An exercise in concurrent software engineering: The MUSIC360 digital ecosystem

1st Yulu Wang [*] 2nd Ed Green [†] 3rd Roel Wieringa [†] 4th Jaap Gordijn [*]
[*] *Computer Science Department*, *Vrije Universiteit Amsterdam*, The Netherlands
Emails: y.wang4@vu.nl, j.gordijn@vu.nl
[†] *The Value Engineers*, The Netherlands
Emails: ed@thevalueengineers.nl, roel@thevalueengineers.nl

***Abstract*—In complex digital business ecosystems like MUSIC360, practitioners develop various requirement perspectives concurrently and independently due to time constraints and time-to-market considerations, but by doing so also create alignment issues between these perspectives. We argue that a knowledge graph-based approach enables systematic analysis and reasoning about traceability between these perspectives: in our case, the business value model of the ecosystem, associated data models, and security requirement items. In the knowledge graph, the elements of the business value model (e.g., economic actors), data elements (e.g., data assets), and security requirement item elements (e.g. access controls) are nodes connected by edges representing semantic/logic relationships (e.g., protects, authorizes). We illustrate the use of the traceability knowledge graph using the EU MUSIC360 project. The resulting traceability knowledge graph allows us to identify the perspectives' incompleteness and inconsistency, analyze the causes, and derive suggestions for improving model and requirements quality while using concurrent engineering. Theoretically, we contribute a cross-perspective traceability framework that unifies value-oriented, data-oriented, and security-oriented requirement-related artefacts into a single semantic network, plus reusable traceability patterns and rules to guide early-phase alignment. Practically, we show how to embed traceability analysis into concurrent engineering to improve requirement quality without sacrificing speed.**

***Index Terms*—Traceability, Security requirement, Knowledge graph, Concurrent software engineering, Music digital ecosystem.**

I. Introduction

Often, a complex software platform has to be developed in a relatively short time frame, e.g., due to a short time-to-market requirement. In our situation, namely the Horizon Europe project MUSIC360, the strict release planning is imposed by the funding agency, but also by ourselves. The MUSIC360 project develops a digital business ecosystem (DBE) and supporting software platform to better understand the value of music, and the related intellectual property rights on music. A DBE is a system of economic actors that depend on each other for their economic well-being and survival [1]. The MUSIC360 software platform is inherently complex because of the multiple stakeholders involved and the intricate music intellectual property right management itself. The aim of the platform is to collect data from users of music (which music is played where) and Collective Management Organizations (CMOs), and to show the results to the rightholders, music users, and policymakers at different levels of aggregation. CMOs represent the rightholders, such as performers and authors, and divide the collected money from the professional users of music, such as bars, shops, and radio stations.

To enable reflective learning, the project uses a staged delivery process. The first year is used to develop an initial prototype of the Music360 platform, which can be evaluated by the stakeholders, and the remaining time of the project is used to develop a second, improved version of the software. It means that the requirements must also be elicited in the first year. How do we collect these requirements for such a complex system in a relatively short timeframe?

To do so, we choose concurrent software engineering [2], which utilizes the idea that development tasks, and in our case, requirements engineering tasks, can be performed in parallel rather than sequentially. We apply this to *requirements* engineering in the sense that we develop different requirement viewpoints [3] at the same time. Our viewpoints are organized by topic and stakeholder concern: (1) the business model of the MUSIC360 ecosystem (who are the actors, and what are they exchanging with each other of economic value), (2) what data is collected, aggregated and presented, and (3) what are the requirements with respect to security. These three viewpoints are developed relatively independently of each other. There are a few synchronization points in time, where information about the viewpoints under design are exchanged, but the viewpoints are mainly developed in parallel. While this strategy accelerates development, it introduces traceability and consistency challenges also: artefact elements may be inconsistently defined in relation to the other viewpoints, incompletely linked, or redundantly specified.

The topic and contribution of this paper is to understand whether concurrent software/requirement engineering is a realistic option, e.g., to achieve a faster time-to-market. To do so, we analyze the found requirements for traceability, from different dimensions (e.g., completeness, consistency). We do so by creating knowledge graphs, linking the elements of requirements found in the various viewpoints.

This paper is structured as follows. In Sec.II we review related work on concurrent software/requirements engineering and traceability methods, and highlighting gaps in linking different artefacts. Sec.III presents our research design. In Sec.IV we introduce the MUSIC360 project and describe

Funded by the European Union under project No. 101094872

the constructed viewpoints, namely the business value model, data model, and security requirement items. Sec.V details the knowledge-graph construction design, including node categories and typed relationships, and explains how artefact elements are instantiated. In Sec.VI we show how the KG-based approach works for traceability analysis for the requirements viewpoints as constructed by the MUSIC360 project. Finally, Sec.VII discusses possible solutions for traceability issues and the overall effectiveness of concurrent software/requirement engineering. Sec.VIII summarizes our work and contributions.

II. Related Work

A. Conceptual Modeling and Security Alignment in DBEs

The notion of Digital Business Ecosystem (DBE) supposes a digital platform as the core and proposes a novel collaborative approach that leverages resources across multiple players and industries to meet the complex needs of diverse stakeholders [4]–[6]. Some research in DBEs [7] has emphasized the need for conceptual models that reflect the complexity of value co-creation, multi-actor governance, and platform-mediated interactions. Models such as UML data models, *e^3value* models [8], and value network maps [9] have been used to represent economic and data flows. However, these models are often used in isolation or only for business analysis, without being integrated with system-level concerns such as security. Because of collaboration and resource sharing across diverse participants in DBEs, this always leads to complex security challenges. Traditional security requirement engineering (SRE) approaches, such as KAOS, *i** [10], misuse and abuse cases [11], [12], or CORAS and OCTAVE [13] provide valuable mechanisms to capture adversarial behaviors and protection needs while ignoring business-oriented perspectives such as data utility or value creation. In DBEs, where stakeholders interact through shared data and cross-organizational value flows, these traditional methods may overlook security concerns that stem from economic motivation or value exchanges.

Often, there is a disconnect between what is modeled as valuable or sensitive in the ecosystem and what is protected at the implementation level [14]. Most contributions still focus on conceptual alignment at the level of isolation goal or risk modeling, rather than formalizing the traceability relations between different perspective artefacts such as data models (DM), value models (VM), and security requirement items (SRs). While evaluating traceability and consistency between requirement viewpoints, we build upon these insights and extend them by explicitly tracing relationships from elements of the *e^3value* model (VM) and UML class data model (DM) to relevant Security Requirement items (SRs) using semantically typed links within a knowledge graph. This enables deeper analysis of how specific value-oriented or data-sensitive concepts are governed by particular security controls, support both validation and iterative refinement of the models and SRs involved.

