# Enhancing Healthcare Efficiency and Tracking Precision with Devs Java-Based Patient Tagging Systems

**PARDIS SADATIAN MOGHADDAM - IAN PRUITT***, Georgia State University-Department of Computer Science, USA

In today's global context, public health systems worldwide face increasing pressure to meet growing demands for healthcare services within financial limitations. This necessitates optimizing public healthcare spending while ensuring the best possible health outcomes. Recognizing this challenge, the Ministry of Health (MOH) is leveraging corporate performance improvement methodologies to enhance hospital operations and prioritize patient care. This project focuses on optimizing healthcare delivery, particularly for trauma and emergency cases, by introducing specialized queues and streamlining workflows. Leveraging Devs Java technology, the software ensures efficient patient tracking and resource allocation in real-time. Simulation models aid in understanding patient flow dynamics and optimizing queuing systems. The tool's purpose includes improving efficiency, reliability, security, customization, and scalability of queuing systems, while also offering cost-effectiveness, interoperability, predictive analytics, real-time monitoring, user experience enhancement, and regulatory compliance. Overall, the project aims to achieve optimal health outcomes within public healthcare budgets by leveraging advanced technology and innovative methodologies.

Additional Key Words and Phrases: Public health systems, Healthcare services, Financial limitations, Devs Java technology, Queuing systems, Cost-effectiveness, Patient care

**ACM Reference Format:**
**PARDIS Sadatian Moghaddam - Ian Pruitt**. 2024. Enhancing Healthcare Efficiency and Tracking Precision with Devs Java-Based Patient Tagging Systems. 1, 1 (April 2024), 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 PROBLEM STATEMENT

In today's global landscape, public health systems face escalating demands for healthcare services amid financial limitations. The Ministry of Health (MOH) responds by leveraging corporate performance methodologies to enhance hospital operations, prioritizing patient care, especially for trauma and emergency cases.

Our project introduces specialized queues for trauma patients, ensuring prompt intervention and minimizing critical care delays. We address the urgency of efficient workflows, as prolonged wait times lead to decreased productivity and patient dissatisfaction.

Using Devs Java technology, our software optimizes operations and patient tracking. By dynamically managing unique barcode tags, we ensure seamless updates as patients navigate healthcare facilities, facilitating resource allocation and operational improvements.

Incorporating advancements in simulation models and operations research, our software enhances medical care delivery and patient satisfaction. Overall, our project aims to achieve optimal health outcomes within public healthcare budgets.

Author's address: **PARDIS Sadatian Moghaddam - Ian Pruitt**, psadatian1@student.gsu.edu, ipruitt2@student.gsu.edu, Georgia State University-Department of Computer Science, 25 Park Place, Atlanta, Georgia, USA, 30302.


XXXX-XXXX/2024/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 2 MODELING AND SIMULATION GOALS

In recent years, the global population surge has strained healthcare systems worldwide, emphasizing the need for robust IT integration, especially in emergencies. Innovative solutions leveraging cloud computing, IoT, ML, and AI can enhance patient experiences and operational efficiency.

Our project focuses on queuing systems in healthcare, using DEVS Java to optimize processes. Our smart hospital information system models patient flows accurately, with a unique tagging system for real-time tracking.

Powered by DEVS Java, our simulation provides insights into patient flow dynamics and resource utilization, crucial in critical scenarios like heart attacks. As demand for advanced healthcare rises, our project aims to create efficient, reliable solutions that elevate patient care by studying queuing systems' impact on outcomes.

## 3 PURPOSE OF THE TOOL

In leveraging DEVS Java for the development of queuing systems in healthcare, our objectives encompass various critical aspects aimed at enhancing patient care delivery and operational efficiency.

(1) **Efficiency:** The utilization of DEVS Java's robustness and scalability enables the creation of efficient queuing systems. By reducing patient wait times, optimizing resource allocation, and streamlining workflows for healthcare providers, these systems enhance overall operational efficiency.
(2) **Reliability:** : The mature ecosystem and proven reliability of DEVS Java ensure seamless and consistent operation of queuing systems. This minimizes the risk of system failures, thereby maintaining continuity in patient care delivery.
(3) **Security:** : DEVS Java's built-in security features and support for encryption safeguard patient data within queuing systems, ensuring compliance with privacy regulations and protecting sensitive information.
(4) **Customization:** : DEVS Java's flexibility allows for the customization of queuing algorithms and logic to suit different healthcare settings, patient populations, and clinical priorities, enhancing adaptability and relevance.
(5) **Scalability**: DEVS Java's scalability empowers queuing systems to adapt to fluctuations in patient volume, clinical demands, and organizational growth, ensuring long-term effectiveness and responsiveness.

