Distributed Software Engineering

Service Design

Submitted by
Mridha Md. Nafis Fuad
MSSE0901

Project Scenario

Uthao is a local startup business that hired licensed drivers to participate in their ride-sharing business. They became popular due to their competitive pricing of rides but were also facing criticism due to a lack of IT infrastructure. This started causing a major customer mistrust towards the company in terms of reliability and security. They decided to build an online platform for managing their business. They immediately knew that they have to consider the existing pressure of ride requests in office times. On the other hand, according to their statistics, the net revenue was not matching with their predictions and thus suspected their drivers of not giving their proper share. The designed system should keep track of the billing of trips for accountability. They decided to automate the process and calculate the drivers' share. The performance of drivers based on their monthly income and customer feedback needs to be properly analyzed. In order to do so, a monthly report must be generated and checked manually.

The company visions to provide multiple services in the near future based on rides. This vision forces the design of the system to be reusable in nature. Service oriented architecture is a gold standard to support Uthao in their vision, support future business expansions, and respond to unpredictable business demands.

Business Process

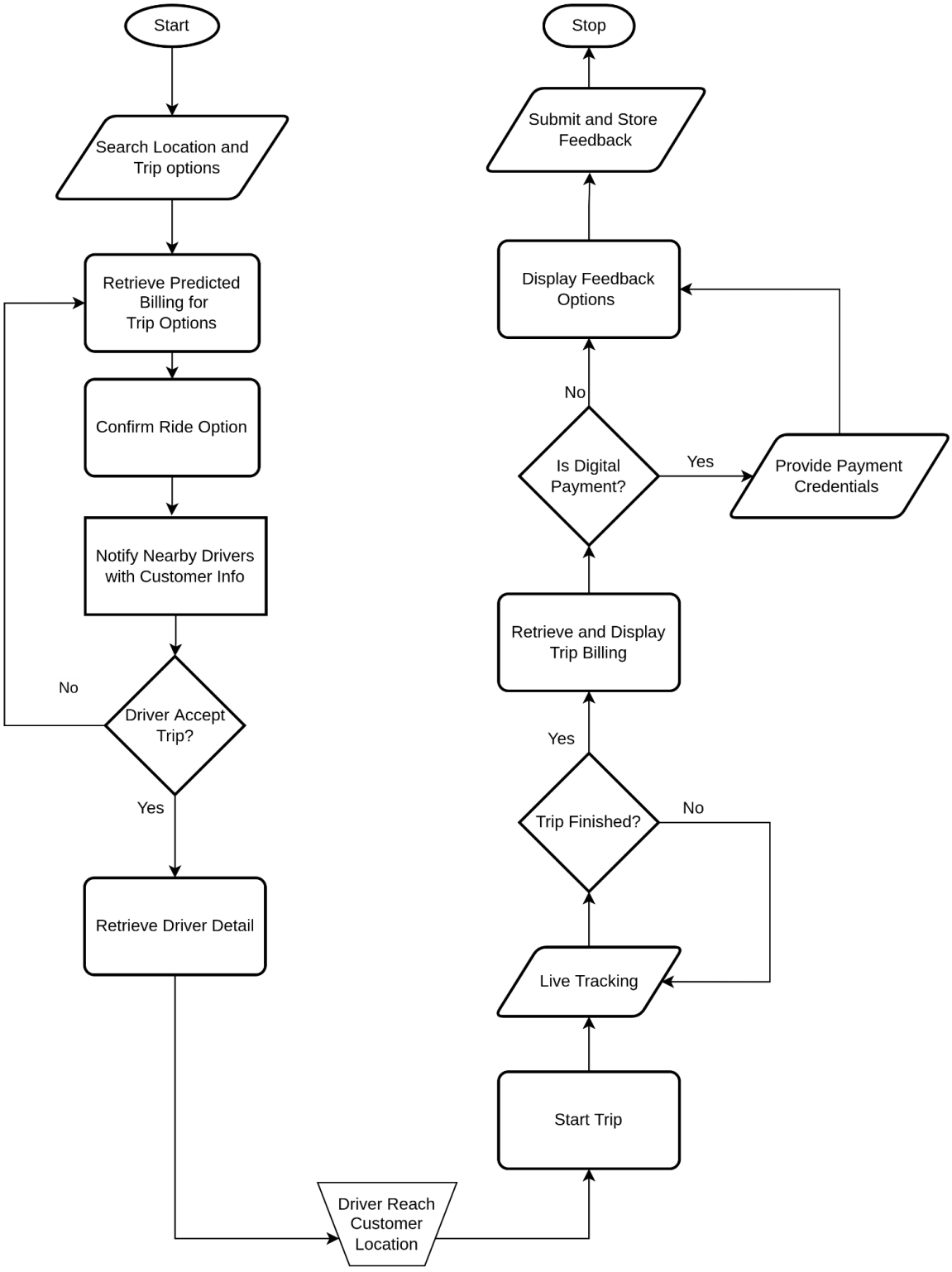


Fig: Ride Sharing Business Logic Flow

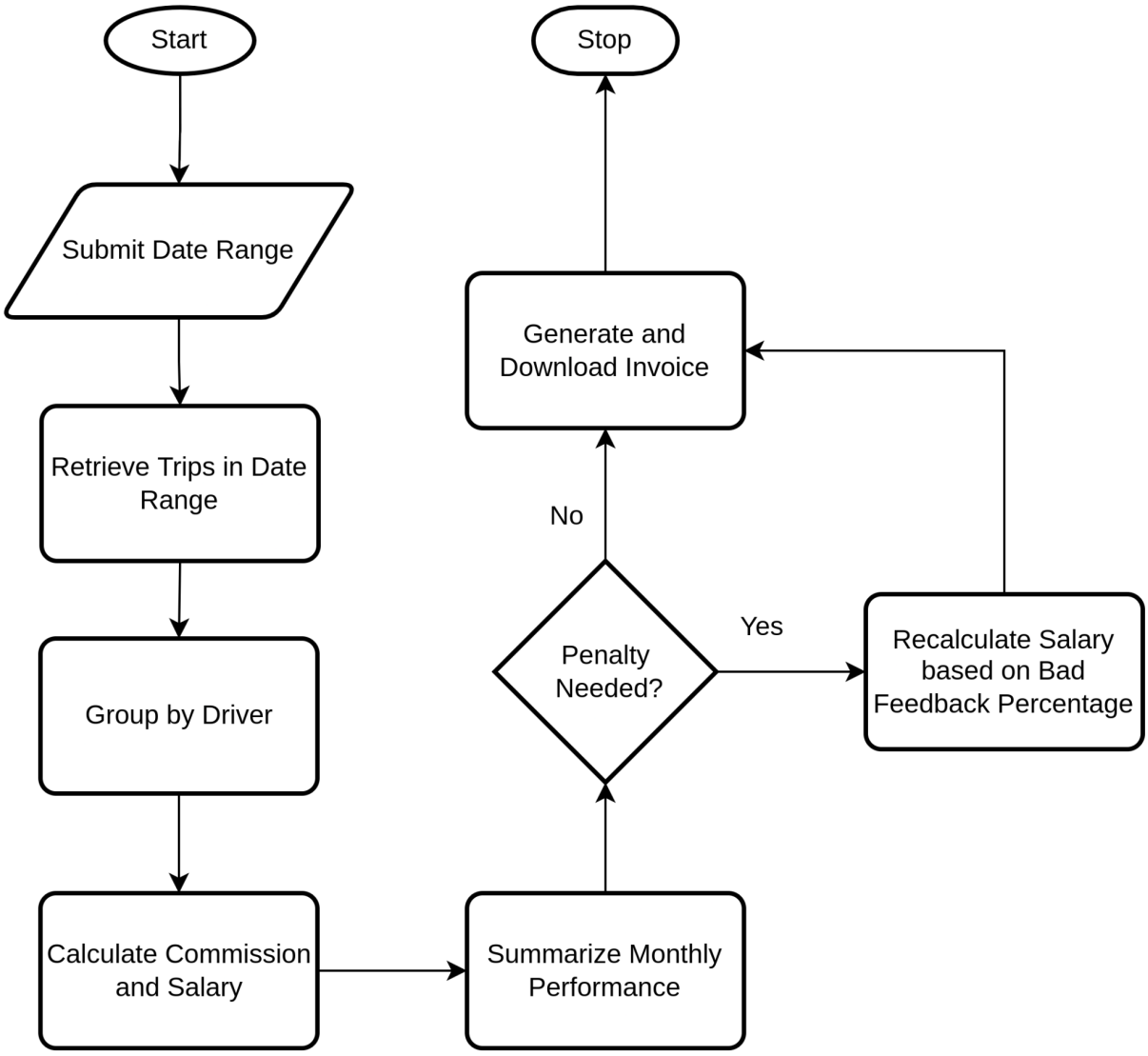


Fig: Salary Generation Logic Flow

Principle 1: Service Contracts and Standardization

This principle states that: *"Services share standardized contracts. Services within the same service inventory are in compliance with the same contract design standards."*

Planned Services

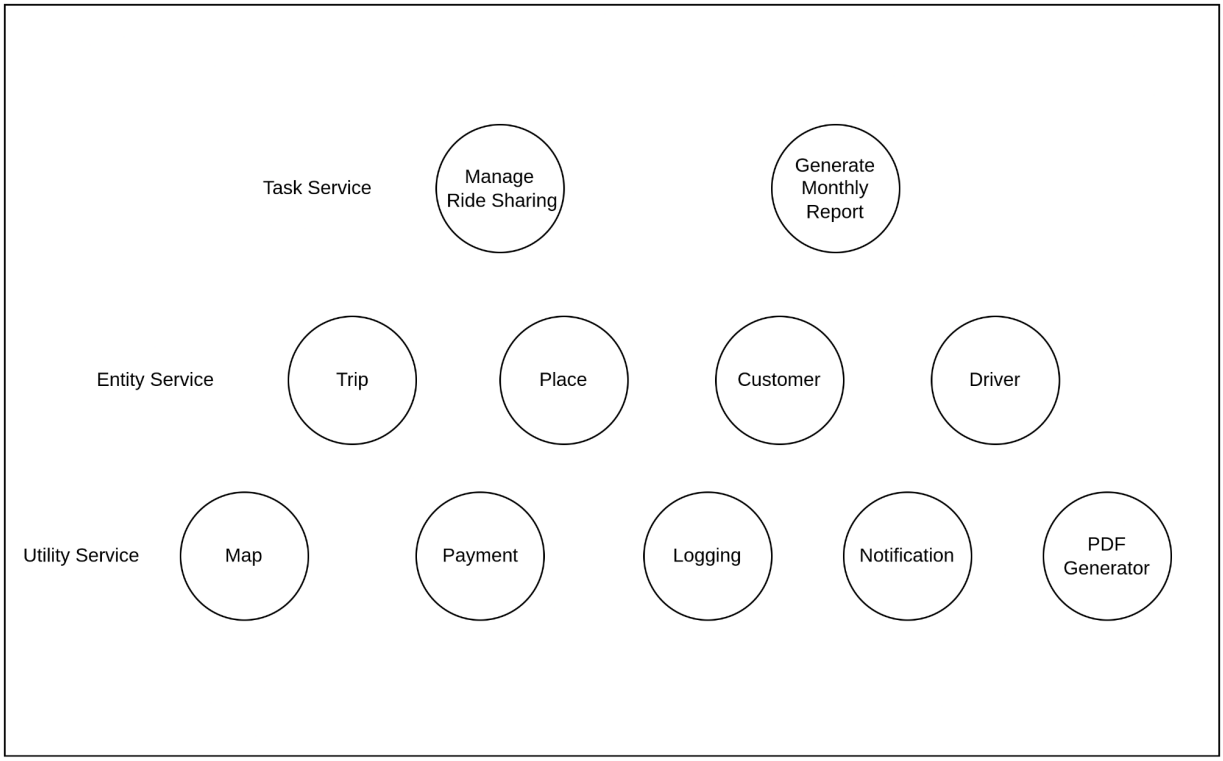


Fig: Service Model

Design Standards

Functional Expression Standards:

- Entity services will be named in accordance with the corresponding business entities from which they are derived, such as Driver/Customer Service
- The names of task services will be based on the process the service is responsible for automating, further prefixed with an appropriate verb, such as Manage Ride Sharing Service.
- Operations for all services will be based on the following naming format: verb + noun. For example UpdateTrip Operation
- The operation name cannot repeat the name of the service.

Data Representation Standards:

- Whenever complex types representing data constructs already established by entity schemas are required, the existing complex types must be used.
- Only when services need new complex types that fulfill processing requirements unique to the service are service-specific schema definitions allowed.

- All XML schema definitions must exist in separate files that are linked to the WSDL definitions.

Standardized WSDL Definition Profiles
The contract specification for the planned services is tabulated below.

Manage Ride Sharing Service	
GetRideOptions Operation	Input: source, destination address Output: ride options with predicted fares
CreateRideRequest Operation	Input: trip header with customer identifier Output: acknowledgement code
UpdateRideState Operation	Input: trip header and customer/driver identifier Output: acknowledgment code
StoreFeedback Operation	Input: trip identifier and feedback document Output: acknowledgement code
MakePayment Operation	Input: trip identifier and payment document Output: acknowledgement code

Table: The contract profile for the Manage Ride Sharing Service

Genarate Monthly Salary Service	
GenerateSalaryInvoice Operation	Input: driver identifier, date range document Output: report document
GetSummaryReport Operation	Input: date range document Ouput: report document

Table: The contract profile for the Genarate Monthly Salary Service

Driver Service	
RegisterDriver Operation	Input: driver document Output: acknowledgement code
UpdateProfile Operation	Input: driver document

	Output: acknowledgement code
BanDriver Operation	Input: driver identifier Output: acknowledgement code
GetDetails Operation	Input: driver identifier Output: driver document
GetVehicleDetails Operation	Input: driver identifier Output: vehicle document

Table: The contract profile for the Driver Service

Customer Service	
RegisterCustomer Operation	Input: customer document Output: acknowledgement code
UpdateProfile Operation	Input: customer document Output: acknowledgement code
BanCustomer Operation	Input: customer identifier Output: acknowledgement code
GetDetails Operation	Input: customer identifier Output: customer document
GetPaymentMedium Operation	Input: customer identifier Output: payment medium header document
StorePaymentInfo Operation	Input: payment medium header document Output: acknowledgement code

Table: The contract profile for the Customer Service

Trip Service	
GetTripDetails Operation	Input: trip identifier Outpu: trip document
CreateTrip Operation	Input: trip document Output: acknowledgement code
UpdateTrip Operation	Input: trip document Output: acknowledgement code
GetTripSummary Operation	Input: trip identifier

	Output: trip header document
Cancel Trip	Input: trip identifier Output: acknowledgement code

Table: The contract profile for the Trip Service

Place Service	
AddPlace Operation	Input: place document Output: acknowledgement code
GetPlaceDetail Operation	Input: place identifier Output: place document
GetCoordicates Operation	Input: place identifier Output: place coordinates
UpdatePlace Operation	Input: place document Output: acknowledgement code
DeletePlace Operation	Input: place identifier Output: acknowledgement code

Table: The contract profile for the Place Service

Map Service	
CalculateRoute Operation	Input: route header document Output: route document
CalculateCongession Operation	Input: Route header document Output: congestion amount
SearchPlace Operation	Input: place coordinates or place name Output: Place document

