This document provides a detailed technical explanation of the **Intelligent Urban Traffic Control System** implemented in the `final_project` package. The system uses **Object-Oriented Extended Timed Petri Nets (OETPN)** to model and control a complex urban environment.

---

1. System Overview

The program is designed as a distributed simulation where various urban components (intersections, roundabouts, and stations) operate as independent Petri Net entities. These entities communicate using TCP/IP network ports to exchange traffic light signals and vehicle data.

Key Architectural Features:

- **Modularity:** Each component is an autonomous Petri Net object.
- **Networking:** Components use `DataTransfer` objects to send information across local ports (e.g., 1081 for Intersection 1, 1091 for Controller 1).
- **Concurrence:** The `RunAll.java` class launches every component in its own execution thread, allowing simultaneous processing of traffic flow across the entire map.

---

2. Traffic Control Logic (Controllers)

The system features two main controllers that manage signal timings dynamically.

Controller 1 (Dynamic 4-Lanes)

Managed in `Controller1.java`, this component controls **Intersection 1**.

- **State Machine:** It cycles through nine primary states representing various signal combinations (e.g., `r1r2r3r4` for all red, `g1r2r3r4` for green on lane 1).
- **Dynamic Delays:** The controller does not use static timers. Instead, a control transition named `tf` monitors input places (`in1` to `in4`) which receive occupancy data from the intersection sensors.
- **Guard Mapping:** Based on which lanes are occupied, the controller selects a specific delay constant (e.g., `Two`, `Five`, `Eleven`) and applies it as a `DynamicDelay` to the active green phase. For example, if only lane 1 has cars, its green light duration is extended to 11 units while others are minimized.

Controller 2 (Dynamic 3-Lanes)

Similar to Controller 1, `Controller2.java` manages **Intersection 2** using three input sensors and signal phases.

---

3. Intersection Modeling

The intersections represent the physical layer where vehicles (tokens) move through queues.

- **Intersection 1 (Crossroads):** This 4-way junction includes a specialized **Bus Lane**. It uses `TransitionCondition.IsPriority` to detect buses, allowing them to trigger transitions differently than regular cars.
- **Sensors:** Each lane has an async transition (e.g., `T_s1`) that acts as a sensor. When a lane queue is full or has cars, it sends a "green" request over the network to the corresponding Controller port.
- **Flow Guards:** Transitions like `T_e1` (Exit) only fire if the input place `P_TL1` contains a "green" string, ensuring vehicles obey the traffic lights.

---

4. Specialized Urban Components

The Roundabout

The `Roundabout.java` component models a 3-lane circular junction.

- **Circular Flow:** It uses three main queues (`P1`, `P2`, `P3`) connected in a loop.
- **Targeting:** Vehicles are moved between segments using `PopElementWithTargetToQueue`. This ensures cars navigate the roundabout based on their internal destination data until they reach their exit buffer (`P4`, `P6`, or `P9`).

Bus and Taxi Stations

These components simulate public transport stops where vehicles leave the main road, stop for a duration, and re-enter.

- **Bus Station:** Specifically filters for buses using `HaveBus` conditions. It includes a station dwell time modeled by a delay of 10 units on the `T_es` transition.
- **Taxi Station:** Implements a coordination logic where a taxi can only exit the station (`P_Station`) if there is a passenger available in the `UserQ`. It uses `TransitionOperation.PopTaxiToQueue` to handle this specific interaction.

Pedestrian System

The `PedestrianController.java` manages a pedestrian crossing.

- **Request Logic:** A pedestrian "button" press is modeled by a token in `P_Request`.

- **Safety Interlock:** The Pedestrian Controller receives the request and sends a `goGreen` command to the pedestrian light while communicating with the main traffic controllers to ensure vehicle lanes are red during the walking phase.

---

5. Component Integration

The entire system is integrated via the `RunAll.java` class, which initializes the following network of Petri Nets:

Component	Logic Port	Controller Port	Key Class
Intersection 1	1081	1091	`Intersection1.java`, `Controller1.java`
Intersection 2	1082	1092	`Intersection2.java`, `Controller2.java`
Roundabout	1083	N/A	`Roundabout.java`
Bus Station	1084	N/A	`BusStation.java`
Taxi Station	1085	N/A	`TaxiStation.java`
Pedestrian	1086	1096	`PedestrianController.java`

Exportă în Foi de calcul

This multi-port architecture allows the simulation to scale; one could theoretically run each component on a different computer as long as the "localhost" IP is updated to the correct network address.