Department of Informatics, University of Zürich

BSc Thesis

Design and Implementation of a Workload Generator for the Oshiya Demo Application

Robert Dewor

Matrikelnummer: 08-715-922

January 23, 2012

supervised by Prof. Dr. M. Böhlen and C. Tilgner

University of Zurich^UZH

Department of Informatics

Abstract

The goal of this bachelor thesis was to design and implement a workload generator for the Oshiya demo application. The demo application displays the functionality of the Oshiya scheduling model and allows the user to create traditional and domain-specific scheduling protocols. In order to develop protocols, the user has to be able to analyse and test them for specific properties or behaviour. The workload generator enables the user to create customized transactions that allow the user to do so.
In this thesis, concept and design of the workload generator will be discussed and presented. The solution that has been implemented allows the user to create customized transactions in a flexible manner.

Zusammenfassung

Das Ziel dieser Bachelorarbeit war das Design und die Implementierung eines Workload Generators für die Oshiya Demo Applikation. Die Demo Applikation veranschaulicht die Funktionalität des Oshiya Scheduling Model und emöglicht dem Benutzer traditionelle und domain-spezifische scheduling Protokolle zu erstellen. Um Protokolle zu erstellen, muss der Benutzer in der Lage sein, diese auf spezifische Eigenschaften und spezifisches Verhalten zu analysieren und zu testen. Der Workload Generator erlaubt dem Benutzer individuell angepasste Transaktionen zu erstellen, die ihm dies ermöglichen.
In diesem Dokument wird das Konzept und Design des Workload Generator präsentiert und diskutiert. Die implementierte Lösung erlaubt dem Benutzer individuell angepasste Transaktionen auf flexible Art und Weise zu erstellen.

Contents

List of Figures

1 Introduction

Modern database systems have to schedule huge amounts of concurrent requests and have to ensure that the produced schedules fulfil certain correctness criteria (e.g., serializability). The state of art is to develop domain-specific schedulers imperatively for a given application. This leads to very complex scheduler implementations. The Oshiya demo application, is a tool for developing scheduling protocols declaratively [1].

The Oshiya demo application implements the Oshiya scheduling model. In the Oshiya scheduling model, scheduler states are stored in relations. The application displays the functionality of the generic Oshiya algorithm. In the Oshiya algorithm, protocols are implemented as scheduling queries. Requests are scheduled by iteratively executing these queries over relations. Every single step of the algorithm is displayed to the user graphically. The user also has the possibility to undo single steps the algorithm has performed. This helps the user to develop protocols as well as analyse and test them for specific properties or behaviour and display the results. Therefore the application provides an intuitive and easy-to-understand opportunity to develop protocols [1].

The aim of this bachelor thesis is to design and implement a workload generator for the demo application. The user will be enabled to create customized transactions. These transactions can be input manually or generated semi-automatically. This allows the user to simulate different clients executing transactions on the scheduler in order to analyse the behaviour of the chosen protocol. That means the user can define transactions that allow him to test protocols for specific behaviour or properties.

This paper is structured as follows: In Section 2, the background information about the Oshiya scheduling model and the Oshiya demo application will be given. In Section 3, the implementation task will be explained. In Section 4, the concept and design of the workload generator will be displayed. In Section 5, the chosen implementation of the workload generator will be shown. In Section 6 possible additional features for the Oshiya demo application are presented. In Section 7, a summary over the thesis is displayed. In Section 8, a scenario is shown, which displays the different GUI elements that have been created in order to allow the user to create an advanced workload.

2 Preliminaries

In this section, background information about the Oshiya scheduling model and the Oshiya demo application is given (Section 2.1). The generic Oshiya algorithm that is used in the Oshiya demo application is explained, as well as the functionality of the Oshiya demo application (Section 2.2). Afterwards necessary terminology is explained (Section 2.3).

2.1 Oshiya

Modern database systems have to schedule huge amounts of concurrent client requests. Standard database systems often do not satisfy domain-specific scheduling requirements, because they do not offer support for service-level agreements and only offer a limited set of fixed consistency levels. The state of the art is to develop schedulers imperatively for a given application [1].

Oshiya is a declarative scheduling model that is highly flexible. It allows the user to implement concise scheduling protocols. In Oshiya, the state of a scheduler is stored in three *scheduling relations*. These three relations are called *PendingRequests* ($\mathcal{R}$), *Executable* ($\mathcal{E}$) and *RelevantHistory* ($\mathcal{H}$). Relation $\mathcal{N}$ stores new requests, that will be scheduled by the Oshiya algorithm [1].

Pending requests are stored in relation $\mathcal{R}$. Requests that have been scheduled for execution are stored in relation $\mathcal{E}$. Relation $\mathcal{H}$ stores already executed requests in their execution order. In Oshiya, a *protocol* is formalized as a set of constraints called *protocol specification*. The constraints are implemented as declarative *scheduling queries*: $Q_{Scheduled}$, $Q_{Revoked}$, $Q_{Irrelevant}$. The scheduling of the requests is performed by executing the scheduling queries repeatedly over the scheduling relations. $Q_{Scheduled}$ identifies pending requests in relation $\mathcal{R}$ that can be selected for execution in this iteration, $Q_{Revoked}$ identifies non executable requests (e.q., deadlocked), $Q_{Irrelevant}$ returns requests that are irrelevant for future scheduling decisions. Irrelevant requests are removed from $\mathcal{H}$ [1].

The *Oshiya algorithm* is the same for every protocol (shown in Figure 2.1). Protocols can be created or modified by changing $Q_{Scheduled}$, $Q_{Revoked}$, $Q_{Irrelevant}$ and the schema of the scheduling relations [1]. The algorithm is executed in seven steps:

- In step 1, requests that were scheduled in the previous iteration are removed from $\mathcal{R}$.
- In step 2, new requests are added to $\mathcal{R}$ from $\mathcal{N}$.

```
  $\mathcal{H} = \mathcal{E} = \mathcal{R} = \emptyset$
  while true do begin
1   $\mathcal{R} = \mathcal{R} - \mathcal{E}$;
2   $\mathcal{R} = \mathcal{R} \cup \mathcal{N}$;
3   $\mathcal{R} = \mathcal{R} - Q_{Revoked}(\mathcal{H}, \mathcal{R})$;
4   $\mathcal{E} = Q_{Schedule}(\mathcal{H}, \mathcal{R})$;
5   $Execute(\mathcal{E})$;
6   $\mathcal{H} = \mathcal{H} \cup \mathcal{E}$;
7   $\mathcal{H} = \mathcal{H} - Q_{Irrelevant}(\mathcal{H})$;
  end
```

Figure 2.1: Oshiya Algorithm [1, Figure 2]

- In step 3, $Q_{Revoked}$ identifies the non executable requests in $\mathcal{R}$. Those requests are removed from $\mathcal{R}$.
- In step 4, $Q_{Scheduled}$ selects all requests from $\mathcal{R}$ that can be executed in this iteration without violating the protocol constraints.
- In step 5 and 6, the requests in $\mathcal{E}$ are executed and added to $\mathcal{H}$.
- In step 7, $Q_{Irrelevant}$ identifies the requests that are irrelevant for future decisions. Those requests are removed from $\mathcal{H}$.

2.2 Oshiya Demo Application

The existing demo application implements and illustrates the functionality of the Oshiya scheduling model and the Oshiya algorithm. It enables the user to develop new scheduling protocols by specifying the schema for the scheduling relations $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{H}$ as well as instances of the scheduling queries $Q_{Revoked}$, $Q_{Scheduled}$ and $Q_{Irrelevant}$. When request scheduling is done by iteratively executing scheduling queries over scheduling relations $\mathcal{R}$, $\mathcal{E}$, $\mathcal{H}$, the application displays the three scheduling relations graphically. This allows the user to follow every step of the Oshiya algorithm. The application allows the user to undo and redo single steps of the Oshiya algorithm. This enables the user to analyse the behaviour of protocols.

2.3 Terminology

A *user* is the person accessing the Oshiya demo application.

A *transaction* is a sequence of read or write operations followed by an abort or commit operation.

A *client* consists of exactly one transaction. Each client inserts one request into relation $\mathcal{N}$ at a time. Once the request has been scheduled for execution, the next request can be inserted into $\mathcal{N}$.

A set of requests of one or more clients is called *workload*.

The Oshiya algorithm consists of seven steps that can be executed in iterations. One step can be displayed as *n:s*. N stands for the current iteration, s for the current step of the iteration.

3 Problem Description

The current implementation of the Oshiya demo application creates transactions with random values that are used to display the functionality of the Oshiya algorithm. In order to develop protocols, the user has to be able to test those protocols for certain criteria or properties. This is not possible with the current implementation of the Oshiya demo application. In order to test protocols for certain criteria or properties, the user has to be able to create her own transactions. For example, if the user wants to test the worst case scenario for the two-phase locking protocol (2PL) she needs to be able to create transactions that create a deadlock.

The main task of this bachelor thesis is the design and implementation of a *workload generator*. The workload generator has to enable the user to configure the requests that have to be scheduled. It includes the ability to simulate clients that execute requests simultaneously. Those requests will be inserted into relation $\mathcal{N}$ in step 2 of the Oshiya algorithm. The main challenges of the development of the workload generator are:

- A client can only insert one request at a time. The next request can only be inserted, once the last request has been executed. That means the right requests have to be selected for insertion into $\mathcal{N}$.
- The user can undo steps from the Oshiya algorithm. The workload generator needs to offer the same functionality. It has to be able to restore the information of the previous steps and redo them.
- The user has to be able to design workloads, where the value column holds references to records in the database. That means, the user has to be able to create workloads that change the value of a data item based on its current value. For example, increasing the value of data item 1 by ten percent will be written as $1 * 1.1. $1 holds the reference to the value of data item 1.

4 Design of the Workload Generator

In the last chapter it was explained, why it is necessary to enable the user to create her own transactions with the Oshiya demo application. In this chapter, we analyse the requirements (section 4.2.1) and describe the design of the workload generator (section 4.1). Problems that arose during the development of the workload generator will be discussed and the chosen solutions explained in detail. In Section 4.2.2 the concept of the workload generator is described. In Section 4.3, workloads will be explained in detail.

In order to test the behaviour of protocols, the user has to be able to create real-world transactions with the Oshiya demo application. That means the demo application has to simulate the behaviour of one or more clients executing request transactions on the Oshiya scheduling model.