Table: The contract profile for the Map Service

Payment Service	
InitiatePayment Operation	Input: payment document Output: acknowledgement code
HandlePaymentSuccess Operation	Input: payment identifier Output: acknowledgement code

HandlePaymentFailure Operation	Input: payment identifier Output: acknowledgement code
GetTransactions Operation	Input: date range document Output: invoice document

Table: The contract profile for the Payment Service

Logging Service	
SaveLog Operation	Input: log document Output: acknowledgement code
GetLogs Operation	Input: date range document Output: log documents
SearchLog Operation	Input: log document Output: log documents
SetDefaultConfig Operation	Input: data mapping document Output: acknowledgement code
GetConfig Operation	Input: Output: data mapping document

Table: The contract profile for the Logging Service

Notification Service	
RegisterDevice Operation	Input: device identifier string Output: acknowledgement code
SendNotification Operation	Input: notification document Output: acknowledgement code
RefreshToken Operation	Input: device identifier string Output: acknowledgement code

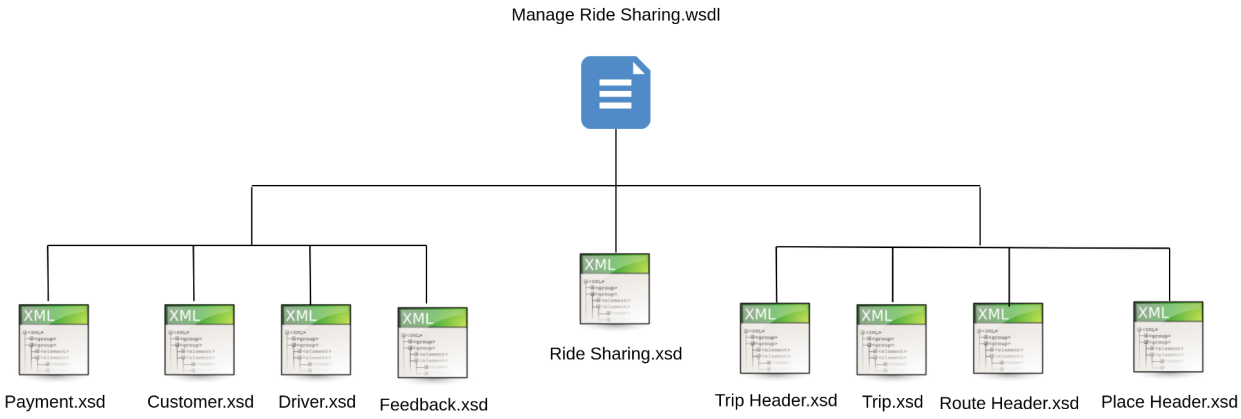
Table: The contract profile for the Notification Service

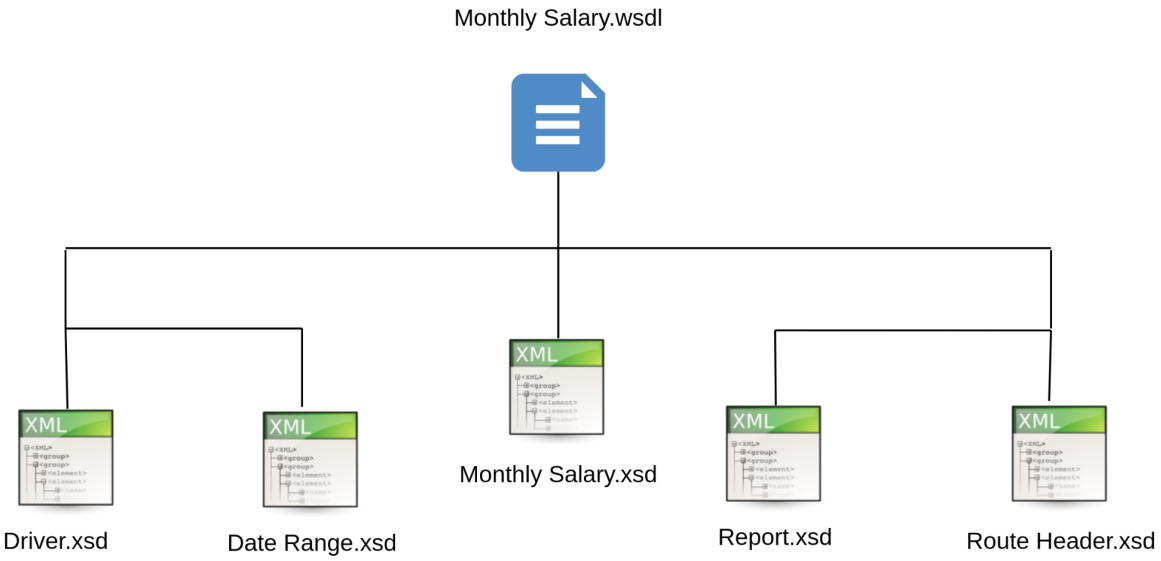
PDF Generator Service	
SetDefaultConfig Operation	Input: data mapping document Output: acknowledgement code
GetConfig Operation	Input:

