Information, Computing and Intelligent Systems, 2025, No. 6, 132 – 151

UDC 004.94, 628.477 https://doi.org/10.20535/2786-8729.6.2025/333736

WASTE MANAGEMENT MODEL WITH TIMED COLORED PETRI NETS

Hryhorii Rozhkov *
National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, Ukraine
https://orcid.org/0009-0009-5343-8974

Klymenko Iryna
National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, Ukraine
http://orcid.org/0000-0001-5345-8806

*Corresponding author: hryhoriirozhkov@gmail.com

Waste management is a key element in the functioning of modern cities. This paper presents a new model of a waste collection system, described as a discrete-event system (DES), implemented using Timed Colored Petri Nets (TCPNs) in combination with an integrated Python server. The model is developed with consideration of container filling dynamics and variable routes, which ensures alignment with real urban conditions.

A key element of the developed model is the interface of a vehicle routing problem with capacity constraints, multiple trips, and time windows (MTCVRPTW), which enables vehicles to service containers multiple times during a scheduled period while adhering to volume and time restrictions. The model supports configuration of parameters such as operational delays, container filling and overflow volumes, and vehicle load capacity. The simulation is implemented in CPN IDE using time series as input data, partitioned for efficient processing. Information about container filling levels and road conditions is periodically updated during real-time simulation, enhancing scalability and performance. The model generates event logs—movement, unloading, overflow, and servicing—which are processed by Python scripts to calculate performance metrics.

The main performance metrics of the waste collection system were defined, including route distance and time, unloading efficiency, container overflow volume, servicing efficiency, and deviations of planned routes from the schedule.

To demonstrate the operation of the model, an experiment was conducted using synthetic data approximating real-world conditions. The locations of 10 containers, unloading points, and depots were determined using the Google My Maps service based on coordinates of real objects in Kyiv. Realistic route distances and travel times were generated using the Google Distance Matrix API. The MTCVRPTW algorithm for two vehicles scheduled two trips per week according to static routes. The simulation of the model generated event logs, which were then used to calculate performance metrics. The analysis of these metrics revealed significant limitations of static route planning and highlighted the need for adaptive strategies that account for the actual state of containers and traffic.

The proposed model is a flexible tool for evaluating, analyzing, and improving waste collection strategies in cities.

Keywords: waste management, waste management optimization, discrete event systems, Petri nets, vehicle routing problem.

1. Introduction

Waste Management (WM) has become an increasingly critical challenge in modern urban environments due to rapid urbanization and population growth. The resulting surge in waste generation demands sophisticated planning and execution to maintain public health and environmental standards. Key objectives of an effective WM system include timely servicing of waste containers to meet predefined service levels, minimizing container overflows to prevent environmental pollution, and optimizing vehicle routes to reduce fuel consumption and emissions.

However, modern WM systems are inherently complex and dynamic. Factors such as new container service contracts, fluctuating vehicle availability, varying road conditions, and stochastic events including vehicle breakdowns and unpredictable changes in container fill levels introduce significant planning uncertainty. Additionally, geospatial elements like multiple depots and unload stations, as well as heterogeneity in both container and vehicle types (e.g., residential, commercial, hazardous waste), further complicate the routing and scheduling process. Some vehicles may support multi-type waste collection, while others are restricted to specific categories, and service times can vary considerably depending on waste type.

Manual planning remains the prevailing method in many cities, including Kyiv, requiring substantial human resources and frequent route adjustments. Yet, this approach is inefficient and prone to suboptimal outcomes. Automating the process through an appropriate formulation of the Vehicle Routing Problem (VRP) tailored to the specific requirements of WM offers a scalable and adaptive alternative.

Given that WM systems evolve in response to discrete, event-driven changes, modeling them as DES is a natural and effective approach. Colored Petri Net (CPN), combined with Python-based simulation, provide a robust framework for capturing dynamic interactions among container states, vehicle movements, and network conditions in real-time.

Considering the aforementioned aspects, a simulation-based prototype model becomes essential to validate such automation. This model should enable the evaluation of alternative collection strategies by generating objective performance metrics such as total travel distance, service frequency, and operational workload under varied conditions. The framework must provide a tool for evaluating route optimization techniques that enhance operational efficiency, reduce costs, and support sustainable urban WM.

2. Literature review and problem statement

The Kyiv municipal WM company [1] is a prime example of the city's WM system. The company runs a fleet of around 250 vehicles of various types, spread across six depot centers and 10 unloading facilities, each designed to handle specific waste categories. The waste collection infrastructure includes a range of container types and designs. Waste collection is organized on weekly route schedules. However, unexpected events like vehicle breakdowns, driver absences, and container overflows are currently handled manually by human dispatchers. The company also offers an interactive mapping service that shows the geographical distribution of container sets. Each set is marked with its location, the number and type of containers, their weekly accumulated load, and the associated service schedule. Most containers are serviced on a fixed-day schedule. In contrast, some containers operate on a load-based collection principle; these are assumed to be equipped with smart container sensors that periodically report their fill levels to the system.

To choose the most appropriate mathematical tool to model the WM system several mathematical models were analyzed, which are presented below.

Finite State Machines (FSMs) serve as a basic modeling tool for systems with a finite number of states and transitions. While FSMs are simple and easy to understand, they lack the expressive power to effectively model complex, concurrent systems. In contrast, Petri Nets can explicitly represent concurrency and synchronization, making them more suitable for modeling DES in WM. However, FSMs can still be effective for modeling simple sequential processes within a Petri Net framework [2].

Markov Chains (MCs) are probabilistic models used to analyze systems with uncertain transitions. They are helpful for modeling stochastic processes, such as demand uncertainty in VRP. However, MCs are limited in their capacity to represent concurrency and synchronization, both of which are critical in WM systems. Petri Nets, especially stochastic Petri Nets, can model both deterministic and stochastic behavior, making them a more comprehensive tool for these applications [3] [4].

Queueing Models (QMs) analyze systems with waiting lines, such as customers awaiting service. They are especially beneficial for assessing performance metrics like waiting times, queue lengths, and service rates. While QMs are effective for analyzing specific aspects of WM systems, such as the waiting time for waste collection vehicles, they do not capture the overall system dynamics, including routing and scheduling. Petri Nets, in contrast, can integrate queueing theory within their modeling framework to provide a more holistic view of the system [5].

Process Algebras (PAs) are formal languages designed for modeling and analyzing concurrent systems that provide a mathematical framework for specifying and verifying system behavior. Although PAs are powerful tools for theoretical analysis, they can be challenging to implement in practical modeling scenarios. Petri Nets, with their graphical representation and intuitive semantics, often provide a more accessible option for practitioners, leading to a user-friendly approach to modeling DES [6].

Simulation-based Models (SBMs) are widely utilized for analyzing and optimizing complex systems. They allow for the simulation of system behavior across various scenarios, making them particularly beneficial for assessing the impact of different routing strategies or WM policies. Petri Nets can be combined with simulation tools, such as ExtendSim, to deliver a more detailed and accurate analysis of system performance. For example, a study on container terminal handling systems integrated Petri Nets with simulation tools to optimize the handling process and enhance efficiency [7].

Petri Nets are a well-established graphical and mathematical formalism within the broader field of DES modeling, particularly effective for describing systems with concurrency, synchronization, and resource sharing. Petri Net models provide formal analysis techniques for properties such as reachability, liveness, and boundedness. They comprise places (representing states or conditions) and transitions (representing events or actions), linked by arcs that define the flow of tokens (representing resources or entities) through the system. Several extensions of standard Petri Nets have been developed to enhance their modeling capabilities. Timed Petri Nets (TPNs) add transition delays for modeling temporal aspects, while Colored Petri Nets (CPNs) assign data types to tokens and support functional inscriptions. Their combination, Timed Colored Petri Nets (TCPNs), enables modeling of complex real-time systems with concurrency and conflict resolution. In contrast, Hierarchical Petri Nets (HPNs) introduce an architectural mechanism for modularity by allowing transitions to be refined into sub-nets, thus supporting structured model decomposition. This hierarchical principle can also be applied within TCPN. It is common in the literature that the term CPN implicitly refers to TCPN, since most practical applications rely on both color and time extensions; therefore, in this article the terms CPN and TCPN are used interchangeably.

In the context of WM, TPNs have been effectively employed to model and analyze various system aspects. For instance, Stochastic Petri Nets have been utilized to address uncertainties in demand and vehicle routing, while CPNs have been used to model complex logistics networks featuring multiple vehicle types and dynamic routing requirements [3] [8].

The matrix analysis of the DES modeling tools is presented in Table 1, where "++" denotes strong support or native capability, and "+" denotes partial or indirect support.

The VRP is a critical component of WM optimization, as it directly influences the efficiency and effectiveness of collection operations in urban settings. A comprehensive overview of VRPs in the context of WM is provided in [9]. The authors have proposed solution methods to VRPs and formulated following constraints, which served as the basis for defining the requirements and limitations for the development of the VRP algorithm interface in the present study.

Capacity Constraints limit the amount of waste that can be collected by a vehicle, ensuring that vehicle does not overload during collection.