The workload generator, has to create a stream of requests. In step 2 of the Oshiya algorithm, requests are selected from the workload that has been specified. The selected requests are then processed by the Oshiya algorithm. Figure 4.1 displays the concept of the workload generator:

4.1 Model

After a workload has been created and finalized by the user, it has to be stored in an efficient manner. In order to simplify the process of selecting requests of the workload, it was chosen to store the workload in a database relation. This relation is called *relation Workload*. Requests are stored as tupel in the scheduling relations. Those requests have to consist of a set of columns that cannot be changed or removed. These columns are: *client id* ('cid'), *transaction id* ('ta'), *sequence number* ('seq'), *operation* ('op'), *object* ('ob') and *value* ('val'). This is because relation $\mathcal{N}$, $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{H}$ are based on this set of columns. Inserting request from relation Workload into relation $\mathcal{N}$ can only be done, when both relations are based on the same schema. That is why the schema of relation Workload and the schema of the scheduling relations will be based on the same set of columns. Additional columns can be added to the schema by the user. The user can also be interested in information generated by the application, for example the current step of the current iteration of the Oshiya algorithm. That is why additional columns can be of two different types.

Columns of type *UserInput* will be based on values the user specifies when she is creating a workload. An example is the column class of the class-based 2PL protocol. In this column, the user can specify a class for each transaction. Requests of transactions with a higher class

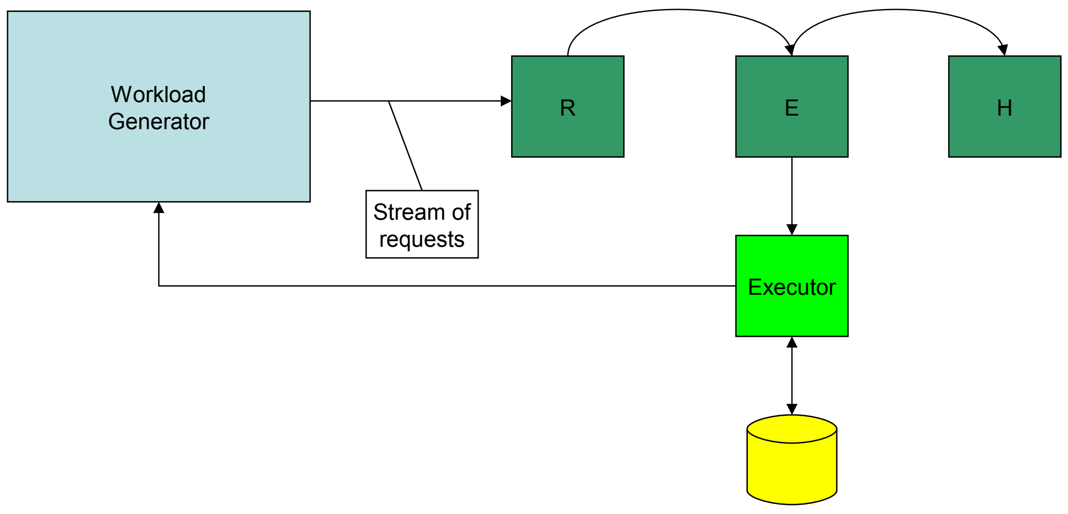


Figure 4.1: Concept of the Workload Generator

value get executed before requests of transactions with a lower class value.

Columns of type *SystemVariable* display values generated by the application and cannot be changed by the user. The column *SI* is a column of type SystemVariable. It stores the current SchedulingIteration step when the request has been inserted into relation $\mathcal{N}$.

It is important to note, that the schema of relation Workload and the schema of relation $\mathcal{N}$ are not equal. The schema of relation Workload only consists of columns of type UserInput. This is done because the user can undo and redo steps of the Oshiya algorithm and the values in columns of type SystemVariable are generated automatically. To avoid storing runtime values, all columns of type SystemVariable in relation $\mathcal{N}$ will be updated every time requests have been inserted into the relation.

Every transaction has a a unique *transaction id*. All transaction ids are stored and mapped to the client, the transaction belongs to. A mapping relation called *client_ta _mapping* will be used to store transaction ids with their corresponding clients. New transaction ids are generated based on the maximum value of all transaction ids in the mapping relation plus one. If a transaction has been committed or was aborted, the new transaction id will be added to the mapping relation. When the algorithm is stepping backwards and therefore an old transaction id has to be restored, the maximum value of all transaction ids of the corresponding client is deleted, and the new maximum value will be used to update the transaction id in relation Workload. The following table displays relation client_ta _mapping:

client_ta _mapping	
CID	TA

Relation $\mathcal{N}$ stores new requests, that will be scheduled by the Oshiya algorithm. In order to create customized workloads, the workload generator must determine the requests in relation $\mathcal{N}$ based on user input. In step 2 of every iteration of the Oshiya algorithm, requests are selected from relation Workload and inserted into relation $\mathcal{N}$. All requests from $\mathcal{N}$ are then inserted into relation $\mathcal{R}$ and all values in relation $\mathcal{N}$ are deleted. Requests in relation $\mathcal{R}$ that have been selected for execution will be inserted in relation $\mathcal{E}$. Requests in relation $\mathcal{E}$ will be executed by the *Executor* and inserted into relation $\mathcal{H}$. In order to automatically store the return values of the executer, an interface has to be created.

The following example illustrates two transactions t_1 and t_2 that can be used to test the two-phase locking protocol (2PL) for a worst-case scenario. Executed under the 2PL protocol, these transaction cause a deadlock.

Example 1 *Worst-case scenario for the 2PL protocol: A workload consisting of two clients, the first client executing t_1 and the second client executing t_2, can be used to display the worst-case scenario graphically with the help of the Oshiya demo application.*

t_1 reads item 1 and item 2, then t_1 changes the value of item 1 to +5.
t_2 reads item 1 and item 2, then t_2 changes the value of item 2 to +6.

Because t_1 is holding locks on data item 1 and 2, t_2 cannot write data item 2 and because t_2 is holding locks on data item and 2 as well, t_1 cannot write data item 1. t_1 is waiting for t_2 to release the locks and t_2 is waiting for t_1 to release the locks. Because both transactions are waiting on each other, a deadlock is created.

t_1 : r(1) r(2) w(1, +5) c
t_2 : r(1) r(2) w(2, +6) c

Those two transactions will result in the following history:

$H_1 : r_1(1)r_2(1)r_1(2)r_2(2)w_1(1,+5)w_2(2,+6)a_1c_2$

The following tables display relations $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{H}$ for Example 1, after the deadlock has occurred:

$\mathcal{R}$					
CID	TA	Seq	Op	Ob	Val
1	1	3	w	1	5
1	1	3	w	2	6

$\mathcal{E}$					
CID	TA	Seq	Op	Ob	Val
1	1	2	r	2	
2	2	2	r	2	

$\mathcal{H}$					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	r	2	
2	2	1	r	1	
2	2	2	r	2	

Under 2PL, one of the transactions has to be aborted, in order to commit the other. Relation $\mathcal{E}$ only displays the requests executed in the last iteration of the Oshiya algorithm, other requests are deleted from the relation.

4.2 Selecting the Right Requests

After the workload has been stored in relation Workload, requests have to be selected from the relation and inserted into relation $\mathcal{N}$. Every client in a workload can only insert one request at a time into relation $\mathcal{N}$. The next request can only be inserted, once the previous request has been executed in step 4 of the Oshiya algorithm. In order to select the right requests from relation Workload, an algorithm has to be created. This algorithm is called *request selection algorithm*. Requests will be inserted into relation $\mathcal{N}$ in step 2 of the Oshiya algorithm.

4.2.1 Requirements

In order to enable the user to test protocols or display their functionality it will be convenient to allow the user to execute transactions in an *infinite mode*, that means, transactions that have been committed or aborted will restart with a new transaction id. An algorithm has to be developed that can create new transaction ids as well as store old transaction ids.

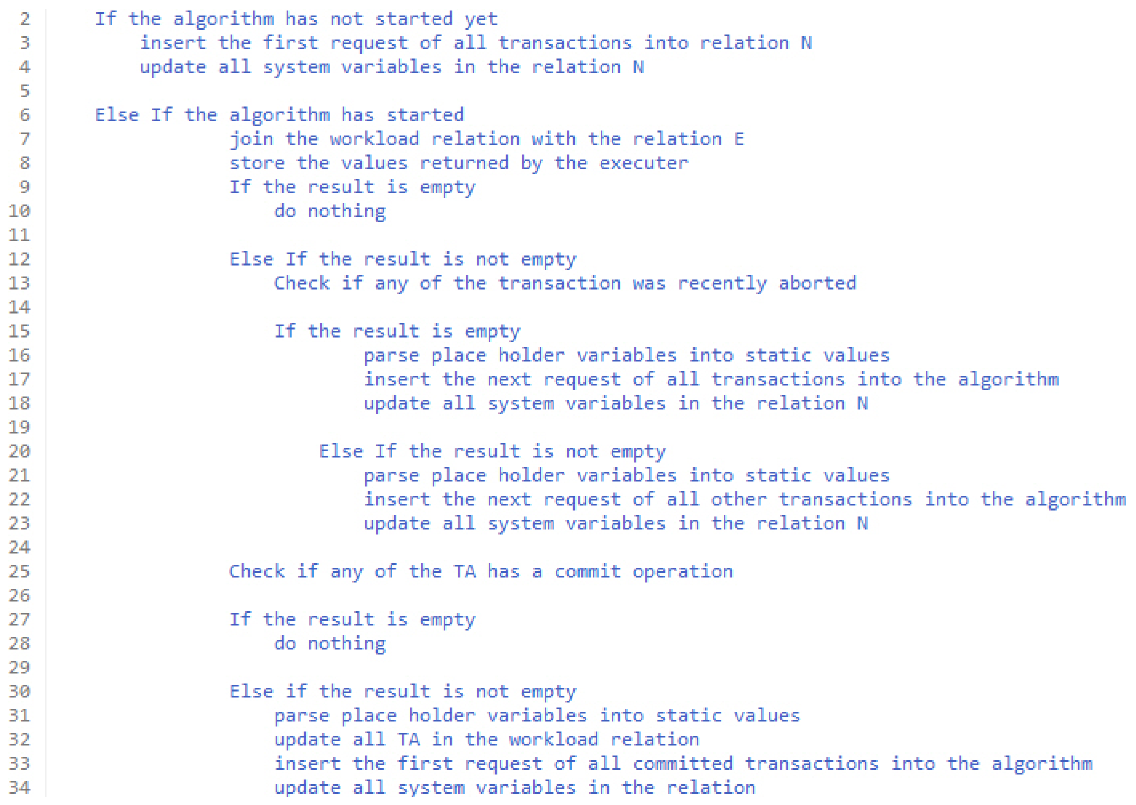
The Oshiya demo application allows the user to undo and redo steps of the Oshiya algorithm. That means, the workload generator must be able to select requests from relation Workload and insert them into relation $\mathcal{N}$, undo those steps and then select and insert them again. A difficulty that occurred during the development of the concept for the request selection algorithm was multiple transactions committing or aborting at the same time. In this case, new transaction ids have to be generated for each of the transactions. That is why the following solution has been chosen.

