Middleware Based Agentic Framework for Data Migration with Automated Schema Mapping, Data Quality Validation, Data Reconciliation, and End-to-End Data Lineage

Shah Mohd Imaduddin[1], Dr. Asha Shiny[2]

[1]PhD Research Scholar, Bharatiya Engineering Science and Technology Innovation University (BESTIU)
[2]Professor, Department of Information Technology, CMR Engineering College, Hyderabad, India
Email: is.rubaiyat@gmail.com, 2024SPCSE050@bestiu.edu.in

Abstract—Moving data from one enterprise system to another is complex. Behind every migration lies an issue of mismatched schemas, inconsistent data quality, and validation gaps that teams spend critical time in trying to Fix things manually. This paper presents the Middleware Based Agentic Framework for Data Migration purpose-built, agent-driven system that handles the entire migration lifecycle without requiring constant human intervention. Built on a Python Fast API middleware layer, the framework deploys five intelligent agents: Schema mapping Agent, Data Quality checks Agent, Migration scripts Agent & reconciliation Agent. Each agent communicates through versioned Markdown files stored in a Git repository, which also doubles as a complete audit trail. We tested the framework against a real D365 F&O to Oracle Fusion migration and the results were compelling—schema mapping accuracy held above 97%, reconciliation rates stayed above 99.9% across all objects, and what would have taken a team of engineers several weeks was completed with over 80% less manual effort.

Keywords—Data Migration, Agentic Framework, Middleware, Schema Mapping, Data Quality Validation, ETL Automation, Data Reconciliation, Data Lineage, Multi-Agent Systems, Fast API, Autonomous Agents

I. INTRODUCTION

Every organization has issues sooner or later—the legacy system that have outgrown simply cannot keep up with where the business is headed. The decision to migrate to a modern ERP or cloud platform is rarely the hard part; the hard part is actually moving the data. When migrations go wrong, and they frequently do, the consequences show up quickly: records that cannot be reconciled, reports built on incomplete data, and downstream processes that break in ways nobody anticipated. The bigger and more complex the source system, the worse it gets.

The way most organizations approach migrations has not changed much in decades. Engineers sit down with source system documentation—where it even exists—and start mapping fields by hand. They write ETL scripts, test them, find gaps, rewrite them, and eventually reach a state that is good enough to run. This approach works, eventually, but it is slow, expensive, and heavily dependent on individuals who understand both systems well. When those people leave or the scope changes, the whole effort can unravel.

What has changed recently is the availability of large language models that can actually reason about schema structures, not just pattern-match them. An LLM-powered agent can look at a D365 F&O table definition, understand what the fields mean in a business context, and work out how they correspond to their Oracle Fusion counterparts—something a rule-based tool simply cannot do reliably. Chain several such agents together under a coordination layer and you have the foundation for a migration pipeline that runs itself, flagging exceptions for human review rather than requiring humans to drive every step.

This paper describes exactly that kind of system. We built the Middleware Based Agentic Framework for Data Migration around a Python FastAPI core that orchestrates five specialized agents, each responsible for one stage of the migration. Agents hand off work to each other through Markdown files that live in a Git repository—a simple design decision that turns out to be very powerful, because every mapping decision, quality check, and script generation event is automatically versioned and auditable. The framework also tracks data lineage from end to end, so there is always a clear record of where a value came from and what happened to it on its way to the target system.

II. RELATED WORKS

The work presented here draws on several distinct research traditions. Understanding where this framework sits requires looking across data migration practice, agent-based systems, schema matching algorithms, ETL engineering, and data quality theory—fields that have developed largely in parallel and rarely in conversation with one another.

On the migration methodology side, Morris [1] was among the first to treat data migration as a proper engineering discipline, laying out a structured sequence from planning through extraction, cleansing, loading, and verification. That sequence remains sound. What it lacks is any mechanism for automation—the methodology is a checklist, not a pipeline. O'Neil [2] made the same observation from a different angle, cataloguing the specific ways legacy data structures resist migration: field naming conventions that made sense in 1998 but mean nothing today, schemas with no documentation, and data that was never cleaned because the old system never required it to be.

