A Data Decomposition Method for Stepwise Migration of Complex Legacy Data

Andreas Martens
adesso AG
Hamburg, Germany
andreas.martens@adesso.de

Matthias Book
University of Iceland
Reykjavík, Iceland
book@hi.is

Volker Gruhn
paluno – The Ruhr Institute for Software Technology,
University of Duisburg-Essen
Essen, Germany
volker.gruhn@paluno.uni-due.de

ABSTRACT

Sooner or later, in almost every company, the maintenance and further development of large enterprise IT applications reaches its limit. From the point of view of cost as well as technical capability, legacy applications must eventually be replaced by new enterprise IT applications. Data migration is an inevitable part of making this switch. While different data migration strategies can be applied, incremental data migration is one of the most popular strategies, due to its low level of risk: The entire data volume is split into several data tranches, which are then migrated in individual migration steps. The key to a successful migration is the strategy for decomposing the data into suitable tranches.

This paper presents an approach for data decomposition where the entire data volume of a monolithic enterprise IT application is split into independent data migration tranches. Each tranche comprises the data to be migrated in one migration step, which is usually executed during the application's downtime window. Unlike other approaches, which describe data migration in a highly abstract way, we propose specific heuristics for data decomposition into independent data packages (tranches).

The data migration approach described here is being applied in one of the largest migration projects currently underway in the European healthcare sector, comprising millions of customer records.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; Software system models; • **Information systems** → *Extraction, transformation and loading*;

KEYWORDS

incremental data migration, data decomposition, data migration tranches, units of migration

ICSE-SEIP '18, May 27-June 3 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5659-6/18/05…$15.00
https://doi.org/10.1145/3183519.3183520

ACM Reference Format:
Andreas Martens, Matthias Book, and Volker Gruhn. 2018. A Data Decomposition Method for Stepwise Migration of Complex Legacy Data. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183520

1 INTRODUCTION

Many companies have long been using enterprise IT applications to support important business processes, and even more importantly, to store business data. However, companies are occasionally forced to switch to a new enterprise IT application, for example due to obsolete technologies, pursuit of digital transformation, corporate mergers and acquisitions, or for other reasons.

There are a number of approaches for migrating from the legacy application to the new application, each of which entails a certain delay, migration time, risk and effort. The fastest and riskiest of these migration approaches are known as "Big Bang" or "Cold Turkey" approaches [5, 12], followed by incremental package replacement (which involves gradually implementing packages in the new application). The slowest and least risky option is long-time or maintenance-driven migration [15].

The most complex aspect of the overall migration is often the migration of the business data from one system to the other. Data migration is defined in [8] as a "*tool-supported one-time process which aims at migrating formatted data from a source structure to a target data structure whereas both structures differ on a conceptual and/or physical level.*"

There are three basic requirements for every data migration project:

- The ongoing business processes should not be interrupted.
- Data consistency should be maintained and data should not be lost.
- The effort and cost of the data migration should be minimized.

The key challenges in data migration typically include varying data models, poor data quality in the legacy system and strong restrictions of the target system. If a large volume of data is to be migrated, the system engineers are forced to decompose the data into independent tranches due to the limited system downtime available for migration. The combination of this required data decomposition and complex dependencies within the data structure represent another key challenge.

In this paper, we present a new approach to handling the data decomposition challenge. Our approach allows engineers in an

incremental data migration project to flexibly split the entire data volume into independent Data Migration Tranches, the unit sizes of which can be adapted to the available system downtime slots. These Data Migration Tranches can then be migrated step-by-step, independently of one another, into the target system. This reduces the risk and effort required for system integration while increasing flexibility and control over the process. In contrast to other data migration approaches, which describe data migration on a very high abstraction level, we suggest concrete data decomposition heuristics, placing particular focus on building independent Data Migration Tranches.

The main concept underlying our approach is building Data Migration Tranches consisting of independent Business Object Networks. Each Business Object Network consists of precisely one Key Business Object and a set of connected Business Objects. Key Business Objects are instances of the Key Business Class (such as "Customer"), and the linchpins of our data decomposition approach.

In Sect. 2, we first outline how to identify the Key Business Class and independent Business Object Networks. In Sect. 3, we then describe how to effectively assign identified Business Object Networks to Data Migration Tranches. Section 4 then describes our experiences with applying this approach in a long-term data migration project currently underway at a large health insurance company in Europe, and includes some lessons we have learned from this experience. We close with an overview of related work (Sect. 5) and an outlook on further work (Sect. 6).

2 STRUCTURING THE DATA

In large data migration projects, the project's IT and business experts need to migrate thousands or millions of records from the legacy to the target system. Each record consists of several fields with data in different formats. The relationships between the records are often very complex and not immediately obvious. Because of this, simply splitting data along database tables or records is not optimal. Our data decomposition approach instead involves splitting data along Business Object Networks, as described in the following subsections.

2.1 Business Classes

A **Business Class** (BC), or an entity in the context of entity-relationship models, is an abstract template for identical **Business Objects** (BOs). This template describes the object structure and attributes that are significant in the business domain and linked to the business logic. A Business Class can cover one or several database tables. Examples of Business Classes include "Customer", "Order", and "Product". These Business Classes would in turn correspond to Business Objects such as "Max Smith", "Order No. 45721", and "Harry Potter and the Cursed Child", respectively. To apply our decomposition approach, the enterprise IT application's entire data model must be conceptually mapped to Business Classes.