By considering these perspectives and leveraging DEVS Java for queuing system development in healthcare, organizations can achieve cost-effective, interoperable, predictive, user-friendly, compliant, and efficiently monitored solutions that ultimately enhance patient care delivery.

## 4 HOW DOES THE DEVS MODEL WORK

DEVS, or Discrete Event System Specification, relies on the concept of experimental frames to structure system modeling. It consists of three main parts: the real system, the model, and the simulator. Treating experimental frames akin to models, DEVS offers a flexible approach to simulation, seamlessly integrating the modeler's objectives.

In clinic-hospital systems, DEVS is valuable for constructing computer simulation models to analyze dynamics and identify improvements. Embracing DEVS methodology, we delineate these components:

**The Model**

(1) **Patient Arrival Time:** Incorporating stochastic arrival times for patients renders the simulation more veritable, circumventing the portrayal of a contrived scenario where all patients converge simultaneously.
(2) **Queues:** : The model discerns between two segregated queues: trauma patients occupy a designated queue, while emergency and regular patients coalesce within another queue.
(3) **State Transitions:** : Tracking the system's state entails monitoring pivotal events such as patient arrivals, queue ingress, and treatment allocation, instigating transitions between states within the DEVS model.

**Simulation:** The DEVS model employs a dedicated simulator to manage time progression and patient flow in clinic-hospital systems. By integrating basic models for patient arrival, queue dynamics, and treatment processes, comprehensive simulations yield insights into metrics like waiting times and resource utilization. This systematic approach allows experimentation with different scenarios, such as adjusting staffing or clinic hours, to optimize patient flow and care delivery.

DEVS facilitates the creation of a digital twin for clinic-hospital systems, enabling stakeholders to explore enhancements and optimize care delivery.

During high patient influx, Crossing Rivers Health implements a triage system, prioritizing individuals based on medical urgency. The triage process includes five levels:

**Level 1:** Resuscitation - For unresponsive or unconscious patients with life-threatening conditions.

**Level 2:** Emergent - For patients with urgent medical needs requiring immediate care but not life-threatening.

**Level 3:** Urgent - For patients with serious conditions needing prompt attention but not emergencies.

**Level 4:** Less Urgent - For patients with non-critical conditions treatable without immediate intervention.

**Level 5:** Non-Urgent - For patients with minor symptoms not considered medical emergencies, likely treatable at home or by a primary care physician.

In our simulation, we classify patients into three categories based on the triage system:

(1) **Regular Patients:** Minor medical issues like flu or ear infections. Lowest priority, queued for 10 minutes with 5 minutes for barcode generation.
(2) **Emergency Patients:** : Injuries or urgent conditions not life-threatening. Higher priority than regular patients, queued for 20 minutes with 10 minutes for barcode generation.
(3) **Trauma Patients:** : Severe cases like third-degree burns. Bypass queue, receive immediate care, then transferred to hospitals. Processing includes 20 minutes for stabilization at the clinic and 15 minutes for hospital transfer.
Following triage assessment, patients are directed to appropriate facilities based on their classification: Regular or emergency patients proceed to clinics for varied durations, where the triage system optimizes care. Trauma patients, needing specialized care beyond clinic capabilities, swiftly transfer to hospitals. Medical-billing centers manage administrative tasks and prioritize patients for efficient billing. Post-hospital discharge, trauma patients undergo administrative procedures at the billing center, which operates on a queue system based on arrival time.

Here is In Fig. 1 is the diagram depicting our model.