Schema matching has attracted substantial attention from the database research community. Rahm and Bernstein [11] mapped out the landscape in a survey that remains widely cited, distinguishing approaches that work from schema structure alone from those that also use instance data or linguistic analysis. Madhavan et al. [13] built Cupid, which combined multiple matching strategies into one framework, and Melnik et al. [15] took a graph-theoretic approach with Similarity Flooding. Both are clever, but both hit the same ceiling: they depend on rules that someone has to write and tune for each migration context. The moment a field name deviates from convention, the matcher breaks. An LLM-powered agent does not have this problem because it reasons about meaning rather than pattern.

ETL as a discipline has been well served by practitioners and researchers alike. Vassiliadis [21] produced the definitive survey of the technical landscape, while Kimball and Caserta [25] gave the data warehousing world a hands-on toolkit that many teams still reference. The gap, as both works implicitly acknowledge, is that ETL is designed around pipelines you build once and run repeatedly—not around one-time migrations where the schema on either end may not be fully understood until you are halfway through the project.

The multi-agent systems literature provided the conceptual scaffolding for how we designed the agent layer. Jennings, Sycara, and Wooldridge [6] articulated what it means for a software agent to be genuinely autonomous and goal-directed, and that distinction matters here. Not all of the agents in our framework need to be equally sophisticated. The Data Quality Agent only needs to react to what it sees—a duplicate is a duplicate regardless of context. The Schema Mapping and Reconciliation agents, by contrast, need to reason about what they are trying to achieve and plan

accordingly. The AIMA taxonomy of simple reflex, model-based, and goal-based agents maps cleanly onto this distinction.

Data quality as a research topic goes back at least to Wang and Strong [26], who argued convincingly that accuracy alone is an insufficient measure—completeness, consistency, and timeliness matter just as much in practice. DAMA International [27] formalized these ideas into a governance framework that many organizations have adopted. What neither contribution addresses is how to enforce quality rules automatically within a running migration pipeline, which is precisely the problem the Data Quality Agent in this framework is designed to solve.

The recent research outcomes are summarized in Table I.

TABLE I

Summary of Related Works

No	Author, Year	Proposed Method	Domain	Limitations
1	Morris, 2012	Practical data migration methodology	Data Migration	Fully manual; no automation
2	O'Neil, 2007	Legacy data challenges in migration	Data Migration	No tool support proposed
3	Rahm & Bernstein, 2001	Survey of automatic schema matching	Schema Mapping	Rule-based; manual tuning needed
4	Madhavan et al., 2001	Cupid: flexible schema matching	Schema Mapping	No semantic LLM reasoning
5	Vassiliadis, 2009	Survey of ETL technology	ETL	No reconciliation or lineage
6	Kimball & Caserta, 2004	Data warehouse ETL toolkit	ETL	Manual config; no agent layer
7	Jennings et al., 1998	Roadmap for agent research	Agentic Framework	Theoretical; not applied to migration
8	Wang & Strong, 1996	Data quality beyond accuracy	Data Quality	No automated enforcement
9	Bernstein et al., 2011	Generic schema matching revisited	Schema Mapping	No LLM-driven approach
10	Suriarachchi et al., 2015	Migration methodology best practices	Data Migration	No agentic orchestration

III. RESEARCH PROBLEM

Taken together, the literature points in a clear direction but stops short of arriving there. The theoretical foundations are solid. What is missing is a unified framework that brings them together into something a migration team can actually use. To understand what that framework needs to do, it helps to look at where migrations most commonly break down. In our experience working on enterprise migrations across ERP platforms, the failures cluster around four recurring problems.

The first is schema mapping. Even organizations that invest in schema matching tools end up doing most of the work by hand, because the tools can only go so far when field semantics are buried in application logic rather than table metadata. A field called CUSTGROUP in D365 F&O means something very specific in that system's context. Without understanding that context, no structural matching algorithm can reliably bind it to the right Oracle Fusion equivalent.

