Our Data Migration Framework

Key Concepts Explained

Abstract

An introduction to the basic concepts of the Migration Framework and how the framework targets complex data migration projects.

Hopp Tech Ltd.

86-90 Paul Street - London EC2A 4NE - United Kingdom
info@hopp.tech

1. About this document

Hopp (hopp) is a collection of components handling the process of complex data migrations. The framework supports all aspects of data migration, from the mapping and implementation of mapping rules, through the actual execution of the data migration right through to the surfacing of the migration results including support for a workflow for tracking and resolving events and problems occurring during the migration.

The purpose of this document is to provide a basic understanding of the key concepts behind the framework, how the framework functions, how the different framework components play their roles in the data migration process, and finally how they integrate internally as well in the external context of where the data migration takes place.

The document should provide the reader with a good, overall understanding of the framework and how this framework is a comprehensive solution targeted specifically for data migration projects.

The mass of detail can be daunting, and another aim of this document is to provide just enough information on each component to facilitate an overall understanding of the processes and functions involved and in this manner, prepare the reader to further explore other aspects and documents of the framework.

Hopp is a meticulously designed suite of components engineered to manage the intricacies of complex data migration projects. This sophisticated framework is engineered to support every facet of the data migration lifecycle. It encompasses tasks ranging from the creation and application of mapping rules to the actual execution of data migrations, concluding with the presentation of migration results. Hopp also incorporates a robust workflow mechanism for the monitoring and resolution of events and issues that may arise during the migration process.

The primary objective of this document is to furnish readers with a foundational comprehension of the fundamental principles underpinning the Hopp framework. We delve into how the framework operates, the distinct roles played by its various components within the data migration process, and how these components interact both internally and within the broader external context of data migration scenarios.

This document is intended to equip readers with a comprehensive grasp of the Hopp framework, highlighting its suitability as a holistic solution tailored explicitly for data migration projects.

Recognizing the potential overwhelmed by the sheer volume of technical details, this document strikes a balance by offering sufficient insight into each component. The goal is to provide an overarching understanding of the processes and functionalities involved, preparing readers for further exploration of additional framework aspects and related documentation.

2. Key concepts

When planning and executing complex data migrations several key issues often prove challenging. The key concept of hopp directly addresses these main concerns, providing robust and complete support that greatly mitigates or indeed eliminates them.

Business Objects	It is valuable to guard the notion that the data being migrated in most cases in fact represents express business concepts that must remain intact after the data migration.
Mapping and implementation	The mapping of the source data to the target data must be specified and maintained and the functionality to perform the migration in accordance to the specification must be developed and maintained.
Execution and in-time assessment	The migration must be executed, the quality of the migration result must be documented, and – in case of unacceptable migration incidents/events – it must be possible to re-execute part or all of the migration.

Business Objects

A tendency when working with data migration is to focus on how to transform the source data into the target data. Reasonably so, as this is the basic task at hand. However, this inherently risks the project evolving into a technical exercise where the mass of data is moved forward through migration steps governed by technical requirements born out of the way the migration has been implemented - without any clear link to the business concepts represented by the data flowing through the migration.

If on the other hand, the coupling to key business concepts remains intact during the mapping and execution of the data migration, valuable and indeed required knowledge residing with people with deep business-related knowledge can be leveraged throughout the lifetime of the migration project.

All elements of hopp are centered on *Business Objects*. A business object represents all the data flowing through the migration-related to one specific item in the business being migrated. Examples of business objects of course depend on the nature of this business, but could be *Customer*, *Account*, *Policy*, *Patent*, *Mortgage,* etc. Business Objects are organized in hierarchies providing a set of root business objects each with a recursive hierarchy of sub-business objects.

Business Objects are the basis for the mapping, for the actual iterations of the migration executions, and for the surfacing of the migration iteration results and events. When executing, every business object passes through the migration steps as one unit.

Business Objects is a basic concept that makes Hopp unique. The benefits are many and varied.

- Business Objects provide a common reference for all users working on and related to the migration project. Business experts, technical users, testers, project managers, etc. alike speak the same language.

- The root business objects and their sub-hierarchies provide a natural partitioning of the task of specifying the mapping. This partitioning is valuable in terms of supporting an iterative and agile process as the project progresses and is equally valuable in terms of measuring mapping progress.

- It is possible to start executing the migration very early when just a small part of any business object has been mapped. An iterative, agile, and incremental process is supported from the start of any data migration project.

- Migration execution can be iterated with complete granularity. For instance, it is possible to point out one specific business object and migrate just that one or iterate all business objects that generated a given event.

- The mapping of business objects includes relationships between business objects. During execution, this information automatically assures the migration of business objects in the correct sequence.

- When surfacing the migration result for a given business object, all the events generated when migrating the business object are presented as well as the complete set of data for the business object for each step in the migration.

When dealing with data migration, there is a common tendency to concentrate primarily on the mechanics of transforming source data into the desired target format. This emphasis is entirely justified, as it constitutes the fundamental task at hand. However, such a focus can inadvertently lead the project down a purely technical path, where data is moved through migration steps dictated solely by technical requirements, with little connection to the underlying business concepts encapsulated within the data undergoing migration.

On the contrary, if we ensure that the linkage to crucial business concepts remains intact throughout the mapping and execution phases of data migration, we unlock valuable knowledge resources held by individuals with deep business acumen. This alignment is indispensable for the entire lifecycle of the migration project.

Within the Hopp framework, every facet is centered around the concept of "Business Objects." A business object represents all the data belonging to a specific entity within the business undergoing migration. The exact nature of these business objects depends on the specific business domain but may include entities like Customers, Accounts, Policies, Patents, Mortgages, and more. These Business Objects are structured in hierarchical relationships, forming a hierarchy of sub-business objects under each root business object.

Business Objects serve as the foundation for various aspects of the migration process, including mapping, execution, and the presentation of migration iteration results and events. During execution, each business object is treated as a single unit, moving through the migration steps cohesively.

The concept of Business Objects is a fundamental differentiator that sets Hopp apart, offering numerous and diverse benefits:

Common Reference	Business Objects provide a universal reference point for all stakeholders involved in the migration project, fostering effective communication among business experts, technical users, testers, and project managers.
Natural Task Partitioning	The root business objects and their sub-hierarchies naturally partition the process of specifying mapping rules. This division facilitates an iterative and agile approach as the project evolves and aids in tracking mapping progress.
Early Migration Start	Migration execution can commence early in the project, even when only a portion of a business object has been mapped. This supports an iterative, agile, and incremental approach right from the project's inception.
Granular Iteration	Migration execution can be iterated with precision. Specific business objects can be singled out for migration, or all business objects associated with a particular event can be iteratively processed.
Relationship Preservation	The mapping of business objects includes inherent relationships between them. During execution, this information ensures that business objects are migrated in the correct sequence.
Comprehensive Result Presentation	When presenting migration results for a given business object, all events generated during the migration of that business object are displayed, along with a complete dataset for the business object at each migration step.

