AUTHORS

This document was prepared by:

Nicholas Lasswell  
Borealis  
Phone: (618) 308-2049  
nicholas.lasswell@siu.edu

Nicholas Pennington  
Borealis  
Phone: (217) 883-8149  
npennington@siu.edu

Michael Smith  
Borealis  
Phone: (618) 571-1256  
michael.r.smith@siu.edu

VERSION HISTORY

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
<th>Author/Reviser</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/9/2018</td>
<td>1.0</td>
<td>Internal Meetings</td>
<td>Team Borealis</td>
</tr>
<tr>
<td>2/26/2018</td>
<td>1.1</td>
<td>Internal Meetings</td>
<td>Team Borealis</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS

1. Introduction .............................................................................................................. 1

2. GENERAL OVERVIEW AND DESIGN GUIDELINES/APPROACH ...................... 1

   2.1 Assumptions / Constraints / Standards ............................................................. 1

3. ARCHITECTURE DESIGN ......................................................................................... 2

   3.1 Software Architecture: High level architecture diagram ................................... 2

   3.2 Performance ....................................................................................................... 3

4. SYSTEM DESIGN ...................................................................................................... 3

   4.1 Use-Cases ......................................................................................................... 3

   4.2 User Interface Design ....................................................................................... 3

   4.3 Application Program Interfaces .................................................................... 4

   4.4 Algorithm Design ............................................................................................ 6

   4.5 Class Diagram .................................................................................................. 8
1. **INTRODUCTION**

The Product Design Specification document documents and tracks the necessary information required to effectively define architecture and system design in order to give the development team guidance on architecture of the system to be developed. The Product Design Specification document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and development team. Some portions of this document such as the user interface (UI) may on occasion be shared with the client/user, and other stakeholders whose input/approval into the UI is needed.

2. **GENERAL OVERVIEW AND DESIGN GUIDELINES/APPROACH**

2.1 **Assumptions / Constraints / Standards**

- **Assumptions**
  - The connections between the nodes within a model are static, and cannot be altered.
  - The specifications of the systems being modeled are known well enough to reasonably be modeled.
  - The properties of the model, including service rates and the set of resources, are fixed. The only property within the model that can be changed by the optimization process is the amount of each resource.

- **Constraints**
  - The project is under the strict time constraints of a single college semester. As such,
    - Features such as user management and shift management will be left out
    - Users will not be able to save their created stochastic flow models
    - The optimization will not be able to adjust any connections between nodes
3. **ARCHITECTURE DESIGN**

3.1 **Software Architecture: High level architecture diagram**

![Architecture Diagram](image-url)

- Event Queue
- Simulation
- Graph
- Simulation Inputs
- Simulation Outputs
- FrontEnd
- Optimizer Inputs
- Optimizer Outputs
- Optimizer
3.2 Performance

- The system must be designed with performance in mind due to its high computational requirements.
- The optimization process shall use a thread pool where appropriate, to make use of all of the processing power available.

4. SYSTEM DESIGN

4.1 Use-Cases

- An owner of a business wants to increase the number of customers that can be processed in a day, and reduce the number of customers leaving from impatience, while keeping costs low.
  - The business owner can model the customer’s processing as a stochastic flow system using the editor.
  - The business owner can then run the simulation to determine any bottlenecks in the system.
  - The business owner can run the optimization to determine what changes could be made to improve throughput in the system while keeping factors such as cost low.
- A researcher wants to improve the efficiency of communications within a network.
  - The network can be modelled as a stochastic flow system using the editor.
  - The optimization can be run, pointing out adjustments that can be made to increase the efficiency of the communications.
- A hospital emergency department wants to improve its patient throughput to decrease the number of patients leaving.

4.2 User Interface Design

- The application consists of one main webpage, being an editor for stochastic flow models.
- The editor shall be built with a convenient drag and drop interface, using the Javascript library jsPlumb.
- Double clicking on the editor creates a new node under the mouse, after prompting the user with options
  - The options determine properties on the node, including processing rate, name, and cost.
- Right clicking on a node allows the user to edit the relevant properties of the node.
- The user shall be able to drag and drop connections between these nodes, prompting the user with options.
  - The options determine properties on the connection, including the probabilities for clients to transition using the connection, and the type of edge.
- The bottom of the page should contain “Run”, “Optimize”, and “Edit Resources” buttons.
  - The “Run” button runs the simulation, and outputs the results. The colors of the nodes should adjust to reflect the amount clients had to wait at them, turning redder the more of a bottleneck the node is.
  - The “Optimize” button should run the genetic algorithm, and return an optimal configuration for the nodes in the flow model.
  - The “Edit Resources” button should load a modal containing a table, with all of the currently defined resources, their names, their costs per resource, the minimum and maximum amounts (for optimization), and the current amount (for simulation).
4.3 Application Program Interfaces