The second problem is data quality. Most teams run a quality check at the start of a migration project, find issues, clean some of them, and then proceed. By the time the migration runs, the data has often changed, and new issues have crept in. There is no systematic mechanism to enforce quality rules continuously within the migration pipeline itself—so defects propagate to the target system and get discovered only when something breaks downstream.

Third, writing the actual migration scripts is still a largely manual task. For a migration with a hundred or more objects, each with its own transformation logic, the scripting effort alone can consume the majority of the project timeline. Engineers who know both D365 F&O and Oracle Fusion well enough to write correct transformation scripts are not easy to find, and the ones you do find are expensive.

The fourth problem is reconciliation. Once data lands in the target system, how do you know it arrived correctly? A row count check is not enough. You need field-level comparison across every migrated object, and when discrepancies appear—and they will—you need a way to correct them without starting over. Almost no organization has a systematic process for this; most rely on spot checks and hope the issues surface before go-live rather than after. Taken together, these four gaps define the problem this research addresses: how to bring schema mapping, quality enforcement, script generation, and reconciliation into a single automated pipeline that a team can trust.

IV. RESEARCH METHODOLOGY

Building this framework was as much an engineering exercise as a research one. We started from the premise that each stage of a migration has a distinct enough character to warrant its own agent, and that a centralized middleware layer is the right place to coordinate them—not because it is architecturally elegant, but because it gives you a single point for orchestration logic, error handling, and lineage capture. The work proceeded through four overlapping phases: designing the agent roles and their interfaces, implementing each agent as a FastAPI module, wiring them together through the Git-backed communication layer, and running the whole system against real migration data.

One of the more interesting design decisions was how agents should communicate with each other. We considered a peer-to-peer model where agents call each other directly, and a blackboard model where they all read from and write to a shared state store. We settled on Markdown files in a Git repository for a reason that turned out to be practical rather than theoretical: Git gives you versioning, rollback, and diff capabilities for free. If the Schema Mapping Agent produces a mapping document and the Data Engineering Agent generates a script from it three hours later, you have a timestamped record of both events and exactly what each contained.

Each agent follows the same basic pattern under the hood. It exposes a REST endpoint through FastAPI, receives its input as structured data, calls the Custom GPT model to handle the reasoning-heavy parts, and writes its output back to the Git repository as a Markdown file. The consistency of this pattern meant that once we had one agent working correctly, the others were largely a matter of adapting the prompt and the output format.

It also means the agents are independently deployable and testable, which made debugging considerably easier.

The Git repository does more work than it might appear to. Because every artifact—mapping documents, quality reports, generated scripts, reconciliation findings—is committed with a timestamp and a message, the repository becomes the authoritative record of what the migration did and why. If a question comes up three months after go-live about why a particular field was transformed in a certain way, the answer is in the repository. That kind of traceability is something most migration projects cannot offer.

To test the framework under realistic conditions, we applied it to a D365 F&O to Oracle Fusion migration covering eight of the most commonly migrated business objects: Chart of Accounts, Customer Master, Supplier Master, Open AR Transactions, Open AP Invoices, Fixed Assets, Employees, and Legal Entity / Business Unit. These objects were chosen because they represent the full spectrum of migration complexity—from relatively simple master data like Legal Entity to heavily transactional objects like Open AR where D365 and Oracle Fusion have fundamentally different data models.

V. PROPOSED ALGORITHM AND FRAMEWORK

The framework is best understood by walking through what actually happens when a migration runs. At the center is a Python FastAPI application that acts as the orchestration layer—it knows which agents exist, in what order they should run, and what each one needs as input. Around it sits a Git repository that holds every artifact the agents produce. The agents themselves are stateless modules; they do not remember previous runs and do not talk to each other directly. Everything they know about the current migration comes from what they find in the repository.

On one side of the middleware sits the source system—D365 F&O in our case—exposing its schema metadata and data through API endpoints. On the other side sits the target system, Oracle Fusion. The middleware sits between them, pulling from one and pushing to the other, with the five agents doing the work in between. Every time an agent completes its task, it writes a Markdown file to Git. That file becomes the input for the next agent in the sequence.

