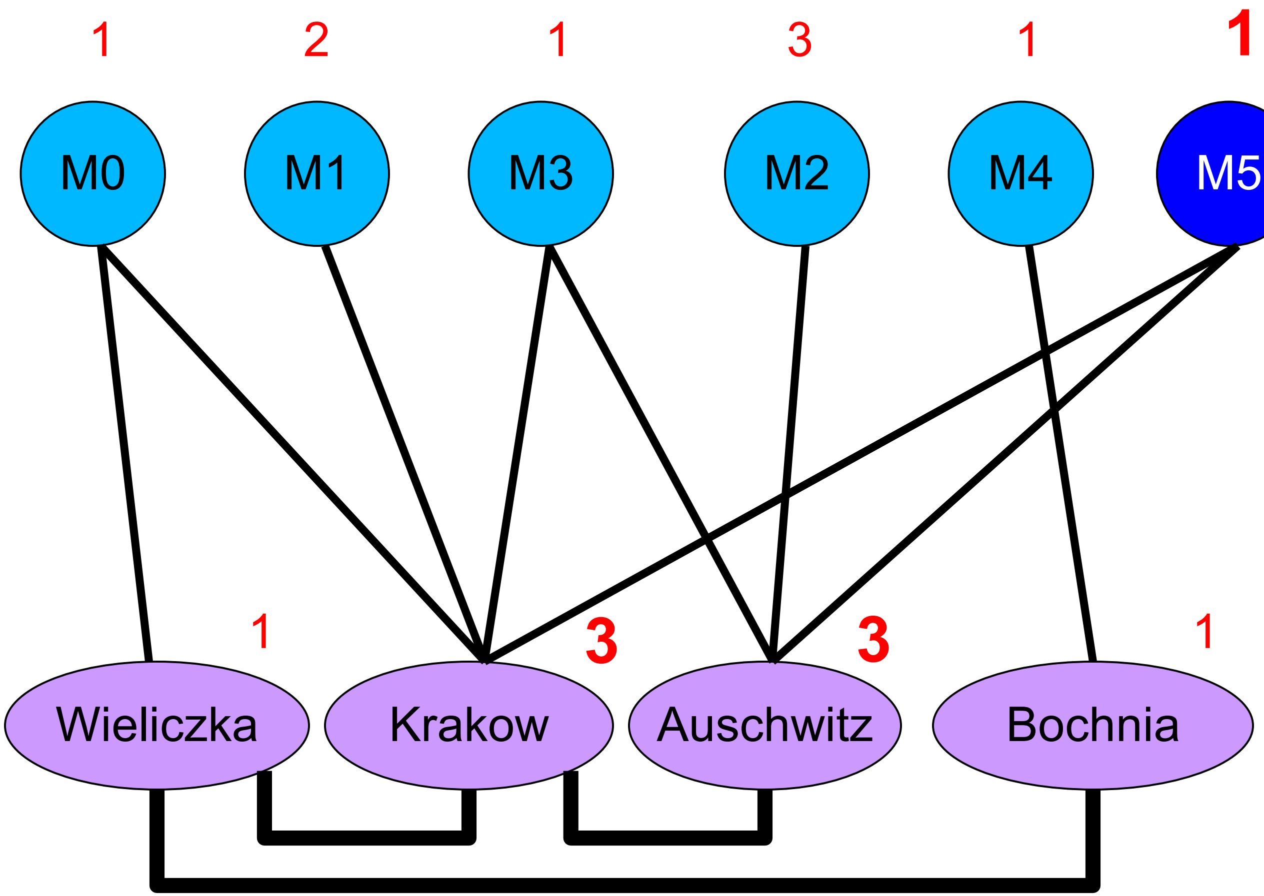
	FindPOI	FindRoute
FindPOI	1	2
FindRoute	5	2

Method Consequence Matrix

M nodes represent the INSTANCES of operation invocations

no personal data are stored in the graph

What is inside RPS? (2)



MET={$M_0 \ldots M_5$}

$$\begin{array}{c|cc} & \text{FindPOI} & \text{FindRoute} \\ \hline \text{FindPOI} & 1 & 2 \\ \text{FindRoute} & 5 & \mathbf{3} \end{array}$$

Method Consequence Matrix

New request M5 is issued.

