Automation of a Virtual Data Integration Systems for the Management of Genomic Data

Ana León Palacio

Valencian Research Institute for Artificial Intelligence (VRAIN)
Universitat Politècnica de València (UPV)
aleon@vrain.upv.es

September 2023

1 Introduction

This document describes a theoretical proposal for creating an Automated Virtual Data Integration system (AVDIS) based on current solutions. The proposed AVDIS aims to solve or mitigate the main problems derived from data integration in data-intensive domains such as Genomics.

The methodology used to develop the research work is the Design Science Research Methodology for Information Systems [17]. The selected DSRM involves a rigorous process to design artifacts to solve observed problems, make research contributions, evaluate the designs, and communicate the results to appropriate audiences. It consists of six activities: Problem identification and motivation, Define the objectives for a solution, Design and development, Demonstration, Evaluation, and Communication.

Since the result of this research work is to propose a theoretical approach, this document describes the following activities: problem identification and motivation, the definition of the objectives for a solution, and the design of the solution. Therefore, after this introduction, Section 2 is focused on describing the main problems found when integrating data based on the experience of the PROS Research Center in the management of genomic data. Section 3 describes the related works or proposals that try to address the problems found in data integration. Section 4 describes the fundamental constructs required to design a Virtual Data Integration System (VDIS) according to the tasks that must be supported by the system. Section 5 describes the proposed framework for automating as much as possible these tasks and constituting the main result of this research work, the AVDIS framework. The document ends with the conclusions of the resulting research work.

2 Problem Statement

Genomic data management implies collecting, integrating, and querying many data sources with different data schemas, quality, and formats. To deal with genomic data, the PROS Research Center created the Delfos platform. Delfos is made of four modules whose main tasks are to extract and integrate data from different genomic repositories (Hermes module), evaluate the quality of the genomic data and select high-quality information (Ulises module), store the results in a database (Delfos module), and perform queries over the data (Sibila module). The pipeline of the Delfos platform to manage genomic data is shown in Fig. 1.

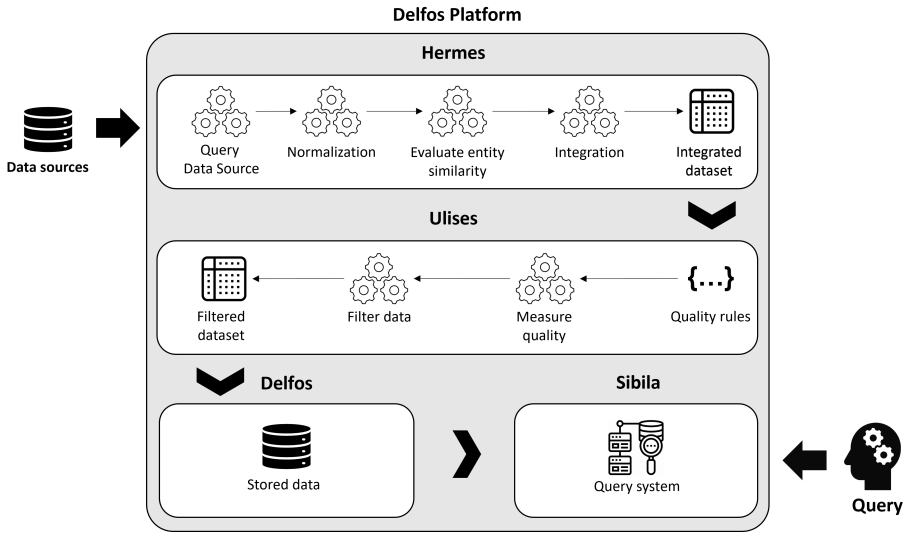

Figure 1: Data management pipeline of the Delfos platform.

The process starts by performing a set of pre-defined queries over different data sources using their available API/REST services. Once the data is retrieved from the data sources, each dataset is transformed into a common format (the Conceptual Schema of the Genome). After normalization, every element of each dataset is compared to the elements of the other datasets to determine which ones represent the same real-world entity. Finally, records are merged and conflicts solved obtaining a single and integrated dataset. Once the integrated dataset is obtained, a set of quality rules are applied to measure the quality of each record. Then, a threshold is applied to obtain a filtered dataset with high-quality results. This dataset is stored in the Delfos database (a relational SQL database) that the user can query using the query system provided by the Sibila module.

Although the Delfos platform has been satisfactorily used to manage genomic data in multiple scenarios, the current approach used by the Hermes module

has some disadvantages and bottlenecks that require a different approach to improve its efficiency and scalability.

Nowadays, data is extracted from the sources through a set of wrappers specifically developed for each repository that uses the API/REST connections provided. The advantage of this approach is that the retrieved data is always up-to-date, the space required by the system is low since no source data is stored locally, and only a subset of the source data is retrieved. Nevertheless, the functionality of the system depends on the internet connection, the availability of the service when the data source is queried, and the amount of data allowed to be downloaded. Additionally, each time a new database is added to the system, a new wrapper must be implemented along with the set of pre-defined queries to extract the required data. This increases the efforts required for maintenance and reduces the flexibility of the system to perform additional queries.

The normalization step takes the extracted data represented in a specific format and uses a set of transformers specifically implemented for each repository to transform the source schema into the common structure provided by the Conceptual Schema of the Genome (CSG). This process is conceived to ease the integration process. Nevertheless, the functionality of the system again depends on the stability of the data source schema. If the source data schema changes, the program in charge of transforming the data to the common format fails. Also, if the CSG changes all the transforming programs affected by the change must be updated. Additionally, if a data source is added to the system a new transformation program must be implemented. This requires a deep study of the source schema, to determine how the different concepts map with the concepts represented by the CSG, and if the original data must be transformed, which is commonly a very complex and time-consuming process.

When more than one data source is queried, the transformed datasets are compared to evaluate the entity similarity and remove duplicates. This is a pairwise process where each element of one dataset is compared to each element of another dataset. This is feasible for small sets of records and a small number of datasets. Nevertheless, genomic data is characterized by the huge size of the datasets and the number of sources to integrate. Therefore, the entity similarity process is commonly a time-consuming task that constitutes a bottleneck in the Hermes module that reduces its scalability.

