

# Near Real-time Call Detail Record ETL Flows

Munir Cochinwala, Euthimios Panagos

Telcordia Applied Research  
One Telcordia Drive, Piscataway, NJ 08854  
{munir, [thimios](mailto:thimios@research.telcordia.com)}@research.telcordia.com

**Abstract.** Telecommunication companies face significant business challenges as they strive to reduce subscriber churn and increase average revenue per user (ARPU) by offering new services and incorporating new functionality into existing services. The increased number of service offerings and available functionality result in an ever growing volume of call detail records (CDRs). For many services (e.g., pre-paid), CDRs need to be processed and analyzed in near real-time for several reasons, including charging, on-line subscriber access to their accounts, and analytics for predicting subscriber usage and preventing fraudulent activity. In this paper, we describe the challenges associated with near real-time extract, transform, and load (ETL) of CDR data warehouse flows for supporting both the operational and business intelligence needs of telecommunication services, and we present our approach to addressing these challenges.

**Keywords:** Real-time business intelligence, real-time ETL

## 1. Introduction

Today, telecommunication companies are facing significant challenges in reducing subscriber churn and increasing average revenue per user (ARPU) as they try to stay competitive and expand their service offerings. Gathering intelligence via real-time visibility into subscriber usage is a critical element in both business activity monitoring and real-time decision support solutions. Timeliness of the intelligence gained from real-time visibility into subscriber usage is important for revenue gain as well as protection from revenue loss. Consider, for example, the case where the absence of a rule in the logic associated with real-time charging of voice calls results in free international calls for subscribers who dial a specific prefix. Obviously, not being able to detect this abnormal condition in a timely fashion could result in substantial loss of revenue. On the other hand, revenue enhancement opportunities are time sensitive and, often, event based, such as access to tickets for a band or a video of a missed goal in the soccer world cup.

Subscriber usage in telecommunications is recorded in call detail records (CDRs). CDRs represent a wealth of information that can be mined in order to discover patterns related to both calling behavior and service feature usage (e.g., SMS, MMS, etc.). The majority of CDRs are generated by the telecommunication switches

## 2 Munir Cochinwala, Euthimios Panagos

(numbering around 25,000 in the United States) and intelligent network (IN) call processing nodes that handle specific service transactions, such as toll free telephone number translation and mobile number portability. CDR files are fetched from switches and IN call processing nodes on a periodic basis (e.g., every two minutes) and stored in a staging area that offers reliable near-term storage functionality since telephone switches and other network elements have limited storage capacity. Generally, the near-term CDR storage component does not provide any advanced CDR analysis functionality because the emphasis is on billing (including real-time charging) which requires efficient and timely off-loading of the original CDR files from in-line session management network components.

Enabling near real-time CDR mining and analytics requires that the infrastructure responsible for extracting CDRs from the near-term storage component and inserting them into longer term storage (for billing applications as well as data warehouses for decision support and business intelligence) is able to operate in near real time. Such extractions are typically performed by customized extract, transform and load (ETL) flows. These ETL flows must address many challenges not typically found in traditional data warehousing projects. In particular:

1. Handle introduction of new service CDRs in a timely manner and with minimum changes in processing flows;
2. Handle changes to existing CDR attributes with minimal impact in processing flows;
3. Support multiple CDR versions for the same service (this occurs when different service functionality is enabled for sub-sets of subscribers);
4. Accommodate introduction of new near-term CDR storage repositories (required when the number of subscribers exceeds specific thresholds);
5. Support multi-tenant solutions where either multiple services co-exist (e.g., CDMA and GSM versions of a pre-paid offering) or different versions of the same service are being offered to different subscriber groups.