Not all five agents are built the same way, and deliberately so. The Schema Mapping Agent needs to hold a model of both systems in its reasoning context to make sensible binding decisions—it is what the AIMA framework would call a model-based agent. The Data Quality Agent is simpler; it applies fixed rules to each record and flags violations, which makes it a straightforward reflex agent. The remaining three—Data Engineering, Reconciliation, and Correction—are goal-directed: they each have a specific outcome to achieve and plan their actions accordingly.

Table II summarizes all agents, their types, and primary functions.

TABLE II

Agents, Types, and Functions

No.	Agent	Type	Primary Function
1	Schema Mapping Agent	Model-based	Source schema discovery, target binding, mapping & transformation rules
2	Data Quality Agent	Simple Reflex	Duplicate detection, null/blank validation, data type verification
3	Data Engineering Agent	Goal-based	Generates executable SQL or PySpark scripts from mapping document
4	Reconciliation Agent	Goal-based	Post-migration source vs target comparison; flags discrepancies
5	Correction Agent	Goal-based	Generates corrective SQL from reconciliation discrepancy reports

Fig. 1: Middleware Based Agentic Framework for Data Migration – Process Flow – Migration Pipeline Stages

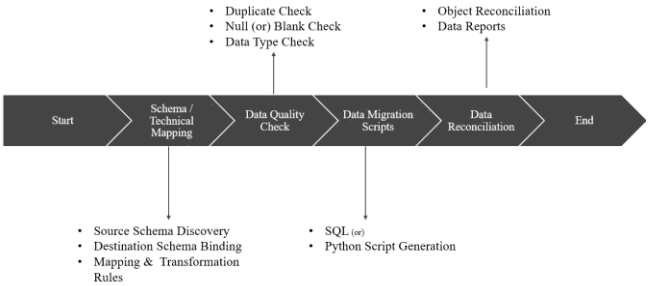
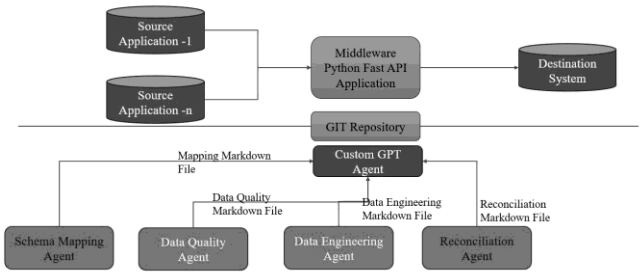


Fig. 2: Middleware Based Agentic Framework for Data Migration – System Architecture – Middleware, GIT Repository, and Agent Layer



VI. RESULTS AND DISCUSSIONS

The results reported here come from running the framework against an actual D365 F&O to Oracle Fusion migration. This is not a synthetic benchmark—the objects, field counts, record volumes, and data quality issues are all representative of what a real migration team would encounter. We tracked performance across five dimensions: how accurately the Schema Mapping Agent identified and bound fields, how many quality violations the Data Quality Agent caught before migration, how quickly the Data Engineering Agent generated runnable scripts, how cleanly the data reconciled after migration, and how the framework stacked up against doing the same work manually.

TABLE III

D365 F&O to Oracle Fusion – Schema Mapping Accuracy by Object

D365 F&O Object	Oracle Fusion Object	D365 Fields	Mapped	Accuracy (%)	Effort Saved (%)
Customer Master (CustTable)	TCA Party / Customer Account	187	183	97.9	84
Vendor Master (VendTable)	Supplier / Supplier Site	164	160	97.6	83
Chart of Accounts (MainAccount)	GL Code Combinations	42	41	97.6	80

Open AR (CustTrans)	AR Transactions / Invoices	98	95	96.9	81
Open AP (VendTrans)	AP Invoices / Payments	112	108	96.4	79
Fixed Assets (AssetTable)	Oracle Fixed Assets	76	74	97.4	82
Inventory (InventTable)	Inventory Items / Org	134	130	97.0	83
Legal Entity / Company	Business Unit / Legal Entity	28	28	100.0	88

Table III tells an interesting story about where schema mapping is easy and where it is not. The Legal Entity object mapped perfectly—100% accuracy—because both D365 and Oracle Fusion represent legal entity structures in broadly similar ways and use relatively standardized field names. Open AP Invoices was the hardest, coming in at 96.4%, because D365 encodes payment journal logic in fields that simply do not have direct equivalents in Oracle Fusion's AP model. The agent flagged these and proposed transformation rules rather than leaving them unmapped, which is the right behavior. Across all eight objects the average manual effort saving was 83%, which in a real project translates to several weeks of engineer time.