Predicted requests are:

M2 (rank value=45)

M1 (rank value=30)

M3,M0 (rank value=9)

How are invocations ranked?

- Neighbourhood of current invocation in RPG graph is considered
- 3 types of weights:
  - w1=„popularity” of operations instances
  - w2=„popularity” of entities (parameters) in operation invocations
  - w3=method invocation consequence
- Ranking formula (heuristic)
  - rank=w1*mean(w2)*w3

Formalism (UpdateGraph algorithm I)

```
algorithm UpdateGraph(R=(M,P),L)
Input: Current request R (method M and parameters list P) and a previous request L submitted
by the same user

begin
foreach p ∈ P do // update the set of parameters
        if p ∉ PAR then PAR ←PAR ∩ {p};
t ← 0;
foreach p ∈ P do // update weights in the parameters layer
        parw(p) ← parw(p)+1;
        if parw(p) > threshold_parw then t ← 1;
if t=1 then ModifyParW();
Y ← P;
y ← take_from(Y); //take first element from the list and remove it
// X_1 is the set of methods for which y is the first parameter
X_1 ← {m ∈ MET: kind(m)=M ∧ (m,y) ∈ WM ∧ ord(m,y)=1};
i ← 1;
if Y ≠ ∅ then
        repeat
                y ← take_from(Y);
                i ← i + 1;
                // X_i is the set of methods for which y is the i-th parameter
                X_i ← X_i-1 ∩ {m ∈ MET: kind(m)=M ∧ (m,y) ∈ WM ∧ ord(m,y)=i};
        until X_i=∅ ∨ Y=∅;
```

New parameters are added if necessary

Parameter weights are increased, if they are too large graph is rescaled

Find vertex in MET that represents the call

Formalism (UpdateGraph algorithm II)

```
// Xi is the set of all methods with parameters set P and with the same order as in method M. If
// Xi=∅ then add a vertex representing instance of M.
if Xi = ∅ then // notice that |Xi|=0 or |Xi|=1
        MET ← MET ∩ instance(M); // instance(M) is a vertex representing M
        foreach p ∈ P do
                WM ← WM ∩ (instance(M),p);
        popm(instance(M)) ← 1; // initialize the weight of a new method
        cons(L,M) ← cons(L,M)+1;
        if cons(L,M)>threshold_cons then ModifyCons();
else // M is already present in MET, so update weights only
        e ← take_from(Xi);
        popm(e) ← popm(e) + 1;
        if popm(e)>threshold_popm then ModifyPopM();
        cons(kind(e),M) ← cons(kind(e),M)+1;
        if cons(kind(e),M)>threshold_cons then ModifyCons();
end.
```

If there was no such call before add new vertex with weight 1

If there was such call increase its weight by 1

In both cases increase entry in the consequence matrix

Formalism (PredictRequest algorithm I)

algorithm `PredictRequest`(R=(M,P))
Input: Request R (method M and parameters list P)

begin
// in lines 5.-12. we find the vertex representing the instance of M (there can be at most one such vertex)
$Y \leftarrow P$;
$y \leftarrow$ take_from(Y);
$X\_1 \leftarrow \{m \in MET: kind(m)=M \wedge (m,y) \backslash in\ WM \wedge ord(m,y)=1\}$;
$i \leftarrow 2$;
while $(Y \neq \varnothing) \wedge (X_{i-1} \neq \varnothing)$ **do**
  $y \leftarrow$ take_from(Y);
  $X\_i \leftarrow X_{i-1} \cup \{m \in MET: kind(m)=M \wedge (m,y) \in WM \wedge ord(m,y)=i\}$;
  $i \leftarrow i+1$;
  $Z \leftarrow X_{i-1}$; // notice that $|Z|=0$ or $|Z|=1$;
//if $|Z|=0$ then M or some of it's parameters were removed from the graph
if $Z=\varnothing$ **return** 0; // we predict nothing

Find vertex in MET that represents current call (similarly as in UpdateGraph)

Call issued first time: prediction impossible

Formalism (PredictRequest algorithm II)

else

Neigh ← P; // Neigh will represent P and its 1-neighborhood

P_R ← ∅; // P_R will be the set of methods whose parameters sets are totally included in Neigh (lines 19.-23.)