In this paper, we present our work in the area of real-time processing of CDRs in the context of a hosted pre-paid service for mobile virtual network operators (MVNOs). In particular, we present a flexible and extensible CDR extraction, transformation, and load solution that handles dynamic changes to CDR attributes, introduction of new service CDRs, and versioning of CDR in a simple and efficient manner. The solution can be easily generalized to support services in other domains that exhibit the same characteristics with respect to the data that needs to be processed within strict timing constraints. The remainder of this paper is organized as follows. Section 2 provides background information with regard to the MVNO business model. Section 3 discusses the problem we address in this paper. Section 4 outlines our solution. Section 5 covers related work and, finally, Section 6 concludes our paper.

## 2. MVNO Background

The mobile telecommunications market is considered to be a very lucrative one. However, building a mobile network and purchasing spectrum licenses can be very expensive for companies that wish to enter this market. Fortunately, the costs associated with mobile networks and spectrum licenses have “driven” existing Mobile Network Operators (MNOs) to embrace a business model that supports leasing of their networks to third parties that wish to offer (mostly) non-competing mobile services. Such third parties are referred to as Mobile Virtual Network Operators (MVNOs). Examples of existing MVNOs include Kajeet (<http://www.kajeet.com>) and Boost Mobile (<http://www.boostmobile.com>).

Depending on the MVNOs core industries (e.g., existing land-line carrier that wishes to enter the mobile market or an entertainment company that wishes to add a new distribution channel), the required system integration and customer care services may vary. On one hand, MVNOs may already have their own supply of headsets, pricing structure and customer care facilities. On the other hand, MVNOs may just represent different brands and depend on services offered by MNOs and system integrators for billing and customer care, among other services.

In response to the market opportunity created by the MVNO business model, several telecom network operators and telecom service providers have started hosting MVNOs on their wireless intelligent network (WIN) platforms and offering hosted real-time voice and data pre-paid services. CDRs generated by these services must be made available to MVNOs within a short time window after their creation so that the MVNOs can track subscriber usage and potential problems in near real-time. In addition, these CDRs must be analyzed in near real-time for the purpose of detecting revenue leakage, fraud, and undesired service configuration side-effects. Near real-time ETL flows are required for addressing these requirements.

What is important to mention is that MVNOs may require customization to the services they offer to their subscribers quite frequently. Such customization may result in the generation of new service CDRs or changes in the information captured in existing CDRs. While such changes may not create substantial challenges in a single-tenant environment, multi-tenant environments (the most common ones in the MVNO model) must be able to accommodate service CDRs that contain different attributes for different MVNOs.

## 3. Problem Statement

CDRs are generated by real-time call processing components whose top priority is to handle session signaling (e.g. connecting a caller to a callee). Because of this, CDRs are typically buffered in memory for a short period of time and then written to an operating system file in a file system local to the telecom switch or IN call processing node responsible for session signaling. Due to the limited resources available at telecom switches and IN call processing nodes, CDR files are either overwritten or

#### 4 Munir Cochinwala, Euthimios Panagos

deleted when disk space is running out. Since lost CDR files have a negative business impact for both MVNOs and MVNO enablers, an off-board short-term storage approach is commonly used for retrieving CDR files from switches or IN call processing nodes.

The off-board short-term storage approach serves two purposes. Firstly, it is responsible for offering short-term (in the order of 30 to 90 days) persistent storage for CDRs generated by call processing nodes. Secondly, it is responsible for providing access to these CDRs to all operational and business components that require such access, including revenue assurance, fraud detection, billing, and reconciliation components. Such access is typically made available by treating this storage component as the source used for populating a data warehouse via ETL flows.

Typically, a relational database is used for satisfying the above-mentioned requirements of the off-board short-term storage approach. However, the schema of this database is very generic and the number of supported indices is quite small. In many cases, a single table is used for storing all CDRs. Furthermore, this table contains a fixed number of string columns (VARCHAR) for storing CDR attributes. There are two main reasons why this is the case. Firstly, when the volume of generated CDRs is large, e.g., several thousand per second, the database insertion rate must be able to accommodate the CDR generation rate. Secondly, changes in the information captured in CDR attributes or the introduction of new CDRs should have no impact on the database schema and storage configuration (e.g., database files and partitions).

