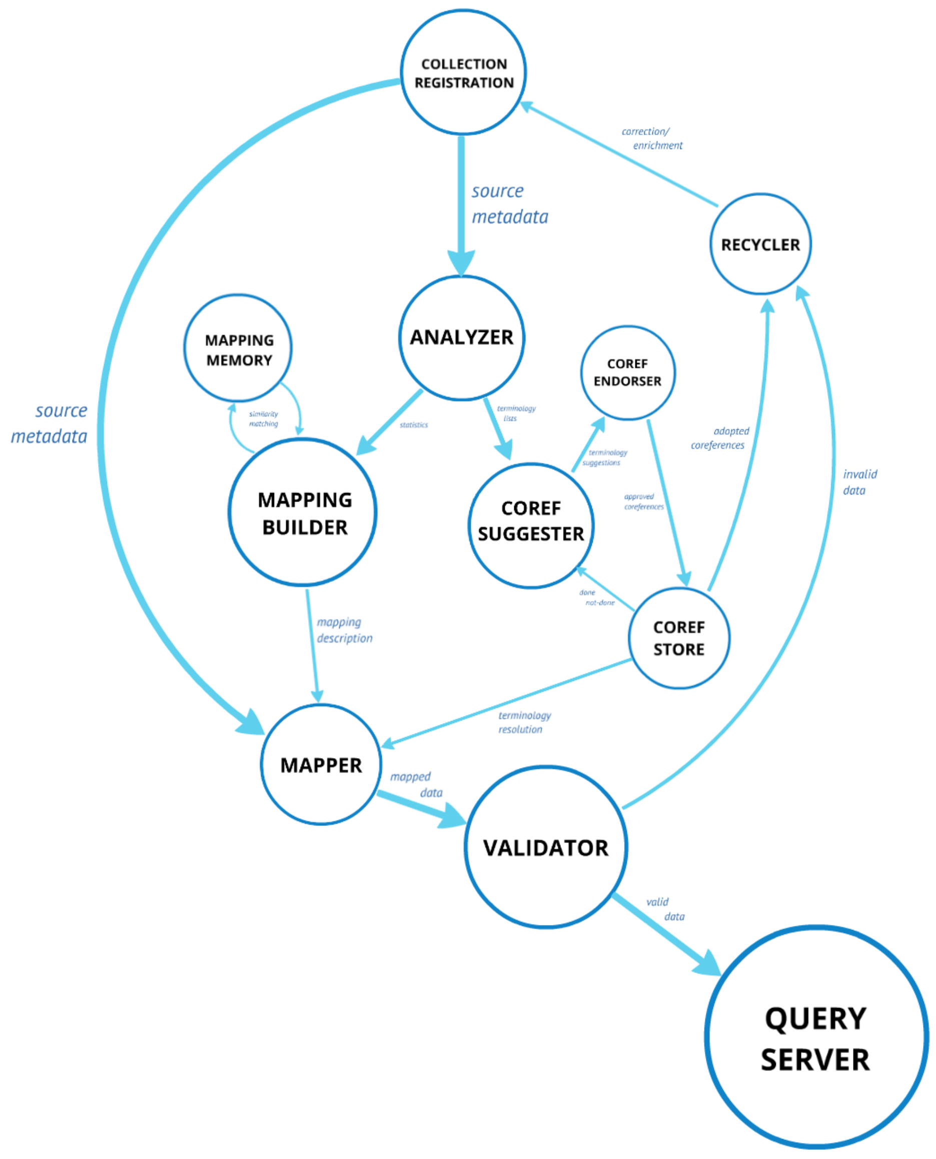
Component Design for Data Mapping Pipeline

This document is currently in draft form and is intended to complement the theoretical description contained in “A Reference Model for Data Mapping tools” by Martin Doerr, released as draft in August 2013.

draft version: 2013-10-20
author: Gerald de Jong
gerald@delving.eu



3) Component Design

In the following sections we describe the necessary infrastructure building blocks for realizing the Process Model as an efficient and sustainable way of working, as depicted in the above diagram. Then we discuss some important aspects of how the blocks can best be connected to create a network pipeline.

The pipeline we envision will strive to streamline the work that human participants must perform by providing intuitive interactive tools, while automating all of the underlying machine processes to be triggered by notifications of changes in the data.