B. Traceability in Concurrent Software & Requirement Engineering

Concurrent Software & Requirements Engineering (CSRE) emphasizes the parallelization of software and requirement development phases to accelerate delivery while managing interdependencies across distributed teams [2]. The dynamic nature of CSRE introduces challenges in maintaining traceability across rapidly evolving artefacts such as requirements and requirement-perspective models [15]–[18]. Traceability, defined as the ability to track relationships between development artefacts throughout the software lifecycle [19], becomes critical in CSRE to ensure that DBE design and security requirements are complete and comprehensive. For instance, [20] highlights that incomplete trace links in agile CSRE workflows often lead to security gaps, as teams prioritize feature velocity over documenting requirement-to-code mappings. To address this problem, many studies are investigating the improvement of integrated tool infrastructure for supporting the traceability analysis in complex systems [21]–[23]. For example, TRACEM [24] uses a metamodel to represent trace information standardly and integrates static and dynamic analysis techniques during the reverse engineering process. Hence, conceptual modeling not only helps in visualizing complex interdependencies [25] but also acts as a supplement or cross-validation method for CSRE when integrated with traceability analysis.

C. Knowledge Graphs in traceability

A knowledge graph (KG) is a structured semantic network that represents entities (nodes) and their relationships (edges) with formal semantics, enabling efficient data integration, reasoning, and querying across heterogeneous sources [26]. Compared to traditional traceability matrices [27], which often suffer from scalability issues and lack semantic reasoning capabilities, knowledge graphs offer a flexible, ontology-like structure that can unify heterogeneous artefacts while preserving their semantics. This is particularly beneficial in concurrent software engineering, where artefacts evolve in parallel and semantic misalignments are common [28].

KGs are increasingly applied to address challenges in intelligent code recommendation or vulnerability detection for mapping dependencies between requirements, design components, and security controls. For instance, [29] conducted a systematic literature review highlighting KG's role in automating code generation, bug localization, and API recommendations by linking code artefacts with contextual metadata. Some recent research [28], [30], [31] investigated conceptual models in industry and applied KG to achieve data fusion, deep correlation analysis, or risk identification among data. These achievements are more focused on problem identification and solving in the late stages of software development.

Existing KG-applied research in the early stages of software engineering is limited and often lacks adaptability to DBEs where value flows or data elements and security constraints are tightly connected. Traditional approaches, such as traceability matrices, struggle with scalability and fail to present complex relationships in a multi-stakeholder, multi-business scenario project [32]. Semantic discrepancies between business-oriented value models and technical security requirements usually hinder effective mapping [33]. However, KG's features, which are usually constructed on element types, make it ideal for illustrating the relationships between conceptual models and security requirement items. Query-supporting and reasoning abilities of KG can also contribute to more efficient traceability analysis of these artefacts.

III. Research Methodology

We are interested in understanding whether concurrent software & requirements engineering is effective, specifically if multiple requirement viewpoints are developed in parallel. We followed a four-phase research approach depicted in Fig. 1: (1) problem investigation, (2) artefact development, (3) Traceability abalysis by KG-based approach, and (4) evaluation of effectiveness. The method emphasizes lightweight synchronization and post-hoc semantic alignment using a knowledge graph (KG) to expose inconsistencies and incompleteness across viewpoints.

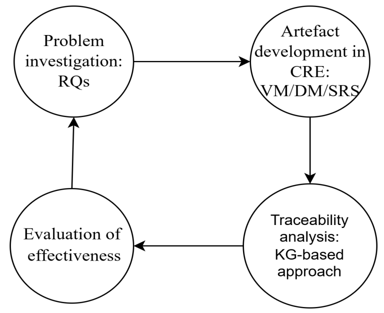
Problem investigation: RQs
Artefact development in CRE: VM/DM/SRS
Traceability analysis: KG-based approach
Evaluation of effectiveness

Fig. 1. Research design cycle

A. Research approach

- **Phase 1: problem investigation.** Concurrent software & requirements engineering develops multiple requirement viewpoints at the same time with the goal to speed-up the development process. While doing so, inconsistencies between requirements may emerge, and also requirements in one viewpoint may be incomplete when considering the other viewpoints. The question is whether concurrent software & requirements engineering is still effective, in relation to the reduced lead time of the project.
- **Phase 2: Artefact development.** We construct several artefacts concurrently. Three partially overlapping subteams independently developed: (1) an e^3*value* model capturing actors and value exchanges; (2) a UML class model specifying data entities, relationships, and sensitive attributes required by the platform; (3) a security requirements document comprising 27 items covering confidentiality, integrity, access control, and transparency. As good as possible, the security requirements reference data assets derived from the data model. Iterative workshops enable progressive refinement and validation of each artefact individually against emergent insights. Consequently, the development of the three requirement viewpoints did not happen completely in isolation, as would be the case for any industry-strength project.
- **Phase 3: Traceability analysis by KG-based approach.** Afterwards, we formalize artefact interrelationships within a knowledge graph (KG). We instantiate KG node types (e.g., value transactions, data assets, and security requirements items), and define semantic/logic edge classes (e.g., tracedFrom_VT, tracedFrom_DA) and corresponding class types (e.g., protects, supports). This graph-based representation facilitates declarative queries (e.g., "What data elements or value elements are protected by the specified security requirements?") and possible automated reasoning to reveal latent dependencies. By encoding artefacts' elements in a unified KG, we transform hard-to-identify interrelationships into rigorous trace links and even construct a semantic network for easy consistency and completeness analysis of the viewpoints involved
- **Phase 4: Evaluation of effectiveness.** The concurrent software & requirements engineering approach is evaluated for effectiveness by considering traceability problems (e.g., incompleteness, inconsistency). Queries and manual inspection to verify that each critical model element was linked to the appropriate security requirements. This analysis identifies missing or ambiguous connections (e.g., data entities with no associated security requirement or value flows lacking clear enforcement).

B. Project management

Development of the first-cycle artefacts began in early 2023 under a staged delivery plan, with iterative workshops during 2023–2024 to refine the e3value model, UML class model, and the 27 SR items. These artefacts evolved in parallel with a small number of synchronization checkpoints, consistent with the project's time-to-market constraints. Key modeling decisions and interim snapshots were recorded to support subsequent KG mapping and adjudication of trace conflicts.