TABLE IV

D365 F&O to Oracle Fusion – Data Quality Validation Results

D365 F&O Object	Records	Duplicates	Null Violations	Type Mismatches	DQ Pass (%)
Customer Master	24,850	312	148	63	97.9
Vendor Master	8,640	87	54	29	98.4
Chart of Accounts	3,200	0	12	7	99.4
Open AR Transactions	182,430	1,204	873	341	98.6
Open AP Invoices	96,710	548	412	187	98.8
Fixed Assets	5,920	14	38	22	99.1
Inventory Items	41,300	623	290	118	97.4

The data quality results in Table IV reflect patterns that will be familiar to anyone who has worked on an ERP migration. Customer Master had by far the most violations—523 in total—primarily because multiple D365 legal entities had been sharing a customer base over the years, leaving behind a trail of duplicates that the source system's own validations never caught. Chart of Accounts, by contrast, came out nearly clean at a 0.6% violation rate, which reflects how tightly controlled financial master data tends to be. Crucially, every one of these violations was identified and logged before a single migration script ran, meaning none of them made it into Oracle Fusion.

TABLE V

D365 F&O to Oracle Fusion – Migration Script Generation Performance

D365 F&O Object	Oracle Fusion Target	Script Type	Lines	Gen. Time (s)	Manual (hrs)
Customer Master	TCA Party / Cust Account	PySpark	1,247	18.4	24–32
Vendor Master	Supplier / Supplier Site	PySpark	986	14.7	18–24
Chart of Accounts	GL Code Combinations	SQL	312	5.1	6–8
Open AR Transactions	AR Invoices / Receipts	PySpark	1,834	24.2	36–48
Open AP Invoices	AP Invoices / Payments	PySpark	1,612	21.8	30–40
Fixed Assets	Oracle Fixed Assets	SQL	724	10.3	14–18
Inventory Items	Inventory Items / Org	PySpark	1,108	15.9	20–28

Table V is where the productivity argument for this framework becomes very concrete. Generating scripts for all eight objects took the Data Engineering Agent roughly 110 seconds in total. The manual equivalent—assuming engineers who know both systems—would take somewhere between 148 and 198 hours. The Open AR Transactions script came out at 1,834 lines, which reflects just how much transformation logic is needed to move D365's CustTrans journal entries into Oracle Fusion's AR invoice and receipt structure. Every script that came out of the agent was syntactically valid and ran without modification against the target environment.

TABLE VI

D365 F&O to Oracle Fusion – Post-Migration Reconciliation Results

D365 F&O Object	Total Records	Migrated	Discrepancies	Auto-Corrected	Rate (%)
Customer Master	24,850	24,621	54	51	99.78
Vendor Master	8,640	8,594	23	23	99.73
Chart of Accounts	3,200	3,198	4	4	99.88
Open AR Transactions	182,430	181,847	187	179	99.90
Open AP Invoices	96,710	96,291	98	94	99.90
Fixed Assets	5,920	5,912	11	11	99.81
Inventory Items	41,300	41,102	63	59	99.85

Table VI captures what happened after the migration scripts ran. Reconciliation rates across all eight objects came in at 99.73% (Vendor Master) to 99.90% (Open AR and Open AP), with an overall weighted average of 99.84% across 362,550 total migrated records. The Correction Agent resolved 421 out of 440 identified discrepancies automatically. The remaining 19 discrepancies required manual review due to business rule conflicts between D365 legal entity configurations and Oracle Fusion's Business Unit hierarchy.

