CHAPTER 06

APPLICATION OF REAL TIME
CONSTRAINT NOTATION - MOBILE
DATABASE MANAGEMENT SYSTEM

6.1 DESIGNING MOBILE DATA LOGGING
FRAMEWORK USING UBIQUITOUS
COMPUTING ENVIRONMENT

6.1.1 Introduction

A framework to store the incoming calls to a specific or any
general mobile along with some specific features to identify
the service provider about each call has been proposed. Two
steps have been described, in this chapter the first involves
development of Ubiquitous computing environment for all
incoming call sources. Secondly we have developed a mapping
or equivalent scheme of Object Constraint Language OCL to
Database Management System. Since OCL is used over Object
Oriented Language the entire communication framework will
be in Object Oriented Language.

6.1.2 Framework for Ubiquitous Communication

Mobile environments are converging in a shared
communication sphere. Mobile Communication and
Semantic Web [90, 91, 92, 93] based technologies are
integrated for converging towards ubiquitous mobile communication with ontology.

Knowledge Management and Semantic technologies fields provides various method for description, specification and management of information in such a form that machines can read. i.e. if enables acquiring, evolving, reusing, and combining knowledge. Architectural knowledge and ontology support evaluation of mobile communications and service platforms. Interoperability can be achieved integrating Web and Mobile Service. Shared schema based representation of knowledge is used for services and telecommunication systems. Yet, ontology based solutions provides an interoperability for mobile systems has not come into existence. The developments related to ontology till date, have been presented is in, utilized in construction of database, ontology sharing as well as maintenance, ontology reuse and prototypical technologies of web semantics. We confronted various technical, collaborative and social challenges while developing software which revealed that, in order to standardize the efforts and setup the best practices in this field. The work of ontology experts are required to be aligned across the projects of mobile communication shown in Fig 6.1.
Fig. 6.1: Framework for Ubiquitous Communication

Context information is provided at first three layers from various networks and providing support of context - model by ontology method. Formal text model supports the ontology framework making it capable for representing context, sharing of context and semantic interoperability of heterogeneous system.

Our focus is to develop a model supporting interoperability with different platforms.
Fig 6.2: Class representation of Ubiquitous Context Ontology

Ontology context-aware is shown in Fig. 6.2 it is based on computing supporting conversion of high level context to low level context (using different logic reasoning mechanisms available). Herein entities are represented in a structured way depicting the conceptual or physical object.

Relationships among different Class can be developed using this framework. The Dynamic Ontology aspects, as specified in Fig 6.3 depict acquired relationship with specified attributes of the class is expressed.
Fig 6.3: Dynamic Ontology Mapping

For this domain, we have an architecture view of the system having Environment Manager Organizer, Context Manager Organizer and Task Manager Organizer. Position and region of the current user within the network are detected by the Environment Manager. When a user changes its position the Ubiquitous atmosphere or environment enables an automatic or self mode movement of computing process via Task Organizer or Manager. The Communication Organizer or Manager is accountable for detecting the entry of the user end and the user related information is provided by the Task Manager. Task Manager performs the following:

1. Detecting the number of digits of the mobile number.
2. Checks if it lies in the same region then it detects the City, Area and Cell Location.

3. Detects of the first four digits indicate service provider:

- 9353 – RELIANCE COMMUNICATION
- 9253 – TATA INDICOM
- 9918 – VODA PHONE
- 9415 – BSNL

Low level message passing (supported by Java Remote Method Invocation) is used to pick the number. Number_Requester sends messages to Number_Provider on a remote host, and Number_Provider sends the number to Number_Receiver. Simple Object Access Protocol (SOAP), (a light weight protocol which is based on XML and is used for exchanging information in a distributed, decentralized domain) is used for message passing. Corresponding to every number received by the tracer an Apache-SOAP deployment descriptor in XML is generated.

<SOAP> <Phone List_Phone Number/></SOAP>

As a basic Ubiquitous computing supports an artificial intelligent mapping using SOAP descriptors and RMI.

