

Reporting of data load errors

Case: IBM WebSphere Commerce

Tomáš Varčok

Bachelor's thesis

December 2016

School of Technology, Communication and Transport

Degree Program in Information and Communications Technology

Author(s) Varčok, Tomáš	Type of publication Bachelor's thesis	Date December 2016
		Language of publication: English
	Number of pages 47	Permission for web publication: yes
Title of publication Reporting of data load errors Case: IBM WebSphere Commerce		
Degree programme Information and Communications Technology		
Supervisor(s) Salmikangas, Esa		
Assigned by Solteq Oyj		
<p>Abstract</p> <p>The assigner Solteq Oyj has been struggling with the process of handling errors occurring during the data loading process into IBM WebSphere Commerce web store solution. This process has required a great deal of manual work on a daily basis.</p> <p>Therefore, it was proposed to try to identify the possibilities of improving this process. At first, studying and understanding the IBM WebSphere Commerce Data Load utility was needed in order to be able to identify the available options. After evaluation and decision-making process it was possible to continue to planning, designing and developing the improvements.</p> <p>The main functionality was implemented in the form of a module connected to the Data Load utility. Programming languages Java and Python were utilized during the implementation phase.</p> <p>The thesis resulted in a major automation possibility. Based on the provided configuration, the module is to a certain extent capable of performing all the previous manual steps such as information gathering, analysis and forming understandable and self-explanatory error messages. The employees can then focus on different tasks, which offers the possibility of saving time and reducing the costs to a considerable extent.</p> <p>The solution for a real problem was provided, and the results are very beneficial for the company. Exploring and understanding the Data Load utility resulted in gaining a great deal of useful knowledge for future use.</p>		
Keywords/tags (subjects) Data processing, automation, web store, e-commerce		
Miscellaneous		

Contents

1	Introduction	4
1.1	Objective.....	4
1.2	Hosting company.....	4
1.3	Outline of the thesis	5
2	IBM WebSphere Commerce	5
2.1	E-commerce.....	5
2.2	IBM WebSphere Commerce	6
3	Data Load utility in WebSphere Commerce	7
3.1	Main data entities	7
3.2	Data Load utility	8
3.2.1	Utility in general	8
3.2.2	Working with Data Load utility.....	9
3.2.3	Architecture of the Data Load utility	11
3.2.4	Log files	14
3.2.5	Data load errors	16
3.2.6	Handling of data load errors.....	17
4	Assignment	18
4.1	Current situation and background	18
4.2	Initial idea for improvement	19
4.3	Initial requirements.....	20
5	Research and planning.....	21
5.1	Simple architectural overview.....	21
5.2	Choosing the best approach.....	22
5.2.1	Available options	22
5.2.2	Post processing the log file.....	22

	2
5.2.3 Making modifications already to Data Load utility	23
5.2.4 Chosen solution	25
5.3 Vision of the solution after exploring and planning phase	25
5.4 Technical planning.....	26
6 Implementation.....	27
6.1 Used software development methodologies.....	27
6.2 Connecting module to Data Load utility	29
6.2.1 Architecture analysis	29
6.2.2 Connecting in the mediators	30
6.2.3 Connecting in some central place of utility	31
6.2.4 Decided solution	31
6.3 Functional analysis (algorithm)	32
6.4 Components	35
6.4.1 Module overview	35
6.4.2 Connecting components.....	36
6.4.3 Processing components.....	37
6.4.4 Assisting components.....	41
6.4.5 Configuration components.....	41
6.4.6 Configuration testing tool.....	44
7 Results	44
7.1 Current status and results	44
7.2 Putting into use	45
8 Conclusion.....	45
References	47

Figures

Figure 1. Data flow	8
Figure 2. User roles interaction with the utility and flow of the processes.....	10
Figure 3. Data Load utility architectural overview	12
Figure 4. Example of entry in a log file.....	14
Figure 5. Part of default error message written to log in case of error	15
Figure 6. Initial idea for providing more understandable error reports	20
Figure 7. Simple overview of the data load solution architecture.....	21
Figure 8. Class diagram of important part of the data load solution.....	29
Figure 9. Activity diagram of the module's algorithm	33
Figure 10. Processing of the message part of the algorithm	34
Figure 11. Log file script post-processing part of the algorithm	35
Figure 12. Schema of the module	36
Figure 13. Examples of messages used in Dynamic parameters component.....	38
Figure 14. Processing of messages by Dynamic parameters component.....	39
Figure 15. Error report with list of affected items	40
Figure 16. Configuration of the module in the XML file	42
Figure 17. Configuration of known errors, reporting messages and other related settings for the specific error	43
Figure 18. Error where the source of the error is not clear	43
Figure 19. Information from reporting e-mail	44

1 Introduction

1.1 Objective

Automation is a big trend today, since it saves time and other resources in general and especially, money. Even though information technologies are in many cases a tool to set up the automation, there are also many processes and areas which can be automated there.

One of the vital parts of running a web store is uploading data there and keeping the current data up to date. Sometimes these operations can be done on a daily basis. Data integration brings big benefits, however, on the other hand, various challenges and issues as well. It also applies to the data loading process into the IBM Web-Sphere web store solution. The current process of handling these kinds of errors in the hosting company Solteq Oyj consists of many manual procedures. Information is gathered from different sources, analyzed and the problem is explained to a data provider by telling what happened, why it happened and what steps are needed to fix it.

Even though the same types of problems reoccur, the necessary steps are always almost the same, which means that the time and cost requirements of this process are not decreasing despite growing quantity and quality of the available knowledge. Because of these reasons, the need for some improvements emerged from the employees of the company.

To solve this problem, it would be very good to introduce some solution to automate all or at least some of the steps which are now performed manually. This solution should be able to collect all the necessary data and transform it into easily understandable error messages.

1.2 Hosting company

Solteq Oyj is a company providing various information technology solutions. It is a Finnish company with several sites in Finland, and nowadays also in other European

countries. It characterizes itself as a middle-sized company. It is possible to find large Finnish companies as well as international clients among its customers.

1.3 Outline of the thesis

The thesis starts with an introduction of the product IBM WebSphere Commerce and its Data Load utility, which is the main point of interest in the whole complex solution. The problems of data loading process are introduced there as well. Latter chapters explain the current process and why improvement was needed. Additionally, the analysis and the implementation of a new module providing certain improvements is discussed.

2 IBM WebSphere Commerce

2.1 E-commerce

E-commerce is the term commonly used for electronic commerce, which is an umbrella term covering all the areas belonging to or operating electronic business – offering and selling products or services using electronic communication, especially over the Internet. Nowadays the main core of e-commerce consists of web stores providing an online option to browse a catalog of available products and services, make orders and payments, track the status of the order, etc. However, there are also other parts of e-commerce, for example, online marketing, which tries to promote the online store, its offers etc. in order to increase the number of sales and generated profit. (What is e-commerce, 2016)

Today, e-commerce is not only about providing basic options of browsing a catalog of products and capability of buying some product. Much more complex solutions are currently used to satisfy business needs of sellers of all sizes from small businesses to huge retail companies. The online store is no longer a separate system operating on its own but it is strongly integrated with other systems and services. This co-operation of multiple software solutions brings a need for big data integration and

migration. The easy flow of data brings benefits to both sellers and customers. Sellers benefit from less manual work and automation of many different processes. It then brings faster processing and delivery of the order to end customers, which is a very important aspect and competitive advantage over other online stores.

E-commerce at its highest level is divided into several groups according to the categorization of the buyer of products or services. In relation to the standard online stores, two of them are applicable. The first option is business to business (B2B), which means that both seller and buyer are some kinds of business units. Another one is a business to customer (B2C), where an end customer is a single person. There are many differences between these two areas because the whole online store (same applies to regular real stores) should always be adjusted to its customers, their needs and habits. (What is e-commerce, 2016)

2.2 IBM WebSphere Commerce

IBM® WebSphere® Commerce Enterprise is an omni-channel eCommerce platform that enables business-to-consumer (B2C) and business-to-business (B2B) sales to customers across channels—web, mobile, social, call center or store. It supports seamless marketing, selling and fulfillment with precision marketing, merchandising tools, site search, customer experience management, catalog and content management, social commerce and advanced starter stores. It dynamically optimizes content for various device types and formats including web, mobile and tablet. (WebSphere Commerce Enterprise, 2016)

IBM WebSphere Commerce (its various versions) provides companies of all sizes with a complex set of tools for operating their e-commerce businesses. It is a huge software solution trying to provide all the options which sellers might need to reach their targets – be it a basic type of service like catalog management, various marketing tools, etc.

3 Data Load utility in WebSphere Commerce

3.1 Main data entities

Data model of an online store can be complicated. However, there are still main data entities, which are the core of everything both from technical and the end user's point of view. Not all of them have to be used or they can be used only to a certain extent depending on the size of the shop, the relation of seller-buyer (B2B or B2C), its range of offered products, etc.

Category

Categories (catalog groups) represent sets of products. They can be formed into a more complex architecture when subcategories are used.

Product

Product (catalog entry) is a central entity in the web shop. It has relations to many other entities and what the end user sees as a product in the storefront is usually a combination of all related entities.

Attributes, attachments

The product can be described by different attributes (length, color, etc.), which are loaded separately, and the product can then have different values of these attributes (105 cm, blue). Also, attachments such as specification or documents with instructions can be provided to end customers.

Prices

As in a real world, also in an online store more types of prices can be found, especially differentiated by their relation to taxes. Also, it is very usual that not all the potential buyers have the same price as the seller can specify different discounts from regular prices or totally different prices for each customer (especially in B2B stores).

Customers (organizations)

Customer organizations are used in so-called B2B (business-to-business) stores, where the end customer is another organization and not a single private person.

Users

Roles and capabilities of users of the WebSphere Commerce strongly depend on the type of the store from a business point of view – whether it is B2B or B2C store.

When the user is the end customer, he/she can do basically all the operations in the web store such as buying the products in particular. In B2B stores all the users of the application always have to belong to some organization for which they can make purchases, see already placed orders, etc. Users within organizations can have different roles, which specify their available actions to perform in the web store.

