StreamLoader: An Event-Driven ETL System for the On-line Processing of Heterogeneous Sensor Data

M. Mesiti, L. Ferrari, S. Valtolina,
G. Licari, G. Galliani
Dip. di Informatica, Università di Milano, Italy
{mesiti,lferrari,valtolina}@di.unimi.it

M. S. Dao, K. Zettsu
Universal Communication Research Institute
NICT, Kyoto, Japan
{dao.minhson,zettsu}@nict.go.jp

ABSTRACT

ETL (Extraction-Transform-Load) tools, traditionally developed to operate offline on historical data for feeding Data-warehouses, need to be enhanced to deal with big and fresh data and be executed at network level during data streams acquisition. In this paper, we present StreamLoader, a Web application for the specification of conceptual ETL dataflows on heterogeneous sensor data that leverages the peculiarities of network configuration, data stream management, and specification and deployment of ETL operations in a programmable network. It can be used for feeding traditional/ real-time data-warehouses or visual analytic tools.

1. MOTIVATION

Nowadays we are witnesses of the proliferations of different sensor devises able to produce heterogeneous types of data that can be profitable used for detecting, handling and advising people of the verification of events such as natural disasters (like flooding, storming, extreme temperatures etc.), traffic congestions (due to accidents, strikes, football matches), and social web interactions. Beside the physical sensors, able to detect data about physical phenomena (like, temperature, humidity, wind, rain, pressure, level of see water), there is a proliferation of social sensors able to collect data from people (like, twitter data, traffic information, train or flight schedule). These data are characterized both from the temporal, spatial and thematic dimensions that can be exploited from for the identification of meaningful events in a given context.

Several challenges should be faced for handling sensors and their data especially in emergency situations. First, sensors (both physical and social) are located in different networks and made available by different institutes, agencies and NPOs. In this context, network configuration, sensor detection and discovery are difficult issues to be solved. Moreover, data produced by sensors are heterogeneous in structures (different types), in spatial and/or temporal granularities (e.g. temperature in a room versus temperatures in a geographical area), in thematics (data about traffic jams vs data about pollutions). Therefore, there is the need of ETL (Extract-Transform-Load) operations that can be applied on data streams for their reconciliation. These operations should be applied during data acquisition and bound with reactive capabilities in order to properly identify the relevant streams when abnormal events occur and undertake the proper actions. Finally, the specification and actuation of the ETL operations should be efficiently performed on-line and on fresh and timely data in order to properly handling big real-time data streams. All these technical requirements should be addressed in graphical, user-friendly environments supporting the user in the design and execution of the operations.

Many systems have been proposed for configuring programmable networks ([4, 2, 9]), for data stream management (e.g. Niagara [14], TelegraphCQ [5], Borealis [1]), for the specification and actuation of ETL operations (e.g. [16, 10, 13]) and dataflow (e.g. Talend www.talend.com, Pentaho-kettle www.pentaho.com, CoverETL www.cloveretl.com), and for complex event processing (e.g. Apache S4 [15], Storm [12], StreamBase www.tibco.com). However, all of them are quite complex to use, seldom provide web GUIs for designing and monitoring dataflows and are not integrated in a single tool. This limits their use in the management of emergency situations.

In this paper we propose StreamLoader, a Web application for the specification of conceptual ETL dataflows on heterogeneous sensors to be applied during data stream acquisition on a programmable network. It exploits different technologies (AngularJS, Cytoscape, SparkJava) for proving the graphical environment. StreamLoader is equipped with an interactive environment that supports the user in charge of handling events to discover the sensors useful in a given situation, specify the adequate dataflow for extracting, filtering, integrating, (eventually) storing, and analyze the data coming from the identified sensors, optimize the schedule for the execution of the dataflow and visualize the results. By exploiting samples produced by the involved sensors, the user can easily debug the developed dataflow. Once the dataflow is consistent (i.e. it can be soundly activated at network level), the translation is automatically invoked. Then, the underlying network is configured and the processes associated with the ETL operators executed. At this point, logs of the execution of the dataflow components are directly shown in the interactive environment to provide statistics on the dataflow execution.

In the remainder, Section 2 presents the requirements we

10.5441/002/edbt.2016.65

started from. The main characteristics of the system and of the interactive environment are discussed in Section 3. Section 4 presents the features of the system we wish to demonstrate and discusses the system significance.