As illustrated in the example above, many Business Objects are related and can not be considered outside of their context. For the purposes of our approach, we will call a group of related Business Objects a Business Object Network, if it satisfies the following definition: A **Business Object Network** (BON) is a group of connected

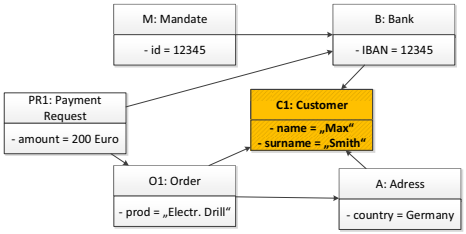

Figure 1: Example of a Business Object Network

Business Objects related to exactly one common Key Business Object, which also belongs to the group. This means that Business Objects within a Business Object Network have strong dependencies on each other (and in particular on the Key Business Object), while dependencies on objects outside the Business Object Network are much weaker. The dependencies are manifested by references between the objects (which gives BONs a more concrete structural foundation than Fowler and Sadalage's concept of "Aggregates" in NoSQL databases [13], defined more arbitrarily as *"collections of related objects that we wish to treat as a unit."*). As an example, Fig. 1 shows a simple Business Object Network around an instance of a "Customer" Key Business Class. In the following subsections, we discuss concrete criteria for choosing the Key Business Class and for determining the boundaries of Business Object Networks in more details.

2.2 Key Business Class

A Key Business Object is an instance of the **Key Business Class** (KBC), which is designated once for the entire enterprise application to be migrated. From a business perspective, the Key Business Class should represent high-priority Business Objects for the company, such as "Customer". In other words, the **Key Business Objects** (KBOs) should be at the focus of the company's main business processes. These objects will typically have large related BONs. A well-chosen KBC yields the following main benefits:

- **Workload balancing**: The amount of migrated BONs is proportional to the workload shifted from the source to the target system. As each BON is migrated, more workload is shifted to the new system.
- **Integration**: The number of Business Objects covered by BONs will be maximized. None of these Business Objects will require additional synchronization between source and target system after migration, or produce additional integration overhead.

From a conceptual perspective, the Key Business Class should be a "top" class without any outgoing references (but many inbound references from other Business Classes); for example, a booking would reference a customer, but not vice versa. Usually, there also should not be any relationships between individual Key Business Objects; for example, different customers are usually not related in the system (if some KBOs are however related, we treat them as described in Sect. 2.4). These structural properties yield the following benefits:

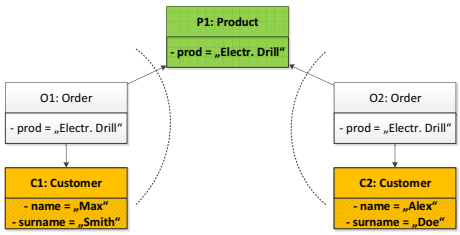

Figure 2: Example: BONs of two Customer instances

- **Data completeness**: The risk of forgetting some important objects during migration is very low, since we are migrating the Key Business Objects and any objects they depend on.
- **BON independence**: Independence of the Key Business Objects results in independent BONs, which can be migrated independently of each other.

2.3 Bounds of BONs

Establishing independent Business Object Networks means that each BON should have its own clear boundaries. In most cases, though, companies will not be able to identify completely independent BONs in the data by tracing relationships from the Key Business Object to other Business Objects. In other words, the boundaries of Business Object Networks can usually not be identified by simply making a "deep copy" of all the objects that are directly and indirectly related to a particular Key Business Object. In the worst-case scenario, all Business Objects in the application might actually be indirectly referencing each other.

Instead, our approach defines the boundaries of a Business Object Network as follows: The Business Object Network of the Key Business Object K contains all of the application's Business Objects that are directly or indirectly connected *exclusively* to this Key Business Object K. This means that all Business Objects that are directly or indirectly related to several KBOs (e.g., Business Object P1 in Fig. 2) should not be considered as part of the BON.

We call the data in all of the Business Object Networks formed this way **Primary Data**. Primary Data therefore consists of instances of the Key Business Class and instances of directly and indirectly exclusively related classes, which are called **Primary Business Classes**. All other Business Objects outside of the BONs are either instances of so-called **Secondary Business Classes** or **Configuration Classes**. Secondary Business Objects usually include widely known, relatively static business data that is not KBC-specific, such as (in the healthcare domain) agencies, providers, or suppliers. The configuration data comprises Business Classes describing business (e.g. "Products") and system configuration aspects (e.g. "User Roles"). The generic name for all of these Business Objects is **Secondary Data** (Fig. 3).

The example in Fig. 2 shall help to clarify this definition: Two customers (Max and Alex) each place an order (O1 and O2) for the same product (P1). Although an "Order" Business Object can only be connected to exactly one "Customer" Key Business Object, the ordered Product can be connected to more than one Order and, consequently, to more than one Key Business Object. In this example, the Order belongs to the same Business Object Network as the Customer (it is Primary Data), whereas the Product does not (it is Secondary Data).

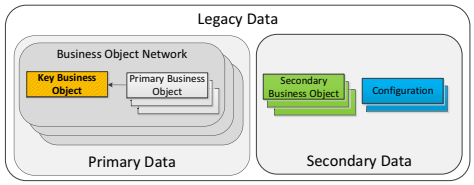

Figure 3: Primary and Secondary Data

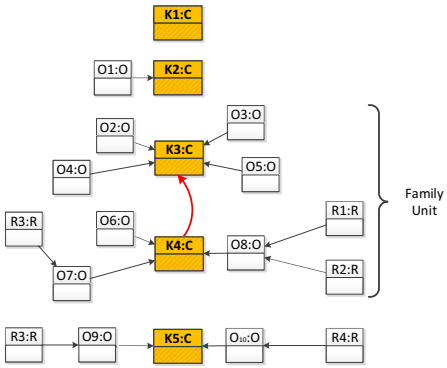

Figure 4: Dependencies between Key Business Objects

2.4 Family Units

In some data migration projects, the KBOs themselves can be directly connected to each other, causing their BONs to overlap. Since KBOs cannot be treated as Secondary Data, the related KBOs are instead grouped into a **Family Unit**. The BONs of that Family Unit will then be considered as one large BON in the next steps. (Note that this approach only works if the number of KBOs in a Family Unit is limited, e.g. if we are indeed dealing with a few persons who are members of the same family. Our approach would not work in a system like a social network where large numbers of KBOs can be related to each other.)