Because of the above properties of the short-term CDR storage solution, CDRs are extracted, transformed, and loaded into proper data warehouses on a periodic basis. In a hosted MVNO solution, CDRs are typically pushed to MVNOs on a periodic basis so that they can be incorporated into MVNO-owned data warehouses and business intelligence applications. Such CDR pushes require extraction of MVNO-specific CDRs and formatting of CDR attributes according to MVNO requirements. This process may be performed as part of the extraction, transformation, and load (ETL) flow or may be initiated after the CDRs are inserted into a data warehouse.

While the extraction and load part of the ETL flow do not present any challenges not already addressed by existing ETL solutions, the transformation component must be able to address, in real-time, the following challenges:

1. Identify the “type” of information present in CDRs in order to determine the transformation rules to be applied;
2. Identify new service CDRs for which no transformation rules are available and handle these CDRs in a way that enables re-processing of these CDRs once the appropriate rules are established;
3. Support multiple CDR versions and accommodate transformation rules that apply to all or a selective sub-set of these versions, including different transformation rules per CDR version.

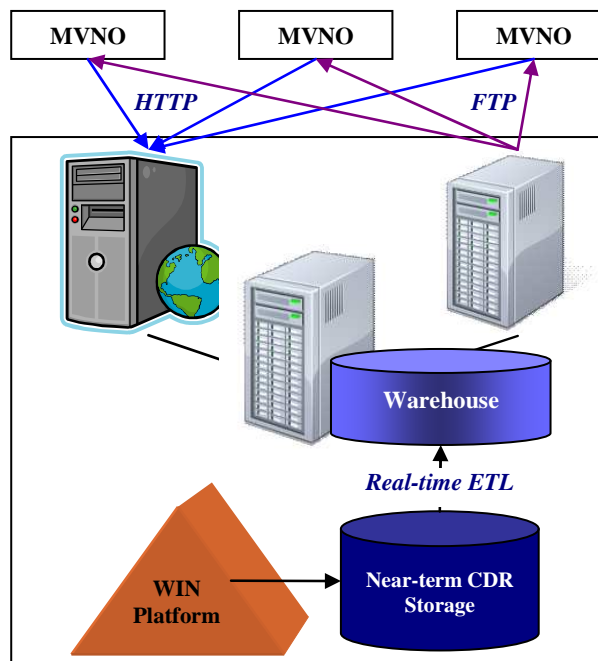
In addition to the above-mentioned challenges, the ETL flows must support the following (partial) list of properties:

1. Track the ETL flow persistently to avoid duplicate CDR processing or missed CDRs;
2. Enforce a run-time deadline for each ETL iteration in order to accommodate periodic ETL invocations without requiring synchronization between such invocations;
3. Identify when a speed-up (e.g., catch-up) in CDR processing is required in order to maintain an upper bound on the time it takes between CDR creation and CDR processing by an ETL flow.

The following section describes our solution to the above challenges. We should note that a version of this solution has been an integral component of the Telcordia hosted MVNO service for the past several years.

#### 4. Our Solution

In this section we describe the architecture of our approach to near real-time ETL flows for CDRs. Figure 1 shows the overall system architecture. The main functionality of the data warehouse component is to store CDRs. A custom ETL application was developed for fetching call detail information from the short-term CDR storage tables and storing this information into the data warehouse on a periodic basis (every 5 minutes).



**Figure 1:** Overall CDR processing system architecture

## 6 Munir Cochinwala, Euthimios Panagos

Once CDRs are inserted into a data warehouse, they are available to MVNOs in one of the following ways. Firstly, a periodic FTP process is responsible for packaging recent MVNO-specific CDRs into appropriately formatted files and then pushing these files to MVNO FTP servers (FTP over a VPN connection or SFTP over the Internet). Secondly, a custom Web application allows access to recent MVNO-specific CDRs to authorized MVNO customer service representatives (CSRs). This application allows MVNO CSRs to access subscriber CDRs that have not yet pushed to the MVNO via the periodic FTP process.

