# Use Case Design Specification

## **Recommend a solution to fulfil Request-to-Answer process with lowest cost or shortest response time for X number of <Customer Inquiries> with current resource plan**

Step 1: Define business rules for calculation of Cost and Response time for each node and relationship

#1.1 Introduce Business Rules and heuristic function to calculate the cost for each node according to the X load variable and Resource parameter a or b.

**Rule1:** Cost for People f(x)

**IF** node label is People

**Then** cost function f (x) = a+x/0.5\*a ( a stands for value of resource property for People node)

**Rule2:** Cost for Technology g(x)

**IF** node label is Technology

**Then** cost function g(x) = 2b+x/b (b stands for value of resource property for Technology node)

**Rule3:** Cost for Information t(x)

**IF** node label is Information

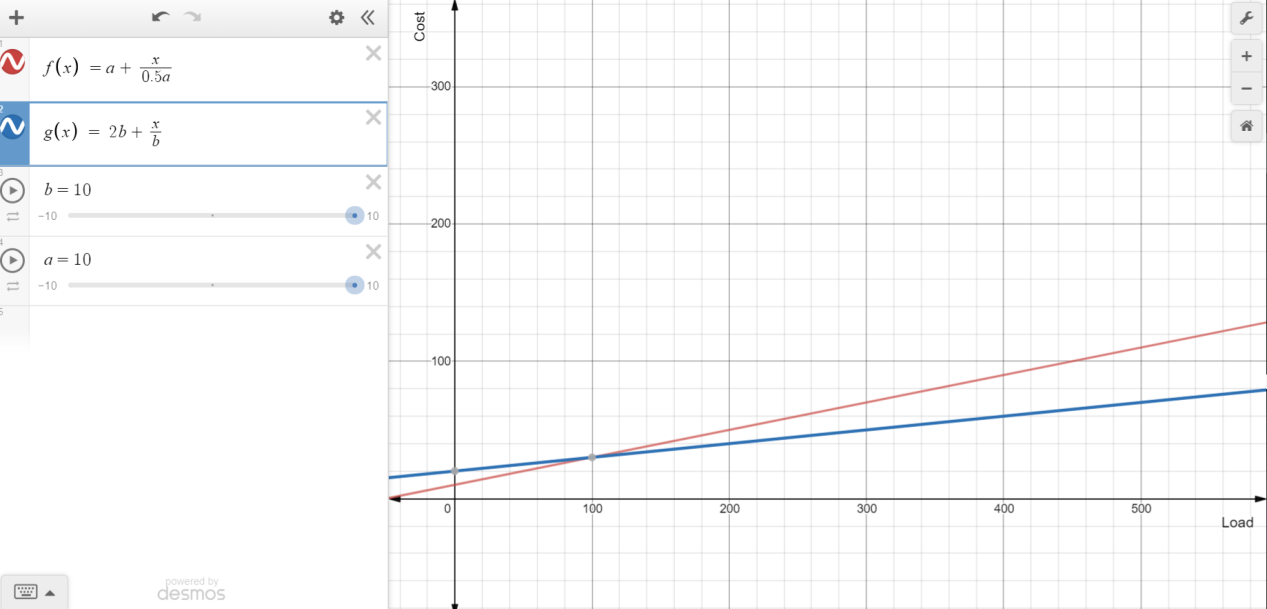
**Then** cost t (x) = 1 by default, because Information is a virtual entity.

**Rule1:** Cost for Process z(x)

**IF** node label is People

**Then** cost function z (x) = c+x/0.5\*c ( a stands for value of resource property for People node)

Refer to below Function graph view for cost function



#1.2 Introduce Business Rules and heuristic function to calculate the response time for each node according to the load X variable and Resource parameter a or b.