6.1.3 Applying Constraint on Data Base:

Constraint defines as, restrictions imposed on some values of object oriented model [94]. Syntactical constructs can be
used for association-multiplicity to define the constraints. Generally in Database Management System we can have three types of constraints:

1. **Explicit Constraint**: Business rules are represented by this constraint.

2. **Implicit Constraint**: Data models are imposed by some integrity rules which are represented by this constraint.

3. **Inherent Constraint**: Constraints mentioned in a schema are called the inherent constraint.

For applying constraints, invariant rules (expressed in OCL) are used on SQL Table which has all the information about the Caller, location of the caller and his number. Conversion pattern is shown below.

**OCL Invariant:**

**NAME**: OCL INVARIANT CREATION

**Description**: OCL for information gathering of Caller Number detail.

\[ Context \text{ <class name> inv <constraint name> : <OCL expression (self)> is transmitted to : <class name> all Instance \rightarrow \text{for All (<OCL expression (Self)>)} \]

Now applying the invariant in our Ontology Schema

**OCL Expression:**

Context Call inv Number type:
Self.Number type → for all (first four digit = false/true)

Self. Service provider→ Select (C: Company/C.location = within region.cellarea)

The OCL expression includes all collection of operations that are having boolean value, such OCL expression can be mapped to SQL predicate using nested sub queries (if required). Goal of this approach is to enable a procedural SQL Code [3] with full OCL language.

By above approach we focus on how context modeling can be used for developing Unified Ubiquitous Processing or computing environment. As a basic element of Ubiquitous Processing or Computing, the context structure or framework allows an intelligent or logical context of process of entering or input date by collecting generalized knowledge of basic context and extending it for domain specific Ontology in a precedence or in hierarchical manner. The heterogeneous group or network is also considered and detecting number plays a major role. In the second phase one can apply OCL for specifying database integrity constraints. Further one can try to include ‘pre’ and ‘post’ perspectives of executable SQL Code.
6.2 ENHANCEMENT OF USER’S CALL LOGGING FACILITIES USING PUSH DOWN AUTOMATA (PDA) WITH REAL TIME CONSTRAINT NOTATION (RTCN)

6.2.1 Introduction

Herein our goal is to provide a structure which can store or record the incoming calls to a specific or particular mobile service provider with some specific features of logging each call’s service provider. For this, two basic steps have been identified; the first step involves developing Ubiquitous Computing atmosphere or environment. The incoming calls can be from different sources. Secondly, developing push down automata model for call logs database management system. As OCL [95] is used on Object Oriented Language [96] whole communication framework is represented in Object Oriented Language. It is best suitable for database schema in database applications.

6.2.2 An Introduction to Push Down Automata (PDA):

A Push Down Automata (PDA) is finite automata with auxiliary storage devices called stacks. A pushdown automaton (PDA) [97, 98, 99] may be pictured as a finite automaton with the stack or pushdown store onto which
symbols may be ‘pushed’ or from which they may be ‘popped’ as shown in Fig 6.4.

Fig. 6.4: Push Down Automata Model

6.2.2.1 Transitions in Pushdown Automata

A Transitions in Push Down Automata (PDA) is depends on

1. The current state of the machine;
2. The symbol read from the input tape;
3. The top symbol on the stack.

Outcomes of the transition are

1. After reading the input alphabet current state may be transit to new state or on same state (loop condition)
2. On the top of the Stack new Stack Alphabet may be Pushed or current Top of the Stack Alphabet may be popped
3. No change on the Top of the Stack
6.2.2.2 Push Down Automata (PDA) and its Formal Definition

The formal representation of PDA, M is given by seven tuples as $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

1. $Q$ is finite set of states
2. $\Sigma$ is finite set of input alphabets
3. $\Gamma$ is finite set of Stack alphabets
4. $q_0$ is the initial state and $q_0 \in Q$
5. $F$ is the finite set of final states and $F \in Q$
6. The definition of the Transition Function $\delta$ is given as

$$\delta: Q \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow (Q \times \Gamma^*)$$