2. StreamLoader REQUIREMENTS

Starting from the idea to develop a tool that is easy to use for people without a computer science background, the following requirements have been posed at the basis of the development of our tool.

Dynamic (and automatic) configuration of ETL dataflow on events. Starting from several data streams, the user interface should offer the possibility to identify relevant sources of information depending on the verification of events. For example, suppose we have several sensors for detecting the humidity and temperature of a given area; apparent temperature represents the temperature that is perceived by humans and depends on both temperature and humidity. The computation and acquisition of the apparent temperature in a given area can be triggered when the temperature is greater than 24° C. Events can be used both for triggering or stopping the acquisition and elaboration of streams. Moreover, ETL dataflows should be generated on the fly depending on the needs, immediately actuated and its execution monitored in the same tool. The dataflow should become "live" and give execution feedbacks to the user.

ETL operations for integrating heterogeneous streams. The heterogeneity of the data flows requires the application of common operations developed in the context of data integration and data fusion in order to identify different representations of the same real world entities. For this purpose, the system should be equipped with transformation operations: (1) for changing the unit of measure (e.g. from yards to meters) or geographical coordinates (from one standard to another one); (2) for introducing virtual properties relying on the values assumed by other attributes (e.g. the apparent temperature discussed above); (3) for checking that data conform to given validation rules (e.g. dates conforming to given patterns). Moreover, operation for filtering data relying on different conditions should be included and for culling data belonging to a temporal interval or a geographical area according to a reducing factor. Finally, operations for aggregating and joining streams should be included for combining information coming from different streams.

Discovery of sensor data sources. The large amount of sensors with different levels of availability that can be monitored by the developed system imposes the adoption of different solutions for their discovery and management. First, sensors should be handled by means of a publish-subscribe system in order to handle the dynamicity with which they can join and leave the network. Then, sources of dataflows should be specified by means of the sensor and location characteristics. Finally, sensors can be organized according to different criteria (temporal/spatial, type/location) in order to facilitate the specification of dataflows.

Isolation of data traffic based on the ETL dataflow. Hard-coded configurations of network architectures and paths where data traffics are routed are not an easy task and prevent the possibility to adapt to new user requirements. Approaches based on declarative networking [4, 2, 9] have been recently proposed for the application of database query language and processing techniques to the domain of networking. In [8] an extension of the declarative networking [4] approach has been proposed that consists of two components: declarative service networking (DSN) and network control protocol stacks (SCN). DSN provides a method to model and describe a high-level network of information services for an application, which includes service discovery, service monitoring, execution control, and service message exchanges. SCN aims at capturing application requirements and requesting appropriate configuration to the network platform more directly and effectively. The network control protocol stack interprets the DSN description and dynamically coordinates the network configurations, such as data flows, segmentations, and QoS parameters. The dataflow graphically described by the user should be then translated in DSN/SCN language to be actuated in the network and monitored.

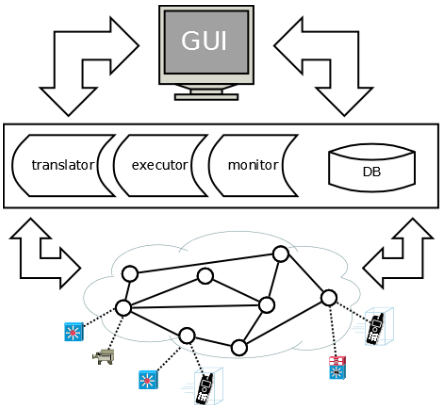


Figure 1: StreamLoader Architecture

3. StreamLoader OVERVIEW

Architecture. Figure 1 reports the architecture of the StreamLoader system. At the bottom there is a network. Each node of the network is in charge of managing a bunch of sensors and can execute the proposed ETL stream processing operations. Sensors are handled through a distributed publish-subscribe system [3]. Each time a sensor is published, its type, schema, and frequency of data generation are made available to subscribers.

Our Web environment is made available for the design and monitoring of dataflows. When a conceptual dataflow is realized, the `translator` module is in charge to translate it in DSN/SCN and execute it at network level. Processes are generated for each operation of the dataflow and executed on a network. The `executor` module coordinates their execution. For the execution, the sources are bound to specific sensors handled by the network nodes, and operations located on the machines that, depending on workload, apply the logic specified in the conceptual dataflow.