In essence, the concept of Business Objects is central to Hopp's uniqueness, enabling efficient collaboration, flexibility, and precision in data migration projects.

Mapping and Implementation

The mapping rules that govern the data migration must be specified and maintained. This task can be undertaken in many ways. It is very common to use some kind of textual specification – as in Word documents, Excel Worksheets, or likewise. In some respect, this makes good sense, as the users producing these specifications must have deep knowledge of the data involved and notable the business supported by these data. In many cases, these ‘business experts’ are more comfortable producing the specifications in well-known applications such as Word or Excel.

Based on the specifications, a set of some kind of executables must be constructed (implemented) and maintained. It is common to task a team of developers with writing a system of programs to execute the data migration in accordance with the specifications. Historically using some sort of third-generation programming language but recently it is more and more common to leverage the capabilities of one of the many powerful ETL tools out there.

The tools commonly used for the specifications lack support for cross-referencing or validation of the consistency of the mapping being produced. At the very best, inconsistencies can lead to difficult communication and wasted effort when implementing the specifications. Worse scenarios are that inconsistencies of this kind remain undiscovered or are discovered late in the lifetime of the data migration project, making them complicated, expensive and risky to correct.

Regardless of the toolset chosen for the specification and the implementation, these tasks remain a very big and crucial part of any data migration project. The success of any data migration is directly linked to the quality of the specifications and how they are translated into the executables performing the actual data migration. This is underlined by the fact that the specifications often grow to significant size and complexity. It is difficult to maintain validity and coherence in the specifications themselves.

Additionally, in many cases, it proves impossible over time to maintain complete fidelity in the consistency between the specifications and the executables. As the migration project progresses and deadlines approach, there is a severe risk that specifications are left behind while the implementations are modified directly. Very often, the effort invested in the specifications are rendered valueless as the specifications drift further and further behind the reality implemented in the executables.

Finally, it is a common consequence that – as the complexity of the implementation grows – requirements and/or wishes put forward by the business experts in charge of the mapping are rejected by the team in charge of the implementation, due to earlier implementation choices based on earlier specifications.

A key component of hopp is Studio. This is a dedicated, multiuser productivity application providing a comprehensive and consistent interface to produce the mapping. Using Studio, a team of business experts can collaborate to produce the mapping for the executables.

Studio contains rich cross-reference and cross-validation functionality to ensure a very high degree of endurable consistency and coherence in the mapping. Most importantly, Studio facilitates and enforces mapping of an extremely structured nature. In fact, the specifications are so structured that they serve as input to a code generator that generates the migration executables.

The structured specifications and the generated code is a key quality of hopp. The benefits are significant:

- Complete guarantee of consistency between the mapping and the actual executable at all times
- Ever increasing quality of the mapping over time as it is reused from project to project

- The specification is the implementation. The quality of the generated code is such that it is never manually retouched. There is no longer any implementation process or indeed any implementation team

- The business expert is empowered and in charge. Modifications to the specifications done by the business exports directly modify the generated executable

In the context of data migration, the formulation and maintenance of mapping rules are of paramount importance. This task can be approached through various methods, with a common practice being the creation of textual specifications, often using tools like Word documents or Excel worksheets. This approach aligns well with the fact that the individuals responsible for crafting these specifications typically possess profound insights into the data and the associated business processes. Many of these "business experts" feel more at ease utilizing familiar applications like Word or Excel for this purpose.

Subsequently, based on these specifications, a set of executable components must be developed and upheld. Traditionally, this entailed enlisting a team of developers to code a suite of programs that would execute the data migration as specified. Historically, these programs were coded using third-generation programming languages, but in recent times, there has been an increasing trend toward harnessing the capabilities of powerful ETL (Extract, Transform, Load) tools available in the market.

However, the tools conventionally used for creating these specifications often lack built-in support for cross-referencing or validation to ensure the consistency of the mapping. At best, inconsistencies may result in challenging communication and wasted effort during implementation. In more adverse scenarios, these inconsistencies may remain undetected until later stages of the data migration project, making rectification complicated, costly, and risky.

Irrespective of the chosen toolset for specification and implementation, these tasks constitute a substantial and critical segment of any data migration endeavor. The success of such projects hinges directly on the quality of the specifications and their faithful translation into executable components responsible for the actual data migration. This is underscored by the fact that specifications often grow in size and complexity over time, making it challenging to maintain their validity and coherence.

Furthermore, as migration projects progress and deadlines loom, maintaining complete alignment between specifications and executables becomes increasingly difficult. Specifications may lag behind while implementations are adjusted directly. Often, the effort invested in crafting specifications becomes futile as they deviate further from the realities implemented in the executables.

Another common consequence is that, as implementation complexity increases, requirements or preferences voiced by the business experts responsible for mapping may be rejected by the implementation team due to prior decisions based on earlier specifications.

A pivotal component of Hopp is the "Studio." This dedicated, multiuser productivity application offers a comprehensive and standardized interface for creating mapping specifications collaboratively. With Studio, a team of business experts can work together to produce the mapping for the executable components.

Studio boasts robust cross-referencing and cross-validation capabilities, ensuring a high degree of enduring consistency and coherence in the mapping. Most importantly, Studio promotes and enforces the creation of highly structured mapping specifications. These structured specifications serve as input to a code generator that automatically produces the migration executables.

The structured specifications and the generated code constitute a cornerstone of Hopp, yielding substantial benefits, including:

Consistency Assurance	A guarantee of unwavering consistency between the mapping and the actual executable components is maintained at all times.
Specifications as Implementation	Over time, the mapping quality improves as it is reused across multiple projects.
Specifications as Implementation	The generated code is of such quality that manual adjustments are unnecessary. This obviates the need for a separate implementation process or team.
Empowerment of Business Experts	Business experts have direct control and authority over the specifications, as modifications made by them directly impact the generated executables.

In essence, Hopp's Studio streamlines and enhances the mapping and implementation processes, ensuring enduring quality and alignment between specifications and execution in data migration projects.

Execution and In-time Assessment

Once the executables to perform the data migration have been specified and developed, they must be put to work in an execution context to migrate the data. During execution, events will occur that needs to be documented. Some events are benign and of purely informational character, while other events to some extent may invalidate the migration results. In the latter case, it may be necessary to go back, perform some correctional measures, and re-iterate the parts of the migration that were invalidated.

A key concern in any migration project is the way this type of re-iteration is done. Aggravating the problem is the probable presence of dependencies in the data. As an example: If the problem at hand raised the need for a set of Accounts to be re-iterated, this may very well result in the need for a recursive re-iteration of data dependent on these Accounts.

A worst-case scenario is an execution structure bringing the entire mass of data forward in a waterfall, batch-like process of steps. In this case, it can be difficult or even impossible to separate the data relevant to the events in question, in effect forcing a complete iteration of the total migration. A complete re-iteration may be problematic if there is limited time available. In addition, it may introduce unnecessary risks, as data not affected by the problems at hand nevertheless is included in the re-iteration.