Transition Function $\delta$ can be defined the moves as two types:

First type of moves PDA reads one input symbol from the input tape. Let $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$ and

$$\delta(q, a, Z) = \{(q_1, \gamma_1), (q_2, \gamma_2), (q_3, \gamma_3), \ldots, (q_m, \gamma_m)\} \quad \text{(1)}$$

where $q_1, q_2, q_3, \ldots, q_m \in Q$ and $\gamma_1, \gamma_2, \gamma_3, \ldots, \gamma_m \in \Gamma^*$

Equation (1) can be defined as: present state of the PDA is at $q$, from input tape it reads input alphabet $a$ and top of the stack alphabet is $Z$. PDA changes its state to any one of $q_1, q_2, q_3, \ldots, q_m$ and replace top of the stack by $\gamma_1, \gamma_2, \gamma_3, \ldots, \gamma_m$ i.e. if the next state is $q_1$, $Z$ will be replaced by $\gamma_1$, if the next state is $q_2$, $Z$ will be replaced by $\gamma_2$, and so on.
Second type of moves in which no input symbol is read from
the input tape i.e. existence of null string (ε) then in this case
Equation (1) can be represented as

\[ δ(q, ε, Z) = \{(q_1, γ_1), (q_2, γ_2), (q_3, γ_3), \ldots, (q_m, γ_m)\} \] .... (2)

6.2.2.3 Instantaneous Description (ID):

PDA configuration at a given instant of time is called
instantaneous description (ID), is defined to be a member of \( Q \times Σ \times Γ^* \): The first component is machine’s state, the second
is the portion of the input string to be processed and third is
stack string i.e. for every \((q, γ) \in δ(p, a, Z)\) and for every \(x \in Σ^*\)
and \(a \in Γ^*\) we define

\[(p, ax, Za) \vdash (q, x, γa)\]

6.2.2.4 Acceptance by PDA through Empty Stack:

Let a string \(w\) is accepted by Empty Stack or Null Stack by \(M\)
iff

\[(q_0, w, Z_0) \vdash (q, ε, ε)\] for some \(q \in Q\)

A Push Down Automata accepted by empty stack is
represented as:

\[M = \{Q, Σ, Γ, δ, q_0, Z_0, Φ\}\]

6.3 OBJECT CONSTRAINT LANGUAGE (OCL) & REAL
TIME CONSTRAINT NOTATION (RTCN) IN USER’S
CALLS LOGGING:
Software models can be developed using OCL and hence is regarded as a modeling language. OCL when used with UML is accepted as a standard as it adds additional features. OMG (Object Management Group) is a standard for object-oriented designing and analysis. Each OCL expression depends on the types used in UML diagram i.e., interfaces, classes etc. OCL expressions therefore definitely involve few UML aspects.
OCL expressions are used to represent a value inside a model. This value may represent either value (say an integer), a collection of values which is an object reference, or a set of references to objects. Boolean value can also be represented by an OCL expression depicting either a message in an interaction diagram or a state chart condition.
A distinguishing OCL feature is that it is founded on predicate logic and mathematical set theory, and its semantics is also formal.
A real time constraint notations are used in object-oriented language that supports real time specifications as required from a real time language, while integrating these specifications within the object-oriented tapestry [100, 101, 102]. Object oriented real time modeling has some significant aspects as follows:
1. The use of inheritance and redefinition of real-time
constraints via hierarchy of inheritance and extension of the
inheritance of the state and behavior of a class to include the
definition of temporal constraints.

2. The reuse of the temporal constraint specifications through
the inheritance mechanism and across class boundaries.
These temporal constraints can be referenced to formulate
new constraints in a consistent manner in classes.

3. Time-abstraction seems like a natural approach to
specifying timing constraints, where the constraint is defined
at the class definition and then associated with the class
implementation. The temporal behavior of an object is made
independent of the object’s actual implementation by relating
the temporal constraint to a labeled code block.