In the example shown in Fig. 4, the Business Object Network of the Key Business Object K3, which is related to the KBO K4, will be joined with the BON of the KBO K4 to form a new large Family Unit BON to be used in further steps of the data migration process.

2.5 Properties of Primary and Secondary Data

Primary and Secondary Data serves different purposes in an application, and will therefore be treated significantly differently during the migration. Primary Data represents the largest portion of the data migrated in a data migration project. Primary Data also has high requirements in terms of data quality. The company should analyze the criticality of each error that occurs during test migrations, as any unexpected and critical errors that occur during the

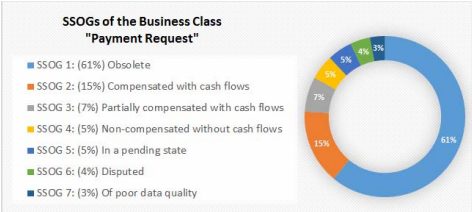

Figure 5: SSOGs of the Business Class "Payment Request" with corresponding volumes

real migration will cause the migration of the entire affected Business Object Network to be rejected. The tolerance for invalid data states, which would be fixed during the post-migration phase using additional scripts, is very slim.

In contrast, Secondary Data is data that will usually be required for processing Primary Data in certain business processes. The wrong migration of Secondary Data is less critical for the project. Some of this data comes from external sources, which means that it can be reconstructed much more easily in the event of failure, and it can be imported into the new system via defined interfaces at comparatively lower effort. For Configuration Classes, whose numbers of Business Objects are typically small, manually correcting data in the event of a failure may even be an acceptable solution. However, the migration of Secondary Data is a precondition for the migration of Primary Data. In practice, a lower quality of migrated Secondary Data is therefore sometimes tolerated, as it would otherwise not be possible to migrate large amounts of dependent (but more critical) Primary Data at all.

2.6 Distinguishing Same-State Object Groups

Depending on their state, different Business Objects instantiating a single Business Class may need to be migrated in different ways, or may even need to be excluded from the migration (e.g. submitted, payed and canceled orders are all instances of the Business Class "Order", but they are in different states and must be therefore be treated differently during the migration). We therefore distinguish Business Objects that instantiate the same Business Class but are in different states, and group them into so-called Same-State Object Groups. A **Same-State Object Group** (SSOG) is a set of Business Objects instantiating the same Business Class, being in a similar business state, and needing to be migrated in a similar way. The rules for data selection, transformation, and data loading are identical for all Business Objects in the same SSOG. This also means that exactly one ETL program[2] [19] must be designed and implemented for each identified SSOG. This definition of a SSOG is applicable to Business Objects both in Primary and Secondary Data.

The starting points for the identification of SSOGs are the Business Classes. Each Business Class should be analyzed separately. One of the most important aspects that should be taken into account when defining Same-State Object Groups is the time range of business validity. In the example of our health insurance migration project (Sect. 4), the project stakeholders had to define the cut-off point for past payment requests that were processed many years ago (SSOG 1 in Fig. 5). They are no longer relevant for the business and do not need to be processed further. Such Business Objects should be excluded from the related BONs. In most legacy systems, these kinds of Business Objects represent the biggest portion of Business Objects of a Business Class. They can typically be archived with minimum effort in external archiving systems for information and revision purposes, but do not need to be migrated to the target system. The next three SSOGs in Fig. 5 represent three groups of Business Objects that differ widely in terms of business and data structure:

SSOG 2 represents compensated payment requests with cash flows. These are Business Objects in final workflow states that are usually no longer required for future business processes and closed from a business perspective, except in the event of retroactive corrections. These kinds of payment requests can be generated in the target system using its own program logic according to the combination of migrated input data and the system configuration, and are not migrated via more expensive ETL programs. (Designing and implementing ETL programs only makes sense if the volume of data intended to be migrated with the program is large enough and there are no cheaper alternatives.)

The partially compensated (SSOG 3) and non-compensated payment requests (SSOG 4) are not yet in a final business state. They need to be migrated into the target system (using appropriate ETL programs) so that the payment process can be completed. Unlike non-compensated payment requests, partially compensated payment requests already contain cash flows. These Business Objects therefore differ in structure, and need to be migrated with different ETL programs. This justifies the distinction between SSOG 3 and SSOG 4.

The next three SSOGs represent rather small groups of payment requests in exceptional states. However, these groups should also be taken into consideration to define the processing methods for the corresponding Business Objects.

All payment requests in exceptional and temporarily pending states, such as payment requests that need to be approved by company supervisors in a separate business process step, should be grouped into their own SSOG (SSOG 5 in the project in Fig. 5). Business Object Networks with these kinds of BOs should be excluded from the migration until the pending cases are transformed into a stable workflow state. This saves the team the effort of unnecessarily developing special ETL programs for exotic cases.

Similarly, payment requests in complex workflow states that require long-term additional clarification should get their own SSOG, such as SSOG 6 in our project. An example of such a payment request would be one that was not accepted by the counterpart and for which the corresponding legal ruling is still pending. If there are very few BOs of this type, this SSOG would be a candidate for a manual data transfer, as developing a suitable ETL program would involve more effort than a manual data transfer. Manual data transfer should also be considered in certain other cases, e.g. when equivalents of the data required in the target system do not exist in the source system (this kind of data can only be identified from the target system by following the direct and indirect object references [6, 7]).

[2]The migration programs are called Extract-Transform-Load (ETL) programs in data migration projects. These are routines for selecting, transforming, and storing data.

The final SSOG to consider (SSOG 7 in our project) represents Business Objects of poor data quality. They are illegal from a technical or business point of view. To enforce high data quality, the target system will not be able to process these kinds of Business Objects after migration. BONs with these kinds of BOs cannot be migrated into the target system until the BOs have been manually corrected in the source system.