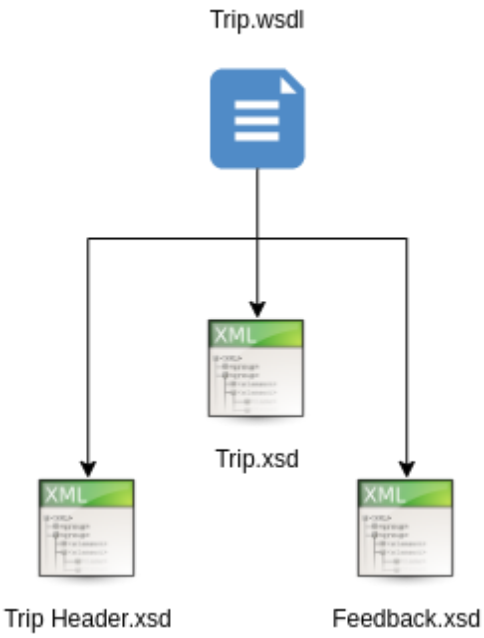
	Output: data mapping document
AddTemplate Operation	Input: template document Output: acknowledgement code
DeleteTemplate Operation	Input: template identifier Output: acknowledgement code
GeneratePdf Operation	Input: pdf template identifier, data mapping document Output: report document

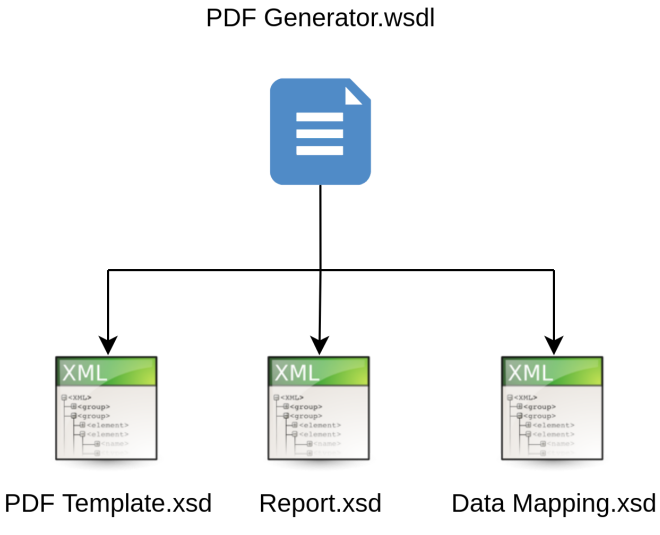
Table: The contract profile for the PDF Generator Service

