COLORED PETRI NETS IN THE SIMULATION OF ETL STANDARD TASKS THE SURROGATE KEY PIPELINING CASE

Diogo Silva **Orlando Belo** **João M. Fernandes**
ALGORITMI R&D Centre
University of Minho
Portugal
E-mail: diogosantossilva@gmail.com, obelo@di.uminho.pt, jmf@di.uminho.pt

KEYWORDS
Data Warehousing, ETL Systems, Coloured Petri Nets, Simulation and Evaluation of ETL Processes, Standard ETL Tasks, and Surrogate Key Pipelining.

ABSTRACT

ETL (Extract-Transform-Load) systems are formed by processes responsible for the extraction of data from several sources, cleaning and transforming it in accordance with some prerequisites of a data warehouse, and finally loading it in its multidimensional structures. ETL processes are the most complex tasks involved within the development of a Data Warehousing System, being crucial to model them previously so that, during the implementation stage, the correct set of requirements is considered. Coloured Petri Nets are a graphical modelling language used in the design, specification, simulation and validation of large systems, characterized as being strongly concurrent. The objective of this manuscript is to discuss the application of Coloured Petri Nets to the specification and validation of ETL systems. To demonstrate their viability for such tasks we have selected one of the most relevant and used case in ETL systems implementation: a surrogate key pipelining.

INTRODUCTION

The volume of information generated by organizations has been growing exponentially, due to the advances in information technologies, which made it easier to store, query and manage large volumes of data. Today business activities are supported by all the information that is stored in organizational data repositories and used to simplify and assist decision-makers, namely through *Data Warehousing Systems* (DWS) facilities.

ETL (Extract-Transform-Load) systems are one of the most important components of a DWS (Kimball and Caserta, 2004), as they are responsible to feed the data warehouses, ensuing high levels of data quality and, consequently, adding value to decision making processes. They are formed by specific processes of extraction, cleaning and integration of data, usually taking place in a Data Staging Area, adjusting, correcting and structuring data coming from disparate information sources so that decision makers can exploit it. The result is a highly specialized single repository, containing high quality data that is detailed, historic, subject oriented and non volatile (Inmon, 1996; 2004).

Usually, an ETL process comprises three main stages, extraction, transformation and loading. The first consists on the extraction of relevant information from its operational sources; in most cases the collected data has poor quality and errors that makes it inadequate to be directly used for populating the Data Warehouse (e.g. duplicate data, impossible or wrong values, inconsistent values due to typing errors). In the following stage – transformation – a series of rules is applied in order to increase the quality of the extracted data. This is done by correcting several errors through rectification and homogenization processes, and through the conversion of the format of the data to the one used in a data warehouse, like measure units conversion, derived attributes calculation, surrogate key generation, matching of data from different sources, among other things. Finally, the populating process that can be done by two distinct methods – refresh or update – the former rewrites information stored in the data warehouse and the latter updates it.

The ETL is the most complex and technically challenging process among all of the data warehouse process phases (Golfarelli and Rizzi, 2009), as it easily consumes 70% of the resources needed for its implementation and maintenance (Kimball and Caserta, 2004), as well as a big slice of the project time and budget. Technically, this is a difficult process to implement due to the high learning curve presented by a lot of the ETL tools available on the market, which don't offer the possibility to model the system conceptually, forcing the design and development to be made in an ad-hoc fashion by many organizations (Vassiliadis et al., 2002a). Furthermore, a poor implementation of an ETL process, which may result in low quality information, undertaking entirely a DWS (English, 1999), bringing unbearable additional costs. For these reasons, it becomes necessary to adopt some means for conceptual modelling, design and validation methodologies for the development and implementation of ETL systems, as well as proper tools to model these systems. Despite being already the centre of many research efforts (Vassiliadis et al., 2002) (Simitsis, 2003) (Abelló et al., 2006) (Golfarelli, 2008), ETL conceptual modelling stills remain almost as an island in the entire ETL life-cycle development, remaining a significant lack between it and the next ETL stages, namely the ones related to logical modelling and, of course, physical implementation.