The payment requests in our health insurance system could have been split into further SSOGs. Alongside the seven SSOGs described above, the payment requests could also be considered from the perspective of other Business Classes. The payment requests could also be split into SSOGs according to their affiliations with certain divisions of the company, according to different product or service types, or according to customer types (new or returning). Which criteria a team uses to define SSOGs is highly project-dependent, but the motivation is always to group BOs that will require similar treatment through suitable ETL programs, manual migration or other processing steps.

Splitting all the Business Objects of one Business Class into several separate SSOGs is an important step in our approach. For one thing, building small but similar groups of BOs will make it easier to handle complex Business Classes. For another, the distinction between SSOGs also enables companies to clearly define the ETL programs and the migration processes for each SSOG. Different SSOGs belonging to the same Business Class can be migrated in different Data Migration Tranches, which means the contents of DMTs can now be defined on the level of SSOGs.

Doing so can reduce the effort required to migrate Business Objects in many different situations. It also expands the definition of the term "data migration" beyond meaning copying data from the source to the target system. As shown above, the data in a data migration project can also be archived, generated, manually entered, or imported into the target system.

2.7 Heterogeneity of a BON

The Business Object Networks defined by our approach can obviously have quite different structures: While some Key Business Objects are not connected to any other Business Objects, such as customers for whom only core data is available, another Key Business Object's Business Object Network might contain hundreds of Business Objects that are instances of different Business Classes and belong to different SSOGs. This means the BONs differ in terms of their heterogeneity. The **Heterogeneity** of a Business Object Network (defined as a function het(KBO of the BON) = #SSOGs) refers to the variety of SSOGs represented in that Business Object Network. For instance, a het(KBO) will increase by one if one new Business Object of a SSOG not already included in the BON is included in the Business Object Network. The number of Business Objects belonging to a particular SSOG is not relevant for this definition.

Due to the incremental nature of the migration, the heterogeneity and volume of a BON may change over time, as users continue to work with the legacy system. The BON can expand at any time (through the addition of new Business Objects of already included SSOGs) and become more complex by increasing its heterogeneity

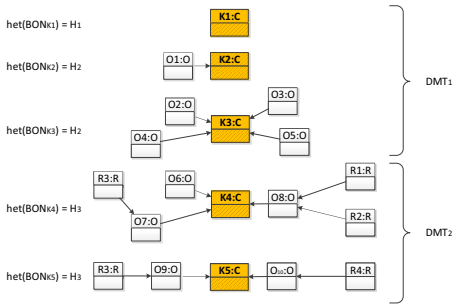

Figure 6: Three Heterogeneity stages and assigned DMTs

(through the addition of new Business Objects of SSOGs that are not yet included).

The Business Object Network with the Key Business Object K1 in Fig. 6, for example, has het(K1) = 1. It will move to the next heterogeneity level 2, if a new SSOG is added to the network. The Business Object Networks with Key Business Objects K2 and K3, respectively, have the same heterogeneity: het(K2) = het(K3). Each of these networks only contains Business Objects from the same two SSOGs. The same is true for the even more heterogeneous BONs of K4 and K5.

Defining this property of BONs mainly serves to enable companies to compare and distinguish between "simple" and "complex" BONs. Simple BONs should be migrated first, followed by complex BONs; potentially in different data migration tranches.

3 MIGRATING THE DATA

The Business Object Networks described in the previous section can already be migrated into the target system independently of each other. Due to the high organization overhead incurred by the migration process, it however makes sense to bundle several BONs into much larger **Data Migration Tranches** (DMTs).

The following subsections describe how to build DMTs that can be migrated into the target system within a single data migration step, and how to migrate Secondary Data.

3.1 Combining and migrating BONs

Data Migration Tranches consist of groups of complete Business Object Networks, which means they consist entirely of Primary Data. Although the number of Business Object Networks in a single Data Migration Tranche can be selected arbitrarily, this depends on the conditions of the data migration project and the conditions of the related data migration step. At one end of the spectrum are Data Migration Tranches that contain only one Business Object Network. These might result in a unjustifiably high number of data migration steps and correspondingly high organizational overhead compared to the value added by the migration. At the other end of the spectrum are single DMTs that contain all of the system's existing Business Object Networks. When choosing this Big Bang approach, companies need to consider all of the known disadvantages and problems it entails. The benefit of an incremental

approach, in contrast, is that the volume of each tranche can be adapted to the planned downtime windows of the application. Annual financial statements, inventory creation, and major company events can also affect scheduling.

In order to build DMTs, all of the identified Business Object Networks should be sorted according to their heterogeneity, in ascending order. Following this sorting, the top layer will contain simple BONs with the lowest number of represented SSOGs. Meanwhile, the bottom layer will contain the most complex BONs with the highest number of represented SSOGs. This ordering provides a good basis for building independent data tranches. The migration engineers can group Business Object Networks into tranches virtually as desired, since they are highly independent.

There are two reasons for sorting BONs in this way. Firstly, grouping Business Objects Networks with greatly differing heterogeneity into one data migration tranche would lead to an unpredictable number of data sets in this tranche. Grouping BONs with the same or similar heterogeneity, in contrast, allows the migration engineers to control the volume of data in the tranche. When doing so, it is crucial that the heterogeneity of each BON is known. The first tranche can contain a high number of simple BONs, and the last tranche can contain a low number of complex BONs with the highest heterogeneity.

Moreover, in the absence of sorting, the whole set of ETL programs required for the migration must be clarified, developed, and tested for all SSOGs before the first migration step is triggered. This also means that the largest portion of the artifacts resulting from the data migration project must be created before the first migration step begins. Only the data migration itself will be carried out iteratively. In contrast, sorting ahead of time allows project members to develop only those ETL programs that need to be applied to the SSOGs of the currently migrated data tranche. This means that from the software development perspective, a DMT can be defined as the set of SSOGs for which ETL programs have to be implemented for the upcoming migration step.