Some of the components described here are purely computational, while others are examples of where interfacing with people receives far more attention. The reference process model describes components as being sometimes part human and part computer, with an emphasis on humans with growing computer-augmented intelligence, rather than autonomous artificial intelligence.

3.1) Collection Registration

Organizations are of course already involved in the process of registering the data for their collections, and they have been doing so for some time. For this they are using their own chosen tools, so although the Collection Registration component of the pipeline is identified here, it is the domain in which we have the least influence. Software vendors and their customers decide when upgrades happen, so we can only propose the model to them which should survive a cost-benefit analysis and therefore provide incentive to adoption.

3.1.2) Integration with the Pipeline

We cannot concern ourselves with the specifics of how an existing Collection Registration system functions, because these systems are many and varied. The only thing that we need to talk about is how these systems can eventually arrive at increasing levels of integration with the pipeline over time, and what we will do in the meantime during the transition.

The issue is that full integration will prompt modifications in how the software behaves for the users, so it will require some software development on the part of the vendor. Presumably the incentive to do so will increase as our infrastructure takes shape.

3.1.2) Open Source vs Closed Source

There are opportunities for accelerating the integration of Collection Registration systems with our pipeline when the systems are open source. We then have the opportunity to define projects in which the necessary integration components can be built and ensure that the developers approve of the approach.

It may make sense at some point for our infrastructure components to include an offering specifically for Collection Registration, which would then be thoroughly integrated with the pipeline out of the box.

3.2) Analyzer

When a dataset is introduced by a provider for the first time, very few assumptions can or should be made by the pipeline about what it contains. Exports expected to be precisely according to an established interoperability standard can prove to be lacking or even misleading in important ways. People tend to apply standards in different ways, depending on how well the associated educational efforts are coordinated.

The only way to be certain about the structure and content is to perform a thorough analysis which assumes nothing the first time. In the end we also want to track the change in the analysis through history, to expose the improvement (or regression) over time. For this to work, all of the analysis products, the statistics, should be recorded in XML for easy consumption by subsequent components of the pipeline and other tools.

3.2.1) Multiple Repeatable Phases

The sheer size of some datasets can make analysis more challenging, but the software can be built to tackle the problem bit by bit. Analysis can be performed in multiple phases, so that the conclusions of one phase informs the scope of the next one. This is purely an optimization which avoids combinatorial explosions which would overwhelm common computer hardware. When the process is broken into phases, it becomes more interactive, and there should be no exceptional hardware requirements.

For example, the difference between an identifier field which has unique values and a descriptive paragraph which also effectively has unique values can be detected without the need to exhaustively examine all values at the outset. A paragraph is a very bad identifier, but in terms of uniqueness it looks the same. Once the real identifier is revealed, the exhaustive test for uniqueness, which is very resource-intensive, can be restricted to the chosen field and not bother considering the paragraph field to be a candidate.

Once a dataset has been taken through all of its phases the first time by a human technician with insight into the data, the analysis should be complete. The decisions made to connect one phase to the next will have been recorded and these instructions will allow for more rapid re-analysis in the future in a single pass. The human interaction can be bypassed since the knowledge is already present in the analysis instructions.

3.2.2) Value and Occurrence Histograms

The first phase of the analysis scans the dataset gathering histograms of the values that appear. In XML these values are associated with paths, so the histograms contain counts of how many times each value appears at every encountered path. This is a straightforward algorithm if it is careful to omit specific paths when they exceed resource usage thresholds. Recording a histogram for a field with unique values is of course futile, since all counts would be equal to one, so a threshold-based approach will have overburdened histograms “explode” and be discarded, freeing up resources for others.

The practical issue of deciding when a threshold has been crossed is of course somewhat arbitrary, but this is not a concern since we know that future phases in the process can compensate. The person doing the analysis can take smaller steps on the way to a full result. There should be enough information gathered for the paths that have been abandoned for the technician to adjust the initial naive assumptions and rerun the histogram analysis with resource usage in mind.

3.2.3) Vocabulary Enumeration

Metadata in Collection Registration systems will inevitably reference elements lists or tables of entities, assuming that the data is maintained in normalized form. These lists can be seen as vocabularies, although often their scope doesn't extend beyond the local database. Even if the data is not normalized, it is often the case that certain fields contain only a relatively small number of different values. In either case, the references may or may not refer to elements of shared lists outside of the local system.

