Enterprise Transformation Projects-A Generic Data Storage Concept-The Implementation (GDSCI)

ANTOINE TRAD
Transformation Projects
IBISTM
Paris La-Defense
FRANCE

Abstract: Data and Data-STorages -Base (DST) related technologies support and enable the implementation of common, technical, and business requirements, transformational-changes' activities, and enterprise-wide evolution. This evolution is a long-term and well-planned evolution that has to predict major risks. These major transformation-initiated disruptions need a GDSCI [1,36], because most enterprises and organizations (simply Entity) are DST-Centric (DSTC), but such Entities use and apply different types of DSTs. Therefore, Entities must implement Entity transformation projects (simply Project) that enable profound changes and at the same time abstract by unifying of DSTs and their DataSets (DS). A GDSCI depends on and results from a robust and performant Digital-Transformation's (DT) phase (known as Digitization) that depends on the hyper-evolution of various types of technologies, employees' Polymathical-skills, and clients' demands. Projects are very complex and their implementation-teams must be capable of implementing a GDSCI that results in a concrete and sustainable DSTC and GDSCI. The GDSCI offers a set of applicable-DST management-patterns and building blocks that are used by the Project and is also used for Day-to-Day (D2D) Automated Development and Operations (ADevOps) activities, like managing DST Models (DSTM) [2]. As the GDSCI and its implementation (the GDSCI) are critical for Projects and Digitization, there is a need to use existing legacy components and to offer workable-patterns to interface external modules. This article will try to present such DST-management patterns and show how they can be implemented using the latest avant-garde technologies and an adequate Enterprise Architecture Models (EAM) and approaches.

Key-Words: Data-Storages, Data-Bases, DSTC implementations, GDSCI, Mathematical-Models, Artificial Intelligence, Business and Common Transformation Projects, Weightings-Concepts, Polymathical Information Systems, Meta-Models, Enterprise Architecture, Enterprise Agile-Methodologies, Organizational (re)engineering, Factors, and Indicators.

Received: April 11, 2024. Revised: December 22, 2024. Accepted: January 16, 2024. Published: April 9, 2025.

1 Introduction

Dynamic Entities and competition force them to change and transform their Distributed Communication Systems (DICS) fast and adapt to new challenges and realities, where there is the need to completely (re)think their DICSs, GDSCI, DSTs, Project-based strategies, business models, (re)structure their organizational-models, review

working models, and transform their information-platforms, and design models concepts. Projects are very complex, because of the lack of structured-generic approaches, heterogeneous Information and DICS modules and environment(s), various types of (in)compatible DSTs systems, fast and unnecessary DICS hyper-evolution, chaotic combination and relations of different APplication Domains (APD), and the lack of adoption of Polymathic-concepts, like the GDSC4AI [1,36], and GDSCI which and use a DST-first-approach [39,40]. DST-first-approach is also compatible with Code-first-approach, and Polymathics privilege interdisciplinary approaches for Projects' and GDSC4AI's implementations, which in this work is based on the GDSCI and Mathematical Models (MM). The Project and GDSCI use (the author's) Applied-Holistic and Polymathical MM (AHMM) for GDSCI (AHMM4GDSCI), which is mainly used for DSTs' and GDSCI's modeling and integrity-validity checks.

The AHMM4GDSCI supports the Polymathic Enterprise MetaModel (PEMM4D) for DST (PEMM4D), Projects, and common modules. In this section the author makes a summary of the RDP and the implemented sections of the Internally Implemented (II) Polymathical Transformation (IIPT) Framework (IIPTF) and the previously developed modules. II is also referred to an In-House Implementation (IHI).

1.1 The IIPTF and its Parts

The IIPT Sections (IIPTS) are the following (shown in Fig. 1): 1) Is implemented to support Projects; 2) It uses a real-world IIPT Platform (IIPTP); 3) It is described using the IIPT Case (IIPTC); 4) It offers sets of Blocks (like patterns) in the context of the II IIPT Blocks and (compound) Patterns (IIPTB); 5) It offers the IIPT Dictionary (IIPTD); and 6) It offers a concrete implemented IIPT Environment (IIPTE).

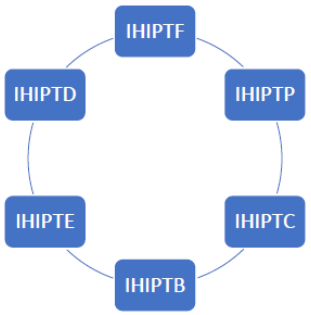

Fig. 1. The IIPTS.

A Project needs the IIPTS that includes:

- The IIPTB delivers sets of Blocks (like patterns), and standard EA methodologies artefacts, like The Open Group's (TOG) Architecture-Framework (TOGAF).
- The IIPTC describes a Project's typical Applied Case-Study (ACS) that can be in Conceptual Proof of Concept (CPoC).
- The IIPTD is a basic Project-dictionary.
- The IIPTE offers a real-world implemented Environment (that is implemented using Microsoft .NET).
- The IIPTF manages the Project and abstracts various DICS-technologies and related methodologies.
- The IIPTP presents the Project's heterogeneous DICS-platform(s) and infrastructure.

For the GDSCI all the mentioned sections need to be implemented.

1.2 The GDSCI and the IIPTB

The IIPTBs usage depends on the orientation and the approach that was chosen, like:

- GDSCI (or the DST and DST-first-approach) and DST Compound Patterns (DSTCP) are this article's another important focus.
- DSTCPs contain needed DST-access or repository-patterns.
- Adopts the Code-first-approach that automates DST's manipulations and modifications.
- The classical and recommended IIPTF Methodology (IIPTFM), CModels, EA Methods, and Design-first-approach are automated and refined.
- ….

For the GDSCI, CModels (and especially ERMs and UMLs) are developed to make them available for EA-Specialists.

1.3 Disassembling, Blocks, IIPTFM, and Views

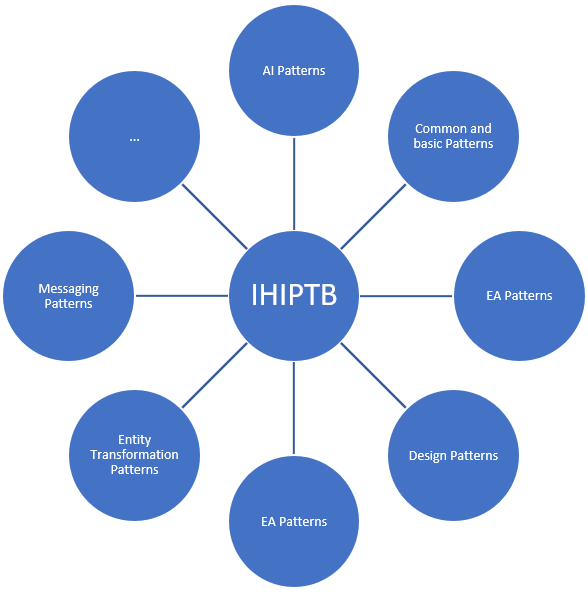

Fig. 2. IIPTB's categories of Project patterns.

Disassembling processes deliver Building Blocks (BB), Composite BBs (CBB), Compound Design Patterns (CDP), and DICS' artefacts (like Services) that are the gluing substance which creates the future DICS and its major components like DSTs' interfaces and avoid the classical siloed-approaches that result in major Very High Failure Rates (VHFR). The Entity's Disassembling Processes (EDP) is a series of unbundling (or Disassembling) processes that transform and change: Monolithic (or legacy) DICS' resources, DST(s) structures, DICS' and DSTs' administration, Data-Assets/Resources, Applications (and related Services), Business Processes (BP) Modelling (BPM), and Interaction (internal/external) scenarios. An EDP shown in Fig. 2, generates repositories of reusable/heterogenous CBBs which are applied for modelling Architectural BBs (ABB).

EDPs encounter problems, complexities, and resistances when interacting with various implementation and transformation components like GAP Evaluations (GAPE), Polymathic/Rating, and Weighting-Concept (PRWC)... [35]. The IIPTF, GDSCI, and GDSCI use the IIPTF, IIPTFM, and the Polymathic Transformation Development Method (PTDM) to synchronize and coordinate CBBs, generated-Blocks, and patterns that are applied for implementing APD's architecture, modeling, and implementation activities. The IIPTFM supports CPDs to be used by the Project and standard methodologies, like TOGAF, Unified Modelling Language (UML), Domain-Driven Design (DDD), Entity Relational Modelling (ERM)… There is a critical need to align various types of methodologies and frameworks [28], where the Project results in a pool of DSTCPs and common-patterns; these patterns can have the following views: Static, Methodological, or Dynamic [1], as shown in Fig. 2. A successful Project results in a pool of Blocks that are the base of Ready to Transform (R2T) patterns [32,33,36]. The IIPTB includes the description of Project's (and Entity's) used categories of CPDs:

- Basic Standard-Patterns.
- PEMM4D-Patterns.
- DST and Data-Management Patterns.
- API-Patterns.
- AI-Patterns.
- CPDs' Interfaces.
- Project Transformation-Patterns.
- …

For the GDSCI, CPDs are modeled, designed, and implemented (this is considered as the methodological view). In this view and phase a DST Oriented Integrated Development Environment (DOIDE), the DOIDE includes libraries and DST modules for a GDSCI Oriented Framework (GDSCIOF).