Each Business Object of the migrated BON, meaning the Key Business Objects with all related Business Objects of the Primary Data, must be immediately locked for write access in the legacy system after migration. Alternatively, they can be moved into an external data warehouse. Manipulation of the migrated Business Objects in the legacy system must be prohibited. From this point, only the target system is responsible for processing the migrated Primary Data. That means that all outgoing messages for the migrated KBOs will be generated in the target system and all incoming messages related to the migrated KBOs should be routed to the target system.

3.2 Initial migration of Secondary Data

So far, our discussion has focused on Primary Data. As mentioned in Sect. 2.3, however, most Business Object Networks have dependencies with Business Objects of Secondary Data. These dependencies need be severed during the data migration; this way, the Business Object Networks will be excised from the "background" of the Secondary Data (Fig. 7) in the legacy system. During the migration, the Business Object Networks will be copied to the target application and connected to the respective Secondary Data objects already

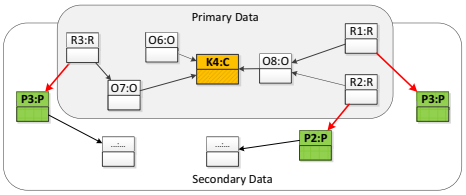

Figure 7: Relations between Primary and Secondary Data

present in the target system. To allow this, all required Secondary Data need to be migrated to the new system before the migration of Primary Data begins.

This approach requires the initial migration of all Secondary Data, even if the first Data Migration Tranche only contains simple Business Object Networks. These simple BONs in the Data Migration Tranche can immediately start expanding after the migration to become large and complex BONs with hundreds of new SSOGs, as described in section 2.7. This also means that the target system must be completely developed and configured during the initial phase. For example, all user profiles, infrastructure elements etc. must be set up in advance. To refer to our earlier example with Customer and Order classes (Primary Data), all orderable Products (Secondary Data) must already exist in the target system.

It should be emphasized that both systems remain in operation during the entire data migration project. After the first data migration step, some BONs that will be processed in the target system and some BONs from the legacy system will require common Secondary Data. This is why this kind of data must be simultaneously present and maintained in both systems. The Business Objects of the Secondary Data must be handled in a completely different way to the Primary Data.

The Business Objects of the Secondary Data should also always be in sync in both systems. This requires additional synchronization or system integration mechanisms. The synchronization overhead for the Secondary Data in the source and target systems remains comparatively small, however. This is mainly because:

- The number of SSOGs of Secondary Data is much lower than the number of SSOGs of Primary Data.
- The number of Secondary Business Objects represents a mere fraction of the total volume of data, which can comprise millions of records. The Secondary Data, in contrast, is usually not mass data.
- Moreover, unlike with Primary Data, the number of write accesses to Secondary Data is negligible. Each day a large number of Primary Business Objects such as orders will be collected in the system, but only some new product descriptions will be collected or updated in the system each week.
- Some more volatile Secondary Data, such as exchange rates, should be regularly imported into the system via defined system interfaces. No additional synchronization mechanisms will be required to do so. The data will be updated within the scope of daily business processes.
- Some configuration data, such as user roles, is often held in external systems such as Active Directory. When this is the

case, both the source and target systems need access to the common data pool.

It also should be noted that some Secondary Data required in the target system may not have any counterpart in the legacy system. It therefore needs to be incorporated manually in the target system but does not need any migration logic.

4 PROJECT EXPERIENCE

The data decomposition and migration approach outlined in the preceding sections was and is being applied in an ongoing data migration project carried out within one of the largest IT projects in the European healthcare sector, proceeding over the course of five years. The following subsections describe the (anonymized) project setup, details about the data decomposition, and our experiences with the approach.

4.1 General Conditions

A large health insurance company has many millions of customers who are served by a large number of regional offices around the country. In an iterative data migration project, a monolithic legacy system that had been developed in COBOL many years ago was to be replaced by a state-of-the-art application written in Java. The target system's database schema consists of over 5,000 DB2 database tables. The legacy system has a hierarchical database structure with several terabytes of data that are relevant for migration.

4.2 Execution of Migration

The data migration project is being executed in the context of a company merger. For this reason, the core system of one of the merging companies, which was selected as a new target system, already contained business data. Consequently, some of the Secondary Data was already present in the target system, and other Secondary Data was migrated with ETL programs, entered manually, or imported via system interfaces.

For the Primary Data we defined three Data Migration Tranches, each separated by about one year, for this data migration project. The main purpose of this split and the long timeframe was to build manageable Data Migration Tranches with acceptable downtime windows, risks and overhead. Scheduling the DMTs at one-year intervals was necessary to have sufficient time to design and implement the ETL programs required for the next Data Migration Tranche. We structured the DMTs in this way because, due to strict business targets, the data migration could only be carried out during narrow windows of time over the course of the calendar year.

The "Customer" Business Class was selected as the Key Business Class for this project. This seemed natural from the insurance company's business perspective, and the customers usually belonged to very broad Business Object Networks that covered a large part of the existing Business Classes in the target system. However, the KBOs (and thus the BONs) were not completely independent because the company's customers could, e.g. in the case of family members, have dependencies on each other. We therefore applied the Family Unit concept (Sect. 2.4). The average family sizes were negligible (in most cases, two to five customers), and there were no dependencies between families. Therefore, it was possible to define small Family Units and join related BONs.

For this project, the joined Business Object Networks were assigned to Data Migration Tranches based on the highest heterogeneity of all of the Business Object Networks belonging to each Family Unit. Otherwise, the customers of one family would have been distributed across different migration tranches and thus both systems, which would have prevented execution of business processes involving two customers from the same family.