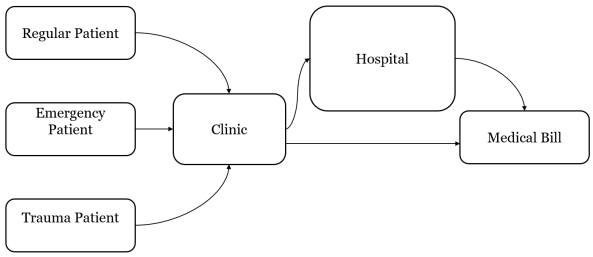


Fig. 1. Clinic-Hospital Model

### 4.1 Clinic-Hospital Class:

The Clinic-Hospital class serves as the core of the hospital management system. It extends the ViewableAtomic class, inheriting its functionalities. This class orchestrates the flow of patients through different treatment pathways based on their conditions. It features multiple ports for receiving patient information categorized by their urgency levels, including Trauma, Emergency, and Regular Patients. Upon receiving patient data, it calculates the processing time and schedules the patient accordingly. Additionally, it manages a queue (DEVSQueue) to handle patient overflow, ensuring efficient patient flow within the hospital.

Algorithm 1 represents the pseudocode of this class.

### 4.2 Hospital Class:

The Hospital class, extending ViewableAtomic, represents a treatment facility within the hospital management system. It receives patient information, particularly regarding Trauma cases, and processes them accordingly. Upon receiving a Trauma patient, it schedules the treatment process and generates a medical bill upon completion. This class contributes to the overall patient flow management within the hospital system.

### 4.3 MedicalBill Class:

Similar to the Hospital class, the MedicalBill class extends ViewableAtomic and is responsible for handling the generation of medical bills within the hospital management system. It receives patient data, including Emergency cases and Regular Patient information, and generates corresponding medical bills. This class plays a crucial role in the financial aspect of hospital operations, ensuring accurate billing for the services provided to patients.

### 4.4 The Clinic-Hospital System Class

The ClinicHospital-System class is a pivotal component of our hospital management simulation, orchestrating the interactions between various system components. Let's delve into its functionalities and operational mechanisms:

**Initialization:**

The constructor methods ClinicHospital-System() and ClinicHospital-System(String name) initialize the digraph, ensuring its seamless integration within the simulation framework. Essential parameters and components are instantiated to facilitate the flow of patients and information within the system.

**Components Initialization:**

**Algorithm 1** Clinic Hospital Class

```
procedure INITIALIZE
        passivate() // holding passive
        myQueue = new DEVSQueue()
procedure DELTEXT(e, x)
        Continue(e)
    if phaseIs("passive") then
        for each i in x.getLength() do
            if messageOnPort(x, "TbarCode", i) then
                job = x.getValOnPort("TbarCode", i)
                processingTime = ((PatientEntity)job).getProcessingTime()
                currentJob = job
                holdIn("busy", processingTime)
            else if messageOnPort(x, "EbarCode", i) then
                job = x.getValOnPort("EbarCode", i)
                currentJob = job
                holdIn("busy", 20)
            else if messageOnPort(x, "RegularPatientbarCode", i) then
                job = x.getValOnPort("RegularPatientbarCode", i)
                currentJob = job
                holdIn("busy", 10)
    else if phaseIs("busy") then
        for each i in x.getLength() do
            if messageOnPort(x, "TbarCode", i) then
                myQueue.add(x.getValOnPort("TbarCode", i))
procedure DELTINT
    if myQueue.isEmpty() then
            passivate()
    else
            currentJob = (entity)myQueue.pop()
            holdIn("busy", 20)
procedure OUT
        m = new message()
        name = currentJob.getName()
    if name.charAt(0) ==' r' or name.charAt(0) ==' E' then
            conMedicalBill = makeContent("MedicalBill", currentJob)
            print("Contenttobesent : port : MedicalBillvalue : " + currentJob.getName())
            m.add(conMedicalBill)
    else if name.charAt(0) ==' T' then
            conHospital = makeContent("Hospital", currentJob)
            print("Contenttobesent : port : Hospitalvalue : " + currentJob.getName())
            m.add(conHospital)
        return m
procedure GETTOOLTIPTEXT
        return currentJob != null ? super.getTooltipText() + "number of patients in queue:" +
myQueue.size() + "my current job is:" + currentJob.toString() : "initial value"
```

The ClinicHospital-System class instantiates several atomic models and a transducer, each representing a distinct entity within the simulation:

RegularPatientGenerator, EmergencyGenerator, and TraumaGenerator are responsible for generating regular, emergency, and trauma patients, respectively. These components emulate patient arrivals based on predefined parameters, facilitating the dynamic influx of patients into the system.