4.2.2 Request Selection Algorithm

Figure 5.2 shows the algorithm that is used to determine which requests will inserted into relation $\mathcal{N}$ in step n:2 of the Oshiya algorithm. The algorithm is executed in 5 steps.

- Step(1): In row 2 the algorithm checks if any requests have been inserted into relation $\mathcal{N}$ in step n-1:2. If no requests have been inserted, the first request of each transaction is inserted into relation $\mathcal{N}$. Then, the system variables of relation $\mathcal{N}$ are updated.
- Step(2): In row 7 relation Workload is joined with relation $\mathcal{E}$, this is done to check which requests of which transactions have been executed in step n-1:2.
- Step(3): In row 13 it is being checked, if any of the transactions were aborted in iteration step n-1:2
- Step(4): In rows 15 to 23 the requests of all transactions that did not abort recently are inserted into relation $\mathcal{N}$. After that, the system variables of relation $\mathcal{N}$ are updated.
- Step(5): In row 25 the algorithm checks if any transaction has been committed in step n-1:2. For all committed transactions, the transaction id is updated in relation Workload. Then the first request of all transactions that have been committed in step n-1:2 are inserted into relation $\mathcal{N}$. After that the system variables of relation $\mathcal{N}$ are updated.

Forward Mode
There are **three different cases** when requests are selected from relation Workload and inserted into relation $\mathcal{N}$.



```
If the algorithm has not started yet
    insert the first request of all transactions into relation N
    update all system variables in the relation N

Else If the algorithm has started
            join the workload relation with the relation E
            store the values returned by the executer
            If the result is empty
                do nothing

            Else If the result is not empty
                Check if any of the transaction was recently aborted

                If the result is empty
                        parse place holder variables into static values
                        insert the next request of all transactions into the algorithm
                        update all system variables in the relation N

                    Else If the result is not empty
                        parse place holder variables into static values
                        insert the next request of all other transactions into the algorithm
                        update all system variables in the relation N

            Check if any of the TA has a commit operation

            If the result is empty
                do nothing

            Else if the result is not empty
                parse place holder variables into static values
                update all TA in the workload relation
                insert the first request of all committed transactions into the algorithm
                update all system variables in the relation
```

Figure 4.2: Pseudo Code of the Request Selection Algorithm

Case 1: The Oshiya algorithm is in the initial state, where relations $\mathcal{N}$, $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{H}$ are empty. No request has been inserted into relation $\mathcal{N}$. This means that the first request of every transaction has to be inserted during the first iteration 1:2. Example 2 displays case 1 based on relation Workload from Example 1:

Example 2 *In this example, the first requests from transactions t_1 and t_2, o_1 and o_2 are inserted into relation $\mathcal{N}$. Then both requests are inserted into relation $\mathcal{R}$. After that the transaction ids are inserted into relation client_ta _mapping along with the client ids of the clients, the transactions belong to.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*1*	*1*	*r*	*1*	
1	*1*	*2*	*r*	*2*	
1	*1*	*3*	*w*	*1*	*5*
1	*1*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*1*	*r*	*1*		*1*
2	*2*	*1*	*r*	*1*		*1*

client_ta _mapping	
CID	*TA*
1	*1*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*1*	*r*	*1*		*1*
2	*2*	*1*	*r*	*1*		*1*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*

In case 1, the following actions have to be performed:

- insert the first request of every transaction t_i of relation Workload into relation $\mathcal{N}$
- update all columns of type SystemVariable in relation $\mathcal{N}$ with their current values
- insert the transaction and client ids into relation client_ta _mapping
- insert all requests from relation $\mathcal{N}$ into relation $\mathcal{R}$

Case 2: The Oshiya algorithm is in step n:2 with n>1. If request o_1 with o[seq] = i is not an abort or commit operation and has been inserted into relation $\mathcal{H}$ in scheduling iteration n-1, the next request o_2 with o_2[seq] = i+1 is selected from relation Workload for insertion into relation $\mathcal{N}$. Example 3 visualizes case 2 based on relation Workload from Example 1:

Example 3 *In this example, the Oshiya algorithm is in scheduling iteration 2:2. The request o_1, o_2 with o_1[seq] = 1, o_2[seq] = 1 from transactions t_1 and t_2 have been executed in scheduling iteration 1:4. Request o_3, o_4 with o_3[seq] = 2, o_4[seq] = 2 from t_1 and t_2 will be inserted into relation $\mathcal{N}$. Then both requests will be selected from relation $\mathcal{N}$ and inserted into relation $\mathcal{R}$.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*1*	*1*	*r*	*1*	
1	*1*	*2*	*r*	*2*	
1	*1*	*3*	*w*	*1*	*5*
1	*1*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*2*	*r*	*2*		*2*
2	*2*	*2*	*r*	*2*		*2*

client_ta _mapping	
CID	*TA*
1	*1*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*2*	*r*	*2*		*2*
2	*2*	*2*	*r*	*2*		*2*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*1*	*r*	*1*		*1*
2	*2*	*1*	*r*	*1*		*1*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*1*	*r*	*1*		*1*
2	*2*	*1*	*r*	*1*		*1*

In case 2, the following actions have to be performed:

- check for the last request o[seq] = i of every transaction t_i inserted into relation $\mathcal{N}$ in preceding scheduling iterations
- if request o[seq] = i has been inserted into relation $\mathcal{E}$ in scheduling iteration n-1, and therefore has been executed:

  - insert request o[seq] = i+1 into relation $\mathcal{N}$
- update all columns of type SystemVariable in relation $\mathcal{N}$ with their current values

Case 3: The algorithm is in step n:2. Multiple requests have been selected from relation Workload and inserted into relation $\mathcal{N}$ in preceding scheduling iterations. If a request o_i of a transaction t_i is a commit or abort operation, a new transaction id has to be created for transaction t. Then the first request of transaction t_i is then inserted into relation $\mathcal{N}$. Example 4 displays case 3 based on relation Workload from Example 1:

Example 4 *In this example, the Oshiya algorithm is in scheduling iteration 6:2. Transaction t_1 has been committed in scheduling iteration 5:4. A new transaction id has to be created for t_1. The new transaction id for t_1 will be t_3. After relation Workload has been updated with the new transaction id, the first request of transaction t_3 will be inserted into relation $\mathcal{N}$. Then the new transaction id is inserted into relation client_ta _mapping.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*3*	*1*	*r*	*1*	
1	*3*	*2*	*r*	*2*	
1	*3*	*3*	*w*	*1*	*5*
1	*3*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
2	*2*	*3*	*w*	*2*	*6*	*4*
1	*3*	*1*	*r*	*1*		*6*

client_ta _mapping	
CID	*TA*
1	*1*
1	*3*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
2	*2*	*3*	*w*	*2*	*6*	*4*
1	*3*	*1*	*r*	*1*		*6*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*2*	*r*	*2*		*3*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*2*	*r*	*2*		*3*

In case 3, the following actions have to be performed:

- check for the last request o[seq] = i of every transaction t_i inserted into relation $\mathcal{N}$ in scheduling iteration n-1.
- if request o[seq] = i has been inserted into relation $\mathcal{E}$ in scheduling iteration n-1, and is a commit or abort operation:

  - create a new transaction id for the transaction

  - insert the transaction and client ids into the client_ta _mapping relation

  - update the transaction id in relation Workload

  - insert the first request of the transaction

- update all columns of type SystemVariable in relation $\mathcal{N}$ with their current values

Backward Mode
The user has the opportunity to undo steps of the Oshiya algorithm. If the user decides to undo one step from step n:2 to n-1:1, information has to be restored. The scheduling relations $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{H}$ are restored by the Oshyia demo application. In order to restore relation Workload, it has to be checked, if there is a transaction t_i that has been committed or aborted in scheduling iteration n-1. If transaction t_i has been committed or aborted, the old transaction id t_i has to be restored. Example 5 displays this functionality for Example 1:

Example 5 *In this example, the Oshiya algorithm is in scheduling iteration 6:2. The user has decided to undo the last step. The Oshiya algorithm is now in scheduling iteration 6:1. In order to restore relation Workload, the old transaction id of t_3 has to be restored. The transaction id of transaction t_3 in relation Workload will be updated with its old value. The following tables display scheduling iteration step 6:1 after relations Workload, $\mathcal{N}, \mathcal{R}, \mathcal{E}$ and $\mathcal{H}$ have been restored.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*1*	*1*	*r*	*1*	
1	*1*	*2*	*r*	*2*	
1	*1*	*3*	*w*	*1*	*5*
1	*1*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*3*	*w*	*2*	*6*	*6*

client_ta _mapping	
CID	*TA*
1	*1*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*3*	*w*	*2*	*6*	*6*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*3*	*w*	*1*	*5*	*4*
2	*2*	*2*	*r*	*2*		*3*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*3*	*w*	*1*	*5*	*4*
2	*2*	*2*	*r*	*2*		*3*

The following actions have to be performed:

- check for the last request o[seq] = i of every transaction t_i inserted into relation $\mathcal{N}$ in scheduling iteration n-1:2
- if request o[seq] = i has been inserted into relation $\mathcal{E}$ in scheduling iteration n-1:4, and is a commit or abort operation:

  - delete the highest transaction id of the client the transaction belongs to from

  the client_ta _mapping relation

  - update the transaction id in relation Workload with the highest transaction

  - id of the client the transaction belongs to from the client_ta _mapping relation

4.3 Static and Dynamic Workloads

Recall relation Workload of Example 1:

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	r	2	
1	1	3	w	1	5
1	1	4	c		
2	2	1	r	1	
2	2	2	r	2	
2	2	3	w	2	6
2	2	4	c		

In this example,the values for item 1 and 2 can be written with the following queries:

UPDATE N **SET** VAL = 5 **WHERE** OB = 1

UPDATE N **SET** VAL = 6 **WHERE** OB = 2

Any value held by item 1 and 2 will be lost.

There are real-world transactions that perform changes to existing data items, for example, increasing the salary of all employees by ten percent. In order to enable the user to create transactions like that, the user has to be able to modify the values of an item when a data item is written. The value for employee 1 can be written with the following query:

UPDATE N **SET** VAL = VAL∗1.1 **WHERE** OB = 1

The workload generator supports two different types of Workloads. *Static workloads* are workloads, where the values that are written on objects are static integer numbers. The table of the Workload Relation of example 1, is an example for a static workload.

The second type of workload is the *dynamic workload*. When creating a dynamic workload, the user can specify *place holder variables* in the column 'Val'. A place holder variable is specified as a single '$' and the sequence number (Column 'Seq') of a read operation of the same transaction (e.g. $1). The place holder variable holds a reference to the value returned by the read operation with the corresponding attribute 'seq'. The value written on an item will consist of arithmetic operations on one or more place holder variables. This will allow the user for example to increase the salary of employee 1 who currently earns fifty Euro by ten percent. The following table displays this example:

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	3	w	1	$1*1.1
1	1	4	c		

Requests that have been scheduled for execution by the Oshiya scheduling model are executed by the *Executor*. The workload generator replaces $1 with the value returned by the executor. This value is then inserted in relation $\mathcal{N}$.