The distinction between Primary and Secondary Data (Sect. 2.3) was particularly useful in the migration of the company's CRM data: Since many customers were members of the company's bonus scheme, the payment requests of many customers referred to the same bonus scheme objects. Without the distinction between Primary and Secondary Business Objects, this would have created overlaps in the BONs of many thousands of customers, preventing their independent migration. By designating the bonus scheme objects as Secondary Data, however, all customers' BONs could be kept independently migratable from each other.

For the migration of the bonus payment requests, the concept of Same-State Object Groups (SSOGs, Sect. 2.6) proved useful as well: Originally, it had seemed that the ETL programs for migrating these objects would be too complex to tackle in the first migration tranche, which would have delayed the migration of these payment requests (and more critically, the associated Customer objects) by at least a year. Closer analysis of the bonus payment request objects revealed however that most of them were already in a "completely compensated" state that could be quite easily replicated in the target system. Much fewer payment requests were in a "non-compensated" or "partially compensated" state that would require a more complex ETL program for migration. By defining separate SSOGs for the completely, partially and non-compensated payment requests, we were able to address the bulk of these (the completed payment requests) already in the first DMT, thereby preventing a delayed migration of the respective customer BONs. The SSOGs for the more complex cases were then addressed in the second DMT a year later, when the necessary ETL programs had been developed.

While both the legacy and the target system are running in parallel during the multi-year migration, access to both systems is managed on two different levels: At the user interface level, an additional dialog was added to route users to the legacy or target system, depending on the entered customer identifier. At the level of the electronic data interchange, a data router was inserted to route incoming messages from external systems to the legacy or target system as needed. Since our migration approach ensures that customers are only migrated together with their complete Business Object Network, we are able to use this quite simple high-level routing, as only one system will ever be needed to execute any business process affecting any particular customer.

4.3 Results of Migration

The first DMT, migrated in 2016, comprised about one million Key Business Objects (i.e. 20 percent of all Key Business Objects) and related Business Object Networks. The second DMT, migrated in 2017, comprised the majority of the Key Business Objects (about three million KBOs) with their associated BONs. The last DMT is planned to be a comparatively smaller DMT containing the most

complex and exceptional BONs, which are scheduled for migration in 2018.

At the time of this writing, the first two Data Migration Tranches have successfully been migrated into the target system. Using the described approach, we were able to make optimal use of the given small downtime windows, since we had precise control of the volume, complexity and dependencies of the data that would be migrated in each step. Having DMTs of precisely defined volume and complexity helped to reduce uncertainty and thus risk in the migration project, as the team knew what challenges to expect in each tranche. Risk was also reduced by the fact that the development and testing efforts are distributed over the entire project time of several years, rather than being bunched up as they would have been in a Big Bang approach.

The additional effort incurred by following this approach in our project was quite light. The identification of Business Classes was necessary anyway to define mapping rules between the systems. The effort for identifying the Key Business Class, defining BONs and grouping them into DMTs was minor. The effort for defining each Business Class' SSOGs was part of the development effort of the ETL programs, which would have to be implemented anyway. The most time-consuming part of the process (claiming a few days of intensive work) consisted of distinguishing between Primary and Secondary Data. In addition, determining how to handle Family Units required some workshops between the migration engineers and domain stakeholders.

4.4 Recommendations

Based on our experience with the aforementioned large-scale migration project, we can derive the following recommendations for the definition of Data Migration Tranches:

Each Data Migration Tranche should contain the maximum amount of data that can be migrated in the planned application downtime window. However, additional buffer time should be scheduled for unexpected events that arise during the migration, such as organizational issues, technical errors, and, in the worst-case scenario, rollback.

We also recommend separating Business Object Networks by stages of heterogeneity. Sorting Business Object Networks in this way has clear advantages: If a given Data Migration Tranche contains only Business Object Networks of heterogeneity level 1, then the project team should implement and test only ETL programs that migrate data from the SSOGs belonging to that level. This also means that the team can migrate data from the first Data Migration Tranche before the ETL programs required for further Data Migration Tranches have been developed.

In the aforementioned health insurance project, we occasionally used the target system's business logic for data generation—an interesting opportunity. With the help of the target system's software vendor, several SSOGs (some of which belonged to very complex Business Classes) could simply be generated with the target system's program logic rather than having to be migrated, saving considerable effort for the development and testing of ETL programs. We recommend checking for opportunities like this in other migration projects as well, depending on the capabilities of the target system.

While most of the actual data migration is obviously automated by way of the ETL programs, we doubt that the preparatory steps of our data decomposition approach can be automated effectively: Those activities that could possibly be automated, given a complete and up-to-date specification of the legacy system's data model (which may not even exist for a deprecated system), are comparably trivial and consume only little effort (such as the definition of BONs). Those activities that required more effort (such as the definition of SSOGs) did so in our experience because they involved semantic knowledge about domain aspects (such as objects' business states) that are not included in a data model and thus are not amenable to automation.

4.5 Time Considerations

In order to save time and resources, the history depth for the data migration was defined as four years in our project. This meant that if the end date of the last relationship with a given customer was over four years in the past, said customer records were not considered for the migration but archived. As a general rule of thumb for data migration projects, all Business Objects that are important for business operations must be included in the migration, while Business Objects that are only needed for informational purposes should be archived. It should be noted that the choice of cut-off date is a management decision and does not represent part of our data decomposition approach.

5 RELATED WORKS

Many technical publications and academic works address the topics of data and system migration. The most famous and essential works in this field are probably "Migrating Legacy Systems" by Brodie and Stonebraker [5] and the paper describing the "Butterfly" approach presented in 1997 by Bing Wu et al. [18]. One of the most extensive works focusing on project management in data migration projects is Morris' "Practical Data Migration" [9]. Our approach is partially based on this work. For instance, Morris describes "Units of Migration" as *"a lowest level of data granularity of meaning to the business."* In this paper, we significantly expanded upon and clarified this definition through the introduction of Business Object Networks, Same-State Object Groups and the Heterogeneity measure.