foreach p ∈ P **do** Neigh ← Neigh ∩ {r ∈ PAR: {p,r} ∈ WP};

foreach n ∈ Neigh **do**

T ← {m ∈ MET: (m,n) ∈ WM};

foreach t ∈ T **do**

if {p ∈ PAR: (t,p) ∈ WM} ⊆ Neigh **then** P_R ← P_R ∩ {t};

P_R ← P_R \ take_from(Z); // exclude current request from the set of predictions

foreach q ∈ P_R **do** // compute ranks of all request from P_R

v ← 0;

foreach p ∈ P: (q,p) ∈ WM **do**

v ← v + *parw*(p);

v ← v/|P|; //v = mean weight in clique of parameters

rank(q) ← *cons*(M,*kind*(q)) * *popm*(q) * v;

SortAndCut(P_R); //sort requests regarding their ranks and leave only //a fixed number of requests with the highest rank

end.

All parameters that historically appeared together with parameters of current call

All history calls that contain parameters from set 'increased' by 1 neighbourhood

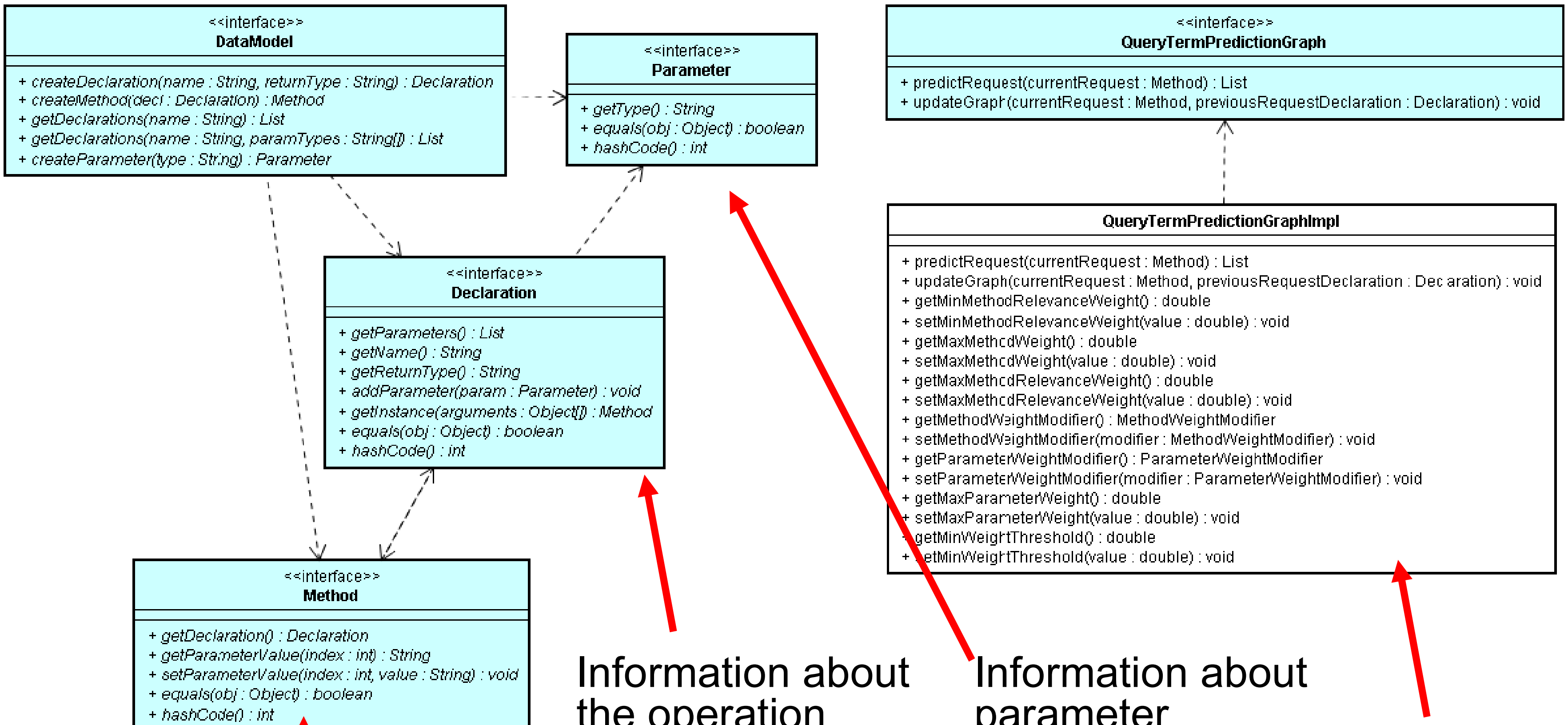
Heuristic ranking of found calls

RPS implementation – some details

Difficulty:
ASK-IT interfaces are build according to document style and RPS assumes that the requests are issued according to RPC style.