3.2 Data Load utility

3.2.1 Utility in general

The Data Load utility is an enhanced business object based loading utility. This utility provides an efficient solution for loading information into your WebSphere Commerce database. You can also customize the Data Load utility to load other types of data. The Data Load utility is the recommended loading utility. (Overview of the Data Load utility, 2016)

Data load utility is a set of components responsible for loading data to an existing web store. Technically it provides the capability of moving data from input sources to database in such a form and structure, which is required by WebSphere Commerce. (See Figure 1)

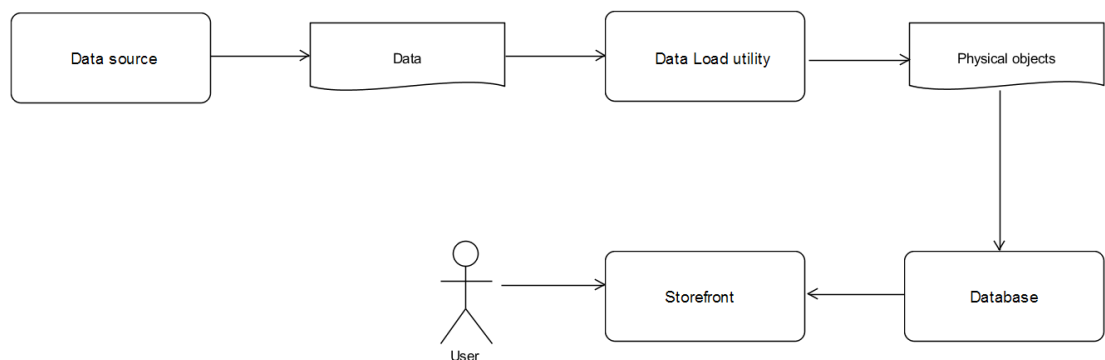


Figure 1. Data flow

Typically, data load is a part of more complex architecture consisting of more various tools and programs with different responsibilities. Their cooperation then finally provides the needed behavior.

In real use, it is not enough to get some data and load it into the database because source data can come from multiple systems, usually ERP (Enterprise Resource Planning). Also, those systems are often not capable of providing data in the format requested by the Data Load utility, which then brings the need to make modifications or put another system in place to take care of translation of the source data into needed format and structure. After the translation data can be delivered to data load solution itself and loading into the database can start. The area between translation and starting data load has space for various adjustments and modifications and more different programs, scripts or software solutions can be placed there.

A certain set of data is always given to the Data Load utility. Data are being loaded in groups of entity types in order, which is required by relations between them. For example, all the categories have to be loaded before products, which are referencing the categories.

3.2.2 Working with Data Load utility

Several user roles are needed to successfully set up the Data Load utility, provide the data in expected format, load the data and verify the results of the data load. The diagram in Figure 2 illustrates different user roles and flow of the processes.

User roles responsibilities can be defined as follows:

Business user is responsible for managing the business data. Developer is responsible for defining the data source template, business object mappings, and customizing the Data Load utility. Site administrator is responsible for the day-to-day operation of the Data Load utility. (Overview of the Data Load utility, 2016)

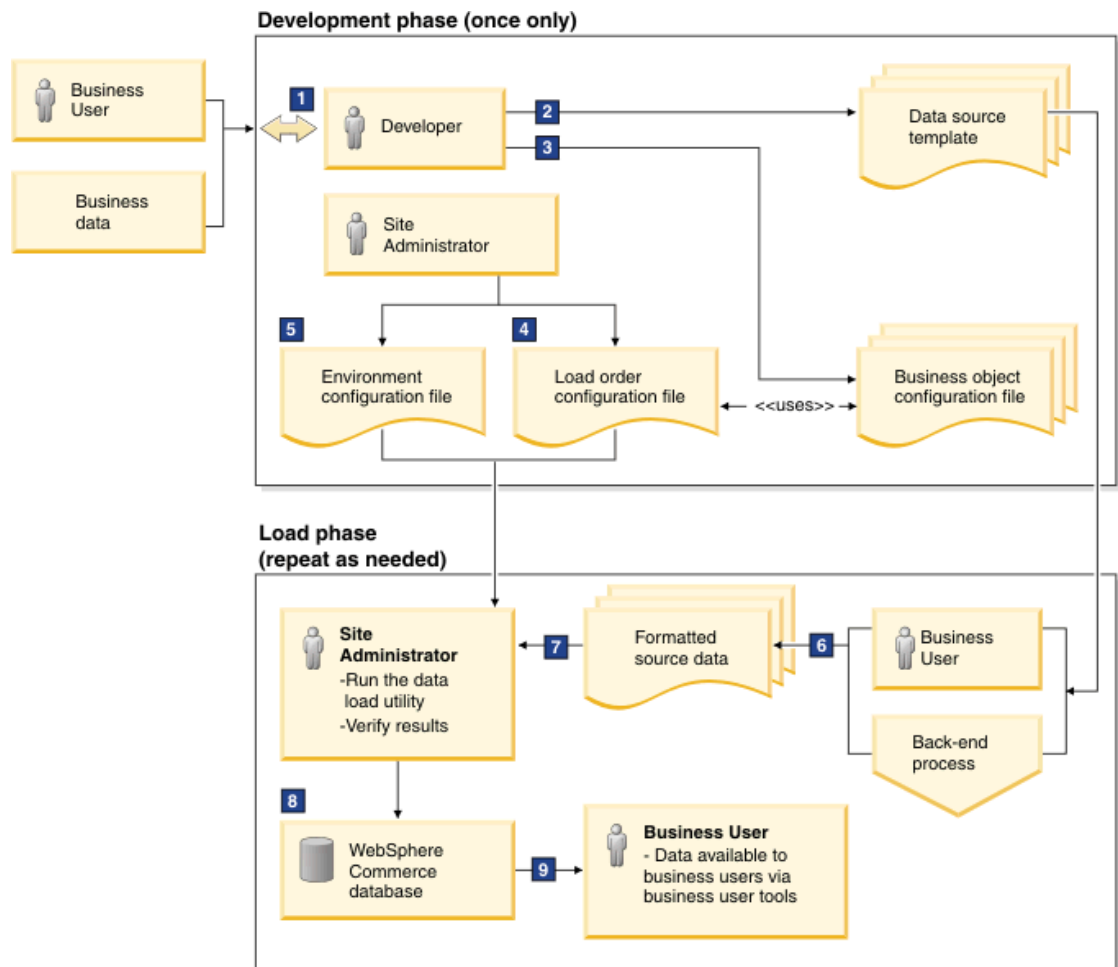


Figure 2. User roles interaction with the utility and flow of the processes (adapted from Overview of the Data Load utility, 2016)

As it is visible in the diagram, there are many different processes regarding data handling in the IBM WebSphere Commerce. All the steps and processes are explained in the following list.

1. *The business user provides the developer with the business data.*
2. *The developer creates a data source template, which defines how source data must be formatted before the data is loaded.*
3. *The developer also creates the business object configuration file. The business object configuration file defines how the Data Load utility maps the input data to the business object and how to transform the business object to physical data.*

4. *The site administrator uses the business object configuration file to define and create the load order configuration file.*
5. *The site administrator sets the store and database settings in the environment configuration file.*
6. *The business data is formatted according to the rules of the data source template before the data is loaded to the database.*
7. *The formatted source data is provided to the site administrator.*
8. *The site administrator runs the Data Load utility along with the three configuration files (environment, load order, and business object configuration files) to load the formatted source data into the WebSphere Commerce database. After the utility runs, the site administrator also verifies the results of the load.*
9. *The business data is available in WebSphere Commerce to be managed by the business user.*

(Overview of the Data Load utility, 2016)

3.2.3 Architecture of the Data Load utility

In order to work with the Data Load utility and do customizations, it is at first crucial to understand the way it works and to know its structure (see Figure 3). There are several components working together within the Data Load utility to perform all of its tasks.

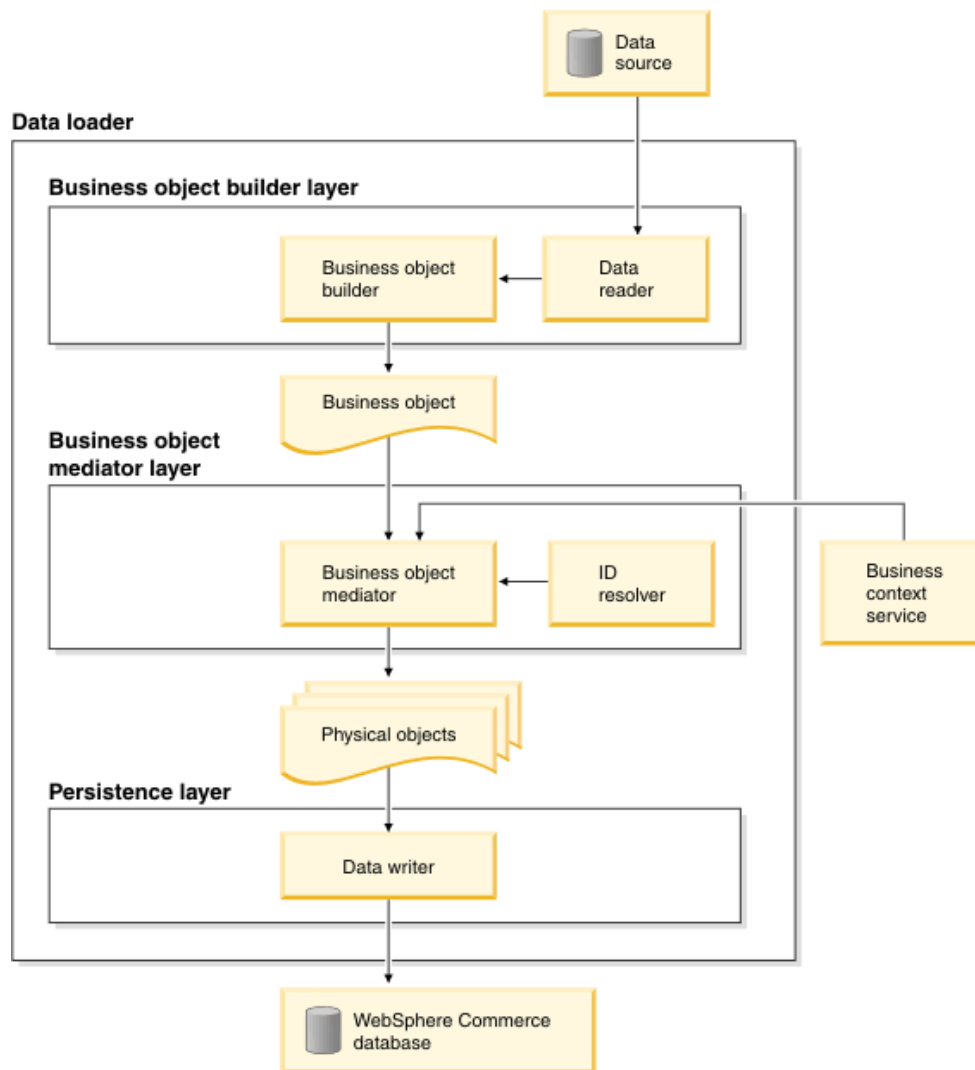


Figure 3. Data Load utility architectural overview (adapted from Data Load utility architectural overview, 2016)