The business object approach of hopp renders it completely trivial to point out a precise subset of business objects to be re-iterated. In addition, the integral handling of dependencies between business objects points out the dependent business objects that need to be re-iterated as a consequence. In fact, the hopp runtime can automatically resolve dependencies and recursively re-iterate all dependent business objects.

Another key concern is at what time in the migration execution information concerning the quality of the migration result surfaces. If this information surfaces late, time is lost. In addition, a greater part of the migration may need to be iterated in terms of dependent data than if the information was available immediately when the problem occurred.

In many migration solutions, events occurring during the migration execution do not surface until a part of the migration actively aggregates and surfaces the information in some way. Delays of this kind may introduce unnecessary risks.

It is common that severe problems surface indeed very late - even after the migration as such is finished and has delivered the migrated data. This is the case if the migrated data is erroneous in a way that went unnoticed in the migration but nevertheless renders it impossible to place the data in the target system.

Hopp meets these concerns in two ways. On one hand the framework surfaces events real-time as the migration executes, there is no unnecessary delay at all. In this way – in case of serious events – the migration execution may be immediately paused, the causing problem corrected, surgical re-iteration of affected business objects executed, and the overall migration execution resumed, with minimal risk and loss of time.

On the other hand, hopp contains rich support for integral runtime validation as part of the migration, making it possible to implement validation rules uncovering problems that would arise when placing the migrated data in the target system. In this way, the framework is able to surface this kind of problem as events – early and real-time as all other events.

The inherent, iterative nature of the entire framework migration process results in the constant improvements of these validation rules over the lifetime of a migration project greatly enhancing the overall quality of the target data resulting from the migration. In scenarios where hopp is used repeatedly in different projects migrating to the same target system, the number and quality of these validation rules will continue to grow and improve even between projects.

Once the executable components for executing the data migration have been defined and developed, they must be deployed in an execution context to commence the data migration process. During this execution phase, various events will occur that require documentation. These events can range from benign, providing purely informational insights, to more critical events that may, to some extent, invalidate the migration results. In the latter case, it becomes necessary to backtrack, implement corrective measures, and reiterate the parts of the migration process that were affected.

A pivotal concern in any migration project revolves around the methodology employed for such re-iteration. Complicating matters is the likelihood of data dependencies. For example, if an issue necessitates the re-iteration of a set of Accounts, it may, in turn, trigger a recursive re-iteration of data linked to these Accounts.

A worst-case scenario involves an execution structure that advances the entire dataset through a waterfall-like batch process of steps. In such cases, it can be challenging or even impossible to isolate the data pertinent to the specific events in question. This situation effectively mandates a comprehensive re-iteration of the entire migration process. Performing a complete re-iteration can pose challenges when time constraints are tight and may introduce unnecessary risks, as unaffected data is also included in the re-iteration.

Hopp's business object approach simplifies the identification of a precise subset of business objects for re-iteration. Moreover, its handling of dependencies between business objects automatically identifies dependent objects that need re-iteration. The Hopp runtime can efficiently resolve these dependencies and recursively re-iterate all affected business objects.

Another critical concern is when information regarding the quality of the migration outcome becomes available during the execution. Delayed information retrieval can lead to time wastage and necessitate more extensive re-iteration involving dependent data, compared to having immediate access to information when the problem arises.

In many migration solutions, events occurring during migration execution do not surface until a specific stage in the process actively aggregates and presents the information. Such delays may introduce avoidable risks. Severe problems can sometimes surface very late in the process, even after the migration itself is considered complete. This can occur if the migrated data contains errors that went unnoticed during migration but prevent the data from being correctly integrated into the target system.

Hopp addresses these concerns through two core mechanisms:

Real-time Event Handling	Hopp surfaces events in real-time as the migration progresses, eliminating unnecessary delays. In cases of critical events, the migration can be promptly paused, the issue corrected, surgical re-iteration of affected business objects executed, and overall migration execution resumed with minimal risk and time loss.
Integral Runtime Validation	Hopp offers robust support for runtime validation as an integral part of the migration process. This allows the implementation of validation rules that can detect issues that may arise when integrating the migrated data into the target system. The framework can surface these problems as events in real-time, ensuring early identification.

The iterative nature of the entire Hopp migration process leads to continuous improvements in these validation rules throughout the project's lifespan. This iterative approach significantly enhances the overall quality of the target data resulting from the migration. In scenarios where Hopp is deployed across various projects migrating to the same target system, the number and quality of these validation rules continue to grow and improve, even between projects.

3. Migration Framework Components

Hopp is made of a set of linked and collaborating components. Each component is a feature-rich application and the combined suite of components provides a complete foundation and support for all aspects of the data migration process.

Studio	Studio is a Windows productivity application used to create the mapping. Studio supports and enforces highly structured specifications. In addition, Studio contains extensive cross-referencing and reporting functionality, significantly improving the overall understanding and overview of the mapping. Finally, Studio validates the mapping, clearly reporting any errors and inconsistencies that in turn would cause incorrect or invalid data migration results.
Core	Studio exports the entire mapping in a format usable by the Core Code Gen to generate the program code to execute the data migration. The Core as such contains the code generators generating the engine code as well as base class libraries containing common, supporting functionality for the generated code. While the code generators will generate by far most of the code necessary to execute the data migration, certain migration rules may be implemented by hand. The generated code contains stubs for these rules making their manual implementation straightforward. While manual rule implementations are left alone (not overwritten) by the code generator, any modifications in the mapping that in turn modify the interface for a manual rule implementation will be instantly discovered at compile time. The quality of the generated code is such that it is never manually retouched. While situations certainly occur where the generated code does not produce the desired migration results, these situations are invariably corrected by modifying the mapping and generating the code again.
Portal Operations	The Core Runtime is the execution framework that uses the generated engines to execute the data migration. Its UI is fully integrated into the Portal component. Through the Portal Operations interface, the user loads

source data, populates value sets, executes the data migration and offloads the target data produced by the generated engines.

Using the Portal Operations, it is possible to iterate over the data migration in a very fine-grained manner. It is possible to iterate all business objects that generated a specific event during migration, to iterate a specific business object, etc.

In addition, the Core Runtime supports the operation and execution of multiple data migration projects across a host of different servers.

Portal

The Portal application surfaces the results of a data migration iteration in a web-based interface. These results consist of:

- All events produced
- The data for all business objects used and produced by the iteration
- Deltas showing the differences between iterations

In addition to the passive presentation of the results, the Portal contains rich workflow functionality allowing the involved users to manage responsibility, comments, and state (accepted, fixed, etc.) for all events.

Studio: Producing the mapping

A fundamental element in any data migration scenario is the way it is specified how to migrate the source data to the target data.