IV. Case Study: MUSIC360 Digital Ecosystem

A. Case context & problem investigation

The MUSIC360 ecosystem is an example of the digital business ecosystem, which aims at providing insights into the value of music to creatives (music performers and authors), venues (restaurants, retail shops, offices), and policymakers (EU officials, national authorities and lobbyists). The Music360 ecosystem provides data about the value of music, where 'value' can have an economic, societal, or cultural connotation. To do so, MUSIC360 collects data about the music used in venues (e.g., by installing music finger-printing devices in venues to discover what is actually played), effects of music played (for example increased revenue), and metadata of music such as rightsholders' (performers and authors) information, music work (including recordings), and earnings

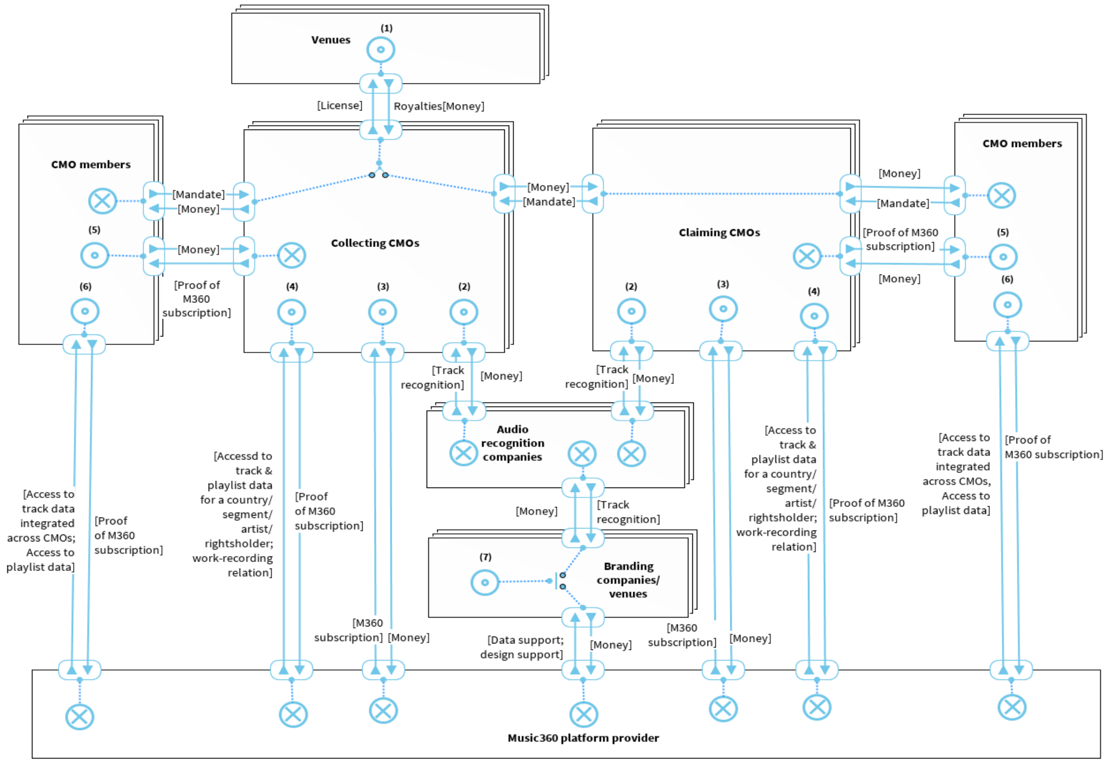
Venues
(1)
[License]
Royalties[Money]
CMO members
[Mandate]
[Money]
(5)
[Money]
[Proof of M360 subscription]
(6)
Collecting CMOs
(4)
(3)
(2)
[Money]
[Mandate]
Claiming CMOs
[Money]
[Mandate]
[Proof of M360 subscription]
[Money]
CMO members
(5)
(6)
(2)
(3)
(4)
[Track recognition]
[Money]
[Track recognition]
[Money]
Audio recognition companies
[Money]
[Track recognition]
(7)
Branding companies/ venues
[Access to track data integrated across CMOs; Access to playlist data]
[Proof of M360 subscription]
[Accessd to track & playlist data for a country/ segment/ artist/ rightsholder; work-recording relation]
[Proof of M360 subscription]
[M360 subscription]
[Money]
[Data support; design support]
[Money]
[M360 subscription]
[Money]
[Access to track & playlist data for a country/ segment/ artist/ rightsholder; work-recording relation]
[Proof of M360 subscription]
[Access to track data integrated across CMOs, Access to playlist data]
[Proof of M360 subscription]
Music360 platform provider

Fig. 2. E^{3}value model for the MUSIC360 platform

by rightsholders. Important stakeholders in this DBE are the collective Management Organizations (CMOs). They collect money from venues in return for a license to play music and pay the money subsequently (minus an administrative fee) to the rightholders.

Due to time constraints, we adopted a CSRE approach to expedite this process. In parallel, we developed the business value model (VM), data model (DM), and security requirement items (SRs) to some extent for capturing business processes, data structures, and security controls. However, this method may introduce potential deficiencies, such as delays in requirement updates and inconsistencies between models. Consequently, a thorough subsequent traceability analysis is necessary to continuously refine and optimize the mappings and relationships between VM, DM, and SRs.

We illustrate **phase 2 (artefacts development)** from our research design cycle in this section.

B. MUSIC360 e^{3}value *model*

An *e^{3}value* model is meant to understand the actors involved in an ecosystem, and what value they exchange with each other. We elicit and design a value model based on the *e^{3}value* modeling language [34]. For this case, we follow a normal elicitation process for *e^{3}value* models, as explained in [8]. For reasons of brevity, we immediately present it in Fig. 2. In the current MUSIC360 revenue model, the economic actor/market segment is presented in 6 types: **EA1 - Venues**, **EA2 - CMO members**, **EA3 - Collecting CMOs**, **EA4 - Music platform provider**, **EA5 - Audio recognition companies** and **EA6 - Branding companies**. We also numbered the transaction paths in the revenue model and explained the revenue scenarios for these paths below.