For the Aggregator to be able to attach meaning to the values in these fields, connections must be made with the values of lists that can be shared by all providers. Without these connections, the Aggregator cannot provide unified access to all datasets.

The analysis must initially reveal which fields abide by vocabularies and subsequently ensure that all their values are gathered together for the process of resolution. Resolving a value amounts to establishing the connection via a coreference, indicating that the local value is to be interpreted as a given value from a shared list.

No assumptions can be made about whether the vocabularies were properly used. Instead, an expert must evaluate the usage of vocabulary values based purely on what appears in the source. The Analyzer will be able to reveal which fields correspond to vocabularies and assemble lists of all the values used together with their frequencies. These compiled lists are to be passed on to the Coreference Suggester (3.5 below) to initiate the process of establishing the connections.

3.2.4) Uniqueness Validation

For the data to be useful, it must be possible to refer unambiguously to its constituent parts, so these must be identified with fields which function as unique identifiers. Often these fields are automatically generated and therefore guaranteed to be unique, but not always. In a dataset with millions of records, we must be able to verify that each one of the millions of identifiers never appears twice, regardless of how the data is stored.

Testing uniqueness can be a resource intensive process since there is no way to do it without holding a complete list of values and comparing, so we should expect to aim it at very specific fields manually as an optimization.

3.3) Mapping Builder

The Mapping Builder will be the most advanced and elaborate graphical user interface in the whole infrastructure, because building a mapping is no simple matter. Users must be able to work with sophisticated tools which expose knowledge of the target schema, and capable of hiding much of the underlying complexity.

Building a mapping is best done through a collaboration among multiple experts representing different points of view. Since the ultimate result has to be a single set of instructions, we are clearly dealing with a process of discussion and negotiation.

Also, since a mapping change can have far reaching consequences (triggering much work, hopefully all automated), the process and its individual steps must be managed carefully. There is a strong parallel between building a mapping and writing software code (if they are not essentially the same), so we would be wise to look to software development practices for inspiration, and to use the same tools where possible.

3.3.1) Structure Transformation

Transforming the structure of data from a source schema to a target schema should involve at least two participants, since expertise is required regarding each of the two schemas. The data provider knows the structure and intent of the source schema (Source Schema Expert), and the aggregator is has a solid knowledge of the target schema (Target Schema Expert). The transformation has to produce data which can be merged with data from other providers so the aggregator can provide unified access.

Building the required series of instructions is generally a process which often follows the 80-20 rule (Pareto Principle) where the bulk of the work is easy and quick relative to the effort required to finish the job. An initial transformation can be put together quickly by choosing “from” and “to” fields, but the devil is in the details that follow.

An example of a more difficult challenge would be if a field in the source data corresponds to more than one field in the target schema. This could be because the field is “coded” with an internal syntax which should be parsed to reveal its parts, or because the choice of which output field to use depends on the field values.

The first case requires some work to parse and interpret the field and the ability to create multiple field values as a result. The second would be a conditional mapping which decides which output field to fill depending on a lookup of the incoming value.

3.3.2) URI Generation

When a hierarchical record in the source schema is decomposed into a set of semantic triples, all of the entities must be identified uniquely with a URI. When these URIs are not already present, they can usually be generated. It is not enough to make up new random values for the URIs because there will be multiple URIs which should be able to consistently refer to the same thing. Generating the URI needs to be a predictable repeatable process whenever possible to make this work. The URI generator functions therefore base their work on contextual data, and the Mapping Builder must allow users to choose parameters from related parts of the source documents.

The tools for URI Generation will have to provide for a flexibility and transparency in the description and execution of the various strategies. The functions must be published and maintained in a form which both explains how they work and makes them available to be included as part of the mapping execution. Only by open publication of the functions can we hope that they are optimally re-used by various people trying to accomplish the same thing.

This process is the responsibility of the URI Expert, who will also need tools to help evaluate whether the decisions made were correct. The generated URIs must be checked for appropriateness and uniqueness, and this will involve software to collect and compare on a large scale. It must be possible to verify the effectiveness of the strategies used, and the only way is to have an overview of all URIs used. The statistics coming from this should clearly reveal errors before they become a problem, and they should also expose samples of the results so that they can be evaluated in terms of their structure.

3.3.3) Negotiation

A mapping bridges two worlds, in a sense. This is why we speak of building the mappings as a process of negotiation. The Provider and the Aggregator must get together in order to ensure the authenticity of the transformation in properly rendering the data's meaning.