The success of any data migration is directly linked to the quality of this specification and how it is translated into the executable that performs the actual data migration. This is only underlined by the fact that this specification often grows to enormous size and complexity. It is difficult to maintain validity and coherence in the specification itself. Most importantly: In many cases, it proves impossible to maintain complete fidelity in the consistency between the specification and the executable.

A key component in Hopp is Studio. This is a dedicated multiuser productivity application providing a complete and consistent interface to produce the mapping. Using Studio, a team of users can collaborate to produce the mapping for the framework executables.

Studio contains rich cross-reference and cross-validation functionality to ensure a very high degree of consistency and coherence in the mapping. Most importantly, Studio enforces mapping of an extremely structured nature. The specifications are so structured that they serve as input to a code generator that generates the migration executables.

It is a key quality of Hopp that the consistency between the mapping and the actual executable is inherently guaranteed.

In any data migration process, the specification detailing the transformation of source data to target data is crucial. The success of the migration hinges on the accuracy and quality of this specification, and its subsequent translation into the executable tasked with carrying out the migration. Given that these specifications can become large and intricate, maintaining their validity, coherence, and alignment with the executable is a challenge.

Hopp's "Studio" is a pivotal tool designed to address this challenge. As a multi-user productivity application, Studio offers a comprehensive and consistent interface to create these data mappings. With Studio:

Collaborative Design	Teams can collaboratively work to design data mappings suited for the framework's executables.
Cross-referencing and Validation	The application includes robust cross-referencing and validation capabilities, ensuring mapping consistency and coherence.
Structured Mapping	Studio emphasizes mappings that are highly structured. Such structured specifications serve as inputs to a code generator, producing the migration executables directly.

A standout feature of Hopp is the inherent guarantee of alignment between the data mapping and the resulting executable, eliminating discrepancies and ensuring successful data migration.

Collaboration

Studio is a Windows application running locally on a PC or laptop. The mapping produced by Studio is a collection of (xml) files residing locally on the user’s machine.

While this enables an individual user to work on a given mapping locally on his/her Windows machine, Studio can be backed by a central repository (a SQL Server database). The repository provides the functionality necessary for a team to collaborate on the same mapping (checkout, check-in and get-latest).

The investment in the mapping can be safeguarded by implementing a suitable backup scheme using the given facilities in SQL Server.

Studio is a Windows-based application tailored for execution on personal computers or laptops. Internally, it processes mappings as a set of XML files stored on the local user's device.

While the design enables users to engage with mappings on their individual Windows platforms, Studio can seamlessly interface with a centralized SQL Server database, serving as its repository. This central repository is equipped with collaboration tools facilitating:

Version Control	Facilitates functionalities such as checking out, checking in, and retrieving the latest versions of mapping files.
Collaboration	Provides an infrastructure where multiple users can collectively work on a shared mapping.

To protect the accumulated work and effort invested in the mappings, it's prudent to incorporate an appropriate backup strategy. The inherent utilities within SQL Server can be harnessed to establish and maintain a robust backup regimen, ensuring the preservation of mapping assets.

Mapping types

In the case of repeated data migrations from varying source systems to the same target system, a clear separation must exist between the mapping for source data and the mapping for target data.

Using Studio, the mapping for any data migration is separated into two different mapping types ensuring the highest degree of reuse of these specifications from migration project to migration project.

Target Map	The Target Map is founded on the description of the Target data. The Target Map eliminates internal references and data that can be derived from other data and exposes the data that cannot be derived and thus must be received. In addition, the Target Map can implement a wide host of runtime validations to ensure the highest quality possible of the target data being produced by the data migration. The Target Map is strongly linked to the target system and this mapping can be reused in all migrations to the same target system. The value of improving/extending the Target Map is retained over time, from project to project. From Studio, it is possible to export the Target Map in two ways:

As an interface specification that can be imported into Studio when working on the Source Map (see below)

As a complete, structured specification that serves as input for the engine generator generating the target migration engine

Source Map

The Source Map is based on both the source data descriptions as well as the data requirements exposed by the Target Map.

While the target mapping exposes the data that must be received, it does so in terms of the target system. In addition, all validation is founded on value sets known by the target system.

On the other hand, the Source Map describes how - based on the source data - to produce the data required by the target map.

Finally, the export can be exported from Studio as a complete, structured specification as input for the code generator generating the export engine.

Quality tools

Mapping commonly grows to significant size and complexity. In many cases users need to communicate around the specifications – for instance, users with knowledge of the target system may need to communicate with users with knowledge of the source system.

The Studio application works as a frame around the different mapping types providing rich facilities useable across the different mapping types.

Cross-reference

Studio provides where-used, cross-referencing capabilities. These capabilities are useful as the mapping grow in size, providing support for where-used analysis and a general control over complex mapping often lacking in other specification tools.

Validation

Using Studio any user can perform a complete validation of the consistency of the mapping. Validation errors indicate that code generation may fail or the generated code may fail to build.

For this reason, the validation is an integrated part of the workflow. In addition, the validation in combination with the checkout/check-in collaboration facilities provides support for what-if scenarios.

A user can check out an item (or any number of items), perform some modification – for instance import a new version of the target data structure - and then perform a validation. The validation report will give a good indication of the impact of the changes, and in case of unforeseen consequences for the consistency of the mapping, the user can simply undo the previous checkout and revert to the previous state of the mapping.

Reporting

Studio provides a palette of reports common for all mapping types as well as reports specific for each mapping type. The reports provide support for communication with other users not in contact with Studio.

Using Studio, items can be annotated with descriptions and comments adding value to extracted reports.

Hopp Core framework, Code generation, and manual rules

The Core contains the code generators generating the code to execute the migration. In addition, the Core provides the base class libraries supporting the generated code.

Apart from supporting the generated code, the base class libraries also contain interface functionality that enables the Core to discover the generated code and call it to perform the different steps in the migration.

Based on the mapping the generated code automatically handles by far most of the migration logic.

In some cases, the mapping contains the information describing the interface to a manual rule that will be called by the generated engine. In these cases, the generated code will contain a default implementation of the rule. This default implementation will report an error event to the Core Runtime, that this rule has not been implemented.

Visual Studio

Visual Studio is the state-of-the-art Integrated Development Environment (IDE) provided by Microsoft to write and maintain .NET program code.

In practice, the generated code for a given mapping resides inside a folder in a normal Visual Studio c# class library project.

Manual rules are implemented in this Visual Studio project by overriding a virtual method provided in the generated code. This overriding method is specified manually in a separate file (using the partial class mechanism in c#) protecting the manual implementation from being overwritten by the code generator. This is a simple, well-known, mainline mechanism and the implementation of manual rules is indeed very straightforward.