Business object builder layer

The business object builder layer contains the data reader and the business object builder. The data reader is responsible for reading the raw data and passing it to the business object builder for processing and building business objects. The business object builder takes the data as input from the data reader, then populates and instantiates the business objects. Each business object is defined as a common entity throughout the WebSphere Commerce data model. In other words, you only have to understand a single representation of the data through the store front, authoring tools, and the data load infrastructure. (Data Load utility architectural overview, 2016)

The business object layer supports different types of input data sources. The data can come from CSV files (comma-separated values), XML files, external databases and other systems (mainly ERP - Enterprise Resource Planning). A CSV and XML data readers are provided out of the box with the Data Load utility. However, these readers work only with specified data format. If data which is about to be loaded into the WebSphere Commerce database has a different format, it is necessary to make some modifications to the configuration files or create new custom data reader. If the input data does not come stored in CSV or XML format, new custom data reader is always necessary.

Business object mediator layer

The business object mediator layer contains the business object mediator. The business object mediator converts the business objects into objects that represent the physical database schema, also referred to as physical objects. Several mediators are available for catalog, inventory, and price components. (Data Load utility architectural overview, 2016)

The Data Load utility also offers an option to be modified in a way that allows it to be used to load data to any table using *TableObjectMediator*. This might be useful in cases when some custom table is created in the database for some WebSphere Commerce customization. The data there can be then loaded in the same way as for any other entity, which makes the process of regular loading all types of data much simpler, because there are no exceptions, different processes, etc.

The ID resolver, which is a part of this layer, is used to retrieve the primary key of the physical object. The physical object represents a row in a table. If the object already exists in the database, its primary key is returned. If it does not, either a new primary key is returned (for this new physical object) or in some cases, it can result in an error, which will stop processing of the current object. This situation can occur if there is a reference to a non-existing object in the database, e.g. when some product is being loaded and it refers to a category, which does not exist in the database. ID resolver then tries to obtain primary key of this specific category entry, which results in an error.

Persistence layer

The persistence layer saves physical objects received from the previous layer into the database using data writers.

3.2.4 Log files

Each run of the Data Load utility produces a log file with the sum of all the information related to that specific run. This file is a key for debugging problems of data, performance or Data Load utility itself. An example of summary information from the log file can be seen in Figure 4. Some parts of this log entry were modified to hide sensitive information.

Processing ProductUpdate...

Load summary for load item: ProductUpdate.

Business Object Configuration: *****

Data loader mode: Replace.

Batch size: 1.

Commit count: 100.

Error Tolerance Level: 1.

Error Count: 0.

Amount of data processed: 12.

Amount of business objects processed: 12.

Amount of business objects committed: 12.

Data loader initialization time: 0 seconds.

Data loader execution began: Sat Nov 12 11:27:59 EET 2016

Data loader execution ended: Sat Nov 12 11:28:02 EET 2016

Data loader completed in 2.819 seconds.

Total flush time: 0 seconds.

Total commit time: 0.004 seconds.

Total ID resolver time: 1.519 seconds.

CSV file location: *****

Affected tables (10):

Table	Total	Insert	Update	Delete
-----	-----	-----	-----	-----
CATGPENREL	12	0	12	0
ATCHREL	20	0	0	20
MASSOCCECE	0	0	0	0
CATENTRYATTR	0	0	0	0
CATENTRY	12	0	12	0
CATENTDESC	12	0	12	0
CATENTSHIP	12	0	12	0
CATENTREL	6	0	6	0

Figure 4. Example of entry in a log file

The log shows times of start and end of the whole process as well as the duration of loaders of each entity type. Those can come useful for performance analysis. There is summary information about the amount of data which has been processed, how many business objects and database tables have been affected. The log also contains information about what operation was performed (inserting a new item, updating existing one, removing) and how many times based on the provided data. (Data Load Utility in WebSphere Commerce Introduction, 2016)

If everything goes smoothly, the log does not have to be inspected very often. The main reason can be monitoring performance, gathering statistical data, planning and doing some performance improvements.

If any problem occurs during loading the data there are error messages, exception messages and stack traces of exceptions written into the log file (See Figure 5). The log file then needs to be checked properly, and all the important information from there has to be investigated in order to identify the real cause of the problem.

```
An exception was caught: com.ibm.commerce.foundation.dataload.exception.DataLoadApplicationException: The ID was not resolved for the table CATGROUP
with the unique index data [NonExistingCategory, 20000000000000012345].
Application message:
An error occurred while transforming the data object into physical objects. Data Object:
com.ibm.commerce.catalog.facade.datatypes.impl.CatalogGroupTypeImpl14aa24aa2 (displaySequence: 1.0, dynamicCatalogGroup: <unset>, navigationPath:
null, topCatalogGroup: false), Physical objects:

Exception message:
The ID was not resolved for the table CATGROUP with the unique index data [NonExistingCategory, 20000000000000012345].
Stack trace:
com.ibm.commerce.foundation.dataload.exception.DataLoadApplicationException: The ID was not resolved for the table CATGROUP with the unique index
data [NonExistingCategory, 20000000000000012345].
    at com.ibm.commerce.foundation.dataload.idresolve.IDResolverForOneTable.resolveId(IDResolverForOneTable.java:454)
    at com.ibm.commerce.foundation.dataload.idresolve.IDResolverImpl.resolveId(IDResolverImpl.java:441)
    at com.ibm.commerce.foundation.dataload.util.DataLoadHelper.resolveIds(DataLoadHelper.java:2877)
    at com.ibm.commerce.foundation.dataload.util.DataLoadHelper.resolveIds(DataLoadHelper.java:2922)
    at com.ibm.commerce.foundation.dataload.util.DataLoadHelper.resolveId(DataLoadHelper.java:2825)
    at com.ibm.commerce.foundation.dataload.businessobjectmediator.AbstractBusinessObjectMediator.resolveId(AbstractBusinessObjectMediator.java:1273)
    at com.ibm.commerce.foundation.dataload.businessobjectmediator.AbstractBaseCatalogMediator.resolveCatalogGroupUniqueID(AbstractBaseCatalogMediator.java:546)
    at com.ibm.commerce.catalog.dataload.mediator.AbstractBaseCatalogMediator.resolveCatalogGroupUniqueID(AbstractBaseCatalogMediator.java:668)
    at com.ibm.commerce.catalog.dataload.mediator.AbstractCatalogGroupMediator.populateCATGRPREL(AbstractCatalogGroupMediator.java:275)
    at com.ibm.commerce.catalog.dataload.mediator.CatalogGroupMediator.transform(CatalogGroupMediator.java:337)
    at com.ibm.commerce.catalog.dataload.mediator.CustomCatalogGroupMediator.transform(CustomCatalogGroupMediator.java:83)
    at com.ibm.commerce.foundation.dataload.businessobjectmediator.AbstractBusinessObjectMediator.execute(AbstractBusinessObjectMediator.java:456)
    at com.ibm.commerce.foundation.dataload.businessobjectbuilder.AbstractBusinessObjectBuilder.processData(AbstractBusinessObjectBuilder.java:571)
    at com.ibm.commerce.foundation.dataload.businessobjectbuilder.AbstractBusinessObjectBuilder.processData(AbstractBusinessObjectBuilder.java:516)
    at com.ibm.commerce.foundation.dataload.businessobjectbuilder.AbstractBusinessObjectBuilder.execute(AbstractBusinessObjectBuilder.java:285)
    at com.ibm.commerce.foundation.dataload.AbstractBusinessObjectLoader.processBusinessObjectBuilder(AbstractBusinessObjectLoader.java:2077)
    at com.ibm.commerce.foundation.dataload.AbstractBusinessObjectLoader.processDataObject(AbstractBusinessObjectLoader.java:2019)
    at com.ibm.commerce.foundation.dataload.AbstractBusinessObjectLoader.loadData(AbstractBusinessObjectLoader.java:1853)
    at com.ibm.commerce.foundation.dataload.AbstractBusinessObjectLoader.execute(AbstractBusinessObjectLoader.java:481)
    at com.ibm.commerce.foundation.dataload.DataLoaderMain.execute(DataLoaderMain.java:458)
    at com.ibm.commerce.foundation.dataload.DataLoaderMain.main(DataLoaderMain.java:215)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:60)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:37)
    at java.lang.reflect.Method.invoke(Method.java:611)
    at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:267)
```

Figure 5. Part of default error message written to log in case of error

The messages in the log file can sometimes be quite clear for a person with better knowledge of this area, however, still not really understandable for not so skilled person as they are not very user-friendly and easy to interpret. Some error messages can be a complicated case even for a specialist knowing the product and it can take a longer time to totally understand what is wrong and what the cause of the error is.