The objective is to reduce implementation and maintenance costs, as well as the development time and risk of failure of the final DWS, which are associated frequently with a poor implementation and validation of an ETL process. As a first approach to this problem, *Petri Nets* (PNs) (Petri, 1966) come into attention, being a mathematical modelling language applied to a wide variety of systems. PNs are very adequate to describe and study information processing systems that are characterized as being concurrent, asynchronous, distributed, non-deterministic and stochastic (Murata, 1989). Being these some of the characteristics that are present in ETL systems, using PNs to model them seems to be advantageous, once it is possible to graphically represent them using a series of places,

transitions and tokens through which the concurrent activities of the system can be simulated. *Coloured Petri Nets* (CPNs) constitute a discrete event modelling language that combines the capabilities of the PN to those of a high-level programing language (Jensen and Kristensen, 2009). Additionally, they introduce hierarchic and data concepts , making them ideal for modelling of ETL systems. CPNs,
The objective of this paper is to provide a formal and well-sustained model to specify and model the behaviour of ETL systems using CPNs. Thus, we present here one of our studies carried out about modelling and validation of ETL processes: the surrogate key pipeline case, one of the most used currently in any conventional ETL system. In section 2 a brief introduction to CPNs is presented: what they are, what are their characteristics and what are the advantages of using them; followed by two examples of their industrial use through the proper supporting tools. Section 3 presents a review of the advantages of modelling ETL processes with CPNs and provides a brief presentation of the selected case study. The model created for the surrogate key pipeline case is then presented in section 4 and in section 5 a brief description of the simulation carried out. Finally, in section 6 we present our conclusions and some future research lines.

COLOURED PETRI NETS

CPNs (Jensen 1994; 1997; 1998) are a graphical language for constructing models of concurrent systems and analyse their properties. They were developed by the CPN group at Aarhus University in Denmark since 1979. CPNs are a variant of the regular P-nets that were created to fill two gaps regarding their practical application: the inexistence of hierarchic and data concepts (Jensen and Kristensen, 2009). CPN models are a combination of PNs with the capabilities of CPN ML (CPN ML, 2012), a high level functional programing language based on Standard ML, that has been used to model a wide variety of systems where concurrency and communication are among their main characteristics. The foundations for the graphical notation used in the modelling processes come from the P-nets. The primitives for the definition of data types and for describing data manipulation are provided by CPN ML, which allows the creation of compact and parameterisable models. Therefore, CPNs are high-level P-nets, but at the same time they integrate hierarchical characteristics since its possible to create models composed of several smaller modules that can be specified and tested independently.

CPN models are executable. With the CPN modelling language, a system model is both state- and action-oriented, which means that it describes the events that make a the state of a system to change. Through these models, it's possible to simulate executions in order to study the behaviour of a system. This is very useful for investigating details of a system model as well as to verify if its behaviour is correct, as is the case of the interactive simulations, in which an user controls the next steps taken in the CPN and observes the effects of each of them graphically. Simulations can also be done automatically (i.e., without user interaction) and the model is executed in order to test if it is conflict-free.

The advantages of using CPNs for system modelling are quite evident and some of them have already been described, such as the existence of an appealing graphical notation, the possibility to carry out simulations in order to test a wide variety of systems, and the possibility to have hierarchic representations that, together with CPN ML, allow us to create compact models, which is quite advantageous for large systems. Many of the concepts associated with CPNs are also present in other programing languages that modellers are certainly familiar with, lowering its learning curve. It is also possible to add time concepts to models and to verify the associated properties through its formal representation. Finally, there is support for the design, simulation and formal analysis of CPNs through computer tool applications.