Demand Constraints ensure that all customer demands are met, meaning that service level is met and all containers are services with enough frequency.

Labour Constraints impose limits on the workforce available for waste collection. This includes restrictions on the number of working hours for employees and the duration of shifts for collection crews. Such constraints can affect the overall efficiency of waste collection operations.

Feasibility Constraints prevent the formulation of infeasible solutions, they ensure that all customers are visited in a logical sequence and routes are feasible and practical. An example includes constraints that eliminate subtours, which are routes that visit only a subset of customers, also it must be guaranteed that vehicle unloads before returning to the depot.

Driver Lunch Break Constraints account for mandatory breaks for drivers, which can affect the scheduling and timing of waste collection routes. Properly incorporating these breaks into route planning is essential to avoid conflicts with time windows.

Table 1. Comparison of DES models

Criterion	**FSMs**	**MCs**	**QMs**	**PAs**	**SBMs**	**TPNs**
Concurrency modeling				+	+	++
Stochastic behavior support		++	+	+	++	++
Timing constraints support		+	++		+	++
Sequential process modeling	++	+	+	+	+	+
Graphical representation	+				+	++
Formal verification capability	+	+	+	++		++
Scalability to complex systems		+	+	+	+	+
Integration with simulation engines			+		++	++
Ease of implementation	++	+	+		+	+
Support for routing and scheduling				+	+	++

A Smart WM model using Stochastic Petri Nets is proposed in [10], simulating random waste drop-offs and enabling the evaluation of various collection strategies. The model supports parameterization to identify optimal collector levels and reduce vehicle visits, based on assumptions about average drop-off frequency and travel time. It emphasizes real-time inventory control using sensor data but lacks integration of time-series inputs or detailed routing logic. While stochastic timing allows to model probabilistic behavior, it restricts the ability to simulate specific temporal scenarios. A major limitation is scalability: a system with 100 collectors leads to over 30,000 places and 70,000 transitions, creating significant graphical complexity. This hinders formal analysis and makes it difficult for users to embed custom algorithms such as ILP-based VRP models, reducing the model's flexibility for tailored applications.

A practical and effective algorithm for WM route planning, accommodating multiple trips per period, service frequency obligations, and varying road conditions, is the Multi-Trip Vehicle Routing Problem with Time Windows (MT-VRPTW) presented in [11]. This approach extends the classic VRP by allowing vehicles to perform multiple trips within a planning horizon, such as a week, and incorporates time windows for servicing containers. The paper demonstrates that adopting a multi-trip vehicle routing strategy with a one-week planning horizon leads to better performance in logistics operations. They proved that weekly scheduling reduces the total travel distance compared to traditional daily scheduling. The weekly plan also resulted in a 33.52% decrease in the number of vehicles required.

The study [12] introduces Colored Petri Net Markup Language (CPN ML) and its foundational role in modeling systems using Petri Nets. CPN ML is a powerful language that facilitates the declaration of color sets, variables, functions, and constants, which are essential for defining the attributes of net elements. The chapter explains that Petri Nets are represented as bipartite directed graphs, consisting of places (depicted as circles or ovals) and transitions (represented as bars). In this context, tokens serve as dynamic objects that move between places as transitions fire, allowing for the modeling of complex systems. The chapter emphasizes the importance of color sets in distinguishing different types of tokens, which enhances the expressiveness of the models.

Additionally, it highlights the integration of CPN ML with CPN Tools, a simulation system developed at the University of Aarhus, which supports the creation, simulation, and analysis of Petri Net models in various applications, particularly in telecommunications.

The current literature shows that there are still important gaps in how Petri Net models are used to support VRP in WM. While Petri Nets are useful for building dynamic models that simulate waste collection and help identify inefficiencies, existing approaches often lack a flexible and practical way to design, test, and compare different VRP algorithms in realistic urban settings. Although route optimization can reduce travel distance, fuel use, and emissions, it remains difficult to apply Petri Net models effectively in real-world waste collection scenarios, especially in cities with complex layouts. One major gap is that many models do not include VRP methods that are specific to the challenges of waste collection. Another issue is that it's often hard to insert or change routing algorithms in these models, making it less convenient to test new strategies. Additionally, there is a lack of well-defined and useful performance metrics – such as how often containers overflow, how well vehicles are used, or how long services take – which are needed to evaluate and improve WM systems. So, this article is devoted to creating a better Petri Net-based model that supports flexible algorithm integration for evaluating WM systems, defining objective metrics to measure system performance generated through simulation.

3. The aim and objectives of the study

This study aims to develop an improved Petri Net-based simulation model for urban WM systems that supports flexible integration of routing algorithms and enables formal evaluation of system performance. The proposed model addresses limitations in existing approaches by incorporating real-world constraints such as road congestion, service-level agreements, vehicle resource availability, and multi-trip planning. It also enables the simulation of realistic operational scenarios and the generation of objective performance metrics.

The objectives of the study are as follows:

1. To design and implement a simulation model that supports experimentation with different routing algorithms under practical constraints;

2. To formalize a set of performance metrics for evaluating vehicle routing and overall system efficiency;

3. To perform experimental analysis using the developed model to demonstrate its applicability in assessing collection strategies and identifying system bottlenecks.

4. The study materials and methods of modeling the WM problem

4.1 Petri Net modeling

In this article the TCPN, which includes TPN functionality, is proposed. Events, such as vehicle drive, park, serve, and unload operations; container fill levels change; and route conditions change cause concurrency in the WM system. Timed transitions are used to model state changes in the concurrent WM system. The time is divided into discrete points with a minimal period of 1 second. Places represent system states, and transitions correspond to state changes. A transition is executable when all its input places contain tokens available at the current simulation time and, if applicable, a guard condition evaluates to true. Transitions may introduce delays on output arcs to simulate non-instantaneous processes. The simulation is governed by a global, non-decreasing model time represented in discrete seconds. When no transitions are immediately enabled, time advances discretely to the earliest future point at which a transition becomes enabled. The integration of TPN with route optimization algorithm is a natural fit for modeling WM.

4.2 Modeling tools and services

Google My Maps is an interactive mapping service that enables the manual marking of locations on a real-world map. It has been used to annotate the geographic positions of waste containers, depot sites, and unloading stations. The service supports exporting maps in the kml format, an XML-based structure that can be parsed using custom Python scripts to extract and process location data for simulation inputs.

The modeling and simulation of the system were performed using CPN IDE, a modernized successor to the discontinued CPN Tools. CPN IDE provides a graphical interface for building and simulating TPN models. It supports modular design through sub-nets and operates in two modes: an editor for model construction and a simulation mode that supports both animated (step-by-step) and fast-forward (multi-step) execution. For debugging purposes, breakpoint monitors and fast-forward simulations were employed to analyze model behavior in response to specific state transitions.

While CPN ML, the functional language used within CPN IDE, is suitable for defining TPN logic, it lacks expressive support for complex algorithmic tasks such as vehicle routing or data analysis. Therefore, an external service integration approach, similar to an approach described in [13], was adopted to couple CPN ML with a Python 3 socket server. This enabled efficient prototyping of routing algorithms and metric collection. A known limitation of CPN IDE is its inability to report compilation errors originating from helper functions defined outside action block. Such functions were temporarily moved within action blocks during debugging, as a workaround.

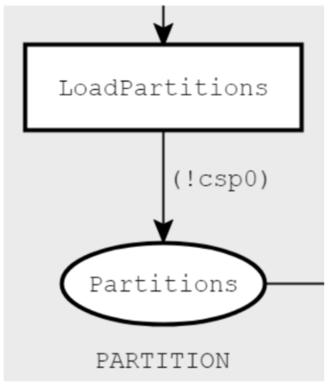
Python 3 has been employed as the primary language for route computation, simulation integration, and metric extraction from simulation logs. The pandas library was used extensively to transform and analyze event data produced by the model, enabling the computation of detailed performance metrics. Additionally, the Plotly graphing library has been utilized to produce interactive diagrams illustrating vehicle state timelines and route execution patterns.

4.3 Input tokens loading from files

There is limited documentation about loading token multisets from file in CPN ML. The following approach was utilized to initialize timed and untimed token multisets from files. There is an example provided for loading `PARTITION` tokens from file which are needed to load partitioned datasets into the WM model. Figure 1 defines a *LoadPartitions* transition along with CPN ML action code block and CPN ML declarations.

```
colset PARTITION = time timed;
globref csp0 = empty: PARTITION timed ms;

action
let
  val arq_ent = TextIO.openIn(
    “partitions.txt”
  );
  fun read_all_tokens file =
    case TextIO.inputLine file of
         NONE => empty
       | SOME line =>
           let
             val input_stream = TextIO.openString line
             val token = PARTITION'timed.input_ms input_stream
             val _ = TextIO.closeIn input_stream
           in
             token ++ read_all_tokens file
           end;
  val data = read_all_tokens arq_ent;
  val _ = TextIO.closeIn arq_ent;
in
  csp0 := data;
  ()
end;
```



```
(* partitions.txt format
1`0@0
1`10800@10800
...
*)
```

Fig 1. An approach to load PARTITION multiset from file

Action block loads tokens from an external text file and assigns it to the global reference variable `csp0`. The input file must contain one token per line. The file is opened for reading, and a recursive function *read_all_tokens* processes each line by converting it into a timed `PARTITION` token using `PARTITION'timed.input_ms` (for untimed multisets `PARTITION.input_ms` should be used instead). The resulting tokens are combined into a multiset. Once all tokens are read, the input stream is closed, and the constructed multiset is assigned to `csp0`. The token multiset is read from that global variable and tokens are created in *Partitions* place.