After similar entities are identified, they are automatically merged. Genomic data sources are commonly of different quality and trusted data should come from more accurate sources. Furthermore, genomics data are dynamic and often evolve over time, and some data sources can copy from each other so errors can be propagated quickly. Hermes is not currently able to solve all the conflicts that may appear, and a manual repair process must be done in most cases to obtain the integrated dataset.

Considering the mentioned problems, a new approach is required for the Hermes module to:

- Avoid system dependence on internet connections and API/REST services.

- Generalize the implementation to reduce the maintenance and implementation process required by wrappers and transformation programs.
- Improve the system's flexibility to perform different types of queries.
- Improve the adaptability of the system to changes in the source schema or the CSG.
- Ease the addition of new sources to the system.
- Improve the entity similarity evaluation and the integration processes.
- Improve the conflict-solving process.

To avoid Hermes depending on the connection to the API/REST services provided by the data sources, the source data must be locally available to be queried at any time. This reduces the currency of the data and increases the storage size as long as new sources are required. Nevertheless, avoids crashes and problems when retrieving huge datasets from the web. API and REST services must be used only for those repositories that do not allow downloading the complete content of their database.

To ease the addition of new sources, improve query flexibility, and reduce the maintainability efforts, a virtual data integration approach can be helpful. Virtual data integration systems rely on data as they are in their original sources, build an intermediate infrastructure to provide virtual data integration, and provide means to retrieve data at query time [9]. Following an incremental approach can allow adding new sources to Hermes without affecting the already existing ones.

To obtain the integrated dataset, Hermes can benefit from using an incremental record linkage approach based on clustering techniques to reduce the number of entities to be compared when measuring similarity. Finally, to improve the conflict-solving process Hermes must consider data fusion techniques that evaluate the accuracy, the freshness, and the dependence of the sources.

3 Related Works

Data integration encompasses three main tasks: schema integration, record linkage, and data fusion. Much literature and solutions have been produced for each of them but in an isolated manner. For schema integration, the most up-to-date work is the one proposed by [9]. In this work, the authors propose a virtual data integration system based on a bottom-up approach, where the structure of the source schemas is extracted from the datasets and generates a globally integrated schema. The authors also provide a fully automated and open-source tool. This approach avoids the possibility of having a conceptual model of the domain as the target schema. This is interesting when a domain conceptual model is not known and an approximation is required to be built from the data. Nevertheless, if the domain conceptual model is known and the sources need to

be integrated following the structure of this target schema to ensure semantic integrity the approach does not provide a solution. The platform proposal presented in this document is based on this approach but extended to provide the possibility of considering the domain conceptual model to ensure the semantic quality of the data.

Record linkage and data fusion solutions are mainly focused on providing algorithms such as the ones compared by [16] and [15]. These algorithms are not fed with domain information, which means that it is quite complicated to resolve data conflicts considering the characteristics of the data sources such as currency, dependency, or reliability. The platform proposal presented in this paper complements the schema integration with the record linkage and data fusion tasks to provide an already treated dataset to the user as an answer to the question performed over the system. It also includes domain knowledge information about the data to improve performance and results during record linkage and data fusion.

4 Virtual Data Integration System

As depicted in Fig. 2, a Virtual Data Integration System (VDIS) needs to perform three levels of tasks [7, 9]:

- Schema integration: First, a VDIS needs to resolve heterogeneity at the schema level by establishing semantic mappings between the contents of disparate data sources.
- Incremental Record linkage: Second, a VDIS needs to resolve heterogeneity at the instance level by detecting records that refer to the same real-world entity.
- Data fusion: Third, a VDIS needs to combine records that refer to the same real-world entity by fusing them into a single representation and resolving possible conflicts from different data sources.

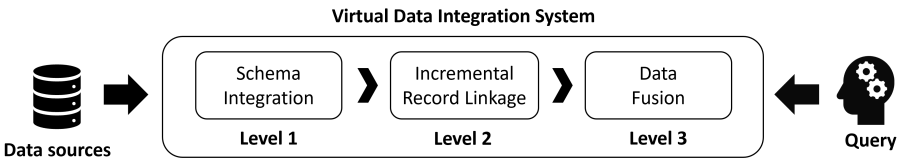

Figure 2: Three levels of data integration in VDISs.

Schema integration and record linkage aim at removing redundancy and increasing the conciseness of the data. Data fusion aims at resolving conflicts in data and increasing the correctness of data [7].

4.1 Schema Integration

Schema integration for a VDIS must support flexible and on-demand extraction of schemata from the sources and homogenization and integration into a global schema [9]. Since schema integration and ontology integration areas have much in common, the knowledge from both areas is combined to define an approach for enabling schema integration. The general workflow of the schema integration level covers the following main tasks [15]:

- Pre-processing: Since schemas are commonly expressed in different languages, a normalization step is necessary to translate them from one language or formalism into a uniform representation without changing their semantics.
- Schema matching: It generates correspondences between schemas, i.e. relationships between one or more elements of one schema and one or more elements of the other [2].
- Schema merging: It merges the input schemas into an integrated global schema.
- Post-processing: It evaluates, repairs, and refines the resulting schema by checking its consistency, and coherence, resolving its cycles and its coherence and conservativity violations, and pruning its redundancies [15].

Having a conceptual model representing the knowledge to be integrated reduces the tasks required during the schema integration. For example, the Conceptual Schema of the Genome (CSG) constitutes a unified and verified global representation of the genomic data. Therefore, the schema merging task and the post-processing task are not required because there is enough knowledge about the domain, and a new data schema is not needed. Only the mappings between the source schemas and the CSG are required. If no domain schema is available, the Schema merging and the Post-processing tasks are required to generate an integrated model from the sources.

4.2 Incremental Record Linkage

Data sources may contain different, complementary, or additional information, which together can provide more complete information. The goal of incremental record linkage, also known as entity resolution, is to identify records that refer to the same logical entity across different data sources. Incremental record linkage procedures are needed when data sources cannot be linked via a unique identification number and the remaining attributes of the entities must be used [8].

Record linkage can be seen essentially as a clustering problem, where each cluster contains records that correspond to a single distinct real-world entity [10] and which general workflow has three main tasks:

- Blocking: This task aims to reduce the number of record pairs to be compared by building an index or block key for each record. More details about blocking techniques can be found in [5, 16, 19].