1) **VT1:** Venues pay licence fees to play background music.
2) **VT2:** Some CMOs use fingerprinting devices provided by audio recognition companies.
   Revenue model: The audio recognition companies (ARC) use devices and other technologies to collect audio data and create a list of tracks played. It provides read access to this data to some stakeholders (venue or CMO) according to the contract.
3) **VT3:** CMOs pay a fee to the M360 platform to get access to track and playlist data from different CMOs, integrated by the platform.
   Revenue model 1: CMOs pay a flat yearly subscription fee to get unlimited access. The fee could be proportional to the number of artists registered with the CMO.
   Revenue model 2: Each CMO pays a base fee to get access and a small amount for each usage of the data, i.e., for every query.
   Whatever the revenue model, subscription just provides access to the data of other CMOs but does not change ownership. Access permissions, on the other hand, may depend on the type of subscription.
4) **VT4:** A CMO with a subscription can access track data, playlist data, and the work-recording relation established by M360 (after CMO members accessed the platform for

the first time and connected their IPI and IPN).

5) **VT5:** If a CMO subscribes, all of its members are subscribed. They pay for this in an implicit transaction: the subscription fee of the CMO is paid by members by deducting it from the earnings they receive.
6) **VT6:** A CMO member that has an M360 subscription can access data about their own tracks and playlists in which they occur, integrated across CMOs.
7) **VT7:** Venues or branding companies who want experiments to measure the background music impact, can pay an audio subscription company to track venue play data, use the M360 platform to store it, and get support for value analysis. During the experiment, measured track data is owned by the venue or branding company. After the experiment is finished, the audio recording device can be reused by another venue or a CMO, if they acquire the service of track and playlist data from the audio recognition company.
8) **VT8:** Other stakeholders can get aggregate information about background music.
   Revenue model: This can be a freemium model. M360 publishes an abbreviated free report yearly. If a stakeholder wants more, it has to pay. CMOs who have a subscription get this data without additional cost. Academic researchers may get it at a lower price.

C. MUSIC360 data model (UML class model)

Understanding the structure of the data itself (for MUSIC360, we do not use unstructured data) usually plays a crucial role in the software engineering process. We executed a data modeling task using standard UML class modeling. This effort takes several workshops for identify key data entities and their relationships, and results in a data model in Fig. 3.

For reasons of brevity, we explain the most important data elements. **Rightholders** (e.g., **neighbouringRightholder** or **authorRightholder**) have **claims** on a **creation** (either a **recording** or a **work**). Rightholders can transfer their claims to a **beneficiary** and can use an **agent** to represent them. Collective management organizations (**CMOs**) collect compensation for usage of claimed intellectual property rights by **venues**. CMOs have a **mandate** for recordings that are played, usually as part of a **performedplaylist** by venues (**chains**, restaurants, etc.). These venues have a **license** to do this, which they obtain from a CMO. **Plays** are monitored by a company called the **monitor**, doing identification of the played recording of work. One or more plays of a recording or work result in **earnings**, via a claim, for the rightsholder. In the following traceability analysis, for simplification, we will focus on valuable data elements and classify them.

D. MUSIC360 security requirement items

We elicit the (data) security requirement items (SRs) in several workshops using the following steps:

- **Stakeholder identification.** Stakeholders can be identified based on the project statement, which can be listed as CMO members (rightsholders, beneficiaries, and agents), CMOs, policymakers, third parties, and venues.
- **Security goal refinement & Asset identification.** We conduct a series of interviews and semi-structured workshops with the identified stakeholders to derive their security concerns based on the general goals and business needs of our project. We explore specific scenarios where security threats might arise, such as data breaches, unauthorized access, and denial-of-service attacks. The security goal and critical assets will be refined and identified. During this process, the data model acts as a reference of assets.
- **Security requirement items.** We identify SRs based on the security goal refinement, identified assets, and possible threats. They can be specified into access control requirements, encryption requirements, and requirements for secure data transmission. Also, requirements for compliance with GDPR and other relevant privacy laws need to be included. In practice, we listed 27 security requirement items covering confidentiality, integrity, and authentication, part of the SRs is shown in Table I.

V. Knowledge Graph Construction

Since CSRE was utilized in the early stages of this project, different artefacts, such as VM, DM, and SRs, were developed almost independently and simultaneously by different practitioners, which easily leads to consistency and completeness issues between artefacts. In order to systematically evaluate the consistency and completeness of these artefacts, we construct a knowledge graph (KG) to perform traceability analysis. This approach has the following advantages:

- **Strong relationship expression:** A KG can express relationships between elements in multiple artefacts as nodes and edges, which, on a meta-level, are the constructs used in the VM, DM and SRs.
- **Convenient for reasoning and analysis:** Through semantic relationship graph modeling, rule definition, and graph analysis, using software tooling such as Neo4j, queries on the KG can be defined that identify missing traces, potential conflicting relationships, etc [1].
- **Supports formal verification:** The KG can be used to verify and explain the completeness and consistency of the viewpoints.

The whole KG-based traceability analysis approach can be divided into 4 steps: **(1) Node & edge class design of KG**, **(2) KG node instantiation**, **(3) traceability mapping (semantic KG edge)**, and **(4) traceability problem identification**. The different node classes that are subject to traceability analysis are concisely presented in Table II, whereas the edge classes can be found in Table III. These tables reflect the most important elements that are elicited in each requirement viewpoint, as well as the possible kinds of relationships between them.

Based on designed classes, we identified and labeled the important information (elements) in different MUSIC360 arte-