1.4 Using the GDSCIOF

The GDSCIOF like Microsoft's Entity Framework (MEF) includes and supports [42]:

- The Dynamic Object Relational Mapping (DORM) enables developers to work with Relational DSTs (RDST) using APD-specific classes/objects, as shown in Fig. 3.
- Maps to the central Distributed Model View Control (DMVC) pattern.
- Reduces sources-code's volume(s).
- Is based on Microsoft's ADO.NET which supports DST-oriented software applications and requirements' mapping-mechanisms.
- Abstracts the applied DST (rows and columns) in which data are persisted.
- Uses dynamic-concepts (Give-life) to models: APD-model, Logica- model, and Physical-model.
- The Code-first-approach, applies a Data Conceptual-Model (DCM) which is mapped to a DST-models found in Project's source-files.
- Interfaces Project's DST-Modelling environments, the applied DCM, the data-persistence-model, and needed maps. These maps are implemented using eXtensible Markup-Language (XML) Schemas (XMLS).
- The Tuple Data Model's case-environments generate dynamic In-memory DataSet (IDS)/data-types (like a class) which are founded on the DCM.
- Using the Mapping-Specification-Language (MSL) that is used to map data-storage-elements and DCMs.
- Mapping types (or a class) to IDS/data that reflect the RDST's table structures.
- Enables the access/change DST's entity data.
- Enables the interaction between DICS' client IDS-Providers and connections, to convert DCM-Queries to data-sources Queries, and outputs usable IDS.
- Entity Framework Architectural Diagram that includes data-providers, links to various types of patterns like infrastructure-patterns…

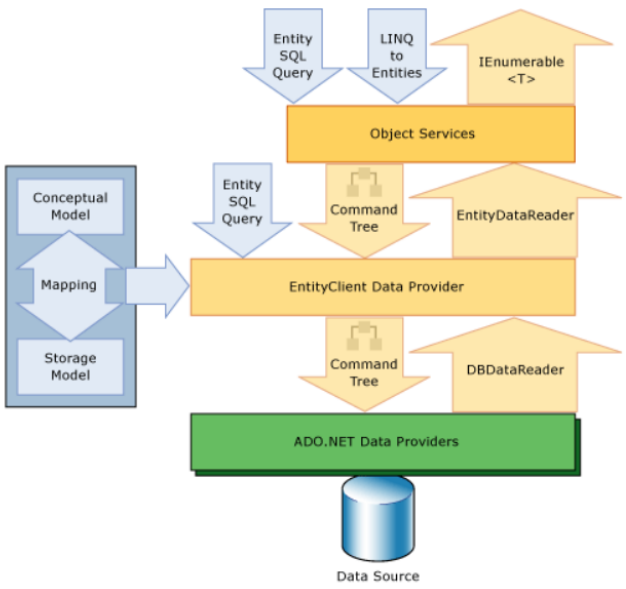

Fig. 3. MEF's components [42].

For the GDSCI the MEF is used as a layer for all DST and DSTCPs implementations.

1.5 The Infrastructure and Other Support-Patterns

The needed infrastructural-patterns include the following sets of CDPs to wrap:

- Operating Systems (OS) integration and management, like Unix (and Linux), Windows, Windows System for Linux (WSL), Containers...
- A Cloud platform components' interfacing mechanism.
- Messaging frameworks, like Kafka.
- …

The Project must establish a GDSCI and has to check its feasibility and analysis of the platform's integration [1].

1.6 GDSCI'S Outcomes

This section presents GDSC4AI's analysis outcomes [1,36], in which CSAs' Data-Objects Weighting and Rating Enumerator (CDOWRE) ranges-limits is shown in Fig. 4.

CTWRE Label	Limit's Value
Proven, Mature	9.01-10.00
Possible, Feasible	8.51-9.00
Risky	8.01-8.50
Complex	7.01-8.00
VeryComplex	5.01-7.00
Impossible	0.00-5.00

Fig. 4. The CDOWRE's values.

The most important outcomes were [1,13,29,36]:

- Modules like the PRWC, GAPE, GDSCI, were used.
- The transformed DICS improves data-quality and Factors' evaluations.
- Using a previous professional Project in which an international insurance company: transformed the legacy DICS into an XML-based Object Mapping System (OMS).
- A third and actual ACS is related to the transformation of the legacy system using GDSCI, DSTCPs, and other types of CDPs; which can be considered as a P4TT-based project. P4TT is DST-centric and was done for the Ministry of Education in the European Union.
- A CDP ensures that multiple DICS components work together to perform a unique task.
- Processing and estimating CSA_DTs, applies that the GDSC4AI uses Intelligence (and PRWC/Factors) whose results are shown in Table 1, and using the CSA_DT's CDOWRE that is shown in Fig. 1.
- Selected Factors were related to a targeted node of the HDT.
- Table 1 presents GDSCI's Phase 1 results that propose that it is "Feasible".
- Uses relationships that link the GDSCI, requirements, Blocks, Factors, and Global Unique IDentifiers (GUID).
- Phase 2 or the "Solving a Concrete Problem-type" phase showed how to solve a real-world problem-type.

For the GDSCI and GDSC4AI, the CSA_DTs showed that DST and DSTCPs implementations are "Feasible". And it is recommended to refer to the GDSC4AI work [1] for more information on the mentioned phases.

CSA Category of CSFs/KPIs	Transformation Capability	Average Result	Table
The RDP's Integration	Mature	From 1 to 10, 9.0	1
Setup and IHIPTF	Feasible	From 1 to 10, 9.0	2
Critical and Standard Topics	Risky	From 1 to 10, 8.25	3
DAMC	Feasible	From 1 to 10, 9.0	4
CModels, EA, and DPs	Feasible	From 1 to 10, 9.10	5
The Roles of Enterprise Transformational Patterns	Feasible	From 1 to 10, 8.60	6
Phase's 1 Outcome	Feasible	From 1 to 10, 8.8	7

Evaluate First Phase

Table 1. The RDP's outcome is (rounded) 8.80 [1,36].

1.7 Implementation Phases and Categories

GDSCI's main phases and actions are [1,36]:

- Finalizing EDPs that deliver CBBs, Blocks, basic CModels...
- Implementing DSTCPs' Catalogues, Dictionaries, and repository of Blocks.
- Integrating DST/Data-Modelling to generate CModels, by using methodologies (and IIPTFM).
- Integrating embedded DICS/system data sources, which are block box tools.
- Interface and integrate CDPs like AI patterns.
- Implements a pseudo Strangler-pattern.
- Apply model-transformation DPs.
- Integrate DSTCPs with other CDPs.

For the GDSCI the repository of CBBs and CModels is ready.

2 GDSCI'S Implementation

2.1 Basic and Previous Implementations

The Project adopts and II which is an Anti Locked-in Strategy (ALS) tries to define a GDSCI based concepts, strategies, and defined goals to support the Entity's activities related to business, DST/data-management, and DICS' autonomy from commercial solutions. Therefor there is a need to respect market-standards and to try to use its own loose coupled ALS DSTCPs; which is basis for Project's patterns. A Project has to try avoid the usage of various redundant standards and conventions, and has to try to build its II unique CModels and EA approach in order to support its long-term ALS. The Project-team selects the applied set(s) of main and necessary patterns' types, to develop DSTCPs. That invokes various Project's critical-requirements, and offers implementation's generic-concept that is based on patterns (and types) and are classified in categorized-groups. As already mentioned, the GDSCI and GDSC4AI (and related modules) were implemented in previous professional-projects like [29,36]:

- Patterns for Transformation Technics (P4TT)-DSTC Centric Patterns.
- Building an XML based OMS.
- Various Projects that used Object DORM Framework (DORMF) like MEF.
- And many others.

All these implementations are RDST and DSTCPs based; and support II and ALS implementations, solutions, and concepts which can be iteratively changed/transformed in the Entity's internal (or external) DICS, without the need to use colossal-investments in buying and trying to integrate external commercial-products. RDSTs and GDSCI support Projects, because RDSTs are used in all DICS DST-accesses, operations, and subsystems. RDST's evolutionary architecture is assisted by Real-time (RT) traceability, which is a real challenge for Projects and its team; and in the same time traceability is used for D2D-operations. For such Projects and transformation-phases a possible solution would be to implement a PEMM4D-based RDST's architecture. Such and architecture is generic and evolutive, uses patterns which maintains DICS' Blocks traceability; in fact the Blocks that are involved in RDST's execution, maintenance, and

development processes.

RDST's architecture is generic and independent of specific modelling-techniques and it includes a specialized component(s) that are devoted to persisting and translating conceptual schemas to logical-schemas. These DICS' components, various types of modifications that are done to conceptual-schemas, and are traced, without the need to (re)generate original-schemas to go back to initial-stages. And afterwards to propagate these modifications to the physical and extensional-levels [10]. RDSTs contain the needed meta-data/information, structures, integrity-checking mechanisms, PEMM4D's relations, usage of an AHMM construct, and the definition of the Project's vision and the related levels-of-granularity [1,35].

2.2 Project's Vision and Levels of Granularity

The Project's needed vision and levels-of-granularity imply that [30,31].

- The need to apply an atomic BB architecture that adapts easily to actual dynamic DICSs, DSTs, and supports Entities' business and operational activities.
- Fierce competition needs a loosely-interconnected GDSCI that operates in a wide-networked business-environment.
- An Entity needs a robust DICS and DST system, in order to ensure its sustainable business-goals and critical-transactions.
- GDSCI based transactions adapt easily to very frequent business (and common) transformation's change-processes.
- Levelling to such cases Blocks-based solutions support GDSC4AI and GDSCI's actions, but they have to define the optimal levels-of-granularity.
- A BB and Blocks-based Project's goals and implementation strategies (for highly frequent changes) demands sets of GDSC4AI and GDSCI activities that support Entity's resources-management.
- The Project is decoupled in the phases of design, implementation or re-engineering.
- A fundamental architectural requirement and constraint is to have an automated and agile DICS platform that is supported by the IIPTF and which can generate CModels.
- The IIPTF and IIPTFM include various categories of patterns that can be Just-In-Time (JIT) implemented.
- The IIPTF proposes an atomic DICS architecture's vision, concept and applied sets of patterns which are of strategic value for the Project's implementation phase.
- Transformed patterns are used in a JIT-way, by using Blocks.
- The Project must implement a controlled governance concept for DICS architecture's patterns and DSTCPs; but that is a very complex process…
- The transformed repository of patterns, catalogues, and dictionaries is needed for the EDP(s).

For the GDSCI the vision and levels-of granularities are defined and implemented in CModels.