4.4 Input data partitioning

The CPN IDE simulation environment exhibits notable performance degradation when places contain more than 1,000 tokens, rendering visual and animated simulations impractical. To address this limitation, a data partitioning strategy was devised and implemented for the input time-series datasets, namely *container_states* and *route_states*, which represent container fill level increase and dynamic route conditions, respectively. For the container states dataset, the input data is segmented into multiple files, each corresponding to a fixed time interval (e.g., two-hour intervals, with the first partition covering the period from 0 to 7200 seconds). Additionally, metadata describing the available partitions is maintained in a separate file. This metadata is represented as a multiset of timed integer tokens, where each token denotes the logical identifier of a partition and its timestamp corresponds to the earliest event time within that partition. The simulation loads data in two stages:

1. All the partition tokens are loaded at the start as described in chapter 4.3.
2. As the simulation progresses, the relevant partition data is dynamically loaded into the model once the current simulation time reaches the specified time threshold associated with each partition.

The sub-net illustrated in Figure 2 implements the described partitioning mechanism for the *container states* dataset, utilizing three places and two transitions.

```
colset PARTITION = time timed;
colset CLIENT_CAPACITY = intinf;
colset CLIENT_STATE = record id:STRING * cap:CLIENT_CAPACITY timed declare input_ms;
globref csp0 = empty: PARTITION timed ms;
globref c0 = empty: CLIENT_STATE timed ms;
var client_state_partition: PARTITION;
```

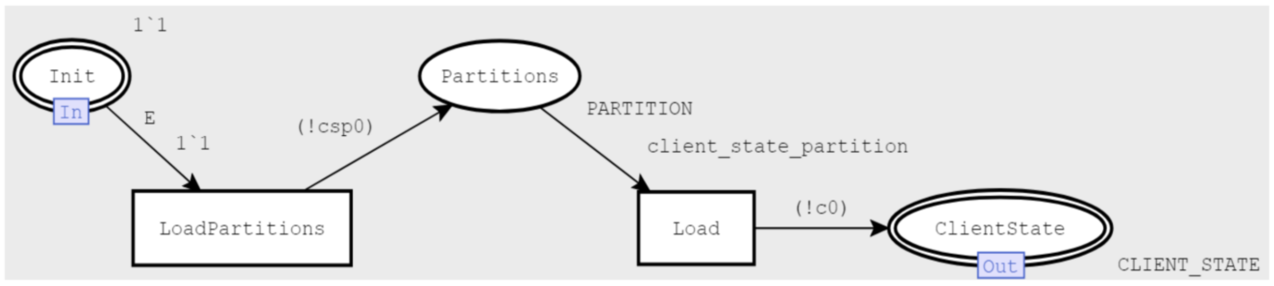


Fig 2. Container states partitioned loading

The *Init* place contains a single token that enables the *LoadPartitions* transition to fire once, loading `PARTITION` multiset to global variable `csp0` and placing that token multiset into the *Partitions* place. The formal definition of partition is

$$PARTITION = \{(p, t) | p \in \mathbb{N}, t \in T\}, \tag{1}$$

where, p is the partition number, t is the time of the first token in the partition. As the simulation progresses and the model time reaches the time associated with a given `PARTITION` token, the token becomes enabled, binds to the variable `client_state_partition`, and triggers the *Load* transition. This transition loads the corresponding partition data as multiset stored in global variable `c0` and outputs it to the *ClientState* place, thereby making the data available for subsequent simulation steps.

Similar approach was utilized for loading route states dataset.

5. Results of the study

5.1 WM model architecture and formal definition

The proposed model constitutes a formalized representation of the WM system and is capable of assessing the efficiency of the scheduling algorithm under various conditions, simulating edge cases, and accommodating overload scenarios. The real-world characterization of WM presented in literature review section is intricate and encompasses a multitude of variables; therefore, the problem is streamlined by positing the following assumptions:

1. All vehicles are homogeneous, each possessing a defined trunk volume capacity.
2. The quantification of waste is articulated in terms of its volume capacity.
3. All containers are homogeneous, with established thresholds for full capacity (exceeding 80% load) and overflow capacity (at or above 100% load). In instances of container overflow, waste is deposited externally, resulting in environmental pollution. The collection vehicle is capable of servicing both overflowed and non-overflowed containers, though the volume handled is constrained by trunk capacity; thus, in cases of significant overload, there may be circumstances wherein only a portion of the container is serviced.
4. New vehicles and containers may be integrated into the system at any point; however, such additions are only recognized from the subsequent model period.
5. There exists a service compatibility among all combinations of container and vehicle pairs.
6. The model operates on a periodic basis, with the designated period being 604800 seconds (equivalent to one week). The smallest discrete unit of time is defined as one second.
7. Route is the itinerary for an individual vehicle throughout a singular model period. A vehicle can execute at most one route within a period.
8. Trip is the journey from *depot* to *depot* for the same vehicle within a Route. A vehicle begins at the *depot*, visits containers, may *unload* several times if needed, and returns to the *depot* with an empty trunk to begin another trip if required. Route may contain multiple trips.
9. A vehicle may collect the same container multiple times within a single model period.
10. All vehicles commence operations from the same *depot* and are required to unload their trunks prior to returning to the *depot*, thereby ensuring readiness for subsequent routes.
11. Upon reaching full trunk capacity, a vehicle must proceed to unload before continuing to the next designated location.

The model's assumptions facilitate a realistic definition of the WM system, enabling effective simulation and evaluation of various operational scenarios.

Model entity types are formally defined as:

$V \equiv string$ is vehicle ID,

$C \equiv string$ is container ID,

$T \equiv \mathbb{N}$ is a model time domain,

$W \equiv \mathbb{N}$ is waste volume, measured in dm^3,

$L \equiv \{'depot','unload'\} \cup C$ is location ID,

$D \equiv \mathbb{N}$ is distance driven by a vehicle, measured in meters.

The model architecture components are depicted in Figure 3. A comprehensive description of each of the individual model components, presented in a logical sequence that mirrors the order of events that take place throughout the usage of this model.

The ***Model inputs*** consist of a series of files that contain timed multisets. In particular, time series container ID tokens are loaded from *clients.txt*, and specify containers as they appear in the system, formally defined as

$$C_{ID} = \{(c,t) | c \in C, t \in T\}, \tag{2}$$

Time series vehicle ID tokens are loaded from *vehicles.txt*, define vehicles as they appear in the system, formally defined as

$$V_{ID} = \{(v,t) | v \in V, t \in T\}\ . \tag{3}$$

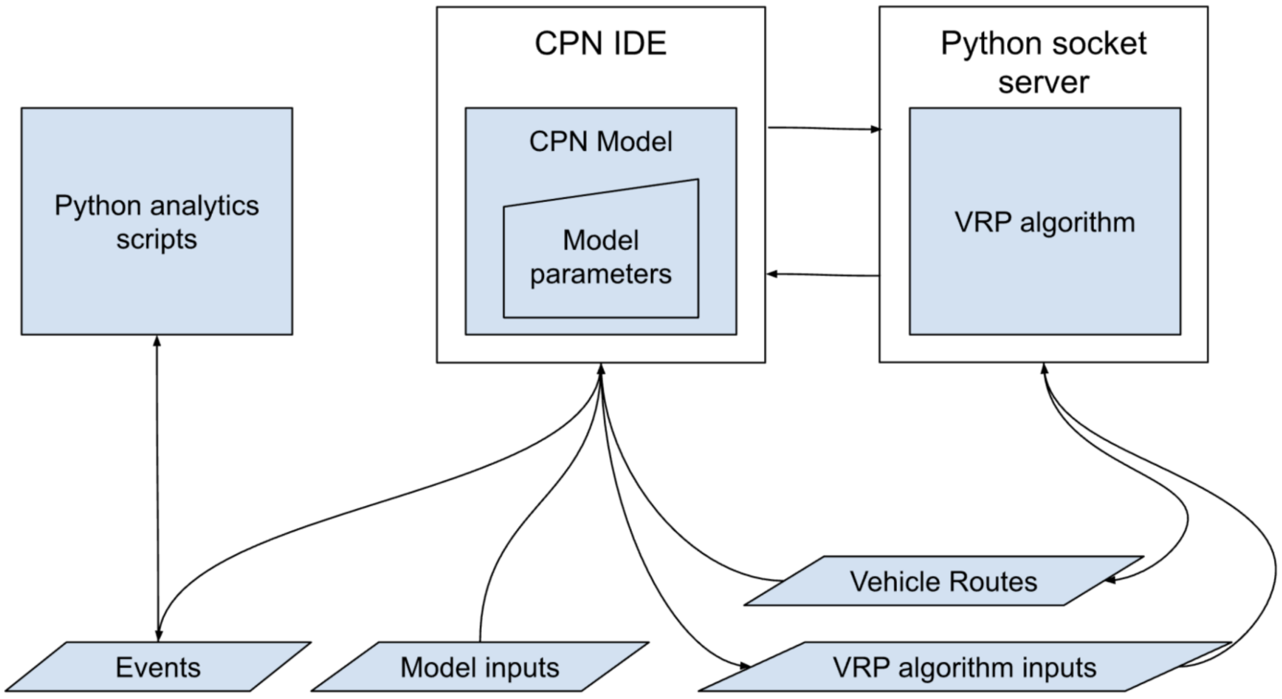


Fig 3. Model architecture components

Additionally, *client_states_partitions.txt* and *route_states_partitions.txt* declare partitions (1) for *container states* and *route states* datasets correspondingly. Each partition's *container states* dataset is loaded from *client_states_{i}.txt* , that contains container volume incremental updates from the current partition's start_time until the next partition's start_time. *Container states* dataset is defined as