The data provider's experience is in trying to carefully codify their data as well as they can within constraints in terms of budget, manpower, and equipment. Challenges on the data provider side are usually met by trying to abide by standards where possible and where they are understood, and keep the data as normalized as much as possible (low redundancy).

The aggregator's view of the mapping involves getting the data from various sources into a state where it can be effectively queried as a whole. Every effort that they expend to make the data more explicitly compliant to a model such as the CIDOC-CRM is well spent because only by carefully homogenizing all the data can it properly answer queries over many datasets.

The bridge between the two worlds can be built in any of a number of ways, but there are some basic features of the interaction which must form the foundation. Either participant must be able to present a proposal for a change, it must be possible to carry on a discussion of the proposal which persists for later reference, and there must be a distinct approval event which causes the change to be officially adopted. It is precisely this process which takes place already on a daily basis among software developers, so this is a good place to look for inspiration.

3.3.4) Version Control

All serious software developers work with version control systems which track every individual change in their source code over time. This is common practice because the potential consequences of small changes can be immense, and accountability is indispensable. The same is true for mapping, so there is every reason to think that a similar discipline is required.

The software community has refined version control practices through several generations already, and now there is an advanced system of version control with branching and merging in common usage called "git". There are clear notions of identity, ownership, and accountability for changes.

Changes can be proposed, prepared for execution, and discussed before those responsible make the ultimate decision to have them adopted. The terminology is about creating a "pull request" in another branch which can be executed to modify the "master" branch by those in charge. A pull request has a discussion attached and it can be discarded if it is not judged as fit.

Software developers have all of these functions built into their development environments, so we should work to ensure that any tools that the various responsible actors in the mapping pipeline also have the "git" functionality built in. Some of the complexity underlying the process can be hidden, but it is important to know that there is a rock-solid version control system underneath. If problems appear, software developers will be able to rectify them "by hand" given their daily experience with the underlying tools.

3.4) Mapping Memory

Each dataset is of course in some respect unique, and often they are recorded in different ways even if the name of the software package or storage schema is the same. However, there are generally many more similarities between datasets than there are differences, so it makes sense to have a system in place which encourages sharing of accumulated mapping experience.To enable the effective sharing of mapping knowledge and experience, we speak of Mapping Memory, which should be made available to those collaborating on building a mapping. They should be able to receive hints which offer easy adoption of the elements of mappings which have already been done. This is a shared service, and should be constructed as a kind of query mechanism searching the contents of all existing mappings to date. As the Mapping Memory grows, it becomes easier to build new mappings.

It makes sense to have the Mapping Memory also be the place where all published mappings are stored, because its logic will have to refer to them. This will probably be the Git version control system.

3.4.1) Variations on a Theme

In many real-world cases, a mapping that needs to be constructed will look surprisingly similar to one that has already been done. This is even more obvious in cases where a provider stores a number of datasets in related or the same information systems, and in these cases one mapping is usually either identical or a small variation on one that has just been built.

The Mapping Memory should allow users to kick off a mapping by first adopting an existing one, yet available for minor adjustments. In practice, it’s surprisingly rare that a mapping can be applied to two datasets as is. Mapping descriptions are sufficiently small that we can readily store multiple copies on the assumption that small modifications will nearly always be needed.

3.4.2) Experts in Live Contact

The Mapping Memory service should also provide services which support live communication among mappers while they are working. Since they are all trying to accomplish similar things, it can be very helpful to encourage them to share their expertise with others. A pool of expertise could be formed, which may be integrated with an associated educational program. Features like this are relatively easy to implement and can raise overall quality considerably.

3.4.3) Advanced Matching

Ultimately there are really attractive opportunities for applying some more sophisticated techniques to the process of searching for similar mappings. Even simple tricks like fuzzy matching of paths or tag values can help find non-exact matches. Beyond that, there will be any number of collaborative filtering, clever indexing, or machine learning strategies which could prove very useful. It is a worthy research direction.

3.4.4) Mapping Comparison

The nature of mappings is such that they are diverse in their details but there are great similarities among them as documents. When all mappings are described in X3ML we have the ability to create an inexpensive recursive comparison algorithm.

An efficient comparison algorithm would allow us to create visualizations by projecting mappings into 2D or 3D spaces and have them cluster according to similarity, so that we begin to understand more of the macroscopic features of a mapping infrastructure. A Mapping Manager could look for collaboration with others in the “nearby” space to tackle a given challenge.