The Data Load utility is able to receive data mainly in XML and CSV formats. Even though from a performance point of view XML is worse, for investigating errors of data loading process it is better thanks to human readability and better structure. Finding an error or any kind of problem there is not so hard if the person knows what he/she is looking for. CSV format brings much smaller amounts of data, which is a big benefit for the performance of data load process. If the load runs without problems, it really is more beneficial, however, in the case of frequent occurrence of errors in the data, investigation and data analysis becomes harder.

3.2.5 Data load errors

The term “data load error” is next defined for all the possible errors and problematic situations, which can occur during running the Data Load utility. Those errors can be mainly distinguished into two groups – errors not caused by data and related to data.

Data load errors not caused by data

There are more causes of errors, which prevent the Data Load utility from the correct handling of the data. The first group is not caused by the data itself. The main groups of their causes can be:

Wrong configuration – the Data Load utility provides many options for settings and if it is not set up properly, it can cause different problems.

Incorrect customizations – different parts of the data load solution can be customized in many ways, and various problems can occur there, for example because of error in the code.

Database problems – the database of IBM WebSphere Commerce is quite huge and the amount of data can be big. It can happen that some database process or script is

running at the same time as the data load, which can result in tables being locked and inaccessible for the data writers of the Data Load utility. In these cases, it is usually enough to re-run the data load with the same data a bit later.

Data load errors related to data

The amount, variety, and structure of data for web store are big and complex. There is different information related to single entities. For example, products can have relations between each other, multiple attachments, they can have more prices linked to themselves, etc. In such a complex data model, the occurrence of problems can be quite high.

Data load utility is responsible for loading the data into the database and if there are any restrictions on the data which are not met, this is the place where problems will be found. Also, relations between entities are transformed into database relations between tables and if there is some incorrect reference (to not existing entity, etc.), it will not be possible to load the data correctly. Most typical problems with data are:

Incorrect references - This problem will arise when some part of the data, which is just being loaded (processed by the Data Load utility), refers to something that does not yet exist in the database. As an example, the relation between a product and a category where it belongs can be used. If the category referred to by the product does not exist, it can be caused by a simple mistake in its name (or identifier) or a fact that even though the category has to be loaded before the product, it did not take place for some reason.

Incorrect format of data - For some information, there are specific restrictions on their format. An example of this kind of error can be a date or an e-mail in an invalid format.

3.2.6 Handling of data load errors

Handling of data load errors can become a quite annoying manual activity in the life of a support team member; however, it is important to understand that these errors are usually alerts notifying that something is wrong and should be taken care of. In some situations, when for example some requirement is not met, it might not be

preventing loading the data totally but it is better when loading is stopped in order to prevent incorrect or inconsistent data from being moved to the web store and endangering its correct working for the end customers.

4 Assignment

4.1 Current situation and background

After the previous introduction into WebSphere Commerce, the Data Load utility and other information, this chapter describes a specific case in hosting company. What was the situation before the idea for the topic of this work, how was it handled, what were the possible areas for improvements, etc.

After some web store is developed and finished, the support team starts being in charge of taking care of any problems when the store is “live”, i.e. it is accessible, used by end customers and real purchases are made there. This live usage brings the necessity to update the data in the store sometimes even on a daily basis. Any problem or delay with updating the data in running store is critical for the store owner and therefore, any data load errors have to be handled (investigated and solved) as soon as possible. However, there are many other responsibilities for the support team members as well and if the data load error is caused by data, sometimes somebody else responsible for preparing or generating the data has to act and solve it, which means that the process of solving data load errors can take some time.

The current process of handling the data load errors is as follows. Data is received from a data provider, certain translations and modifications are done and then the data load runs. When the Data Load utility finishes with some error status, an e-mail is generated and sent with information about the incident. A ticket is created in an issue tracking system. Some member of the support team has to start investigating what happened and what is the cause of the problem. It usually starts with reading the log file, searching in the source data and also in the database. After the cause is identified either some actions are made on the support team side or, in case it is

needed, the problem has to be explained to the data provider with explanation what happened, why and what needs to be done to fix it.

From the point of view of a member of the support team, in some case data load problems are complicated, however, on the other hand, some of the problems often reoccur in very similar form. The time to solve the error (find its cause) then varies very much, however, there is still certain time spent regardless of the error itself just by processing related activities, which means that even without complicated logs and not very helpful error messages, handling these errors can be quite demanding especially from time consumption point of view and if there is any room for improvement, it is more than welcome.

For some of the errors, there is already an existing document, which basically contains a table with information about error message written from the Data Load utility to the log and some instructions or information to that specific error; nevertheless, this document serves only as guidance for members of the support team and it has to be manually checked every time when needed.


Look to the future shows that this situation will not get any easier as complexity and amount of data loads including related errors will most probably increase all the time. On a longer scale, this can result in higher workload on the support team members and longer times for solving problems in general. Anything that would make handling data load errors easier can, therefore, save time and costs.

4.2 Initial idea for improvement

The basics of the idea about some improvements were brought by two employees – a company's data load specialist and a manager in charge of the support team. The perfect situation would be to have some sort of automation which would recognize errors and provide some more understandable error messages. Figure 6 illustrates the basic idea of translating the default error message. A wish or an idea of some kind of improvement existed already some time but a clear and precise idea what and how can be actually done was missing. It was also not clear if something can be done at all because the Data Load utility, as well as the whole WebSphere Commerce

solution, is really complex and many parts cannot be modified at all. However, both employees emphasized that any improvement which would make work of support team members easier is more than welcome.

The ID was not resolved for the table CATGROUP with the unique index data [CategoryName, 20000000012345].



Category which does not exist in the database was referenced in the data. Name: CategoryName.

Figure 6. Initial idea for providing more understandable error reports

The specialist of the company who is responsible for the data load area already did certain improvements, however, usually, they were only “ad hoc” (for the specific single case). He was also not aware of options how to bring some better solution, where exactly to place it, and so on.

4.3 Initial requirements

Because it was not clear what the possibilities of making any kind of improvement to the Data Load utility or the company’s processes are, no very precise requirements on the final solution were specified at the beginning of the work.

The main tasks could be grouped into these phases:

- studying and understanding the current process of handling data load errors,
- understanding and exploring the Data Load utility and options of its extending or modifying,
- trying to identify options for improvements (especially technical ones) in order to achieve time and costs reductions,
- evaluating the previous phases and identified improvements,
- planning, designing and developing the technical improvements.

Except the above, discussions and consultations were expected during the phases to keep supervising persons up to date with the current progress and to eventually modify and add the requirements or change the approach.

5 Research and planning

5.1 Simple architectural overview

The basic architecture of used data loading solution can be represented by the following diagram. (See Figure 7)

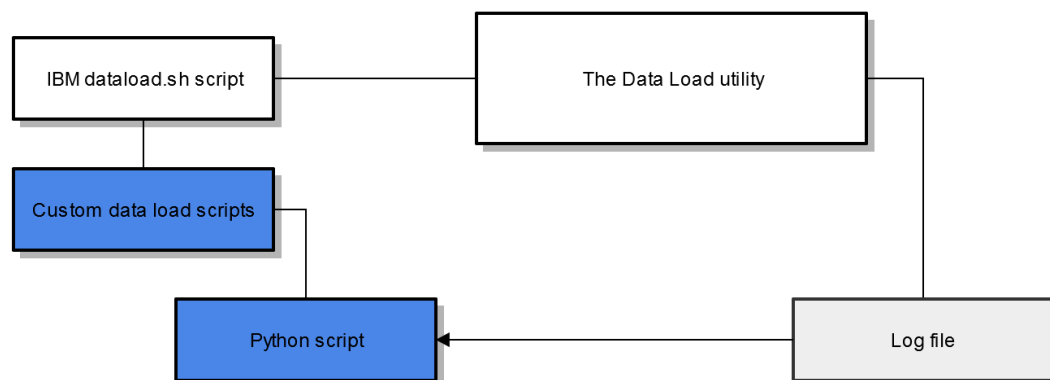


Figure 7. Simple overview of the data load solution architecture

The whole process of loading the data is started in custom data load scripts which do certain activities before the Data Load utility is started and then when everything is ready, they start the load via IBM original *dataload.sh* script. This bash script starts the Java process of the Data Load utility itself. The log file is constructed during the loading process and after the process is finished, it is given to the Python script which is responsible for post-processing of the log file's content and also picking up information, which will be placed into an e-mail informing about data load errors.

The places, which are marked as blue, were identified as fully accessible and modifiable. No major technical or license restrictions apply to them. The other parts might be modifiable to a certain extent, however, some restrictions or complications have to be taken into consideration.

5.2 Choosing the best approach

5.2.1 Available options

To achieve the goal – having some solution providing more understandable messages from the Data Load utility – it was necessary to come up with a new solution in form of some kind of module, which would be connected to the Data Load utility and which would perform operations with identified errors. There were two basic approaches available from the beginning, based on the placement of the new solution and the basic way of working.

5.2.2 Post processing the log file

Taking a standard Data Load utility log file and processing its content probably with some scripting programming language is one of the options. Small basics in form of few ad-hoc solutions for specific errors and situations were already in place by using Python script to read the log file's content line by line and trying to identify the pre-defined text. Searched strings were specified directly in the source code as the purpose of this solution was nothing more complex than taking care of few single cases.

This Python script could have been modified so it would bring some improvements into the process, however, it would be quite limited to information contained in the log and it would be able to perform mostly only data extraction and translation. Nevertheless, even translating the error messages contained within the log into more understandable explanations and instructions could bring some benefits if that information was separated and nicely structured.

From a technical point of view having the new solution totally separated from the Data Load utility itself would be a big benefit as it could not affect the loading of the data in any way in case of some error or failure. Also, it would run after the data load

finishes and data are loaded into the database, so it would not affect data load run time at all. Data load utility always works with single messages and so the log file is built incrementally. It could be beneficial to have the option to access it as one final set and be able to make summaries, analysis, etc. and not just work on a level of simple messages being written into the log.