$$C\_STATE = \{(c, w, t) | c \in C, w \in W, t \in T)\}, \tag{4}$$

where, w is the amount of waste added to container c at time t.

Each partition's route states dataset is loaded from *route_states_{i}.csv*, which contains route states replacements from the current partition's start_time until the next partition's start_time. Route states dataset is defined as

$$R\_STATE = \{(sloc, eloc, et, ed, t) | sloc \in L, eloc \in L, et \in T, ed \in D, t \in T\}, \tag{5}$$

where, $sloc, eloc$ are start and end locations, et, ed are estimated time and distance travelled, t is a time of the route record availability.

The ***CPN model*** is hierarchical and consists of the following sub-nets:

1. Sub-nets *init_clients*, *init_vehicles*, *init_client_states*, *init_route_states* load corresponding tokens from files which are formally defined by (2-5). Sub-net *init_clients* is presented in Figure 2 and *init_vehicles* is implemented in the same way, utilizing data partitioning. Sub-nets *init_client_states* and *init_route_states* are implemented similarly but without data partitioning.

2. Sub-net *scheduler* emits a token when period starts to open a latch to train vehicle routes, also it emits another token to a different place when period ends to store collected events as files.

3. Sub-net *vrp_algorithm* is depicted in and implements integration with Python socket server.

4. Sub-net *route_execution* is depicted in Figure 4 and implements vehicle route execution mechanics.

5. Sub-net *main* glues all above submodels into a single TCPN model.

The ***Model parameters*** consist of: *vehicle_cap_max*, which represents the maximum allowable value of the vehicle's capacity measured in dm³, and *client_cap_full*, which signifies the threshold value of the fill also measured in dm³; in real-world systems, this particular value typically hovers around 80% of the total volume of the container. Additionally, *client_cap_overflow* is defined as the maximum volume that the container can hold, and if this threshold is surpassed, individuals may place waste outside the container, thereby contributing to environmental pollution. It is also possible to change service and unload time delays within the model itself.

The ***Python socket server*** listens for incoming requests from *vrp_algorithm* sub-net to plan vehicle routes. The communication is done utilizing the library proposed in [13]. For each algorithm invocation new socket connection is established. The inputs are written to files and parameters are sent via the socket connection. The following transition blocks until the resulting vehicle routes file name is received via established connection. Finally, the connection is closed and VEHICLE_ROUTES token is obtainet.

The ***VRP algorithm inputs*** are recorded in files, which encompass a set of containers, a set of vehicles that are available for route scheduling and the map data which represents route time and distance information at different time periods. The rest of parameters that are captured include the current time, the start scheduling time, the end scheduling time, *vehicle_cap_max*, *client_cap_full*, *client_cap_overflow*, and the iteration number, all of which are transmitted as part of the socket command.

The ***VRP algorithm*** is declared as a Python function that has formal definition as

$$(p \in \mathbb{N}, w_v \in W, w_f \in W, w_o \in W, t_s \in T, t_e \in T, t \in T, C_t \subseteq C, V_t \subseteq V) \longrightarrow RS, \tag{6}$$

where, p is model iteration period index (e.g., number of the week for weekly scheduling), w_v is *vehicle_cap_max*, w_f is *client_cap_full*, w_o is *client_cap_overflow*, t_s is scheduling window start, t_e is the scheduling window end, t is routes creation invocation time, C_t is set of available container IDs at time t, V_t is set of available vehicle IDs at time t. The constraints $w_o \geq w_f, t_e \geq t_s$ hold.

The result RS is a set of route schedules for the available vehicles, defined as a set of tuples

$$RS = \{(v, ls, t) | v \in V, ls \in LS, t \in \mathbb{N}\}, \tag{7}$$

where, each route schedule contains vehicle ID v, list of scheduled locations ls and route start time ts. Location schedule LS is represented as an ordered list of tuples consisting of location ID lid and earliest time the vehicle can leave the location $leaveat$

$$LS = [(lid, leaveat) | lid \in L, leaveat \in \mathbb{N}], t_s < leaveat < t_e. \tag{8}$$

The ***Vehicle Routes*** created with Python MTCVRPTW algorithm are written to file as a single `VEHICLE_ROUTES` token and filename is send to CPN model via socket connection. The token is subsequently read by the CPN model for simulation. The `VEHICLE_ROUTES` is a list of `VEHICLE_ROUTE` tokens that are formally defined by (7).

The sub-net *route_execution* is presented in Figure 4.

Arc inscriptions are bind operational variables. Color `VEHICLE_ID` is formally defined by (3). Color `CLIENT_STATES` is a list of `CLIENT_STATE` which is defined by (4). Color `ROUTES` is defined by (5). Color `MODEL_META` represents list of vehicles involved in current route schedule and period number.

Vehicle routes execution is implemented as FSM, that begins when first token in *PreparedRoute* place becomes available and finishes with token availability in *FinishedRoutes* place. Places *Driving*, *Unloading*, *Serving*, *Depot*, *Waiting* represent vehicle states. Transitions *Drive*, *Serve*, *Park*, *Unload*, *WaitAtDepot*, *WaitAtUnload*, *WaitAtDepot* represent vehicle state changes. Places *ClientStates* and *ClientStateUpdates* represent list of current container fill levels and requests to reduce container fill level after serve operation correspondingly. Purple places and transitions capture execution events into global variables.

Vehicle route execution starts with a transition *StartRoute* which consumes corresponding token from *FreeVehicle*. The token is moved to *Driving* place to represent the vehicle which is driving to depot to begin its route. After that, transition *Park* is activated and token is moved to *Depot* place. *WaitAtDepot* will not delay the vehicle at the first time and the token becomes immediately available in *Waiting* place. The vehicle is located at $sloc$ at this point and is ready to drive to the next location $eloc$. The *Drive* transition picks up route information state (5) for $sloc$, $eloc$ pair, such as drive time and distance, and sends updated token to *Driving* place. The new token has updated current location

$sloc$ to $eloc$ value, increased vehicle drive distance and incremented time to a point when vehicle is available at $sloc$. The token can be moved to one of *Unloading*, *Depot*, *Serving* places from the *Drive* place depending on the $sloc$ type. The transitions *WaitAtDepot*, *WaitAtUnload*, *WaitAtDepot* may delay tokens if the corresponding vehicle arrives earlier than the expected $leaveat$ timestamp, and in such cases, wait time is measured as the difference between expected and actual arrival. The *Serve* transition reduces container load and increases vehicle trunk load, it can also insert *unload* as next location before the actual scheduled location in case the vehicle trunk becomes full. When vehicle has visited all its assigned locations its state is moved to *FinishedRoute* place which contains tokens representing cumulative vehicle state computed during route execution. The *FinishRoute* transition releases vehicle to *FreeVehicle* place and removes vehicle from the list in *ModelMeta* place. After all the vehicles finish their routes the transition *RoutesFinished* is triggered and *FinishedRoutes* contains a token. The expected vehicle route schedule serves as a benchmark for comparing with actual behavior, allowing for detailed performance analysis of the routing algorithm.