- Intermediate layer is added that translates document style requests to RPC style requests,
- Some arguments that are transient (e.g. timestamp) are removed, some are transformed (e. g. geographic coordinates, arrays).

RPS implementation - architecture



Information about the operation and all parameter types

Information about parameter type and significance

Algorithm

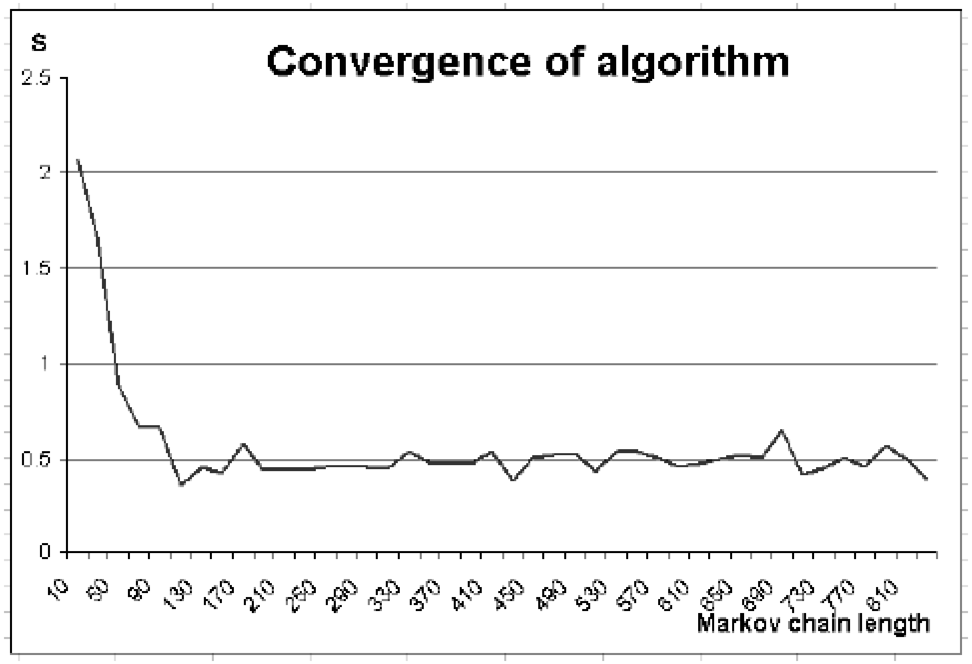
Instance of request

Prototype tests

Sequence of calls from the set of 5 generated by Markov chain used to train the RPS. Predicted requests are compared with the entries in the Markov matrix.

Two criteria: best choice of request and S measure – distance between Markov matrix and normalized ranking

$$S = \sum_{i=1}^{5} \sum_{j=1}^{5} (M(i,j) - M'(i,j))^2$$



i	Order of elements in I-th row of M	Element chosen by algorithm	Comment
1	5, 2, 4, **3**, 1	**3**	4th one was chosen
2	**5**, 1, 3, 4, 2	**5**	optimal choice
3	**5**, 4, 2, 1, 3	**5**	optimal choice
4	1, 3, **5**, 2, 4	**5**	3rd one was chosen
5	**3**, 1, 4, 2, 5	**3**	optimal choice

To do

- Perform more prototype test in order to tune the algorithms using system logs
  - ranking formula
  - weight update
  - analysis of larger neighbourhood in graph
- Perform tests with real life data
- Integration with ASK-IT environment
  - system agents
  - match-making
- Add more data mining functionalities
  - user group profiles

Concluding remarks

- Effective algorithm that predicts the forthcoming calls in the system based on Web services was proposed.
- The algorithm has smaller computational cost than algorithms based on Markov chains that are usually used in this context.
- The tests (for small size of input data) confirmed the efficiency of algorithm.

Thank you!

Questions?

uipodola@if.uj.edu.pl
roman@ii.uj.edu.pl
kalita@softlab.ii.uj.edu.pl