6.4 PROPOSED WORK: PUSH DOWN AUTOMATA
MODEL FOR MOBILE CALL LOGS STORAGE IN
CLUSTERED WAY:

Proposed approach using Push Down Automata has been
carried out in two phases. In the first phase the mobile
call storage has been performed without number
portability service, whereas second phase demonstrates the facility of number portability.

6.4.1 Proposed Model without Mobile Number Portability

Working of Push Down Automata Model for Mobile Call Logs Storage in Cluster way can be represent by the flow chart as shown in Fig. 6.5

Flow Chart for working of Push Down Automata Model for Mobile Incoming Call Logs Storage in Cluster way is divided into following steps:

![Flow Chart](image-url)
**Step1:** Occurrence of Incoming Call on user mobile through a Mobile Service Provider

**Step2:** Call received by the User

**Step3:** Call record updating process by STACK PUSH operation

**Step4:** Check for STACK condition: (FULL / NOT FULL)

**Step5:** Two cases may occur:

IF YES: Call record clustering process start by STACK POP operation

IF NO: GO TO STEP 3

**Step6:** User View: Clustered view of incoming call records of different Mobile Service Providers

Steps 01 to 06 as mentioned above can be modeled with the help of Push Down Automata. We have proposed a model for storage of Mobile Call logs for different Mobile services, which can be clustered for summary describing about the calls received by various types of mobile services.

To achieve this goal first we design the Push Down Automata (PDA) as

\[ M_{MOBILE} = \{ Q_{MOBILE}, \Sigma_{MOBILE}, \Gamma_{MOBILE}, \delta_{MOBILE}, q_0, Z_0, \Phi \} \]

\( Q_{MOBILE} \) : is the set of states for maintaining the call logs from different mobile service provider and it is represented by two states as : \( q_{push} \) and \( q_{pop} \)
\( q_{\text{push}} \): is the state when the incoming calls are received by the user and stack alphabets assigned to different service provider is PUSHED on the stack and at that time stack is not FULL i.e. (STACK ≠ FULL)

\( q_{\text{pop}} \): is the state when the STACK of call log memory is full (STACK = FULL) and user is in the process to get a clustered view of call records for incoming calls from different mobile service providers with the help of POP operation of STACK alphabets allocated to different mobile service provider calls

\( \Sigma_{\text{MOBILE}} \): is finite set of input alphabet for incoming calls for different mobile service provider i.e.

\( \Sigma_{\text{MOBILE}} = \{a, b, c, d\} \) where

- \( a \) represents : BSNL
- \( b \) represents : VODAPHONE
- \( c \) represents : RELIANCE
- \( d \) represents : IDEA

\( \Gamma_{\text{MOBILE}} \): is finite set of STACK alphabets for incoming calls for different mobile service providers i.e.

\( \Gamma_{\text{MOBILE}} = \{Z_{\text{BSNL}}, Z_{\text{VODAPHONE}}, Z_{\text{RELIANCE}}, Z_{\text{IDEA}}, Z_{0}\} \)

For STACK operation (PUSH) when a call received from different mobile service provider is represented as:
\[ Z_{BSNL} : \] stack alphabet is pushed on the STACK when a call is received from BSNL

\[ Z_{VODAPHONE} : \] stack alphabet is pushed on the STACK when a call is received from VODAPHONE

\[ Z_{RELIANCE} : \] stack alphabet is pushed on the STACK when a call is received from RELIANCE

\[ Z_{IDEA} : \] stack alphabet is pushed on the STACK when a call is received from IDEA

\[ q_0 : \] is the initial state when mobile phone is just switched ON and transit to \( q_{\text{push}} \) when the first call is received after the mobile is switched ON

\[ Z_0 : \] is the initial STACK alphabet

\[ \delta_{\text{MOBILE}} : \] is the Transition Function to update the STACK when different types of calls are received from different mobile service providers and is defined as:

\[ \delta_{\text{MOBILE}} : \) Transition Function is defined as

\[ Q_{\text{MOBILE}} \times (\Sigma_{\text{MOBILE}}) \times \Gamma_{\text{MOBILE}} \to (Q_{\text{MOBILE}} \times \Gamma_{\text{MOBILE}}^*) \]

**6.4.1.1 Push Down Automata (PDA) Transitions for Incoming Call Updation:**
Fig. 6.6 represents the incoming call updation by PUSH operation according to call received through a specific Mobile Service Provider.