Rueping [11] and Wagner and Wellhausen [17] have defined several data migration patterns that focus on the technical and project management aspects of data migration. Aversano et al. [1] decompose legacy systems into their user interface, application logic and database according to the client-server concept. There are also many other papers that describe aspects relevant to data migration, such as tools, ETL programs, and definition languages for data migration purposes.

As mentioned in several sources (e.g. [5, 8]), there are two major strategies for data migration: The first involves migrating all of the data into the target system through a single step, followed by an abrupt transfer to the new system ("Big Bang", "Cold Turkey" [5], "Cut-and-Run", etc.). The second strategy involves incremental data migration through step-by-step ("Chicken Little" [5]), gradual, partial, or soft migration (e.g. [3, 12]) or abrupt transfer processes ("Butterfly" [18]). All of these approaches describe system and data migration in a highly abstract way.

The majority of authors do not address how to build data packages that can be migrated step-by-step as independent Data Migration Tranches in incremental data migration projects. Nevertheless, one can recognize the following four main concepts underlying data decomposition in those works:

- Firstly, some authors suggest a process based on the boundaries of the components of the application [9] or other business artifacts of the application, such as entities. Other authors refer to "component replacement" [15]. In these cases, the migration can e.g. begin with all customers, followed by all orders and then all invoices (depending on the interdependencies of business artifacts or the degree of support from the target system and its database [8]). Only a portion of the new application's functionality will be made available to all users within one step. This data decomposition concept requires clear, identical component boundaries in both systems, and is unsuitable for non-decomposable enterprise applications [5].
- A second data decomposition concept is based on business processes supported by the application's functionality. In this case, each subsequent data migration step allows more and more business processes to be performed in the target system [10, 16], depending on the sequence of business processes supported by the target system [8] or on the order of criticality of the business process (the "value-based approach" in [4]). Bernhart et al. describe their practical experience with this decomposition concept in [2].

In these first two concepts, which belong to the group of "vertical migration" approaches [8], the main challenge lies in the complex dependencies between datasets, distinct assignment of data to the corresponding business processes, and corresponding overhead for system integration.

- The third data decomposition concept is a part of the Butterfly data migration strategy: In this case, each Data Migration Tranche only contains the data that has been updated since the last data migration step. Data migration will be performed in parallel to fully operational source and target systems. The most complex and costly part of this concept is the program logic required for routing read and write accesses to the corresponding temporal databases [18].
- The fourth concept, which is the most influential concept for our work, assumes total functionality of the target application for only a portion of the application users in each data migration step. This is known as "horizontal migration" [8]. The data can also be split according to the geographical position of the user, business unit, or the company's organizational structure [8, 9, 16]. The main challenge here is to clearly separate the data that has to be migrated within a single migration step from the entire volume of data, and the following system integration.

The data decomposition approach we present here cannot be considered either a purely horizontal or vertical approach; rather, it combines elements of both. It involves splitting data into several DMTs according to the BONs, as well as splitting it according to the functionalities of the applications (Fig. 8). This approach combines the advantages of both horizontal and vertical decomposition while

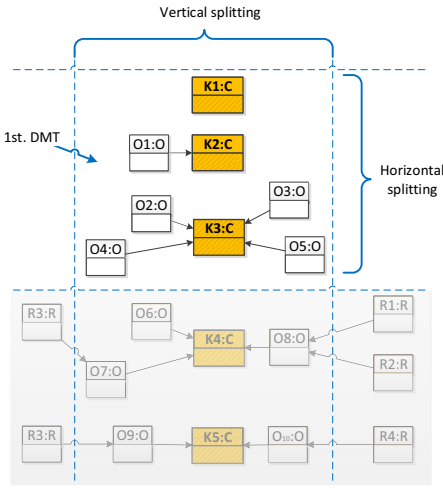

Figure 8: Vertical and horizontal splitting of data into DMTs

avoiding their most critical disadvantages. For example, horizontal splitting requires clear, identical component application boundaries, which is not applicable to monolithic applications. Meanwhile, vertical splitting requires a great deal of system integration. Our approach, in contrast, does not exhibit either of these main disadvantages. In order for our data decomposition approach to be applicable, two preconditions must be however fulfilled. The fact that the migrated BONs can begin to grow immediately after migration partially reduces the advantages of horizontal splitting, as the target application must provide the entire set of required functionalities from day one. This is why the data decomposition approach can primarily be applied if the target system is bought rather than built from scratch. Horizontal splitting concepts do not require this. Secondly, the company must be prepared to work with all functionalities of the target system from day one, so the company employees must be trained accordingly.

6 CONCLUSION

In this paper, we described how to decompose the entire volume of data of a large, monolithic enterprise application into independent Data Migration Tranches. These tranches can then be migrated into the target application incrementally. In brief, we recommend that project teams undertaking data migration projects should start by identifying Business Classes and a Key Business Class in particular, then define corresponding Same-State Object Groups. Next, project members should identify Business Object Networks built from the Business Objects of related Business Classes. Sorting Business Object Networks by heterogeneity and assigning them to Data Migration Tranches creates data packages of adjustable size that can be independently migrated into the target application in successive data migration steps.

Our data decomposition approach has the following key advantages:

(1) Low integration effort: The described Data Migration Tranches (consisting of independent Business Object Networks)

can be migrated independently of each other. Since entire Business Object Networks are migrated at once, no technical or functional references between Primary Data objects will be cut. This does not require any additional systems or data integration. Each migrated Key Business Object can be processed in the target application without any restrictions. All business processes can be executed after each data migration step. This approach can also be applied to monolithic enterprise applications with large numbers of complex dependencies between components, or without clear component boundaries.