3.5) Coreference Suggester

Once the actual values for a vocabulary field have been collected by the Analyzer, they are passed on to the Coreference Suggester. The ultimate goal is to build up maps of coreferences which can be used to resolve vocabulary values in the Mapper, but we will not be willing to trust computers to make these decisions on their own. This is why we refer to this component as a “suggester”, since it’s job is to come up with candidate coreferences which will find their way to the user via the Coreference Endorser. Before a coreference is endorsed by a curator, it is not considered “real”.

3.5.1) Competing Heuristics

There are many ways to look for coreference candidates, and it may be hard to predict beforehand what works best. At the same time, it is of paramount importance to get this right, or at least determine scientifically if we are moving in the right direction.

It would seem sensible to construct the Coreference Suggester in such a way that it can utilize multiple heuristics in a plug-in fashion, so that programmers from various research departments can try their hand at developing algorithms. This is predicated on being able to measure success, and there is no better measure than users.

With the necessary additions to the Coreference Endorser such that the users are easily able to give feedback to the Coreference Suggester, we have the necessary ecosystem for staging a dynamic Darwinian selection of heuristics. The users of the Endorser will have to be prepared to give their opinion of how good the alternatives were. They should not be told which heuristic was responsible, to avoid bias, and it may be best to provide answers coming from multiple heuristics beside each other so that the curator can easily indicate which was the best set of suggestions.

3.5.2) The Question of Why

When someone suggests that you do something, you usually don’t hesitate to ask them why they made this suggestion. There is no reason that a Suggester should not be expected to have an explanation for the suggestions it makes. In fact, starting at a baseline design where a heuristic is required to explain its reasoning is an excellent way to encourage transparency.

The explanation of the reasoning should be available, if not always visible, so that the user preparing to approve or disapprove will be as informed as possible. If a heuristic has presented an answer but for apparently the wrong reason, the curator should be able to indicate this and the feedback must be used.

3.6) Coreference Endorser

The choices of which coreferences to use and which to discard will be made by a human expert. The Endorser component is a user interface which strives to streamline the potentially tedious process of making many small decisions. At its core, it will be a relatively simple interface, but since this process is so important, it deserves extra attention to detail so that it functions optimally for the users.

3.6.1) Coreference Map Building

A coreference map is essentially a lookup table which is ready to provide a replacement value for any value encountered in the source or to augment a value with additional information such as a URI. The decision to use or discard one can be indicated in a single user interface gesture, but it should be presented in as rich a context as possible so that the decision is well grounded.

When a coreference map is to be built for the first time, it amounts to making a potentially large number of decisions, so it may be useful to make sure that the Endorser is also capable of dividing the work and distributing it amongst a number of users.

3.6.2) Coreference Maintenance

Eventually new data will be entered into the local registration system, prompting the addition of new entries in the internally referenced lists. When internal lists are expanded, the new values will not be recognized by the existing coreference map.

An entry with an unrecognized value would presumably be rejected automatically by the aggregator’s validation in the pipeline, and fed back to the data provider. To prevent the unnecessary back-and-forth communication that this implies, there should be a check performed every so often which can locally prompt the addition of the necessary coreferences and trigger the process.

The amount of work is much less than during initial map building, so it will seem much more like making a few more corrections to fix the data so that it can again be exported.

3.7) Coreference Store

The curated coreferences resulting from the Analyzer-Suggester-Endorser pathway must be stored in such a way that they can be used by various other components of the pipeline.

The main user of the accumulated coreferences is the Mapper, since it is responsible for replacing all of the locally-used vocabulary values to values from shared lists advocated by the Aggregator. The large volume of work that the Mapper has to do gives us an incentive to ensure that the Coreference Store is maximally optimized. The use of a triple store is probably not the best approach, but instead we may need to select a specific technology which is built to work at “web-scale” from the start. Lookups must be done very frequently and so speed is important.

The Coreference Suggester must also be informed by the coreferences which have already been chosen so that it avoid asking a user any question which has already been asked and answered. The complete list of vocabulary values used in practice are provided by the Analyzer, and the first process applied to this list will be to filter out the ones already known.

Also, the accumulated coreferences represent a valuable resource which could be used to improve the quality of the source data, so they can be made available to the Recycler for eventual reintegration into the source.