- Similarity computation: This task compares records of the same block according to a set of matching rules and creates a similarity graph where each node represents a record and each edge represents the similarity between two records. The similarity graph can be simplified by omitting edges whose similarity is below a threshold.

- Graph clustering: This step clusters the nodes according to the results of the similarity graph.

4.3 Data Fusion

Data fusion refers to resolving conflicts from different sources and finding the truth that reflects the real world increasing the correctness of the integrated data [8]. There are two types of data conflicts: uncertainty and contradictions [3, 7]. Uncertainty is caused by missing information (null values or missing attributes in a source) and contradiction is caused by different sources providing different values for the same attribute of a real-world entity. These, are the two main tasks that conform to the general workflow related to data fusion.

Conflicts can be ignored, avoided, or resolved. The approach presented in this document proposes a conflict resolution strategy to reduce as much as possible the production of inconsistent results when performing system queries.

5 A framework for Automating the Tasks of a VDIS

In order to automate as much as possible the data integration process, each of the three levels of tasks constituting the VDIS depicted in Fig.2 (Schema Integration, Incremental Record Linkage, and Data Fusion) must be detailed to specify the tasks to be performed and how they can be automated. Additionally, the VDIS must have a Rewriting System to translate the user queries over the global schema into queries over the corresponding data sources.

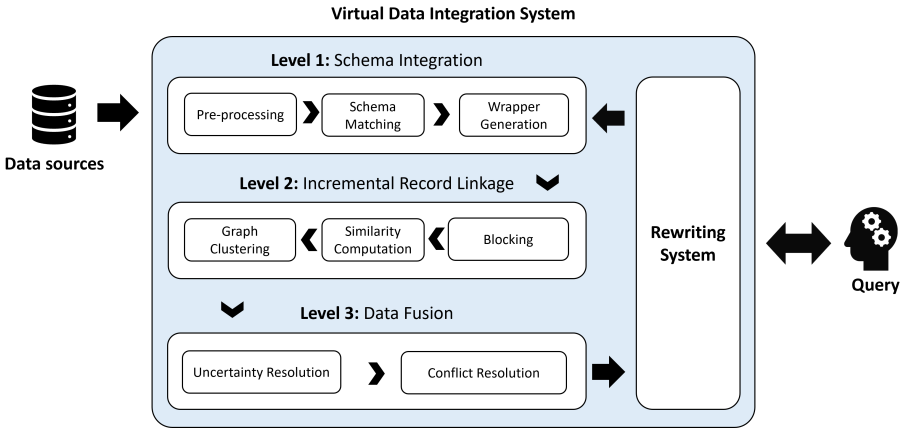

Figure 3: VDIS constructs.

As shown in Fig.3, the Schema Integration level can be divided into the Pre-processing and Schema Matching tasks, the Incremental Record Linkage level can be divided into the Graph Clustering, the Similarity Computation, and the Blocking tasks, and the Data Fusion level can be divided into the Uncertainty Resolution and Conflict Resolution tasks.

5.1 Automating Schema Integration

The schema integration pipeline depicted in Fig.4 is based on the approach proposed by [9]. The pipeline is divided into the Pre-processing, the Schema Matching, and the Wrapper Generation tasks.

5.1.1 Pre-processing Task

The schema integration process starts with the pre-processing of the data and/or meta-data associated with the data sources to be queried. Source data are commonly expressed in different formats (e.g., XML, JSON, CSV). Therefore, bootstrapping and normalization tasks are necessary to infer the schemas from the data sources and translate them from one language or formalism into a uniform

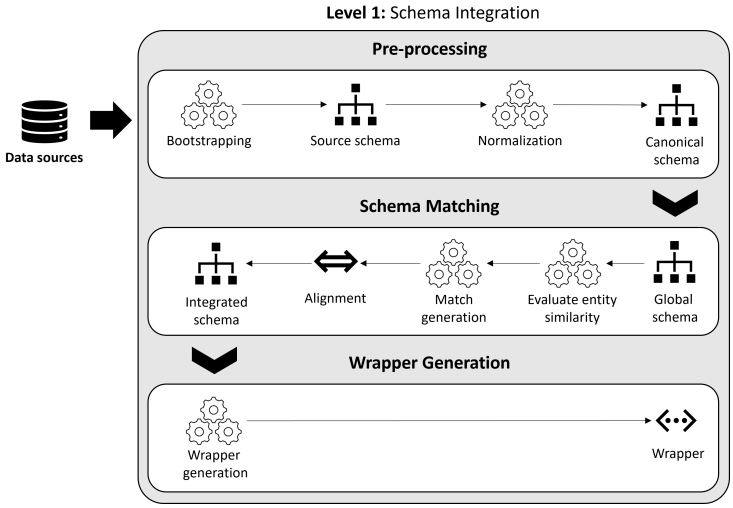

Figure 4: Schema integration pipeline.

representation (or canonical data model) without changing their semantics. To such an aim, [9] suggests representing the source schemas as typed graphs according to a set of metamodels, one metamodel for each type of data source format to be translated. As an example, Fig.5 shows the metamodel used to create typed graphs from JSON data according to [9]. The XML and CSV metamodels can be found in Appendix A.

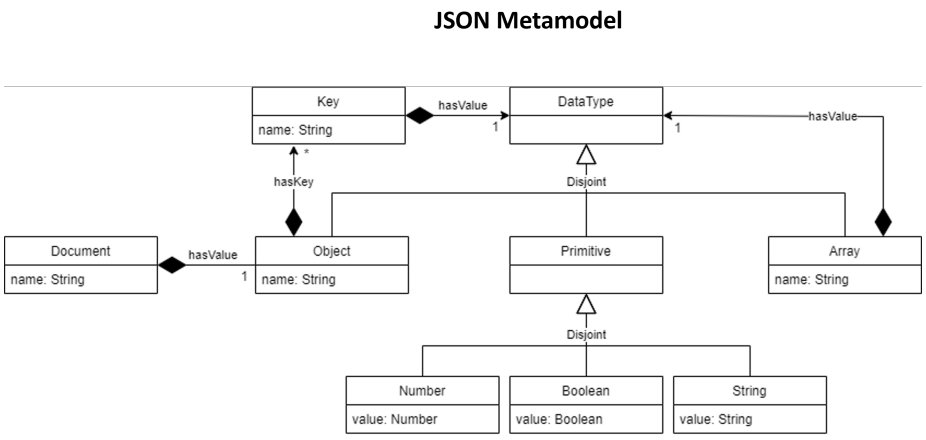

Figure 5: Metamodel used to represent JSON data. Adapted from [9].