CPN Tools is a computer tool that supports the construction and manipulation of CPN models, allowing for their real practical application. They were initially developed at Aarhus University (Jensen et al., 2007) (Jensen and Kristensen, 2009) and are now under the management of the Eindhoven University of Technology, the Netherlands. With this tool, it is possible to edit, simulate and analyse CPN models through a graphical user interface, which is very useful in cases of validation and debugging. With the aid of CPN tools many systems were modelled in several large scale projects belonging to different areas (e.g. software, military systems, networks and protocols), many of them designed and exploited in industrial environments (Aarhus University, 2011). Nokia, in a partnership with the CPN Group, used CPNs to model the interactions between the user and their mobile phone interface. The modelling process was performed at an early stage of the development during the mobile phone architecture design, in order to detect possible errors, and test and analyse design alternatives, giving immediate feedback to the designers (Lorentsen et al. 2001). CPNs were also adopted by Ericsson Telebit to support the design and specification of the ERDP (Edge Router Design Protocol). Based on CPN models and on CPN tools, it was possible to build a formal specification of the system, allowing the analysis of the protocol behaviour and verification of its key points. This resulted in the correction of several design errors (Kristensen and Jensen, 2004).

ETL TASKS MODELING AND SIMULATION

Most of the work and research in the DWS conceptual modelling area has been exclusively applied to schema development activities, either by using UML (*Unified Modelling Language*), variations of the entity-relationship model or dimensional modelling. However, there are some specific efforts regarding the specification of ETL processes. For instance, a generic approach to model the early design stages of these processes is presented in (Vassiliadis et al. 2002a), which is complementary to other ETL logical modelling methodologies such as UML or graphs. Additionally, in (Vassiliadis et al., 2002b) a graphical notation for the construction of graphs that represent ETL processes is defined, as well as the formal definitions for the domains and the several notations that constitute them. Latter, in (Muñoz et al., 2008), UML was presented as the modelling language for activities involved in ETL processes by using activity diagrams, with the advantage of being able to specify important characteristics of the processes (e.g. behavioural and temporal conditions) through the use of a standard language, contrary to what happened in the former approaches. However, none of these alternatives allows the

validation of the specified processes in the conceptual design stage.
CPNs (and its tool support) constitue a powerful solution to model and validate ETL processes. In addition to allowing models to be designed with the CPN Tools, it also allows to simulate their execution. Therefore, it is possible to make a more general analysis of the system to see its behaviour performance-wise, and also a detailed analysis to investigate and study the behaviour of different stages, or some other smaller processes that are integrated in them, allowing a more specific inspection providing strong basis to detect evaluate potential errors and anomalies.

Another great advantage of using CPNs to model ETL processes is that they are hierarchical nets, which allows for building larger modules through the composition of other smaller modules. This makes it possible to build specific ETL processes as a connection (or a cluster) of several CPN models representing smaller processes. In turn, they can be tested and validated independently and reused to build other modules. Taking advantage of these features, a small and simple, yet very important, process was selected as a case study to demonstrate the application of CPNs for modelling and validation of ETL processes: the *Surrogate Key Pipelining* case (SKP).

SURROGATE KEYS GENERATION USING CPNs

Regularly, the SKP process takes place during the loading stage of the records into the fact table. It's one of the last processes of this ETL stage, in which the natural keys of each record are converted into their corresponding surrogate keys. There are different approaches to implement this process, for example, using mapping tables to generate and manage these attributes. However, for maximum performance, lookup tables are used; one table for each dimension. These tables are vital for the pipelining process, as their size is significately smaller than the corresponding dimensions. This turns possible to load and randomly access them in memory, avoiding unnecessary disk reading that provably deteriorates the performance of the system. The records in this kind of tables are called lookup records and are formed solely by each dimensional record's surrogate key, generated in a previous ETL process, and one or more corresponding natural keys.

During this process the natural keys of each record are replaced by the corresponding surrogate key, thereby, there will be no natural keys in the final record but a sequence of surrogate keys, as many as the number of the existing dimensions – assuming, of course, that every single natural key must be substituted. Usually, each record should be passed though memory in a multithreaded process (Kimball and Caserta, 2004), that is, the substitution of the key of a record in memory happens simultaneously as other record substitutions that take place in different memory positions. This guarantees maximum process performance by avoiding that the fact table is written to disk before the mapping for the next dimension begins, for each substitution process of each dimension.