2.2 Entity's Repository of Patterns, DST-Catalogues, Dictionaries, and EDPs

Projects use CDPs like the DMVC, which is a popular and central pattern, and it offers interfaces to DST's data-models, CModels, and CBBs. The DMVC is used to decouple [20]: 1) Data-models, CModels, and the targeted APD; 2) View, phase or presentation-layer; 3) Controller of messages and

other; and 4) Interfaces REpresentational State Transfer's (REST) Create, Read, Update, and Delete, (CRUD) calls-operations. These CRUD-operations are: POST, GET, PATCH, and DELETE. CDPs abstract CModels, EA artifacts, software-engineering sources, business-engineering implementations... There are various ways to implement Holistic CDPs (HCDP) that inherit the Holistic Enterprise Architecture Pattern (HEAP), which can include used patterns to support GDSC4AI, GDSCI, and DST's data-catalogues [32,32]. Projects have various insights on siloed-data-sources and have to leverage DST's IDSs as central artefact (or resource). DST's Data-catalogs support such activities like metadata-management. DST's IDS-catalogues calls-function as an interface (and indexed) and which is searchable for DST's IDS/data; which ensures successful IDS' searches. Efficient search-functionalities are integrated into DST's data-catalogs that enable Project's data-engineers in finding requested IDSs or data-objects [18]. Such operations are complex and therefore, it is recommended to use the DORMF like the MEF which includes the following capabilities [24,32,33]:

- Schema changes that include changes to: Table name, Column types, Delete/merge columns, Using and removing views, Changing RDST's data-types, Changing DST's interfaces, and many other features.
- Linking RDSTs pointers, GUIDs, or relational-keys to Object Oriented (OO) or CModels relations, enabling software-refactoring and EDPs…
- Managing DSTs' data timestamps.
- Enables DST's data-values transformations.
- Automating tests for GDSC4AI and GDSCI's and asserting-successful orchestrations.
- Synchronization between OO Models (OOM), ERM, CModels, and class-diagrams.
- To use HCDP as a basic-construct for classifying and implementing CModels and more sophisticated patterns; and asserting a Polymathical approach for Projects' implementations.
- The HCDP is founded on precisely designed CDPs, and the patterns-models.
- The HCDP based CModel is applied as a structure for JIT CBBs' creation and the creation of different types of Projects' modules (and components).
- An HCDPs-based IIPTFM offers Blocks, like in engineering APDs, and that support flexible, and complex Projects.
- Complicates Projects, need to create a common-denominator-pattern, to integrate standard and external patterns. The HCDP is GDSCI's and GDSCI's backbone.
- Blocks, HCDPs, DSTCPs, catalogues support DST's data-modelling and to generate CModels and diagrams.

For the GDSCI the HCDPs and CModels were implemented and tested.

2.3 DST's Data-Modelling Diagrams

There are various approaches for data-modelling [24,25,26]:

- Unifying OOMs, ERMs and ER Diagrams (ERD), and class-diagrams, by using IIPTFM and generating CModels.
- An ERM and ERD describe interrelated OOM's tables and are composed of

various object formats/types and relationships.

- DSTCPs use ERMs and ERDs to represent OO artifacts for various types of HCDPs and abstract data-modeling and used interfaces.
- DSTCPs use DST/data-architecture Viewpoint that includes different conceptual views; like the Data Dissemination View (DDV).
- DDVs show relationships within the DSTCP's: 1) IDS/data classes; 2) APD IDS/data blocks; and 3) Modules' Blocks. Which enables a flexible architecture of DST/data sources.
- Diagrams like the Data-Migration-Diagram (DMD) show the inter-flow of IDS/data between DICS' various DST's IDSs and sources; it is also used to view Project's IDS/data audit and traceability maps.
- The Data-Security-Diagram (DSD) presents the Project's actors' roles needed for DST/data-transformations.
- Interfaces to embed various types of DST/data-sources.

EDPs contain rules that use calculus of refinement-processes that deliver CModels that include data-models and HCDPs like the state pattern that is used in DST activities [43]. An EDP and related refinement-processes, include integrated DST/data-sources. For the GDSCI, DST's data-models CModels were implemented and tested.

2.5 EDPs and Integrated DST's Data-Sources

CModels interface various (internal and external) modules, components, and methodologies, by using DSTCP's interface-patterns, which enables mappings to business, information, application, and infrastructure layers and artefacts/elements, like [39,40]:

- For business-interfaces actors, roles, components' processes (and functions), and events are used.
- For application-interfaces process components, like the Configuration Management System (CMS), service Knowledge Management System (SKMS), or Availability Management Information System (AMIS), are used.
- For infrastructure we have the DSTs like, the Configuration Management Databases (CMDSTs) or the Known-Error-DST (KEDST)…
- There are also interfaces like the business-objects, data-objects and DST-artefacts; which are used as interface-Blocks. And they are associated by using service-concept.
- To implement CModels frameworks can be used, like OOM, UML, or Archimate. These models can be extracted.
- GDSC4AI and GDSCI refinement-processes use existing HCDPs like the Stangler-pattern.

For the GDSCI, DST's data-sources were integrated.

2.6 The Strangler-Pattern

To avoid VHFR and very agile Projects, an evolutionary approach is optimal. A well-synchronized agile-approach is necessary for the iterative GDSC4AI and GDSCI implementations. Evolutionary approaches focus on targeted components and then refine them; which is a safe iterative replacement of legacy components. The proposed iterative and well-designed approach

offers an evolutive GDSC4AI and GDSCI based Project roadmap and plans' controls possible. Managing GDSCI problem-types, by reducing the Project's risks that are linked to frequent-changes. That in-turn promotes a value-back-approach for Entities in offering agile-delivery of innovated-features, this process is recommenced up to the complete transformation of legacy components. Ensuring that they are sufficiently mature to be used and to replace this monolith-legacy component. Such a concept is a standard concept; and was incented and applied by Chris Stevenson and Andy Pols. Another pioneer, Martin Fowler, applied the Strangler pattern and hammered it as: "*The Strangler Application"*; which strongly impacts the GDSCI. The crucial concept is to unbundle-break legacy monolithic modules into atomic of smaller chunks. Such a breakdown needs precise design and architecture activities; which it result in a light/smooth change and transformation processes and which improve Entity's sustainable business results and solve ever-changing Project problems and requirements.

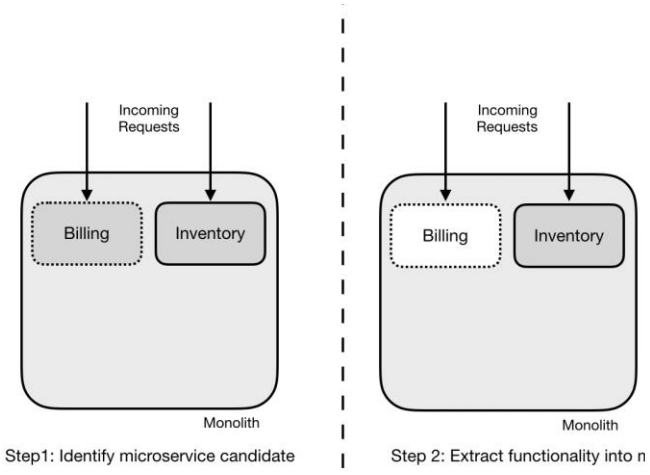

Fig. 5. The Strangler pattern [21].

Such an HCDP requires that the Project includes can unbundle-transform monolith-legacy DICS' parts and components, by [7,16,21]:

- Continuous (and gradual) evolution of monolithic modules to services architectures by applying the strangler-pattern, as shown in Fig. 5. The roles of incremental-EDPs and refactorings of DSTs, DSTs, and associated software-components, are determinants.
- EDPs extract code and DST-interfaces blocks: This approach is based on copying bits of sources-code around in the transformed component, which is the default modus. This modus or option can generate problems, issues like bugs, from the legacy DICS.
- Rewriting DICS' capability: Initially, this approach is an expensive approach compared to copying source-code, but there are advantages of rewriting by capability, which improves the Return On Investment (ROI).
- Event-tapping: Is also known as Event-interception, in which DICS' event-driven-components (or capacities) are tapped-in to the collect and stream of events. Then it begins to implement (or replaces) call-back-functions for the captured-events. This HCDP also enables the implementation of parallel-DICSs to ensure Entity's business-continuity.
- Asset-capture: Where each module or component manages a set of functional or common objects (or assets), like, user-accounts, different types of transactions, historical-records, or product-orders. The transformation DICS' capability of

managing Entity's assets independently by using the strangler-pattern-based Blocks.
- Service bubbles: Almost the majority of DICS' applications, services, and components use/consume sets of APIs. Therefore, the Project team explores EDP's chunking and refactoring of DICS' legacy monoliths into a service-architecture concept.
- The Branch-by-abstraction-pattern which was developed to solve problem-types that result from long-lasting changes on DICS' components without the Project's disruptions.

The Strangler-pattern based strategy checks the Entity's capabilities, to transform the legacy DICS and DSTs, and to create small Strangler based services. These services encapsulate the behavior and logic of all capabilities in loosely coupled CModels that use Models for Transformations (M4T) by using HCDPs (M4THCDP). For the GDSCI, the Stangler-pattern was implemented and is assisted by the M4THCDP.

2.7 The M4THCDP

M4THCDPs have the following characteristics and constraints [14]:

- Includes a HCDP-language that can inter-relate various types of patterns, as shown in Fig. 6.
- Includes a repository and catalogue of common, and standard CDPs which are applied by team-members, and includes directives, descriptions, and application templates.
- The mentioned catalogue includes: Fundamental M4T-patterns, which are common patterns to be used by DICS' applications in various APDs, Project domains, and M4T-languages, and are specialized in other M4T-patterns.
- Modularization-patterns, are patterns that are used for (re)structuring and the decomposition of DICS' modules.

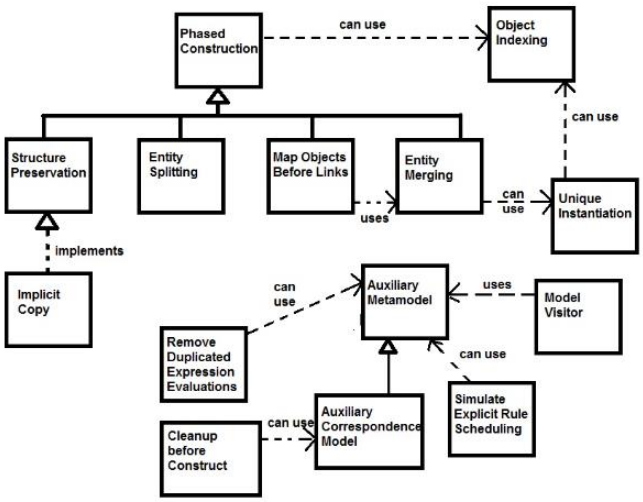

Fig. 6. Interaction of various M4THCDPs [14].

- Optimization-patterns are used to improve the Project's transformation activities and efficiency.
- Model-to-Text (M2T) patterns, are specialized-patterns for documenting transformations' activities.
- Expressiveness-patterns provide technics to expand and improve M4T-language's capabilities.
- Architectural-patterns identify methods to (re)organize DICS' components and subsystems.
- Bidirectional-transformation-patterns propose techniques for implementing categories of transformational-patterns.

For the GDSCI, the M4THCDPs were implemented and is assisted by the global HCDP.