The canonical schema can be represented as a Resource Description Framework Schema (RDFS) according to another metamodel, as depicted in Fig.6.

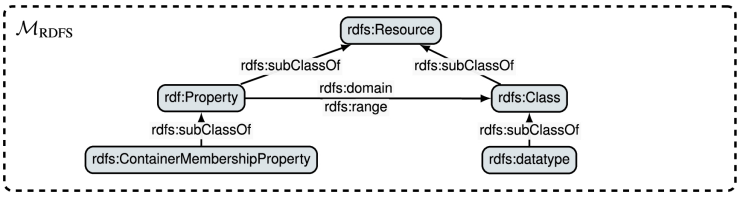

Figure 6: RDFS metamodel used to represent a canonical schema [9].

More details about the transformation rules required to build the RDFS graph can be obtained from [9]. An example of the different schemas generated from XML, JSON, and CSV data can be found in Appendix B.

At the end of the pre-processing task, the data from the data sources are normalized and a canonical schema from each data source has been generated.

5.1.2 Schema Matching Task

Once the canonical schemas are generated, the next stage is identifying semantic correspondences between elements of the canonical schemas and the unified representation of the domain (e.g., the Conceptual Schema of the Genome).

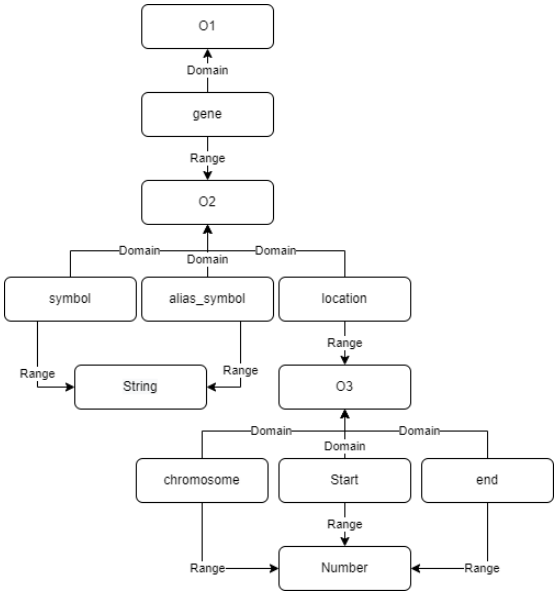

Figure 7: Example of global schema.

The schema matching process takes as input the set of schemas to match coming

from the pre-processing task, and the specification of the global schema, and returns a set of correspondences between entities (classes and attributes) which constitutes an alignment. Fig.7 shows the specification of a global schema that will be used as a running example.

The matching task can rely on measuring the similarity between entities by combining three different metrics related to syntactic, semantic, and structural similarity [4]. Each correspondence commonly has as components the unique identifier of the correspondence, the members of the correspondence, the expressions required to express complex relations and transformations, and a confidence measure assigning a degree of trust in the identified relation [15]. Correspondences can be described using EDOAL[1] (Expressive and Declarative Ontology Alignment Language). Finally, a threshold is applied to the confidence measure to determine the correspondences that will be included in the alignment. A simplified alignment between the JSON example of Appendix A and the global schema of Fig. 7 can be found in Appendix C. Once the alignment is finished, an integration graph containing the mappings between the source schemas and the global schema is created.

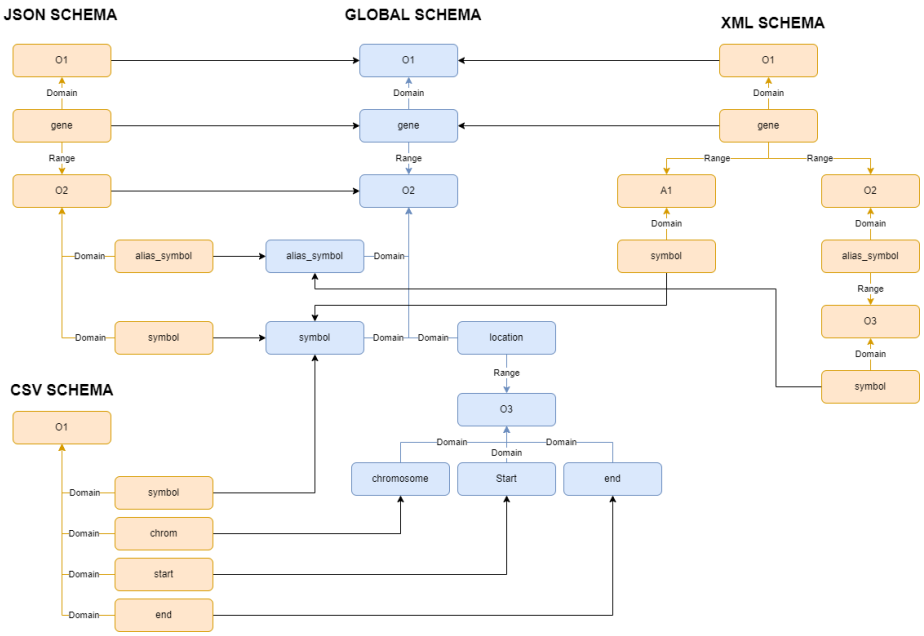

Figure 8: Example of integration graph.

Fig. 8 shows the integration graph resulting from the matching between the examples of Appendix A and the global schema shown in Fig. 7.

[1] https://moex.gitlabpages.inria.fr/alignapi/

5.1.3 Wrapper Generation Task

The Wrapper Generation task consists of creating the programs that allow retrieving data from the sources. Following a Local as View (LAV) strategy, each local schema is described as a function over the global schema, describing which data of the global database are present in each source [13]. This strategy allows adding sources to the system independently of other existing sources. The wrappers translate queries over the global schema into requests understood by specific data sources. The role of the wrappers is to allow the global schema to see each data source as relational, no matter which actual format it uses [1].