Standardized Service and Data Representation Layers

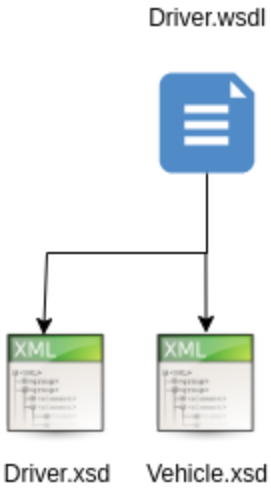


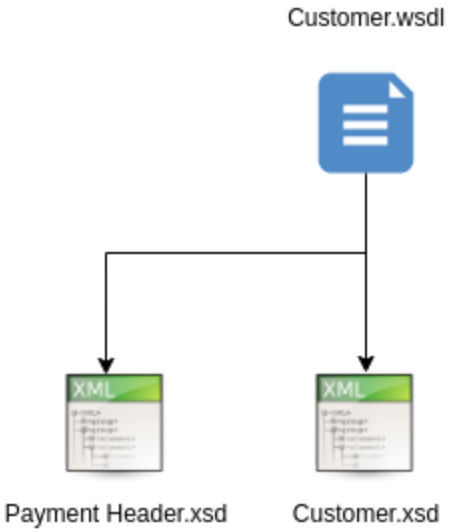

Monthly Salary.wsdl
Driver.xsd
Date Range.xsd
Monthly Salary.xsd
Report.xsd
Route Header.xsd

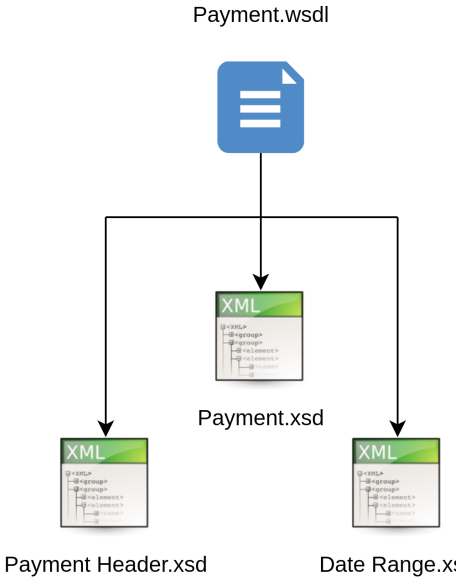

Trip.wsdl
Trip.xsd
Trip Header.xsd
Feedback.xsd

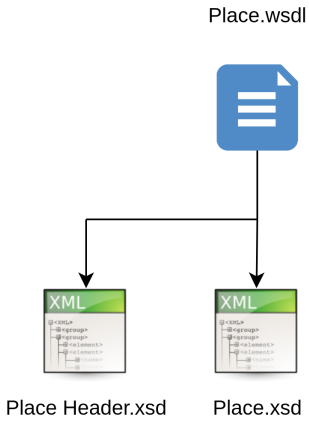

PDF Generator.wsdl
PDF Template.xsd
Report.xsd
Data Mapping.xsd


Map.wsdl
Logging.wsdl


Driver.wsdl
Driver.xsd
Vehicle.xsd


Customer.wsdl
Payment Header.xsd
Customer.xsd


Payment.wsdl
Payment.xsd
Payment Header.xsd
Date Range.xsd


Place.wsdl
Place Header.xsd
Place.xsd


Notification.wsdl
Message.xsd

Principle 2: Service Coupling

This principle states that: "*Services are loosely coupled. Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding Environment.*"

All the services of our design exhibits high level of logic-to-contract coupling, because they are custom services for which standardized service contracts are delivered. The Trip, Place, Customer and Driver services are based on the entity service model, which deliberately decreases potential functional coupling to external or parent business process logic. Run Design Studio, which is a task-centric service, is bound to Uthao's business process, which is a very specific procedure within the ride sharing application. As a result, this service shows targeted functional coupling which is intentionally done during its design.

Principle 3: Service Abstraction

This principle states that: *"Non-essential service information is abstracted. Service contracts only contain essential information and information about services is limited to what is published in service contracts."*
Following are a set of tables that summarize the technology, functional, programmatic, and quality of service abstraction for the designed services.