### 4.1 Transformation Rules

The transformation component of the ETL flows is driven by an XML configuration file. This file contains the rules that must be applied to source CDR attributes before being inserted into the data warehouse. These rules are grouped together using the CDR type as the grouping criterion. Typically, the CDR type is identified by either a particular CDR attribute value, a combination of several CDR attribute values, or a join operation between the main table containing CDR rows and auxiliary tables containing meta-data about these CDRs. The following XML fragment shows an example mapping.

```
...
<sample key="PPN_sample" table="voice_sms_samples">
  <map from="$1" to="$1" />
  <map from="$2" to="$2" isAdd="true" />
  <map from="$3" to="$2" />
  <map from="MTSMS" to="$3" />
  <map from="$4" to="$4" isAdd="true" multi="60"/>
  <map from="$5" to="$4" />
  <map from="$6" to="$5" concat="true" />
  <map from="$7" to="$5" />
...
</sample>
...
```

In the above mapping, the input CDR record, whose type is identified by the “PPN\_sample” value, contains several attributes. These attributes are mapped to table columns in table “voice\_sms\_samples”. As part of the mapping logic, some of the input attributes are added together, while others are concatenated. For example, call duration may be recorded as minutes and seconds in the original CDR. When the CDR is inserted into the data warehouse the call duration is recorded in seconds. This is done by instructing the CDR transformation logic to multiply the number of minutes by 60 and then add the seconds to the resulting number. As another example, the input stream may contain two attributes for capturing date and time information. However, we want to combine these two attributes into one before inserting the record into the data warehouse. This can be achieved by a concatenation rule that concatenates the two values.

Note that the `from` attribute of the `<map>` element in the above CDR mapping example can handle both positional input attributes (e.g., `from="$3"`) and constant values to be assigned to the transformed CDR records (e.g., `from="MTSMS"`). (The same is true for the `to` attribute.) This flexibility is quite useful for handling the following two cases without any modifications to the processing flow: 1) a particular CDR type contains an implicit attribute with a constant value; 2) there are two versions of a particular CDR and one of these versions does not contain an attribute found in the other one (here, for the version of the CDR that does not contain the extra attribute, the mapping rules can assign a fixed value to the missing attribute).

We should note that the XML-driven approach used for describing mappings from input attributes to table columns is quite extensible and can accommodate additional mapping logic (not shown above). Furthermore, when the mapping logic is quite complex, custom Java code can be produced for handling such mappings by using Arroyo, a graphical transformation tool (more on that below).

## 4.2 ETL Flows

The following high-level algorithmic steps describe the ETL flow between the short-term CDR storage component and the data warehouse. The following assumptions are made with respect to these steps. Firstly, each CDR in the short-term storage contains an **InsertTimeStamp** attribute. This attribute records when the CDR was inserted into the short-term database. Secondly, ETL progress status is maintained in a persistent manner (e.g., **ETL\_Status** table in the data warehouse). The status attributes include, among others, ETL process start and end timestamps, timestamp of the most recently processed CDR record (**CDRTime**), and the status of the ETL flow (e.g., **Active**, **Done**, **Error**). Thirdly, the ETL process is processing CDRs whose **InsertTimeStamp** attribute is within a specific time window, say 5 minutes, referred to as **ETL\_window**.

1. Select the most recent entry in **ETL\_Status** whose **Status** is set to “**Done**”;
2. If no entry is found, then either this is the very first ETL process run or all previous runs were unsuccessful. In either case, follow the steps below:
  - 2.1. Select the oldest CDR present in the short-term CDR database based on the value of the **InsertTimeStamp** column;
  - 2.2. If no such record is found, no CDRs have been inserted into the short-term CDR database yet and, hence, there is nothing to be done;
  - 2.3. If a record is found, insert a new entry into **ETL\_Status** table and set its **Status** to “**Done**” and **CDRTime** to the CDR **InsertTimeStamp** minus one second;
3. Let **min\_cdr\_itime** be the **CDRTime** present in the **ETL\_Status** record selected in step 1 or created as part of step 2 above;
4. Select the current maximum **InsertTimeStamp** present in the short-term CDR database, referred to as **cur\_cdr\_itime**;
5. Let **max\_cdr\_time** be equal to **min\_cdr\_itime** + **ETL\_window**. If