The best practice workflow when working with a combination of the mapping from Studio, the code generator, Visual Studio, and finally the Core Runtime is quite simple and supported by extensions to Visual Studio delivered as part of hopp:

Studio	The latest version of the mapping is retrieved from the backing repository. The mapping is then validated and published to a file.
Code generator	The code generator is activated specifying the published file as input and the correct Visual Studio folder as output. The code generator will generate the code and place all generated files in the specified folder, where they become a part of the Visual Studio project.
Visual Studio	If necessary, manual rules are modified/implemented. Then the entire Visual Studio class library project is built and deployed to a location accessible by the Core.
Runtime	The Runtime is reset, causing it to load the new version of the class library containing the latest generated code and manual rule implementations

It is a key quality of Hopp that working through the points above is very fast – normally a few minutes.

The Core, Runtime, and the Execution environment

While the class libraries described above contain the generated code, the manual rule implementations, the interfaces, and indeed all the bits and pieces necessary to migrate one business object, the Core component is charged with the task of migrating all business objects by calling the engine interfaces provided, one object at a time.

In addition, the Core Runtime is responsible for the housekeeping necessary to store all migration results, intermediary results as well as all events occurring during migration, and finally all audit information collected during the migration. It is the responsibility of the Core Runtime to keep track of this information, even when the migration is iterated repeatedly and events and even items may appear, reappear, and disappear.

The complete Core Runtime component consists of two main parts:

Core Runtime	The runtime runs in a server setup and executes all the data migration processing. The runtime runs jobs on a server, performing a multitude of different tasks. The most directly relevant jobs are for instance: Load Source Data, Perform Export Step, Perform Transformation and Target Step, etc. But there are many different job types that in combination make up all functions necessary to run a data migration iteration from start to finish.
Portal Operations	The Portal Operations is a Windows application that connects to the server-based Core Runtime and enables a user to orchestrate the data migration and initiate and monitor jobs as required.

While the collaboration and workflow around the mapping and the generated code takes place locally on a personal machine, once the engine libraries have been deployed in the server environment of the Core Runtime, the rest of the framework execution flow takes place in the Core Runtime and is managed using the Portal Operations.

While the Core Runtime is executing migration jobs, the user monitoring the job execution may view the events raised by the migration engine as they happen. In this way, it is possible to react quickly in case of serious and invalidating events.

Runtime Environment

Normally, in a data migration setup of any significant size, the Core Runtime will run on one or more dedicated servers. However, in a tiny setup, it is perfectly possible to establish and execute the Core Runtime on a local machine as well.

A complete Core setup leveraging the full execution facilities of the Core includes the ability to concurrently iterate over separate data migration projects in the Core setup.

In this full-scale scenario, the Core can execute several, isolated migration projects on one or more dedicated servers in so-called Tracks. One server may host multiple tracks, and the Core can handle multiple servers – each with multiple tracks.

Tracking, monitoring, and collaborating on events and results

The Portal web application presents the migration results and events. As a web application, it is easily reachable by a wider audience, which may include users from the businesses involved in the migration. The aggregations in the Portal application provide a comprehensive understanding of the overall quality of the migration iteration including baseline comparisons that reveal trends since the last iteration.

In addition, the application surfaces detailed information for each migrated business object providing rich support for the users to analyze the results and seek explanations for any issues.

Finally, the Portal provides collaboration functionality enabling the users to keep track of the state of events (new, fixed, accepted, recurring, etc.), to comment on events, and to appoint some user to be responsible for resolving the event.

For any type of business object, the application presents:

- how many business objects of this type have been migrated successfully and how many were rejected during the migration
- an aggregation of the events that occurred for this type of business object

The application enables the user to search or drill down to any specific business object to view:

- the events that occurred for this business object
- links to any related business objects (ancestors and/or descendants)
- the data that were produced for each step in the migration process:
- the data extracted from the source system
- the intermediate result produced by the export engine
- the result produced by the target engine

Translation tables

A special section of the Portal surfaces value sets marked in the mapping as translation tables. This enables users in the migration project (for instance users from the involved businesses) to provide the content for these value sets, normally to provide translations of terms and values in the source system to the target system.

An option is provided for functionality external to hopp to read these translation tables and to write back a validation state (ok or faulty) combined with a validation message. This is useful to ensure that values manually provided by users are correct in terms of the receiving target system.

In the Portal, there exists a designated subsection dedicated to presenting value sets annotated within the mapping as translation tables. This feature facilitates stakeholders involved in the migration process, such as business unit representatives, to populate these value sets, typically for translating terminologies and values from the source system to the target system.

External Interaction - Hopp offers an interface allowing external functionalities to:

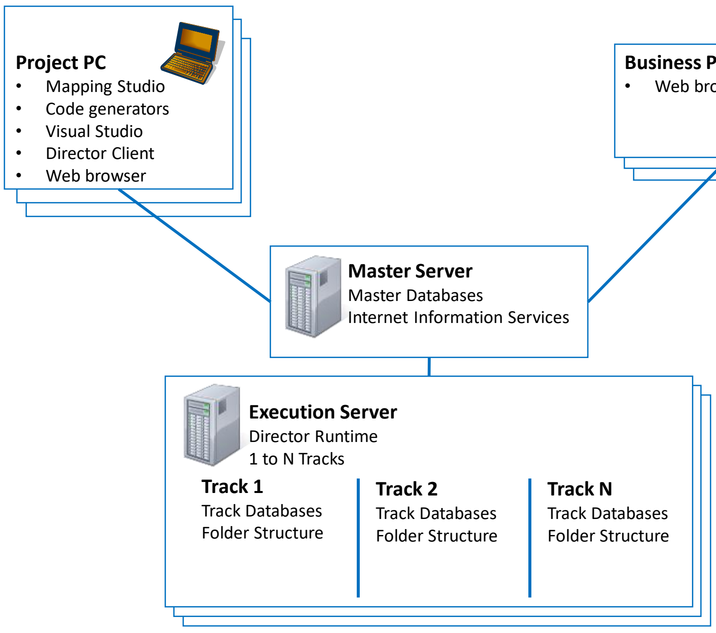
- Retrieve these translation tables.
- Append a validation status (either 'ok' or 'faulty') in conjunction with an explanatory validation message.

Validation Mechanism - This interaction capability ensures that user-provided values align with the expectations and requirements of the target system, preserving data fidelity during the migration process.

Configuration Sample

The entire Migration Framework can be implemented and run on one, single machine – even a laptop. This can be useful for demonstration and prototyping and similar purposes. However, in any context involving larger amounts of data, many different users and maybe even many separate migration projects running in parallel, hopp can scale up to a more elaborate setup.

A sample Migration Framework configuration may look like this:



- Users directly working on the migration project have a Project PC with these local installations.
- Studio so they can collaborate on the mapping. Users all connect the Studio to the same Mapping Repository Database residing on a Master Server.
- The Code generators so they can get the latest version of the mapping, publish it from Studio, and use the code generator to generate new engine code.
- Visual Studio to implement manual rules and build and deploy migration engines in the runtime environment.
- The Portal Operations to initiate and monitor jobs in the Core Runtime.
- A Web browser to view the migration events and results and to collaborate on these.
- Users working outside the migration project, typically involved in the testing and quality assurance, only need a web browser to view the migration events and results and to collaborate on these.