5 Implementation

In the last chapter, the concept and design for the workload generator have been explained. In this chapter, the implementation of different classes that were needed for the workload generator will be described. The implementation of the request selection algorithm will be explained (section 5.1). Then GUI elements of the workload generator, as well as the menu structure of the workload generator will be displayed (section 5.2 and 5.3). After that, patterns will be explained (section 5.4). At last, examples from the library that has been implemented are displayed (section 5.5). Figure 5.1 displays an UML Class Diagram of the classes that have been created in order to implement the workload generator.

5.1 Implementation Request Selection Algorithm

In this section, the SQL statements used in the different steps of the request selection algorithm will be shown and explained.

The SQL statements that were created for the request selection algorithm are called *System Queries*. The System Queries used in the request selection algorithm had to be written dynamically. The schema that is used for the scheduling relations can be changed by the user by adding additional columns to the existing six columns ('cid', 'ta', 'seq', 'op', 'ob', 'val'). The SQL statements have to be independent from the structure of the scheduling relations. They also have to be dynamic concerning the type of the values they process. In order to create dynamic SQL queries, SQL statements will be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times with different values for the specified variables [3].

The Oshiya demo application supports three database management systems: PostgreSQL, MSSQL and Oracle. Therefore all SQL statements used in the request selection algorithm had to be written so they are syntactically correct for all three database management systems at the same time.

The variable *sColumnsString* is used to store the schema of relation Workload. The column SI is a SystemVariable that displays the scheduling iteration step when the request has been inserted into the relation.

In order to allow the user to undo steps of the Oshiya algorithm, all information stored in the scheduling relations has to be stored. This is done in *system relations*. The following table displays the schema of $\mathcal{E}_{Sys}$, the system relation for scheduling relation $\mathcal{E}$.

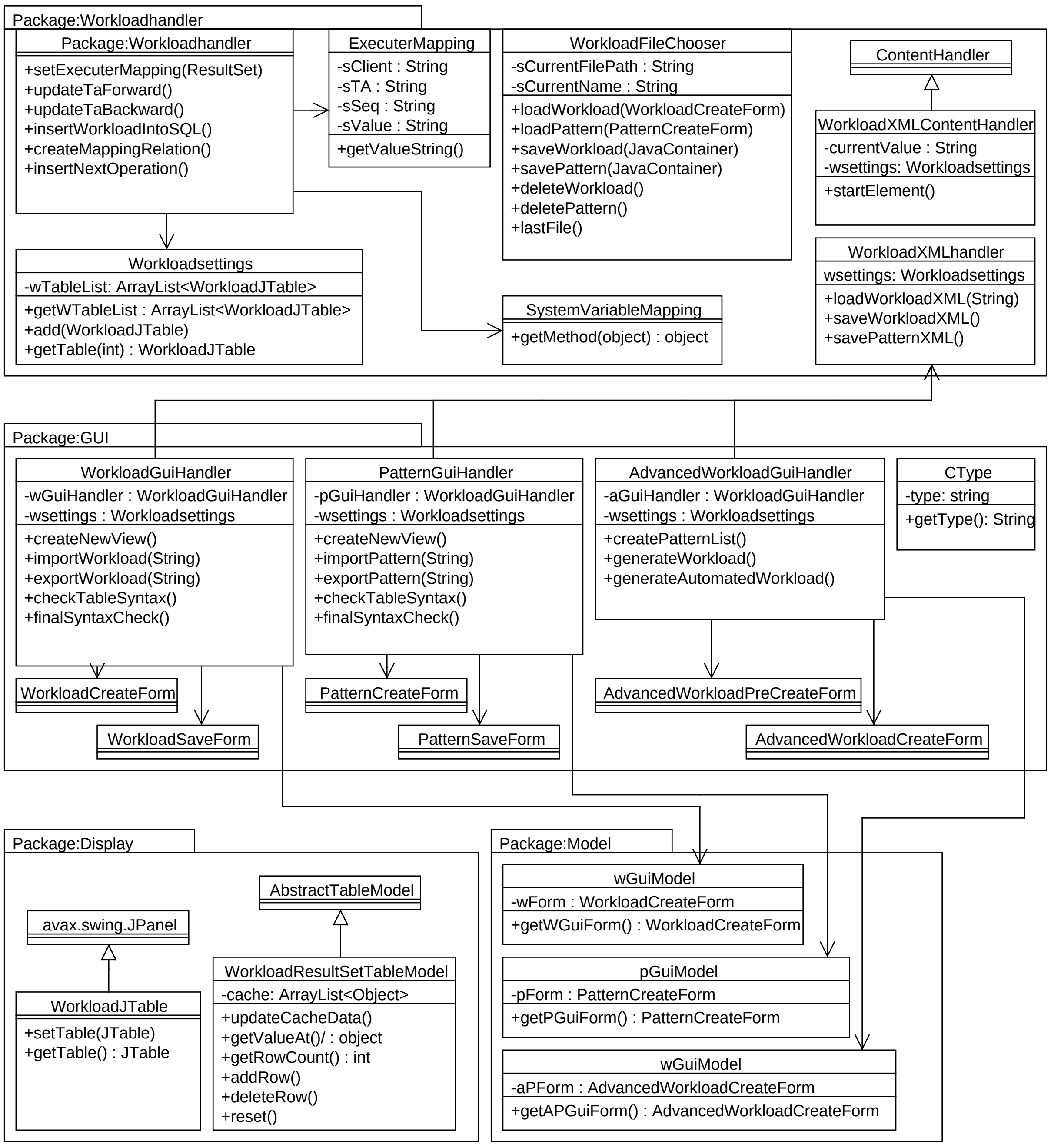


Figure 5.1: UML Class Diagram Workload Generator

$\mathcal{E}_{Sys}$								
CID	TA	Seq	Op	Ob	Val	SI	BeginINT	EndINT

The column *BeginInt* stores the scheduling iteration step, in which the request has been inserted into relation $\mathcal{E}$. The column *EndINT* stores the scheduling iteration step, in which a request has been selected for execution. Because the request selection algorithm is selecting requests from relation Workload in step 2 of the Oshiya algorithm, and requests in relation $\mathcal{E}$ will be executed in step 4 of the Oshiya algorithm, the value for column EndInt will be NULL for the last request that has been added to relation $\mathcal{E}$ when checked in step 2 of the Oshiya algorithm. That means in order to find the last request of a transaction that has been executed, it is being checked which requests in relation $\mathcal{E}_{Sys}$ have an EndInt value of NULL. In the following, the System Queries that were used for the five steps of the request selection algorithm will be displayed.

In **step (1)** of the request selection algorithm, the Oshiya algorithm is in its initial state and the first request of all transactions is inserted into relation $\mathcal{N}$ by executing SQ1.

```
SQ1:
INSERT INTO N(sColumnsString)
SELECT sColumnsString FROM WORKLOAD WHERE Seq = 1
```

After the requests have been inserted into relation $\mathcal{N}$, the columns of type SystemVariable in relation $\mathcal{N}$ are updated with their runtime values. This SQL statement is executed in iterations for every column of type SystemVariable (as explained in Section 4.1) that exists in relation $\mathcal{N}$. The variable SystemVariableValue, for example the SchedulingIteration step, is generated by the application. The System Query SQ2 is used to update all columns of type SystemVariable in relation $\mathcal{N}$ and is executed in iterations, once for each SystemVariable.

```
SQ2:
UPDATE N SET SysColumn = SystemVariableValue
WHERE TA IN(SELECT TA FROM WORKLOAD WHERE Seq = 1
```

All transaction ids and client ids that exist in relation Workload are added to the client_ta _mapping relation by executing System Query SQ3.

```
SQ3:
INSERT INTO CLIENT_TA_MAPPING(CID,TA)
SELECT DISTINCT CID,TA FROM WORKLOAD
```

Example 6 *Recall Example 3 from Section 4.2.2. The first requests of t_1 and t_2 are selected from relation Workload and inserted into relation $\mathcal{N}$ (SQ1, SQ2). New requests in relation $\mathcal{N}$ are inserted into relation $\mathcal{R}$ by the Oshiya algorithm. Then the transaction ids of t_1 and t_2 are inserted into relation client_ta _mapping (SQ3).*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*1*	*1*	*r*	*1*	
1	*1*	*2*	*r*	*2*	
1	*1*	*3*	*w*	*1*	*5*
1	*1*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*1*	*r*	*1*		*1*
2	*2*	*1*	*r*	*1*		*1*

client_ta _mapping	
CID	*TA*
1	*1*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*1*	*r*	*1*		*1*
2	*2*	*1*	*r*	*1*		*1*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*

In **step (2)** of the request selection algorithm, relation Workload is joined with relation $\mathcal{E}_{Sys}$ based on the transaction id by executing System Query SQ4. This is done to check which requests have been executed in scheduling iteration n-1:4.

```
SQ4:
SELECT CID,TA, Seq, Op, Ob, Val
FROM WORKLOAD WHERE TA IN (SELECT TA FROM E_sys)
```

In **step (3)** of the request selection algorithm, it is checked if any transaction, that has executed a request in scheduling iteration n-1:4, has been aborted (SQ5).

```
SQ5:
SELECT CID,TA, Seq, Op, Ob, Val
FROM WORKLOAD
WHERE TA IN (SELECT TA FROM E_sys WHERE OP = 'a' AND ENDINT is NULL)
```

In **step (4)** of the request selection algorithm, it is being checked if any transaction has been committed or aborted in scheduling iteration n-1:4. Only requests from transactions that did not commit or abort in scheduling iteration n-1:4 will be selected for insertion into relation $\mathcal{N}$.

If no transaction has been aborted or committed, System Queries SQ6 and SQ7 will be executed.

The next request for all transactions that have executed a request in scheduling iteration n-1:4, will be inserted into relation $\mathcal{N}$.

```
SQ6:
INSERT INTO N (sColumnsString)
SELECT sColumnsString FROM WORKLOAD W, E_SYS E
WHERE W.Seq = E.Seq+1 AND W.Ta = E.Ta AND E.Op != 'c'
AND E.EndInt is NULL)
```

After the requests have been inserted into relation $\mathcal{N}$, the columns of type SystemVariable in relation $\mathcal{N}$ are updated with their runtime values.

```
SQ7:
UPDATE N SET SysColumn = SystemVariableValue
WHERE TA IN (SELECT W.TA FROM WORKLOAD W, E_SYS
WHERE W.Seq = E_SYS.Seq+1 AND W.Ta = E_SYS.Ta AND E_SYS.Op != 'c'
AND E_SYS.EndInt is NULL)
```

If one or more transactions have been aborted, SQ8 and SQ9 will be executed.