Clinic-Hospital, Hospital, and MedicalBill represent the clinic, hospital, and medical billing centers, respectively. These components handle patient triage, treatment, and billing processes, ensuring efficient management of patient flow and resource utilization. The transducer component serves as an interface between the simulation model and external visualization tools, facilitating real-time monitoring and analysis of simulation dynamics. Couplings and Interactions:

The ClinicHospital-System class establishes couplings between system components to regulate the flow of information and patients within the simulation:

Couplings between patient generators and the clinic (Clinic-Hospital) enable the transmission of patient data based on their classification (regular, emergency, or trauma). Couplings between the clinic, hospital, and medical billing centers facilitate seamless transitions between different stages of patient care, including treatment and billing processes. Additionally, couplings between the medical billing center (MedicalBill) and the transducer enable the visualization of simulation results, enhancing the interpretability and analysis of simulation dynamics.

**Algorithm 2** Clinic Hospital System Algorithm

**procedure** ClinicHospitalSystem
    **add_outport**"out"
    $regular\_patientGenr \leftarrow RegularPatientGenerator(\text{"}regular\_patientGenr\text{"}, 5)$
    $emergencyGenr \leftarrow EmergencyGenerator(\text{"}emergencyGenr\text{"}, 10)$
    $traumaGenr \leftarrow TraumaGenerator(\text{"}traumaGenr\text{"}, 15)$
    $Clinic\_Hospital \leftarrow Clinic_{H}ospital(\text{"}Clinic\_Hospital\text{"})$
    $Hospital \leftarrow Hospital(\text{"}Hospital\text{"})$
    $MedicalBill \leftarrow MedicalBill(\text{"}MedicalBill\text{"})$
    $trand \leftarrow Transducer()$
    **add**$(traumaGenr)$
    **add**$(emergencyGenr)$
    **add**$(regular\_patientGenr)$
    **add**$(Clinic\_Hospital)$
    **add**$(Hospital)$
    **add**$(MedicalBill)$
    **add**$(trand)$
    **add_coupling**$(traumaGenr, \text{"}out\text{"}, Clinic\_Hospital, \text{"}TbarCode\text{"})$
    **add_coupling**$(emergencyGenr, \text{"}out\text{"}, Clinic\_Hospital, \text{"}EbarCode\text{"})$
    **add_coupling**$(regular\_patientGenr, \text{"}out\text{"}, Clinic\_Hospital, \text{"}Regular\_Patient\_barCode\text{"})$
    **add_coupling**$(Clinic\_Hospital, \text{"}Hospital\text{"}, Hospital, \text{"}TbarCode\text{"})$
    **add_coupling**$(Hospital, \text{"}MedicalBill\text{"}, MedicalBill, \text{"}TbarCode\text{"})$
    **add_coupling**$(Clinic\_Hospital, \text{"}MedicalBill\text{"}, MedicalBill, \text{"}E\_RP\_Code\text{"})$
    **add_coupling**$(MedicalBill, \text{"}out\text{"}, self, \text{"}out\text{"})$
    **add_coupling**$(MedicalBill, \text{"}out\text{"}, self, \text{"}out\text{"})$
    **add_coupling**$(Clinic\_Hospital, \text{"}Hospital\text{"}, trand, \text{"}ariv\text{"})$
    **add_coupling**$(MedicalBill, \text{"}out\text{"}, trand, \text{"}solved\text{"})$

### 4.5 Package HandsOnProject: RegularPatientGenerator

The RegularPatientGenerator class is a vital component of our hospital management simulation, responsible for generating regular patients and regulating their influx into the system. Let's dissect its functionalities and operational mechanisms:

**Initialization:**

The constructors RegularPatientGenerator() and RegularPatientGenerator(String name, double generateTime) initialize the generator, allowing for the specification of parameters such as the generator's name and inter-generate time (time between successive patient arrivals). In addition, the generator adds an output port named "out" through which patient arrival information is transmitted.

**Inter-Generate Time Calculation:**

During initialization, the generator calculates the precise time for the arrival of the next regular patient based on the specified inter-generate time. It utilizes a random number generator (myrand) to introduce variability into patient arrivals, ensuring a stochastic arrival pattern reflective of real-world scenarios. The calculation takes into account the variability in inter-arrival times, ensuring a realistic distribution of patient arrivals.