Customer Service	
Functional Abstraction (Content Abstraction)	Concise (the service contract provides targeted functionality with limited constraints)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Customer Service

Driver Service	
Functional Abstraction (Content Abstraction)	Concise (the service contract provides targeted functionality with limited constraints)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Driver Service

Trip Service	
Functional Abstraction (Content Abstraction)	Concise (the service contract provides targeted functionality with limited constraints)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Trip Service

Place Service	
Functional Abstraction (Content Abstraction)	Optimized (the sole operation provided by this Web service has few constraints and could likely not be more efficiently designed)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Place Service

Map Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)

Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Map Service

Payment Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open-to-Controlled Access (source code and design specifications for the Web service are openly available on the local LAN, but information about the Payment database is tightly guarded by a group of DBAs)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Payment Service

Logging Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open-to-Controlled Access (source code and design specifications for the Web service are openly available on the local LAN, but information about the Log database is tightly guarded by a group of DBAs)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Logging Service

Notification Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Notification Service

PDF Generator Service	
Functional Abstraction (Content Abstraction)	Concise (a limited amount of available functionality is exposed via the service contract)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the PDF Generator Service

Manage Ride Sharing Service	
Functional Abstraction (Content Abstraction)	Detailed (due to complex business flow associated with the exchange of trip data this service's contract has a low level of functional abstraction)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)

Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Manage Ride Sharing Service

Generate Monthly Report Service	
Functional Abstraction (Content Abstraction)	Optimized (the sole operation provided by this Web service has few constraints and could likely not be more efficiently designed)
Technology Information Abstraction (Access Control)	Open Access (the technologies used to build and implement this service are openly documented and published as part of architecture specifications)
Programmatic Abstraction (Access Control) Quality	Open Access (source code and design specifications are openly available on the local LAN)
Quality of Service (Access Control)	Open Access (SLA is published alongside service contract)

Table: The abstraction level for the Generate Monthly Report Service

Principle 4: Service Reusability

This principle states that: *“Services are reusable. Services contain and express agnostic logic and can be positioned as reusable enterprise resources.”*

The service contract for the Trip service is closely reviewed with an emphasis on facilitating service consumers beyond the Manage Ride Sharing service. To enhance this service and ensure reusability, the following changes are made.

The UpdateTrip operation will be used to change the data of existing trip record. This includes the details of the trip as well as the dynamic data, that is, the information that will change based on the state of the trip. Currently the Manage Ride Sharing service controls all the business process related to ride sharing. But this task is planned to be distributed in multiple task services and collaborating though orchestration. Throughout the ride sharing process multiple tasks need to update specific parts of the trip data. Thus denormalization is required keeping reusability of multiple services in mind.

The New UpdateTripState Operation
Currently CreateRideRequest, ConfirmRide and FinishRide Operations in the Manage Ride Sharing Service call the UpdateTrip Operation in which the entire Trip document needs to be passed as input. Denormalization is required to only update the state of the trip. This operation can serve the purpose of multiple task services requiring to change only the state of the ride sharing process. Since this operation also can handle the functionality provided by Cancel Trip Operation, we merge the responsibility to the newly created UpdateTripState Operation.

The New AdjustTripMetrics Operation
Currently the trip pricing is calculated in the task service (since billing algorithm is business centric) by fetching the trip information. But in near future, the billing might be dynamically adjusted based on rerouting of the trip, congestion, time spent on the total trip, etc. These metrics need to be stored in the trip database in order to dynamically calculate the billing in the task service.

Adding the aforementioned operations considering reusability, the Trip Service contract profile is adjusted.

Trip Service	
CreateTrip Operation	Input: trip document Output: acknowledgement code
UpdateTrip Operation	Input: trip document Output: acknowledgement code
UpdateTrpState Operation	Input: trip header document Output: acknowledgement code

AdjustTripMetrics Operation	Input: trip document Output: acknowledgement code
GetTripSummary Operation	Input: trip identifier Output: trip header document
Cancel Trip	Input: trip identifier Output: acknowledgement code

Table: The revised service contract for the Manage Ride Sharing Service