The next request for all transactions that have executed a request and have not been aborted or committed in scheduling iteration n-1:4, will be inserted into relation $\mathcal{N}$.

```
SQ8:
INSERT INTO N(sColumnsString)
SELECT sColumnsString FROM WORKLOAD W, E_SYS
WHERE W.Seq = E_SYS.Seq+1 AND W.Ta = E_SYS.Ta AND E_SYS.Op != 'c'
AND E_SYS.Op != 'a' AND E_SYS.EndInt is NULL
```

After the requests have been inserted into relation $\mathcal{N}$, the columns of type SystemVariable in relation $\mathcal{N}$ are updated with their runtime values.

```
SQ9:
UPDATE N SET SysColumn = SystemVariableValue
WHERE TA IN (SELECT W.TA FROM WORKLOAD W, E_SYS
WHERE W.Seq = E_SYS.Seq+1 AND W.Ta = E_SYS.Ta AND E_SYS.Op != 'a'
AND E_SYS.Op != 'c' AND E_SYS.EndInt is NULL)
```

Example 7 *Recall Example 4 from Section 4.2.2. No transaction has been committed or aborted. The first requests of t_1 and t_2 have been executed in scheduling iteration n-1:4. Therefore the next requests have to be selected from relation Workload and inserted into relation $\mathcal{N}$. The requests o_1 with o[seq] = 2, o_2 with o[seq] = 2 from t_1 and t_2 are selected from relation Workload and inserted into relation $\mathcal{N}$ (SQ4, SQ5 and SQ6, SQ7 / SQ8, SQ9). New requests in relation $\mathcal{N}$ are inserted into relation $\mathcal{R}$ by the Oshiya algorithm.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*1*	*1*	*r*	*1*	
1	*1*	*2*	*r*	*2*	
1	*1*	*3*	*w*	*1*	*5*
1	*1*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*2*	*r*	*2*		*2*
2	*2*	*2*	*r*	*2*		*2*

client_ta _mapping	
CID	*TA*
1	*1*
2	*2*

$\mathcal{R}$							$\mathcal{E}$							$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*	*CID*	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*	*CID*	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*2*	*r*	*2*		*2*	*1*	*1*	*1*	*r*	*1*		*1*	*1*	*1*	*1*	*r*	*1*		*1*
2	*2*	*2*	*r*	*2*		*2*	*2*	*2*	*1*	*r*	*1*		*1*	*2*	*2*	*1*	*r*	*1*		*1*

In **step (5)** of the request selection algorithm, it is checked if any transaction has been committed in scheduling iteration n-1:4 (SQ10).

```
SQ10:
SELECT TA FROM E_Sys
WHERE Op = 'c' AND ENDINT is NULL
```

If one or more transaction have been committed in scheduling iteration n-1:4, new transaction ids have to be generated so the transactions can be restarted (as shown in section 4.1). The generation of new transaction ids will be shown in Section 5.1.1.

Then the first request of all transactions that have been committed in scheduling iteration n-1:4 is inserted into relation $\mathcal{N}$ (SQ11).

```
SQ11:
INSERT INTO N(sColumnsString)
SELECT sColumnsString FROM WORKLOAD
WHERE Seq = 1 AND TA NOT IN (SELECT TA FROM R)
AND TA NOT IN(SELECT TA FROM E_SYS)
AND TA NOT IN(SELECT TA FROM N)
```

After the requests have been inserted into relation $\mathcal{N}$, the columns of type SystemVariable in relation $\mathcal{N}$ are updated with their runtime values (SQ12).

```
SQ12:
UPDATE N SET SysColumn = SystemVariableValue
WHERE TA IN (SELECT TA FROM WORKLOAD
WHERE Seq = 1 AND TA NOT IN(SELECT TA FROM E_SYS))
```

5.1.1 Creating new Transaction Ids

When new transaction ids have to be created, the System Queries SQ13-SQ16 are executed. The System Queries SQ13-SQ16 are executed in iterations, once for each transaction that has been committed.

In a first step, all transactions that have been committed in scheduling iteraton n-1:4 are selected (SQ13).

```
SQ13:
SELECT TA FROM WORKLOAD
WHERE TA NOT IN (SELECT TA FROM E_SYS WHERE Op='c' AND EndInt is NULL)
```

Then for each transaction, the System Queries SQ14 and SQ15 are executed. First, the highest transaction id is selected from relation client_ta _mapping (SQ14).

```
SQ14:
SELECT max(TA) as M FROM CLIENT_TA_MAPPING
```

System Query SQ15 will be used to update the ids of transactions in relation Workload. The new transaction id will be the highest transaction selected from relation client_ta _mapping id plus one. The query has been written as a prepared statement.

```
SQ15:
UPDATE WORKLOAD SET TA = ? WHERE TA = ?
```

After relation Workload has been updated, the new transaction ids are added to relation client_ta _mapping (SQ16).

```
SQ16:
INSERT INTO CLIENT_TA_MAPPING(CID,TA)
SELECT DISTINCT CID,TA FROM WORKLOAD
WHERE TA NOT IN(SELECT TA FROM E_SYS)
AND TA NOT IN (SELECT TA FROM CLIENT_TA_MAPPING)
```

Example 8 *Recall Example 4 from Section 4.2.2. t_1 has been committed in scheduling iteration n-1:4. A new transaction id has been created and inserted into relation client_ta _mapping (SQ13, SQ14, SQ16). Then the transaction id of t_1 has been updated in relation Workload (SQ15). Transaction t_1 is restarted transaction t_3. The first request of t_3 is selected from relation Workload and inserted into relation $\mathcal{N}$ (SQ10, SQ11, SQ12). New requests in relation $\mathcal{N}$ are then inserted into relation $\mathcal{R}$ by the Oshiya algorithm.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*3*	*1*	*r*	*1*	
1	*3*	*2*	*r*	*2*	
1	*3*	*3*	*w*	*1*	*5*
1	*3*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*3*	*1*	*r*	*1*		*6*

client_ta _mapping	
CID	*TA*
1	*1*
1	*3*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
2	*2*	*3*	*w*	*2*	*6*	*4*
1	*3*	*1*	*r*	*1*		*6*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*2*	*r*	*2*		*3*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*2*	*r*	*2*		*3*

5.1.2 Restoring old Transaction Ids

When the user decides to undo one step of the Oshiya algorithm, one step after a transaction has been restarted with a new transaction id, the old transaction id has to be restored. In order to do that, the System Queries SQ17-SQ20 are executed: First all transactions that have been committed in scheduling iteration n-1:4 are selected from relation Workload. The System Queries SQ17-SQ20 will be executed in iterations, once for each transaction id that has to be restored. First all transactions that have been committed are selected (SQ17).

```
SQ17:
SELECT TA FROM WORKLOAD
WHERE TA NOT IN (SELECT TA FROM E_SYS
WHERE Op='c' AND EndInt is NULL)
```

Then the selected transaction ids are removed from relation client_ta _mapping relation (SQ18).

```
SQ18:
DELETE FROM CLIENT_TA_MAPPING
WHERE TA NOT IN (SELECT TA FROM E_SYS)
```

After that, the highest transaction id for each client will be selected from relation client_ta _mapping. The variable currentTA, stores the currently selected transaction, for which the transaction id has to be restored (SQ19).

```
SQ19:
SELECT max(C.TA) as M
FROM CLIENT_TA_MAPPING C
WHERE C.CID IN(SELECT CID FROM WORKLOAD
WHERE TA = currentTA
```

System Query SQ20 is a prepared statement. It will be used to update the transaction ids of transactions that have to be restored with the highest transaction id selected from relation client_ta _mapping plus one.

```
SQ20:
UPDATE WORKLOAD SET TA = ? WHERE TA = ?
```

Example 9 *Recall Example 5 from Section 4.2.2. The Oshiya algorithm is in scheduling iteration 6:2. The user has decided to undo the last step. In scheduling iteration 6:1, t_1 has been committed. A new transaction id has been created and transaction t_1 became t_3. The Oshiya algorithm is now in scheduling iteration 6:1. First t_3 is selected (SQ17). Then the old transaction id of t_3 is restored in relation Workload (SQ18, SQ19, SQ20). The scheduling relations $\mathcal{R}, \mathcal{E}$ and $\mathcal{H}$ are restored by the Oshiya algorithm.*

Workload					
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*
1	*1*	*1*	*r*	*1*	
1	*1*	*2*	*r*	*2*	
1	*1*	*3*	*w*	*1*	*5*
1	*1*	*4*	*c*		
2	*2*	*1*	*r*	*1*	
2	*2*	*2*	*r*	*2*	
2	*2*	*3*	*w*	*2*	*6*
2	*2*	*4*	*c*		

$\mathcal{N}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*

client_ta _mapping	
CID	*TA*
1	*1*
2	*2*

$\mathcal{R}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*4*	*c*			*5*
2	*2*	*3*	*w*	*2*	*6*	*4*

$\mathcal{E}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*3*	*w*	*1*	*5*	*4*
2	*2*	*2*	*r*	*2*		*3*

$\mathcal{H}$						
CID	*TA*	*Seq*	*Op*	*Ob*	*Val*	*SI*
1	*1*	*3*	*w*	*1*	*5*	*4*
2	*2*	*2*	*r*	*2*		*3*

5.2 Creating the Workload

A GUI has to be created, that allows the user to create and test different workloads. The GUI must be user-friendly and guide the user through the process of creating and testing workloads. In addition, false input must be handled. A syntax checker must check workloads for errors, before they can be stored in relation Workload. The input created by the user must be stored efficiently. Good performance is important for the request scheduling. Bad performance will cause delays between the request processing. In presentations, the scheduling process must run without large delays because the scheduling of the requests is displayed graphically in the Oshiya demo application.

The GUI has to offer good usability. Before a workload can be created, protocol and scheduling relations have to be created. After a workload has been created, the user has to have ability to go back and change the protocol without having to recreate the workload. There also has to be the opportunity to apply changes to workloads after they have been created. The user has to have the ability to create, schedule and change workloads in iterations.

That is why, in order to create a workload, the user must complete the following steps: First she must chose a connection to a database system. Then she must choose or create a protocol. After that she has to create the scheduling relations and save her selection. The user then can decide what type of workload she wants to create or load an already existing workload. The GUI must guide the user through these different steps. This is achieved by disabling functionalities that require pre-steps that have not been completed. In order to maximize the performance, only the finalized workloads or patterns are stored in SQL relations. Everything else is stored in internal java structures.