(2) Low migration effort: The effort required to migrate Business Objects in many different states can be reduced. As discussed in Sect. 2.6, the data in a data migration project can also be archived, generated, manually entered, or imported into the target system.

(3) Flexibility: The approach provides project members with a clear set of concepts for defining Data Migration Tranches. The size of the Data Migration Tranches can vary depending on the available downtime slots of the application. The number of Data Migration Tranches can also be defined individually.

(4) Quick feedback: The project team can focus on migrating the SSOGs of one Data Migration Tranche at a time. This means that the team members only need to define, implement, and test ETL programs for the SSOGs of the upcoming Data Migration Tranche, and not for the entire set of Business Objects at once. Initial test and migration results will be available very quickly.

All these advantages were confirmed in the application of our approach in an ongoing migration project for a large health insurance company in which about six million KBOs with their attached BONs (i.e. several terabytes of customer records) are being migrated in three DMTs over the course of several years.

Our approach is subject to the following main restrictions, which should be taken into consideration when evaluating its applicability to a particular project:

(1) In-depth understanding of the business: This approach requires an in-depth understanding of existing business processes in the business domain. Project stakeholders must identify artifacts such as Business Classes, SSOGs, and Business Object Networks. However, the process of identifying these artifacts can be supported by software tools used for analyzing dependencies between Business Classes and existing Business Object Network structures on the database level.

(2) Completeness of the target application: When the first Business Object Network is stored in the target application's database, the whole set of target application system functionalities must be available for users. This means that our decomposition approach is only applicable for target systems that are completely developed and cover all the important functionalities of the source application.

(3) Independent Key Business Objects: Last but not least, this approach can only be applied effectively if the migration engineers can identify a suitable Key Business Class (Sect. 2.2). The most important property is that all of Key Business Class' Business Object Networks must be independent of each other to the greatest possible extent.

An interesting future research challenge would be the creation of DMTs from very large application landscapes with thousands of subsystems, where some subsystems could be shut down after the initial migration steps to save costs. Merging data following company mergers also poses interesting research challenges, as the data will not only be migrated in these cases, but also merged with existing data in the target system. Finally, it would be interesting to examine ways of migrating data to and from the structures used by NoSQL databases [14], where the identification of SSOGs would be complicated by the lack of a clear data structure.

REFERENCES

[1] Lerina Aversano, Gerardo Canfora, Aniello Cimitile, and Andrea De Lucia. 2001. Migrating legacy systems to the web: an experience report. In *Software Maintenance and Reengineering, 2001. Fifth European Conference on*. IEEE, 148–157.

[2] Mario Bernhart, Andreas Mauczka, Michael Fiedler, Stefan Strobl, and Thomas Grechenig. 2012. Incremental reengineering and migration of a 40 year old airport operations system. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 503–510.

[3] Jesús Bisbal, Deirdre Lawless, Bing Wu, and Jane Grimson. 1999. Legacy information systems: Issues and directions. *IEEE software* 16, 5 (1999), 103–111.

[4] Matthias Book, Simon Grapenthin, and Volker Gruhn. 2014. Value-based migration of legacy data structures. In *International Conference on Software Quality*. Springer, 115–134.

[5] Michael L Brodie and Michael Stonebraker. 1995. *Legacy Information Systems Migration: Gateways, Interfaces, and the Incremental Approach*. Morgan Kaufmann Publishers Inc.

[6] Klaus Haller. 2008. Data Migration Project Management and Standard Software-Experiences in Avaloq Implementation Projects.. In *Data Warehousing Conference (DW2008): Synergien durch Integration und Informationslogistik*. 391–406.

[7] Klaus Haller. 2009. Towards the industrialization of data migration: concepts and patterns for standard software implementation projects. In *International Conference on Advanced Information Systems Engineering*. Springer, 63–78.

[8] F Matthes and C Schulz. 2011. Towards an integrated data migration process model-State of the art & literature overview. *Technische Universität München, Garching bei München, Germany, Tech. Rep* (2011).

[9] Johny Morris. 2012. *Practical data migration*. BCS, The Chartered Institute.

[10] Ricardo Perez-Castillo. 2012. MARBLE: Modernization approach for recovering business processes from legacy information systems. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 671–676.

[11] Andreas Rüping. 2013. Transform! Patterns for Data Migration. In *Transactions on Pattern Languages of Programming III*. Springer, 1–23.

[12] Philip Russom. 2006. Best practices in data migration. *Renton/USA* (2006).

[13] Pramod J Sadalage and Martin Fowler. 2012. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education.

[14] Karla Saur, Tudor Dumitraş, and Michael Hicks. 2016. Evolving nosql databases without downtime. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 166–176.

[15] Harry M Sneed, Heidi Heilmann, and Ellen Wolf. 2016. *Softwaremigration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung*. dpunkt. verlag.

[16] Sabine Wachter and Thomas Zaelke. 2015. *Systemkonsolidierung und Datenmigration als Erfolgsfaktoren*. Springer-Verlag.

[17] Martin Wagner and Tim Wellhausen. 2010. Patterns for Data Migration Projects. In *15th European Conference on Pattern Languages of Programs (EuroPLoP) – Writer's Workshops*.

[18] Bing Wu, Deirdre Lawless, Jesus Bisbal, Ray Richardson, Jane Grimson, Vincent Wade, and Donie O'Sullivan. 1997. The butterfly methodology: A gateway-free approach for migrating legacy information systems. In *Engineering of Complex Computer Systems, 1997. Proceedings., Third IEEE International Conference on*. IEEE, 200–205.

[19] Nurhidayah Muhamad Zahari, Wan Ya Wan Hussin, Mohd Yunus Mohd Yussof, and Fauzi Mohd Saman. 2015. Data Quality Issues in Data Migration. In *International Conference on Soft Computing in Data Science*. Springer, 33–42.