- max\_cdr\_time**  $\leq$  **cur\_cdr\_itime** - **ETL\_window**, add **ETL\_window** to it (we are behind in processing CDRs and hence, we need to accelerate the CDR processing rate). Else if **max\_cdr\_time**  $\geq$  **cur\_cdr\_itime** then set **max\_cdr\_time** to **cur\_cdr\_itime**;
6. Insert a new record into **ETL\_Status**, with **Status** set to “**Active**” and **CDRTime** set to the **min\_cdr\_itime**;
  7. Fetch from the short-term CDR storage database all CDRs having an **InsertTimeStamp** value that is greater than **min\_cdr\_itime** and less than **max\_cdr\_time**;
  8. Apply transformation rules to fetched CDRs and insert them into the data warehouse. At the same time, remember the maximum **InsertTimeStamp** value present in the processed CDRs;
  9. Update the record created in step 6 and set **Status** to “**Done**” and **CDR\_Time** to the maximum **InsertTimeStamp** computed in step 8.

Some of the important implementation properties of the steps outlined above include: concurrent CDR fetch from the short-term CDR storage database and application of CDR transformation rules (steps 7 and 8), direct or bulk CDR load into the data warehouse depending on volume of fetched CDRs during each ETL execution, handling of CDRs for which no transformation rules exist, and handling of CDR versioning using a CDR type hierarchy that depends on values present in one or more CDR attributes. We will elaborate on the last two properties in the following paragraphs.

In order to accommodate introduction of new service CDRs and different versions of the same service CDRs, the ETL process uses a generic, catch-all, mapping exception rule. This rule is triggered when no other rule exists for handling CDR transformation and data warehouse insertion. In such case, the original CDR is inserted into a generic table that has the same schema as the source CDR table in the short-term CDR storage database. In addition, an alert is generated and sent to the network operations center (NOC) so that the appropriate staff member can examine these CDRs and update the transformation rules appropriately<sup>1</sup>.

The use of a generic table that mirrors the table used in the near-term CDR storage database allows us to run the ETL flow against this table once the appropriate CDR transformation rules are put in place without having to rely upon different post-processing application logic. The only difference between running the normal ETL flow between the short-term CDR storage database and the data warehouse and the ETL flow between the exception CDR table in the data warehouse and the data warehouse is that in the latter case the ETL flow will not attempt to re-insert exception CDRs into the exception table (a configuration property or command line argument can be used for this purpose).

---

<sup>1</sup> While one could partially automate the handling of new service CDRs by trying to infer transformation rules based on CDR “similarity”, we chose to not do so because inaccurate transformation rules may have a negative impact on the logic used by MVNOs for processing the CDR files pushed to them via FTP.

Versioning of CDRs is introduced when the same telecommunications service is offered to two different groups of subscribers (belonging to either the same MVNO or different MVNOs). In such cases, the attributes present in the “same” service CDR may be different, or even the order of these attributes may be different. In order to address such cases, the ETL flows must be able to identify the specific CDR version based on some of the CDR attributes. This is achieved by just extending the transformation rules to include the necessary criteria that examine the values of specific CDR attributes. In most cases, the criteria are quite simple, such as MVNO or subscriber class of service CDR attribute set to a specific value.

As we mentioned in the previous section, ETL flows may have deadlines associated with them. Being able to observe such deadlines is extremely important in order to avoid complications arising from concurrent ETL flows that may result in duplicate CDR processing. The way we address this in our solution (not shown is the steps presented above) is as follows. When an instance of an ETL flow starts, a “watcher” thread is created to monitor overall progress of the flow and keep track of the time remaining before the maximum allowed execution time of the flow. This thread will stop further CDR processing and trigger the execution of step 9 described above when the ETL flow is within a predefined distance (e.g., 1 minute) from the flow deadline.