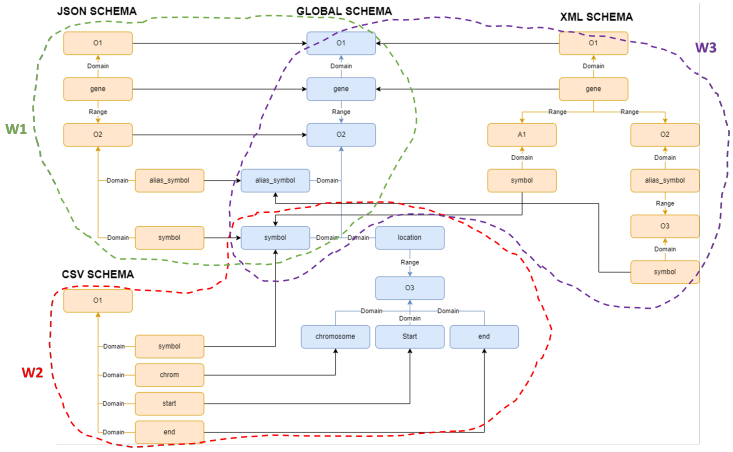

Figure 9: Example of how wrappers cover different parts of the global schema in an integration graph.

Fig. 9 shows how three wrappers cover different parts of the global schema in the integrated graph. More details about the implementation of the wrappers can be found in [1, 13, 14].

5.2 Automating Incremental Record Linkage

The Incremental Record Linkage is executed after a user executes a query on the VDIS. The result of this query is a dataset containing the integrated data coming from the sources and represented according to the global schema. The Incremental Record Linkage pipeline depicted in Fig.10 is divided into the Blocking, the Similarity Computation, and the Graph Clustering tasks.

5.2.1 Blocking Task

The record linkage process starts with an indexing technique commonly called blocking, which splits the datasets into non-overlapping blocks, such that only records within each block are compared with each other [5]. To such an aim,

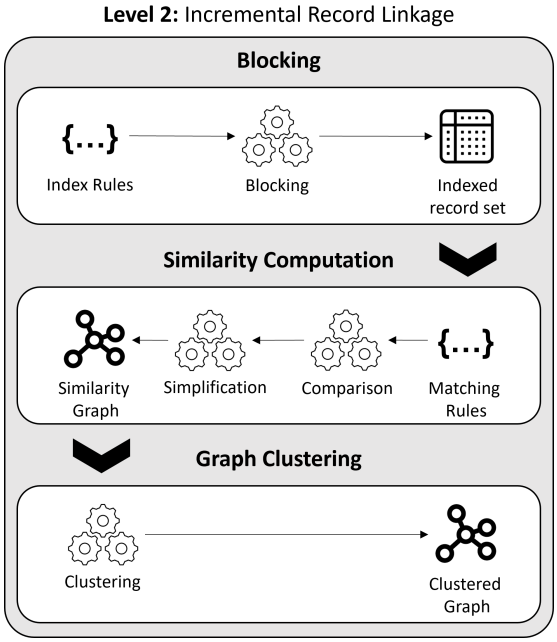

Figure 10: Incremental record linkage pipeline.

a classical blocking approach can be used since the description of the domain knowledge is provided by the global schema (i.e., the CSG).

To build the blocks, a set of rules must be defined so an index or blocking key is assigned to each record. This step requires domain knowledge to select the most appropriate blocking keys. In this case, the CSG may contain the definition of which attributes are required to identify an entity univocally or they can be defined during the specification of the CSG by the domain expert. For example, the set of rules to define the blocking keys for each entity of the global schema shown in Fig. 7 could be $R1 : gene \rightarrow symbol$ and $R2 : location \rightarrow chromosome \cup start \cup end$. $R1$ defines *symbol* as the attribute used as the blocking key for each gene entity and *R2* defines the union of attributes *chromosome*, *start*, and *end* as the blocking key to identify each location entity.

The blocking key can be based on a single record field (attribute), or the concatenation of values from several fields. To overcome errors and increase accuracy, several blocking keys based on different record fields can be generated (i.e., blocking schema). For example, given a dataset corresponding to the integration of the data sources described in Appendix B, the blocking schema S used to index each record is defined as $S \rightarrow \{R1, R2\}$. Matching records have at least one blocking key in common and will be inserted into the same block as shown in Fig. 11.

It is crucial that this calculation be scalable since it must be checked for all pairs of records. When all records are indexed with their corresponding blocking

	Record Fields					Blocking Schema	
ID	**symbol**	**alias_symbol**	**chromosome**	**start**	**end**	**R1**	**R2**
ID1	AARSD1	MGC2744, AlaXp				AARSD1	
ID2	PSEN1		14	73136417	73223691	PSEN1	14-73136417-73223691
ID3	AARSD1	MGC2744, AlaXp				AARSD1	
ID4	AARSD1		17	42950525	42964453	AARSD1	17-42950525-42964453

Block 1		
ID	**R1**	**R2**
ID1	AARSD1	
ID3	AARSD1	
ID4	AARSD1	17-42950525-42964453

Block 2		
ID	**R1**	**R2**
ID2	PSEN1	14-73136417-73223691

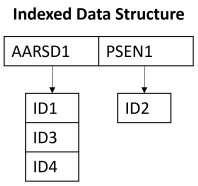

Figure 11: Example of blocking keys and how records are indexed and distributed in blocks.

keys, they are inserted into appropriate index data structures.

5.2.2 Similarity Computation Task

After blocking, all entities of a block are pairwise compared with each other. For each entity pair, the similarity of their attribute values is calculated according to a set of matching rules that specify the required minimal similarity for the considered attributes. For example, using the blocks obtained in the example of Fig. 11, for each pair of records r_i and r_j, some similarity rules could be $SR1 : r_i.symbol == r_j.symbol$, $SR2 : r_i.chromosome == r_j.chromosome$, $SR3 : r_i.start == r_j.start$, and $SR4 : r_i.end == r_j.end$. Therefore, the similarity measure of each pair of records could be the sum of the similarity rules. This step requires domain knowledge to define the similarity rules. The similarity values are used in the following clustering step to decide whether or not a pair of entities match.