The benefits of this approach are listed as follows:

- Deployment and changes are much easier.
- It would be a totally separated solution not influencing the Data Load utility.
- It cannot affect the data load run time.
- There is an option to not work only on the level of single error messages.

The disadvantages of this approach:

- There is a very limited amount of information available.
- Queries into the database are a bit more complicated but possible.

5.2.3 Making modifications already to Data Load utility

From the first impression making some modifications to the Data Load utility or connecting some external solution to it would seem as a much better way to achieve the specified goals. Especially, it seemed to be possible to access some more detailed information than those provided in the error messages in the log file, e.g. precise name or identifier of the item, which failed as this information is usually missing in the error messages and is important for solving the cause.

The main problem is that the Data Load utility has only a few extendable or modifiable components. The core of the solution cannot be touched from a license point of view and its configuration is really weak. There are more things to configure or modify in the Data Load utility, however, they are only in few certain areas of the whole

solution. A major part of the utility was simply not meant to be modified, configured or extended because normally there is no need to do that.

Data available in all parts of the utility varies very much, and primarily this would be the main aspect of choosing a place where to connect the new module for improved reporting. With the above mentioned strong limitations in place, finding a connection point gets very hard and can basically turn the whole idea into being impossible to implement or achieve much smaller improvements than originally expected.

Being part of the Data Load utility, modifications of the module would become a bit harder because of the deployment processes and practices. If there was some problem with this solution, it could negatively affect run of the data load, thus, the requirements for a safe implementation are much higher. Also, as a part of the utility and run of the data load, it could affect the run time. However, as the slowest part of the whole data loading process are many queries to the database, the effect should be minimal unless there are many extra database queries performed by the module.

Components of the Data Load utility usually work with single items being loaded. If there is any error message, the main point of interest in this study, the message is processed as a single item going from the place of origin through some processing classes to the log.

Benefits of this approach are listed as follows:

- There is an option to access more detailed information.
- It is easier to make queries into the database.

Disadvantages of this approach are:

- Deployment and changes are complicated.
- The possibility that it can negatively affect data load run.
- It is affecting data load run time.
- It brings limitation of working on single error message level.

- It is very hard to connect into the data load solution.
- There are many limitations because of non-extendibility of the Data Load utility.

The list of disadvantages is much longer than in the first option and also longer than benefits, however, the option to gather more precise data is very important and so it outweighs most of the negatives.

5.2.4 Chosen solution

As the precision and amount of provided information about the data load error, which occurred during loading the data, is a crucial aspect for evaluating the benefits of this module, it is necessary to connect the module in direct touch with the Data Load utility. On the other hand, some possible improvements can be done only by post processing the log file and also sending e-mails is handled on that side, therefore, it was also decided to keep some functionality on the post-processing side.

5.3 Vision of the solution after exploring and planning phase

After exploring the data load solution and available options it was possible to form its more precise vision.

The solution with the capability of providing more detailed information about data load errors occurring during loading the data will be created in a form of a module, which will receive error messages from the Data Load utility in their original form. Different processes and activities will be performed in order to process and eventually enrich the contained information about the error. The target is to provide detailed information about the error to the end user analyzing the data load error reports.

The module will be connected close to the Data Load utility in order to be able to perform its operations above the resulting error messages. It will be implemented in Java as the rest of the Data Load utility. A certain part of the solution will be placed in

Python script, which is post processing the log file after the data loading process ends.

At first, the people in touch with the module and its outcomes will be only support team members. After testing and fulfilling the database of known errors, explanations, and instructions, also other people can start working with the module's outcomes as they will already be precise and informative enough also for not so experienced people with strong technical knowledge.

The whole module can save time and costs during the process of handling data load errors and later, it can eventually make the process simpler and reduce the amount of work to be done exclusively by the support team.

5.4 Technical planning

The module will receive single logging messages, check the presence of known error messages there and if some will be identified, it will make further steps to process the data and provide as much information as possible for this specific error. Basic processing will mean adding configured explanation to this error. If possible and applicable, any other more detailed information should be provided as well to make understanding and solving the cause of the error as simply and fast as possible.

Settings will be kept in XML configuration files, and it must be possible to configure the database of known data load errors and also the module itself via these files.

After the message is processed and more detailed information is written into the log file, post processing script then reads the file and picks certain parts of information to put it into the e-mail informing that some error has occurred during data loading process.

6 Implementation

6.1 Used software development methodologies

During the development, principles of some known software development methodologies were used. Some were used on a larger scale than the others and from some of them, only certain principles or practices were used based on this specific project's needs.

Exploratory programming

Exploratory programming is an important part of the software engineering cycle: when a domain is not very well understood or open-ended, or it's not clear what algorithms and data structures might be needed for an implementation, it's useful to be able to interactively develop and debug a program. (Exploratory Programming, 2016)

As mentioned, the whole Data Load utility is a quite closed solution with only some areas modifiable or configurable. For the purpose of developing module reporting data load errors, these constraints presented critical obstacles which might have caused a fail of the whole development in any moment when some new critical obstacle were to be identified. With this situation, exploratory programming seemed a very good and necessary approach to planning and developing the solution.

At first, it was necessary to explore and understand the Data Load utility. After that it was needed to identify important points of the utility as steps in the process, places in the data flow and specific classes and test available options and data there. Even though some places are modifiable, it does not mean they are suitable for the planned purpose. The main problem except few modifiable places was availability or accessibility of the data at those places. As the documentation to non-modifiable places is very weak or not available at all, the only way how to find out what options and data are available was to use approaches of exploratory programming.

Agile methodologies

As precise planning and specification of the solution in advance was almost impossible due to the aforementioned constraints and lack of knowledge of the domain, using non-flexible and non-adaptable methodologies such as Waterfall model could not be considered. It was necessary to be able to modify requirements, plans and specifications during the development process, which corresponds to one of the key ideas of agile methodologies – regular and expected adaptation to changing circumstances.

Precise and complete requirements could not have been specified at the beginning of the assignment, which made it harder to deliver something that would really satisfy the needs, as there could have been some misunderstanding of those needs and some of them might have been forgotten. For that reason, it was necessary to be able to prepare and show a demonstration of the current status to have some base over which the discussions and further planning can be held.

Agile software methodologies offer many methods and practices which perfectly fit into the specific needs of this project to make development possible and to make sure expected product is delivered in the end.

Following steps in the development were gathered in a backlog, and the sprint backlog was always derived from it according to the current situation. As the solution is not so huge and the use of exploratory programming approach was necessary very often, the sprints were usually very short compared to bigger projects.

Iterative and incremental development

Because of the need to deliver demo versions for presentations and consultations of the current status, developing in iterations and building the product incrementally was very useful.

6.2 Connecting module to Data Load utility

6.2.1 Architecture analysis

Connecting the module to the Data Load utility posed a critical obstacle. To connect the module to the Data Load utility, it was necessary to find a class with an access to the majority of information and especially, a possibility to modify in order to establish a connection to the module. A closer look at the architecture of the solution can be seen in the following diagram. (See Figure 8)

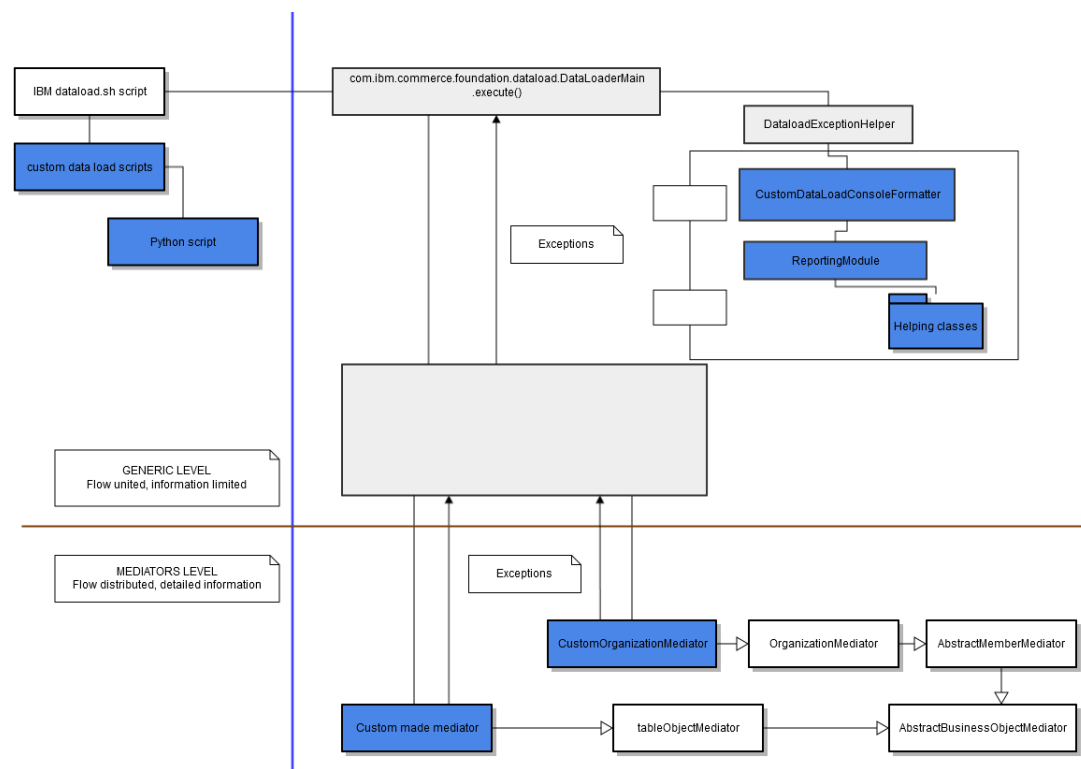


Figure 8. Class diagram of important part of the data load solution

The diagram is divided vertically into the Java part on the right and the script part on the left. Horizontal division separates generic and mediators' level. At the mediators' level, the flow of processes is distributed by using different mediators. It is then hard to have a central place to work with information, however, on the other hand, mediators have access to the details of the data. The generic part then contains the core

part of the Data Load utility. For formatting the log messages, the utility uses defined formatter class. The blue color shows the modifiable parts of the solution.