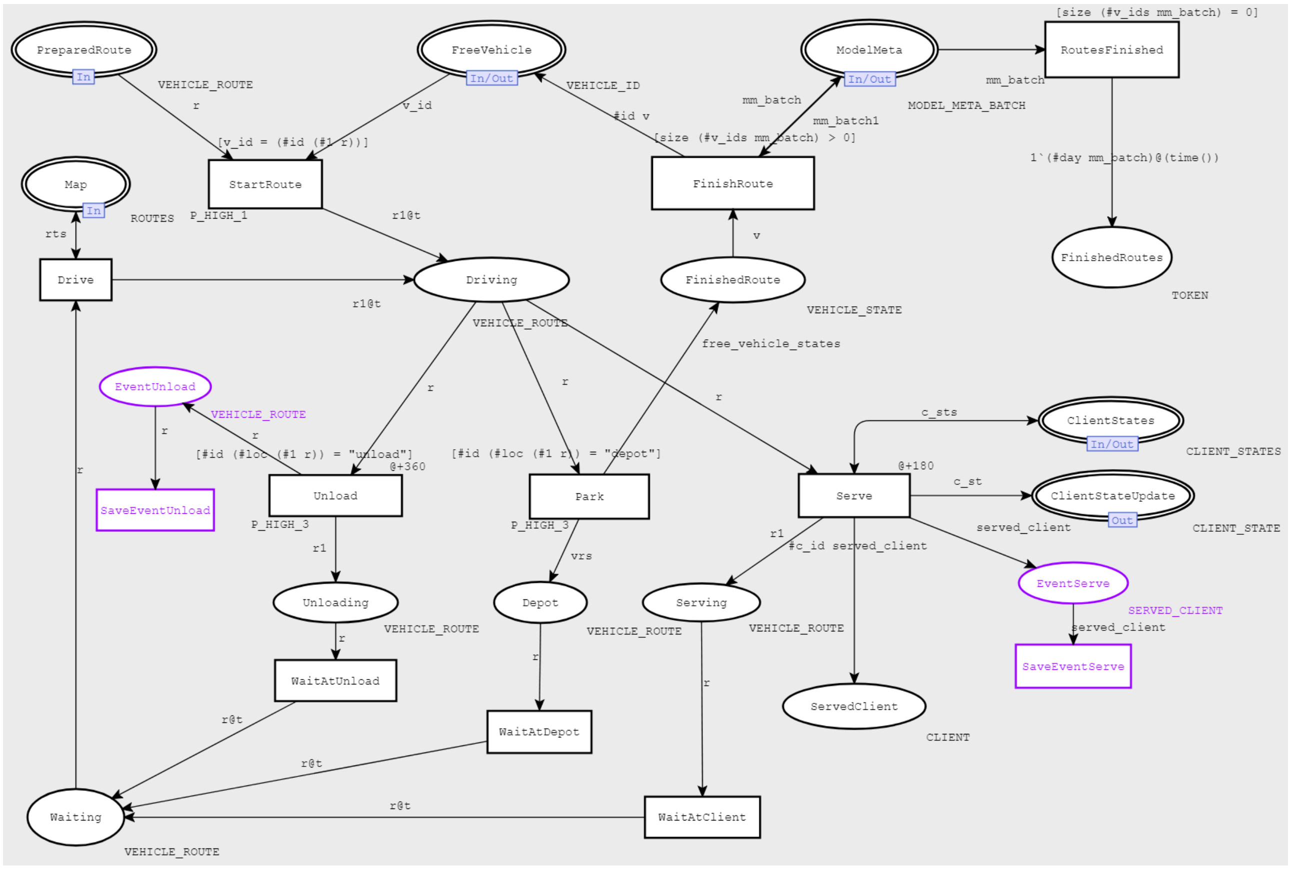


Fig 4. Sub-net that executes vehicle routes

The model captures ***Events*** during state transitions, which are written to the file system when the 604800-second period is finished.

Drive segment events:

$$D = [(v, l_s, l_e, t_s, t_e, d) | v \in V, l_s \in L, l_e \in L, t_s \in T, t_e \in T, d \in D], \tag{9}$$

where, v is vehicle ID, l_s, l_e are start and end locations, t_s, t_e are timestamps of departure and arrival, d is the drive distance.

Unload events:

$$U = [(v, d_u, t_u, w_u) | v \in V, d_u \in D, t_u \in T,\ w_u \in W], \tag{10}$$

where, v is vehicle ID, d_u is total distance travelled before the unload, t_u is finish time of the unload, w_u is unload volume.

Container fill level events:

$$S = \left[\left(c, t_c, w_f, b_f, b_o\right) \middle| c \in C, w_c \in W, b_f \equiv boolean, b_o \equiv boolean\right], \tag{11}$$

where, c is container ID, t_c is a timestamp of change, w_f is updated container volume at t_c, b_f is container fullness flag, b_o is a container overflow flag.

Container service events:

$$R = [(v, c, w_c, o, r, t_s) | v \in V, c \in C, w_c \in W, o \equiv boolean, r \equiv boolean, t_s \in T], \tag{12}$$

where, v is vehicle ID, c is container ID, w_c is waste collected volume, o is overflow flag at the time of service, r is service completion flag, t_s is a timestamp of service.

Vehicle drive expected E_e and actual E_a events:

$$E_e = [(v, l_e, e_e, t) | v \in V, l_e \in L, e_e \equiv string, t \in T], \tag{13}$$

where, e_e is *started_drive* event type, t is time when vehicle v must leave location l_e. The actual dataset has same structure

$$E_a = [(v, l_a, e_a, t) | v \in V, l_a \in L, e_a \equiv string, t \in T], \tag{14}$$

where, e_c is one of the *finished_wait*, *finished_drive*, *serve*, *unload*, *depot* event types, l_a is location of vehicle v at time t.

5.2 WM effectiveness metrics

The model effectiveness metrics are calculated by the *Analytics scripts* from the event datasets described by expressions (9 – 14). Direct calculation in TPN requires many aggregation places with complex logic, which overloads the model, distracting the user from the core logic. This separation of simulation and analysis enhances the modularity of the system, simplifies the model structure, and enables greater flexibility in defining and refining metric computations without modifying the core model. The following metrics are calculated.

Drive metrics are utilized to evaluate vehicle efficiency, the distribution of travel workload, and overall operational performance, thereby supporting the assessment of routing strategies in WM simulations. They are calculated based on dataset (9). The recorded route segments are grouped by vehicle identifier, and detailed metrics are computed on both per-vehicle and global levels. For each vehicle, the total, mean, median, minimum, maximum, and standard deviation of drive distances and drive times are calculated. Subsequently, these metrics are aggregated across all vehicles to generate a comprehensive system-wide summary.

Unload metrics are calculated based on dataset (10). These values are aggregated to compute the minimum, maximum, average, median, and standard deviation across all events. The total waste removed from the system is obtained by summing all discharge volumes. Additionally, the number of unload operations is counted for each vehicle to assess their relative utilization. These metrics provide a quantitative basis for analyzing routing efficiency, temporal distribution of unloading actions, and load management across the vehicle fleet.

Overflow metrics are key waste collection performance metrics based on dataset (11). For each container, it detects periods of overflow, defined as consecutive timestamps where the container remains overfilled, and computes both the duration and volume of overflow for each such episode. The volume is estimated as the difference between the last recorded capacity during the overflow and the capacity at the overflow start. Additionally, total waste generated per container is calculated by summing capacity values whenever a drop in fill level indicates a collection event. Then overflow durations and volumes are aggregated to compute descriptive statistics (total, average, median, min, max, and standard deviation), and the containers with the most and least severe overflow durations and volumes are identified.

Container service metrics are quantitative indicators related to service duration, service volume, and visit frequency. They are calculated based on dataset (12). The number of times each container was serviced during the observation period is first aggregated, yielding a per-container service frequency. Descriptive statistics are then computed for both service time and collected volume, including total sum, minimum and maximum values, arithmetic mean, median, and standard deviation. These metrics are used to characterize the temporal distribution and operational workload of waste collection, enabling the evaluation of consistency, service regularity, and overall system balance across containers.

Route metrics evaluate the temporal accuracy and efficiency of vehicle operations by comparing actual and expected schedules, calculated based on datasets (13, 14). Lateness is quantified by measuring how much later a vehicle completes service at a location compared to its expected departure time, and only positive delays are aggregated to assess deviations from the planned route. Earliness is measured as the time spent waiting at a location after service has been completed, reflecting how much earlier the vehicle arrived relative to its scheduled departure; however, waits at the depot are excluded since early returns there are permissible. Both lateness and earliness metrics are expressed through descriptive statistics, including total delay or wait time, average, median, minimum, maximum, and standard deviation. Additionally, wait times are grouped per service location to identify spatial patterns in early arrival behavior, providing insight into scheduling alignment and temporal slack across the service network.

5.3 Model simulation experiment

Model parameters were configured with *vehicle_cap_max* set to 20000 dm³, *client_cap_full* set to 4000 dm³ and *client_cap_overflow* set to 5000 dm³. There are 2 vehicles in the system $\{v_1, v_2\}$. and 10 containers $\{c_1, \dots, c_{10}\}$.

Google My Maps service was used to create 10 containers, a *depot*, and an *unload* location as presented in Figure 5.