According to STACK record as mentioned in Fig 6.6, sequences of incoming calls are:
BSNL(a), VODAPHONE(b), BSNL(a), IDEA(d), RELIANCE(c), BSNL(a)

According to above sequence of call received the input string is identified on the input tape is

$$\textbf{abadca} \quad \ldots \ldots \quad (3)$$

PDA Transitions for input string as mentioned in (3) is :

1. $\delta(q_0, a, Z_0) = (q_{\text{PUSH}}, Z_{\text{BSNL}}Z_0)$
2. $\delta(q_{\text{PUSH}}, b, Z_{\text{BSNL}}) = (q_{\text{PUSH}}, Z_{\text{VODAPHONE}}Z_{\text{BSNL}})$
3. $\delta(q_{\text{PUSH}}, a, Z_{\text{VODAPHONE}}) = (q_{\text{PUSH}}, Z_{\text{BSNL}}Z_{\text{VODAPHONE}})$
4. $\delta(q_{\text{PUSH}}, d, Z_{\text{BSNL}}) = (q_{\text{PUSH}}, Z_{\text{IDEA}}Z_{\text{BSNL}})$
5. $\delta(q_{\text{PUSH}}, c, Z_{\text{IDEA}}) = (q_{\text{PUSH}}, Z_{\text{RELIANCE}}Z_{\text{IDEA}})$
6. $\delta(q_{\text{PUSH}}, a, Z_{\text{RELIANCE}}) = (q_{\text{PUSH}}, Z_{\text{BSNL}}Z_{\text{RELIANCE}})$
6.4.1.2 Instantaneous Description (ID) for Incoming Call updation

As per the rules Instantaneous Description (ID) for the input string as mentioned in (3) is:

\[ q_0, \text{abadca}, Z_0 \vdash q_{\text{PUSH}}, \text{badca}, Z_{\text{BSNL}}Z_0 \]
\[ \vdash q_{\text{PUSH}}, \text{adca}, Z_{\text{VODAPHONE}}Z_{\text{BSNL}}Z_0 \]
\[ \vdash q_{\text{PUSH}}, \text{dca}, Z_{\text{BSNL}}Z_{\text{VODAPHONE}}Z_{\text{BSNL}}Z_0 \]
\[ \vdash q_{\text{PUSH}}, \text{ca}, Z_{\text{IDEA}}Z_{\text{BSNL}}Z_{\text{VODAPHONE}}Z_{\text{BSNL}}Z_0 \]
\[ \vdash q_{\text{PUSH}}, \text{a}, Z_{\text{RELIANCE}}Z_{\text{IDEA}}Z_{\text{BSNL}}Z_{\text{VODAPHONE}}Z_{\text{BSNL}}Z_0 \]
\[ \vdash q_{\text{PUSH}}, \varepsilon, Z_{\text{BSNL}}Z_{\text{RELIANCE}}Z_{\text{IDEA}}Z_{\text{BSNL}}Z_{\text{VODAPHONE}}Z_{\text{BSNL}}Z_0 \]

6.4.1.3 Push Down Automata (PDA) Transitions to generate Cluster view of Call Logs (Incoming) when STACK = FULL:

Incoming call log updation, as shown in Figure (3), will continue till STACK ≠ FULL. When the STACK = FULL then transition for Cluster view generation is started according to following steps:

**Step 1:** Suppose that as STACK = FULL then # symbol is generated on the input tape and \(q_{\text{PUSH}}\) is converted to \(q_{\text{POP}}\)