**Rule4:** Response Time for People f(x)

**IF** node label is People

**Then** cost f (x) = x+x/a ( a stands for value of resource property for People node)

**Rule5:** Response Time for Technology g(x)

**IF** node label is People

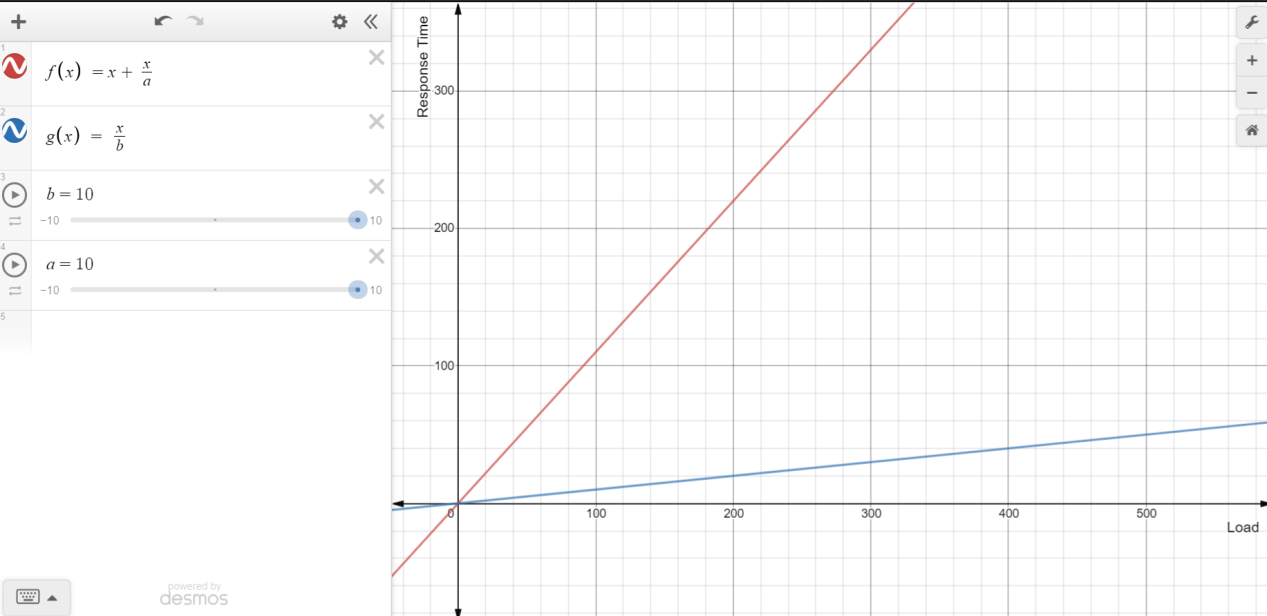
**Then** cost g (x) = x/b (b stands for value of resource property for Technology node)