3 Integration of the Global HCDP

3.1 Basics

The Global HCDP (GHCDP) that is applied to change DST's schema, DSTCPs, and to refine/refactor associated software-components when the DICS' receives a static (or dynamic) change-request.

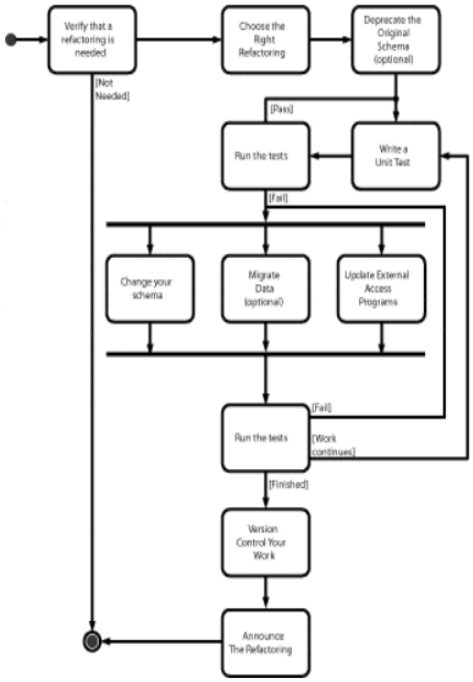

Fig. 7. The DSTRR [19].

The GHCDP supports Projects in the following activities:

- Is applied in methodological, static, and dynamic, transformation-activities: 1) In Methodological-phase(s), CDPs are designed and implemented by using IIPTFM's defined notations, like OOM, UML, or Archimate; 2) Static-activities refers to classical-implantations of CDPs; and 3) Dynamic-activities refer to the injection of CDPs.
- Automated DST-based transformations using CDPs.
- By using existing frameworks and concepts, like the MEF, and related IDSs or Datatables, which can be easily refined/refactored.
- Using the DMVC-pattern to map and refine: 1) M-variables; 2) V-variables; and 3) C-variables.
- GHCDP-based transformations based on refactoring and refinements.
- PTDM, ADevOps, and Project's implementation's activities synchronization.
- And other components' related activities.

DSTCPs support continuous and synchronized DSTs' based-transformations, integration-activities, and agile-implementation activities. They offer abilities to change DICS' and DSTs' components in static (or in JIT-runtime). DSTCPs support also continuous-activities that enable Project-teams to refine DSTs' artefacts, like source-code; this is considered a Polymathical concept of refactoring and refinement of DSTs. The mentioned facts support the following GHCDP-based DST transformation operations [9,19]:

- Verifying DST's Refining and Refactoring (DSTRR) statuses (if they are appropriate), as shown in Fig. 7.
- Deprecate original-legacy DST-schemas and apply testing scenarios.
- Modification of DST-schemas and the migration of related DST's source-data.
- Applying continuous ADevOps and the integration of version-control systems.

- Extracting and refining DST-scripts and the automation of DST-schema's creation or modifications.
- The creation, removal, update, and reading artefacts/objects in DSTs.
- The modification of external access software-components and applying regression-tests.
- The removal and (re)creation of DSTs components by using ADevOps.
- To simplify DICS' Project's developers' activities related to the manipulation of DSTs.
- To integrate all ADevOps check-in and naming Natural Language Programming's (NLP) scripts.
- Automating DSTs' transformations NLP script for creation activities.
- Implementing DSTs' version-control/checking, and deployment processes, to support continuous-delivery of transformed components.
- The mentioned Project's operations need the refinement and refactoring of legacy software-components; which needs also to refine CModels' design without the change their structure and semantics.
- DSTRR's operations are light modifications of DSTs' schemas (which are tables-structures, data-values, stored-procedures, triggers…); which improve CModels' design without the change of its semantics, and DICS' performances.
- The DSTRR supports evolutionary implementations of DSTs' related processes, using an iterative (and incremental) approach that is coordinated by the PTDM.
- DSTRR is riskier and more complex than basic refactoring of software-modules, because, it the DSTRR must maintain informational-semantics and not just the behavioral-semantics.
- GHCDP-based DST transformations, mean that all DSTs modifications and transformations are done through GHCDPs.
- GHCDPs manage DST's complex changes and trace their impacts on quality of data, and security, by using specific CDPs.
- GHCDPs manage DST's statuses and changes' impacts on various types of performances, availability, scalability...
- GHCDPs manage DST's attributes (columns and rows from ERM data-sources) and various types of attribute changes.
- GHCDPs manage DST's data-values transfers and related changes.
- GHCDPs manage DST's DICS source-locations changes.

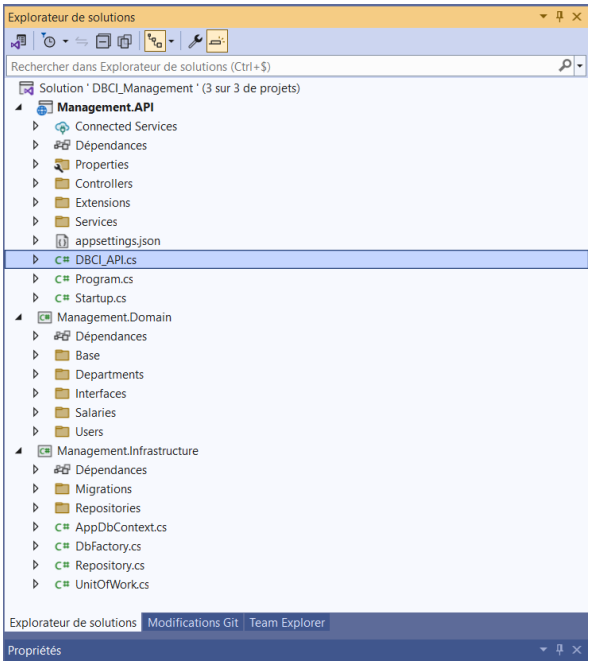

Fig. 8. CPoC's section for setting-up the basic functionalities.

GHCDP-based refinements include the following activities:

- Synchronizing related DORM software components and their refactoring.
- Related CModels, OOM's class-Diagrams, ERMs, EAMs, choreographies (and BPMs), CDPs, and other models are refined.
- Related CBBs and services', Blocks and APIs are refined.
- CModels (and other) relations/associations, pointers, and DST's GUIDs (or keys) are refined.
- DSTRR supports relations (and associations) in Project's transformation-processes.

As shown in Fig. 8, the CPoC basic parts were implemented using Microsoft Visual-Studio and .NET (MVSNET). For the GDSCI, the DSTRR and MVSNET were setup.

3.2 Relations-Associations, and DSTRR

As elements like CModels, CBBs, Blocks, and other, are refined and refactored, their relations and associations (and other types of links) are also refactored and refined. The DSTRR applies processes which use CDP-based technics for refactoring to refine [6]:

- Association, links, pointers, and relationships.
- OOM's various types of relations like in programming-languages like JEE, C/C++.
- Programming-languages artefacts like C/C++ pointers and classical-structures.
- The refactoring of 2nd and 3rd generation programming-languages like COBOL, ALGOL, PLI…
- DST's GUISs, keys and Views.
- NoSQL based DSTs and their OO-relations.
- Programming-languages JIT compilations.
- …

For the GDSCI, the DSTRR was used to refine relations and associations.

3.3 The DSTCP, GHCDPs and Views

The DSTRR, DSTCPs, and GHCDPs are applied to refine CDPs like the DMVC pattern. It ensures also its integration, and when a change-request (or event) is issued. The transformed artefacts and elements are:

- Artefacts specific to the View (Static, Dynamic, or Methodological) and quantifies possible impacts and implications.
- The DICS, EA/IIPTFM, and cartographies/BPMs are dynamically generated, implemented, and maintained.
- All related CModels are refined.
- All related cartographies/BPMs are refined.
- The integrated CBBs, Blocks are refined.
- Interaction and evolution of DMVCs.
- ….

For the GDSCI, the Views were setup to abstract various DSTs' data-sources.

3.4 DSTCP's and DMVC's Integration and Interactions

DSTCPs are used to modify and change the DMVC-pattern, when an event is captures. The transformed artefacts are:

- The DORM (or in this case the MEF) synchronizes DMVC, CPDs, CModels/ERMs, and OOM related changes (and associated software-components).
- DORM/MEF, IDSs and Data-Tables are refactored.
- DMVC-pattern's mapped M-variables are refactored.
- DMVC-pattern's mapped V-variables are refactored.
- DMVC-pattern's mapped C-variables are refactored.
- DST's attributes (column and/or rows) are refactored and changed.
- DST's data-values transfers are implemented.
- DST's attribute-formats are changed.
- Related DORM software-components are refactored.
- CModels like ERMs, are refined.
- ...

MEF supports DMVC's, DSTCPs, and CModels (ERMs and OOMs) integration and interactions. MEF is inspired from DORM and supports the interactions between DMVC, DST(s), and OOM software-components.

```
public class EndClientConfiguration:
        EntityTypeConfiguration<EndClient
        >;
{
    public EndClientConfiguration(): base()
    {
        HasKey(p => p.Id);
        Property(p => p.Id).
        HasColumnName("Id").
        HasDatabaseGeneratedOption(DST_G
                eneratedOption.Identity)
                .
        IsRequired();
        Property(p => p.dtUpdated).
        HasColumnName("EntryTs").
        IsRequired();
        ToTable("EndClients");
    }
}
```

Fig. 9. The EndClientConfiguration class.

It is a combination of model-first, code-first, or DST-first approaches; but it is nevertheless a classical DST-ERM concept and sets of features. MEF is a framework that support routine DST-operations, migrations, and complex transformation-activities. These transformational-activities are tuned to automate-changes and to avoid data-losses, because during DSTRR operations (like refactoring and refining) data-values are lost and wrongly modified.

MEF models DSTs in their actual-state(s), and enable transformational-activities related to DSTs in an iterative-manner. MEF's API is adapted to the code-first-approach and generates Plain Old Class Objects (POCO) from DST(s) and by simply accessing software-components, as shown in the example in Fig. 9, which illustrates the creation of a configuration-class which derives from the EntityTypeConfiguration-class.

The transformation is applied by the MEF and

through its APIs: modelBuilder.Entity<ClassName>().ToTable("DSTTableName", DSTSchema"). That is also one to the Project's codebases. For the GDSCI, the DMVC pattern was applied in all the Project's codebases.