The output of this task is the set of matching entity pairs together with a similarity value per link. This output is stored as a similarity graph (see Fig. 12) where entities are represented as vertices and match links as edges [18].

Block 1		
Record 1	Record 2	Similarity
ID1	ID3	4
ID1	ID4	1
ID3	ID4	1

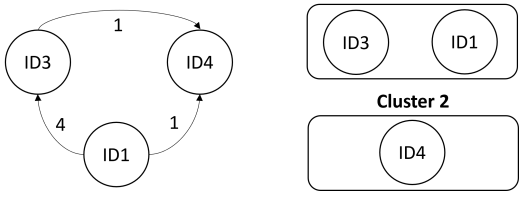

Figure 12: Example of similarity graph and clusters for entities in block 1.

5.2.3 Graph Clustering Task

Clustering algorithms typically try to group entities such that the similarity between entities within a cluster is maximized while the similarity between entities of different clusters is minimized [18]. Each cluster is represented by a cluster graph with the clustered entities and intra-cluster similarity links. In this work, we propose using the CLIP algorithm [18] for the graph clustering stage. An example of the clusters created for block 1 is shown in Fig. 12.

5.3 Automating Data Fusion

Once the integrated data is clustered, the next level is to solve the conflicts between the entities of each cluster to get a single merged entity. As shown in Fig. 13, the Data Fusion pipeline is divided into the Uncertainty Resolution and the Conflict Resolution tasks.

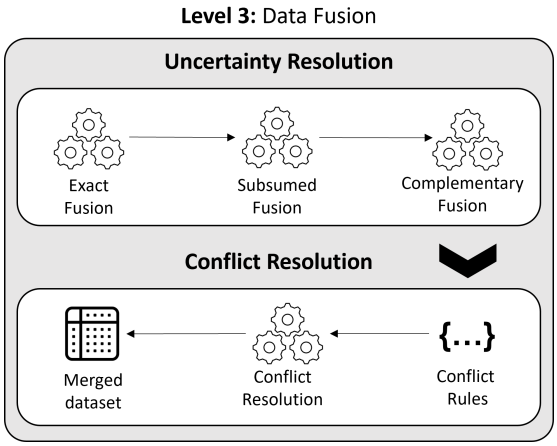

Figure 13: Data Fusion pipeline.

Once each cluster is processed, the results are joined to obtain the final dataset.

5.3.1 Uncertainty Resolution Task

Uncertainty resolution is based on solving the presence of null values or missing attributes in one source. At this stage, the entities are represented according to the global schema, which means that the missing attributes in a source are represented as null values. Therefore, the strategy to solve uncertainty can focus on solving the following scenarios [3]:

- Presence of exact tuples which requires identifying those entities having the same values in all of the attributes.
- Presence of subsumed tuples, i.e. tuples having more null values.

- Presence of complement tuples, i.e. tuples with null values and not contradictions.

The challenge in solving uncertainty is how to resolve these conflicts efficiently and in a meaningful way. To such an aim, relational attributes such as *join* and *union* have been traditionally used. The presence of exact tuples is basically a redundancy-solving problem. For example, the *full disjunction* operator is a variation of the *join* operator that maximally combines tuples from connected relations while preserving all information in the relations [6].

	Name	Age	Room
1	Alice	23	A764
2	Charles	32	S432
3	Alice	23	
4	Alice	23	A764

Table 1: Example dataset with duplicated records.

The result of applying *full disjunction* to Alice records in table 1 would be the first record. This operator solves the presence of subsumed and complement tuples while preserving uniqueness.

5.3.2 Conflict Resolution Task

Conflicts or contradictions are caused by different sources providing different values for the same attribute of a real-world entity. Solving contradictions requires considering the accuracy, the freshness, and the dependence of the sources. This information is domain-dependent which means that requires expert knowledge to define the conflict rules and information about the source of each record.

	Name	Age	UserID	online_time	Department	Source
1	Alice	23	A764	1:02:33	D1	Source_1
2	Charles	32	S432	4:20:02	D2	Source_2
3	Charles	32	S432	4:20:02	D3	Source_3
4	Alice	23	A764	2:02:46	D1	Source_4

Table 2: Example of a dataset with conflict records.

The conflict rules define the actions to be performed in the following scenarios:

- The conflict requires computing a new value (e.g., max, min, merge, or join). In this case, the formula to compute the new value is required. For example, when merging Alice records in table 2, a rule for the value of the *online_time* column could be *online_time = MAX(online_time)* and the result would be record 4.
- The conflict requires selecting one of the provided values according to the priority of the sources. For example, when merging Charles records in

table 2, if source 3 has more accurate data for a user department a rule for the *Department* column could be *Department = Department WHERE Source == "Source_3"* and the result would be record 3.

At the end of this task, the dataset is ready to be sent to the user as a result of the performed query.

5.4 Rewriting System

The user interacts with the VDIS by executing queries over the global schema and getting the corresponding answers. To achieve this functionality, user queries must be translated into a set of specific queries over the sources, considering the particular data schema of each source. Therefore, the system must provide the required rewriting algorithms. Examples of rewriting algorithms can be found in [11, 12].

The Rewriting System uses the wrappers created during the Wrapper Generation stage to extract the required data from the sources. After the Incremental Record Linkage and the Data Fusion levels, the data is returned to the user as the answer to the query.

6 Conclusion

Genomic data management implies collecting, integrating, and querying many data sources with different data schemas, quality, and formats. To deal with genomic data, the PROS Research Center created the Delfos platform. Although the Delfos platform has been satisfactorily used to manage genomic data in multiple scenarios, the current approach used by the Hermes module has some disadvantages and bottlenecks that require a different approach to improve its efficiency and scalability. To ease the addition of new sources, improve query flexibility, and reduce the maintainability efforts, a virtual data integration approach can be helpful. To obtain the integrated dataset, the platform can benefit from using an incremental record linkage approach based on clustering techniques to reduce the number of entities to be compared when measuring similarity. Finally, to improve the conflict-solving process the platform must consider data fusion techniques that evaluate the accuracy, the freshness, and the dependence of the sources. To ensure its usefulness and applicability, the entire process must be as much automatic as possible.

Based on the current state of the art, this document describes a theoretical proposal for creating an Automated Virtual Data Integration system (AVDIS). The proposed AVDIS aims to solve or mitigate the main problems derived from data integration in data-intensive domains such as Genomics.