As a first approach to the SKP modelling, two assumptions were made: there are four dimensions in this example, and the dimensional records come from one and the same operational source, having only one natural key. Being so, each fact table record is processed four times for the substitution of its natural keys and, therefore, it is necessary to have four lookup tables, one for each substitution stage.

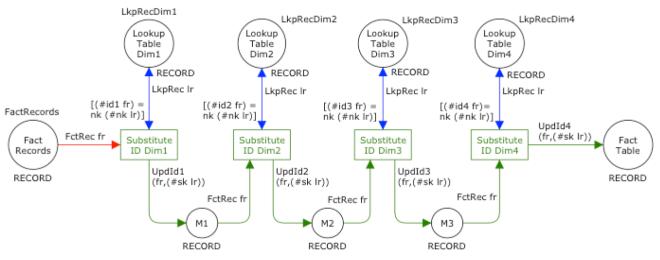


Figure 1 – The SKP model

The CPN model presented in Figure 1 represents the SKP process described previously. It is composed of nine places and four transitions. Every existing place in the CPN model is of the colour set *RECORD*. This colour set is used to model a relational database record and it is defined as the union of the different kinds of records that are used in this process, namely:

```
colset RECORD = union LkpRec:LKPREC +
FctRec:FCTREC;
```

Only two kinds of records are used in this process: the lookup records and the main records that are processed and loaded into the fact table (i.e., fact records). Still, this generalization makes the model more uniform and simple, as well as improves its readability. The places *Fact Records* and *Fact Table* are used to model the fact records before and after the key substitution process is applied to them, that is, before the natural keys of the initial records are replaced by the corresponding surrogate keys. Meanwhile, each of these replacements, the places *M1* to *M3* are used to model memory positions staged by the fact records before they are actually loaded into the fact table. For this reason, all these five places receive the same type of token, with colour set *FCTREC*, representing a fact record that was defined as follows:

```
colset FCTREC =
record id1:ID * id2:ID * id3:ID * id4:ID *
fct:NO;
colset ID = union sk:NO + nk:ST;
colset NO = int;
colset ST = string;
```

This kind of record has five fields, four of them corresponding to its keys, and the fifth field 'fct' representing a business measure. The *id* fields with the colour set 'ID', defined as the union of an integer (representing the surrogate key) and a string (representing the natural key) are used so that the same field can assume the value of a natural key or the value of a surrogate key. Therefore, a single colour set can be used to model every state of the fact record during the substitution processes, without the need to define an independent colour set to represent a fact record in each of the memory positions, as well as the initial and final fact tables, where the colour set of the id fields varies.

The remaining four places, named *Lookup Table Dim(1-4)*, are used to model the lookup tables that correspond to the four existing dimensions in the data warehouse and they are managed during the surrogate key generation process. The colour set of these places is also *RECORD* and they receive tokens of the colour set *LKPREC*. The lookup record is a simple record with one field that represents the natural key of a record inserted in the corresponding dimension in a previous process, and another to represent the corresponding surrogate key:

```
colset LKPREC = record sk:NO * nk:ST;
```

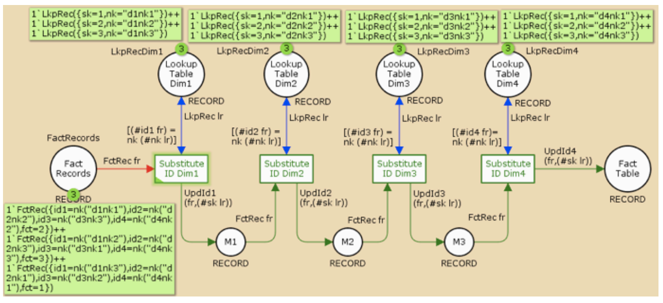
Finally, the four Substitute *ID Dim(1-4)* transitions represent the actual substitution event of the natural key of a fact record by a surrogate key. Each transition receives two tokens, one of them representing a fact record to be processed and the other representing a lookup record used to match the natural key to be substituted in that step with the corresponding surrogate key. The existing guards in each of the transitions are used to assure that the id field of the fact record is matched with a natural key belonging to one of the tokens representing the lookup records in the respective *Lookup Table Dim* place. The *UpdId(1-4)* functions in the output arcs of transitions are used to update the id fields of the fact records with the surrogate key value of the lookup record received as one of the transitions input. This function has two parameters: the actual fact record and the value of the surrogate key to be updated in the respective field:

```
fun UpdId1(fr:FCTREC, k) =
1`FctRec(FCTREC.set_id1 fr (sk k));
```

The four 'UpdId' functions are identical, varying only in the field that is being updated.