There were basically two options to consider: either to connect the module on the low level represented by mediators, or to some higher level, where the flow of processes and especially information is united.

6.2.2 Connecting in the mediators

Business object mediators (described in the chapter about the architecture of the Data Load utility) are an ideal place from the information point of view as they are very close to the data, and it is possible to access more detailed information about the item, which caused some problem and its loading failed. The problem is that at this level, information and process flow is distributed in large scale into many different places (mediators). Some of them are customized, however, in many cases, the default ones are used. These mediators have sometimes a complicated, however, mainly very different inheritance hierarchy. Parent classes are usually IBM default implementations, which cannot be modified. This means that it was impossible to identify a single common place for all the mediators. Any connection of the module at this level would require modifying basically all the custom mediators and overriding other ones, which would mean modification of a large scale affecting the whole data loading solution.

The benefit is as follows:

- Allows easy access to detailed data of a problematic item.

The disadvantages are the following:

- It would require modification of all custom mediators.
- It would require overriding all the default mediators which are used.

6.2.3 Connecting in some central place of utility

The ideal connecting point would be located in the central part of the diagram, where the information flow is united, however, there is still access to some additional information. As the amount of changes in this common central place was to be very small, this modification would not have very big impact on the data loading solution. However, as shown in the diagram, this area has no modifiable classes because modifying this part of the Data Load utility was never expected or meant to be allowed.

Benefits of this solution are:

- There is a possibility to catch all the error messages at one central place.
- Only low amount of changes to the Data Load utility is needed.

The disadvantage is as follows:

- Access to information about the problem is reduced.

6.2.4 Decided solution

Extending the logging formatters has been identified as the best possible connection way. Exploration showed that the formatters are used to format all the messages written into the log files, which means that all the information contained in the log files goes through the formatter before being written there. That makes it the perfect place to catch and investigate all the messages. There are more such places, however, they are not accessible because their code is not modifiable. On the other hand, the formatter class used by the Data Load utility can be set in data load logging properties file.

Connecting the module in this central place brings the capability of processing all the data load errors in one place, which is a very big advantage. However, the possibility to collect more detailed information in this place is strongly lower. For some errors, it is not necessary to provide much more additional information, however, sometimes

default error messages do not contain enough information, which is why it was decided to also modify certain mediators as well. This step is to be performed only in case of an error occurring often, and manual work will be strongly reduced or totally eliminated when also more detailed information from the mediator will be provided.

6.3 Functional analysis (algorithm)

The implemented algorithm of the module is presented in the diagram in Figure 9. At first, the module receives the message, which should be written into a log file. This message is checked for the presence of known errors. If no error is identified, the algorithm ends. If one of the set errors is found, it is processed in order to get the reporting message ideally with as many details as possible. As the same error messages can be written into the log file multiple times, the algorithm checks if the same error was already identified and stored to be reported. After the whole data load ends and the log file is finished, its post-processing by a script starts in order to extract important and summary information. The steps of processing the message and script post processing of the log file are described separately in their own activity diagrams and description.

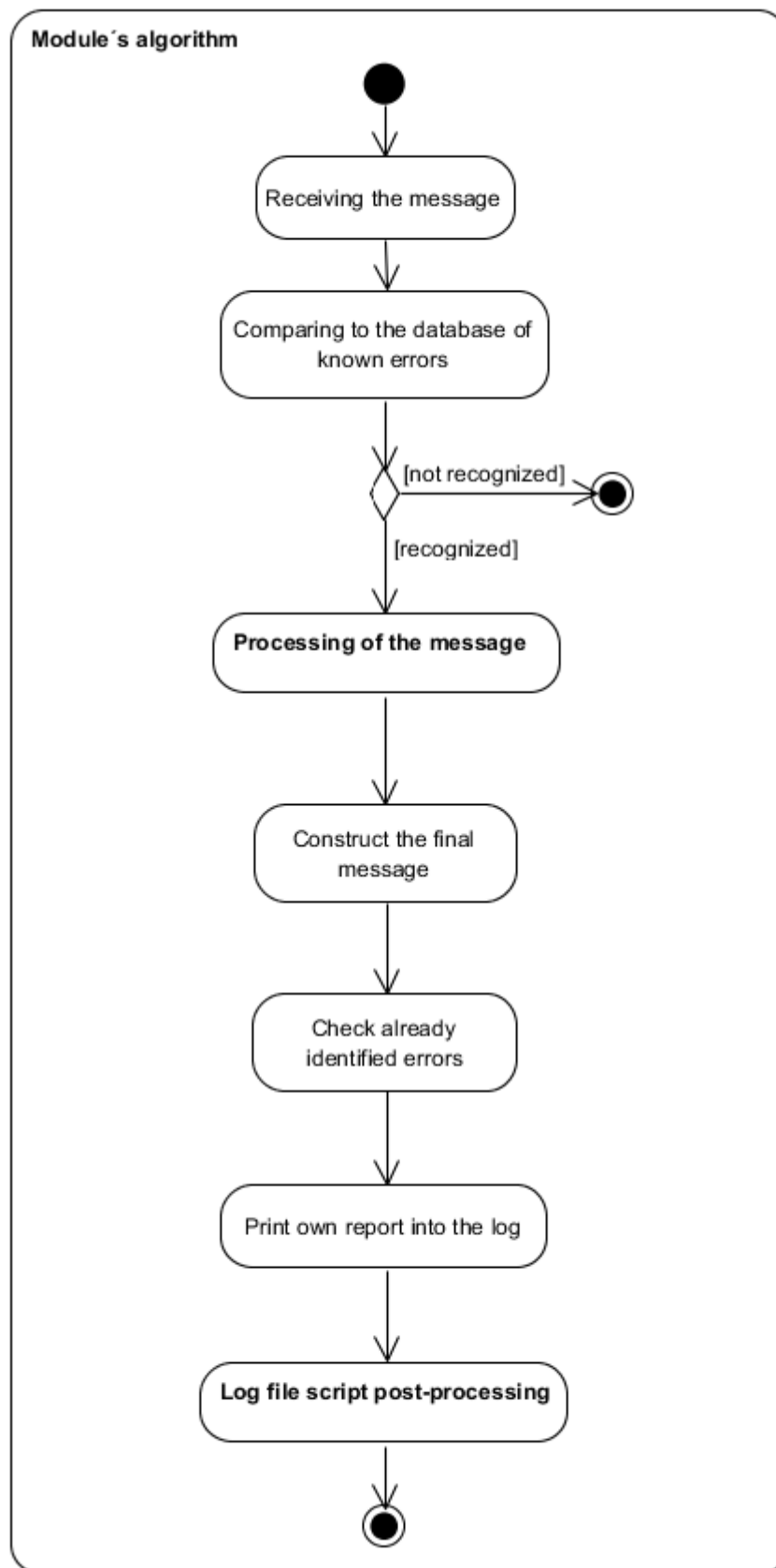


Figure 9. Activity diagram of the module's algorithm

Sub-activity of processing of the message is illustrated in Figure 10. After the error message is identified, it is translated into a pre-configured report message. This message is more understandable and explains the problem in a way that enables its fast solving. If there is some useful dynamic information (related to the specific occurrence of that error type) contained within the message, it is extracted and placed into the reporting message. It can be, for example, the ID of the item which caused the error. Both dynamic and static parameters are explained later. If there is a database query specified for this error type, it is performed and the result can be written into the final reporting message.

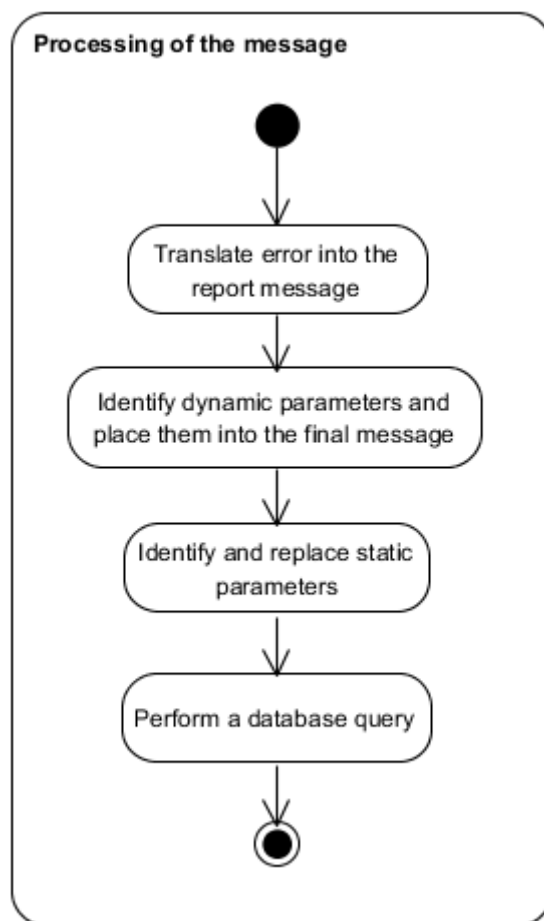


Figure 10. Processing of the message part of the algorithm

After the data load is finished and the log file is saved, the Python script goes through its content. See Figure 11 for the activity diagram of this process. During the reading process it collects statistical information (e.g. a number of processed objects, error counters, ...) and it also extracts the module's reporting messages. After the file reading is finished, it utilizes all the collected information to form an informative e-mail providing the most important information immediately after the first look.