**State Initialization:**

The initialize() method initializes the generator's state, setting up the initial conditions for patient generation. It calculates the arrival time for the first patient based on the inter-generate time and holds the generator in the "active" state until the arrival of the next patient.

**External Input Handling:**

The deltext(double e, message x) method handles any external input encountered during simulation execution. In the case of the RegularPatientGenerator, it simply continues the current state without perturbation upon receiving external input.

**Internal Time Advancement:**

Upon internal time advancement (deltint()), the generator recalculates the arrival time for the next regular patient based on the inter-generate time and introduces variability using the random number generator. This adaptive mechanism ensures a realistic simulation environment with dynamic patient arrivals.

**Output Generation:**

During the "active" phase, the generator produces output messages (out()) containing pertinent information about the newly generated regular patient. Each message includes a unique patient identifier (Id) adorned with the prefix "regular-patient-barCode", facilitating unambiguous identification of regular patients within the simulation environment. These messages are transmitted via the designated "out" port for further processing within the simulation.

Algorithm 3 represents the pseudocode of this class.

### 4.6 Package HandsOnProject: EmergencyGenerator

The EmergencyGenerator class is a crucial component of our hospital management simulation, tasked with generating emergency patients and regulating their influx into the system. Let's delve into its functionalities and operational mechanisms:

**Initialization:**

The constructors EmergencyGenerator() and EmergencyGenerator(String name, double generateTime) initialize the generator, allowing for the specification of parameters such as the generator's name and inter-generate time (time between successive patient arrivals). Additionally, the generator adds an output port named "out" through which patient arrival information is transmitted.

**Inter-Generate Time Calculation:**

**Algorithm 3** RegularPatientGenerator Class

```
1:  procedure RegularPatientGenerator
2:      interGenerateTime ← 20, Id ← 0, myrand ← rand(1234)
3:      procedure RegularPatientGenerator
4:          super("RegularPatientGenerator", 20), addOutport("out")
5:      procedure initialize
6:          myrand ← rand(1234)
7:          lowBound ← max(10, interGenerateTime − 20)
8:          nextPatient ← myrand.uniform(lowBound, interGenerateTime + 10)
9:          holdIn("active", nextPatient)
10:     procedure deltint
11:         lowBound ← max(10, interGenerateTime − 20)
12:         nextPatient ← myrand.uniform(lowBound, interGenerateTime + 10)
13:         holdIn("active", nextPatient), Id ← Id + 1
14:     procedure out
15:         if phaseIs("active") then
16:             m ← new message()
17:             con ← makeContent("out", new entity("regular_patient_barCode_" + Id + ""))
18:             m.add(con)
19:             return m
```

During initialization, the generator calculates the precise time for the arrival of the next emergency patient based on the specified inter-generate time. It utilizes a random number generator (myrand) to introduce variability into patient arrivals, ensuring a stochastic arrival pattern reflective of real-world scenarios. The calculation takes into account the variability in inter-arrival times, ensuring a realistic distribution of patient arrivals.

**State Initialization:**

The initialize() method initializes the generator's state, setting up the initial conditions for patient generation. It calculates the arrival time for the first patient based on the inter-generate time and holds the generator in the "active" state until the arrival of the next patient.

**External Input Handling:**

The deltext(double e, message x) method handles any external input encountered during simulation execution. In the case of the EmergencyGenerator, it simply continues the current state without perturbation upon receiving external input.

**Internal Time Advancement:**

Upon internal time advancement (deltint()), the generator recalculates the arrival time for the next emergency patient based on the inter-generate time and introduces variability using the random number generator. This adaptive mechanism ensures a realistic simulation environment with dynamic patient arrivals.

**Output Generation:**

During the "active" phase, the generator produces output messages (out()) containing pertinent information about the newly generated emergency patient. Each message includes a unique patient identifier (Id) adorned with the prefix "EbarCode", facilitating unambiguous identification of emergency patients within the simulation environment. These messages are transmitted via the designated "out" port for further processing within the simulation.

By meticulously implementing the EmergencyGenerator class, our simulation model accurately replicates the arrival of emergency patients within a healthcare system, enabling comprehensive analysis and optimization of patient flow and resource allocation.