THE SKP SIMULATION

In this section, a simulation of the model execution is presented and described with the help of some images taken from three different markings of the simulation carryout.



1`LkpRec({sk=1,nk="d1nk1"})++
1`LkpRec({sk=2,nk="d1nk2"})++
1`LkpRec({sk=3,nk="d1nk3"})
1`LkpRec({sk=1,nk="d2nk1"})++
1`LkpRec({sk=2,nk="d2nk2"})++
1`LkpRec({sk=3,nk="d2nk3"})
1`LkpRec({sk=1,nk="d3nk1"})++
1`LkpRec({sk=2,nk="d3nk2"})++
1`LkpRec({sk=3,nk="d3nk3"})
1`LkpRec({sk=1,nk="d4nk1"})++
1`LkpRec({sk=2,nk="d4nk2"})++
1`LkpRec({sk=3,nk="d4nk3"})
3
LkpRecDim1
Lookup Table Dim1
LkpRecDim2
Lookup Table Dim2
LkpRecDim3
Lookup Table Dim3
LkpRecDim4
Lookup Table Dim4
RECORD
LkpRec lr
[(#id1 fr) = nk (#nk lr)]
[(#id2 fr) = nk (#nk lr)]
[(#id3 fr) = nk (#nk lr)]
[(#id4 fr)= nk (#nk lr)]
FactRecords
Fact Records
FctRec fr
Substitute ID Dim1
Substitute ID Dim2
Substitute ID Dim3
Substitute ID Dim4
UpdId4 (fr,(#sk lr))
Fact Table
UpdId1 (fr,(#sk lr))
UpdId2 (fr,(#sk lr))
UpdId3 (fr,(#sk lr))
M1
M2
M3
1`FctRec({id1=nk("d1nk1"),id2=nk("d2nk2"),id3=nk("d3nk3"),id4=nk("d4nk2"),fct=2})++
1`FctRec({id1=nk("d1nk2"),id2=nk("d2nk3"),id3=nk("d3nk1"),id4=nk("d4nk3"),fct=3})++
1`FctRec({id1=nk("d1nk3"),id2=nk("d2nk1"),id3=nk("d3nk2"),id4=nk("d4nk1"),fct=1})