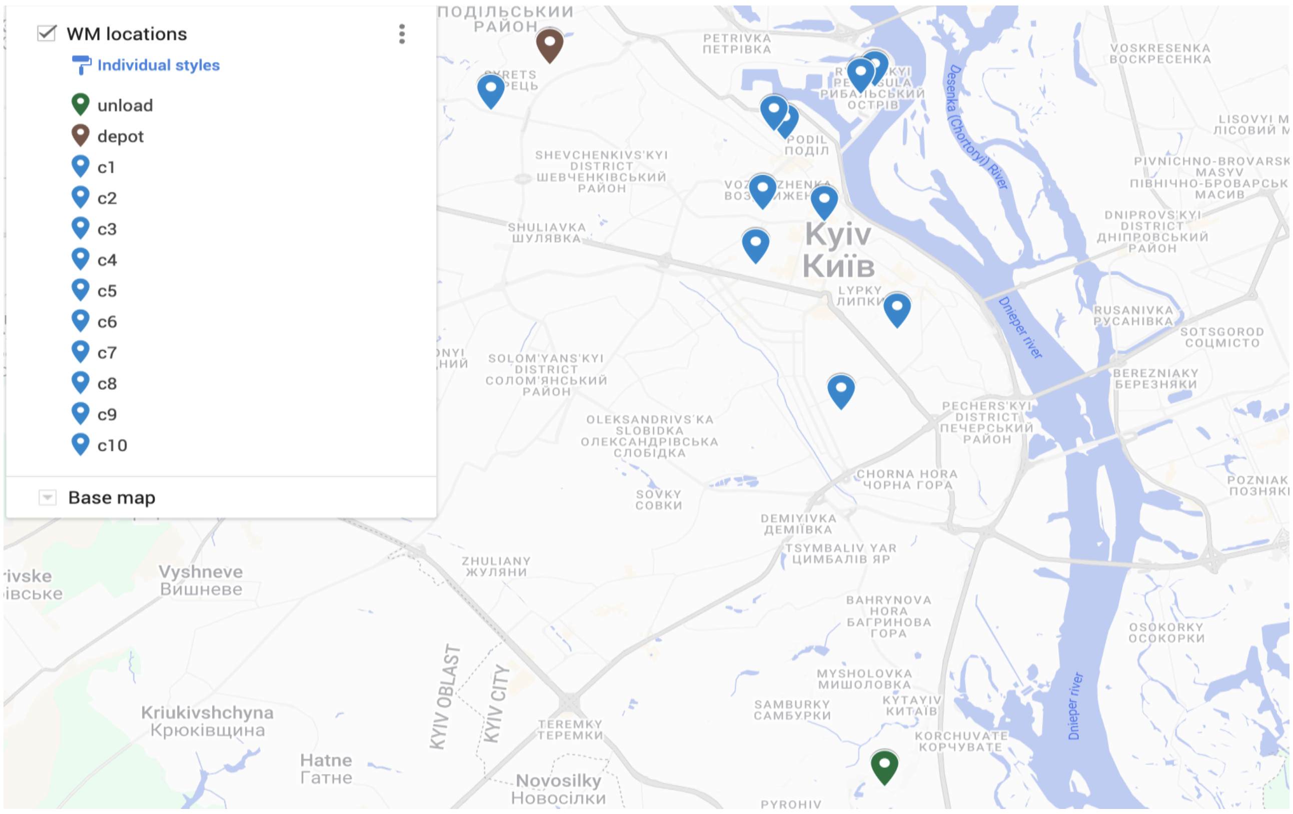


Fig 5. WM system locations

The KML file was exported, and locations were extracted to a separate file *locations.csv*, containing:

```
name,lat,lon
unload,50.3553941,30.5355109
depot,50.4830364,30.4533355
c1,50.4792403,30.5331494
c2,50.4778194,30.5294658
c3,50.4696738,30.5112153
c4,50.4713126,30.5084258
c5,50.4572033,30.5055653
c6,50.4749274,30.4390745
c7,50.4552136,30.5208752
c8,50.4476171,30.5037949
c9,50.436302,30.5384705
c10,50.4219645,30.5248047.
```

A tool was developed to compute time-dependent travel distances and durations between predefined locations using the Google Distance Matrix API. Based on a locations.csv file containing coordinates for the depot, unload point, and ten containers, batched API queries were issued for all origin–destination pairs at 3-hour intervals over a seven-day horizon, beginning from the upcoming Monday. To comply with API constraints, the origin and destination sets were dynamically partitioned. The resulting data, comprising distance, typical duration, and duration under traffic, were stored in *route_states.csv*, with each entry timestamped relative to the scenario start. A total of 12 distinct locations (the depot, unload point, and container locations) were processed, creating 12 × 11 = 132 unique origin–destination pairs per query. By sampling travel times at three-hour intervals over a seven-day planning horizon (24h / 3h = 8 intervals per day × 7 days = 56 time steps), a total of 132 × 56 = 7,392 route records were produced in *route_states.csv*. Each record corresponds to one origin-destination pair at a specific timestamp, thereby enabling fine-grained, time-aware routing analyses. The output was partitioned into 56 files, each containing 132 records.

A script was developed to simulate the temporal evolution of container fill levels for a set of waste collection containers over a seven-day period, partitioned into 2-hour intervals. Each container was defined with parameters including an initial capacity and a fixed fill interval (i.e., time required to accumulate 200 dm^3 of waste). Containers were loaded from the locations.csv file, and periodic records were generated for each 2-hour interval during which a container filled incrementally. For each update, the container ID, timestamp, and current capacity were recorded. The final output, written to *client_states.csv*, consists of a chronologically sorted sequence of capacity updates. In total, 8,400 records were generated, covering 12 locations across 336 time steps (every 2 hours over 7 days). This data serves as a synthetic input stream for time-aware waste collection models. The script produced 84 time partitions, each representing a 2-hour interval within the simulation horizon.

The embedding of a fully featured VRP algorithm was considered beyond the scope of this article. However, to demonstrate the tool in operation, a simple multi-trip route planning algorithm was developed that deterministically generates weekly waste collection schedules for a fixed number of vehicles and containers over a seven-day planning horizon. In the evaluated configuration, the algorithm was executed for 10 containers and 2 vehicles. Containers were assigned to vehicles using a round-robin strategy, with each vehicle consistently responsible for servicing a distinct subset of 5 containers throughout the week. Each vehicle was scheduled to perform exactly two trips per week, beginning at 01:00 on predetermined days. Specifically, vehicle v_1 was scheduled to operate on days 0 and 3 (corresponding to Monday and Thursday), while vehicle v_2 was assigned to days 1 and 4 (Tuesday and Friday), ensuring an even distribution of workload and temporal coverage. Each trip was designed to begin at the depot, include sequential visits to all assigned containers with fixed service durations (1 hour per container) and inter-container travel times (30 minutes), and conclude with a 2-hour pre-unload window followed by arrival at the unload location. Each vehicle returned to the depot after completing each trip, aligning accurately with downstream simulations.

The unload operation delay was specified as 1800 seconds (30 minutes), and the serve operation was defined as 300 seconds (5 minutes). The model simulation for a one-week period produced a route diagram presented in Figure 5, in which expected route location rules were visualized using the earliest-leave-at principle. For each planned route segment (red bar), an accompanying bar was shown to represent the actual route execution.