Logs of the activities are then collected by the `monitor` module and made available to the Web Interface to show statistics on the dataflow execution. Specifically, we are able to report on the Web Interface the number of tuples that each operation handle per second, the node that suffers because of high workload, which node is in charge of executing an operation and when the assignment changes.

Operation	Symbol	Meaning
Aggregation	$@^{t,\{\mathtt{a}_1,\ldots,\mathtt{a}_n\}}_{op}(s)$	Every t time intervals, aggregate s on the attributes $\{\mathtt{a}_1,\ldots,\mathtt{a}_n\}$ and apply the aggregation function $op \in \{\mathtt{COUNT},\mathtt{AVG},\mathtt{SUM},\mathtt{MIN},\mathtt{MAX}\}$
Cull Time	$\gamma_r(s,\langle t_1,t_2\rangle))$	Culling the tuples in the temporal interval $[t_1,t_2]$ by a reducing rate r
Cull Space	$\gamma_r(s,\langle coord_1,coord_2\rangle))$	Culling the tuples that fall in the area delimited by $coord_1, coord_2$ by a reducing rate r
Filter	$\sigma(s,cond)$	Filter out tuples in s that do not adhere to the condition $cond$
Join	$s_1 \bowtie^t_{pred} s_2$	Every t time intervals, s_1 and s_2 are joined according to the join predicate $cond$
Transform	$\diamond_{trans}s$	The transformation function $trans$ is applied on the tuples in s
Trigger On	$\oplus^{\mathtt{ON},t}(s,\{s_1,\ldots s_n\},cond)$	Every t time intervals the condition $cond$ is checked on the tuples collected from s. If the condition is verified, the streams of the sensors $\{s_1,\ldots s_n\}$ are activated
Trigger Off	$\oplus^{\mathtt{OFF},t}(s,\{s_1,\ldots s_n\},cond)$	Every t time intervals the condition $cond$ is checked on the tuples collected from s. If the condition is verified, the streams of the sensors $\{s_1,\ldots s_n\}$ are de-activated
Virtual property	$\uplus_s\langle p,spec\rangle$	A new attribute p is added to the schema of s according to the specification $spec$

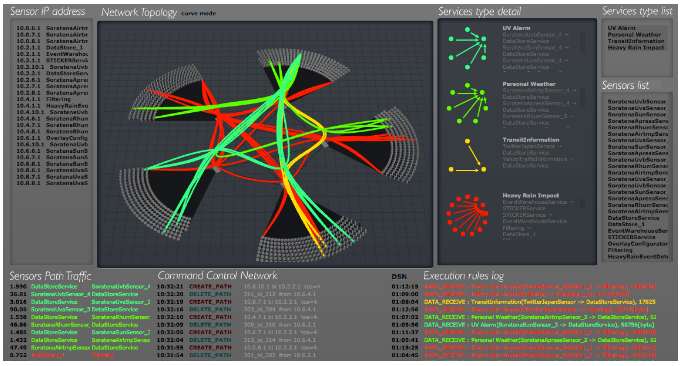
Table 1: Stream Processing Operations

Stream Processing Operations. Sensors produce streams of tuples according to the multigranular space, time and thematic (STT) data model [7]. Relying on the concepts of temporal and spatial granularities, we exploit the concept of *event*, that is a value associated with a spatial object at a given time according to given thematics. Therefore, an event is a value represented at a given spatio-temporal granularity for which thematic information is added. Granularities are used for identifying correlations among data produced by different sensors and for imposing consistency constraints in the composition of sensor data produced by heterogeneous devices. We remark that whenever a sensor is not able to produce the spatio-temporal information of the produced data, this information is added by the Publish-Subscribe system that we adopt in our architecture.

Several operations have been developed for processing and combining the streams produced by the sensors in accordance with the requirements previously discussed at network level. Table 1 reports the principal operations concerning the application of filters, transformation, aggregation and composition, and event detection. Among the operations we point out those that are non-blocking (filter, cull-time/space, transform, virtual property) from those that are blocking (aggregation, trigger, join). The former are directly applied on each tuple when they are processed, whereas the others require the maintenance of a cache of tuples that are processed every t time intervals (e.g. 1 second, 2 minutes).