3.8) Mapper

There is a clear distinction between the building of a mapping by experts in collaboration and the actual execution of the mapping in the production pipeline. Building is a slower more painstaking human challenge, while execution can be triggered at any time. Needless to say, execution of the mapping will make up part of the refinement process during the creation of the mapping by experts, but that can take the form of statistical analysis of the results of the mapping run locally, perhaps only on representative samples.

3.8.1) Reacting to Changes

The notion of a pipeline is important in this discussion because it metaphorically implies that the moving parts are things that start to work whenever they are called upon by the flow of data. We see mapping execution as something which should not require human attention once it is constructed and configured. It is a connection between pipes, with one input and two outputs. Individual records may flow through, or batches.

3.9) Validator

The Validator plays a critical role in the pipeline as the gatekeeper which separates the good complete data from the data which cannot be used. It must keep track of several important aspects of the data that it evaluates, and the effectiveness of the fixing and enrichment processes depends heavily on how well the Validator describes the nature of the problems that it has identified. It is not enough just to separate the good from the bad.

3.9.1) Target Schema Validation

The main output of the mapping engine will be the correctly mapped records, so to validate their correctness is the job of the engine. Output validation will pay attention to both structure and content, since the resulting data must express relationships properly and the entities referred to must be known to exist.

3.9.2) Error Filtering and Rerouting

The other output of the mapping engine is the error stream, which will be filled with everything that did not map to satisfying output data. As records are mapped and validated, any problems will manifest themselves as error records in the pipeline.

The pipe leading from the error output must transport the data to the Recycler so it can be communicated to the data provider, including a description of why it was not accepted as valid. Ideally there should be an integration with the collection management system used by the provider to automate the notification and manage the fixing process. In reality, the process will probably have to start as a partially manual operation.

3.9.3) URI Uniqueness Checking

The data being sent into the Query Engine must have no ambiguity in the way it identifies its parts. The Validator must therefore pay close attention to the URIs generated by the Mapper to ensure that they are indeed properly constructed to be completely unique. Only the Aggregator can ensure this uniqueness, and since the list of different URIs from all providers together must be maintained and checked against, this uniqueness checking can be resource intensive.

3.10) Recycler

A difficult challenge presents itself for the data provider in this scenario, and it is one requiring the most multi-faceted tools. The provider will experience a flow of invalid records and correction requests back through the pipeline, and there must be a way to evaluate and act upon them so that invalid records can be promptly resubmitted with corrections.

3.10.1) Step by Step

The main infrastructure challenge is to automate, streamline, and refine this curation process, but an parallel challenge is to facilitate the transition to the new scenario universally. Every kind of provider with their own existing system should be able to participate from the beginning. It is our challenge to make this curation streamlined and fully integrated, but this cannot take place immediately.

To smooth the transition, we will need to provide a way to easily build and maintain a potentially large list of corrections and enrichments so that they can be re-used as long as they are needed (ie until their contents have been adopted and they are rendered obsolete). This collection of adjustments is referred to as a coreference map, where each entry asserts that something in the local data store is related to an element of published shared data such as a list of places.

Feedback from the pipeline will be sent back to the data provider, and there is a potential that their data management systems can be built to receive the feedback and make it easy to act on it. We cannot assume that this will be built into existing systems anytime soon, but this process could start more simply. Feedback may initially be sent via email, for example, so that humans can act on them and make the corrections manually in their existing systems.

By this means, we can at least start the process of improvement. As more providers participate, we should look for ways to integrate and automate, either with existing systems or by migrating existing systems to software that already has the mapping pipeline integrated.

3.10.2) Reintegration

Eventually, and completely at the discretion of the data provider, the knowledge embodied in the coreference map can be adopted into the source data store. New fields can be added for storing extra metadata such as URIs or cached Geo-coordinates, multilingual labels, or anything else available in the shared data.

How this is done specifically in terms of software is impossible to describe in general if it is to be performed by various closed-source software vendors. It will probably involve adjustment of data models and restructuring of user interfaces. Open source solutions for collection management can more easily be adapted to play a more integrated role in the pipeline.

To complete the implementation would involve providing network interface elements like a REST HTTP or XMPP interface so that the registration software can be reached by the pipeline to accept change requests. The pipeline itself will be accessible in standard and well known network interfaces, so it should not be difficult to add client features to existing software.