1. \((q_{\text{PUSH}}, \#, Z_{\text{BSNL}}) = (q_{\text{POP}}, Z_{\text{BSNL}})\)
2. \((q_{\text{PUSH}}, \#, Z_{\text{VODAPHONE}}) = (q_{\text{POP}}, Z_{\text{VODAPHONE}})\)
3. \((q_{\text{PUSH}}, \#, Z_{\text{RELIANCE}}) = (q_{\text{POP}}, Z_{\text{RELIANCE}})\)
4. \( (q_{\text{PUSH}}, \#, Z_{\text{IDEA}}) = (q_{\text{POP}}, Z_{\text{IDEA}}) \)

**Step 2:** It is assumed that when \( \text{STACK} = \text{FULL} \) that user will not be able to receive any incoming call till the STACK is not empty. In this case READ head will continue read \# symbol on the input tape and following transitions will occur for the generation of Cluster view of incoming call logs with the help of STACK POP operation as follows:

1. \( (q_{\text{POP}}, \#, Z_{\text{BSNL}}) = (q_{\text{POP}}, \varepsilon) \)

2. \( (q_{\text{POP}}, \#, Z_{\text{VODAPHONE}}) = (q_{\text{POP}}, \varepsilon) \)

3. \( (q_{\text{POP}}, \#, Z_{\text{RELIANCE}}) = (q_{\text{POP}}, \varepsilon) \)

4. \( (q_{\text{POP}}, \#, Z_{\text{IDEA}}) = (q_{\text{POP}}, \varepsilon) \)

According to incoming call sequence as shown in equation (3), all POP operation are carried out as transition 1 to 4 mentioned above and clustered view will be generated as:

![Cluster view of Incoming Calls](image URL)

**Fig. 6.7: Cluster view of Incoming Calls**
Fig. 6.7 shows that the cluster view of input string abadca and can be interpreted as:

No. of BSNL calls : 03 (count of a’s)
No. of VODAPHONE calls : 01 (count of b’s)
No. of RELIANCE calls : 01 (count of c’s)
No. of IDEA calls : 01 (count of d’s)

**Step 3:** When Top of the STACK = $Z_0$ it implies that all the stack alphabets corresponding to different service provider is popped out and Top of the STACK is $Z_0$ then state $q_{POP}$ is transit to $q_{PUSH}$ and mobile set is ready to receive incoming call. Step 3 condition in PDA model can be completed with the help of transition:

1. $(q_{POP}, \#, Z_0) = (q_{PUSH}, Z_0)$

6.4.2 Proposed model with Mobile Number Portability (MNP)

6.4.2.1. Introduction

MNP (Mobile Number Portability) is implemented using various methods across the globe. The International and European standard is for a customer wishing to port his/her number to contact the new network (called Recipient), which then sends the number portability request (NPR) to the current network (called Donor). This is known as 'Recipient-
Led' porting. The UK and India are the only exceptions to implement the Donor-Led system. The customer wishing to port his/her number is required to contact the Donor to obtain a Porting Authorization Code (PAC), which is then expressed to the recipient network to proceed. Once having received the PAC the Recipient continues the port process by contacting to the Donor. Such form of porting is known as 'Donor-Led' and has been criticized by some industry analysts as being inefficient, though prevents MNP scams. It has also been observed that it may act as a customer deterrent as well as allowing the Donor an opportunity of ‘winning-back' the customer. This might lead to distortion of competition, especially in the markets with new entrants that are yet to achieve scalability of operation.