Visual Interactive Environment. By using a visual interface in Figure 2, users can drag-and-drop data-sources and apply the proposed operations on streams. In a window placed at the bottom of the canvas used for designing the data-flow, the user can see the schema of data that are processed by the operation, specify the conditions of each operation and visualize a data sample coming from each source. The user interface provides different checks in order to draw only dataflows that can be soundly translated in the DSN/SCN specification. We remark that data schema are not fixed but depend on the sensors. Finally, at the use phase, the dataflow developed at design time will be annotated with information coming from the SCN about the execution of the dataflow. In this way, the dataflow becomes "live" and the domain expert can monitor its execution.

Scenario. There are different sensors in the area of Osaka that produce data about the temperatures and levels of rains monitored in the current year. Moreover, tweets and traffic information from the same area in the current year can be acquired. Suppose, that there is interest in acquiring the data about torrential rain, tweets and traffic only when the temperature identified in the last hour is above 25° C. The dataflow reported in the canvas of Figure 2 realizes the proposed behavior. The acquired data can be stored in a data-warehouse or sent to a visualization tool in order to perform further analysis. Moreover, Figure 3 shows the flows of data that are monitored for this and other dataflows that are under control.



```
10.0.6.1   SoratenaAirtm
10.0.7.1   SoratenaAirtm
10.0.8.1   SoratenaAirtm
10.2.1.1   DataStore_1
10.2.1.1   EventWarehou
10.2.1.1   STICKERServi
10.2.10.1  SoratenaUvb
10.2.2.1   DataStoreServ
10.2.6.1   SoratenaApre
10.2.7.1   SoratenaApre
10.2.8.1   SoratenaApre
10.4.1.1   Filtering
10.4.1.1   HeavyRainEve
10.4.10.1  SoratenaUvb
10.4.6.1   SoratenaRhun
10.4.7.1   SoratenaRhun
10.4.8.1   SoratenaRhun
10.6.1.1   OverlayConfig
10.6.10.1  SoratenaUvb
10.6.6.1   SoratenaSunS
10.6.7.1   SoratenaSunS
10.6.8.1   SoratenaSunS
10.8.6.1   SoratenaUvaS
10.8.7.1   SoratenaUvaS
10.8.8.1   SoratenaUvaS
```

```
UV Alarm
SoratenaUvbSensor_4 →
DataStoreService
SoratenaSunSensor_4 →
DataStoreService
SoratenaUvaSensor_1 →
DataStoreService

Personal Weather
SoratenaAirtmpSensor_4 →
DataStoreService
SoratenaApressSensor_4 →
DataStoreService
SoratenaRhumSensor_5 →
DataStoreService

TransitInformation
TwitterJapanSensor →
DataStoreService
YahooTrafficInformation →
DataStoreService

Heavy Rain Impact
EventWarehouseService →
STICKERService
STICKERService →
EventWarehouseService
Filtering →
DataStore_1
```

```
SoratenaUvbSensor_
SoratenaUvaSensor_
SoratenaSunSensor_
SoratenaApressSens
SoratenaRhumSens
SoratenaAirtmpSens
SoratenaUvaSensor_
SoratenaSunSensor_
SoratenaApressSens
SoratenaUvbSensor_
SoratenaRhumSens
SoratenaAirtmpSens
SoratenaUvaSensor_
SoratenaUvbSensor_
SoratenaSunSensor_
SoratenaApressSens
SoratenaRhumSens
SoratenaAirtmpSens
DataStoreService
DataStore_1
EventWarehouseSer
STICKERService
OverlayConfigurator
Filtering
HeavyRainEventDet
```

```
1.590 DataStoreService      SoratenaUvbSensor_4   10:32:21 CREATE_PATH 10.6.10.1 bi 10.2.2.1 tos=4.
36.01 SoratenaUvbSensor_4   DataStoreService      10:32:20 DELETE_PATH 311_bi_312 from 10.6.4.1
3.016 DataStoreService      SoratenaUvaSensor_3   10:32:19 CREATE_PATH 10.8.7.1 bi 10.2.2.1 tos=4.
90.05 SoratenaUvaSensor_3   DataStoreService      10:32:16 DELETE_PATH 303_bi_304 from 10.6.4.1
1.538 DataStoreService      SoratenaRhumSensor_   10:32:10 CREATE_PATH 10.4.7.1 bi 10.2.2.1 tos=4.
46.86 SoratenaRhumSensor_   DataStoreService      10:32:09 DELETE_PATH 309_bi_310 from 10.6.2.1
1.485 DataStoreService      SoratenaSunSensor_3   10:32:05 CREATE_PATH 10.6.7.1 bi 10.2.2.1 tos=4.
1.432 DataStoreService      SoratenaAirtmpSenso   10:32:04 DELETE_PATH 313_bi_314 from 10.6.4.1
47.49 SoratenaAirtmpSenso   DataStoreService      10:31:55 CREATE_PATH 10.0.6.1 bi 10.2.2.1 tos=4.
0.752 DataStore_1           Filtering             10:31:54 DELETE_PATH 301_bi_302 from 10.6.2.1
```