**Rule6:** Response Time for Information t(x)

**IF** node label is Information

**Then** cost t (x) = 1 by default, because information is a virtual entity.

Refer to below Function graph view for Response Time function

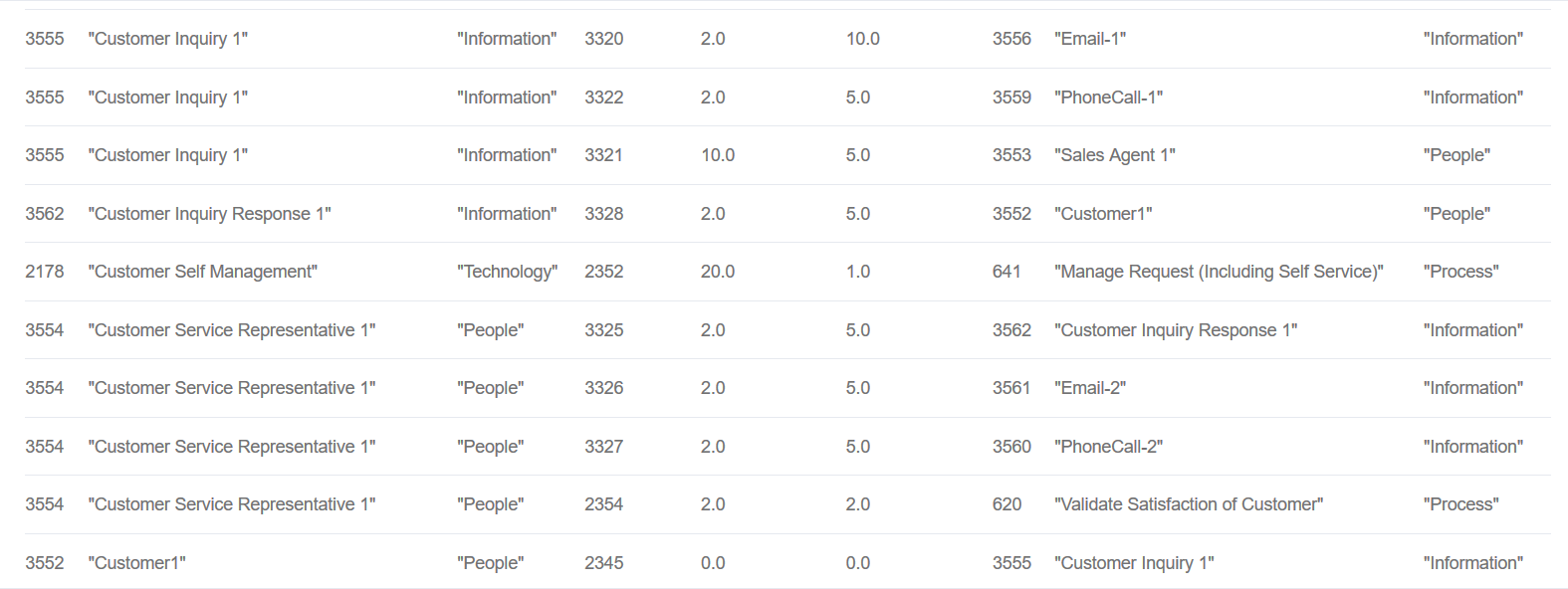


Step 2: Spread and Distribute the X number of Customer Inquiries into Request\_to\_Answer Process (Breadth First Distribution)

#2.1 Generate Table view (Search Space) for Request\_to\_Answer Process

**match (n) - [r:Request\_to\_Answer] -> (p) return n.nodeID as Start, n.longName as Start\_Name, labels(n)[0] as Start\_Label, id(r)as relation\_ID,r.relationCost as Relation\_Cost, r.relationTime as Relation\_Time, p.nodeID as End, p.longName as End\_Name, labels(p)[0] as End\_Label order by Start\_Name, End\_Name**





#2.2Reset all node’s state to empty:

**match (n) - [r:Request\_to\_Answer]-> (p) set n.state='',r.relationState='',p.state='' return n,p**

#2.3 Define a Frontier List (This is used for the load distribution algorithm described in step 3)

Define a Frontier List[**Node ID**], a FIFO queue to contain list of child nodes in the load distribution algorithm (Breadth-first distribution), set to empty list first.

#2.3 Load in X from starting node, for example X=100.

**match (n:People {nodeID:3552,longName:'Customer1'}) set n.load = 100,n.state='Loaded'**

**Get all the Child Node ids for Customer1 from above table (Search space), and push the child nodes into Frontier List.**

Step 3: Traverse the path in a Breadth First way to Spread and Distribute the load to each node and relationship, also calculate the cost and response time properties.

**While (**Frontier List is not empty)

**Loop Do**

node A = Pop the Frontier List (FIFO)

**IF** node A’s state property is “Distributed” **then return** (This means finished distributing all loads)

**For** rows (actions) starts with A in above table (Search Space)

**IF** number of rows= 1

**Then** Distribute the whole load X from A to child node B:

Set X value to relationLoad property between A and B.

Set resource property value of node A to relationResource property between A and B.

Set “Distributed” to relationState property between A and B.

**IF** node B’s state is “Loaded”

Append X to load property in node B. (**set b.load = b.load+X**)

**Else** set X to load property in node B (**set b.load = X**)

Set “Loaded” to state property in node B.

Set “Distributed” to state property in node A.

**match (a {nodeID:3552}) - [r:Request\_to\_Answer] -> (b {nodeID:3555}) set r.relationLoad = 100, r.relationState='Distributed',b.load=100,b.state='Frontier', a.state='Distributed' return a,r,b**

//End of IF number of rows = 1

Find matching rules from both cost function and time function for node A.

Fire matched rules for node A and its relationship to all the child nodes

**match (a {nodeID:3552}) set a.cost = f(x)/g(x)/t(x), a.time=f(x)/g(x)/t(x), using node resource and load property value in the function.**

**match (a {nodeID:3552}) - [r:Request\_to\_Answer] -> (b {nodeID:3555}) set r.relationCost = f(x)/g(x)/t(x), using relationResource and relationLoad property value in the function.**

**match (a {nodeID:3552}) - [r:Request\_to\_Answer] -> (b {nodeID:3555}) set r.relationTime = f(x)/g(x)/t(x), using relationResource and relationLoad property value in the function.**

Remove A from Frontier List and add B to Frontier list

**IF** number of rows > 1

**Then** Randomly distribute the x% of load X from A to child nodes B, C, …:

Set x% load value to relationLoad property between A and B, C, ….

Set x% of resource from node A to relationResource property between A and B, C,….

Set “Distributed” to relationState property between A and B, C,….

**IF** node B, C,…’s state is “Loaded”

Append X to load property in node B, C,…. (**set b.load = b.load+X**)

**Else** set X to load property in node B (**set b.load = X**)

Set “Loaded” to state property in node B, C, …,.

Set “Distributed” to state property in node A.