[1]https://neo4j.com/

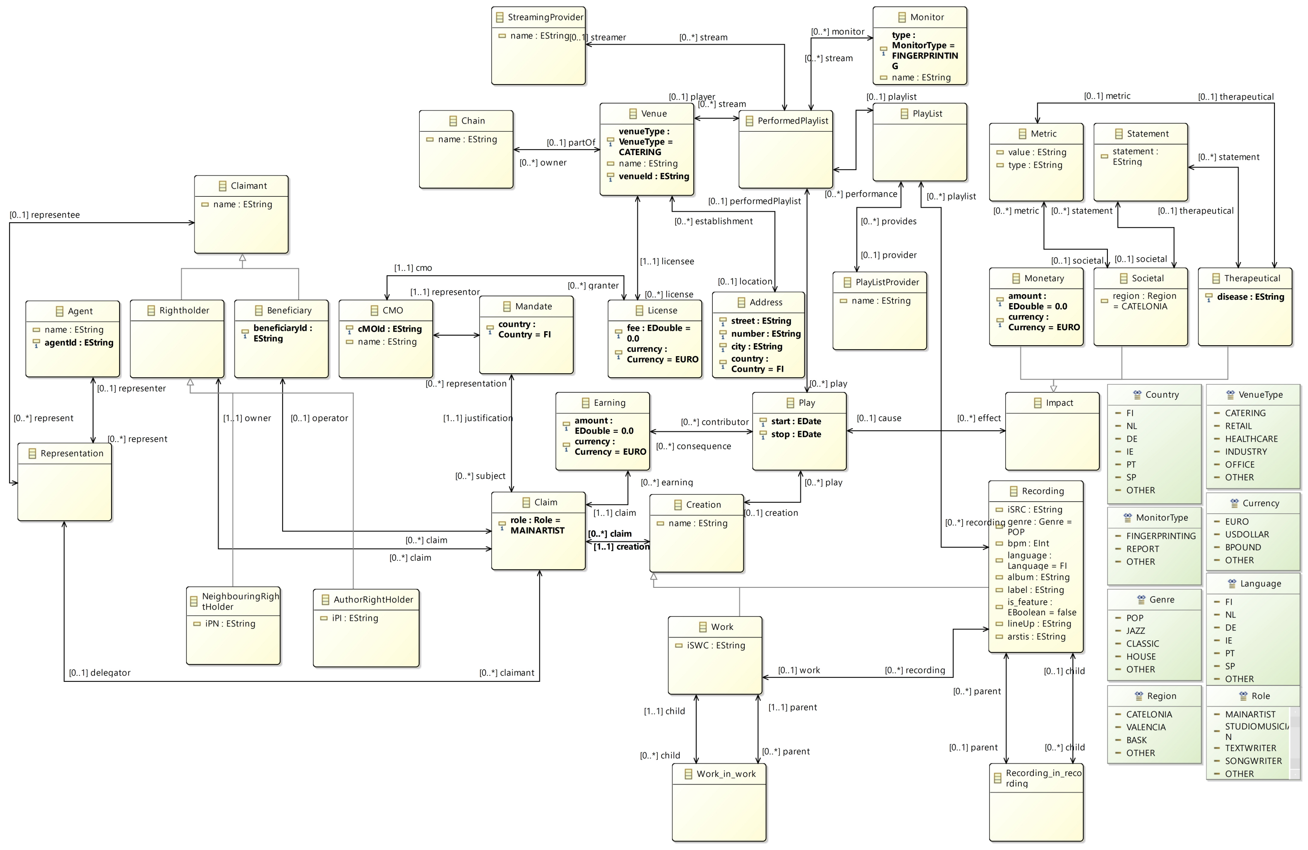

Fig. 3. The MUSIC360 UML data model: classes Venue, CMO, Claim, Earning, etc., with attributes (e.g. Venue.name, Earning.amount) and associations.

Index	Description
SR_2	Data providers **OUGHT** to make data in their custody, but owned by other parties, available to the ecosystem, within the reasonable scope of the project's data accessibility requirements.
SR_5	CMOs **MUST** provide means for the following user groups to manage their data: creatives (e.g., artists), venues and policymakers.
SR_7	Data owner(s) **MUST** be empowered to grant data access to requesting third parties within the ecosystem.
SR_11	Access or revocation of data to and from the various parties of the ecosystem **MUST** be done in a timely and consistent manner.
SR_17	Ecosystem database(s), notably, but not limited to, described in D2.1 **MUST** be encrypted using cryptographically secure symmetric or asymmetric algorithms with sufficient collision entropy for the lifespan of the ecosystem; e.g., in the case that at year "n" RSA2048 is not predicted to provide sufficient collision resistance, RSA3072 must be phased in at year "n - (some reasonable time period)".

TABLE I
(DATA) SECURITY REQUIREMENTS ITEMS (PARTLY AS AN EXAMPLE)

facts as KG nodes systematically. Table IV shows some examples of instantiated nodes we have for KG-based traceability analysis.

VI. TRACEABILITY ANALYSIS

The remaining two steps in the KG-based traceability analysis approach are: **(3) traceability mapping (semantic KG edge)**, and **(4) traceability problem identification**. We illustrate how we construct the traceability KG of between the VM and SRs perspectives and the DM and SRs viewpoints. We evaluate whether each element node in the VM and DM can be traced and linked to the SRs nodes in the graph.

A. Traceability mapping between VM and SRs

We now build a knowledge graph (KG) for traceability analysis between the VM and SRs. First, each VM element (economic actor, value transaction) (following the process we present in Sec.V) becomes a node. Traceability between VM and SRs nodes are expressed by semantic/logic relationships. One example of such a relationship between the VM and SRS on SRS_5 (see Table I) node is shown in Fig. 4, the problem detected here is 'Policymakers in SR_5 has not been traced in VM, it should be a missing economic actor'.

Second, depending on the semantic relationships between the VM and the SRs, traceability problems are found. To simplify, we show our final findings as a summary in Table

Node class	Origin	Description
Value Transaction	VM	A value exchange behavior.
Economic Actor	VM	Stakeholders involved in value transactions, e.g., Venue, CMO.
Data Asset	DM	Data info that needs to be protected.
Data Record	DM	Logical data entities, e.g., tables, records.
Data Owner/User	DM	Data owner or data user, e.g, rightholders, CMO, Venue.
Security Req.	SRs	Clearly described security requirements.

TABLE II
NODE CLASS DESIGN IN KG

V.

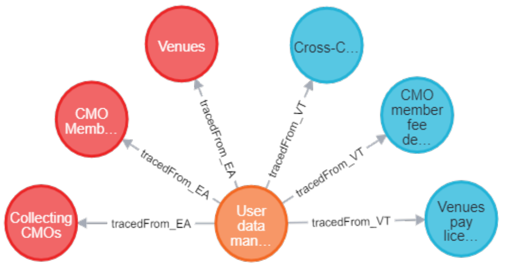

Fig. 4. Examples: node SR_5 traceability mapping in VM-SRs KG

B. Traceability mapping between DM and SRs

We do the same exercise, but now for the DM and SRs viewpoints. An example of a traceability relationship between the DM and SRs on SR_5 node is shown in Fig. 5. The problem detected here is: (1) Creatives (e.g., artists) are expressed as rightsholder in DM, resulting in an alignment problem in tracing. (2) The word 'Data' in SRs is not specified as 'claim', resulting in incompleteness in tracing. Using the semantic relationships between the DM and the SRs, we found all the traceability problems. To simplify, we just summarize our findings in Table V.