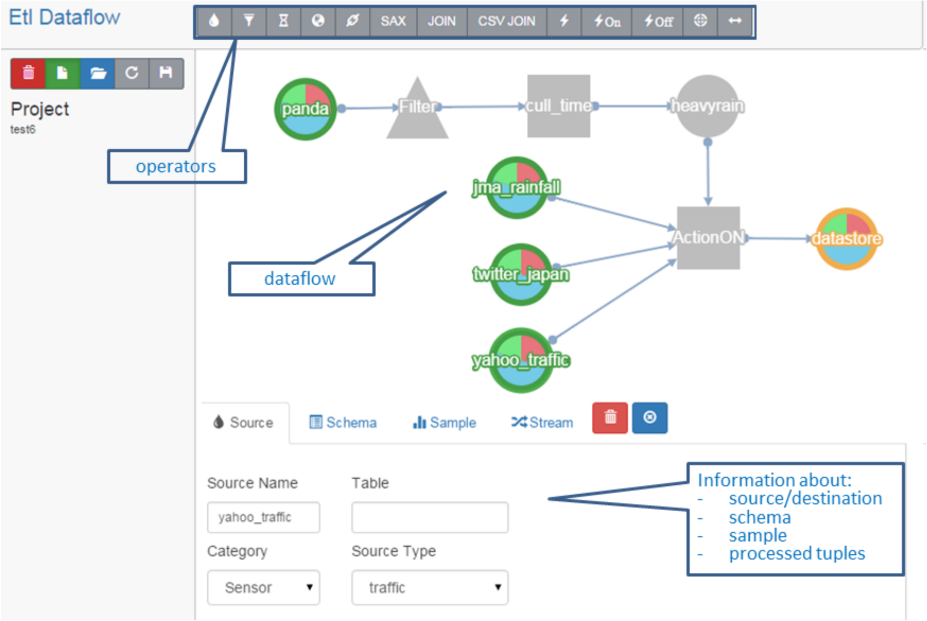
```
01:12:15 DATA_RECEIVE : Heavy Rain Impact(PandaSensor_60x60x1_0 -> Filtering), 2189014[
01:09:00 DATA_RECEIVE : Heavy Rain Impact(Filtering -> DataStore_1), 12737[byte]
01:08:04 DATA_RECEIVE : TransitInformation(TwitterJapanSensor -> DataStoreService), 17825
01:12:56 DATA_RECEIVE : Heavy Rain Impact(TwitterJapanSensor -> Filtering), 10721[byte]
01:07:02 DATA_RECEIVE : Personal Weather(SoratenaAirtmpSensor_2 -> DataStoreService), 62
01:05:56 DATA_RECEIVE : UV Alarm(SoratenaSunSensor_3 -> DataStoreService), 58756[byte]
01:11:37 DATA_RECEIVE : Heavy Rain Impact(PandaSensor_60x60x1_2 -> Filtering), 2188470[
01:05:41 DATA_RECEIVE : Personal Weather(SoratenaApressSensor_2 -> DataStoreService), 62
01:15:25 DATA_RECEIVE : Heavy Rain Impact(PandaSensor_60x60x1_3 -> Filtering), 2188441[
01:04:45 DATA_RECEIVE : Heavy Rain Impact(PandaSensor_60x60x1_1 -> Filtering), 2190319[
```

Figure 3: Monitoring the execution of the dataflow

4. DEMO WALKTROUGH

In the demonstration of StreamLoader we will considering a network (established at NICT in Japan) in which different physical and social sensors related to the described scenario are connected and produce continuous data streams to be processed. By using this setup, we will show the capabilities of our tool in an interactive demo for the automatic configuration (and re-configuration) of a programmable network by means of our visual tool for *more efficient and interactive* analysis when transmitting data from sensors (sources) to the Event Data Warehouse (destination) [6]. The demo will consist of the following parts:

P1 By exploiting the Web interface of our tool, users can create their own dataflows. Specifically, they will be able to identify the different sensors that are currently available in the network and select those on which they wish to specify ETL operations. Moreover, they will be able to apply different processing operations on such sources and check, step-by-step, their results on samples made available from the source.