A user might want to create large workloads, consisting of many transactions with a lot of requests. Creating such a workload manually requires a lot of time and effort. In order to provide functionality to the user that allows the creation of such workloads more easily, two different types of workload will be defined. Workloads that are created manually by the user will be referred to as *simple workloads*. Workloads that are generated semi-automatically by the application based on the specifications of the user will be referred to as *advanced workloads*.

The behaviour of some real-world transactions follows *patterns*. In order to simplify the creation of real-world transactions, the user has the opportunity to design and create patterns in order to generate workloads. It was decided, that an advanced workload can be created in one of two ways: based on a pattern, or based on generic settings. A pattern is a template the user can create manually. This template can be used to generate large workloads semi-automatically. The *pattern generator* is a GUI form that allows the user to create templates for advanced workloads.

The *simple workload generator* is a GUI form that allows the user to create a set of clients. The user can enter requests for the transaction manually. The columns client id, transaction id and sequence number are generated by the application and cannot be modified by the user. This is to ensure that each client only has one transaction. These columns will be displayed to the user, in order to provide a complete picture of a transaction.

The *advanced workload generator* consists of two GUI forms that allow the user to generate *partial generated workloads* and *generated workloads*. Generated workloads are normal workloads that have been created semi-automatically by the application based on generic settings and/or templates the user has specified. One partial workload is generated based on one pattern and one set of generic settings. Multiple partial generated workloads can be added to one generated workload. The structure of the advanced workload generator is displayed in figure 4.3. This design allows the user to create workloads in iterations by adding up multiple partial workloads. It allows the user to create one workload based on multiple different patterns and different generic settings. This provides flexibility to the user when designing generated workloads.

5.3 Managing the Workload

The schema of the tables in the GUI elements, that are used to create workloads, must be equal to the schema of the scheduling relations $\mathcal{R}$, $\mathcal{E}$ and $\mathcal{H}$ specified in the protocol at all times. Requests in relation Workload that are based on a different schema than the scheduling relations can lead to wrong behaviour or errors in the Oshiya scheduling model. In order to avoid that, the schema of the workload tables is created based on the schema of the scheduling relations. If the user changes the schema of the scheduling relations, the schema of the workload tables is re-created automatically. Workloads created based on a different schema of a protocol, can be be used even after the schema of the scheduling relations for that protocol have been changed.

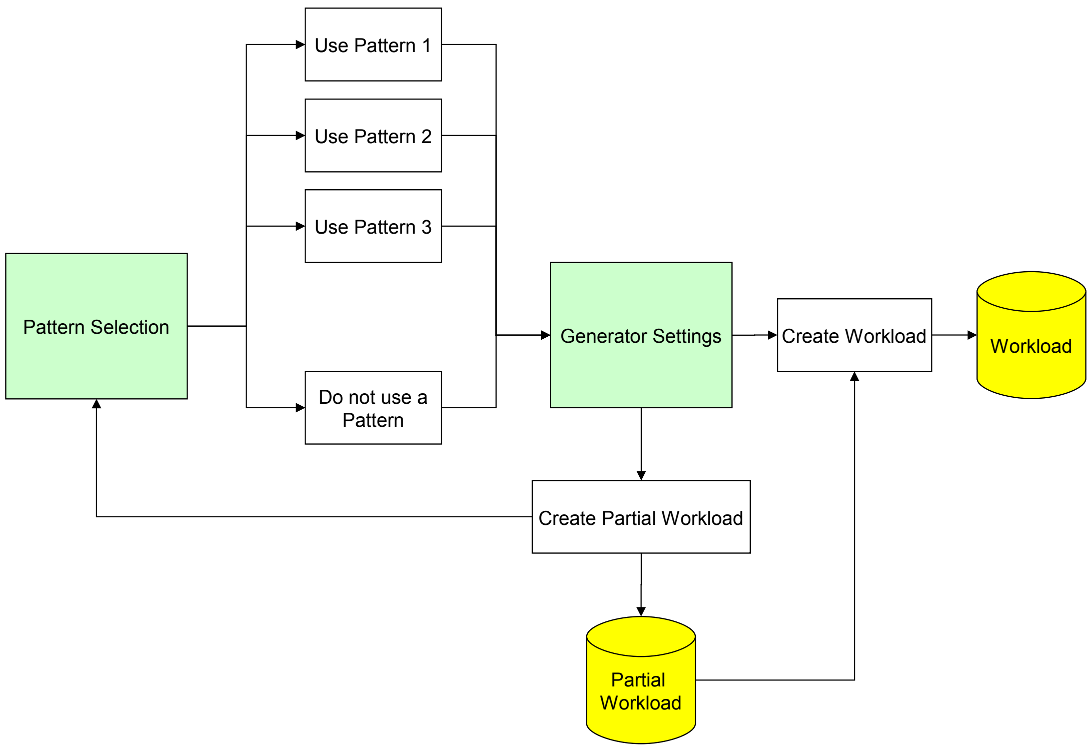


Figure 5.2: Structure of the Advanced Workload Generator

The columns of type SystemVariable are generated by the application. The user cannot change the values in those columns. It makes sense for these columns to be invisible to the user. It was decided, that the workload tables in the GUI will consist of six basic columns and possible additional columns of type UserInput. Columns of type SystemVariable will not be part of the schema of relation Workload.

Recall relation Workload from Example 1, the following table displays the workload and the basic schema of relation Workload:

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	r	2	
1	1	3	w	1	5
1	1	4	c		
2	2	1	r	1	
2	2	2	r	2	
2	2	3	w	2	6
2	2	4	c		

In order to test multiple protocols with multiple workloads, the user has to have the ability to save and load workloads. Every time a protocol is created a folder for the protocol is created as well. This folder is used to store workloads belonging to the protocol. This is done because workloads are protocol-specific. A user can save or load all types of workload in a xml file. This allows the user to create a library of workloads and to exchange them with other users. The workloads can also be exchanged between different users of the Oshiya demo application.

The workload generator offers a lot of different functionality to the user. In order to enable the user to manage this functionality in a user-friendly way, the *Workload Menu* will be created. It will consist of 3 parts: *Manage Patterns*, *Manage Workloads* and *Recently used workloads*. The Manage Patterns menu allows the user to delete, load and create patterns with the help of separate GUI elements. The Manage Workloads menu allows the user to delete, load and create simple and advanced workloads. It also allows the user to edit the *currently active workload*. Figure 4.2 displays the structure of the workload menu.

5.4 Creating Patterns

Imagine a company pays the salary to three employees at the end of the month. Item 1 represents the bank account of the company, items 2, 3 and 4 represent the bank accounts of the employees. The workload created based on this example can be the following:

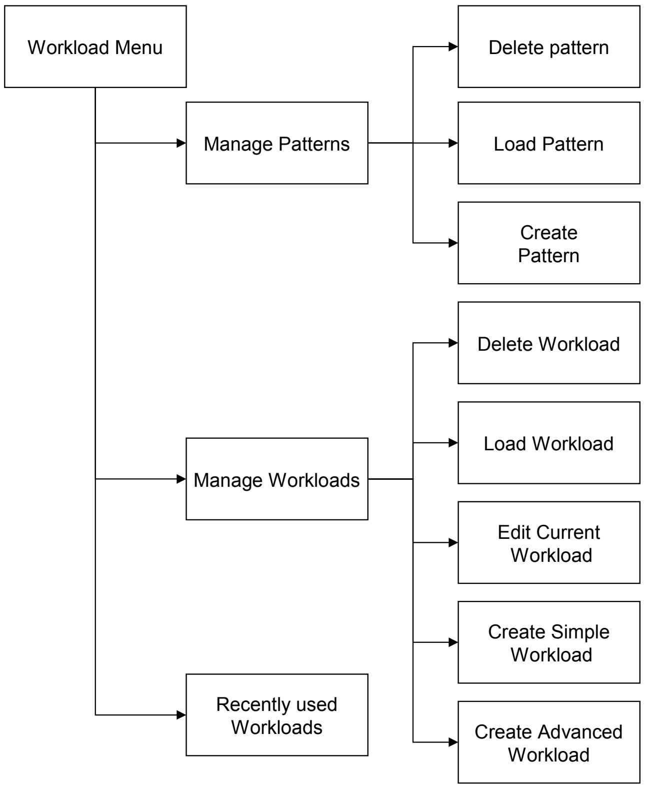


Figure 5.3: Structure of the Workload Menu

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	w	1	$1-150
1	1	3	r	2	
1	1	4	w	2	$3+50
1	1	5	r	3	
1	1	6	w	3	$5+50
1	1	7	r	4	
1	1	8	w	4	$7+50

A workload like that can be created manually by the user. But if the company had 300 employees instead of 3, creating the workload manually is a lot of effort. The workload for this transaction contains a pattern. This pattern is displayed in the following table:

Pattern			
Seq	Op	Ob	Val
1	r	1	
2	w	1	$1-50
3	r	2	
4	w	2	$3+50

A pattern is a template that consists of four columns (sequence number, operation, object, value). This template can be used to generate the salary payments for all 300 employees semi-automatically by duplicating the three requests of the template and only changing the number of the bank account of the employee for each duplicate. The advanced workload generator allows the user to do so.

5.5 Flexibility vs. Usability

By allowing the user to create simple workloads manually and advanced workloads semi-automatically the user is offered high flexibility when creating workloads. But flexibility is not the only important aspect for the Oshiya demo application. Usability is also important. In order to provide that to the user, a library of patterns and workloads has been created. In the following, three workloads and one pattern that are in the library will be displayed.

Simple Bank Transaction: Ten Euro are subtracted from bank account 1 and added to bank account 2.

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	w	1	$1-10
1	1	3	r	2	
1	1	4	w	2	$2+10
1	1	5	c		

Stock booking transaction: This workload displays two transactions that try to access the same stock concurrently. t_1 wants to add 10 items to stock 1 and 20 items to stock 2. t_2 wants to subtract 5 items from stock 1 and add 50 items to stock 2.

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	w	1	$1+10
1	1	3	r	2	
1	1	4	w	2	$3+20
1	1	5	c		
2	2	1	r	1	
2	2	2	w	1	$1-5
2	2	3	r	2	
2	2	4	w	2	$3+50
2	2	5	c		

Write Skew Anomaly: The *Snapshot Isolation Protocol* will be supported by the Oshiya demo application in the near future. This workload displays the write skew anomaly that can occur under this protocol.

Workload					
CID	TA	Seq	Op	Ob	Val
1	1	1	r	1	
1	1	2	r	2	
1	1	3	w	1	$1+5
1	1	4	c		
2	2	1	r	1	
2	2	2	r	2	
2	2	3	w	2	$2+6
2	2	4	c		

Bonus Salary: This pattern allows the user to create a workload, that can consist of one or more bonus salary payments from a company to its employees. Every employee is getting a salary bonus of ten percent.