- A single Master Server typically contains the database serving as the repository for the Studio and the Master Database of the Core Runtime, containing housekeeping of all Execution Servers and Tracks. The Master Server also hosts the Tracking web application on an Internet Information Server instance.

- One or more Execution Servers contain the Core Runtime libraries and Services as well as the databases and folders necessary to run migration iterations for separate migration projects in one or more Tracks. One Execution server may be allocated to serve as the Master Server as well.

A Track

A given track is a container for all the artifacts necessary to execute data migration iterations for one project:

- An application repository: A folder containing the engine libraries deployed as a result of the code generation.

- A Staging database: A SQL Server database containing (generated) tables and SQL functionality (generated stored procedures) used by the generated source engine to produce the export result.

- A Runtime database: A SQL Server database used by the Core Runtime to store the intermediate and results of the data migration as well as the events that occurred for each business object.

To keep track of all this, the Runtime also includes a Master database containing all information necessary for the Runtime to manage all the tracks:

- Which migration projects are executing in which tracks

- Folder locations for the track

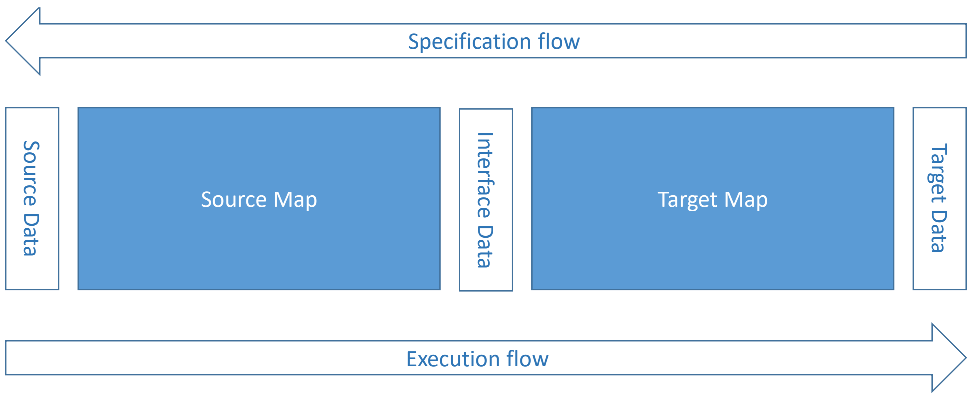
- Database connection strings for the track

- Etc.

Mapping flow and Execution flow

When working on the mapping in Studio, the Target Map is the starting point. From the Target Map, an interface specification can be published and imported into the Source Map.

Put in another way, the Target Map exposes the requirements of the data the target expects to receive from the Source Map.



When on the other hand the generated engines are executed by the Core Runtime, the actual data being migrated flows in the opposite direction. It is extracted from the source data by the source engine and the export result is passed to the target engine. The target engine produces the final target data.

Note that the process of publishing and importing from one specification to another, from the Target Map to the Source Map, is very fast. Flowing through all the steps from a modification in the Target Map, through the step of publish/import, and finally, generation and deployment of new engines rarely takes more than a few minutes.

Mapping semantics

Specifying the mapping rules for any data migration is complicated. The specifications tend to grow indeed very big, there is a myriad of dependencies, and users producing the specifications normally need deep and comprehensive knowledge of the business behind the data being migrated.

While there is no silver bullet to eliminate these factors, Hopp and especially the Studio productivity applications provide complete support for the users to keep on top of the volume and complexity as the migration project progresses, facilitating the production and maintenance of mapping of high and durable quality.

This section is a high-level outline of how the semantics in Studio provide a foundation for highly structured specifications while all the same enabling the flexibility and openness to absorb the peculiarities and specialties that invariably exist in any real-world data migration scenario. Studio is a rich application and for each mapping type there is of course a myriad of details. This section aims to give a basic idea of the three different mapping project types, their areas of responsibility and how they connect and collaborate.

Common elements

While the two project types (Source Map and Target Map) are different in nature, they do use a common set of elements. Because of this, Studio interface remains consistent across the two project types, and manual rule implementation is completely similar.

Constants

A constant is a value and a given data type that can be used everywhere in the mapping. Constants come in two flavors:

- Constant: The value for the constant is provided in Studio and this value is typically incorporated in the generated code as a literal value.

- Parameter: The value for the constant is not incorporated in the generated code, but provided through the Portal Operations interface. Thus, the value may change between iterations.

Value sets

A value set is a table of data organized in columns and rows. Using Studio, the user defines the columns of the value set, giving their names and data types. Once defined, a value set is used by rules to look up values or – in the case of manually implemented rules – in any way needed.

Value sets come in three flavors:

- A static value set: The user populates the value set by typing values directly in Studio.

- A dynamic value set: The value set is populated by the Core Runtime using parameters specified for the value set in Studio. By default, the Core Runtime can read value sets from Excel worksheets, but an extension point is available to implement context-specific value set providers (for instance reading values from the target system).

- A translation value set: The value set is automatically shown in the Portal Web Application enabling external users to populate the value set.

Rules and Flags

Rules are used throughout the mapping in Studio in many different contexts. No matter the context, any rule is defined in Studio by specifying the data type of its return value and name plus data type for each of parameters to be passed to the rule.

Of special interest is that for any rule zero or more flags may be defined – each flag a way for the rule to notify the hopp runtime that it encountered some relevant situation. The flags in combination with Events (see below) is the way the hopp framework decouples the implementation from the invocation context of a given rule. A rule may be invoked in different contexts, each reacting differently to the flags raised by the rule.

Rules are used extensively throughout the mapping, performing a variety of different tasks. For instance:

- Validation rules: Validating input fields

- Mapping rules: Assigning values to output fields (note that in most instances field values can be assigned without the use of mapping rules using other facilities in Studio)

- Condition rule: Deciding whether a certain sub-element should be processed or not

- Exit rules: Typically, cross-validating a business object at the point it is completed

- Etc.

Rules come in two flavors:

- Lookup rules: A rule that uses one or more parameters to look up and return a value from a value set (see above). Rules of this kind are automatically generated by the code generator, no manual implementation is necessary

- Manual rules: The code generator creates a virtual method to be manually implemented in Visual Studio as described above

Events

Using Studio, users can define User Events to be fired when a rule raises a flag. A user event is basically an event code combined with a message text. The message text can contain placeholders for context specific values and can be supplied in multiple languages. If the message text contains placeholders, it is possible to specify the values to be merged into the message text when the event is raised.