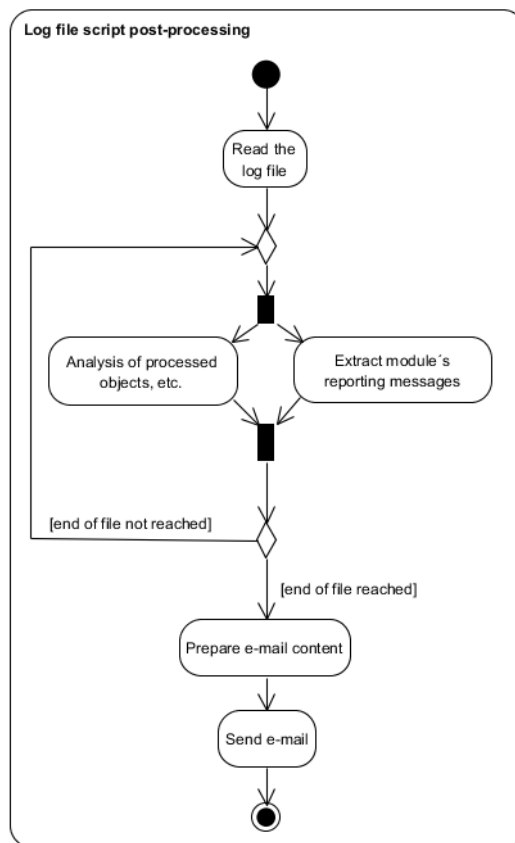


Figure 11. Log file script post-processing part of the algorithm

6.4 Components

6.4.1 Module overview

Figure 12 represents schema of the module. Central component controlling all the other ones is ReportingModule. There are few exceptions, but usually other components do not communicate directly to each other. Connecting components are

placed in the top part of the schema. Components related to the configuration are on the left. Processing and assisting components are then in the right and bottom part.

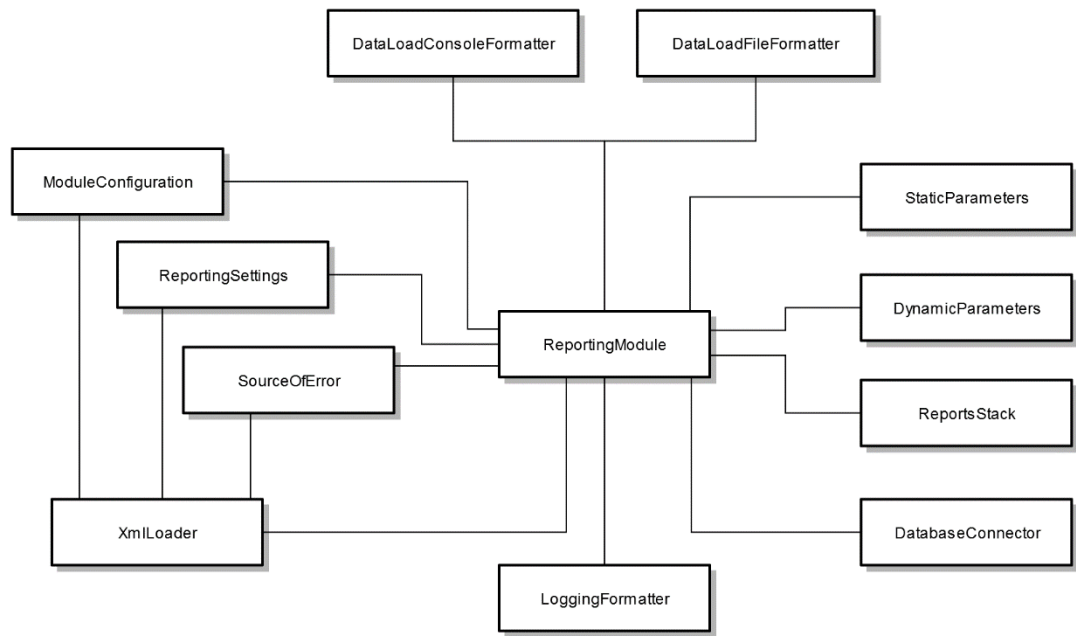


Figure 12. Schema of the module

6.4.2 Connecting components

These components are used to allow the connection of the module to the Data Load utility.

Data Load console formatter

Class *CustomDataLoadConsoleFormatter* can be considered as a connection point for the new reporting module. This class extends the originally used *DataLoadConsoleFormatter*. The purpose of this class is to receive a *LogRecord* object into its *format* method then responsible for formatting the message content into the desired form before it is written into a log file.

There have been no changes to the message formatting, which stayed completely in the competence of the inherited logic from the parent class. The only change was connecting the new module here and passing the processed message there. Eventual error handling logic was also placed here for a case that some problem will occur in the module. In any possible situation, error in the module cannot have any effect on the data loading process.

Data Load file formatter

Some of the exceptions and error messages are written into separate log files stored in a different location than the main data load log file. The responsible class for formatting these messages was originally the default Java *SimpleFormatter* class. The same operations as with *CustomDataLoadConsoleFormatter* were performed here in order to be able to catch and analyze also messages written to these different log files.

6.4.3 Processing components

Following components are responsible for the main activities of the error messages processing.

Reporting module

This is the central component of the whole solution. It controls the whole process and holds the whole module together with references to other components.

Dynamic parameters

Some of the error messages already contain certain important information beneficial to be transported into the final reporting message, which is why the basic process of translation of one message (error message) to another one (reporting message) is extended by functionality provided by component *Dynamic parameters*. This component is responsible for identifying and extracting important information and placing it into a reporting message on specified places and in a specified order.

See Figure 13 for examples of messages, which come into, or are produced by this component. “String to match” specifies positions, where the targeted information is

located. “Report message template” is a template for final reporting message with placeholders for information extracted from the error message. They can be placed anywhere in any order, which is controlled by indexes. “Final report message” is then a final message containing a static explanation of the error with instructions enriched by dynamic parameters for this specific case. Figure 14 shows the processing of these messages.

Original error message

```
The ID was not resolved for the table CATGROUP with the unique  
index data [NonExistingCategory, 20000000000000012345].
```

String to match

```
The ID was not resolved for the table CATGROUP with the unique  
index data \[(.*?)\], (.*?)\]
```

Report message template

```
Unknown category x[0]x was referenced in the data. Please upload  
this category first and then re-upload any data that are  
referencing it.
```

Final report message

```
Unknown category NonExistingCategory was referenced in the data.  
Please upload this category first and then re-upload any data that  
are referencing it.
```

Figure 13. Examples of messages used in Dynamic parameters component.

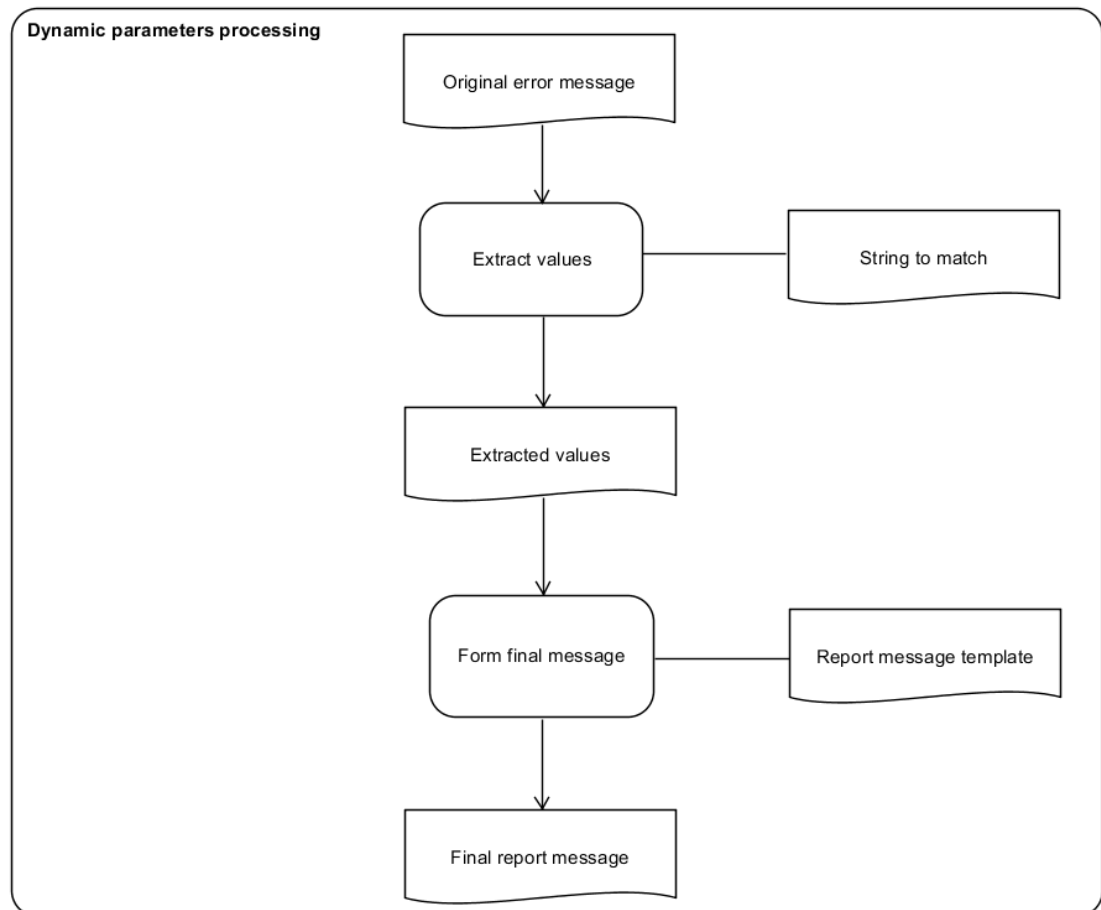


Figure 14. Processing of messages by Dynamic parameters component

The component receives an error message and “String to match”. Based on these two strings, the algorithm extracts information from the error message and provides them to another algorithm placing them into a report message template.

Static parameters

Basic information used for reporting is usually taken only from logging messages written into the log file. Extra information about a failed entity can be connected as well in certain cases. Nevertheless, sometimes it is necessary or beneficial to include it or use also other information. If they do not have relation only to certain cases or items, however, are common, for example for the whole data load process, they can be called “static parameters”.