Source Name: yahoo_traffic

Table:

Category: Sensor

Source Type: traffic

operators

dataflow

Information about:
- source/destination
- schema
- sample
- processed tuples

Figure 2: Main screen of the Web Application

P2 Once the dataflow is consistent, we will show its translation in the DSN/SCN language and deployment at network level. Then, we will monitor its execution by means of the tools presented in this paper. Finally, we will show how the data processed by means of the dataflow can be stored in the Event Data Warehouse [6] or visualized in the Sticker visualization tool [11]. Both tools have been developed by NICT.

P3 In the last part of the demo, we will show how it is easy to plug-and-play new sensors to the network and make them directly available to StreamLoader. We will also show how the system react when sensors or operators in the dataflow are modified on the fly. Finally, we will show statistics on the execution of the dataflow and on the performances of the network.

The demonstration will thus prove the flexibility of the developed system in the specification of dataflows to be executed at network level and actuated on the fly. All the ETL operations that have been considered can be applied on-line on fresh data arriving from sensors of different types. Different controls have been included in the dataflow specification in order to guarantee the sound translation and execution of the corresponding DSN/SCN specification.

The code of SCN and ETL dataflow editors are open source and can be freely downloaded from GitHub (`https://github.com/nict-isp`).

5. REFERENCES

[1] D. J. Abadi, et al. The design of the Borealis Stream Processing Engine. In *CIDR*, 277–289, 2005.

[2] P. Alvaro et al. BloomUnit: Declarative Testing for Distributed Programs. In *DBTest*, 2012.

[3] R. Baldoni, et al. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey. Middleware for Network Eccentric and Mobile Applications, 219-244. Springer, 2009

[4] B. T. Loo, W. Zhou. Declarative Networking. Synthesis Lectures on Data Management, Morgan & Claypool, 2012

[5] S. Chandrasekaran, et al. TelegraphCQ: Continuous Dataflow Processing. In *SIGMOD*, page 668, 2003.

[6] M.S. Dao, et al. A Real-Time Complex Event Discovery Platform for Cyber-Physical-Social Systems. In *Multimedia Retrieval* Conf., 201-208, 2014.

[7] M.S. Dao, et al. EventShop. From Heterogeneous Web Streams to Personalized Situation Detection and Control. In *ACM Web Science* Conf., 105-108, 2012.

[8] M. Dong, T. Kimata, and K. Zettsu. Service-Controlled Networking: Dynamic in-Network Data Fusion for Heterogeneous Sensor Networks. In *Reliable Distributed Systems Workshops*, 94–99, 2014.

[9] D. Gay et al. The nesC language: A Holistic Approach to Networked Embedded Systems In *SIGPLAN*, 2003.

[10] M. Gorawski and A. Gorawska. Research on the stream ETL process. In *BDAS*, 61–71, 2014.

[11] K.S Kim, R. Lee and K. Zettsu. mTrend: Discovery of Topic Movements on Geo-Microblogging Messages. In *ACM SIGSPATIAL*, 529–532, 2011.

[12] N. Marz. Storm: Distributed and Fault-Tolerant Realtime Computation, 2012.

[13] M. Mesiti and S. Valtolina. Towards a User-Friendly Loading System for the Analysis of Big Data in the Internet of Things. In *COMPSACW*, 312–317, 2014.

[14] J. F. Naughton, et al. The NIAGARA Internet Query System. *IEEE Data Eng. Bull.*, 24(2):27–33, 2001.

[15] L. Neumeyer, et al. S4: Distributed Stream Computing Platform. In *ICDMW* , 170–177, 2010.

[16] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual Modeling for ETL Processes. In *DOLAP*, 14–21, 2002.