**Simulation Inputs (JSON):**

```
"ArrivalRate": double[5], // arrival rate of clients per priority, null if client data is provided
"EntryNode": int,
"Effort": int,              // the number of times to run the simulation before averaging results
"MaxTime": int,           // the time at which the simulation stops running
"PastTime": Nullable<double>, // how far into the past to run the simulation. null: 0
"Resources": [           // cost per server
    "Id": int,
    "Name": double,
    "Cost": double,
    "MinServers": int, // for optimization -- minimum number of resources possible
    "MaxServers": int, // for optimization -- maximum number of resources possible
    "NumServers": int, // for simulation -- current number of resources
],
"NodeSet": [            // does this node use a priority queue, or a standard queue?
    "Id": int,
    "Name": string,
    "PriorityQueue": boolean,
    "EdgeSet": [
        "ProbabilityVector": double[5], // probability for client to use this edge
        "FromNode": int,
        "ToNode": Nullable<int>, // null: exit the system
        "Type": int,              // 0: regular, 1: one-time
    ],
    "Resources": [        // number of clients served per unit time on average
        "Id": int,
        "ProcessingRate": double[5],
    ],
],
"ClientData": {         // time and priority for clients arriving into the system, can be null if
    "Clients": [         // no data and arrivalrate is provided
        "Time": double,
        "Priority": int,
    ],
}
```
Simulation Outputs (JSON):

```
"WaitAverage": double,    // Average wait time
"WaitDeviation": double,  // Deviation of wait time over iterations
"WaitPriority": double[5], // Time priorities waited
"WaitTime": double[N],    // Wait time at node
"Cost": double,           // Cost of simulation
"LengthOfStayAverage": double, // Average Length of stay per client
"LengthOfStayDeviation": double, // Length of stay per client deviation over iterations
"LengthOfStayPriority": double[5], // Length of stay per client at priority
"LengthOfStayTime": double[N], // Length of stay time per unit time
"NumClientsAverage": double, // Average number of clients
"NumClientsDeviation": double, // Average number of clients over iterations
"NumClientsPriority": double[5], // Average number of clients per priority
"ResourceOutputs": [
    "Cost": double,    // Cost per resource
    "Id": int,
    "Name": string,   // Name of resource
    "UtilTime", double[N] // Utilization of the resource per unit time
],
"NodeOutputs": [
    "Id": int,
    "WaitAverage": double,
    "WaitDeviation": double,
    "WaitPriority": double[5],
    "WaitTime": double[N],
    "LengthOfStayAverage": double,
    "LengthOfStayDeviation": double,
    "LengthOfStayPriority": double[5],
    "LengthOfStayTime": double[N],
    "LengthOfStayTime": double[N],
    "NumClientsAverage": double,
    "NumClientsDeviation": double,
    "NumClientsPriority": double[5],
    "QueueTime": double[N],
    "TimeToAverage": double, // Average time it took to get to node.
    "TimeToDeviation": double, // Deviation of time to node over iterations
    "TimeToTime": double[N],    // ToTimeAverage per unit time
]
```
**Optimization Inputs (JSON):**

```
"SimulationInputs": <SimulationInputs>,
"Budget": int
```

**Optimization Outputs (JSON):**

```
"Before": <SimulationOutputs>,
"After": <SimulationOutputs>,
"Resources": [
  "Id": int,
  "Name": string,
  "CostBefore": double,
  "CostAfter": double,
  "Adjustment": int,
]
```

### 4.4 Algorithm Design

The simulation runs as a discrete event simulator, with a priority queue of events ordered by the time the event takes place. The simulation essentially occurs as follows:

```java
AddInitialEvents();
while (EventQueue.HasEvent()):
  Clock = EventQueue.NextTime()
  EventQueue.Dequeue().Execute();
```

The main type of event is a client transition event, encompassing client arrivals, client exits, and client transitions between states. The pseudocode for this event's execution is as follows:

**Events**

The main type of event is the client transition event, where one client transitions from one state to another. This event takes the following parameters: FromNode, ToNode, and Time. FromNode/ToNode can be null, representing a client entry or exit respectively. The majority of events that occur in the system occur when a client transitions from one state to another, allowing this single type of event to handle most of the logic.

On each client transition, the tasks that need to be done can be split into tasks involving the node the client is leaving, and tasks involving the node the client is entering. In the node the client is leaving, the client must be taken out, and if there is another client ready to be served, they should be served. In the node the client is going to, if there is a server free to serve the client, they should be served immediately. If not, they should be enqueued to be served later. The pseudo-code on the next page describes this algorithm.

**Optimizer**

The optimizer will run as a genetic algorithm, and will decide the number of each resource constrained by a given budget. The user will supply a budget, and the optimizer will attempt to give the optimal number for each resource under that budget constraint. The error measure of this optimizer will be chosen by the user, taking average wait and length of stay into account.
void ClientTransitionEvent(fromNode, toNode, client, time):
    /* Handle FromNode */
    if (fromNode != null):
        fromNode.FinishServing(client)
        // if, after removing this client, there is another client waiting to be served, serve them
        if (fromNode.ClientToServe()):
            Client temp = fromNode.NextClient()
            fromNode.ServeClient(temp)

            // schedule that client's next transition event
            AddEvent(fromNode,
                fromNode.SelectNode(temp.Priority),
                temp,
                time + NextEvent(fromNode.ProcessingRate[temp.Priority])
        )

    else: // fromNode == null => This is a client arrival
        // create a new client and schedule it's arrival event
        Client newClient = new Client(Client.Priority)
        AddEvent(null,
            EntryNode,
            newClient,
            time + NextEvent(ArrivalRate[Client.Priority])
        )

    /* Handle ToNode */
    if (toNode != null):
        // if there is a server free at the new node, serve the client
        if (toNode.ServerFree()):
            ToNode.ServeClient(client)

            // schedule the client's next transition event
            AddEvent(toNode,
                toNode.SelectNode(Client.Priority),
                Client,
                time = NextEvent(toNode.ProcessingRate[Client.Priority])
        )
        else: // if there isn't a server free, enqueue the client
            ToNode.Enqueue(Client)

    else: // toNode == null => this is a client exit
        Client.Exit()