As an example a store identifier or store name can be named (in WebSphere Commerce Extended sites information model). This information can be loaded only once

and it is the same for all the processes within one data load. The values can then be printed into the final reporting message. Another usage can be to use them in SQL queries for the database, where the identifier of the store might be necessary in *where* clause for restricting the set of results.

The Static parameters component contains the logic for loading the information of a certain static parameter and then also method *replaceStaticParams*, which receives a string with the possible presence of the static parameter's placeholders and replaces them with the expected information. This method can be used also for preparing database SQL queries.

Currently, loading the information strongly depends on the case. For example, the store name is available as a data load process parameter via *DataLoadHelper* class. On the other hand, the store identifier cannot be retrieved from anywhere else than from the database, therefore, it is necessary to perform a database query in order to get this information.

Reports stack

If, for instance, several products have the same problem causing data load error of one type, it would not be very clear and efficient if every occurrence of that error were to be printed separately. It is much better to collect all the information and then print only one report about this error type and simply list there all the products, which had the same problem (see Figure 15). That is the purpose of this component. It stores information about the error and affected items.

```
[ATTRIBUTE] Referenced attribute Brand does not exist in the system, please create or load it.
Occured while loading: attributes of products.
Affected products: ABC001, ABC090, ABC145, ABC347.
```

Figure 15. Error report with list of affected items

6.4.4 Assisting components

Assisting components provide various supporting functions that are needed in the module.

Database connector

This component encapsulates the logic to perform queries to the database.

Logging formatter

This formatter provides options of various text modifications needed in the module.

XmlLoader

This is a loading component responsible for loading the data from XML files. It contains the file paths and file names and the logic for opening and reading files. Also, file-type specific logic is implemented here in order to save the loaded data into correct structures.

6.4.5 Configuration components

The configuration of the module is stored in XML files and during the data loading process it is handled within the following components.

Configuration of reporting module

This component stores settings which can affect the module's functionality, a way of working, etc. If it is possible and useful, all the values and configurations are preferably stored in this component rather than in the source code. Changing of any value then does not require any code change or a more complicated or long deploying process.

```

<ModuleConfiguration>
  <debug>true</debug>
  <settingsExpiration>2</settingsExpiration>
  <showErrorLogFilePath>true</showErrorLogFilePath>
</ModuleConfiguration>

```

Figure 16. Configuration of the module in the XML file

An example (visible on Figure 16) of such a setting can be a *debug switch*, which controls whether debugging messages should be written into the log. This can be useful during development and troubleshooting. Also, time expiration of the settings of the known errors is stored here and can be changed very quickly at almost any time.

Reporting settings

All the known errors, which should be somehow automatically processed by the module are configured in the XML file. A part of the configuration file can be seen in Figure 17. This component stores all the information.

Class *ErrorItem* is used to represent a single item of the module's reporting settings. It is constructed from the data contained within the *Item* tag in the settings. It contains an error message by default provided by the Data Load utility. A report message is an explanation of the error providing easier understanding and eventually also guidance what to do in order to fix the problem. The tag serves only for marking the final message. If it can help to provide important information, a database query can be configured. If there are multiple stores on the same platform and it is required to target this error handling configuration only for the specific one, it can be specified in property *stores*. Finally, it can be configured if this message were to be printed into an e-mail or only to the log file.

```

<Item>
  <searchString>An attribute with the specified name (.*) does not exist in the system.</searchString>
  <reportMessage>Referenced attribute x[0]x does not exist in the system, please create or load it.</reportMessage>
  <tag>ATTRIBUTE</tag>
  <dbQuery></dbQuery>
  <stores></stores>
  <putIntoMail>true</putIntoMail>
</Item>
<Item>
  <searchString>The ID was not resolved for the table CATGROUP with the unique index data \[(.*)\], (.*)\]</searchString>
  <reportMessage>Unknown category x[0]x was referenced in the data. Please upload this category first and
    then re-upload any data that is referencing it.</reportMessage>
  <tag>CATEGORY</tag>
  <dbQuery></dbQuery>
  <stores>All</stores>
  <putIntoMail></putIntoMail>
</Item>

```

Figure 17. Configuration of known errors, reporting messages and other related settings for the specific error

Sources of errors

In some cases, default errors describe the certain problem but do not mention the entity where problematic or corrupted data is present. Of course, this applies only to data load errors caused by some problem in loaded data. In that case, it is not immediately clear where to look for the cause of the data load error. Information about which entity was just being loaded when the error happened can be useful because after that it is easier to identify the source data file, where the problem can be present.

This component was developed with the purpose of identifying and providing information about which entity was loaded when the error occurred. The identifying source is based on the manually set relation between class names occurring in the exception's stack traces. Basically, it is a process of identifying a pre-set string in the message being written into a log file and providing information to which this string is mapped. This key-value pair mapping is configured in the XML file and then loaded and stored in this component.

```

[CATEGORY] Unknown category NonExistingCategory was referenced in the data. Please upload
this category first and then re-upload any data that are referencing it.

```

Figure 18. Error where the source of the error is not clear

An example of the benefit of this component can be seen in Figure 18. The reference to a non-existing category can come from the product (category, where the product belongs) or another category (a reference to a parent category). If information about the source is provided, it is clear which source data file to check.

6.4.6 Configuration testing tool

It is expected that the number of errors the module is able to recognize and process will grow every time when some new error occurs. In that case, it will be necessary to add a new item to the module's errors configuration. As there can be some special characters, which need to be handled (in order to prevent their standard interpretation) or the dynamic parameters can be used, it can be quite unsure if the configuration will work as expected. For this purpose, a simple testing tool was developed. It is enough to configure the settings there and provide a message from a log file.

7 Results

7.1 Current status and results

The module's development phase is now finished. The module is capable of automated processing based on configuration and provides reporting information about data load errors which occurred. The output of the module in case of data load with errors can be seen in Figure 19.

```
[ATTRIBUTE] Referenced attribute Brand does not exist in the system, please create or load it.
Occured while loading: attributes of products.
Affected products: ABC001, ABC090, ABC145, ABC347.
Error log location: /Commerce/logs/ATTRIBUTES_ERROR_2016.11.03_13.03.30.290.log.

[CATEGORY] Unknown category CCC010 was referenced in the data. Please upload this category first and then
re-upload any data that is referencing it.
Occured while loading: products.
Affected products: ABC157.
Error log location: /Commerce/logs/PRODUCTS_ERROR_2016.11.03_13.03.30.79.log.
```

Figure 19. Information from reporting e-mail

These example reports immediately provide necessary information about the error. All the texts are configurable and the most important thing is the data. Each of the reports contains tag which provides first brief information about the error. It also helps to order the reports. There is a configurable explanation to that specific error, which should tell the reader what exactly happened, why and what can be done to solve the issue. Then there is an information about where it happened – which entity was being loaded. In certain cases, also precise item identifiers or names are provided. The combination of all this information provides everything necessary to understand the problem, locate it in the data and fix it. If more information is needed especially for support team members, they can easily find the log file as well.

7.2 Putting into use

The testing phase recently started, which means that the module will be tested by more people with much bigger and different sets of data. The main objectives of the testing phase are to confirm expected behavior during different situations, correct handling of error states, identify possible performance issues, etc. Also, the error configuration will be built to recognize different types of errors. It is expected that there will be some new requirements or change requests during the testing period. Certain problems with the solution can be also discovered. After everything has been tested and is working as expected, the real usage can start.

8 Conclusion

The original objectives were successfully met. The Data Load utility was studied and explored, and options for improvements were identified, designed and implemented. The developed module is capable of automated processing of the data load errors and presenting them in an understandable way enriched by additional data to allow the fast solving of the error causes. The module was welcomed by the company representatives as it solves a real issue which has been existing for a longer period of time. It successfully reduces the time and cost requirements of the solving process of the data load errors.

The process leading to the results comprised the research of the Data Load utility, analysis, design of the intended improvements and their implementation.

Communication with the stakeholders in a form of consultations, presentations and feedback gathering was also necessary. The combination of all of these was a very interesting professional experience.

There were many obstacles and limitations during all the phases. The main problems were that there was not enough information available about certain parts of the Data Load utility, and it was also impossible to modify many of its parts due to technical and license restrictions. The faced issues brought a constant risk of failure as at almost any point it could have happened that it would simply not be possible to achieve the specified goals due to some newly discovered issue or limitation.

The main functionality is now finished, however, as mentioned in the previous chapter, certain requests can still occur during testing. Also, some other functionality can be added to the module later on when some new needs will be discovered.

The whole thesis resulted in a solution which is beneficial for the company and meets a real need. From the personal point of view, challenges which came especially from the faced issues offered a big opportunity to learn and develop useful skills. I have already used the gained knowledge about the Data Load utility for tasks not related to the thesis.

References

Data Load utility architectural overview. Page on IBM Knowledge Center. Accessed on 27.09.2016. Retrieved from http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.data.doc/concepts/cmldataloadovdev.htm

Data Load Utility in WebSphere Commerce Introduction. Video on Youtube.com. Accessed on 12.10.2016. Retrieved from <https://www.youtube.com/watch?v=jCVOwqH0Rhw>

Exploratory Programming. Exploratory Programming with Collaborative Programming Languages Accessed on 05.10.2016. Retrieved from <http://steak.place.org/dougo/thesis/plan/>

Overview of the Data Load utility. Page on IBM Knowledge Center. Accessed on 27.09.2016. Retrieved from http://www.ibm.com/support/knowledgecenter/SSZLC2_8.0.0/com.ibm.commerce.data.doc/concepts/cmlbatchoverview.htm

WebSphere Commerce Enterprise. Page on IBM website. Accessed on 29.09.2016. Retrieved from <http://www-03.ibm.com/software/products/fi/websphere-commerce-enterprise>

What Is E-Commerce. Business News Daily. Accessed on 12.10.2016. Retrieved from <http://www.businessnewsdaily.com/4872-what-is-e-commerce.html>