The user must specify a severity for the event, causing the framework to act accordingly if the event is fired. The possible reactions are:

Reject	Rejects the current root business object in its entirety
Reject Child	Rejects the current child business object but not the entire root business object
Error	Nothing is rejected, but the migration result will introduce a (non-fatal) error in the target system that must be rectified
Warning	Nothing is rejected, but the migration result may introduce inconsistency in the target system
Information	Information of action taken by the migration. Data may be modified, or new data introduced to improve quality

Whenever a rule (as defined above) is used in Studio, for instance, to provide the value for a field, the user is presented with a panel to define which value to provide for each of the parameters for the rule. The user can provide a literal value, reference a Constant, or provide other value types depending on the exact context.

The panel also presents the user with all the possible flags that can be raised by the rule. At this point, the user decides how the framework should react to each flag. One possible reaction is to reference a User Event to be fired. A User Event is defined:

- Ignore: The flag is ignored; no event is fired and no action is taken

- User Event: The user can identify a User Event as described above. The severity defined on the event will decide how the framework reacts

- System Event: The user decides that the nature of the flag in this context does not merit a user event. In this case, the framework will generate a standard message text in case the flag is raised. In this case, the user must also specify the severity of the System Event

It is the events (User and System) fired in this manner that are collected by the Core Runtime, shown in real-time when monitoring the migration and shown in the Tracking Web Application.

Target MAP

The Target Map is the starting point for all specifications. It is the responsibility of the Target Map to ensure that the data produced by the migration is valid, and can be delivered to and accepted by the target system without late-occurring errors. It is normally the most extensive of the two map types in hopp but also the sole mapping to be reused, if the same target system over separate migration projects may receive data from different source systems.

Starting a Target Map completely from scratch implies importing the specification of the target data structures that the data migration should produce. These data structures can represent anything, for instance, tables in a database, parameter lists, routine calls, etc.

It is the core purpose of the Target Map:

- to define how to produce data for these structures, when the migration executes
- to enforce runtime validations to ensure that the target data produced is acceptable by the target system

In addition, it is the Target Map that creates the business object hierarchies that serve as a mainstay for the entire specification, execution, and presentation of the migration result.

Developing the Target Map involves these main tasks (avoiding an abundance of detailed tasks):

- Manually define the hierarchies of business objects.
- For a given business object, point out the target system structures that this business object will deliver data to
- For each target data structure, determine how to assign the value for each field in the structure. Many ways exist to internally derive/calculate these values from other values inside the target specification
- In the case a given value cannot be derived/calculated in any way, this value surfaces as an upstream requirement for data to be received. This is done by manually create a so-called External Field on the business object
- In some cases, values, can be retrieved from other, related business objects. In these cases, it is possible to create relationships between business objects and use these relationships to retrieve values. Relationships automatically evolve into execution dependencies to be respected by the Core Runtime

The generated target engine contains the code to receive the exported data and call rules etc. as specified to produce the target result.

Publishing the Target Map for import into a Source Map is in fact just publishing the hierarchies of business objects with their external field requirements.

Source Map

The Source Map is built on two different inputs:

- The published data requirements from the Target Map
- The imported data structures from the source system

It is the core purpose of the Source Map to define how to meet the data requirements of the Target Map using the data structures in the source system.

The business object hierarchies rooted in the target specification, with the alterations imposed by the transformation specification, are presented in Studio. For each business object in the hierarchy, it must be specified how the external fields of the business object will be assigned a value.

For this purpose, the Source Map contains an export-specific toolset. Source data structures can be aggregated into views and these views and source tables themselves can be connected to business objects to provide the data necessary.

The generated export engine resulting from the Source Map contains these main parts:

- A generated SQL Server database containing:
- A generated table for each source data structure
- For each view defined in the Source Map
    - A generated table to contain the data for the view
    - A generated stored procedure to populate the table with the data
- For each business object in the specification stored procedures to retrieve the source data necessary to satisfy the data requirements for the business objects
    - Generated code to execute the stored procedures to populate the views
    - Generated code to execute the stored procedures to retrieve source data for the business objects, call rules as specified, and populate the business objects with field data to complete the export result.

Agile workflow

How does it all fit together in a workflow when a data migration project is underway? When do modifications to the mapping take place concurrently with repeated migration iterations and ongoing tests of the migration results in a target system setup? How does it even start up the first time Hopp is put to the task in a new installation context?

From scratch, little by little

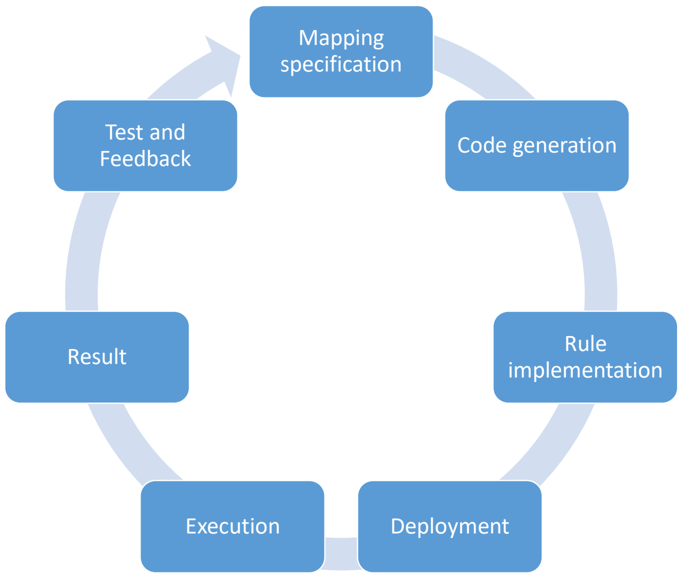
As outlined earlier in this document the mapping sits on top of each other. The Source Map sits on top of the Target Map. This may give the impression that the entire target specification must be completed before progressing to the Source Map. However, this impression is far from the truth.

The iterative nature of the entire framework makes it entirely feasible to proceed in a less daunting and more efficient manner. The best practice is to start the Target Map of just one business object hierarchy and even within this one just a minor part of the entire hierarchy. Moreover, quickly proceed to the Source Map of the same, limited business object hierarchy.

In this way, it is very fast to start the iterative process of the entire framework. The framework calls for this vertical approach where elements in the specification in an incremental fashion are added little by little in Target and Source Maps. Due to the ease of iterating the entire process of migration specification modifications, code generation, deployment, and execution, new areas can constantly and seamlessly be added, and existing areas deepened and enhanced.

Up and running

Once a first limited business object has paved the way and a migration track is operational including the tracking web application, the migration project is in process and the normal workflow is in fact in place. This is an easy iterative flow involving the tasks shown below.