3.5 The DSTCP, Codebases' and Platforms Interactions

As mentioned, the DSTCP is a compound pattern that modifies the DMVC-pattern, when receiving change-request (or event). And the transformed artefacts and elements are:

- Codebases are refactored.
- All related CDPs are refined.
- Error-management is modified.
- All types of CBBs, Blocks, CModels, and APIs are refined.
- ADevOps is synchronized and adapted.
- Estimates DST's changes impacts on DICS' security, performance, and availability...
- …

For the GDSCI, the codebases and platform interactions were optimized.

4 Global HCDPS In APDs

4.1 The Enterprise Application Architecture Patterns (EAAP)

EAAPs are sets of pattern-groups that are applied to design Entity's activities; and these groups are [11];

- Domain-Logic Patterns: Are associated with Transaction-Script, Domain-Model, Table-Module, and Service-Layer.
- Data-Source-Architectural-Patterns: Are associated with Table-Data-Gateway, Row-Data-Gateway, Active-Record, and Data-Mapper.
- Object-Relational-Behavioural-Patterns: Are associated with Unit-of-Work, Identity-Map, and Lazy-Load.
- Object-Relational-Structural-Patterns: Are associated with Identity-Field, Foreign-Key-Mapping, Association-Table-Mapping, Dependent-Mapping, Embedded-Value, Serialized-LOB, Single-Table-Inheritance, Class-Table-Inheritance, Concrete-Table-Inheritance, and Inheritance-Mappers.
- Object-Relational-Metadata-Mapping-Patterns: These are associated with Metadata Mapping, Query Object, and Repository.
- Web-Presentation-Patterns: Are associated with MVC, Page-Controller, Front-Controller, Template-View, Transform-View, and Two-Step-View.
- Distribution-Patterns: Are associated with Remote-Façade, and Data-Transfer-Object.
- Offline-Concurrency-Patterns: Are associated with Optimistic-Offline-Lock, Pessimistic-Offline-Lock, Coarse-Grained-Lock, and Implicit-Lock.
- Session-State-Patterns: Are associated with Client-Session-State, Server-Session State, and DST-Session-State.
- Base-Patterns: Are associated with Gateway, Mapper, Layer-Supertype, Separated-Interface, Registry, Value-Object, Money, Special-Case, Plugin, Service-Stub, and Record-Set.

For the GDSCI, the EAAPs were implemented and they support the Enterprise

Design Patterns (EDP).

4.2 The EDP

EDP delivers sets of pattern groups that can be used to design enterprise activities, and these groups are [12]:

- Behavioural-Patterns: Are associated with Human-Interest, Nurtured-Trust, Powerful-Questions, Listening-to-Understand, Hint, Tangible-Presence, and Walking-Your-Talk.
- Practice-Patterns: Are associated with Evidence, Outside-Inspiration, Hypotheses and Validation, Wearing-Their-Shoes, Dancing-to-Enterprise Rhythms, Corporate-Politics, Focus, Shift, Refocus, Just-Enough-Design, and Unintended-Consequences.
- Creations-Patterns: Are related to Human-Languages, Captured-Cases or Stories, Depicting-Shared and Understanding, Moments In-time, Toolkits-Sparking-Change, Beauty, Tangible Futures, and Management Instruments.
- Entity's DICS(s) and its software components transformations use the strangler-pattern.

For the GDSCI, the EDPs were designed and implemented; and they support CModels and EAMs for different APDs.

4.3 CModels and EAMs for APDs

For a long period decades XML based architecture models were dominant in various APDs like in finance, insurance, education, and other. The XML based models were used to transform Entity's legacy-DICS and create XML based OMS that supported [29]:

- The integration-inclusion of DORM, PPM/ERM, DSTCPs, and DSTs related transformation-scenarios.
- GDSCI for classical client-server-architectures created a deep shift in avant-garde DICSs associated domains.
- The transformation and then replacement of legacy-monolithic mainframe-based DICSs into CBBs and CModels-based DICS were deployed across different platforms.
- Stateless DST/data objects represented by XML-strings supported agile EAMs.
- Stateless DST/data objects (in various formats) unbundle DICS' applications into independent CBBs, EAMs, and CModels that interact across Entity's DICS-network(s), using adaptable interfaces.
- The OMS-based architecture template supports EAM practitioners, architects, and designers in transforming the DICS.
- Supports CModels-based Distributed Transactions Patterns (CDTP).

For the GDSCI, EAMs and CModels were designed and implemented; and they support the integration of CDTPs.

4.4 Integrating CDTPs

CDTPs use CBBs-based CModels, but is a complex approach because how can a Project manage CDTPs across multiple DICS-nodes and accessed platforms. The solution is to apply DSTCPs and GHCDPs. When GDSC4AI and GDSCI are used in a DICS to support the CDTP as shown in Fig. 10.

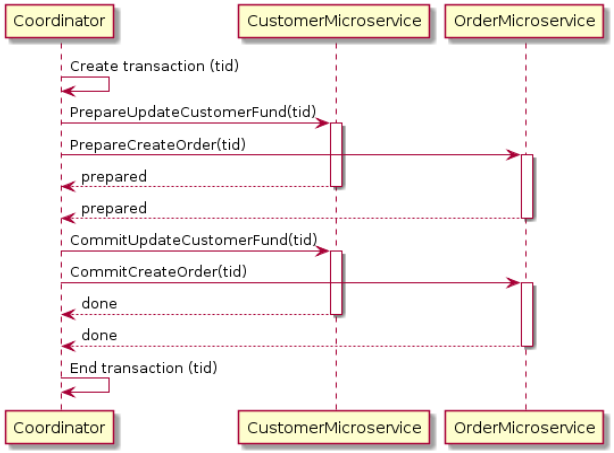

Fig. 10. A CDTP Sequence [41].

Legacy transactions in legacy-DICS have to be converted in CDTP which is a multiple sequence of CBBs, like in the case of EndClient's order example. This example illustrates the context of a legacy-DICS as shown in Fig. 10. CDTP use the Two-Phases-Commit (2PC) that: 1) In phase 1: Prepares commit-operations; and 2) Is the commit phase. In the preparation-phase, all CDTPs (and associated CBBs) enable DST/data-change(s) that are done with an atomic-level-of-granularity. When all CDTP are ready, the commit-phase directs CBBs to execute the changes. These operations need to control and hence they are centralized, therefore, the Project sets up ADevOps coordination for maintaining CDTPs' lifecycle, and to managed requested CBBs (in both phases preparation and committing) [41]. To perform CDTPs' transformations, the Project must prepare DSTCPs and other CDPs the Saga-pattern, as shown in Fig. 11; the Saga-pattern offers [38]:

- A CBBs-based architecture and CModels, which support the DST-per-service concept. Which enables that each APD (or functional domain) service use a DST which best-serves the associated CModels or ERMs.
- The atomic DST-per-service concept, and DSTs (or DSTs) can be integrated and scaled independently.
- If the DST fails (or is locked), then the failure is isolated from other CDTPs and related CBBs.
- CDTPs' integration is very complex because the related transformational-processes need to perform parallel-transactional-operations, so, CDTPs have to be Atomic, Consistent, Isolated, and Durable (ACID).
- Atomicity (a term related to granularity) is a set of operations that can happen simultaneously (or none).
- Consistency for CDTPs transfers IDSs from one valid state to another state.
- Isolation enables concurrent CDTPs to produce same IDS' state which can be executed-sequentially.
- Durability supported CDTPs committing operations and that they remain consistent and credible when errors happen or when DICS fails.
- ACID in a single-service, is not a complicated-issue; but when applying CDTPs across various DICS-nodes, then CDTPs becomes difficult to manage.
- CDTPs require that all its tasks and related CBBs, to commit (or roll back) before that all related operations can continue; and all types of DSTs support such features.
- A DICS' Inter-Processes-Communication (IPC), allows separate-processes to share DST/data, and in this case, CDTPs are then ready to commit.
- An optimal-approach is to implement standard-patterns like the Saga-pattern, which enables DST/data-management and

consistency across CDTPs and related scenarios.

- A Saga is a consistent set of CDTPs that update IDSs and then they publish messages (or events) to trigger the next operation. If any operation fails, Saga performs a compensating action that rolls back the preceding CDTP that has succeeded.
- CDTPs are mainly ACID-transactions and that has to be taken into account when using the Saga-pattern.
- CDTPs are single-unit-of-work which is a set of operations. In a CDTP, events change states on DST conditions and commands-capture all DST/data required to execute an action on a DST-object.
- The Saga-pattern use sets of local transactions (or transactions called from a CBB); where each of these local transactions modifies the DST, and then emits messages (or event) to trigger the next local-transaction in the saga.
- When local transaction has errors or fails, the service within Saga executes a compensating transaction that undoes previous changes.
- Compensating CDTPs are in fact transactions which that be reversed by launcher other CDTPs.
- Implementing pivot-transactions, when committed, then saga persists running until CDTPs related processes finish. These CDTPs cannot be stopped (or compensated).
- Retriable CDTPs come after pivot-transactions, and are committed to succeed.
- Saga-patterns use choreography (or orchestration-operations).

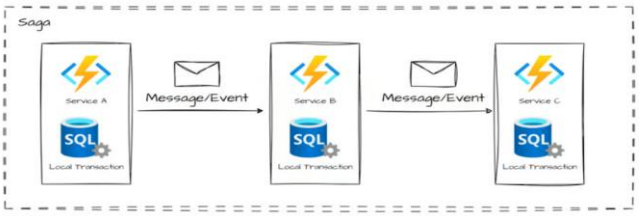

Fig. 11. The Saga pattern [38].

For the GDSCI, Saga-patterns were designed and need a set of choreographies.

4.5 Choreography Operations

The GDSCI and its related CBBs and DSTCPs can coordinate CDTPs and sagas, as shown in Fig. 12, and they support the events-exchange without the using a single centralized point-of-control.

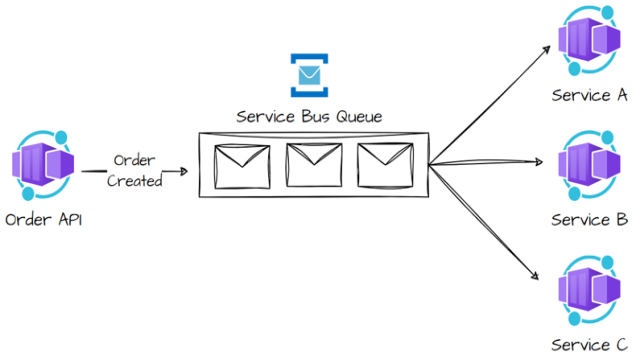

Fig. 12. Choreography activities [38].