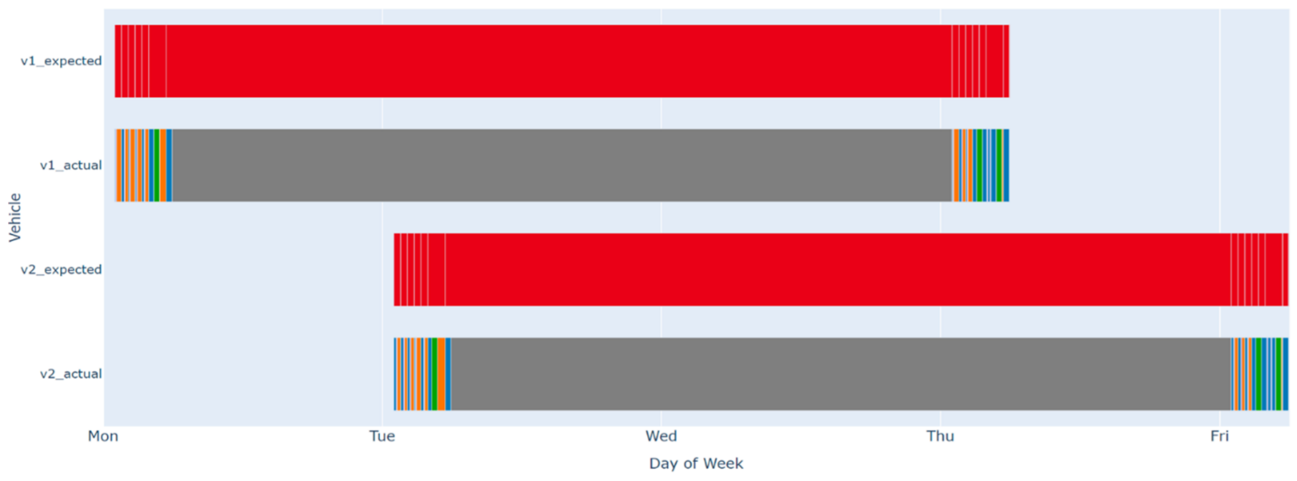


Fig 5. Route execution diagram, planned vs actual

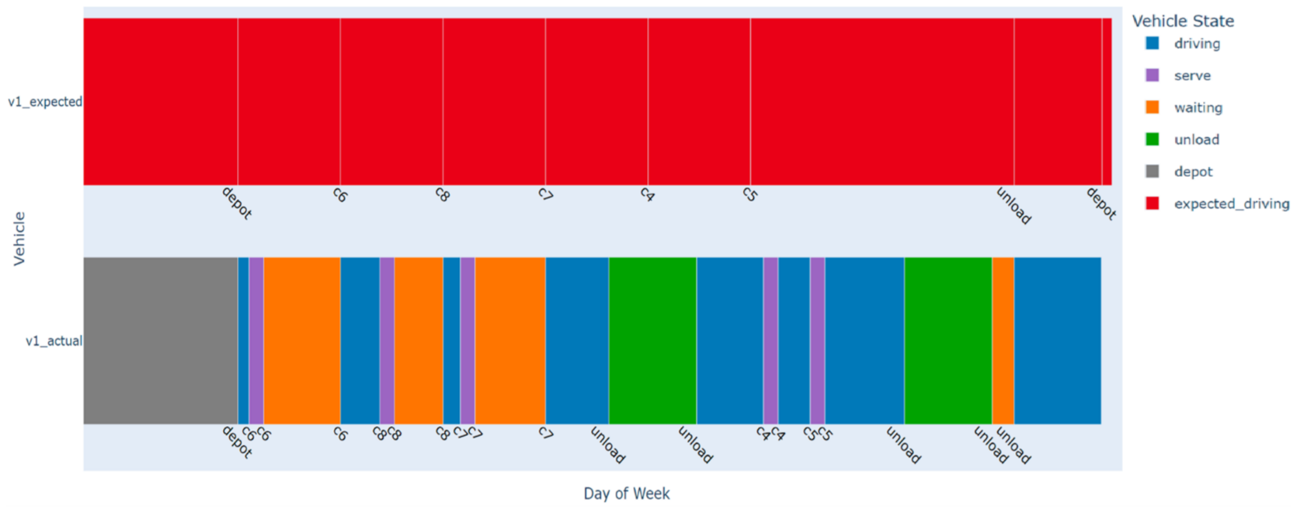


Fig 6. Route execution diagram section, planned vs actual

The *Drive metrics* are represented in Table 2–4.

Table 2. Global Drive Metrics Across All Vehicles

Metric	Total	Mean	Median	Min	Max	StdDev
Drive Distance (m)	341,959	11,398.63	8,214	1,657	24,029	7,802.89
Drive Time (s)	30,146	1,004.87	878.5	225	1,851	503.20

The vehicle v_2 traveled farther (183,311 m) and longer (15,946 s) than v_1 (158,648 m and 14,200 s), indicating a higher workload or longer routes. Vehicle v_1 had greater variability in segment lengths and times. Overall, the system logged 341,959 m and 30,146 s of driving, with average segments of 11,399 m and 1,005 s. High standard deviations and extreme values (up to 24,029 m and 1,851 s) highlight uneven routing and dispersed service points, emphasizing the importance of route optimization to balance workloads and reduce inefficiencies.

Table 3. Per-Vehicle Drive Distance Metrics

Vehicle	**Total (m)**	**Mean (m)**	**Median (m)**	**Min (m)**	**Max (m)**	**StdDev (m)**
v_1	158,648	10,576.53	6,796	1,657	24,029	8,959.58
v_2	183,311	12,220.73	10,167	4,649	24,029	6,663.09

Table 4. Per-Vehicle Drive Time Metrics

Vehicle	**Total (s)**	**Mean (s)**	**Median (s)**	**Min (s)**	**Max (s)**	**StdDev (s)**
v_1	14,200	946.67	806	225	1,851	591.88
v_2	15,946	1,063.07	885	402	1,816	408.58

The *unload metrics* are provided in Table 5. The vehicles v_1 and v_2 unloaded 3 times each. The total recorded unload capacity was 82,600 dm³.

There is variation in distances (39,058 to 159,282 m) and durations (17,534 to 364,711 s). The average capacity discharged per unload was 13,767 dm³, indicating efficient consolidation during trips. However, the large standard deviations across all unload parameters reveal inconsistency in vehicle routing and container targeting strategies.

The *overflow metrics* are provided in Table 6.

Table 5. Unload Metrics

Metric	**Min**	**Max**	**Mean**	**Median**	**Std Dev**
Distance Traveled (m)	39,058	159,282	98,576.5	104,441	47,416.39
Unload Time (s)	17,534	364,711	232,457.33	275,224.5	141,356.43
Capacity (dm³)	600	20,000	13,766.67	14,700	7,130.12

Table 6. Overflow metrics

Metric	**Total**	**Max**	**Min**	**Median**	**Average**	**Std Dev**	**Max**	**Min**
Overflow Delay (s)	2,052,384	208,800	57,600	86,369.5	102,619.2	40,432.68	208,800 (c_7)	57,600 (c_1)
Overflow Volume (dm³)	56,200	5,800	1,600	2,300	2,810	1,143.36	5,800 (c_7)	1,600 (c_1)

A total overflow delay of 2,052,384 seconds was recorded, with an average delay of approximately 102,619 seconds per overflow incident. The most affected container, c_7, experienced the maximum overflow duration of 208,800 seconds, while c_1 had the lowest at 57,600 seconds. Similarly, the corresponding overflow volume ranged from 1,600 dm³ to 5,800 dm³, with an average of 2,810 dm³. These findings suggest that certain containers, such as c_7, are particularly vulnerable to delayed service, due to route scheduling inefficiencies.

The *container service metrics* are provided in Table 7. All the containers were serviced twice.

Table 7. Container Service Time & Volume metrics

Metric	Total	Min	Max	Mean	Median	Std Dev
Served Time (s)	3,638,476	3,825	361,459	181,923.8	181,154.5	140,937.7
Served Volume (dm³)	82,600	0	7,400	4,130	4,100	3,052.02

A total of 82,600 dm³ of waste was collected, with an average of 4,130 dm³ per container and a high standard deviation of 3,052 dm³, indicating uneven fill levels at the time of service. The mean service duration of 181,923 s highlights the need to optimize collection timing to reduce overflow and idle waiting.

The *route metrics* are presented in Table 8–9.

Table 8. Vehicle Lateness and Earliness

Metric	Total	Count	Avg	Median	Min	Max	Std Dev
Lateness	9,136	4	2,284.0	2,361.5	1,521	2,892	618.29
Earliness	23,064	18	1,281.33	1,078	915	2,438	415.80

For locations with a single recorded wait, all statistical measures coincide. The observed wait times were 1398 seconds at c_1, 1357 seconds at c_4, 1093 seconds at c_5, and 918 seconds at c_{10}. For other locations a vehicle has waited twice and the metrics are present in Table 9.

Table 9. Per-Location Wait Time Statistics (seconds)

Location	c_2	c_3	c_6	c_7	c_8	c_9	unload
Avg	1056	985	1574	1436	976	920	2202
Median	1056	985	1574	1436	976	920	2202
Min	1049	966	1573	1427	958	915	1966
Max	1063	1004	1575	1445	994	925	2438
Std Dev	9.90	26.87	1.41	12.73	25.46	7.07	333.75

A total of 9,136 s of service lateness and 23,064 s of earliness were recorded, with fewer late events (4) but longer average delays (2,284 s), compared to shorter, more frequent early waits (1,281 s across 18 cases). These timing mismatches highlight the need for better synchronization between demand and route execution, especially for high-frequency containers. Container wait times ranged from 918 s (c_{10}) to 1,574 s (c_6), with c_6, c_7, and c_4 experiencing higher delays, while c_9 and c_{10} were serviced more efficiently. The unload site averaged 2,202 s of waiting, indicating possible bottlenecks, and high standard deviations at locations like c_8 and c_3 point to inconsistent responsiveness due to route variability.

6. Discussion of the results

Figure 6 presents a segment of the weekly schedule for vehicle v_1, specifically illustrating the execution of its second trip. This visualization juxtaposes the actual state timeline (bottom row) and the expected drive segments (top row), demonstrating the CPN IDE model's temporal accuracy and responsiveness to dynamic conditions. Notably, an unplanned unload operation occurred after servicing container c_7, triggered by vehicle capacity exhaustion. Upon returning from this unload operation, the vehicle proceeded directly to containers c_4 and c_5 without intermediate wait times. This behavior correctly reflects a schedule violation in the form of lateness – the vehicle was behind schedule and thus bypassed any idle period. Following these service completions, the vehicle initiated another unload, after which it returned to the depot. Unlike earlier segments, a waiting period is observed at the depot. This indicates that the vehicle arrived earlier than its planned departure for the next trip segment, and as expected by the model's earliest-leave-at principle, the vehicle correctly waited until the designated time window. This scenario validates the underlying TPN model's correctness: the simulation adheres to a time-guarded execution policy where vehicles depart only when permitted and adapt to runtime events (e.g., capacity overflows). The absence of premature departures and adaptive recovery from overflows illustrates the model's ability to enforce both temporal constraints and route reactivity, critical for smart waste collection systems where timing precision and resource limitations are operationally interdependent.