Pattern			
Seq	Op	Ob	Val
1	r	1	
2	w	1	$1*1.1

6 Future Work

In its current implementation, the Oshiya demo application does not support multiversion concurrency protocols. A future task will be to extend the existing application in order to support the execution of multiversion concurrency control protocols such as Snapshot Isolation [4].

Currently, the application allows the user to develop and test one protocol at a time. In a future project, the Oshiya demo application shall be extended in order to support the concurrent execution of two scheduling protocols allowing to compare their executions [4].

In its current state, the Oshiya demo application displays the Oshiya algorithm. In a future task, the application shall be extended in order to allow the user to collect and display statistical information about the behaviour of protocols in order to compare and analyse them.

7 Summary

The existing Oshiya demo application allowed the user to create protocols and test them with transactions that consist of random requests. In order to enable the user to test protocols for specific behaviour or properties, the workload generator feature was added to the existing Oshiya demo application.

In the preceding elaborations, the concept and design for the workload generator have been displayed. The focus of this thesis has been to develop and implement this feature into the Oshiya demo application. The workload generator enables the user to create customized transactions. This allows the user to simulate different clients executing transactions on the scheduler in order to analyse the behaviour of a protocol. During the development different problems had to be solved. Those problems have been described in Section 4.1.2.

One of the major challenges, was the creation of the request selection algorithm. The algorithm selects the right requests from relation Workload in step 2 of the Oshiya algorithm and inserts them into relation $\mathcal{N}$. The algorithm has been created based on System Queries. These queries were implemented dynamically as prepared statements. This means, that the request selection algorithm is independent from the schema of the scheduling relations or the type of database system. This allows the user to customize the schema of protocols. An infinite mode for transactions has been introduced. It allows the user to run transactions in iterations. This enables the user to test protocols more thorough. Transactions, that are executed in infinite mode get new transaction ids, once they have been committed or aborted. These transaction ids are generated automatically and stored in the client_ta _mapping relation. The user has the opportunity to undo steps of the Oshiya algorithm. Because of that, the chosen solution allows the user to select requests from relation Workload and insert them into relation $\mathcal{N}$, undo those steps and then select and insert them again.

In order to assist the user in creating workloads, a GUI has been created. The GUI guides the user through the process of creating workloads. The user has the opportunity to exchange workloads with other users of the application. A library of workloads and patterns has been created. This library allows the user to use different real-world scenarios in order to test protocols. This library has been presented in Section 5.3.

8 Appendix

In this section, a scenario is described that displays an advanced workload. The different GUI elements that are needed to create the advanced workload, will be displayed in section 8.2 and 8.3.

8.1 Scenario

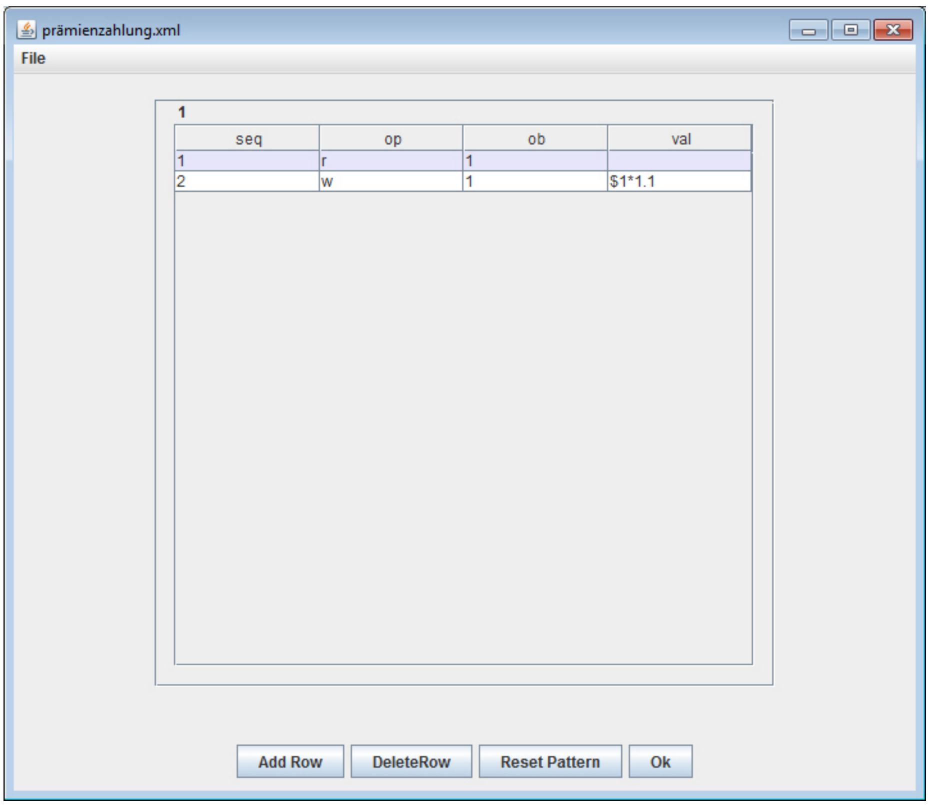
Consider the following scenario: A company wants to award successful employees with a ten percent salary bonus. Depending on the amount of employees the company has, this can lead to a large workload. In the following, we show how to generate an advanced workload for this example that is based on a pattern.

8.2 Simple Pattern Generator

The simple pattern generator allows the user to create a template, that she can duplicate. In this pattern, object 1 stands for the salary the employee earns. It is then increased by ten percent. The simple pattern generator is displayed in Figure 9.1.

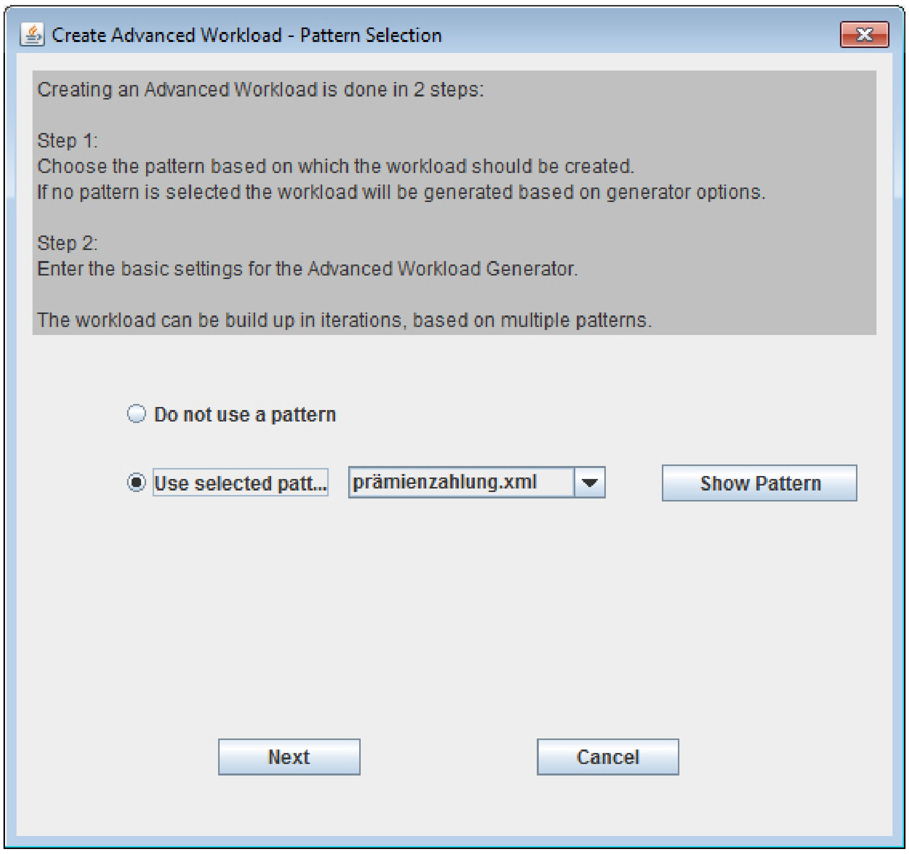
8.3 Advanced Workload Generator

The advanced workload generator can now be used to generate a workload for as many employees as needed. In this example, three employees will be awarded a ten percent bonus. First the pattern has to be selected. This is done in Figure 9.2. After the user has selected the pattern ’prämienzahlung’, she has to specify the amount of clients that shall be generated. In this scenario, one client will be created. This is done in Figure 9.3. After that, the user has to specify the multiplicator for the template pattern. In this case this means the amount of employees that shall be awarded a bonus. In this scenario, three employees will be awarded a bonus. This is shown in Figure 9.4. After the advanced workload has been generated, it is displayed to the user. Figure 9.5 displays the result.



seq	op	ob	val
1	r	1	
2	w	1	$1*1.1

Figure 8.1: Scenario - Template


Create Advanced Workload - Pattern Selection

Creating an Advanced Workload is done in 2 steps:

Step 1:
Choose the pattern based on which the workload should be created.
If no pattern is selected the workload will be generated based on generator options.

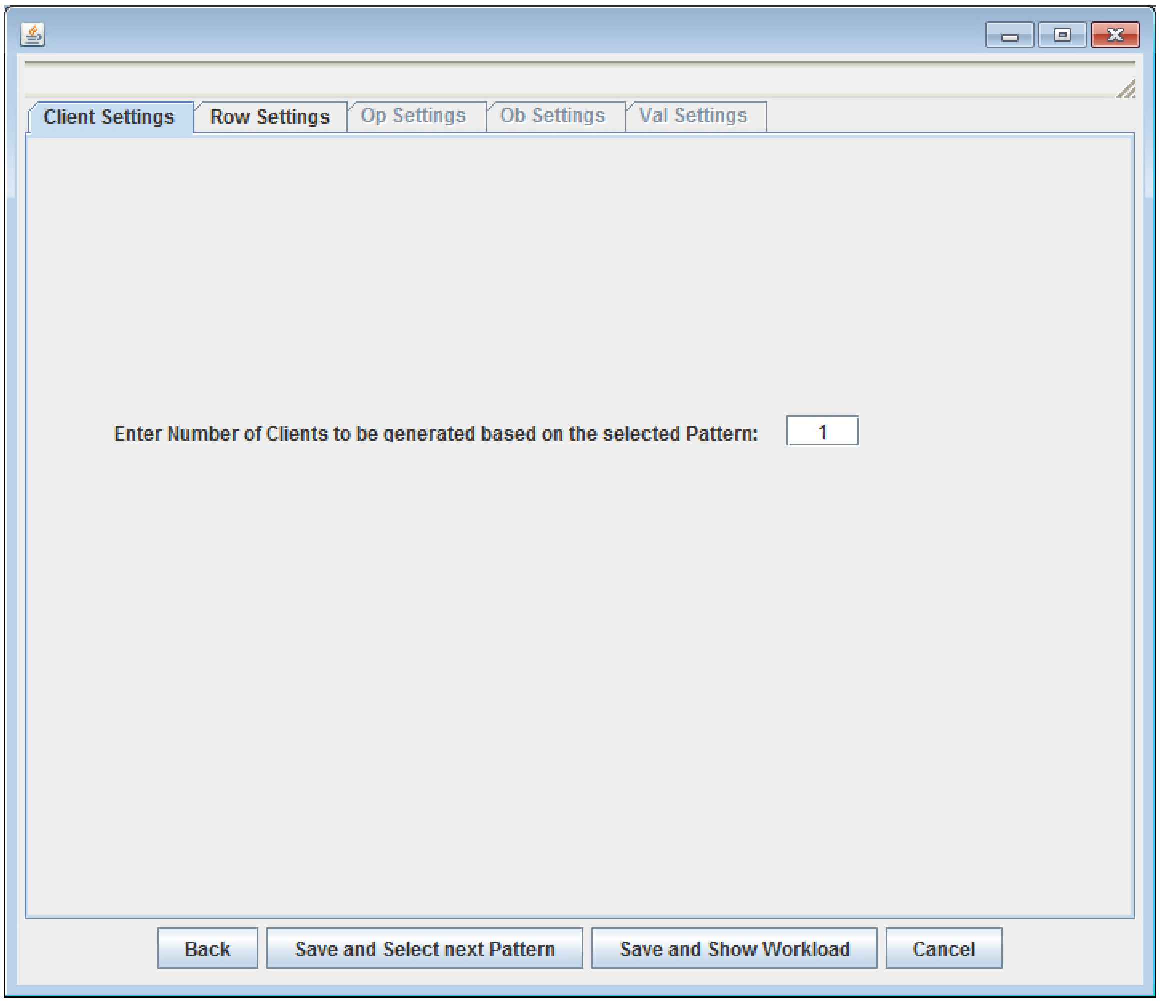
Step 2:
Enter the basic settings for the Advanced Workload Generator.