**match (a {nodeID:3555}) - [r:Request\_to\_Answer] -> (b {nodeID:3320}) set r.relationLoad = 100, r.relationState='Distributed',b.load=100,b.state='Frontier', a.state='Distributed' return a,r,b**

**match (a {nodeID:3555}) - [r:Request\_to\_Answer] -> (c {nodeID:3322}) set r.relationLoad = 100, r.relationState='Distributed',c.load=100,c.state='Frontier', a.state='Distributed' return a,r,c**

**…**

//End of IF number of rows > 1

Find matching rules from both cost function and time function for node A.

Fire matched rules for node A and its relationship to all the child nodes

**match (a {nodeID:3555}) set a.cost = f(x)/g(x)/t(x), a.time=f(x)/g(x)/t(x), using node resource and load property value in the function.**

**match (a {nodeID:3555}) - [r:Request\_to\_Answer] -> (b {nodeID:3320}) set r.relationCost = f(x)/g(x)/t(x), using relationResource and relationLoad property value in the function.**

**match (a {nodeID:3552}) - [r:Request\_to\_Answer] -> (b {nodeID:3555}) set r.relationTime = f(x)/g(x)/t(x), using relationResource and relationLoad property value in the function.**

Remove A from Frontier List and add B, C, … to Frontier list

**IF** number of rows= 0 (Dead end whereby the frontier node doesn’t have child)

**Then** set “Distributed” to state property in node A and remove A from Frontier List.

Step 4: Calculate the shortest path for solution from the data generated in above steps - for lowest cost

#4.1 Shortest Path Algorithm (Dijkstra) - lowest cost

**MATCH (start {name:'Customer1'}), (end {name:'CIR1'})**

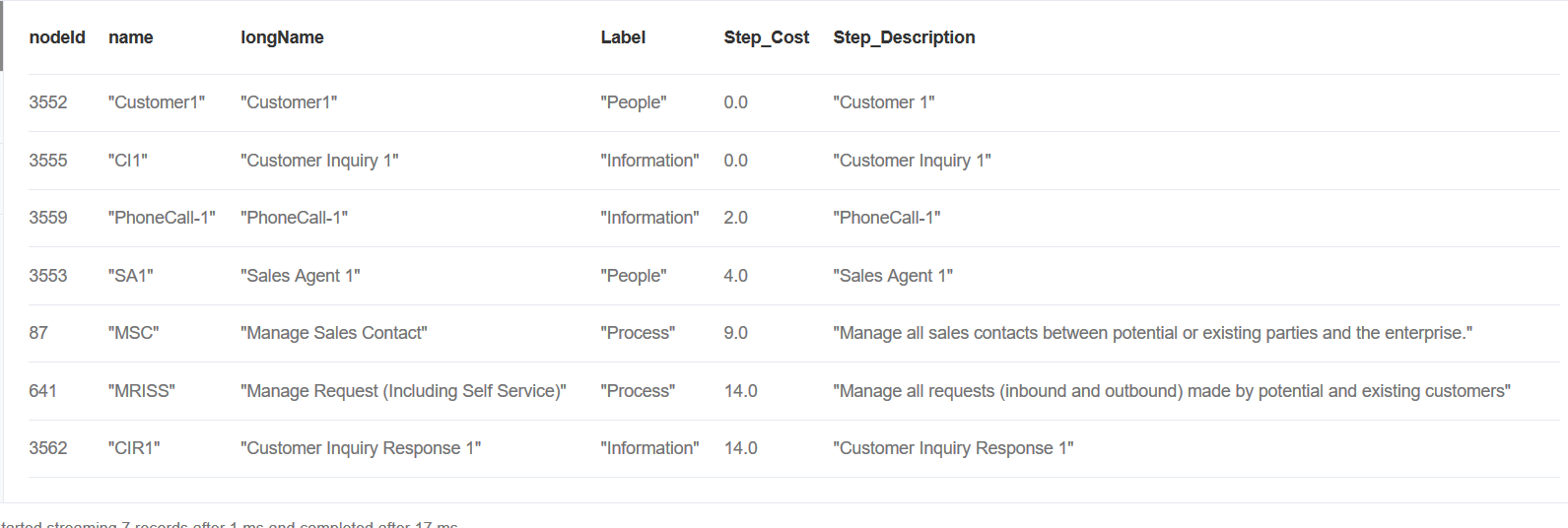
**CALL algo.shortestPath.stream(start, end, 'relationCost',{relationshipQuery:'Request\_to\_Answer',direction:'OUTGOING'})**

**YIELD nodeId, cost**

**RETURN nodeId, algo.asNode(nodeId).name as name, algo.asNode(nodeId).longName AS longName, labels(algo.asNode(nodeId))[0] as Label, cost as Step\_Cost,algo.asNode(nodeId).shortDescription as Step\_Description**