The experimental results provide a comprehensive insight into the performance of the waste collection system via metrics. Collectively, the results demonstrate that although service counts were evenly distributed, variations in overflow frequency, waste generation, and service responsiveness indicate that static scheduling is suboptimal. Instead, adaptive routing strategies, incorporating container fill predictions and route-time balancing, are essential to enhance operational performance and reduce service delays and overflows.

Several areas for future model improvement have been identified. These include the incorporation of smart container sensors and real-time handling of dynamic requests. Additionally, the introduction of stochastic events, such as vehicle breakdowns, could further enhance the model's robustness and realism.

Conclusion

The CPN model for the WM system was developed in this study, utilizing the CPN IDE simulation environment. For the flexibility and convenience, the model has been integrated with the Python socket server. The interface for MTCVRPTW algorithm has been designed and incorporated into the Python socket server. The model produces simulated events of different types into files. A key advantage of the developed model is its flexibility to incorporate custom user algorithms and adapt to diverse environmental setups.

The WM effectiveness metrics for container overflow, container service, vehicle unload effectiveness, vehicle route optimality, vehicle route schedule discrepancy were formalized and their calculation implemented with set of Python analytics scripts.

To demonstrate the model in action the experiment was conducted. The WM system used for analysis consisted of 10 containers, depot and unload place locations in Kyiv city and 2 vehicles. A time-dependent routing dataset was generated using the Google Distance Matrix API. A total of 7,392 timestamped records were produced, capturing travel distances and durations between all location pairs across varying traffic conditions. A synthetic container fill-level dataset was created, consisting of 8,400 time-stamped records simulating the progressive accumulation of waste in containers over seven days. These inputs were used to drive the simulation under realistic conditions. A deterministic MTCVRPTW algorithm was employed, assigning containers to vehicles using a round-robin strategy. Each vehicle was scheduled for two fixed weekly trips, following predefined routes with constant service and travel times. This simplified approach enabled consistent and controlled evaluation within the simulation framework. The simulation was successfully executed, producing both planned and actual route trajectories and enabling analysis of vehicle behavior and system performance.

The results demonstrate the model's capability to simulate schedules that are both temporally coherent and correct, as evidenced by the metrics obtained during simulations. Although advanced routing logic was not embedded in this iteration, the model was structured to support future integration. Overall, a flexible, extensible simulation environment was established to evaluate and improve waste collection strategies under realistic operational constraints.

References

[1] The Kyiv municipal WM company "Kyivcomunservice," Accessed: Aug. 8, 2025. [Online]. Available: https://kks.kyiv.ua/en/.

[2] C. Simon and S. Haag, "Pairing state automata and Petri nets – Simulation of processes in logistics," *in Proc. 38th ECMS Int. Conf. on Modelling and Simulation*, Jun. 2024. https://doi.org/10.7148/2024-0474.

[3] J. I. Latorre-Biel, D. Ferone, Á. A. Juan, and J. Faulin, "Combining simheuristics with Petri nets for solving the stochastic vehicle routing problem with correlated demands," *Expert Systems with Applications*, vol. 168, p. 114240, Jan. 2021. https://doi.org/10.1016/j.eswa.2020.114240.

[4] L. Zhang and J. Li, "Logistics Distribution Process Design Based on Stochastic Petri Nets and Big Data Algorithms," in *Proc. 2022 IEEE 2nd International Conference on Computer Systems (ICCS)*, Qingdao, China, 2022, pp. 55–59. https://doi.org/10.1109/ICCS56273.2022.9987800.

[5] K. B. Priya and R. Paramasivam, "The Combination of Petri Nets and Queueing Theory," *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 9, no. 1S5, pp. 293–294, Dec. 2019. https://doi.org/10.35940/ijeat.A1065.1291S519.

[6] G. Cavone, M. Dotoli and C. Seatzu, "A Survey on Petri Net Models for Freight Logistics and Transportation Systems," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1795–1813, June 2018, https://doi.org/10.1109/TITS.2017.2737788.

[7] D. Du, T. Liu, and C. Guo, "Analysis of Container Terminal Handling System Based on Petri Net and ExtendSim," *PROMTT*, vol. 35, no. 1, pp. 87–105, Feb. 2023. https://doi.org/10.7307/ptt.v35i1.4196.

[8] T. Kossowski, S. Samolej, and R. Davidrajuh, "Simulation in the GPenSIM Environment of the Movement of Vehicles in the City Based on Their License Plate Numbers," *Electronics*, vol. 13, no. 4, p. 683, 2024. https://doi.org/10.3390/electronics13040683.

[9] J. Beliën, L. De Boeck, and J. Van Ackere, "Municipal solid waste collection and management problems: A literature review," *Transportation Science*, vol. 48, no. 1, pp. 78–102, 2012, https://doi.org/10.1287/trsc.1120.0448.

[10] T. Benarbia, A. M. Darcherif, and D. J. Sun, "Modelling and performance analysis of smart waste collection system: a Petri nets and discrete event simulation approach," *International Journal of Decision Support Systems*, 2019 Vol.4 No.1, pp. 18–40. https://doi.org/10.1504/IJDSS.2019.103668.

[11] A. Ouhbi, H. Berrada, H. Boukachour, A. Farchi and H. Hachimi, "Multi-Trip Vehicle Routing Problem with Time Windows and Resource Synchronization on Heterogeneous Facilities," *Systems*, vol. 11, no. 8, p. 412, 2023. [Online]. Available: https://www.mdpi.com/2079-8954/11/8/412.

[12] D. A. Zaitsev, T. R. Shmeleva, and D. E. Probert, "Applying Infinite Petri Nets to the Cybersecurity of Intelligent Networks, Grids and Clouds," *Applied Sciences*, vol. 11, no. 24, p. 11870, Dec. 2021, https://doi.org/10.3390/app112411870.

[13] V. Gehlot, P. Rokowski, E. B. Sloane and N. Wickramasinghe, "Taxonomy, Tools, And A Framework For Combining Simulation Models With AI/ML Models," *2022 Annual Modeling and Simulation Conference (ANNSIM)*, San Diego, CA, USA, 2022, pp. 18–29, https://doi.org/10.23919/ANNSIM55834.2022.9859494.

УДК 004.94, 628.477

МОДЕЛЬ СИСТЕМИ УПРАВЛІННЯ ВІДХОДАМИ З ВИКОРИСТАННЯМ ЧАСОВИХ КОЛЬОРОВИХ МЕРЕЖІ ПЕТРІ

Григорій Рожков
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
https://orcid.org/0009-0009-5343-8974

Ірина Клименко
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
http://orcid.org/0000-0001-5345-8806

Управління відходами є ключовим елементом функціонування сучасних міст. У статті представлено нову модель системи збору відходів, що описується як дискретно-подійна система (DES), реалізована за використанням часових кольорових мереж Петрі (TCPNs) у поєднанні з інтегрованим сервером на Python. Модель розроблена з урахуванням зміни динаміки заповнення контейнерів та змінних маршрутів, що дозволяє досягти відповідності реальним міським умовам.

Ключовим елементом розробленої моделі є інтерфейс алгоритму планування маршрутів транспортних засобів з обмеженнями на вантажомісткість, з кількома поїздками та часовими вікнами (MTCVRPTW), що дозволяє транспортним засобам багаторазово обслуговувати контейнери за плановий період, дотримуючись обмежень за обсягом і часом. Модель підтримує налаштування таких параметрів, як затримки в роботі, обсяг заповнення та переповнення контейнерів, а також вантажомісткість кузова автомобіля. Симуляцію реалізовано в CPN IDE з використанням часових рядів як вхідних даних, розділених на партиції для ефективної обробки. Інформація про наповнення контейнерів і дорожні умови періодично довантажується під час моделювання в реальному часі, що підвищує масштабованість і продуктивність. Модель генерує логи подій: рух, розвантаження, переповнення та обслуговування, які обробляються Python-скриптами для розрахунку метрик ефективності.

Було визначено основні метрики ефективності системи сміттєзбору, що включають відстань та час маршрутів, ефективність розвантаження, обсяг переповнення контейнерів, ефективність обслуговування і відхилення запланованих маршрутів від графіка.

Для демонстрації роботи моделі було проведено експеримент із використанням синтетичних даних, наближених до реальних умов. Розташування 10 контейнерів, точки розвантаження та депо було визначено за допомогою сервісу Google My Maps на основі координат реальних об’єктів у місті Київ. Генерація реалістичних маршрутних відстаней і часу виконувалася за допомогою Google Distance Matrix API. Алгоритм MTCVRPTW для 2-х автомобілів запланував по два виїзди на тиждень відповідно до статичних маршрутів. Симуляція моделі згенерувала логи подій, які були використані для підрахунку метрик ефективності. Аналіз метрик показав значні обмеження статичного планування маршрутів і підкреслив необхідність впровадження адаптивних стратегій, що враховують реальний стан контейнерів і трафік. Запропонована модель є гнучким інструментом для оцінки, аналізу і покращення стратегій збору відходів у містах.

Ключові слова: моделювання збору сміття, оптимізація збору сміття, дискретні системи, мережі Петрі, задача маршрутизації транспортних засобів.