References

[1] Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M.C., Senellart, P.: Data Integration, pp. 193–225. Cambridge University Press (2011)

[2] Bernstein, P.A., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. Proc. VLDB Endow. **4**(11), 695–701 (2011). https://doi.org/10.14778/3402707.3402710

[3] Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. **41**(1), 41 (jan 2009). https://doi.org/10.1145/1456650.1456651

[4] Cheatham, M., Pesquita, C.: Semantic data integration. Handbook of big data technologies pp. 263–305 (2017). https://doi.org/10.1007/978-3-319-49340-4_8

[5] Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE transactions on knowledge and data engineering **24**(9), 1537–1555 (2011). https://doi.org/10.1109/TKDE.2011.127

[6] Cohen, S., Sagiv, Y.: An incremental algorithm for computing ranked full disjunctions. Journal of Computer and System Sciences **73**(4), 648–668 (2007). https://doi.org/https://doi.org/10.1016/j.jcss.2006.10.015

[7] Dong, X.L., Naumann, F.: Data fusion-resolving data conflicts for integration (2009), `https://lunadong.com/publication/fusion_vldbTutorial.pdf`, last accessed 3 August 2023

[8] Dong, X.L., Srivastava, D.: Big data integration. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE) (2013). https://doi.org/10.1109/ICDE.2013.6544914

[9] Flores, J., Rabbani, K., Nadal, S., Gómez, C., Romero, O., Jamin, E., Dasiopoulou, S.: Incremental schema integration for data wrangling via knowledge graphs. Semantic Web **Preprint**(Preprint), 1–38 (2023). https://doi.org/10.3233/SW-233347

[10] Gruenheid, A., Dong, X.L., Srivastava, D.: Incremental record linkage. Proceedings of the VLDB Endowment **7**(9), 697–708 (2014). https://doi.org/10.14778/2732939.2732943

[11] Hai, R., Quix, C., Zhou, C.: Query rewriting for heterogeneous data lakes. In: Benczúr, A., Thalheim, B., Horváth, T. (eds.) Advances in Databases and Information Systems. pp. 35–49. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-98398-1_3

[12] Imprialou, M., Stoilos, G., Cuenca Grau, B.: Benchmarking ontology-based query rewriting systems. Proceedings of the AAAI Conference on Artificial Intelligence **26**(1), 779–785 (Sep 2021). https://doi.org/10.1609/aaai.v26i1.8215

[13] Katsis, Y., Papakonstantinou, Y.: View-based Data Integration, pp. 3332–3339. Springer US, Boston, MA (2009). https://doi.org/10.1007/978-0-387-39940-9_1072

[14] Nadal, S., Abelló, A., Romero, O., Vansummeren, S., Vassiliadis, P.: Graph-driven federated data management. IEEE Transactions on Knowledge and Data Engineering **35**(1), 509–520 (2021). https://doi.org/10.1109/TKDE.2021.3077044

[15] Osman, I., Yahia, S.B., Diallo, G.: Ontology integration: approaches and challenging issues. Information Fusion **71**, 38–63 (2021). https://doi.org/10.1016/j.inffus.2021.01.007

[16] O'Hare, K., Jurek-Loughrey, A., Campos, C.d.: A review of unsupervised and semi-supervised blocking methods for record linkage. Linking and Mining Heterogeneous and Multi-view Data pp. 79–105 (2019). https://doi.org/10.1007/978-3-030-01872-6_4

[17] Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. Journal of Management Information Systems **24**(3), 45–77 (2007). https://doi.org/10.2753/MIS0742-1222240302

[18] Saeedi, A., Nentwig, M., Peukert, E., Rahm, E.: Scalable matching and clustering of entities with famer. Complex Systems Informatics and Modeling Quarterly **16**(95), 61–83 (2018). https://doi.org/10.7250/csimq.2018-16.04

[19] Steorts, R.C., Ventura, S.L., Sadinle, M., Fienberg, S.E.: A comparison of blocking methods for record linkage. In: Privacy in Statistical Databases: UNESCO Chair in Data Privacy, International Conference, PSD 2014, Ibiza, Spain, September 17-19, 2014. Proceedings. pp. 253–268. Springer (2014). https://doi.org/10.1007/978-3-319-11257-2_20

A Metamodels to build typed graphs from JSON, XML, and CSV data

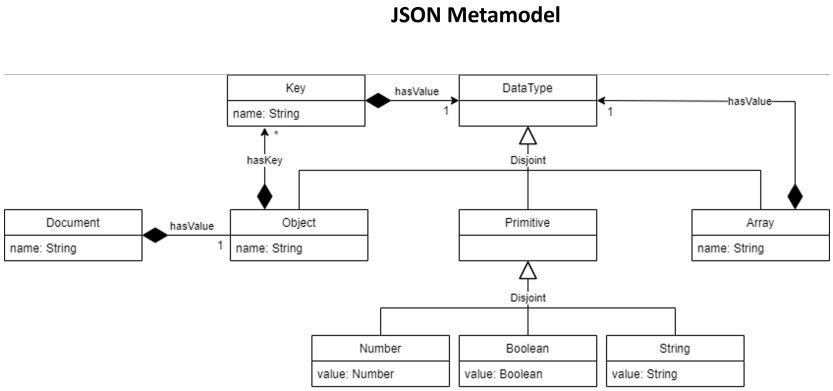

Figure 14: Metamodel used to represent JSON data. Adapted from [9].

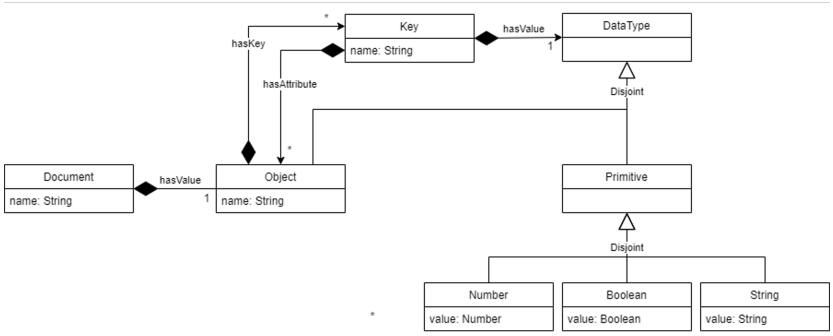

Figure 15: Metamodel used to represent XML data.