### 4.7 Package HandsOnProject: TraumaGenerator

The TraumaGenerator class plays a pivotal role in our hospital management simulation, simulating the arrival of trauma patients and regulating their influx into the system. Let's explore its functionalities and operational mechanisms:

**Initialization:**

The constructors TraumaGenerator() and TraumaGenerator(String name, double generateTime) initialize the generator, allowing for the specification of parameters such as the generator's name and inter-generate time (time between successive patient arrivals). Additionally, the generator adds an output port named "out" through which patient arrival information is transmitted.

**Inter-Generate Time Calculation:**

During initialization, the generator calculates the precise time for the arrival of the next trauma patient based on the specified inter-generate time. It utilizes a random number generator (myrand) to introduce variability into patient arrivals, ensuring a stochastic arrival pattern reflective of real-world scenarios. The calculation takes into account the variability in inter-arrival times, ensuring a realistic distribution of patient arrivals.

**State Initialization:**

The initialize() method initializes the generator's state, setting up the initial conditions for patient generation. It calculates the arrival time for the first patient based on the inter-generate time and holds the generator in the "active" state until the arrival of the next patient.

**External Input Handling:**

The deltext(double e, message x) method handles any external input encountered during simulation execution. In the case of the TraumaGenerator, it simply continues the current state without perturbation upon receiving external input.

**Internal Time Advancement:**

Upon internal time advancement (deltint()), the generator recalculates the arrival time for the next trauma patient based on the inter-generate time and introduces variability using the random number generator. This adaptive mechanism ensures a realistic simulation environment with dynamic patient arrivals. Additionally, it increments the unique identifier (Id) for each patient, facilitating their individual tracking within the simulation.

**Output Generation:**

During the "active" phase, the generator produces output messages (out()) containing pertinent information about the newly generated trauma patient. Each message includes a unique patient identifier (Id) adorned with the prefix "TbarCode", along with additional information such as processing time, urgency level, and patient ID. These messages are transmitted via the designated "out" port for further processing within the simulation.

By meticulously implementing the TraumaGenerator class, our simulation model accurately replicates the arrival of trauma patients within a healthcare system, enabling comprehensive analysis and optimization of patient flow, resource allocation, and emergency response protocols.

## 5 DEMO

In Figure 3, we present a demonstration of our Clinic-Hospital System.

At the onset of our scenario, we envision an infinite timeline within a medical setting, where patients arrive sporadically for treatment. We have three categories of patients: regular, emergency, and trauma patients.

**Regular Patient (First Arrival)**:

This patient arrives at a random time, characterized by a time of 13.23 units on our infinite timeline. Upon arrival, the regular patient undergoes processing with a duration set at 10 times the unit. Subsequently, at 23.23 units, the processing concludes, and the patient's barcode, denoted as regular-patient-barCode-0, is generated and dispatched to the medical billing system.

revise the highlighted section in Figure 2 with a yellow color.

```
time: 13.23291080145463
time: 18.23291080145463
Content to be sent port: MedicalBill value: regular_patient_barCode_0
time: 23.23291080145463 (regular patient) time: 23.23291080145463 (trauma patient)
time: 27.989699356051588
time: 37.98969935605159
Content to be sent port: Hospital value: TbarCode_0
time: 38.23291080145463
time: 42.27764165508576
Content to be sent port: MedicalBill value: TbarCode_0
time: 43.23291080145463
time: 46.23291080145463
time: 47.98969935605159
Content to be sent port: MedicalBill value: regular_patient_barCode_2
time: 52.27764165508576
time: 54.56880818039911
time: 57.27764165508576
time: 66.24857019310758
Content to be sent port: Hospital value: TbarCode_1
time: 72.27764165508576
time: 74.56880818039912
time: 77.26794410287324
Content to be sent port: MedicalBill value: TbarCode_1
time: 77.27764165508576
time: 80.27764165508576
Content to be sent port: Hospital value: TbarCode_2
TbarCode_2 arrived at time:87.27764165508576.
time: 87.27764165508576
time: 89.00248124657163
time: 91.24857019310758
Content to be sent port: MedicalBill value: TbarCode_2
time: 92.27764165508576
time: 94.56880818039912
TbarCode_2 is finished at time:95.27764165508576.
time: 95.27764165508576
Content to be sent port: MedicalBill value: regular_patient_barCode_6
time: 99.00248124657163
the total service time for TbarCode_2 is: 8.0
arriving TbarCode_2: 3  finshed TbarCode_2:5
time: 100.0
```