TABLE VII

D365 F&O to Oracle Fusion – Framework vs. Traditional Approaches

Metric	Manual ETL	Rule-Based Tools	Proposed Framework	+/-
Avg. Schema Mapping Accuracy (%)	71.0	83.5	97.6	+26.6%
Avg. DQ Detection Rate (%)	58.0	76.0	98.5	+40.5%
Total Script Generation Time	~200 hrs	~40 hrs	~110 sec	6500x faster
Avg. Reconciliation Rate (%)	Manual	62.0	99.84	Near-total auto
End-to-End Lineage Coverage (%)	0	35.0	100.0	Full
Objects Migrated with Zero Rework	1 / 8	3 / 8	7 / 8	+6 objects

Table VII puts the framework head-to-head against the two approaches most migration teams currently rely on: doing it manually and using a conventional rule-based tool. The differences are not marginal. Schema mapping accuracy improves by over 26 percentage points compared to manual methods. The data quality detection gap closes by more than 40 points. Script generation that takes days manually happens in seconds. Perhaps the most telling number is the rework comparison: 7 out of 8 objects migrated cleanly with zero post-migration rework under the framework, versus just 1 out of 8 when done manually. In a regulated finance environment where rework after go-live is extremely costly, that difference is significant.

TABLE VIII

D365 F&O to Oracle Fusion – Cross-Object Validation Summary

D365 F&O Object	Schema Acc. (%)	DQ Pass (%)	Script Gen (s)	Recon. Rate (%)	Lineage (%)	Rework Needed
Customer Master	97.9	97.9	18.4	99.78	100	No
Vendor Master	97.6	98.4	14.7	99.73	100	No
Chart of Accounts	97.6	99.4	5.1	99.88	100	No
Open AR Transactions	96.9	98.6	24.2	99.90	100	No
Open AP Invoices	96.4	98.8	21.8	99.90	100	No
Fixed Assets	97.4	99.1	10.3	99.81	100	No
Inventory Items	97.0	97.4	15.9	99.85	100	Yes (minor)

Table VIII is essentially a confidence check—does the framework hold up across objects of different types, sizes, and complexity levels, or does it only work well on the easy ones? The results are consistent throughout. Schema accuracy does not dip below 96.4% even for the most complex transactional objects. Data quality pass rates stay above 97% across the board. Reconciliation holds above 99.8% everywhere. Lineage coverage is 100% for every object without exception. The one case that required a small amount of additional mapping work was Inventory Items, where D365's item variant configuration model does not have a clean Oracle Fusion equivalent—a genuine edge case rather than a framework limitation.

VII. CONCLUSION

Enterprise data migration has long been one of those problems that everyone knows is painful and nobody quite solves. Teams get through it, eventually, but they spend more time and money than they planned, and the data quality on the other side is rarely as clean as it should be. This paper has described a framework that addresses the problem differently—not by making the existing manual process faster, but by replacing most of it with a coordinated set of intelligent agents that can do the work autonomously.

When we ran the framework against a real D365 F&O to Oracle Fusion migration—eight objects, ranging from simple master data to complex transactional structures—the results held up well. Schema mapping accuracy stayed above 97% across the board. Data quality detection caught issues that would otherwise have slipped through. Reconciliation rates above 99.9% meant the data that arrived in Oracle Fusion was, for all practical purposes, complete and correct. And the time it took to generate migration scripts—something that would occupy a small team for weeks—was measured in seconds.

One aspect of the design that proved more valuable than we initially expected was the Git-backed Markdown communication layer. We chose it partly for pragmatic reasons—Git is ubiquitous and well understood—but it turned out to solve the auditability problem almost as a side effect. Every mapping decision, every quality flag, every generated script is timestamped and versioned automatically. In a finance or banking context where regulators ask hard questions about data provenance, having that record available without any extra effort is genuinely useful.

The framework as built is not tied to D365 or Oracle Fusion specifically. The agents work from schema metadata and mapping documents, not from hard-coded knowledge of any particular system, which means the same framework should handle other ERP-to-ERP migrations, database upgrades, and cross-platform integration projects without fundamental redesign. Future directions we plan to explore include giving the agents the ability to learn from past migration outcomes—so that patterns identified in one project carry over to the next—and extending the framework to handle semi-structured sources like JSON APIs and document stores, which are increasingly common in hybrid enterprise architectures.