- Mapping is modified. New business objects may be added, existing business objects may be modified. Mapping is published and imported upwards in the mapping type chain as described. Modifications are constantly checked into the common mapping repository
- New code is generated. A user gets the latest version of the mapping from the common repository and publishes the generator input and runs the code generators
- New manual rules may be developed, and/or existing rules modified
- The engines are built and deployed in the Core Runtime environment
- The relevant track is reset so the new engines are loaded, and execution iterated as needed. Either everything or cherry-picking business objects in one way or another
- The migration results and events are shown in the Portal
- Based on the tracking web application feedback flows back and results on modifications to the mapping and the cycle repeats

This iterative workflow is typical. Moreover, it can be very fast. Not counting the time needed for modifying mapping and manual rules, the inherent processes performed by Hopp to generate, build, deploy, and execute can be measured in minutes.

It is common for a migration team to iterate the migration process in this manner many times daily.

Feedback and problem tracing

Feedback may be introduced into the workflow from several sources. Unexpected occurrences of certain events may be shown in the Portal; a test in a target system test instance may uncover issues, etc.

Hopp provides strong support for tracing problems, right from the event or problem in the target system, through the data flowing through the migration steps and back to the area in the mapping in need of a review.

This support is greatly enhanced by the concept of business objects:

- Events in the Portal and/or problems in the target system are directly related to a given business object
- In the Portal, all information for any specific business object is easily located. This information includes:
- All events that occurred during the migration
- All data for each of the migration steps (source data, export result, transformation result and target result)

- For each of the migration steps the data is clearly organized in the same hierarchy of child business objects as is defined in the mapping

In this manner, hopp and the partitioning of the entire mapping and execution flow in business objects provide efficient means of analyzing problems and locating the exact area in the mapping in need of a review.

Snapshots

While the above workflow enables the migration team to iterate to a very high degree, in many instances it is relevant to freeze a snapshot of the data migration iteration at a certain point in time.

Typically, whenever the target data result is unloaded from Hopp and delivered to a test instance of the target system, it is beneficial to freeze a snapshot exactly corresponding to the data that were offloaded and delivered. When the target system test instance is then undergoing tests to verify the quality of the migrated data, this snapshot can be used as described above to analyse and trace any problems uncovered by these tests.

At the same time, iterations in the original can resume allowing the overall forward process of the entire agile workflow – now including corrective actions fed back from the target system test instance.

A special case of this snapshot is when the migration project is finished and the migrated data are finally delivered to the production instance of the target system. In this case, the snapshot may be kept for an extended period. Analysis of any future issues in the target system may be significantly supported by the knowledge of exactly what data was delivered by the migration project as well as all migration events.

Freezing any migration snapshot in this manner is a simple matter of copying of the databases and other artifacts that comprise the Core Runtime track used by the migration iterations for the project.

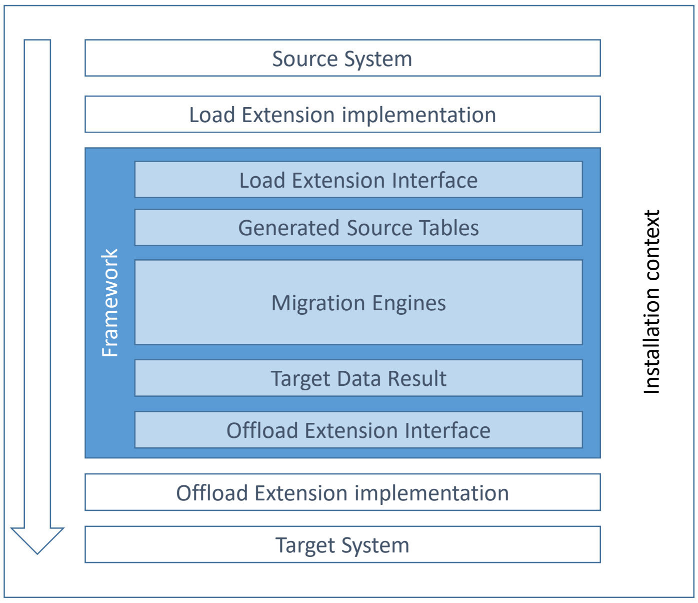
A generic framework in a specific context

Hopp generically approaches data migrations. However, the framework will always exist in a specific installation context and must be able to function in this context. To bridge the gap, hopp comes with several extension points to ease the development of context-specific extensions.

Most obvious are the extensions to load the data from the source system to be migrated and the extensions to offload the resulting target data for delivery to the target system. From the perspective of the generic framework, the migration starts when source data has been loaded into the generated source tables of the export database. Similarly, the migration ends when the resulting target data has been created and stored inside the Core Runtime.

However, from the global perspective of the migration project, this is hardly the entire path. Source data must somehow arrive in the generated tables in the export database and the resulting target data somehow be delivered from the Core Runtime to the target system.

Two important extension interfaces of the framework are the Load Extension Interface and the Offload Extension Interface as illustrated below.



As a principle, tasks of implementing these extensions are part of deploying hopp in a given context. However, some extensions are mandatory, and some are optional. For all mandatory extensions, the framework does come with default extension implementations, allowing rapid initial deployment as well as opening rich possibilities for deeper, proprietary integration.

Extension points

Below is a list of the most common extension points, as well as their default implementations and examples of proprietary implementations.

Load Source Data

Populates the generated source data tables in the export database.

Default implementation
Load from different files formats, such as

- Delimited
- Excel worksheet
- Fixed length
- Db2 textual unload
- Etc.

Proprietary implementation sample
Direct database to database insert or bulk load (if direct access to source system database is possible).

Offload Target Data

The framework holds the created target data for each business object as an XML element. This XML element contains the child business object hierarchy – each business object in the hierarchy contains all target data for this object.

This extension interface receives these XML documents to further process the target data in the implementation context.

Default implementation
Offload to XML files. One XML file for each root business object. Each file contains a copy of all the XML elements for the instances of the business object.

Proprietary implementation sample
If the target data structures represent rows to be inserted in a target database, it is straightforward to implement an offload extension that shreds the target data xml to one file per target table.

Import data structures

This extension interface imports data structures into Studio to use as in the Source Map or the Target Map.

Default implementation
Studio comes with a default extension to import source and target data structures from an Excel spreadsheet.

Proprietary implementation sample
Functionality to read the data structure directly from the table schema in some database management systems.

Populate Value Sets

This extension interface is used to populate value sets with data.

Default implementation
Read data content from Excel spreadsheets.

Proprietary implementation sample
Functionality to read Value Set content directly from the target system.

Audit

This extension interface – if present – is called by the Core Runtime for each business object during migration. The interface permits the extension implementation to hand back audit data for the Core Runtime to store.

Another part of the extension interface is called by the Core Runtime to hand over the collected audit data.

Default implementation
None

Proprietary implementation sample
Commonly the audit data are used to reconciliate the migration results with an expected result from some other data source.

Reject

This extension interface – if present – is called by the Core Runtime for each root business object rejected during the migration.

Default implementation
None

Proprietary implementation sample
Commonly an installation will implement this extension to perform some action for

rejected business objects. For instance, if the bank account is rejected it is nonetheless imperative to place the balance on the account on some technical account in the receiving bank.

There are other specialized extension interfaces outside the scope and detail of this document.