The workload can be build up in iterations, based on multiple patterns.

( ) Do not use a pattern

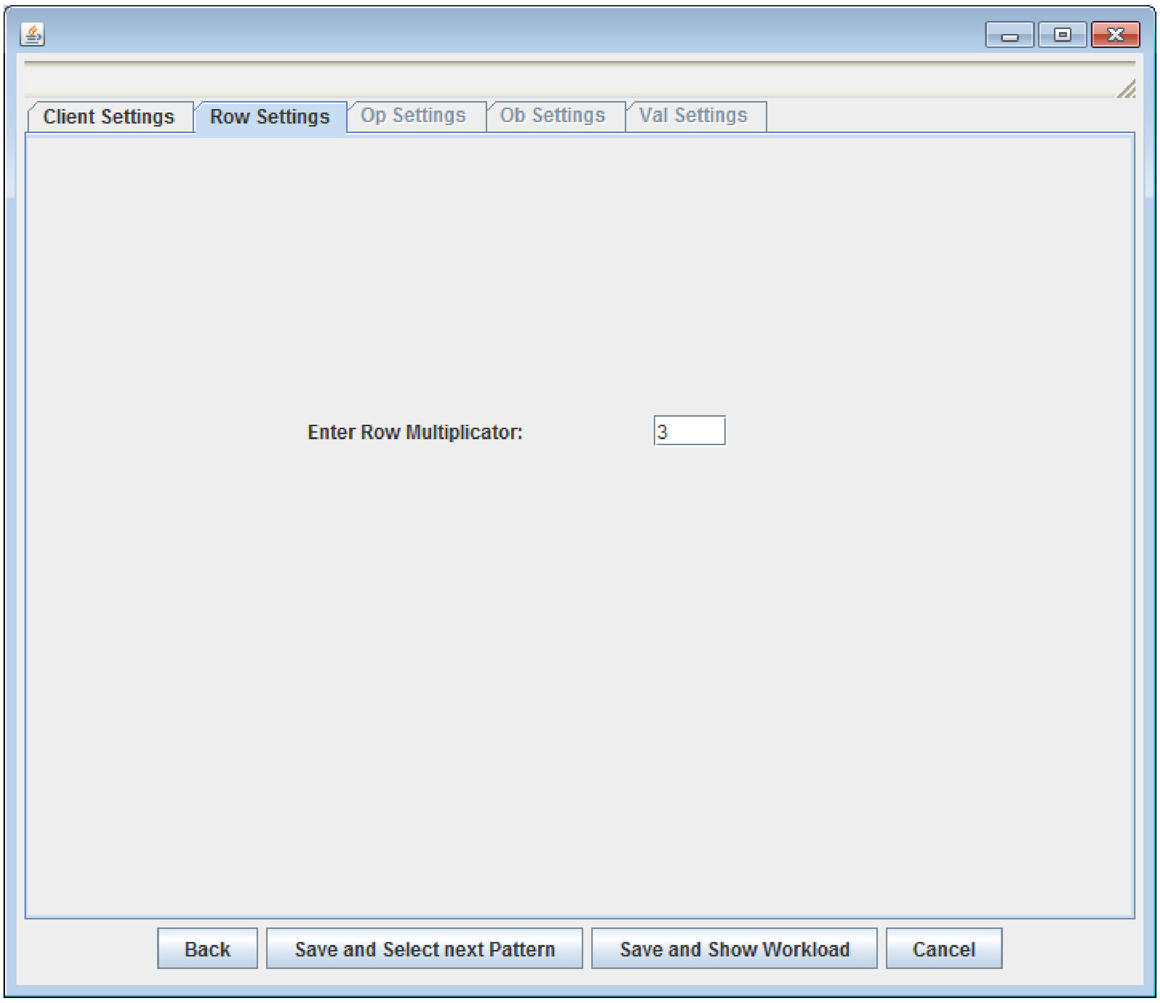
(●) Use selected patt... prämienzahlung.xml

Figure 8.2: Scenario - Pattern Selection



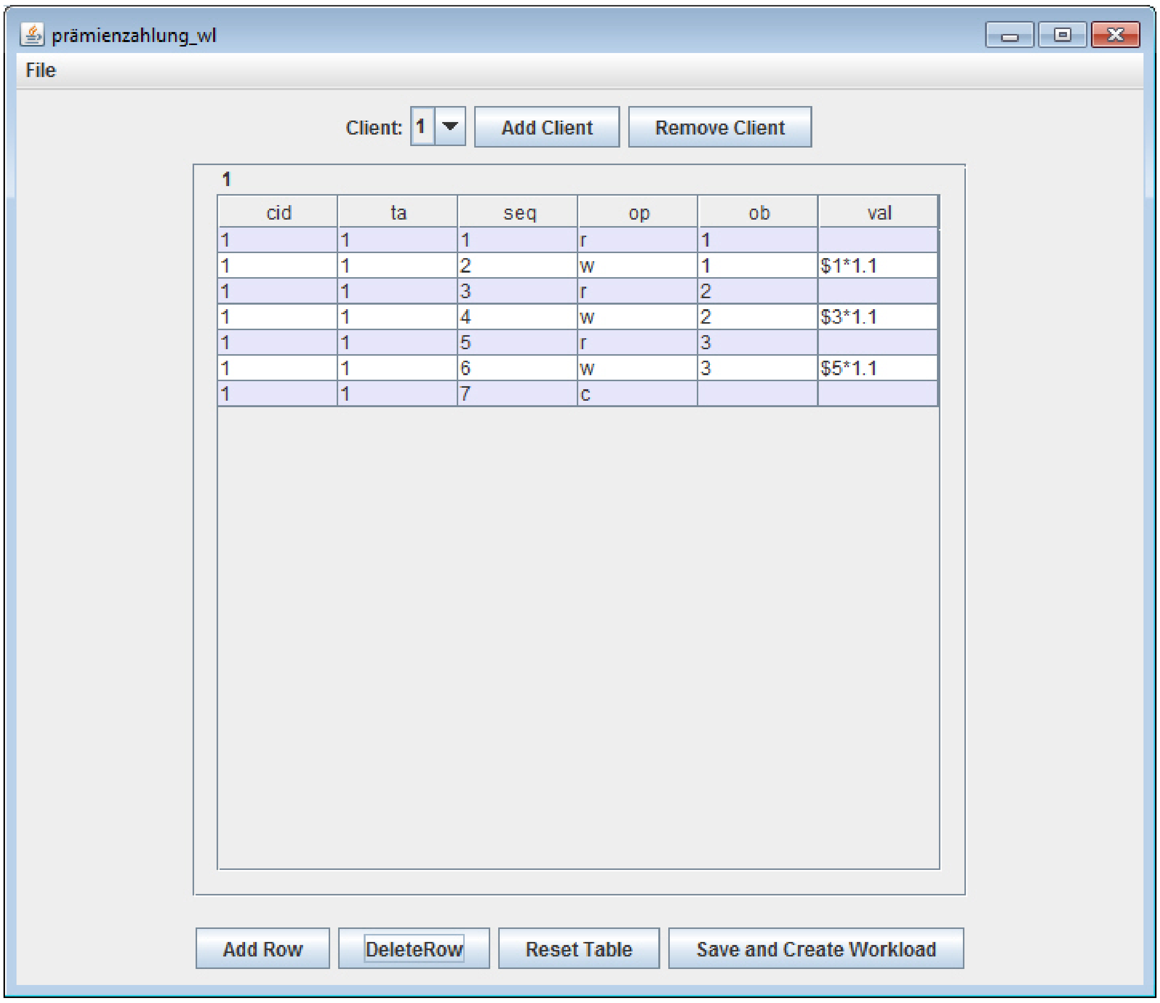
Enter Number of Clients to be generated based on the selected Pattern: 1

Figure 8.3: Scenario - Amount of Clients



Enter Row Multiplicator: 3

Figure 8.4: Scenario - Amount of Employees



cid	ta	seq	op	ob	val
1	1	1	r	1	
1	1	2	w	1	$1*1.1
1	1	3	r	2	
1	1	4	w	2	$3*1.1
1	1	5	r	3	
1	1	6	w	3	$5*1.1
1	1	7	c		

Figure 8.5: Scenario - Result

Bibliography

[1] Christian Tilgner, Boris Glavic, Michael Böhlen, and Carl-Christian Kanne, Smile: Enabling Easy and Fast Development of Domain-Specific Scheduling Protocols: In *BNCOD Posters*, 2011.

[2] Christian Tilgner, Boris Glavic, Michael Böhlen, and Carl-Christian Kanne. Declarative Serializable Snapshot Isolation. In *Fifteenth East-European Conference on Advances in Databases and Information Systems* (ADBIS), September 2011.

[3] http://download.oracle.com/javase/1.4.2/docs/api/java/sql/PreparedStatement.html, 10.1.12

[4] http://www.ifi.uzh.ch/dbtg/research/smile/Studentprojects.html, 10.1.12