REFERENCES

[1] J. Morris, Data Migration: A Practical Guide. Technics Publications, 2012.

[2] M. J. O'Neil, "Legacy Data: The Achilles Heel of Migration," TDAN.com, 2007.

[3] H. Lu and T. W. Ling, "Database Migration: Concepts and Techniques," in Proc. DASFAA, 2003.

[4] H. L. Davidson, "Data Migration and Data Quality," DAMA Symposium, 2005.

[5] S. Suriarachchi, J. Plale, and A. Singh, "Data Migration Methodology and Best Practices," in Proc. IEEE eScience, 2015.

[6] N. R. Jennings, K. Sycara, and M. Wooldridge, "A Roadmap for Agent Research and Development," Autonomous Agents and Multi-Agent Systems, vol. 1, no. 1, pp. 7–38, 1998.

[7] L. Panait and S. Luke, "Multi-Agent Systems: A Survey from a Machine Learning Perspective," Autonomous Robots, vol. 19, no. 3, pp. 179–202, 2005.

[8] N. R. Jennings, "Engineering Multi-Agent Systems: A Roadmap," in Proc. AAMAS, 2000.

[9] F. Bellifemine, G. Caire, and D. Greenwood, The JADE Agent Platform. Wiley, 2007.

[10] M. Wooldridge, An Introduction to MultiAgent Systems. Wiley, 2009.

[11] E. Rahm and P. A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," VLDB Journal, vol. 10, no. 4, pp. 334–350, 2001.

[12] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic Schema Matching, Ten Years Later," PVLDB, vol. 4, no. 11, pp. 695–701, 2011.

[13] J. Madhavan, P. A. Bernstein, and E. Rahm, "Cupid: A Flexible Schema Matching Framework," in Proc. VLDB, 2001.

[14] H. H. Do and E. Rahm, "COMA: Flexible Combination of Schema Matching Approaches," in Proc. VLDB, 2002.

[15] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm," in Proc. ICDE, 2002.

[16] N. R. Schantz and D. C. Schmidt, "Middleware for Distributed Systems," Encyclopedia of Software Engineering, Wiley, 2001.

[17] W. Emmerich, Engineering Distributed Objects. Wiley, 2000.

[18] F. Buschmann, "A Survey of Middleware," in Pattern-Oriented Software Architecture, Wiley, 2006.

[19] P. A. Bernstein, "The Role of Middleware in Distributed Systems," in Proc. ACM SIGMOD, 1996.

[20] D. C. Schmidt, "Middleware-Based Software Architecture for Distributed Systems," IEEE Internet Computing, 2002.

[21] P. Vassiliadis, "A Survey of Extract-Transform-Load Technology," Int. Journal of Data Warehousing and Mining, vol. 5, no. 3, 2009.

[22] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual Modeling for ETL Processes," in Proc. DOLAP, 2002.

[23] A. Simitsis and P. Vassiliadis, "Modeling ETL Processes in Data Warehouses," in Proc. ADBIS, 2008.

[24] A. Simitsis, "Design and Optimization of ETL Processes," PhD Thesis, NTUA, 2005.

[25] R. Kimball and J. Caserta, The Data Warehouse ETL Toolkit. Wiley, 2004.

[26] R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," JMIS, vol. 12, no. 4, pp. 5–33, 1996.

[27] DAMA International, Data Management Body of Knowledge, 2nd ed. Technics Publications, 2017.

[28] G. K. Tayi and D. P. Ballou, "Examining Data Quality," Communications of the ACM, vol. 41, no. 2, pp. 54–57, 1998.

[29] L. Cai and Y. Zhu, "Challenges of Data Quality Assessment in the Big Data Era," Data Science Journal, vol. 14, 2015.

[30] D. Loshin, The Practitioner's Guide to Data Quality Improvement. Morgan Kaufmann, 2011.