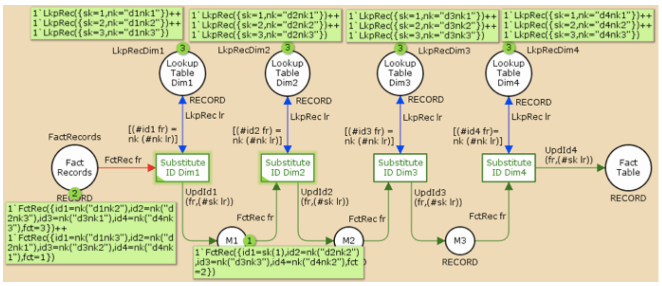
Figure 2 – Initial Marking M0

The initial marking (Figure 2) has three tokens in *Fact Records*, representing the fact records with natural keys, and three tokens in the dimensions' lookup tables, representing the lookup records used in this process. For this simulation, the values of these tokens were defined through the *FactRecords* and *LkpRecDim(1-4)* constants as:

```
val FactRecords =
1`FctRec({id1=nk "d1nk3",id2=nk "d2nk1",id3=nk
"d3nk2",id4=nk "d4nk1",fct=1})++
1`FctRec({id1=nk "d1nk1",id2=nk "d2nk2",id3=nk
"d3nk3",id4=nk "d4nk2",fct=2})++
1`FctRec({id1=nk "d1nk2",id2=nk "d2nk3",id3=nk
"d3nk1",id4=nk "d4nk3",fct=3});
```

```
val LkpRecDim1 =
  1`LkpRec({sk=1,nk="d1nk1"})++
  1`LkpRec({sk=2,nk="d1nk2"})++
  1`LkpRec({sk=3,nk="d1nk3"});
val LkpRecDim2 =
  1`LkpRec({sk=1,nk="d2nk1"})++
  1`LkpRec({sk=2,nk="d2nk2"})++
  1`LkpRec({sk=3,nk="d2nk3"});
val LkpRecDim3 =
  1`LkpRec({sk=1,nk="d3nk1"})++
  1`LkpRec({sk=2,nk="d3nk2"})++
  1`LkpRec({sk=3,nk="d3nk3"});
val LkpRecDim4 =
  1`LkpRec({sk=1,nk="d4nk1"})++
  1`LkpRec({sk=2,nk="d4nk2"})++
  1`LkpRec({sk=3,nk="d4nk3"});
```

The pre-processed fact records are formed solely by strings representing the dimensions natural keys plus the additional business measure field, while the lookup records are constituted by the surrogate/natural key pair. In this marking, *Substitute ID Dim1* is the only transition that is enabled, as it is the only one that has a token in each of its input places; one token is removed from *Fact Records* through the arc expression *FctRec fr* and the corresponding lookup record token is removed from *Lookup Table Dim1*, through the arc expression *LkpRec lr*. The guard expression [(#id1 fr) = nk (#nk lr)] acts as a restriction, so that the value of the lookup record's *nk field* matches the value of the first *id* field of the fact record. Such a guard expression is used in each of the transitions so that, in each step, the correct fact record field is matched against the lookup records' natural keys. When this transition is executed, the *id1* field of the token representing the first fact record is updated with the corresponding surrogate key, through the function *UpdId1(fr,(#sk lr))*, and passed to *M1*; this record is then ready for another key substitution, corresponding to the second dimension's natural key (Figure 3).



1`LkpRec({sk=1,nk="d1nk1"})++
1`LkpRec({sk=2,nk="d1nk2"})++
1`LkpRec({sk=3,nk="d1nk3"})
1`LkpRec({sk=1,nk="d2nk1"})++
1`LkpRec({sk=2,nk="d2nk2"})++
1`LkpRec({sk=3,nk="d2nk3"})
1`LkpRec({sk=1,nk="d3nk1"})++
1`LkpRec({sk=2,nk="d3nk2"})++
1`LkpRec({sk=3,nk="d3nk3"})
1`LkpRec({sk=1,nk="d4nk1"})++
1`LkpRec({sk=2,nk="d4nk2"})++
1`LkpRec({sk=3,nk="d4nk3"})
3
LkpRecDim1
Lookup Table Dim1
LkpRecDim2
Lookup Table Dim2
LkpRecDim3
Lookup Table Dim3
LkpRecDim4
Lookup Table Dim4
RECORD
LkpRec lr
[(#id1 fr) = nk (#nk lr)]
[(#id2 fr) = nk (#nk lr)]
[(#id3 fr) = nk (#nk lr)]
[(#id4 fr)= nk (#nk lr)]
FactRecords
Fact Records
FctRec fr
Substitute ID Dim1
Substitute ID Dim2
Substitute ID Dim3
Substitute ID Dim4
UpdId4 (fr,(#sk lr))
Fact Table
UpdId1 (fr,(#sk lr))
UpdId2 (fr,(#sk lr))
UpdId3 (fr,(#sk lr))
2
1`FctRec({id1=nk("d1nk2"),id2=nk("d2nk3"),id3=nk("d3nk1"),id4=nk("d4nk3"),fct=3})++
1`FctRec({id1=nk("d1nk3"),id2=nk("d2nk1"),id3=nk("d3nk2"),id4=nk("d4nk1"),fct=1})
M1
1
M2
M3
1`FctRec({id1=sk(1),id2=nk("d2nk2"),id3=nk("d3nk3"),id4=nk("d4nk2"),fct=2})