#4.2 Display the following result in the Web UI with one summary statement and a table that illustrate the step cost details

**Solution Summary**: The following steps can fulfil the Request-to-Answer process with the lowest cost 14.0, please refer to below table for the step cost summary

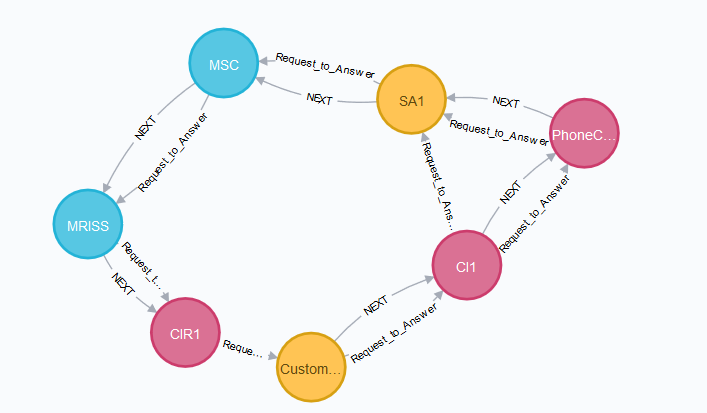


#4.3 Use the following cypher query to display the solution in graph view

**CALL** algo.asPath([3552,3555,3559,3553,87,641,3562])

#4.4: Display one summary statement and the solution graph view

**Solution Graph View**: The following graph shows the end-to-end graph view about the optimized process flow.



Step 5: Calculate the shortest path for solution from the data generated in above steps - shortest time.

#5.1 Shortest Path Algorithm (Dijkstra) - shortest time

**MATCH (start {name:'Customer1'}), (end {name:'CIR1'})**

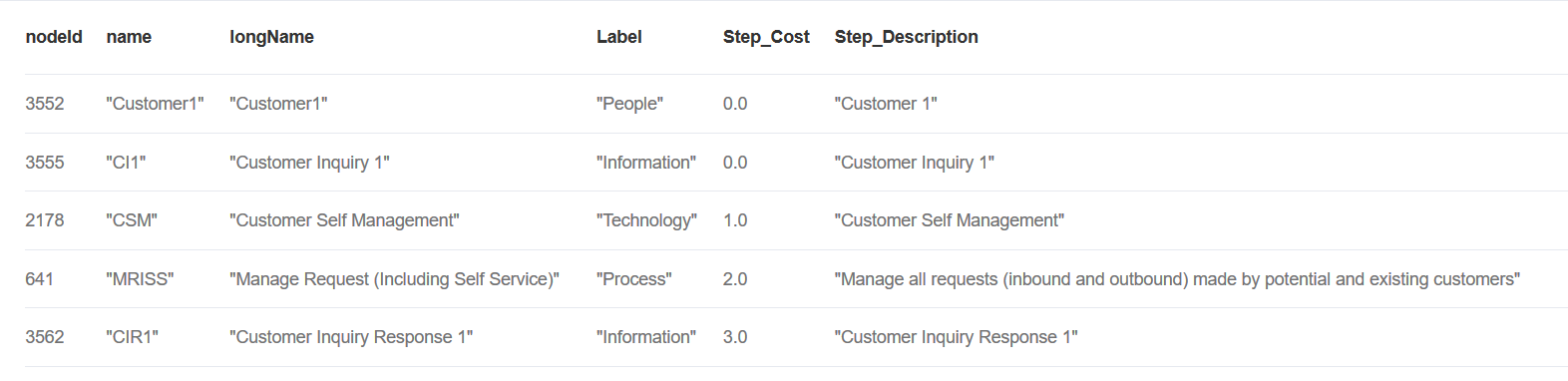
**CALL algo.shortestPath.stream(start, end, 'relationTime',{relationshipQuery:'Request\_to\_Answer',direction:'OUTGOING'})**

**YIELD nodeId, cost**

**RETURN nodeId, algo.asNode(nodeId).name as name, algo.asNode(nodeId).longName AS longName, labels(algo.asNode(nodeId))[0] as Label, cost as Step\_Cost,algo.asNode(nodeId).shortDescription as Step\_Description**

Step 5: Display the following result in the Web UI with one summary statement and a table that illustrate the step cost details

**Solution Summary**: The following steps can fulfil the Request-to-Answer process with the shortest time 3.0, please refer to below table for the step cost summary.



Step 6: use the following cypher query to display the solution in graph view

**CALL** algo.asPath([3552,3555,2178,641, 3562])

Step4: Display one summary statement and the solution graph view

**Solution Graph View**: The following graph shows the end-to-end graph view about the optimized process flow.