There may be a general strategy for this which could be adopted by software builders wanting to participate. We can assume that records can be identified uniquely, so if it were possible to simply flag records as being returned by the aggregator, with an accompanying field of explanatory text, we would be well on the way to integration.

3.11) Query Server

The mandate of the Aggregator is to provide unified access to the data contained in the various datasets they manage. This can be done in many ways, and the future potentials are another worthy domain of research, but within the scope of this pipeline infrastructure the goal is nothing more than to feed them with much needed high quality data. The focus of the pipeline is actually more on the cyclic flow back to data providers which promotes the improvement of the source data.

The only assumption that we should probably make initially about the Query Server component's interface is that it should be fed with CRM-RDF. However, we can see that in the future it will need to be notified when a new dataset or an update is ready, and then be able to request that it be sent.

3.12) Pipeline Integration

The existing data delivery strategies tend to only support movement or copying of data in one direction. We must be conscious that this is currently the way many people think, and that the technologies behind this are barely sufficient for our purposes. They probably represent a hindrance. Instead we have to look for delivery mechanisms which lend themselves to creating a multi-directional flow.

3.12.1) Harvest via HTTP

The OAI-PMH protocol was invented in the first years of the millenium as a means by which metadata records could be fetched via a connected series of HTTP requests. Each request returns a block of records and a token with which the next can be retrieved.
There is nothing wrong with an HTTP approach, were it not that there are many different custom implementations of OAI-PMH and that they often lack features. Also, any HTTP based protocol requires that both sides of a two-way conversation be reachable web servers, which can prove very inconvenient.

3.12.2) XMPP Chat Protocol

Since the data transported around the network is best expressed as XML records, the XMPP protocol seems a natural fit. XMPP was designed as a chat service, so it's presence information and buddy list features are handled out-of-the-box. The participants in the delivery infrastructure will be many, so it is important to easily keep track of which machine is connected to which other one, and that they reflect their presence/absence on the network at all times.

Like in other chat protocols, participants form a network and can exchange messages between peers in any direction at will, and there are established protocols for orchestrating these exchanges like IQ Query Action Protocol (http://xmpp.org/extensions/xep-0099.html). This allows the movement of individual records among the nodes in the infrastructure, which creates opportunities to make the whole infrastructure act as if it is a social network of people and machines.

3.12.3) Component Locations

The components depicted in the diagram and described here are not given a location, either by the Provider or by the Aggregator, but that is because it is not clear that this is an important distinction to make. Since these modules are just software, and the intention is to build them in open source, it doesn't matter if they appear in more than one place. Indeed, it makes great sense to have some of these modules actually function in multiple places where possible.

For example, the experts involved with building a mapping should be able to make use of the Analyzer whenever they want to scan or re-scan the source data to gather more or different information to inform the mapping. But they should also be able to run the Mapper to see what results are being produced by the mapping they are in the process of building, and to evaluate the mapping results they should also be able to use the Validator.

The modularity described here is not the physical modularity. Whenever it makes sense because it reduces communication overhead or inconvenience, logical modules can be made available in several places in the physical infrastructure. The Provider could potentially be able to check that the data being delivered with the Validator before the Aggregator finds out.

3.12.4) Change Notification

A great many different things can change at any time in this process scenario, and most of the changes should have the effect of triggering activity elsewhere.
Some changes are small, like the manual updating of an individual record. In the chat environment, this record could be sent on its way to be mapped and aggregated without delay, and feedback could be immediate as well.

Other changes have much larger implications, such as the adjustment of a source or target schema or changing a mapping decision. Notifications for this kind of change will trigger entire datasets to be remapped and revalidated, so these critical changes must be taken much more seriously and perhaps involve human intervention.

The instructions guiding the mapping machinery can change based on new insights, discovered errors, or new policy decisions, in which case the records previously mapped must be once again run through the mapping engine. Changes can occur in terms of the schema syntax, vocabulary usage, URI generation policy, etc, so they can come from any of a number of different actors as well.

Due to the volume of computation that must take place in the event of a change, it would be wise to provide feedback to those who propose the changes so that they are aware of the size of the cascade that their changes will cause.

Changing the mapping engine component in terms of its currently active instructions is not hard or time-consuming, but re-running millions of records will be, so the triggering of the large scale re-mapping should involve human interaction rather than being automatically triggered.

Providers who launch the remapping of their source data may also have to be prepared for a whole new and different set of errors which can require human attention, flowing back through the error channel.