Figure 3 – Marking M1

In the marking *M1* (Figure 3), reached after firing the *Substitute ID Dim1* transition, there are now two enabled transitions – *Substitute ID Dim1* and *Substitute ID Dim2* – as it is possible to "remove" tokens from their input places and the guard expressions evaluate to true, allowing any of them to be fired. If *Substitute ID Dim1* is fired, a new token representing a fresh pre-processed fact record is removed from *Fact Records*, its first natural key is replaced by the corresponding surrogate key, and then passed to *M1*. If the *Substitute ID Dim2* is fired, the second natural key of the token, representing the first fact record record, is replaced by the corresponding surrogate key, and the transition *Substitute ID Dim 3* will be activated, as the token will be passed to M2. After a few markings, as the tokens are processed, all transitions become active, allowing for several records to be processed in memory at the same time, before they are loaded into the *Fact Table*.

In the final marking (Figure 4) there are three tokens in *Fact Table*, representing the final fact records processed through each of the transitions, and the same three tokens in each of the lookup tables. Every fact record's natural key have been replaced by the corresponding surrogate key value, originating fact records that are ready to be loaded into the corresponding table in the data warehouse.

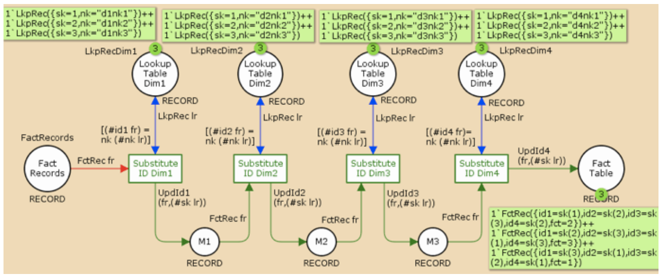

1`LkpRec({sk=1,nk="d1nk1"})++
1`LkpRec({sk=2,nk="d1nk2"})++
1`LkpRec({sk=3,nk="d1nk3"})
1`LkpRec({sk=1,nk="d2nk1"})++
1`LkpRec({sk=2,nk="d2nk2"})++
1`LkpRec({sk=3,nk="d2nk3"})
1`LkpRec({sk=1,nk="d3nk1"})++
1`LkpRec({sk=2,nk="d3nk2"})++
1`LkpRec({sk=3,nk="d3nk3"})
1`LkpRec({sk=1,nk="d4nk1"})++
1`LkpRec({sk=2,nk="d4nk2"})++
1`LkpRec({sk=3,nk="d4nk3"})
LkpRecDim1
LkpRecDim2
LkpRecDim3
LkpRecDim4
Lookup Table Dim1
Lookup Table Dim2
Lookup Table Dim3
Lookup Table Dim4
RECORD
LkpRec lr
[(#id1 fr) = nk (#nk lr)]
[(#id2 fr) = nk (#nk lr)]
[(#id3 fr) = nk (#nk lr)]
[(#id4 fr)= nk (#nk lr)]
FactRecords
Fact Records
FctRec fr
Substitute ID Dim1
Substitute ID Dim2
Substitute ID Dim3
Substitute ID Dim4
UpdId1 (fr,(#sk lr))
UpdId2 (fr,(#sk lr))
UpdId3 (fr,(#sk lr))
UpdId4 (fr,(#sk lr))
Fact Table
M1
M2
M3
3
1`FctRec({id1=sk(1),id2=sk(2),id3=sk(3),id4=sk(2),fct=2})++
1`FctRec({id1=sk(2),id2=sk(3),id3=sk(1),id4=sk(3),fct=3})++
1`FctRec({id1=sk(3),id2=sk(1),id3=sk(2),id4=sk(1),fct=1})

Figure 4 – Final Marking M12

The simulation of the model has revealed great advantages. A step-by-step execution allows for the data flow to be carefully analysed so that errors can be easily detected. In this particular case, as several transitions can be enabled at the same time and the order of their activation may vary from each execution, simulations run with the CPN Tool were very useful as they permit the selection of which transition to be executed and thus a more careful analysis of the behaviour of the ETL process.