6.4.2.2 Technical details:

A significant technical aspect of MNP is related to the routing of calls or mobile messages (SMS, MMS) to a number once it has been ported. There are various flavors of call routing implementation across the globe but the International and European best practice is via the use of a central database (CDB) of ported numbers. A network operator makes copies of the CDB and queries it to find out to which network to send a
call. This is also known as All Call Query (ACQ) and is highly efficient and scalable. A majority of the established and upcoming MNP systems across the world are based on this ACQ/CDB method of call routing. One of the very few countries to not use ACQ/CDB is the UK, where once a number has been ported; calls to that number are still routed via the donor network. This is also known as 'indirect routing' and is highly inefficient as it is wasteful of transmission and switching capacity. Because of its donor dependent nature, indirect routing also means that if the donor network develops a fault or goes out of business, the customers who have ported numbers out of that network will lose incoming calls to their numbers.

MNP gives the subscriber an option to choose from the various service providers. Mobile phone companies will be under constant pressure to improve their services so that their subscribers not only stay with them but they are also able to attract subscribers from other networks. Competition will, hopefully, ensure that they introduce better service plans with lower tariffs. At the same time, it will benefit new players immensely. Introduction of mobile number portability will facilitate the easy exit of disgruntled users. This also means
operators will have to put more effort to provide better services to retain the customers, better services mean the companies will have to set up more mobile phone towers, upgrade their equipment and provide instant rectification of problems so that the subscribers are always able to get connected, surf the internet and make full use of their mobile telephones.

6.4.2.3 Improvised version of proposed approach for Mobile Number Portability (MNP) issues:

Today’s modern smart phones have application, True Caller associated with CallerID service using the concept of crowd sourcing. By this software we identify the network provider of the incoming calls in the environment of number portability also.

With this new idea the proposed approach has been improvised (Fig. 6.8 & 6.9) with the following explanations.

6.4.2.4 Working of PDA Model for MNP

Step1: Occurrence of Incoming Call on user mobile through a Mobile Service Provider

Step2: Call received by the User

Step3: Identification of Service Provider Based on the True Caller Service.
Step 4: Call record updating process by STACK PUSH operation

Step 5: Check for STACK condition: (FULL / NOT FULL)

Step 6: Two cases may occur:

IF YES: Call record clustering process start by STACK POP operation

IF NO: GO TO STEP 3

Step 7: User View: Clustered view of incoming call records of different Mobile Service Providers

For STACK operation (PUSH) when a call received from different mobile service provider is represented as:

$Z_{BSNL}$: $Z_{BSNL}$ stack alphabet is pushed on the STACK when a call is received from BSNL

$Z_{VODAPHONE}$: $Z_{VODAPHONE}$ stack alphabet is pushed on the STACK when a call is received from VODAPHONE

$Z_{RELIANCE}$: $Z_{RELIANCE}$ stack alphabet is pushed on the STACK when a call is received from RELIANCE

$Z_{IDEA}$: $Z_{IDEA}$ stack alphabet is pushed on the STACK when a call is received from IDEA
6.4.2.5 Improvised version of Call Updation Process:

The transition process of PDA model is same as discussed in the previous approach but the identification of service provider has been replaced with true caller service which was previously a string matching approach. The approach proposed with true caller service is effective with the number portability.
Fig. 6.9 Improvised Incoming Call Record Updation

PDA Transitions for input string as mentioned in Fig. 6.9 is:

1. $\delta(q_0, a, Z_0) = (q_{PUSH}, Z_{BSNL}Z_0)$
2. $\delta(q_{PUSH}, b, Z_{BSNL}) = (q_{PUSH}, Z_{VODAPHONE}Z_{BSNL})$
3. $\delta(q_{PUSH}, a, Z_{VODAPHONE}) = (q_{PUSH}, Z_{BSNL}Z_{VODAPHONE})$
4. $\delta(q_{PUSH}, d, Z_{BSNL}) = (q_{PUSH}, Z_{IDEA}Z_{BSNL})$
5. $\delta(q_{PUSH}, c, Z_{IDEA}) = (q_{PUSH}, Z_{RELIANCE}Z_{IDEA})$
6. $\delta(q_{PUSH}, a, Z_{RELIANCE}) = (q_{PUSH}, Z_{BSNL}Z_{RELIANCE})$

6.4.3 Verification Using Object Constraint Language (OCL):

Keppele defines constraint as, restrictions imposed on some values of object oriented model [103]. Syntactical constructs can be used for association-multiplicity to define the
constraints. Generally in Database Management System we can have three types of constraints:

1. **Explicit Constraint:** Business rules are represented by this constraint.