Each legacy local-transaction publishes APD related events which trigger CDTPs in other business-activities. This concept is optimal for basic-workflows that use limited sets of CDTPs or CDPs in the Saga-pattern and they should not be coordinated. In the same time it does not use the Single Point of Failure (SPF) approach, because related responsibilities are distributed across saga, and choreography's implementations do not need extra- CBBs' implementations, integration, or

maintenance efforts.

However, complex CModels and GHCDPs' integration are difficult to track with Saga-nodes (or participants). For that there is the need to listen to all commands, and each CBB in the saga depends on each other and in Project cycles; since they need to consume each other's tasks, and commands. Integration-testing becomes complex, as all CBBs and CDPs will need to be executed to simulate CDTPs [38].

For the GDSCI, choreographies were designed and need a set of orchestrations.

4.6 Orchestrations Implementations

Another possibility is to coordinate Project's sagas, by applying a central-controller that manages (and controls) needed CBBs and CDPs. These CBBs and CDPs are used by sagas and are executed by CDTPs; and they support the following operations, as shown in Fig. 13 [38]:

- The orchestrator manages-handles all CDTPs and informs all used CBBs and CDPs in sagas, which operations they need to execute (and that depends on the received-events), and in the same-time interprets the each task's state, as well as handling-errors and failures with compensating CDTPs.
- Such a concept is optimal for workflows that have many CBBs and CDPs in sagas, and when it is upfront CBBs and CDPs that more CBBs and CDPs will be coupled (or added).
- Cyclical dependencies must be removed (they are problematic for choreographies), because orchestrators depend on all participants in sagas.
- Used CBBs and CDPs in sagas have no information about other CBBs and CDPs, and they insure the separation-of-concerns.
- Unfortunately, this fact introduces more complexity, as CBBs and CDPs implementations require coordination, and insert additional point-of-failures, because orchestrators manage workflows.
- EAM and CModels support the implementation of Saga-patterns, which are complex in Project's implementation phase.
- CDTPs are not local, but are distributed, which is hard to coordinate and manage.
- Saga patterns are complex and hard to debug and test, as the more CBBs and CDPs are added, and the complete concept can become complex.
- DST/data cannot be rolled-back in Saga-pattern because CBBs and CDPs commit changes to their local DSTs.
- EAMs are used to handle transient errors and failures; while idempotency are used to handle DST/data-consistency.
- The used sagas can potentially be made-up of sets of CBBs and CDPs, and there is the need to implement other components to observe all used CBBs and CDPs and ensure the ability to track the workflows used by the implemented sagas.
- Saga-patterns become complicated when having DST/data-durability problems or issues, like unfound-updates, dirty-reads, and non-repeatable reads can occur within sagas.
- Implementing semantic-locks, pessimistic-concurrency, versioning, and commutative-update to reduce errors and anomalies.

- It is important to know when and where to use the Saga-pattern, like in the case of ensuring data consistency in EAMs or DSTCPs, or when there is a roll-back or compensate operation related to a CBBs and CDPs, and the Saga-pattern supports both operations.
- But, if there are cyclic dependencies CDTPs, CBBs and CDPs, tightly coupled CDTPs or compensating CDTPs that can happen in earlier phase of a CBB in a Saga-pattern-based workflow, then the Project should use alternatives.
- DSTCPs interface Saga patterns in 2 ways.
- DSTCP enforced by a Saga-pattern can support complex fields like AI Subdomains (AIS).

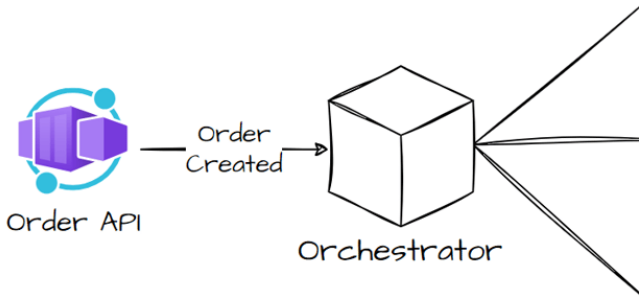

Fig. 13. The Saga pattern orchestration [38].

For the GDSCI, orchestrations were designed and need a set of AISs.

4.7 AIS's Integration

The GDSCI and GDSC4AI offer an II interfacing-concept for DICS, IDSs, and DSTs' components; and they systematically use DSTCPs that are applied all AISs. Where AISs are integrated in various APDs and are critical for Entities. But AISs are complex and actually Machine Learning (ML) is just a field in AIS. To support Project's and GDSCI's clear and structured use of AI Models (AIM), EAM, CModels, CDPs, and efficient-implementation techniques, standardized Blocks support the reuse in IIPTFM modelling and design activities that in turn use DSTCPs in AISs to improve traceability and Polymathic system design. The most popular DSTCPs for AISs are [23]:

- A Pattern is defined as a proven and established structure or template for solutions for recurring design problems that modeled in a DICS agnostic format.
- The idea of patterns originated from building architecture, and has been adopted by DICS practitioners, like GoF.
- CDPs abstract AIS-experts' knowledge and enable design-decisions. There are many existing CDPs sets for AIS, and some of them are unique CDPs used for AID.
- Common, standard patterns, and CDPs, are supported by patterns like the Adapter, Factory method, Observer, Strategy and State.
- It is effective for Entities that require tailored CSPs-based AIS solutions.
- GHCDPs can be used for various AISs like, Data Analytics (DA), Data Sciences (DS)…
- DSTCPs enables design adaptability, by promoting modular AIS that is dynamically-reconfigured and improves problem-solving performances.

- Using generic and unified engines like Apache Spark Ecosystem (ASE) [3,4,5].

For the GDSCI, needed AISs were designed and need a generic and unified engine.

4.8 Generic and Unified Engine

Using a generic and unified engine like the ASE can support II DST and AIS-based solutions and have various features like [3,4,5]:

- Batch management and streaming large-volumes of data.
- Manages DSTs processing in scripts, batches, and enables real-time-streaming.
- Applying existing programming-languages like: C++, Python, (No)SQL, Scala, JEE or R.
- SQL-analytics supports and the run of fast, clustered ANSI-SQL queries for filling dashboards and JIT reporting.
- More efficient than other data-warehouses.
- Supports DA/DS with scaling and Exploratory DA (EDA) on voluminous IDS/data without the need for down-sampling.
- Supports ML, DL, and related training-processes (enforced with algorithms) and enables the use of the same components to scale to fault-tolerant clusters of a huge number of DICS nodes/servers.
- Supports ANSI-SQL and to use the specific editor SQL.
- Offers a distributed-SQL-engine for large-scale DST/data-management.
- Offers an adaptive-query execution to plan at runtime, such as automatically setting the number of reducers and join algorithms.
- Supports structured and unstructured-data, that interfaces structured-tables and unstructured data such as JSON or images.
- Uses IDSs which is a distributed collection of data; and offers the Resilient Distributed IDS (RDIDS).
- The RDIDS offers strong typing, ability to use powerful lambda functions with the benefits of Spark SQL's optimized execution engine.
- An IDS can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter…).
- The IDS API is available in Scala and Java, but Python does not have the support for the IDS API.
- RDIDSs facilitate global-refactoring processes.
- All previously mentioned activities need a global-EDP.

For the GDSCI, needed AISs were designed and need a generic and unified engine.

4.9 Global Refactoring Processes-The EDP

The EDP and its global refactoring ane refinement processes support applications and DSTs refinements by [15]:

- Applying fast-evolution (changes) of software and DST-components and their implementations; are Project issues during Project's phases; and has important impacts on all Project's lifecycles.
- The evolution's of severity depends on the frequency-of-changes which has to be adapted to the Project's realities and DST's components that can be impacted.

- Refactoring is a popular-practice in OOM and DORM based DICS for evolving the software and DST-components and EAMs.
- The evolution of DST-schemas and DST/data is autonomously developed from unbundled-components, such changes have impact on applications.
- DORM complaint frameworks like MEF support the propagation of the evolution from an application to a DST. But such frameworks are not capable of solving complex refactoring cases nor they migrate data (values) properly as shown in Fig. 14.
- The problem of applications' modules and DSTs transformations and evolutions has to be taken from a technical-viewpoint and the formal-model of applications' refactoring processes and their impacts are shown in Fig. 14.
- The Project's implementation uses a model of a persistence layer, which can be transformed into a model of a DST-schema; or directly into a DST-schema.
- The needed-changes of application's-layer can be represented as a sequence of transformational-steps.
- These transformational-steps affect the structure of the application-layer and/or DST-schema.
- This article (and its CPoC) shows how these transformational-steps are used, not only for a structural-changes, but also for an automatic-dynamic generation of DST/data-migration-scripts.
- Basic-refactoring-processes and cases are complex ones and are created as sequences of the basic-refactoring-steps.
- Capabilities of the proposed formal-models are common-refactoring-patterns.
- The evolution of the Entity's software-modules is based on atomic-transformations specific for each software-modules.
- The GDSCI instructs how refactoring-processes are used to change software-modules.
- The basic-iterative and evolutionary-transformation of software-modules is based on basic evolution of an application-module.
- GDSCI's analysis impact(s) application's refactoring on DST-schemas and DST/ data.
- Basic-transformations of application-modules and DST, are done on basic-refactorings.
- The GDSCI assures structural-safe-changes of application-modules and DST data-safe migration of DST-schemas and data.
- The GDSCI proposes a set of models and transformation-rules; and that allows to simulate the variants of possible transformations-steps.
- The application-modules (or source-code) and DSTs co-evolution, improve the capabilities of DOIDEs and improves implementors-efficiencies.
- The impact of advanced and refactoring-processes on DSTs and stored DST/data, are evaluated by the automated-co-refactoring-processes that is possible for basic and complex-changes of an application- modules.
- Project-team members are wary about complex-refactoring-scenarios, but due to the fact that hand-crafting of mappings is not necessary in most cases (mainly related to associations is used as the mapping

function), the proposes model shows advantages over hand-made migration-scripts.