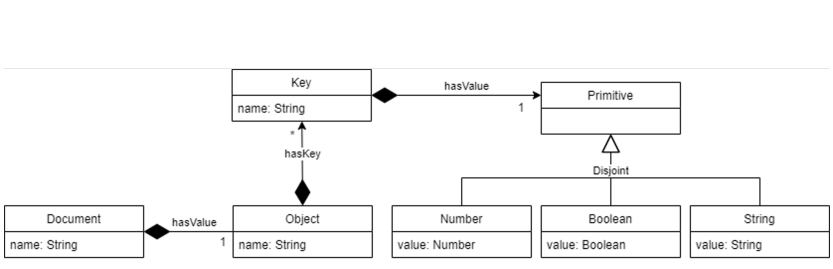

Figure 16: Metamodel used to represent CSV data.

B Example of data transformation into canonical graphs

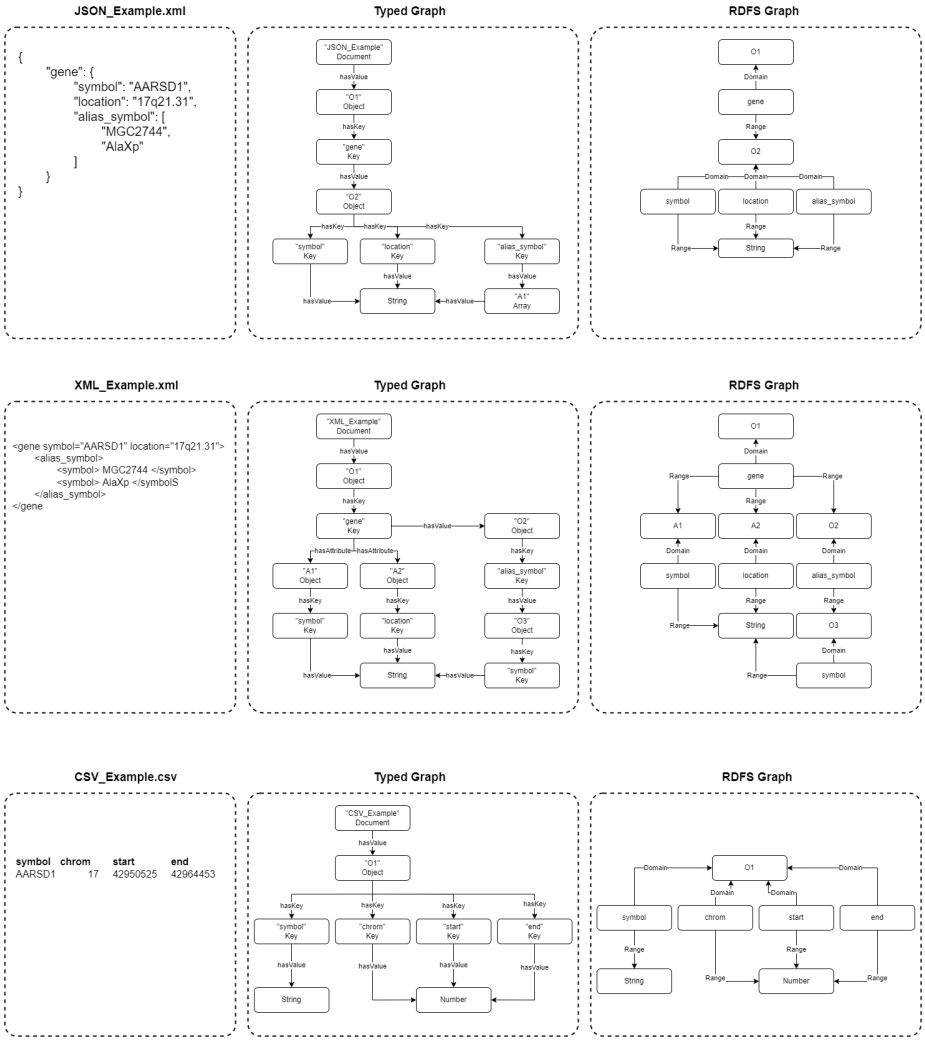

Figure 17: Example of the different schemas generated from data in JSON, XML, and CSV formats.

C Simplified example of an alignment using EDOAL

```
<align:Alignment>
  <align:level>2EDOAL</align:level>
  <align:onto1>
    <align:Ontology rdf:about="JSON_Canonical_Schema"></align:Ontology>
  </align:onto1>
  <align:onto2>
    <align:Ontology rdf:about="Global_Schema"></align:Ontology>
  </align:onto2>
  <align:map>
    <align:Cell>
      <align:entity1><edoal:Class rdf:about="JSON_Canonical_Schema#gene"
          /></align:entity1>
      <align:entity2><edoal:Class rdf:about="Global_Schema#gene"
          /></align:entity2>
      <align:relation>=</align:relation>
      <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
      <edoal:transformation>...</edoal:transformation>
      <edoal:linkkey>...</edoal:linkkey>
    </align:Cell>
  </align:map>
  <align:map>
    <align:Cell>
      <align:entity1><edoal:Relation
          rdf:about="JSON_Canonical_Schema#gene;symbol"
          /></align:entity1>
      <align:entity2><edoal:Relation
          rdf:about="Global_Schema#gene;symbol" /></align:entity2>
      <align:relation>=</align:relation>
      <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
      <edoal:transformation>...</edoal:transformation>
      <edoal:linkkey>...</edoal:linkkey>
    </align:Cell>
  </align:map>
  <align:map>
    <align:Cell>
      <align:entity1><edoal:Relation
          rdf:about="JSON_Canonical_Schema#gene;alias_symbol"
          /></align:entity1>
      <align:entity2><edoal:Relation
          rdf:about="Global_Schema#gene;alias_symbol" /></align:entity2>
      <align:relation>=</align:relation>
      <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
      <edoal:transformation>...</edoal:transformation>
      <edoal:linkkey>...</edoal:linkkey>
    </align:Cell>
  </align:map>
<align:Alignment>
```