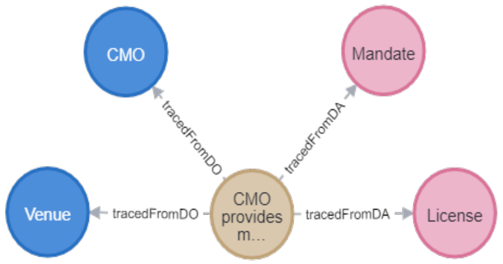

Fig. 5. Examples: node SR_5 traceability mapping in DM-SRs KG

C. Traceability problem statistics result of MUSIC360

We define four traceability problem types and defined the corresponding severity score criteria of these problems for further analysis. The average severity was scored on a three-level scale (1 = low, 2 = moderate, 3 = high) based on the expected impact of each issue on the correctness, security, or completeness of the MUSIC360 platform:

- **AD1: Trace consistency.** Are the associations between different artefacts logically consistent (any contradictions or conflicts)?
  Severity score scale: Inconsistencies are rated by severity: minor semantic divergence (score 1), partial contradiction or unclear logic (score 2), and direct logical conflict between artefacts (score 3).
- **AD2: Trace completeness.** Are all SR items related to all involved VM or DM elements?
  Severity score scale: Missing links are assessed by their impact: non-critical omissions (score 1), gaps involving moderately important elements (score 2), or missing traceability to critical data or value elements (score 3).
- **AD3: Trace redundancy.** Do multiple SR items relate to the same VM/DM elements while having duplications of semantics/logic?
  Severity score scale: Redundancy is scored based on redundancy level: mild phrasing duplication (score 1), overlapping but unnecessary requirements (score 2), or semantically equivalent SRS items leading to maintenance risks (score 3).
- **AD4: SRs missing.** Are there any DM or VM elements not traced by any SRs (any overlooked risk points)?
  Severity score scale: Severity reflects the potential risk of the untraced element: low-impact components (score 1), functionally relevant but non-sensitive elements (score 2), and critical assets or value flows lacking any protection (score 3).

According to the KG graph-based automatic detection and manually verified traceability mappings, we summarize the traceability problems statistics results across VM–SRs and DM–SRs in Table V for simplification. The percentage shown in the table is calculated as the ratio of problem trace relationships to the total expected links (existing correct links + problem links).

Notably, VM–SRs exhibits a higher level of incompleteness (12.7%) and missing SRs links (11.5%) compared to DM–SRS (6.7% and 2.7%, respectively). This indicates that business-level value exchanges were more frequently left unprotected or unlinked to security controls, likely due to the abstraction level of the VM and a lack of explicit mapping guidelines during parallel development, while the development of SRs actually takes the initial UML data model as one of the references. In contrast, the DM–SRs view shows a higher inconsistency rate (6.2% vs. 2.5%), suggesting that more semantic mismatches occurred between structured data elements and the corresponding security requirements. Redundancy remained low in both views (<3%), indicating that duplicated security concerns were relatively rare.

In terms of severity, the average issue severity for VM–SRs reached 2.5 (Moderate–High), compared to 1.9 (Low–Moderate) for DM–SRs. This further reinforces that traceability issues related to value modeling had a greater po-

Relation (edge) class	Domain → Range	Description	Property type
tracedFrom_VT	Security Req. → Value Transaction	SRs protects certain value transactions.	supports, authorizes, constrains
tracedFrom_EA	Security Req. → Economic Actor	SRs protect the interests of the actor.	protects, constrains, enforces_compliance
tracedFrom_DA	Security Req. → Data Asset	Valuable data info needs to be protected.	protects, encrypts, authorizes_access
tracedFrom_DR	Security Req. → Data Record	System record/experiment data to be protected.	audits, validates, restricts_scope
tracedFrom_DO	Security Req. → Data Owner/User	Protect the interests of the actor.	authenticates, enforces, grants_consent

TABLE III
RELATION (EDGE) CLASS DESIGN

NodeID	Class type	Label
VT_1	Value Transaction	PayLicenseFee
EA_1	Economic Actor	Venues
DA_1	Data Asset	Claim
DR_1	Data Record	Monitor
DO_1	Data Owner/User	CMO
SR_2	Security Req.	Data custody & Data availability scope

TABLE IV
KG NODE INSTANTIATION (EXAMPLES)

tential impact, particularly due to overlooked actors (e.g., policymakers, service providers) and untraced value transactions involving sensitive or contractual exchanges. These results suggest that the concurrent development of value models and operational security requirements requires additional support mechanisms, such as traceability templates, shared semantic terms, or structured review checkpoints to ensure alignment.

D. Discussion of traceability problems

We discuss a selection of issues found while analyzing traceability between the VM and SRs, and DM and SRs respectively.

1) **Lack of cross-artefact semantic alignment:** Key concepts or stakeholder labels were inconsistently defined or interpreted across artefacts, leading to omissions or mismatches. For example, the term ‘access to X data’ in VM can be related to ‘data management’, ‘authorization’, ‘data made available to’ or ‘data access’ in SRS. Also, ‘Creatives’ in SRs is expressed by ‘CMO members’ in VM and ”Rightholders” in DM. It will lead to ambiguous and possibly redundant results.
2) **Absence of value consideration:** Due to the CSRE process, security requirements were not systematically derived from the value transaction processes, leaving critical controls unaddressed, e.g., payment process and the track recognition process.
3) **Insufficient stakeholder involvement:** Some key stakeholders (e.g., policymakers, third-party vendors) were almost excluded from model design and requirement elicitation and validation workshops. It causes gaps in actor coverage and value transaction path design in the value model. Also, the corresponding security requirements and practical constraints cannot be reflected clearly due to the lack of domain expertise.
4) **Inherent characteristics of VM, DM and SRs:** The value model emphasizes monetary exchanges and thus overlooks ecosystem activities important to policymakers. The security requirements were derived from high-level goals, general IT architecture, and data-owner needs, rather than specific model elements. Meanwhile, the data model captures broad entity structures but omits many detailed attributes and operations. As a result, some generic SRS items and security measures cannot be precisely or fully traced back to either the value or data models.
5) **Lack of structured SRs:** SRs definers overlooked non-functional data protections (e.g., encryption for DR_3 Statements) due to lack of structured checklists. Some SRs expressions are broad and ambiguous, lacking a clearly defined applied scope/scenario.