- Capabilities of defined transformations is limited, because of the focus, which is on DST data's preservation or transformation-concatenation.
- GDSCI-related transformations are capable of handling many refactoring-cases and are verified by using EDP's refactoring statistics (and choosing transformations' influencing-data).
- Refactorings common-cases are: 1) Renaming, is the most used refactoring-operation; 2) Move refactoring is used in DOIDEs to move-properties from a class to another-class within an inheritance hierarchy or to move classes-between software-modules; 3) Extracting-classes is an often-used refactoring DOIDE's operation; 4) Moving-fields between classes; and 5) Replacing data-values with objects values is a refactoring-case and process.

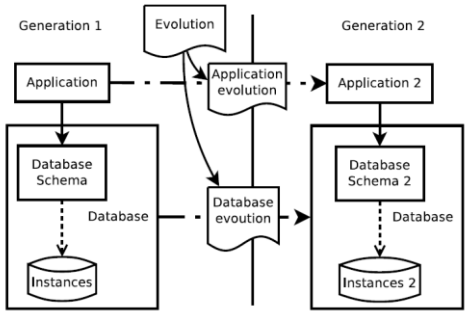

Fig. 14. Components related evolution-processes [15].

All mentioned operations are supported by a framework like ASE's RDIDS which is presented in the CPoC. This section presents the roadmap that enables Projects to avoid major-problems like performance. For the GDSCI, needed EDP refactoring processes were executed and there is the need to manage performance barriers.

4.10 Performance Barriers

Zzzz A major issue and problem for Projects and DICS is the application of various typologies that are based on a variety of CDPs, is the problem of the end-DICS's performance. That can be solved by applying and iterative-method like the PTDM combined with ADevOps; and by major iteration, the performance of the end-DICS is evaluated. That can be improved by using Distributed Digital Integration Hubs (DDIH). As shown in Fig. 15, a DDIH is an advanced platform architecture that aggregates multiple-back-end sub-systems and DSTs, in a low-latency and shared-unique DST. The DST caches and persists IDSs dispersed across various siloed-back-end DSTs. The DDIH makes DSTs available to the end-system's applications through high-performance APIs. Applications access the DDIH, by using API-service-layer and enables important performance-improvements by requesting DST/data from only one DDIH interface to the distributed-store [3,4,5].

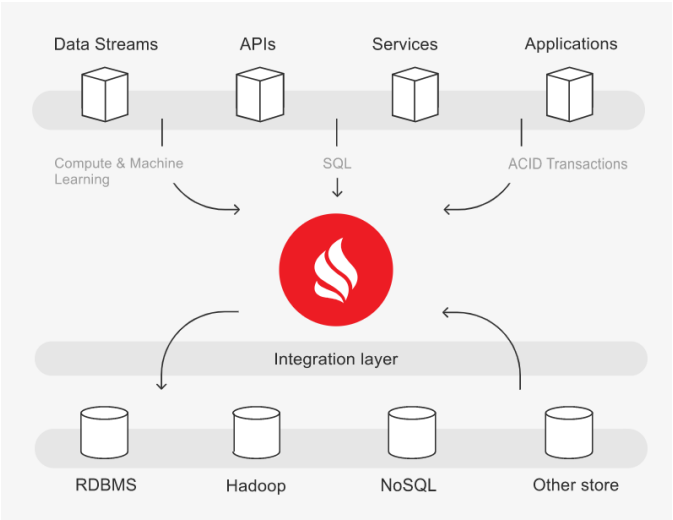

Fig. 15. The DDIH interaction with various DSTs

[3,4,5].

Gartner declares that the DDIH as an avant-garde application architecture that aggregates multiple back-end DICS of record DSTs, int a low-latency and scale-out, high-performance DST. A DDIH typically supports access to data via an API-services-layer. The high-performance DST is synchronized with the back-end sources by using the combination of event-based, request-based, and batch-integration-patterns, as shown in Fig. 16 [17].

For the GDSCI, evaluates need performances levels.

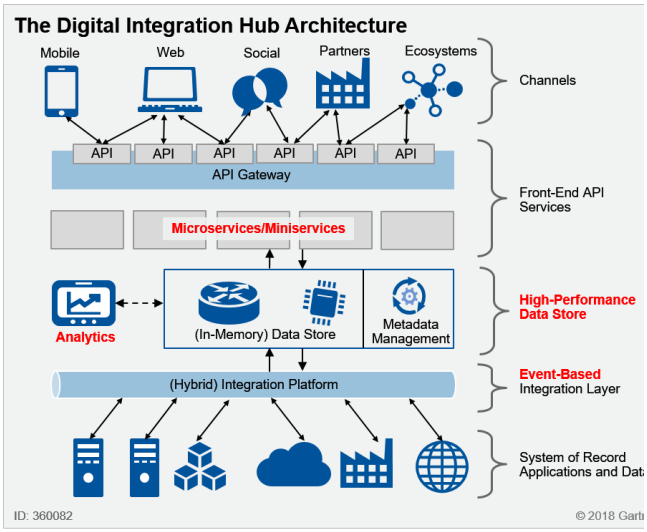

Fig. 16. Gartner's view on DDIH [17].

5 The CPOC

5.1 Setting Up Spark's RDIDS for Complex GDSCI Operations

ASE's RDIDS offers virtual IDSs operations' environment that supports GDSCI by [3,4,5,8,22] :

- Uses the Java Runtime Environment (JRE) version for ASE's integration.
- Setting-up MVSNET for ASE still that uses also Java VM and NET-Spark runs on top of Java runtime.
- Install Apache-Hadoop.
- Generic transformational-capabilities.
- The production of new RDIDS (DST/dataframes/IDSs) from the legacy/actual RDIDSs; where it inputs RDIDS and outputs one or more transformed RDIDSs.
- Inputted RDIDSs are static and cannot be changed since RDIDS are immutable.
- Offering Narrow-Transformation functions like map(), mapPartition(), flatMap(), filter(), union()…
- Offering Wider Transformation like groupByKey(), aggregateByKey(), aggregate(), join(), repartition()…
- Offering actions that create RDIDSs from existing-ones, and on them actions can be performed, like collect(), count(), first(), top()…
- That all supports complex GDSCI and GDSC4AI operations in various APDs.

5.2 Complex GDSCI Operations for AISs

Complex GDSCI for AISs is based on [22]:

- Domains like Big Data Analytics and other.
- The ASE is a generic, scalable analytical DST data-engine that processes large-scale -data in DICSs. And contains common-interfaces for various languages like Python, Java, Scala, SQL, R and MVSNET (which is used in this CPoC).
- As shown in Fig. 17, ASE's includes various libraries, APIs and DSTs and provides a whole (eco)system that

manages all sorts of AIS related DST/data-processing and analysis capacities.

- ASE's Core is the fundament of Spark that is responsible for memory operations, job scheduling, building and manipulating data in RDIDSs…
- Supports In-memory Processing that ensures that no time is spent moving DST data or processes in or out to disk; that makes Big Data Analysis (BDA) very-fast.
- Offers efficiency because it caches-inputted-data in memory by using the RDIDSs, which are fundamental-data structures that support transformation-processes and distributed-processing.
- Each IDS in RDIDS is partitioned logically and each logical-portion are processes on different DICS cluster servers/nodes.
- Real-Time Processing supports streamed-processing that supports DST/data inputting and outputting in real-time.
- Offers a set of APIs to implement AIS solutions.
- To setup for AISs like BDA in the context of MVCNET environment.
- BDA is used in many APDs with colossal-volumes of DST/data coming out from billions of tweets, iMessages, Live streams, Facebook and Instagram posts… Terabytes (and petabytes) of data are generated in minutes.
- Such volumes are not easy to handle and the processing Big Data sets need to be updated frequently. And therefore, for BDA Entities use NoSQL DSTs, Hadoop along, and various types of assisting Analytics-tools like YARN, MapReduce, Spark, Hive, Kafka…
- The mentioned environment and tools make-up BDA's ecosystem and cannot be analyzed in one article.

Fig. 17. The ASE [22].

ASE's architecture follows the driver-executor concept, as shown in Fig. 18, where each ASE-application has a driver and a set of workers (or executors) that are managed by the cluster-manager. The driver includes a user-program and a spark-session. The session-controls user-program and divides into smaller executable-chunks. Each executor takes one of those atomic-tasks from the user-program and executes it. The cluster-manager manages the overall execution of the program.

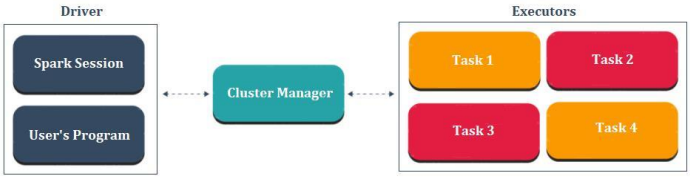

Fig. 18. ASE's architecture [22].

The CDTP uses an ASE-session to access AISs and that is supported by a distributed-processing environment.

5.3 Preparing Blocks to Use the GDSCI for CDTP Operations

ASE and GDSCI based CDTP(s) can be evaluated by the implementation of a concrete ACS and context, and as shown in Fig. 19, CDTP's

class-diagram, abstracts CDTP's interactions in Project's context, and the used level of granularity based on the '1:1' mapping-convention.

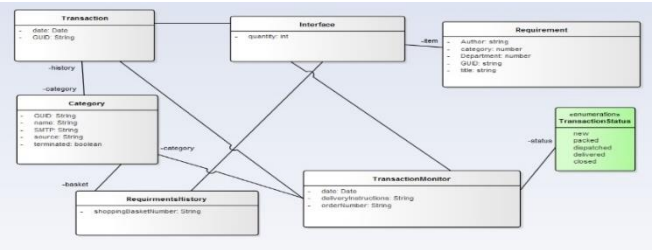

Fig. 19. The atomic CDTP's class diagram.

The logical-view of a series of transactions based on the Service Oriented Architecture type or approach was used as shown in Fig. 20, and the consumption of an atomic web-service in a single transaction. From the atomic business-transaction activity-diagram, the resilience of events exchanged during the transaction's execution is important.

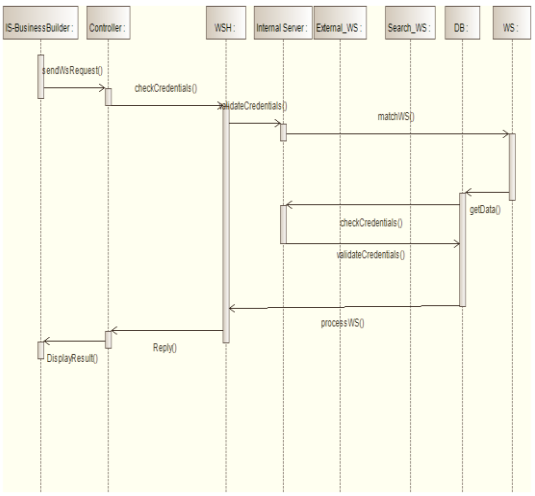

Fig. 20. The CPoC's CDTP.