Fig. 2. Experiment Results and analysis

**Emergency Patient:**

Following the regular patient's arrival, an emergency patient arrives at 18.23 units on our infinite timeline. Trauma Patient (First Arrival): Shortly after, at 23.23 units, a trauma patient enters the medical facility. The trauma patient's processing duration is set at 15 times the unit. At 38.23 units, the processing for the trauma patient concludes, and the corresponding barcode, labeled

as TBarcode-0, is generated and relayed to the medical billing system. Regular Patient (Second Arrival): Continuing our timeline, another regular patient arrives at 27.98 units. Similar to the previous regular patient, this one also undergoes processing with a duration of 10 times the unit. At 37.98 units, the processing for the second regular patient is completed, and their barcode is generated and dispatched to the medical billing system.

revise the highlighted section in Figure 2 with a bright green color.

**Trauma Patient (Transfer to Hospital):**

At 38.23 units, the trauma patient from earlier in our scenario is deemed in need of transfer to a hospital for further treatment, possibly including surgery. This transfer process involves additional processing, with a duration set at 15 times the unit. Following an initial processing time of 5 minutes, the trauma patient's barcode is generated and relayed to the medical billing system at 43.23 units. We've allocated a 3-minute slot for processing the medical bill. Consequently, the trauma patient will be discharged at precisely 46.23.

revise the highlighted section in Figure 2 with a turquoise color.

We strategically introduced a 5-unit "busy time" within the hospital's schedule to hasten outcome observation. This adjustment simulated rapid patient transfer to the surgery room post-admission. After surgery, a 5-times a unit interval preceded communication with stakeholders, enhancing efficiency evaluation in our study.

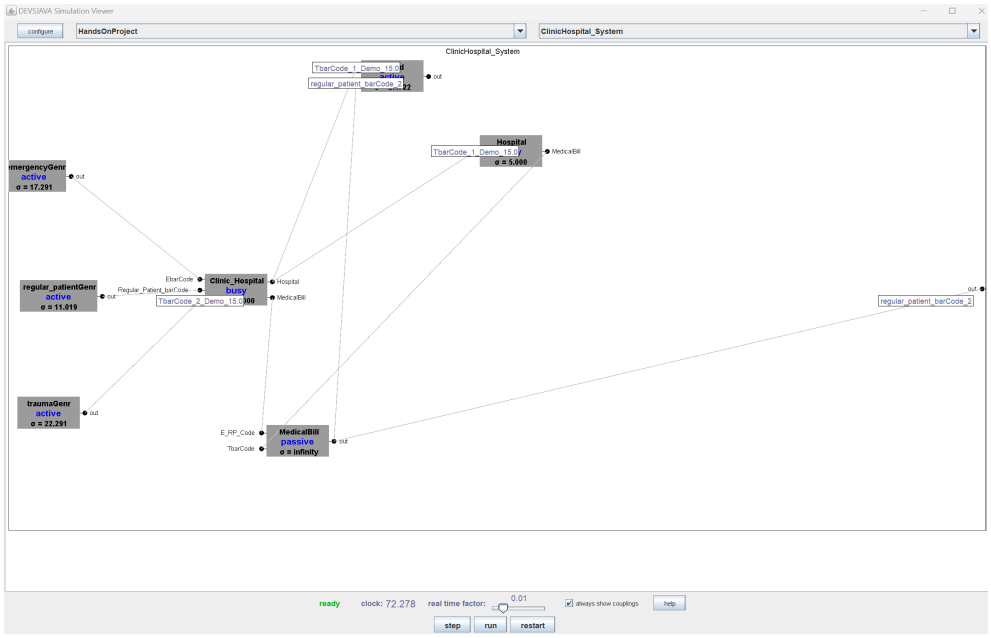


Fig. 3. Clinic-Hospital System

In Figure 3, we introduce a depiction of the demographic flow within our medical scenario. The temporal framework is anchored at 72.27 units. At this juncture, significant events unfold:

**TBarcode-2 Generation:**

At the designated time of 72.27 units, the generation of TBarcode-2 occurs, signifying the completion of processing for a subsequent trauma patient.