VII. DISCUSSION

A. Recommendations for concurrent requirement engineering

1) **Integrated workshops:** Co-design VM/DM/SRs with cross-functional teams (business value, security, database design).
2) **Semantic registries:** Define shared terms (e.g., ‘access to X data’ in VM = ‘authorization (JWT) + the scope of data access’ in SRs).
3) **Clarified security scope** The security requirement items should clearly state the protection scope, e.g., which data/value/activities are related to.
4) **Normalized data model design:** Security requirements usually protect various classes (elements) in the data model. Class attributes or method can help identidy the types of corresponding security requirements to a certain extent and help with traceability analysis. Therefore, the development of data models can be more standardized and enhance integrity.
5) **Extend model design scope:** Extend models to cover edge scenarios in cases (e.g., add third-party service integrations as a new value transaction path in VM).

B. Evaluation of effectiveness

We evaluated the effectiveness of concurrent software & requirements engineering (CSRE) by analyzing traceability

Viewpoints	% Inconsistency	% Incompleteness	% Redundancy	% SRs missing	Avg. Severity
VM–SRs	2.5% (4/165)	12.7% (21/165)	2.5% (4/165)	11.5% (19/165)	2.5 (Moderate–High)
DM–SRs	6.2% (11/179)	6.7% (12/179)	2.2% (4/179)	2.7% (5/179)	1.9 (Low-Moderate)

TABLE V
TRACEABILITY PROBLEM STATISTICS RESULT OF MUSIC360 ARTEFACTS

Item	Description & Validation
RulePattern_2	Each AccessControl Req. related **SRs** node should link at least two types of nodes in DM: the **Data User** empowered/affected and the **data asset/record**.
Validation	*MATCH (sr:SecurityRequirement type: 'AccessControl')* *WHERE COUNT(sr)-[:tracedFrom_DO]− >() = 0 OR COUNT(sr)-[:tracedFrom_DA|tracedFrom_DR]− >() = 0* *RETURN sr.id AS IncompleteAccessControlReq;*
RulePattern_3	If the **data asset/data record** nodes or value objects in **value transfer** nodes are sensitive information, they should at least be traced to two types of **SRs** nodes: confidentiality Req. related and AccessControl Req. related.
Validation	*MATCH (da:DataAsset sensitivity: 'high')* *WITH da, COUNT(da)< −[:tracedFrom_DA]-(sr:SecurityRequirement) AS srCount WHERE srCount < 2* *RETURN da.id AS UnderprotectedData;*
RulePattern_5	If the **SRs** node requires "log/record" (Transparency Req. related), there should be a corresponding **data record** node in DM.
Validation	*MATCH (sr:SecurityRequirement)* *WHERE sr.description CONTAINS 'log' OR sr.description CONTAINS 'audit' AND NOT EXISTS(sr)-[:tracedFrom_DR]− >()* *RETURN sr.id AS UnloggedRequirement;*

TABLE VI
RULE PATTERNS FOR TRACEABILITY KNOWLEDGE GRAPH

problems identified across independently developed artifacts. Through knowledge-graph–based analysis approach, we systematically detected traceability gaps between the value model, data model, and security requirement items. Specifically, we identified problems in four dimensions: inconsistencies, incompleteness (model-side), redundancy across requirements, and entirely missing SR items. These issues were not marginal: for example, several critical data assets and value transactions lacked corresponding security requirements, while some SR items were either unlinked or semantically duplicated. Our findings indicate that, although CSRE accelerates development by enabling parallel modeling, it increases the risk of misalignment when traceability is not explicitly managed. The presence of trace gaps highlights a lack of coordination or shared semantic understanding across modeling teams. These problems could be addressed by introducing structured traceability support during CSRE, such as shared modeling guidelines, intermediate checkpoints for cross-artifact reviews, and trace consistency rules. In this context, the traceability analysis not only reveals weaknesses in artifact alignment but also guides targeted refinements and coordination strategies within concurrent development workflows.

C. Limitations

Our study has three main limitations:

- **Manual verification subjectivity**: manual inspection of trace links and severity scoring may introduce researcher bias, affecting reproducibility.
- **Single-case generalizability:** the approach was evaluated on a single DBE (MUSIC360), so its applicability to other domains remains to be validated.
- **Granularity gaps across viewpoints:** Value models can be abstract, and data models may omit operational attributes, which limits precise mapping to the SRS scope. We plan to introduce "bridge" elements (e.g., value-object classifiers, data-asset sensitivity annotations) and enforce them via KG completeness rules.

D. Future work: rule pattern in traceability analysis

To improve traceability quality, we propose defining a set of rule patterns for the knowledge graph to help detect hidden logic issues, such as the Inheritance relationship or actual constraint. In future work, we plan to integrate a reasoning engine that can automatically check these patterns and infer implicit relationships. Each SR item can be tagged with a type (e.g., confidentiality, access control, transparency), which helps reasoning tools relate them to relevant system activities. Examples of such rule patterns are shown in Table VI.

VIII. CONCLUSION

This paper has investigated the effectiveness of concurrent software & requirements engineering (CSRE) through a traceability analysis of three key artefacts (value model, data model, and security requirement items) developed in parallel within the MUSIC360 digital business ecosystem. By applying a knowledge–graph–based traceability approach, we uncovered a range of issues, including inconsistencies between model semantics and requirements, incompleteness in the coverage of data and value elements, redundancy in requirement statements, and missing SR items. These findings suggest that while CSRE allows for accelerated and distributed artefact development, it also introduces a significant risk of traceability misalignment if cross-model coordination is not systematically supported. Our analysis reveals that many traceability problems stem from ununited semantics, insufficient stakeholder involvement, or the inherent characteristics of each artefacts. To address these challenges, we propose practical recommendations such as introducing lightweight traceability rule patterns, using shared semantics terms, and holding integrated workshops within cross-functional teams. Ultimately, our findings indicate that although CSRE can be effective, its success depends heavily on traceability management mechanisms that ensure artefacts remain semantically and logically aligned. This study thus highlights the importance of embedding traceability analysis into the concurrent engineering process, not merely as a ex-post check, but as an integral activity that guides coordination and quality assurance.

ACKNOWLEDGMENT

Funded by the European Union under project no. 101094872. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] R. Wieringa and J. Gordijn, *Digital Business Ecosystems*. Soest, The Netherlands: TVE Press, 2023. [Online]. Available: https://www.thevalueengineers.nl/digital-business-ecosystems-book?page=digital-business-ecosystems-book

[2] J. Blackburn, G. Scudder, and L. N. Van Wassenhove, "Concurrent software development," *Communications of the ACM*, vol. 43, no. 11es, pp. 4–es, 2000.