### 4.3 MVNO CDR Flows

Call detail records in the data warehouse are pushed to MVNO backend systems on a periodic basis. This push can take place as part of the ETL flows covered in the previous section or after the CDRs are inserted into the data warehouse. In either case, custom formatting may be required and, in addition, only a subset of the available database table columns may be required. In order to achieve this, an XML configuration file is used for specifying the following:

1. Attributes to be included in the push;
2. Order of attributes included in the push;
3. Format of file containing the pushed records.

The following XML fragment shows an example of such a mapping for a particular MVNO.

```

...
<mvno name="MVNO" format="FIXED" header="true">
  <map from="$5" to="Timestamp" order="1" size="30" />
  <map from="$1" to="Client ID" order="2" size="8" />
  <map from="$2" to="Charges" order="3" size="8" />
  <map from="$4" to="Length" order="4" size="5" />
</mvno>
...

```

The above configuration file specifies that the CDR files pushed to an MVNO include fixed-length attributes. For each attribute included in the CDR file, the

<map> element specifies the source CDR attribute to be used, the order of this attribute, and its maximum size. The `to` attribute is used for including a header in the CDR file, if required.

Typically, MVNO CDR files are pushed to MVNOs using FTP. Since network conditions, scheduled maintenance, and various other scenarios (e.g., crash of remote FTP server) may interrupt the FTP flow, a persistent process is required in order to ensure that generated MVNO CDR files will reach the MVNO. In our solution, this process uses a database table for storing status information for each CDR file. The status information includes, among other attributes, a unique ID associated with a CDR file, the minimum and maximum **InsertTimeStamp** values associated with the CDRs in the file, and the status of the FTP transfer (not started, in progress, completed, failed).

The minimum and maximum **InsertTimeStamp** values serve two important purposes. Firstly, should we need to recreate a specific CDR file for a given MVNO, we can do so by just knowing the unique ID associated with this file (included in the CDR file name). Secondly, we can easily detect CDR files that may contain overlapping<sup>2</sup> CDRs or CDRs that should have already been processed in the past. This is extremely important since it prevents accidental push of duplicate CDRs to MVNO. Such accidental push may cause MVNOs to display incorrect subscriber usage information on their portals or, even worse, trigger duplicate subscriber balance debits.

In many cases, the MVNO CDR files are created by an ETL flow that is different from the ETL flow used for fetching CDRs from the near-term CDR database. There are several reasons why this is the preferred approach. Firstly, the two ETL flows can run in parallel using their own periodic schedules. Secondly, these two flows may be executed on different physical servers. Thirdly, the ETL flow responsible for processing CDRs in the short-term CDR database is kept “lean” and decoupled from the consumers of the processed CDRs.

#### 4.4 Arroyo

Arroyo [7] [8], our home-grown ETL tool, is a graphical data transformation development environment supporting a wide range of data filtering capabilities. These capabilities are broadly broken into three basic stages: data source selection, data transformation (including matching, selection, classification) and output staging. Specific filters or functions in each of the stages are chained together to produce an overall data flow. Within each stage, processing block customization and user defined transformation functions can be defined and added to the block. At each stage, external sources can be used for refinement and validation.

---

<sup>2</sup> Files containing overlapping CDRs may be generated when multiple instances of the process handling MVNO CDR extraction and formatting may be active at the same time due to manual execution of one of them.

The Java-based tool is able to process complex data analysis flows across a wide variety of data sources. The tool is readily extensible, where new pre-processing or matching functions can be encapsulated into Java classes and dynamically added to the selection palette. Custom components can be written as Java plug-ins and used as first class components.

We have developed a wide variety of customized transformations which can be configured for specific data transformations. Arroyo has been used in many data cleaning and transformation engagements and most transformations can be accomplished by configuration. Additional components can be easily added by using a pre-defined class where the specifics of the transformation need only be written and placed in a directory. This new component is subsequently incorporated into the overall transformation filters wither via an explicit refresh or when Arroyo is re-started.