CONCLUSIONS AND FUTURE WORK

In this paper we present a formal specification approach based on CPNs for modelling and validating DWS ETL processes, using the popular SKP case as a study case one The CPNs, together with CPN Tools, have proven to be a good alternative to model this kind of processes, because in addition to the powerful support that they offer in such modelling tasks, they also allow the implemented models to be analysed and consequently verified. The initiated study is not finished, since the number of different processes that usually integrate an ETL system is high. Currently, the model describes the behaviour of the SKP process based in predefined values and some basic assumptions. This is not enough for real world ETL systems. Thus, we need to adapt it so that it can be applied to a wide variety of ETL scenarios, as well as to receive hierarchical concepts in order to prepare the model to be reused in higher complexity processes.

REFERENCES

Aarhus University. 2011. Industrial use of CPN. [Online] Availale at < http://cs.au.dk/cpnets/industrial-use/> [Accessed on 27 June 2012]

Abelló, A., Samos, J., & Saltor, F. (2006). YAM2: a multidimensional conceptual model extending UML. Information System, 31(6), 541-567.

CPN ML , Overview of CPN ML Syntax, Version 3 . 0. [Online] Availale at <http://www.daimi.au.dk/designCPN/man/Misc/CpnML.All.pdf> [Accessed on 25 June 2012]

English, L. P., Improving data warehouse and business information quality: methods for reducing costs and increasing profits. John Wiley & Sons, Inc., New York, NY, USA, 1999.

Golfarelli, M., The DFM: A Conceptual Model for Data Warehouse. Encyclopedia of Data Warehousing and Mining (Second Edition), John Wang (Ed.), IGI Global, 2008.

Golfarelli, M., Rizzi, S., Data Warehouse Design: Modern Principles and Methodologies, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 2009.

Inmon, W., Building the Data Warehouse , John Wiley & Sons, 1996.

Inmon, W., Building the Data Warehouse, 4th ed.: Wiley Publishing, Inc, 2005.

Jensen, K., An introduction to the theoretical aspects of coloured petri nets. In A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium. Springer-Verlag, London, UK, 230–272, 1994.

Jensen, K., A brief introduction to coloured petri nets. In Proceedings of the Third International Workshop on Tools and Algorithms for Construction and Analysis of Systems. TACAS '97. Springer-Verlag, London, UK, 203–208, 1997.

Jensen, K., An introduction to the practical use of coloured petri nets. In Lectures on Petri Nets II: Applications, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets. Springer-Verlag, London, UK, 237–292, 1998.

Jensen, K., Kristensen, M., Wells, L., Coloured petri nets and cpn tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Transf. 9, 213–254, 2007.

Jensen, K., Krinstensen, L., Coloured Petri Nets: Modeling and Validation of Concurrent Systems. Springer, New York, NY, USA, 2009.

Kimball, R., Caserta, J., The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleanin. John Wiley & Sons, 2004.

Kristensen, M., Jensen, K., Specification and validation of an edge router discovery protocol for mobile ad hoc networks. In SoftSpez Final Report, H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, Eds. Lecture Notes in Computer Science, vol. 3147. Springer, 248–269, 2004.

Lorentsen, L., Touvinene, A.-P., Xu, J., Modelling feature interaction patterns in Nokia mobile phones using coloured petri nets and design/CPN. In 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01) / Kurt Jensen (Ed.). DAIMI PB-554, Aarhus University, 1–14, 2001.

Murata, T., Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77, 4, 541–580, 1989.

Petri, C.A., Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962. English translation: Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, Vol. 1, Suppl. 1, 1966.

Simitsis, A. 2003. Modeling and managing ETL processes. In VLDB PhD Workshop. Berlin.

Vassiliadis, P., Simitisis, A., Skiadopoulos, S. 2002. Conceptual modeling for ETL processes. In Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP. DOLAP '02. ACM, New York, NY, USA, 14–21.