[3] A. Silva, "Requirements, domain and specifications: a viewpoint-based approach to requirements engineering," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, 2002, pp. 94–104.

[4] P. K. Senyo, K. Liu, and J. Effah, "Digital business ecosystem: Literature review and a framework for future research," *International journal of information management*, vol. 47, pp. 52–64, 2019.

[5] J. Gordijn and R. Wieringa, "The business model of digital ecosystems: Why and how you should do it," in *Advances in Enterprise Engineering XVI*, ser. Lecture Notes in Business Information Processing, C. G. M. J. S. Guerreiro, Ed. Germany: Springer-Verlag, 2023.

[6] S. Suuronen, J. Ukko, R. Eskola, R. S. Semken, and H. Rantanen, "A systematic literature review for digital business ecosystems in the manufacturing industry: Prerequisites, challenges, and benefits," *CIRP Journal of Manufacturing Science and Technology*, vol. 37, pp. 414–426, 2022.

[7] T. Chekfoung, D. Sunil, and G. Binita, "Conceptualising capabilities and value co-creation in a digital business ecosystem (dbe): A systematic literature review," *Journal of Information Systems Engineering and Management*, vol. 5, no. 1, 2020.

[8] J. Gordijn and H. Akkermans, *Value Webs - Understanding e-Business Innovation*. The Value Engineers, 2018. [Online]. Available: https://www.thevalueengineers.nl/product-category/publications/e3value-ref/

[9] Z. N. Khaswala and S. A. Irani, "Value network mapping (vnm): visualization and analysis of multiple flows in value stream maps," in *Proceedings of the Lean Management Solutions Conference*, vol. 614, 2001, pp. 1–18.

[10] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[11] J. P. McDermott and C. Fox, "Using abuse case models for security requirements analysis," *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pp. 55–64, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:206580824

[12] J. Heikka and M. Siponen, "Abuse cases revised: An action research experience," 01 2006, p. 46.

[13] N. Mayer, A. Rifaut, and E. Dubois, "Towards a risk-based security requirements engineering framework," in *Proceedings of REFSQ*, vol. 5, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:8837909

[14] Q. Ramadan, M. Salnitriy, D. Strüber, J. Jürjens, and P. Giorgini, "From secure business process modeling to design-level security verification," in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2017, pp. 123–133.

[15] J. D. Blackburn, G. Hoedemaker, and L. N. Van Wassenhove, "Concurrent software engineering: prospects and pitfalls," *IEEE Transactions on engineering management*, vol. 43, no. 2, pp. 179–188, 1996.

[16] P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, and G. Kappel, "Concurrent modeling in early phases of the software development life cycle," in *International Conference on Collaboration and Technology*. Springer, 2010, pp. 129–144.

[17] L. Bass, I. Weber, and L. Zhu, *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[18] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," *arXiv preprint arXiv:1709.08439*, 2017.

[19] O. C. Gotel and C. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of ieee international conference on requirements engineering*. IEEE, 1994, pp. 94–101.

[20] J. Cleland-Huang, O. C. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Future of software engineering proceedings*, 2014, pp. 55–69.

[21] S. Akman, M. Özmut, B. Aydın, and S. Göktürk, "Experience report: implementing requirement traceability throughout the software development life cycle," *Journal of Software: Evolution and Process*, vol. 28, no. 11, pp. 950–954, 2016.

[22] K. Hardie, J. Church, J. Dodge, B. Fordham, S. Gajadhar, N. Han, J. Miles, F. Santoro, and G. Trancho, "Beyond spreadsheets: enhancing collaboration and traceability by streamlining systems engineering processes with atlassian jira," in *Modeling, Systems Engineering, and Project Management for Astronomy XI*, vol. 13099. SPIE, 2024, pp. 92–114.

[23] G. Jadoon, M. Shafi, and S. Jan, "A model-oriented requirements traceability framework for small and medium software industries," in *2019 International Arab Conference on Information Technology (ACIT)*. IEEE, 2019, pp. 91–96.

[24] C. T. Pereira, L. I. Martínez, and L. M. Favre, "Tracem-towards a standard metamodel for execution traces in model-driven reverse engineering," in *XXVIII Congreso Argentino de Ciencias de la Computación (CACIC)(La Rioja, 3 al 6 de octubre de 2022)*, 2023.

[25] K. Großer, V. Riediger, and J. Jürjens, "Requirements document relations: A reuse perspective on traceability through standards," *Software and Systems Modeling*, vol. 21, no. 6, pp. 1–37, 2022.

[26] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, A. Wahler, D. Fensel *et al.*, "Introduction: what is a knowledge graph?" *Knowledge graphs: Methodology, tools and selected use cases*, pp. 1–10, 2020.

[27] U. Simsek, E. Kärle, K. Angele, E. Huaman, J. Opdenplatz, D. Sommer, J. Umbrich, and D. Fensel, "A knowledge graph perspective on knowledge engineering," *SN Computer Science*, vol. 4, no. 1, p. 16, 2023.

[28] Z. Chen, Y. Wan, Y. Liu, and A. Valera-Medina, "A knowledge graph-supported information fusion approach for multi-faceted conceptual modelling," *Information Fusion*, vol. 101, p. 101985, 2024.

[29] L. Wang, C. Sun, C. Zhang, W. Nie, and K. Huang, "Application of knowledge graph in software engineering field: A systematic literature review," *Information and Software Technology*, vol. 164, p. 107327, 2023.

[30] H. Zhong, D. Yang, S. Shi, L. Wei, and Y. Wang, "From data to insights: the application and challenges of knowledge graphs in intelligent audit," *Journal of Cloud Computing*, vol. 13, no. 1, p. 114, 2024.

[31] A.-M. Ghiran and R. A. Buchmann, "The model-driven enterprise data fabric: A proposal based on conceptual modelling and knowledge graphs," in *International conference on knowledge science, engineering and management*. Springer, 2019, pp. 572–583.

[32] U. Simsek, E. Kärle, K. Angele, E. Huaman, J. Opdenplatz, D. Sommer, J. Umbrich, and D. Fensel, "A knowledge graph perspective on knowledge engineering," *SN Computer Science*, vol. 4, no. 1, p. 16, 2022.

[33] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel, "Model-driven business process security requirement specification," *Journal of Systems Architecture*, vol. 55, no. 4, pp. 211–223, 2009.

[34] J. Gordijn and R. Wieringa, *E3value User Guide - Designing Your Ecosystem in a Digital World*, 1st ed. The Value Engineers, 2021.