2. **Implicit Constraint:** Data models are imposed by some integrity rules which are represented by this constraint.

3. **Inherent Constraint:** Constraints mentioned in a schema are called the inherent constraint.

For applying constraints, invariant rules (expressed in OCL) are used on SQL Table which has all the information about the Caller, location of the caller and his number. Conversion pattern is shown below:

**Context : Session**

*Inv: for Incoming Calls*

*Inv: Self.Stack → Empty()*

*Pre: Self.Stack = QMobile*

*Post: Self.Value = TMOBILE*

**Context : Record Call**

*Session.Record = (if QPUSH! = a then ‘ZBSNL’ or b then ‘ZVODAPHONE’ or c then ‘ZRELIANCE’ else d then ‘ZIDEA’)*
Context: Cluster View Generation

Session.Cluster View =

(if STACK = FULL then

q_{PUSH} = q_{POP}

end if)

Session.READ =

(if Input string = abadca then

Count := 03 for BSNL and

Count := 01 for VODAPHONE and

Count := 01 for RELIANCE and

Count := 01 for IDEA

end if)

By the above approach we are able to focus on development of a feature in mobile phone for users to view number of incoming calls in cluster and are also able to know about the number of calls received from different mobile service providers. Same process can also be implemented for outgoing calls which helps mobile service providers to analyse comparative calls rates from different mobile service provider.

6.5 SIMULATION BY EMFtoCSP

Model Driver Engineering (MDE) is popular paradigm for developing software artifacts. These are transformed into models semi-automatically. For ensuring correctness of
software formal methods are playing an important role. But these are not frequently used based because they are very complex to deal with.

Basically, EMFtoCSP [105] is the extended version of previous tool called UML to CSP. It is used for verification of UML class diagrams but due to its generalized nature, EMF to CSP is applicable in all sorts of models developed. This is also used for domain specific modeling languages.

Constraint programming is identified as a programming tool for the development of constraint in logic programming.

Normal Constraint Satisfaction Problem (CSP) is represented by the three tuples:

\[ \text{CSP} = \langle V, D, C \rangle \]

Here \( V \) is the set of variables;

\( D \) is the set of domains;

\( C \) is the set of constraints;

The most common technique for constraint programming is the backtracking.

EMFtoCSP considering ECLtoPSC constraints programming system.

Finally, this is a tool for the development of fully automatic, decidable and expressive verification of the EMF models along with the OCL constraints. This approach is following the
bounded verification approach ensuring proper termination by limiting the search space. This is a model verification tool with a collaborative version.

The functioning of the tool EMFtoCSP has been carried out in 6 steps. In first step, the tool is configured according to the settings of desired experiment. Domain and cardinalities are configured for the tool as discussed in Fig. 6.8.

In second step, the package explorer is explored and right clicks on the project/model which is being validated. After choose the option of the validation.

In the third step, the OCL constraints are configured in the experimental setup of the tool.

![Fig 6.10 Domains and Cardinalities]
Also, the domains and cardinalities of the attributes are configured as discussed in Fig. 6.8. In this, an instance of the model is selected for the validation. This is step 4 of the whole model verification process of the tool.

In step 5, the properties are selected for the model verification. These properties are validated in this tool as given in Fig. 6.9. Strong, weak satisfiability and liveliness are the different properties to be used for the model. The constraint properties are also configured in this tool, if required.

In the sixth step the validated results are configured for storage in the memory.

![Properties selection](image)

**Fig 6.11 Properties Selection**

The proposed approach in this chapter is simulated with this tool.
The results as shown in Fig. 6.10 are found strongly satisfiable for the proposed model. The results are shown in the form of object diagram for the incoming call record processing.

***************

***************