All events are exchanged between various DICS nodes require a strong encryption setting, that are in the EAMs development phase. From a technological-perspective, the atomic business-transaction is composed of application-components which are the fundamental business CBBs of the DICS. A top-down combination of TOGAF's phases B and D resulted in the optimal construction of a transaction, based on an atomic web-services approach.

5.4 Block's Generator

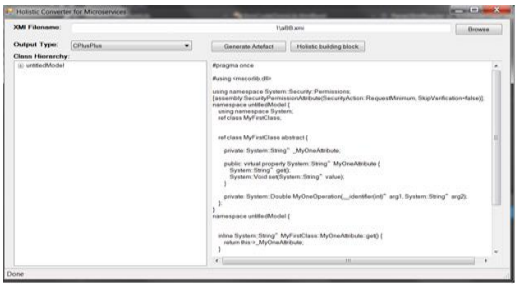

Fig. 21. The Blocks' generator interface.

CBBs and Blocks' generation, as shown in Fig. 21, produces the needed CBBs Blocks that aez used by the GDSCI and DSTCPs. This CPoC assesses GDSCI's integration and has the following feature and requirements: 1) To promote feasible DSTCPs and GDSCI-strategy, and vision to finalize the risky Projects and especially its implementation phase; 2) To use external components like ASE; and 3) To prove the application of ageneric approach of the GDSCI. The CPoC was evaluated through the implementation of ASE, GDSCI, DSTCPs and a CDTP; and using MVSNET and java development environments. The CPoC proved that the GDSCI is feasible and that it is optimal for GDSCI based Projects. The CPoC uses a Blocks based CDTP that interfaces ASE's RDIDS/IDSs and in the mapping of artefacts, activities and tools [30,31].

6 Conclusion

This RDP proposes a set of facts, recommendations, patterns, and transformational actions to support the implementation of IIPTF(M), GDSC4AI, GDSCI, and DSTCPs for Projects, for any APD type. The IIPTF and GDSCI use the

GDSCI, GHCDPs, DSTCPs, FMS, PRWC, GAPE, and Factors to iteratively check Project's feasibility and possible VHFR; and because of the CPoC's successful termination, this article proposes the following recommendations:

- IIPTF and GDSCI shows how to implement an Anti-Locked-In (ALI) transformation framework and related DSTCPs and GHCDPs.
- This RDP uses a mixed concept (qualitative and quantitative) and multi-level-refinement; by using the EDP.
- The GDSCI PRLR proved the existence of an important knowledge gap and the reasons for VHFRs.
- The AHMM4GDSCI based HDT support DSTRRs and GDSCI.
- Cross-functional/Polymathic skills are needed.
- Projects need a GDSCI and such Projects.
- The GDSCI facilitates the integration of IDSs, DSTs, and DSTMs.
- The IIPTF and GDSCI use and interfaces existing frameworks, standards and methodologies, like TOGAF, CModels, ERM...
- The GDSC4AI and GDSCI use the DST-first-approach-approach.
- The GDSCI uses DSTCPs, to interface DSTs and enable architecture/design activities.
- The Project has a pool of DSTCPs and CDPs which can have the following views: Static, Methodological, or Dynamic.
- The GDSCI is influenced by the Strangler and Saga-patterns.
- DSTCPs are specific and specialized GHCDPs which are used to change DMVC's integration elements.
- The GDSCI supports the CDTP.
- The Project can use a generic and unified engines like the ASE.
- The ASE based CPoC or GDSCI checks IIPTF and GDSCI's feasibility.
- The IIPTF and GDSCI's integration is feasible.

References

[1] Trad, A. Enterprise Transformation Projects-A Generic Data Storage Concept for Artificial Intelligence (GDSC4AI). To be summitted.
[2] Tegborg, M. Keep your eye on the ball. 2024.
[3] Apache. Unified engine for large-scale data analytics. Apache. 2024. https://spark.apache.
[4] Apache Spark. Apache Spark. Apache. 2024.
[5] Apache Ignite. Digital Integration Hub With Apache Ignite. The Apache Software Foundation. 2022.
[6] Ben Ammar, B., & Bhiri, M. Pattern-based model refactoring for the introduction association relationship. Journal of King Saud University – Computer and Information Sciences (2015) 27, 170–180. 2014.
[7] Bocanett, W. . Break the monolith: Chunking strategy and the Strangler pattern-Build decoupled microservices to strangle your monolithic application. IBM, Tokyo Garage, IBM Tokyo R&D Lab. Japan. 2022.
[8] c-sharpcorner. Spark RDD Operations. 2024.
[9] Datascientest. Refactoring Databases and Code: comprehensive guide to the essentials. Datascientest. 2023.
[10] Domnguez, E., Lloret, J., Rubio, A., & Zapata, M.A. Medea: A database evolution architecture with traceability. Data & Knowledge Engineering 65(3), 419 – 441 2008.

[11] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R. & Stafford, R. Patterns of Enterprise Application Architecture. Addison Wesley. 2002.

[12] Goebl, W., Guenther, M., Klyver, A., & Papegaaij, B. ENTERPRISE DESIGN PATTERNS-35 WAYS TO RADICALLY INCREASE YOUR IMPACT ON THE ENTERPRISE. Intersection Group. Austria. 2020.

[13] Jonkers, H., Band, I., & Quartel, D. ArchiSurance Case Study. The Open Group.

[14] KCL (2014). Model Transformation Design Patterns IEEE Transactions in Software Engineering. 2012.

[15] Macek, O., & Richta, K. Application and Relational Database Co-Refactoring. Computer Science and Information Systems 11(2):503–524 DOI: 10.2298/CSIS130610033M. 2014.

[16] Quan, L., Zongyan, Q., & Liu, Z. Formal Use of Design Patterns and Refactoring⋆ T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS 17, pp. 323–338, 2008. Springer-Verlag Berlin Heidelberg. Germany. 2008.

[17] Pezzini, M. The Digital Integration Hub Turbocharges Your API Strategy. Linkedin. 2018.

[18] Petrik, D., Untermann, A., & Baars, H. Functional Requirements for Enterprise Data Catalogs: A Systematic Literature Review. Software Business. 14th International Conference, DICSOB 2023. Lahti, Finland, 2023. November 27–29, 2023 Proceedings.

[19] Sadalage, P. Recipes for Continuous Database Integration Kindle Edition. 2007. Addison-Wesley Professional; 1st edition.

[20] Sidell, E.. Choosing the Right API Gateway Pattern for Effective API Delivery. NGINX. USA. 2020.

[21] Seifermann, V. Master Thesis: How to Strangle Systematically-An Approach and Case Study for the Continuous Evolution of Monoliths to Microservices. Institute of Software Technology. University of Stuttgart. Stuttgart. Germany. 2021.

[22] Tahir, M. Big Data Analytics using Apache Spark for .NET. Codeproject. 2019.

[23] Take, M., Alpers, S., Becker, Ch., Schreiber, C., & Oberweis, A. Software Design Patterns for AI-Systems. EMISA Workshop 2021. CEUR-WS.org Proceedings. 2021.

[24] The Open Group. Data dissemination view. TOGAF Modelling. 2015.

[25] The Open Group. Data lifecyle diagram. TOGAF Modelling. 2015.

[26] The Open Group. Data migration diagram. TOGAF Modelling. 2015.

[27] The Open Group. Data security diagram. TOGAF Modelling. 2015.

[28] The Open Group. Alignment with Other Frameworks. The TOGAF® Leader's Guide to Establishing and Evolving EA Capability. The Open Group. 2022.

[29] Trad, A, & Kalpić, D. Building an extensible markup language (XML) based Object Mapping System (OMS). Croatia: IEEE. 2001.

[30] Trad, A. A Transformation Framework Proposal for Managers in Business Innovation and Business Transformation Projects-Intelligent atomic building block architecture. Journal: Procedia Computer Science. Volume 64. Pages 214-223. Elsevier. 2015.

[31] Trad, A. A Transformation Framework Proposal for Managers in Business Innovation and Business Transformation Projects-An Information System's Atomic Architecture Vision. Journal: Procedia Computer Science. Volume 64. Pages 204-213. Elsevier. 2015.

[32] Trad, A. & Kalpić, D. Business Transformation Projects based on a Holistic Enterprise Architecture Pattern (HEAP)-The Basic Construction. IGI. USA. 2022.

[33] Trad, A. & Kalpić, D. Business Transformation Projects based on a Holistic Enterprise Architecture Pattern (HEAP)-The Implementation. IGI. USA. 2022.

[34] Trad, A. Organizational and Digital Transformation Projects-A Mathematical Model for Building Blocks based Organizational Unbundling Process. IGI Global. USA. 2023.

[35] Trad, A. A Relational DataBase based Enterprise Transformation Projects. Journal: International Journal of Mathematics and Computers in Simulation. Volume 17, Pages 1-11. Publisher: NAUN. 2023.

[36] Trad, A. Patterns to Transform (P2T)-DataBase Centric Patterns. Ministry of Education. European Union. 2024.

[37] Trad, A. & Kalpić, D. Business, Economic, and Common Transformation Projects-The In-House-Implementation of The Polymathic Transformation Framework (IIPTF). E-leaders. Slovakia. 2024.

[38] Valeida, W. What is the Saga Pattern? DEV. 2024.

[39] Vicente, A. In defense of extreme database-centric architecture. Memoria Investigaciones en Ingeniería. This journal is published by the Facultad de Ingeniería of the Universidad de Montevideo.

[40] Vicente, M., & Gama, N. Using ArchiMate and TOGAF to Understand the Enterprise Architecture and ITIL Relationship. Conference Paper in Lecture Notes in Business Information Processing · June 2013. DOI: 10.1007/978-3-642-38490-5_11. 2013.

[41] Xiang, K. Patterns for distributed transactions within a microservices architecture. Red Hat OpenShift. 2018. https://developers.redhat.com/blog/2018/10/01/patterns-for-distributed-transactions-within-a-microservices-architecture#

[42] Microsoft. Managed Extensibility Framework (MEF). 2023.