The data flow and the rules are represented in persistent XML with support for queries about the flow as well as the rules. Each of the stages provides common interfaces and introspection for dynamic loading. This allows for maximum flexibility and extensibility with late binding.

## 5. Related Work

Progress in ETL and associated tools have occurred mostly in the commercial arena. Ad-hoc and self-built ETL tools were prevalent in the 80's and the 90's. These tools were scripts using metadata and applied mostly to databases. Pentaho Data [1] Integration (previously kettle), Talend [2] and others are examples of open source ETL tools.

Over the last decade, commercial and open source ETL tools have become more prevalent. In the commercial space, independent ETL vendors have been incorporated into larger companies. For example, Microsoft [3], Oracle [4] and IBM [5] have ETL offerings, with Microsoft and Oracle offering their ETL engine 'free of charge' with their database product.

ETL tools have started to migrate into Enterprise Application Integration systems that now cover much more than just the extraction, transformation, and loading of data. Data transformation and integration are becoming critical in business intelligence projects often requiring a separated storage area (sometimes call staging or work area) for intermediate results, similar to our work. Ralph Kimball [6] states that ETL design and development may amount for up to 70% of the IT side of a BI project which makes ETL tools a crucial component in the BI architecture.

According to Gartner, "the stand-alone data integration technology markets — such as extraction, transformation and loading (ETL), replication and federation, and enterprise information integration — will rapidly implode into a single market for multi-mode, multipurpose data integration platforms." Indeed if one looks at the top vendors in the market, it is clear that this is happening or has happened. Informatica PowerCenter has added a real-time module to their software, allowing Informatica to

brand PowerCenter as an EAI tool; while IBM has added DataStage, acquired from Ascential, currently under the InfoSphere family.

Our approach required real-time ETL as well as the ability to rapidly configure and change data transformations within the ETL framework. Thus, our solution also required software modularity. Arroyo provided both near real-time performance and the Java framework plus underlying XML representation provided the desired functionality.

## 6. Conclusions

In this paper, we have presented our approach to near real-time processing of call details records in the context of telecommunications services. Our approach is based on configurable transformation rules captured in XML configuration files. We have implemented the approach described in this paper in the Telcordia MVNO hosted pre-paid service and have been able to handle introduction of new CDRs, versioning of CDRs for different MVNOs, and different MVNO formatting requests with only changes in the configuration files that drive the ETL flows. Our implementation handles more than 6,000 CDR extraction, transformation, and load operations per second, and it supports more than 20,000 MVNO CDR transformations per second.

Our solution can be easily generalized to support services in other domains that exhibit the same characteristics with respect to the data that needs to be processed within strict timing constraints. Furthermore, both the source of the records to be transformed and the destination of the transformed records do not have to be a relational database. In fact, we have applied our solution to processing several gigabytes of mobile content download transactions stored in operating system files on a daily basis.

## References

1. Pentaho Data Integration, <http://kettle.pentaho.org>.
2. Talend ETL, <http://www.talend.com>.
3. Microsoft SQL Server Integration Services, <http://msdn.microsoft.com/en-us/library/ms169917.aspx>.
4. Oracle Warehouse Builder (OWB), <http://www.oracle.com/technology/products/warehouse/index.html>.
5. IBM InfoSphere DataStage, <http://www.ibm.com/software/data/infosphere/datastage/features.html>
6. Kimball, R. "Data Warehouse Training," <http://www.ralphkimball.com/html/articlesbydate/articles2007.html>
7. Caruso, F., Cochinwala, M., Ganapathy, U., Lalk, G., Missier, P.: Demonstration of Telcordia's Database Reconciliation and Data Quality Analysis Tool. Proceedings of 26<sup>th</sup> International Conference on Very Large Data Bases (VLDB 2000), September 10-14, 2000, Cairo, Egypt.

8. Cochinwala, M., Kurien, V., Lalk, G., Shasha, D.: Efficient Data Reconciliation. Information Sciences, 137(1-4): 1-15 (2001).