**Transfer of Trauma Patient (TBarcode-1) to Hospital:**

Concurrently, the trauma patient corresponding to TBarcode-1 is deemed necessitous of hospital transfer for escalated treatment, potentially including surgical intervention. The transfer protocol includes an initial processing period within the hospital setting, encompassing 5 units of time to facilitate swift comprehension of the subsequent analyses.

**Dispatch of Regular Patient-barcode-2:**

As part of the continuum of medical procedures, the barcode associated with the second regular patient undergoes dispatch subsequent to pertinent medical engagements. This temporal delineation highlights the orchestrated progression of events within our academic investigation. By integrating such meticulous chronology, we aim to afford clarity and cogency to our scholarly discourse, enhancing the comprehensibility and scholarly rigor of our narrative.

## 6 EXPERIMENT DESIGN, EXPERIMENT RESULTS AND ANALYSIS

By integrating a transducer into our DEVS (Discrete Event System Specification) framework, we introduce a crucial element tasked with converting input events into output events according to predetermined rules or conditions.

In the DEVS Java environment, a transducer takes shape as a Java class, defining the handling of input events and their conversion into corresponding output events. This tool allows us to encapsulate complex behaviors or systems, enabling specific responses or actions to be triggered by inputs at the designated time, such as 100.

After 100 iterations of the unit, we obtained the following outcome: The total service time for **TBarcode-2**, totaling **8.0 units**, encompasses the duration from its commencement to its conclusion, emphasizing the temporal commitment necessary for processing and managing this particular patient case.

revise the highlighted section in Figure 2 with a pink color.

**TBarcode-2**'s arrival at time **3** signifies the commencement of its processing within the system. Following this, the completion of its service at time 5 underscores the efficient manner in which the medical procedures related to this patient were executed.

This bar chart In Figure 4 illustrates the numbers of each patient type after processing the medical bill in 1000 simulations. Noticeably, the count of trauma patients surpasses the others, attributed to their priority status in the queue.

## 7 CONCLUSION AND DISCUSSIONS

In conclusion, our project aims to optimize healthcare delivery within global public health systems despite increasing demands and financial constraints. Through corporate performance methodologies and Devs Java technology, the Ministry of Health seeks to enhance hospital operations and prioritize patient care.

We focus on refining healthcare delivery for trauma and emergency cases by introducing specialized queues and real-time patient tracking. Devs Java optimizes resource allocation and enables predictive analytics for queuing systems.

Our tool aims for efficiency, reliability, security, customization, and scalability, offering cost-effective, interoperable, and compliant solutions with real-time monitoring.

The DEVS model strengthens our simulation efforts, enabling systematic analysis and refinement. Experiment results demonstrate our project's efficacy in enhancing healthcare processes.

Overall, our project represents a holistic and innovative approach to improving healthcare delivery worldwide, navigating financial constraints while aiming for optimal health outcomes.

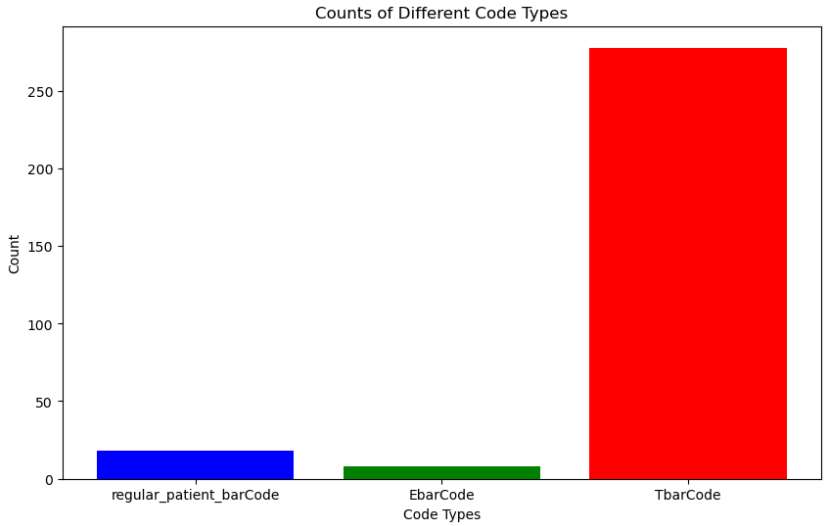


Fig. 4. Clinic-Hospital System-BarCode Types